Version Control

Pertemuan 1

MK Algoritma Pemrograman II

M. N. Fakhruzzaman, S.Kom., M.Sc.

Program Studi S1 Teknologi Sains Data

Fakultas Teknologi Maju dan Multidisiplin

Universitas Airlangga Indonesia

Background

Name	Date modified	Type
Bismillah1	28/08/2021 9:33	Text Document
Bismillah2	28/08/2021 9:33	Text Document
Bismillah3	28/08/2021 9:33	Text Document
Bismillah4	28/08/2021 9:33	Text Document
BismillahFix1	28/08/2021 9:33	Text Document
BismillahFix2	28/08/2021 9:33	Text Document
BismillahPalingFix1	28/08/2021 9:33	Text Document
BismillahPalingFix2	28/08/2021 9:33	Text Document
BismillahPalingFixxxxxxxxxxx	28/08/2021 9:33	Text Document

Keeping multiple copies of files with version numbers

Background

- VCS is essential tools of the **software engineering** world
- Without version control, coordinating a team of programmers all editing the same project's code will reach **pull-out-your-hair levels of aggravation**.

VCS you've already used

- Dropbox
- Undo/redo buffer
- Keeping multiple copies of files with version numbers
- .....

Version control terminology

- **Repository**: a local or remote store of the versions in our project
- **Working copy**: a local, editable copy of our project that we can work on
- **File**: a single file in our project
- **Version or revision**: a record of the contents of our project at a point in time
- **Change or diff**: the difference between two versions
- **Head**: the current version

VCS Supports

- **Reverting** to a past version
- **Comparing** two different versions
- **Pushing** full version history to another location
- **Pulling** history back from that location
- **Merging** versions that are offshoots of the same earlier version

VCS Features

- **Reliable**: keep versions around for as long as we need them, allow backups
- **Multiple files**: track versions of a project, not single files
- **Meaningful versions**: what were the changes, why where they made?
- **Revert**: restore old versions, in whole or in part
- **Compare versions**
- **Review history**: for the whole project or individual files
- **Not just for code**: prose, images, ...
- **Allow multiple people to work together**:
  - **Merge:** combine versions that diverged from a common previous version
  - **Track responsibility:** who made that change, who touched that line of code?
  - **Work in parallel:** allow one programmer to work on their own for a while (without giving up version control)
  - **Work-in-progress:** allow multiple programmers to share unfinished work (without disrupting others, without giving up version control)

VCS Features

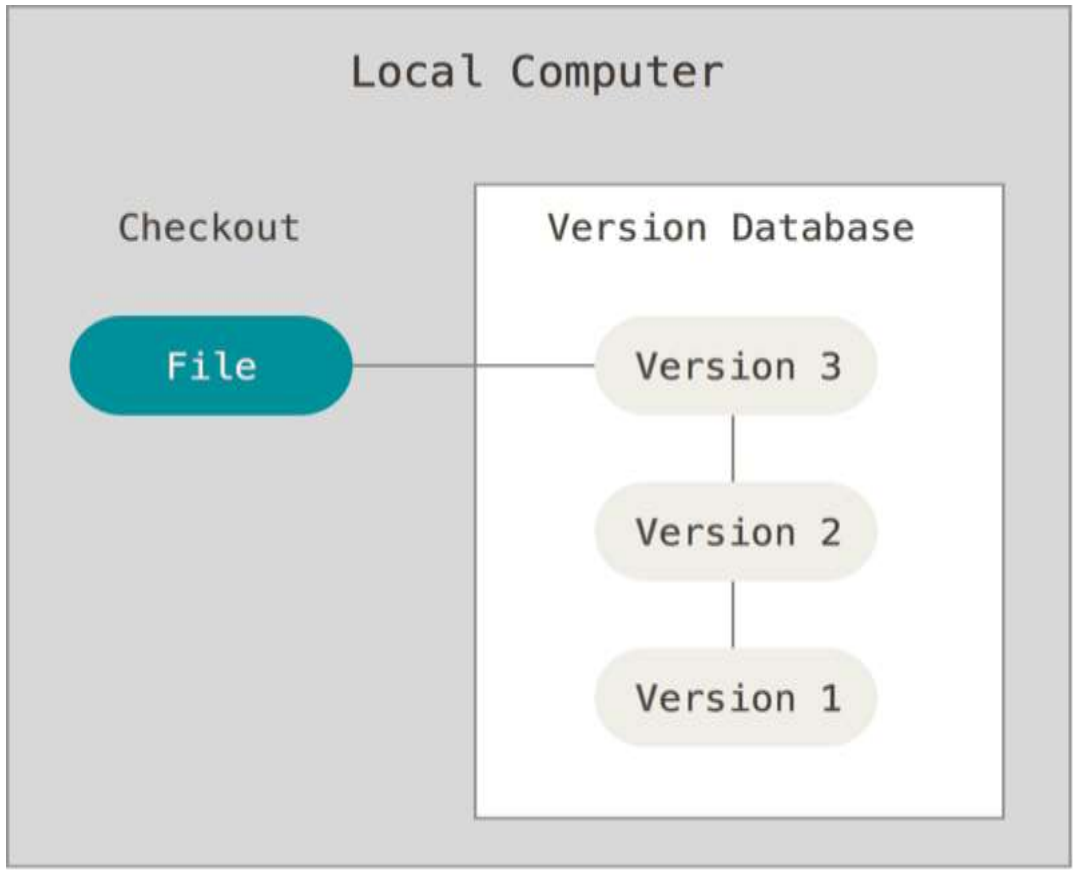
- Modern VCSs also let you easily (and often automatically) answer 3 questions like:
  1. **Who wrote this module?**
  2. **When was this particular line of this particular file edited? By whom? Why was it edited?**
  3. **Over the last 1000 revisions, when/why did a particular unit test stop working?**

VCS for Multiple Developers

VCS type

1. Local version control systems

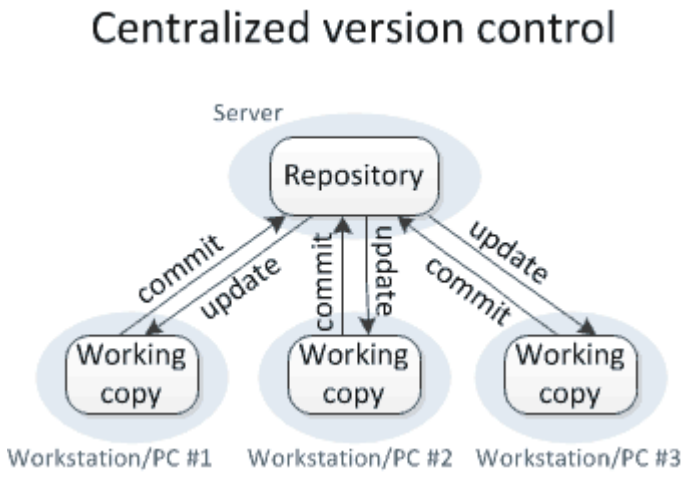
A simple discipline of saving backup files would get the job done.

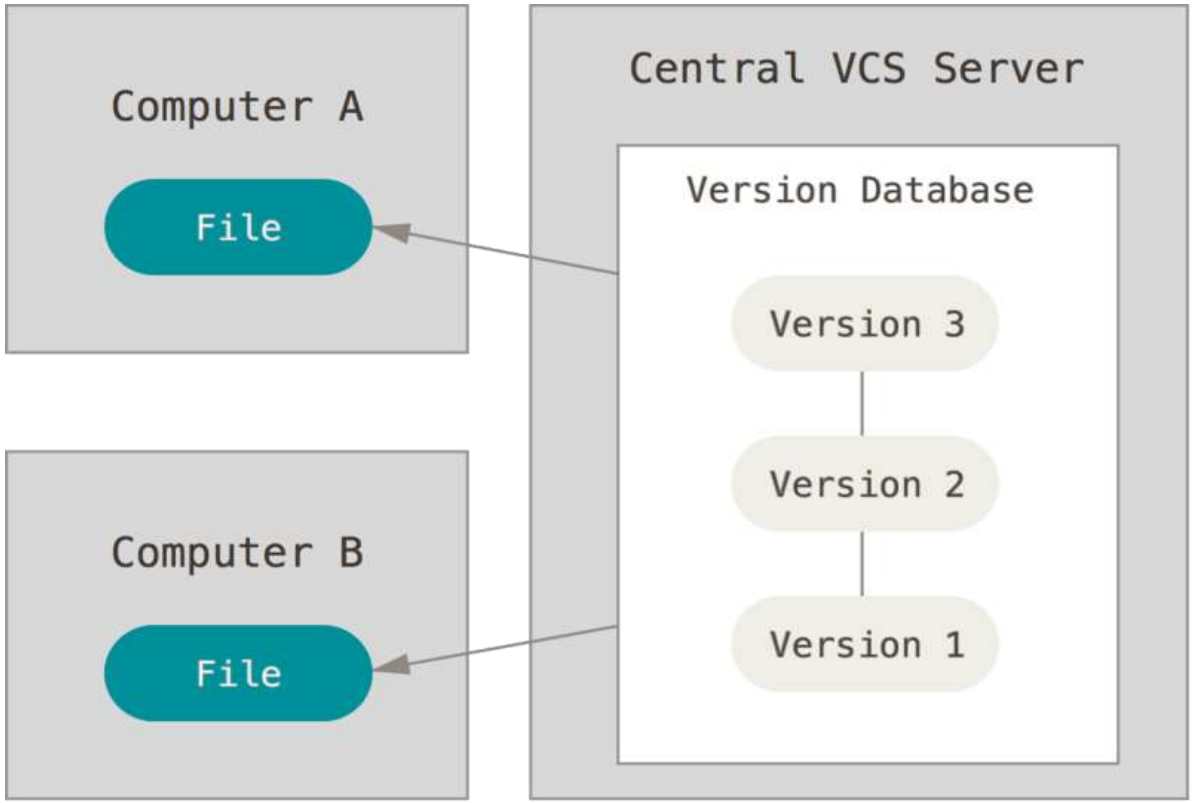


VCS type

2. Centralized version control systems

Everyone must share their work to and from the master repository, and a change is only *in version control* if it's *in the master repository*.

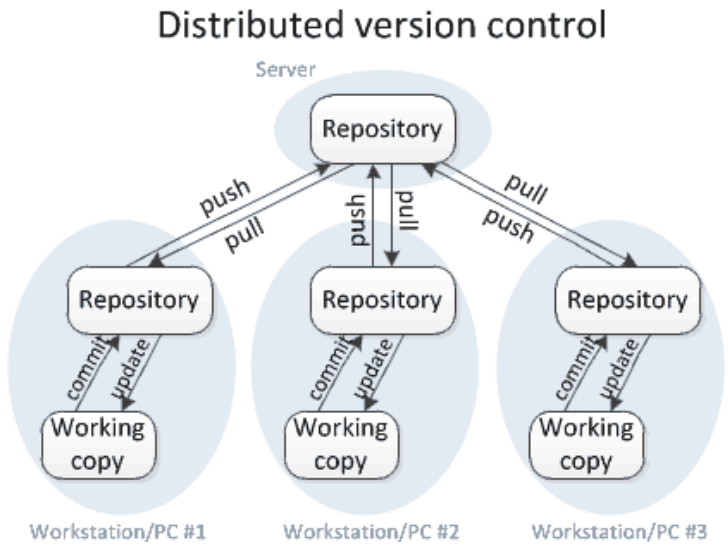


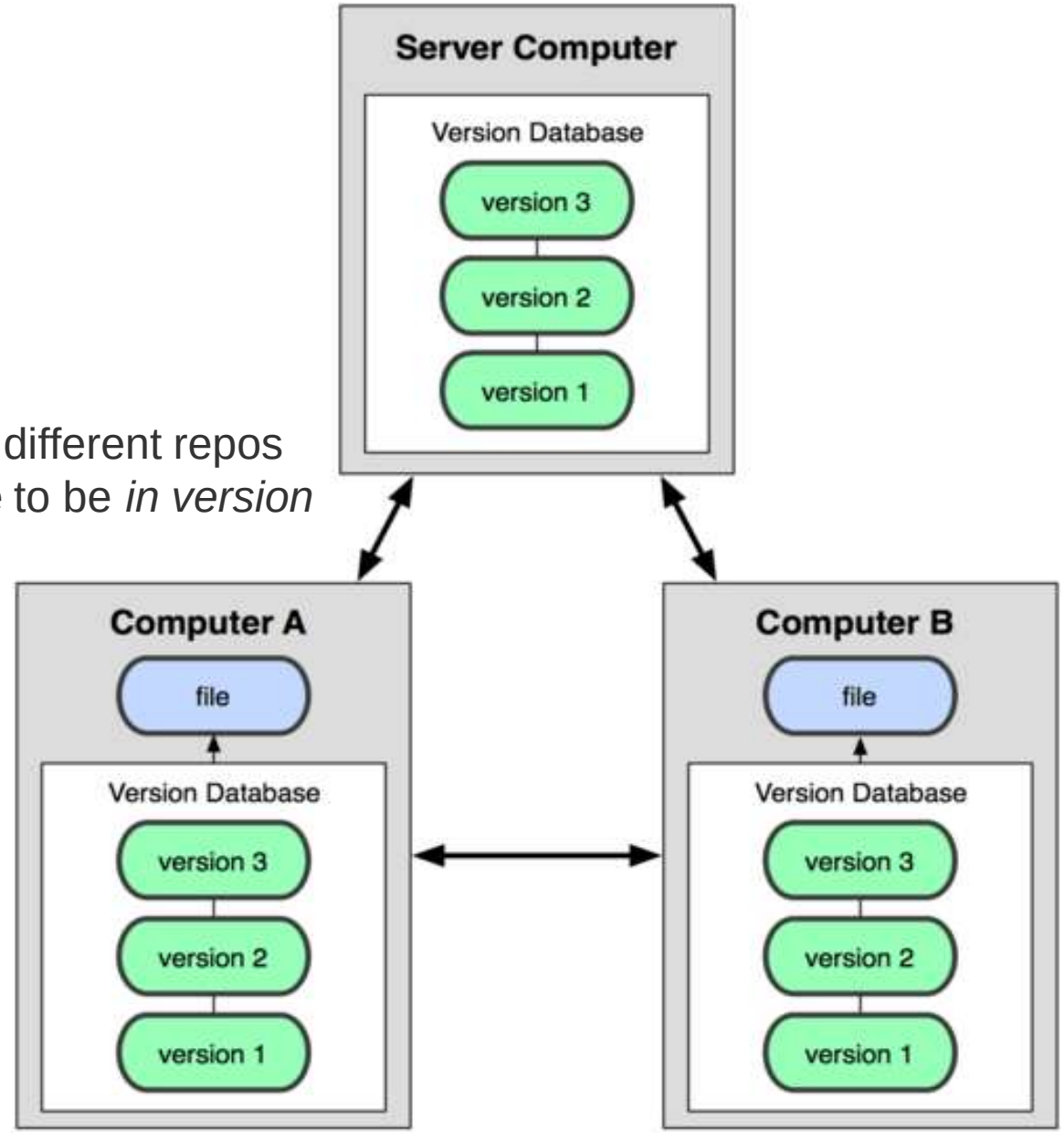


VCS type

3. Distributed version control systems

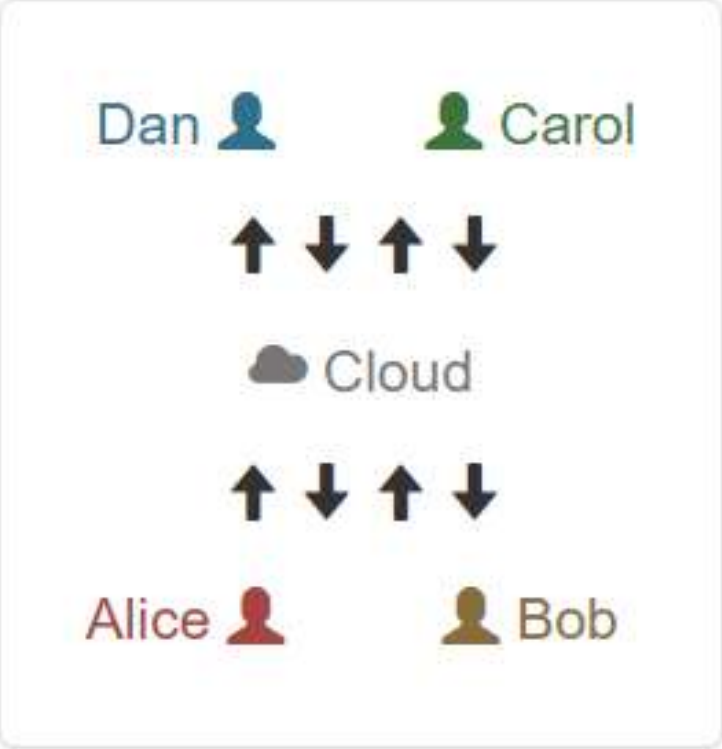
- All repositories are created equal
- It's up to users to assign them different roles
- Different users might share their work to and from different repos
- The team must decide what it means for a change to be *in version control*.



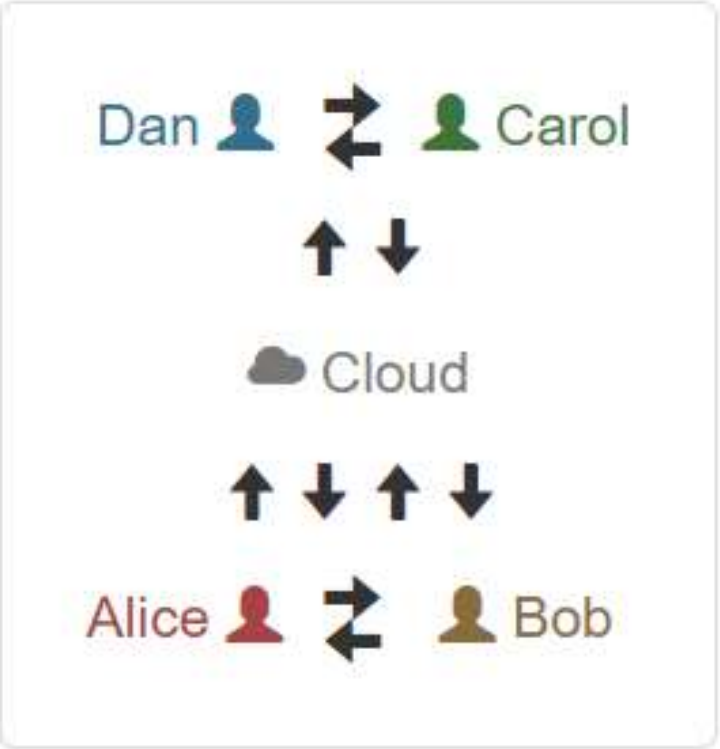


Distributed vs. Centralized

- **Centralized**



- **Distributed**



Version Control System

- Visual Source Safe (Microsoft)
- CVS (opensource) -> Subversion
- **GIT**
- Mercurial
- Monotone
- Bazaar
- TFS
- VSTS
- Perforce Helix Core
- IBM Rational ClearCase

Conclusion

- Version control and the big three

Safe from bugs	find when and where something broke
	look for other, similar mistakes
	gain confidence that code hasn't changed accidentally
Easy to understand	why was a change made?
	what else was changed at the same time?
	who can I ask about this code?
Ready for change	all about managing and organizing changes
	accept and integrate changes from other developers
	isolate speculative work on branches

Tugas Praktikum (Berkelompok anggota 4)

- Set up sebuah Version Control System bertipe Subversion (TortoiseSVN) di salah satu computer teman
- Konfigurasikan repository tersebut sehingga dapat diakses melalui wifi yang sama (satu network)
- Coba gunakan fitur SVN (push, merge, dsb) untuk salah satu file di repository tersebut (file apa saja, bisa dokumen laporan)
- Buat laporan dan jelaskan step by step

Terima Kasih

Referensi:

http://web.mit.edu/6.005/www/fa14/classes/05-version-control/
https://missing.csail.mit.edu/2020/version-control/
https://homes.cs.washington.edu/~mernst/advice/version-control.html

Git & Github

Pertemuan 2
MK Algoritma Pemrograman II
2021/2022

Ika Qutsiati Utami, S.Kom., M.Sc.

M. N. Fakhruzzaman, S.Kom., M.Sc.

Program Studi S1 Teknologi Sains Data

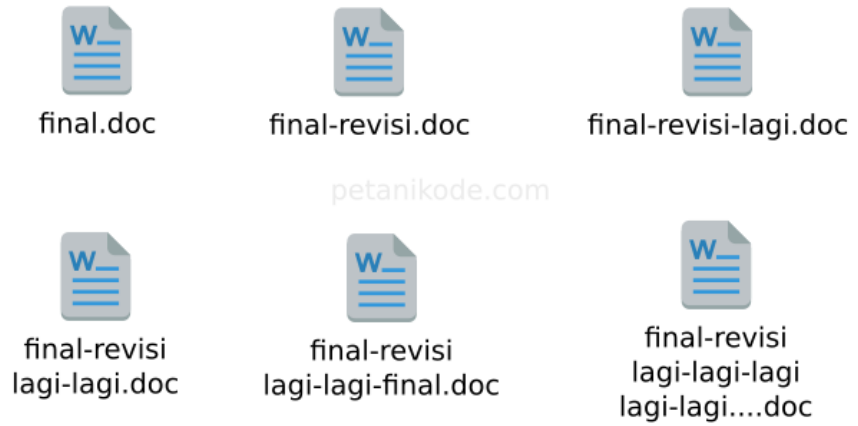
Fakultas Teknologi Maju dan Multidisiplin

Universitas Airlangga Indonesia

Git

What is Git?

- Tools proyek pengembangan software (***version control system & collaboration***)
- Termasuk ***distributed revision control*** (VCS terdistribusi)
- Dikembangkan oleh **Linus Torvald** (2005) untuk Linux



Tradisional

With Git

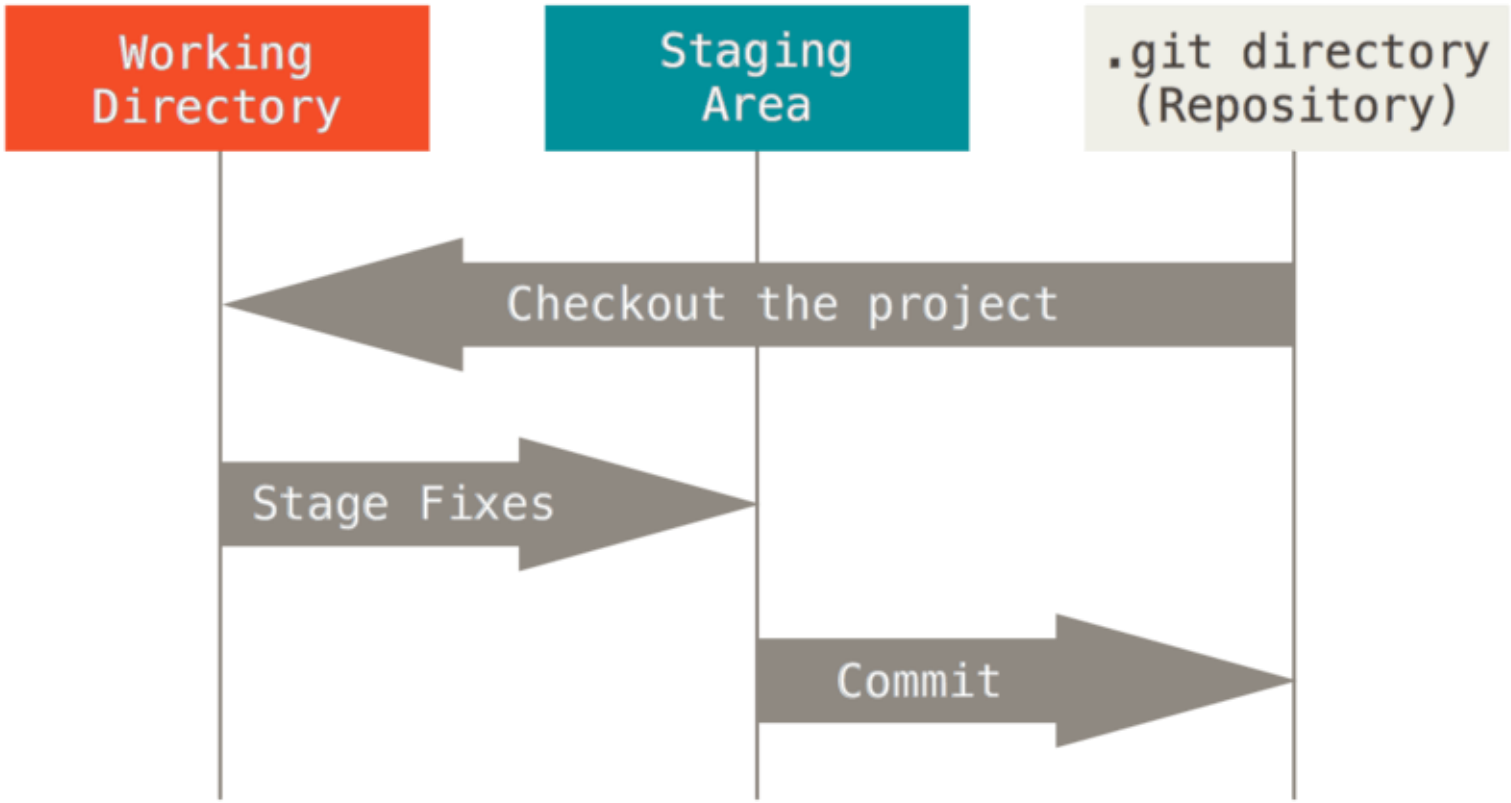
Why using Git?

- Menyimpan seluruh versi source code
- Tools untuk kolaborasi dalam proyek
- Aman untuk kolaborasi, karena dapat di cek apa yang diubah dan siapa yang mengubahnya
- Pintu masuk untuk berkontribusi pada proyek open-source
- Deploy aplikasi modern
- Membuat blog
- Dsb..

The Three States

- Git generally **only adds data**
  - **Modified:** kondisi dimana revisi atau perubahan sudah dilakukan, tetapi belum ditandai dan belum disimpan di version control.
  - **Staged**: kondisi dimana revisi sudah ditandai, tetapi belum disimpan di version control.
  - **Committed**: kondisi dimana revisi sudah disimpan di version control.

Git vs other VCS



Repositori (repository) adalah istilah yang digunakan untuk direktori proyek yang menggunakan Git.

Important Vocabulary

No	Istilah	Keterangan
1	Repo/Repository	Folder suatu project.
2	Commit	Rekaman/snapshot dari repository (Riwayat perubahan repository).
3	Hash	Penanda unik pada sebuah commit (terdiri dari angka dan huruf yang panjang).
4	Checkout	Berpindah ke sebuah perubahan tertentu.
5	Branch	Cabang dari sebuah perubahan.
6	Merge	Menggabungkan dua atau lebih branch.
7	Remote	Resource yang memiliki repository.
8	Clone	Mengambil repository dari *remote*.
9	Push	Mengirim commit ke repository.
10	Pull	Mengambil commit dari repository.

Installing Git

- Linux (https://git-scm.com/download/linux)
- MacOS (https://git-scm.com/download/mac)
- Windows (https://git-scm.com/download/win)

Using Git

Access:

- Command-line
- Graphical user interfaces (GUI)

First setting up (configuration):

- Identity:

```
$ git config --global user.name "John Doe"
$ git config --global user.email johndoe@example.com
```

- Editor:

```
$ git config --global core.editor emacs
```

- Branch name:

```
$ git config --global init.defaultBranch main
```

Getting a Git Repository

1. You can **take a local directory** that is currently not under version control, and turn it into a Git repository

```
$ cd C:/Users/user/my_project
```

and type:

```
$ git init
```

2. You can **clone** an existing Git repository from elsewhere

```
git clone <url>
```

```
$ git clone https://github.com/libgit2/libgit2
```

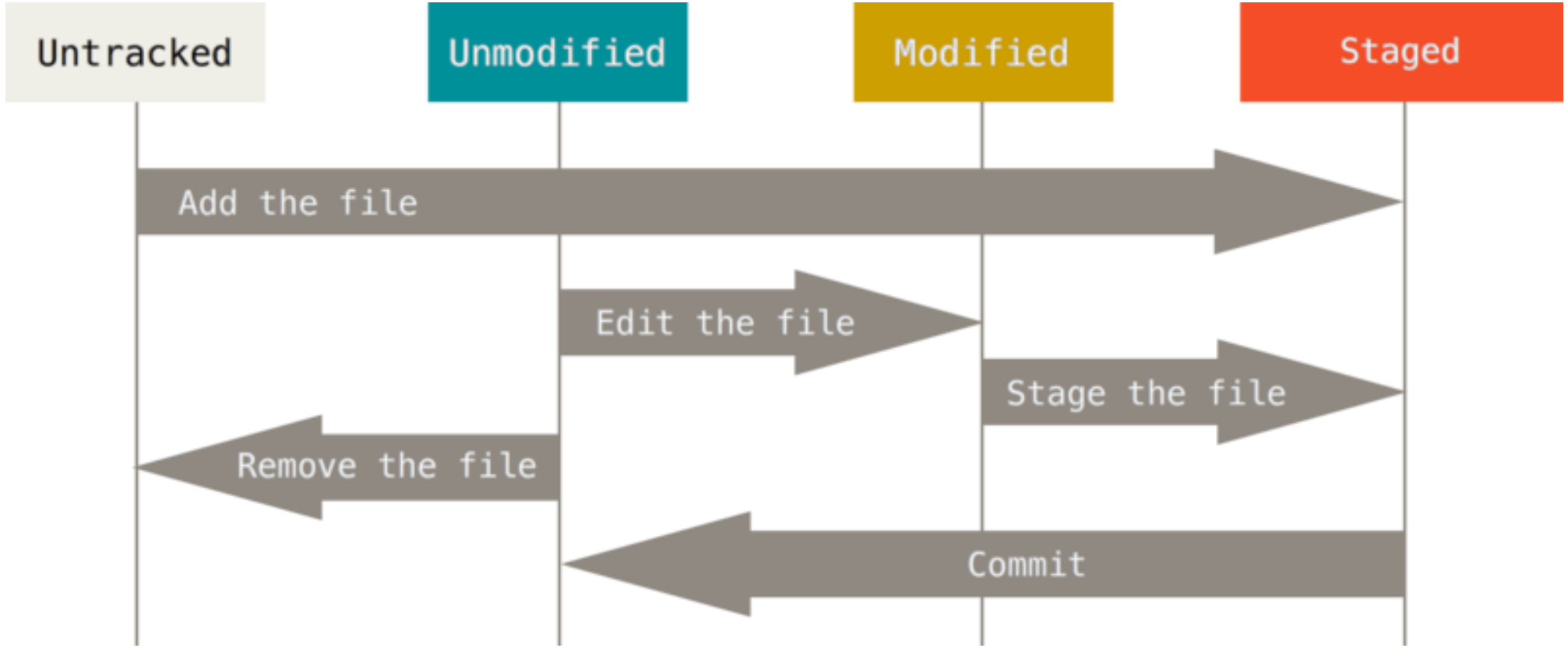
```
$ git add *.c
$ git add LICENSE
$ git commit -m 'Initial project version'
```

Recording Changes to the Repository

Each file in your working directory:

1. **Tracked**
   - Files that were in the last snapshot, as well as any newly staged files
   - They can be unmodified, modified, or staged
   - In short, tracked files are files that Git knows about
   - When you first clone a repository, all of your files will be tracked and unmodified because Git just checked them out and you haven't edited anything
2. **Untracked**
   - Any files in your working directory that were not in your last snapshot and are not in your staging area

Recording Changes to the Repository



The lifecycle of the status of your files

Checking the Status of Your Files

1. **If file exists**

```
$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
nothing to commit, working tree clean
```

2. **If the file didn't exist before**

```
$ echo 'My Project' > README
$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
Untracked files:
  (use "git add <file>..." to include in what will be committed)

    README

nothing added to commit but untracked files present (use "git add" to track)
```

Git Basic Operations

1. git config
2. git init
3. git add
4. git clone
5. git commit
6. git status
7. git push
8. git checkout
9. git remote
10. git branch
11. git pull
12. git merge
13. git diff
14. git tag
15. git log
16. git restore
17. git reset
18. git remove
19. etc..

Github

What is Github?

- GitHub is a **website for hosting Git projects (layanan cloud)**.
- GitHub adds useful **web-based tools to Git** and makes it much easier to collaborate and share your projects with others.
- There is also a downloadable version of GitHub that you can install on Windows and macOS.

- Click **what is github**
- Github **website**
- Github **desktop**

What is GitHub?

- Manajemen project
- Repository online
- Sistem versioning code
- Platform jaringan sosial (untuk para developer seluruh dunia)

Git vs GitHub

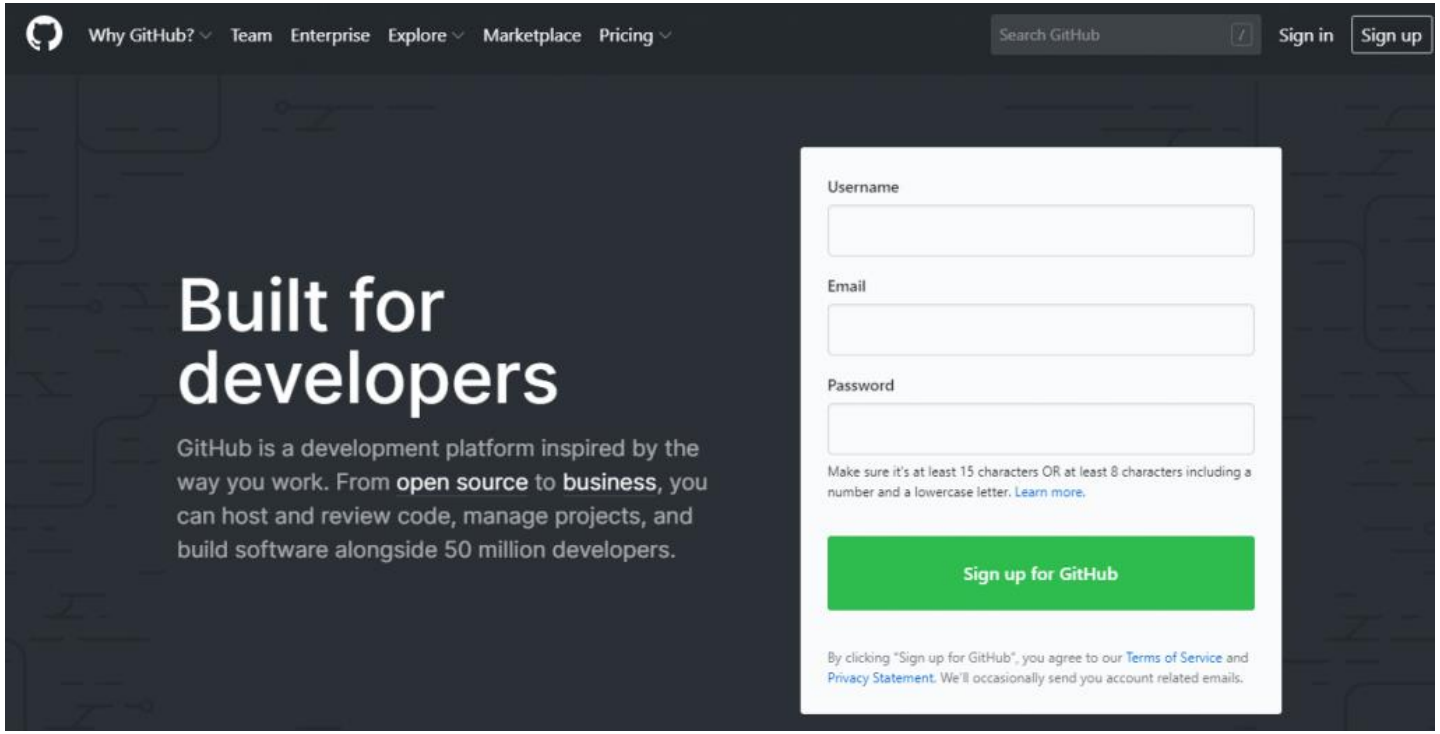
Git	GitHub
1. *Install software* di penyimpanan lokal	1. *Host* melalui layanan *cloud*
2. Dikelola oleh The Linux Foundation	2. Diakuisisi oleh Microsoft pada 2018
3. Fokus pada *version control* dan *code sharing*	3. Fokus pada *source code hosting* terpusat
4. Akses *offline*	4. Akses *online*
5. Tidak menggunakan fitur *user management*	5. Menggunakan *user management*
6. *Desktop interface* bernama “Git GUI”	6. *Desktop interface* “GitHub Desktop”
7. Pesaing: Mercurial, Subversion, IBM, Rational Team, Concert, dan ClearCase	7. Pesaing: GitLab dan Atlassian BitBucket
8. *Open sourced licensed*	8. Pilihan: pengguna gratis atau pengguna berbayar

Git vs GitHub

- **User interface** pada GitHub lebih menarik dan mudah dipahami oleh pengguna awal.
- Fitur lain GitHub: kita dapat membaca berbagai **blog dan feed** yang dibuat oleh sesama pengguna (forum diskusi para programmer)
- GitHub dan Git pada konsep kerjanya hampir sama dengan Dropbox dan Google Drive, hanya saja Git dan GitHub bekerja untuk **mengolah kode script**. Sedangkan DropBox dan Google Drive bertugas untuk mengolah kata.
- GitHub dan Git merupakan **alat version control**, bedanya github sekaligus dilengkapi **penyimpanan cloud**.

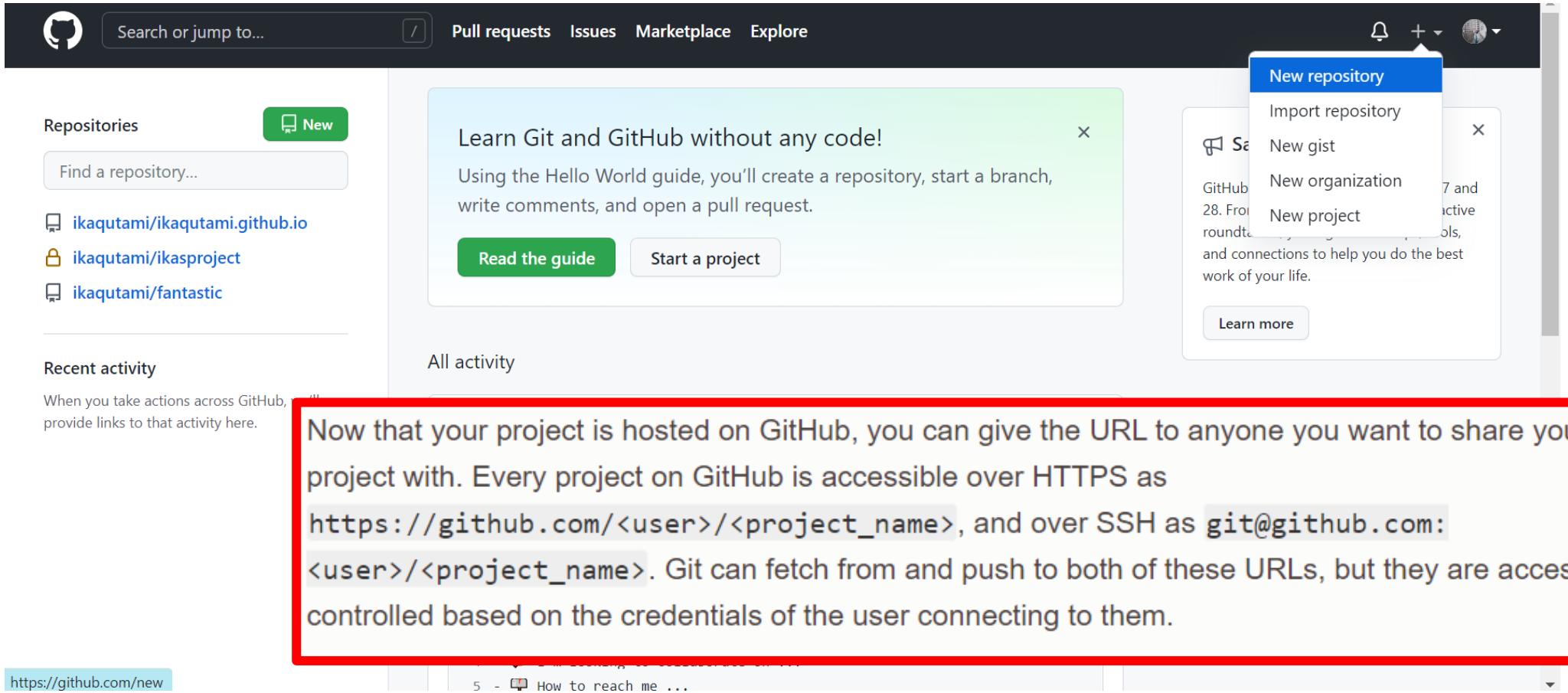
Steps

1. Join account
2. Verifikasi Alamat E-mail



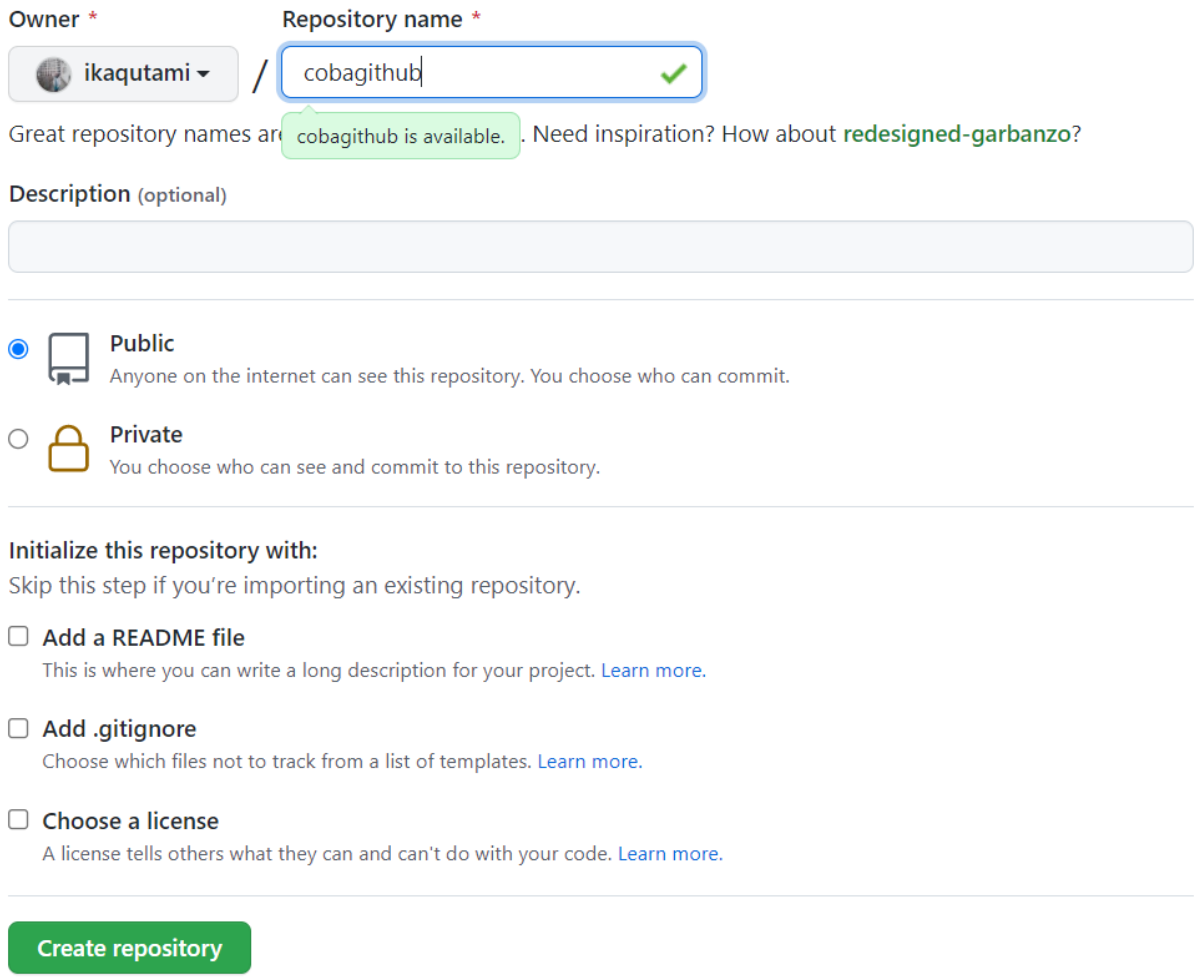
Maintaining a Project

- Creating a New Repository

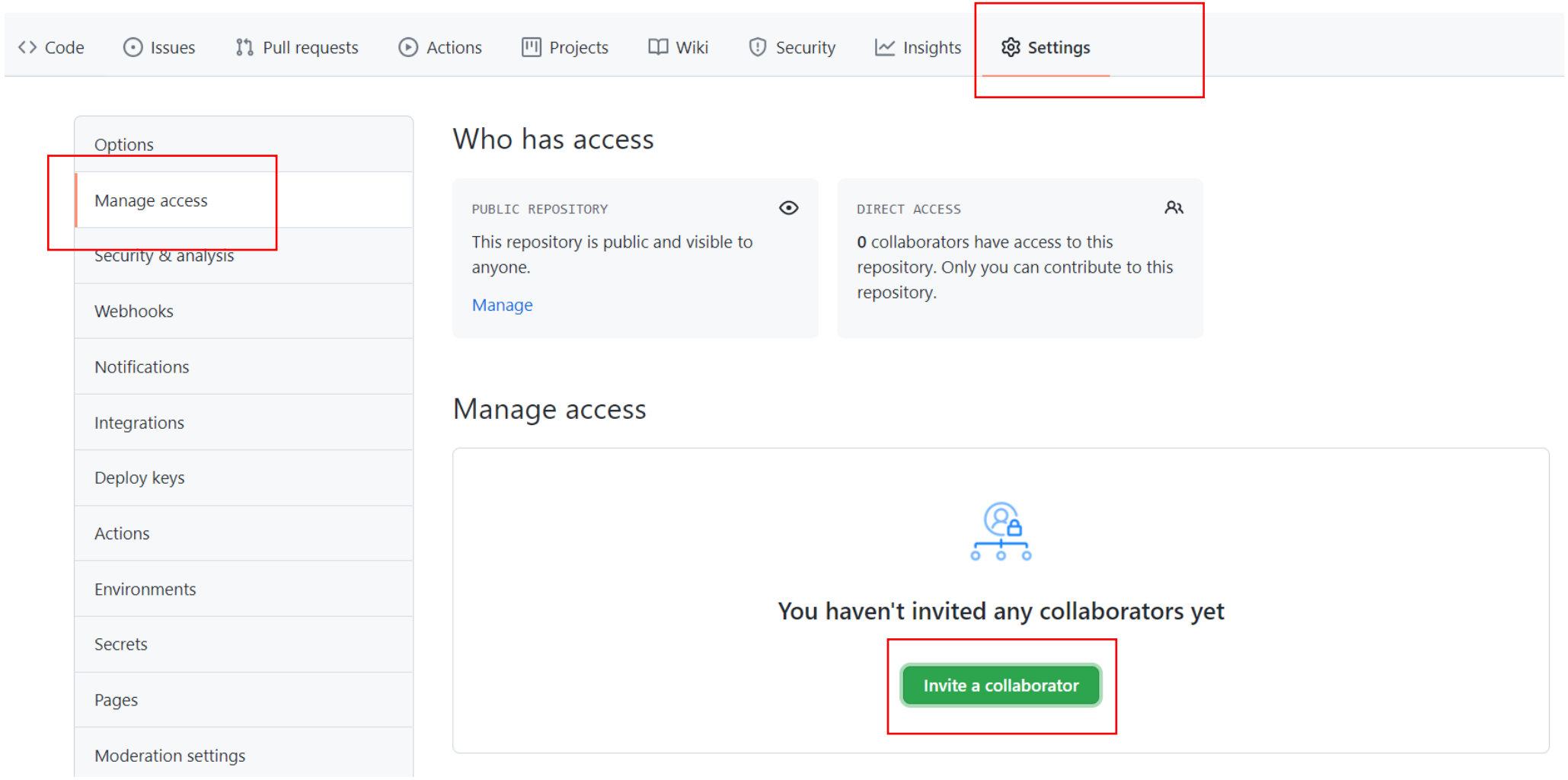


Maintaining a Project

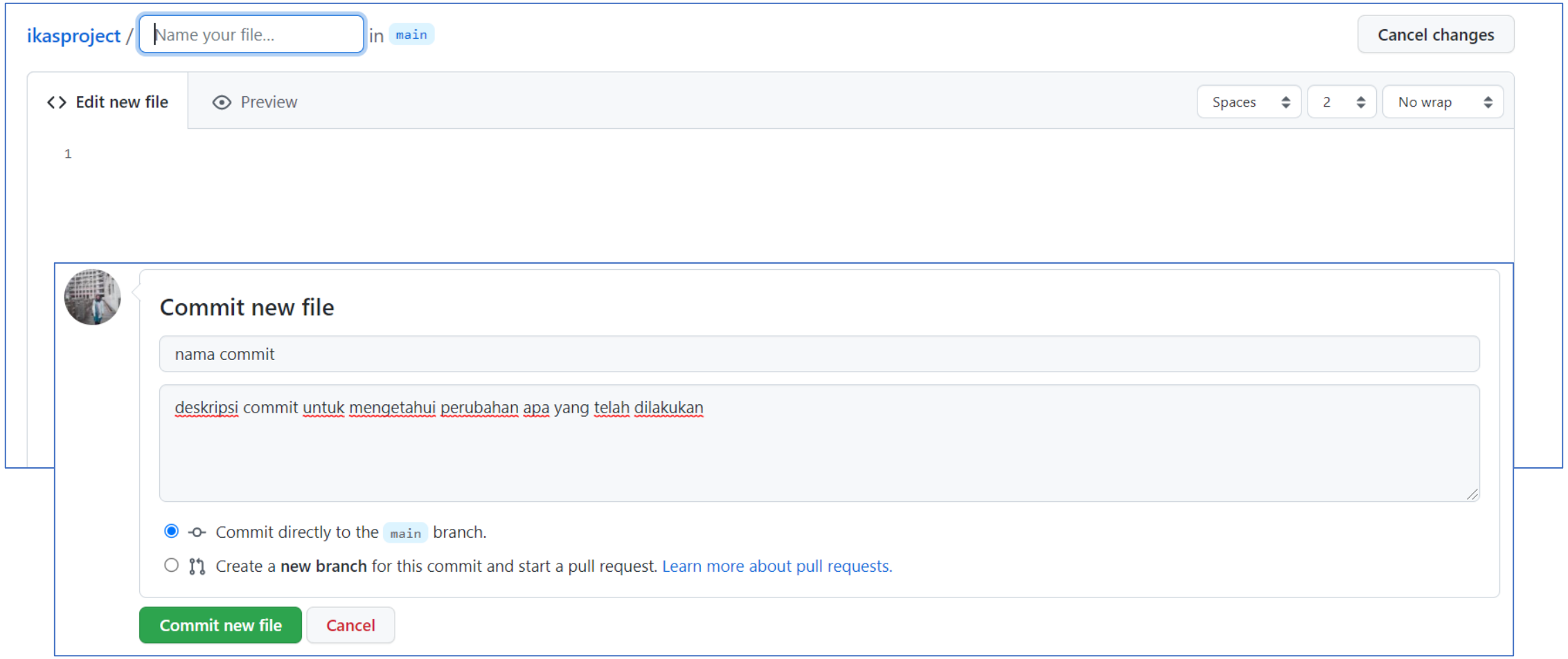
- File **“Readme”** selalu ada di setiap repository untuk menjelaskan penjelasan singkat/detail, cara penggunaan, dan sebagainya.
- Dibuat secara otomatis jika checklist tombolnya.



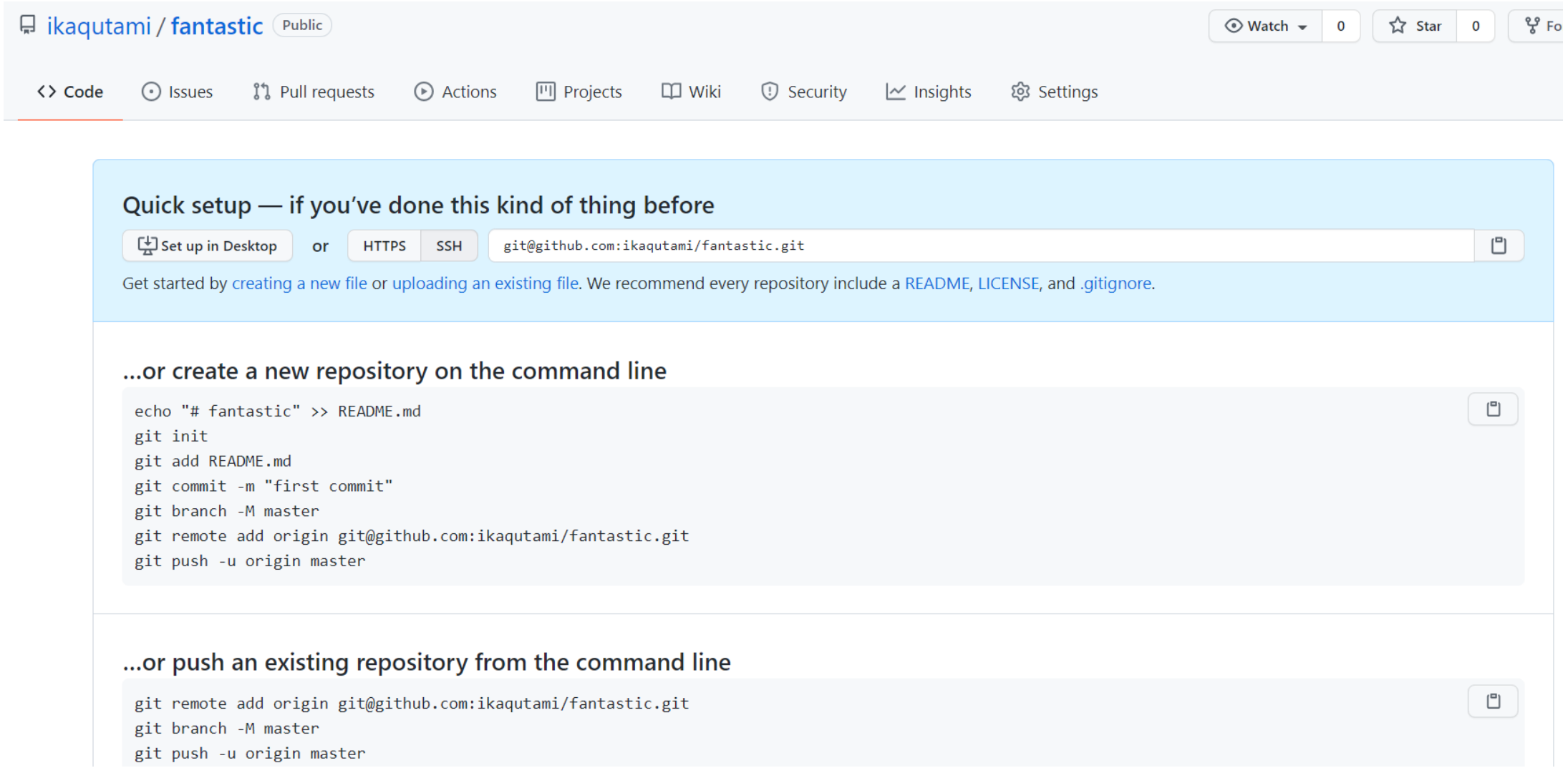
Add Collaborators



Creating new file

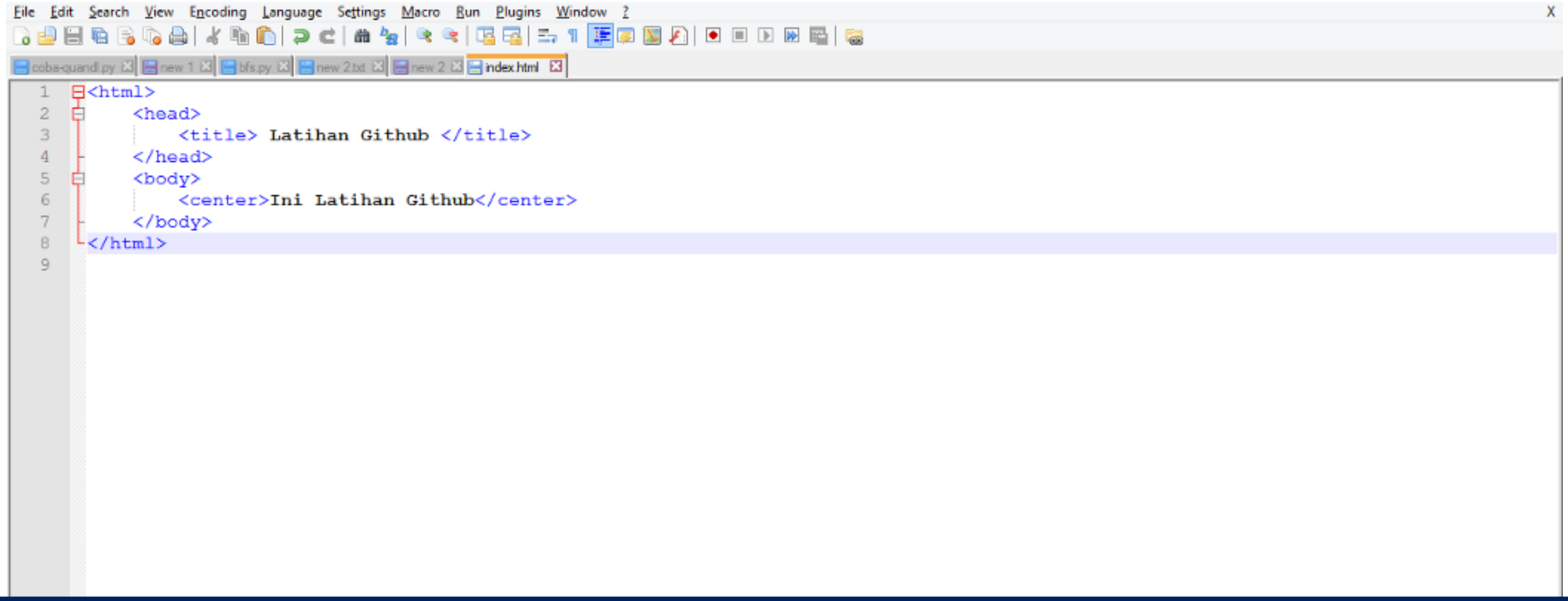


Creating new file



Push File

- Buat folder
- Buat file baru misal **index.html**
- Edit file **index.html** dan masukkan kode berikut

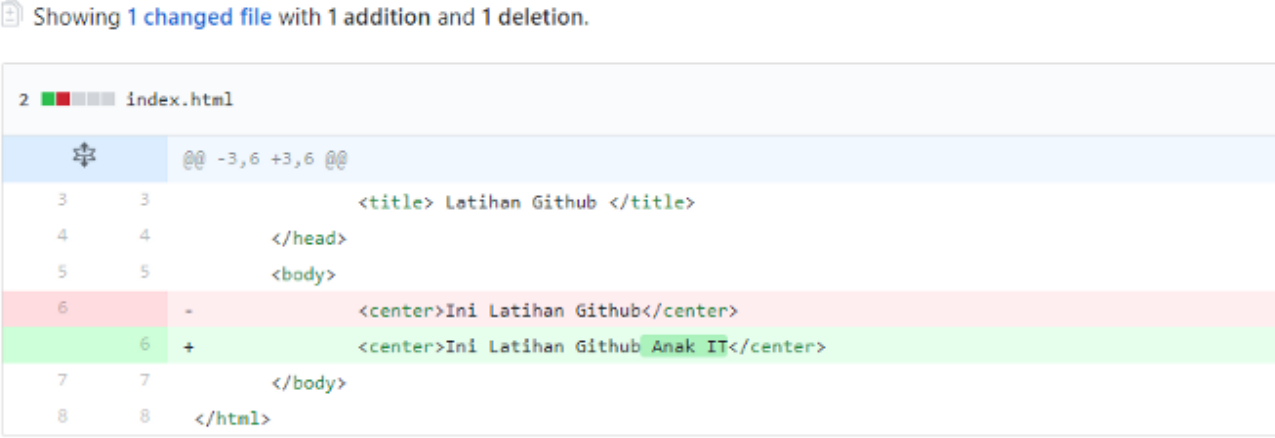


Push File

- Klik kanan dan klik "*Git Bash Here*"
- **Git init -> git add -> git commit -m "first commit"**
- Git remote add origin https://github.Com/username/latihan-github.git
- Git push origin master
- Cek file di github
- Edit file lokal, ulang proses dari atas untuk commit2 selanjutnya.

9 lines (8 sloc) | 121 Bytes — Raw | Blame | History

```
<html>
        <head>
                <title> Latihan Github </title>
        </head>
        <body>
                <center>Ini Latihan Github</center>
        </body>
</html>
```



Showing 1 changed file with 1 addition and 1 deletion.

2 index.html

```
@@ -3,6 +3,6 @@
                <title> Latihan Github </title>
        </head>
        <body>
-               <center>Ini Latihan Github</center>
+               <center>Ini Latihan Github Anak IT</center>
        </body>
</html>
```

Thank you..

References:

https://ocw.mit.edu/ans7870/6/6.005/s16/classes/05-version-control/index.html

https://git-scm.com/book/en/v2

UNAIR HEBAT WORLD CLASS UNIVERSITY #521-530 QS TOP UNIVERSITY 2021

Week 4
Cloud Computing

Muhammad Noor Fakhruzzaman, S.Kom., M.Sc.

MK Algoritma Pemrograman 2
Prodi S1 Teknologi Sains Data
Fakultas Teknologi Maju dan Multidisiplin
Universitas Airlangga

Outline

- Apa itu Cloud Computing?
- Service dan Deployment Models
- Arsitektur
- Security

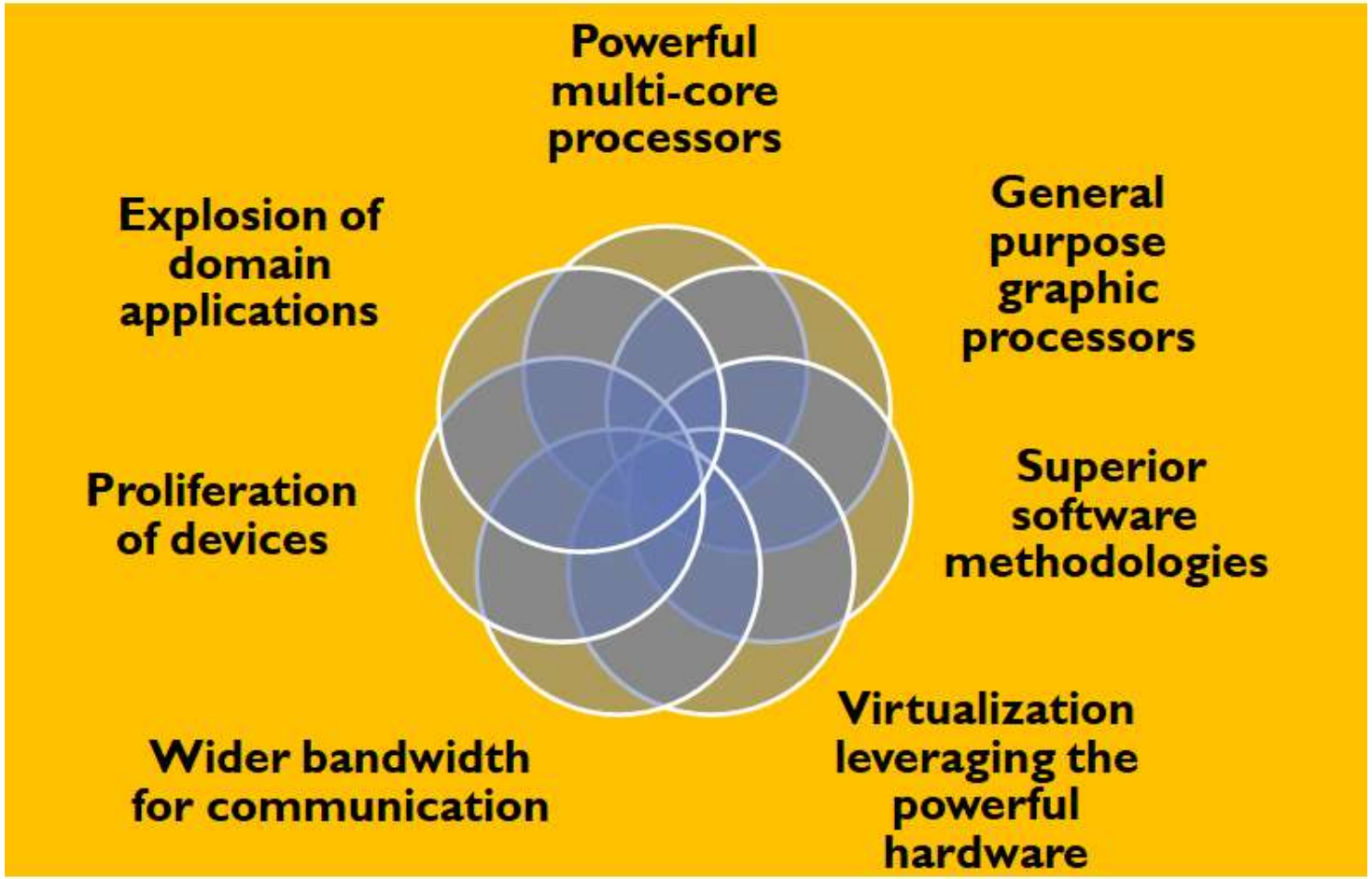
Cloud Computing

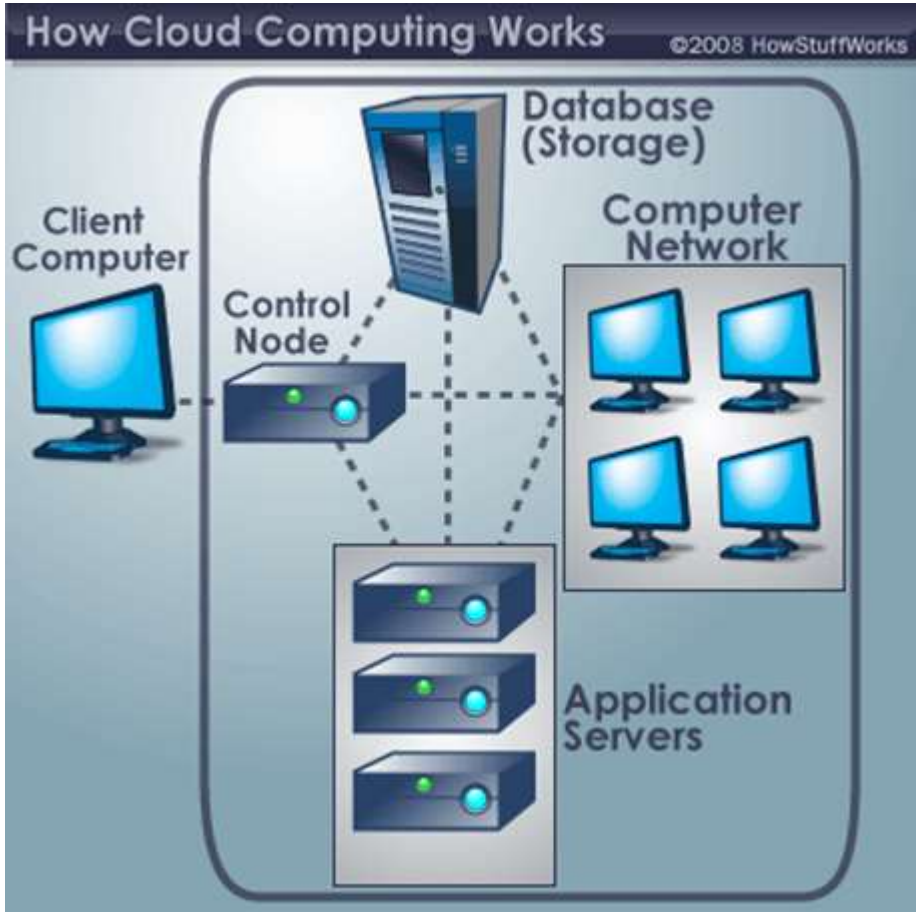
- Literally, komputasi awan
- Proses komputasi berbasis Internet dimana sejumlah server terhubung oleh jaringan internet dan dapat berbagi resource dan data. Server tidak tersentral dan aktivitas komputasi tidak tersentral melainkan dibagi ke server yang terhubung sesuai kebutuhan
- Mudahnya, seluruh kegiatan terkait komputer yang dijalankan di Internet

Cloud Computing

- Pengguna dapat menjalankan software tanpa terikat hardware yang dimiliki, cukup menggunakan web client / browser
- Seluruh kegiatan berada di Internet / shared server, tidak ada yang disimpan di device lokal
- Contoh: Google Suite, Overleaf, Canva

Cloud Computing



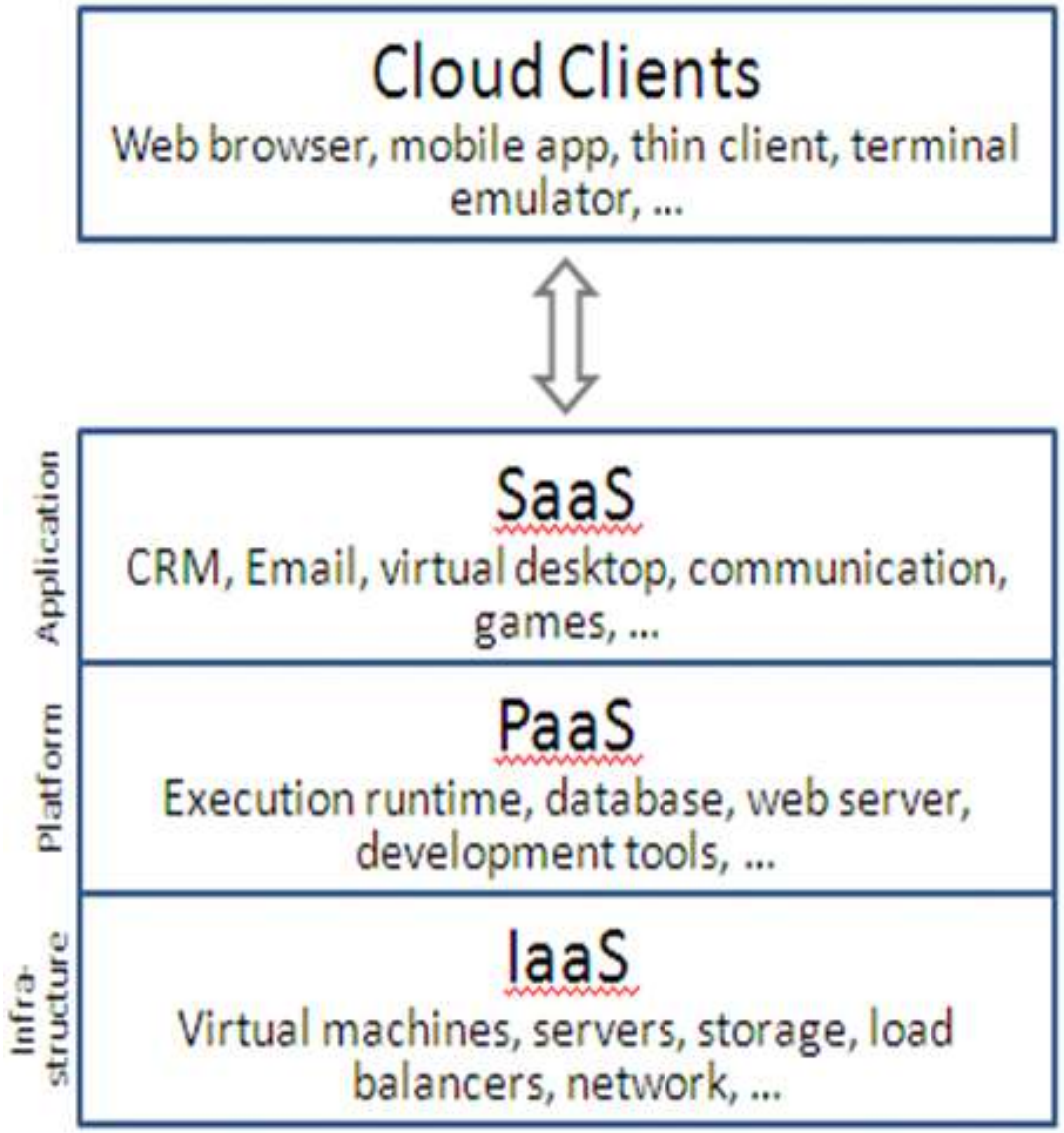


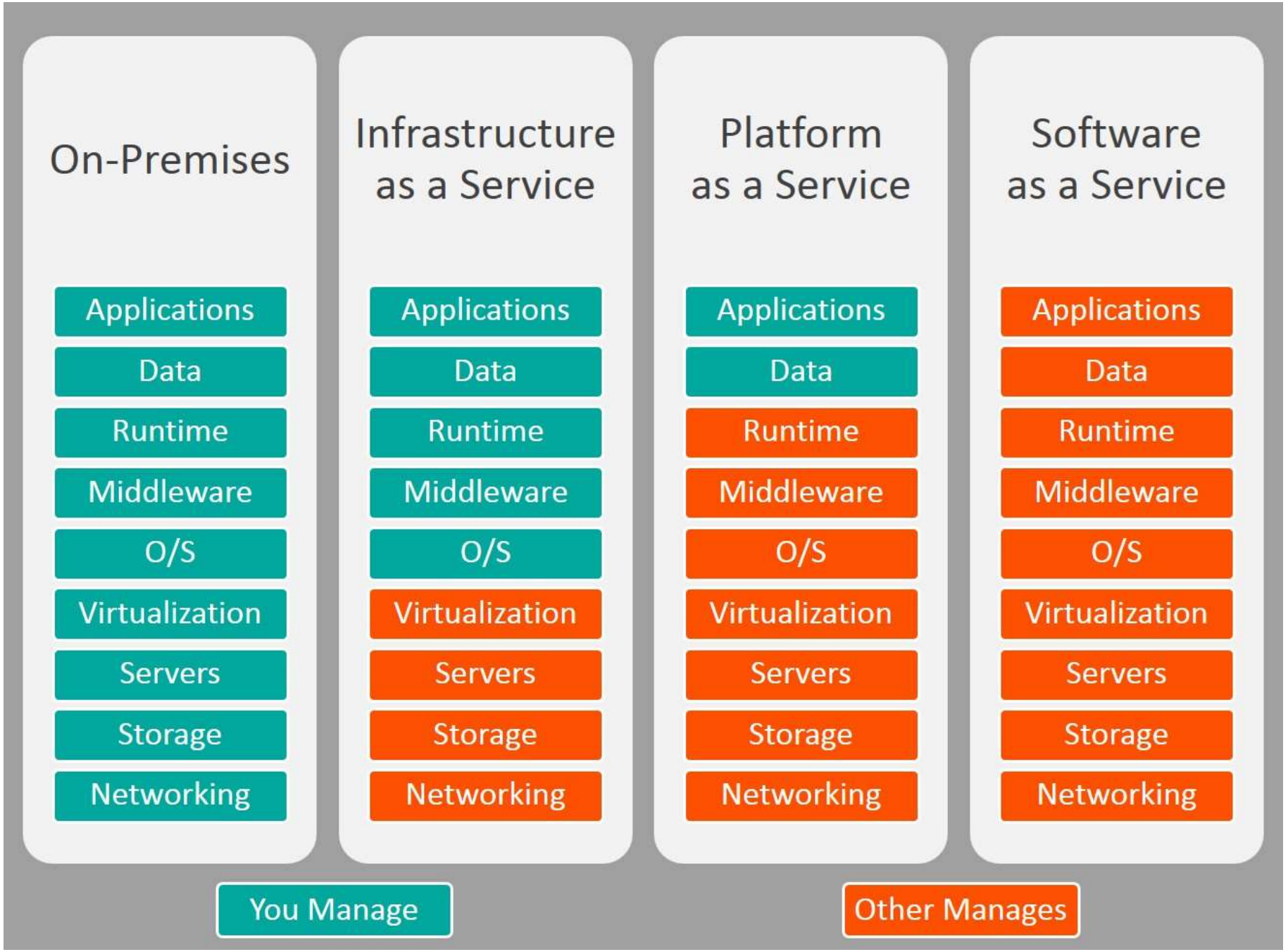
Keuntungan Cloud Computing

1. Relatively cheaper
2. Always Available
3. Device Independent
4. Scalable (mudah dikembangkan jika dibutuhkan di masa depan)
5. Physically secure

Service dan Deployment Models

1. System as a Service (SaaS)
    a. Gmail
    b. Google Colaboratory
    c. Whatsapp Web
2. Platform as a Service (PaaS)
    a. Heroku
    b. Google App Engine
    c. AWS Lambda
3. Infrastructure as a Service (IaaS)
    a. VPS
    b. DigitalOcean
    c. AWS EC2
    d. Google Compute Engine (GCE)



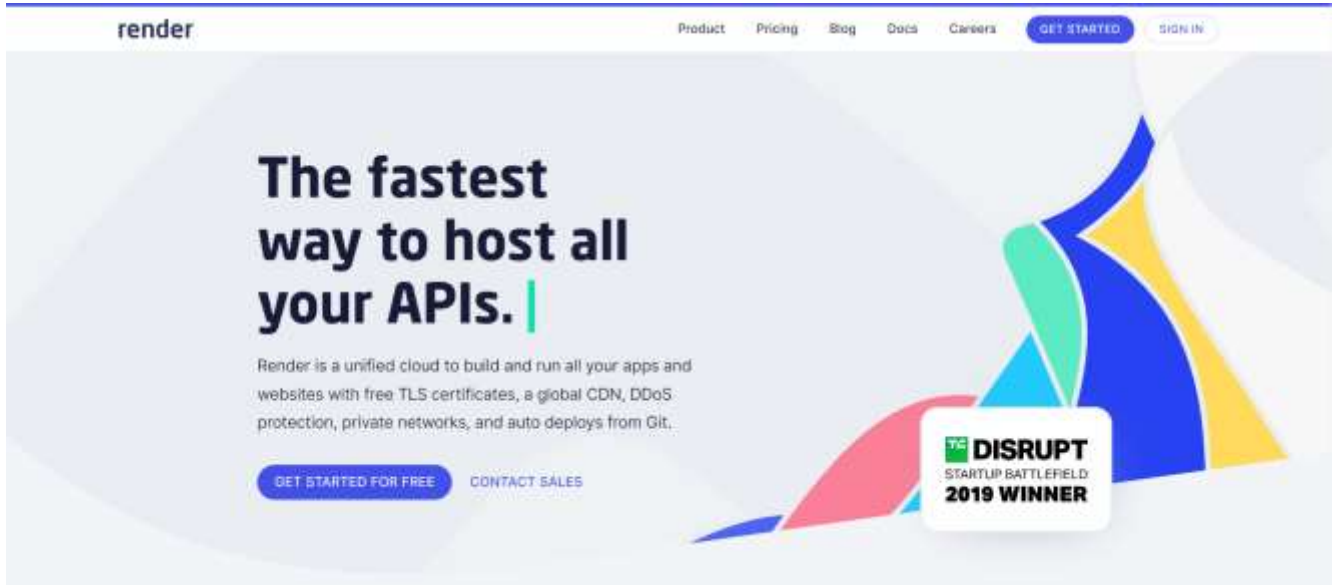

On-Premises
Applications
Data
Runtime
Middleware
O/S
Virtualization
Servers
Storage
Networking
Infrastructure as a Service
Applications
Data
Runtime
Middleware
O/S
Virtualization
Servers
Storage
Networking
Platform as a Service
Applications
Data
Runtime
Middleware
O/S
Virtualization
Servers
Storage
Networking
Software as a Service
Applications
Data
Runtime
Middleware
O/S
Virtualization
Servers
Storage
Networking
You Manage
Other Manages

Infrastructure as a Service

- Layanan yang ditawarkan mirip seperti menyewa device komputer
- Sewa storage, virtual server, dsb
- Highly customizable sesuai kebutuhan
- Pay-as-you-go: harga sewa sesuai resource yang dipakai / dipilih
- Seperti memiliki device kedua tanpa harus mengelola hardware
- Contoh:
  - Virtual Private Server
  - DigitalOcean
  - Amazon EC2
  - Google Compute Engine

Platform as a Service

- Sering digunakan oleh developer yang ingin mengembangkan aplikasi web
- Konfigurasi environment sudah tersedia
- Tidak bisa kustomisasi root / device
- Sebuah platform / environment yang disewa dan siap digunakan untuk aplikasi yang dibuat penyewa
- Contoh
  - Heroku
  - Render
  - Google App Engine

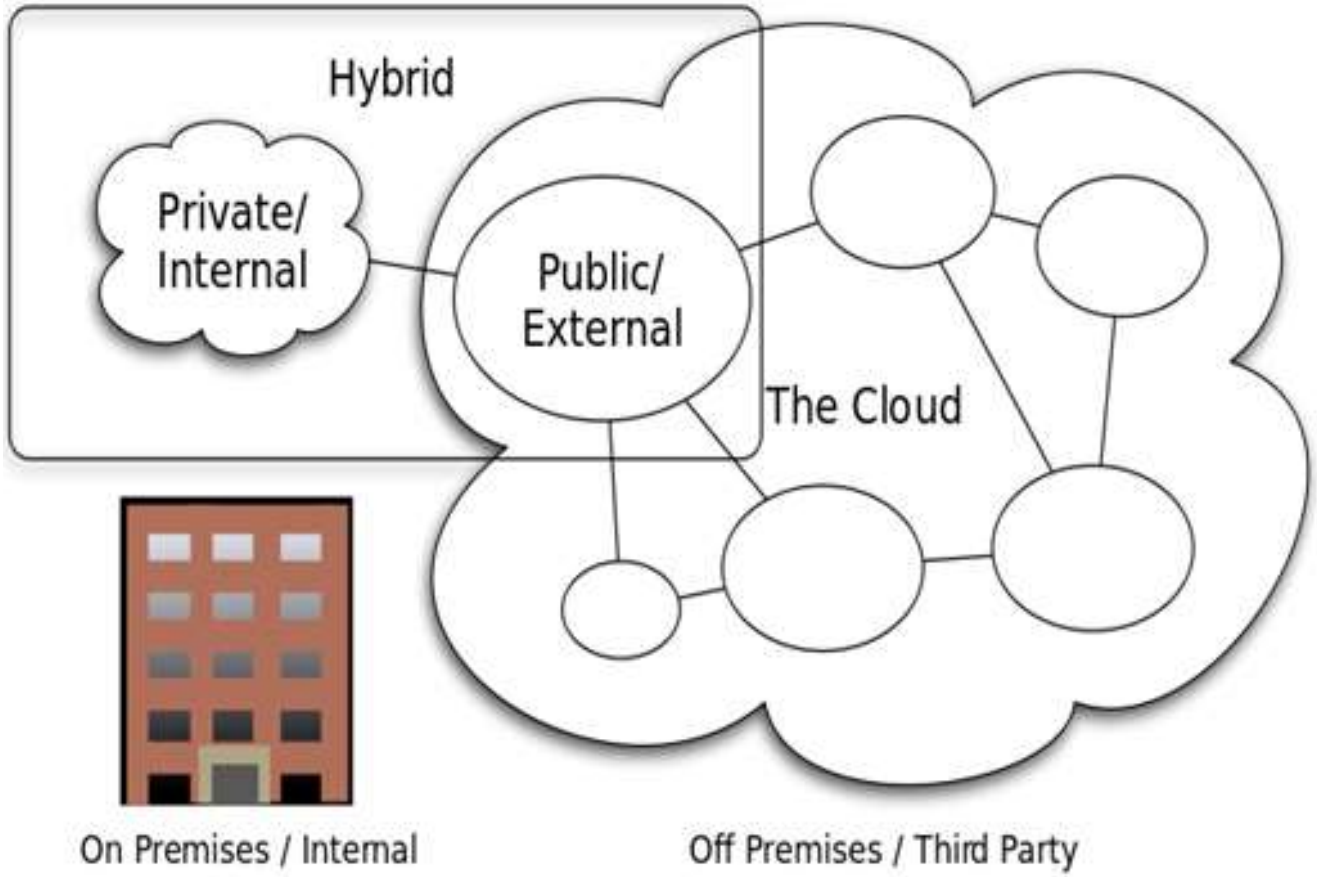


Software as a Service

- Casual user ready
- Aplikasi tinggal pakai tanpa instalasi dan konfigurasi
- Dapat diakses dimanapun
- Kegunaan terbatas sesuai aplikasi
- Contoh:
  - Google Docs, Sheets, Slides
  - Canva
  - Google Colaboratory
  - Overleaf
  - Gmail

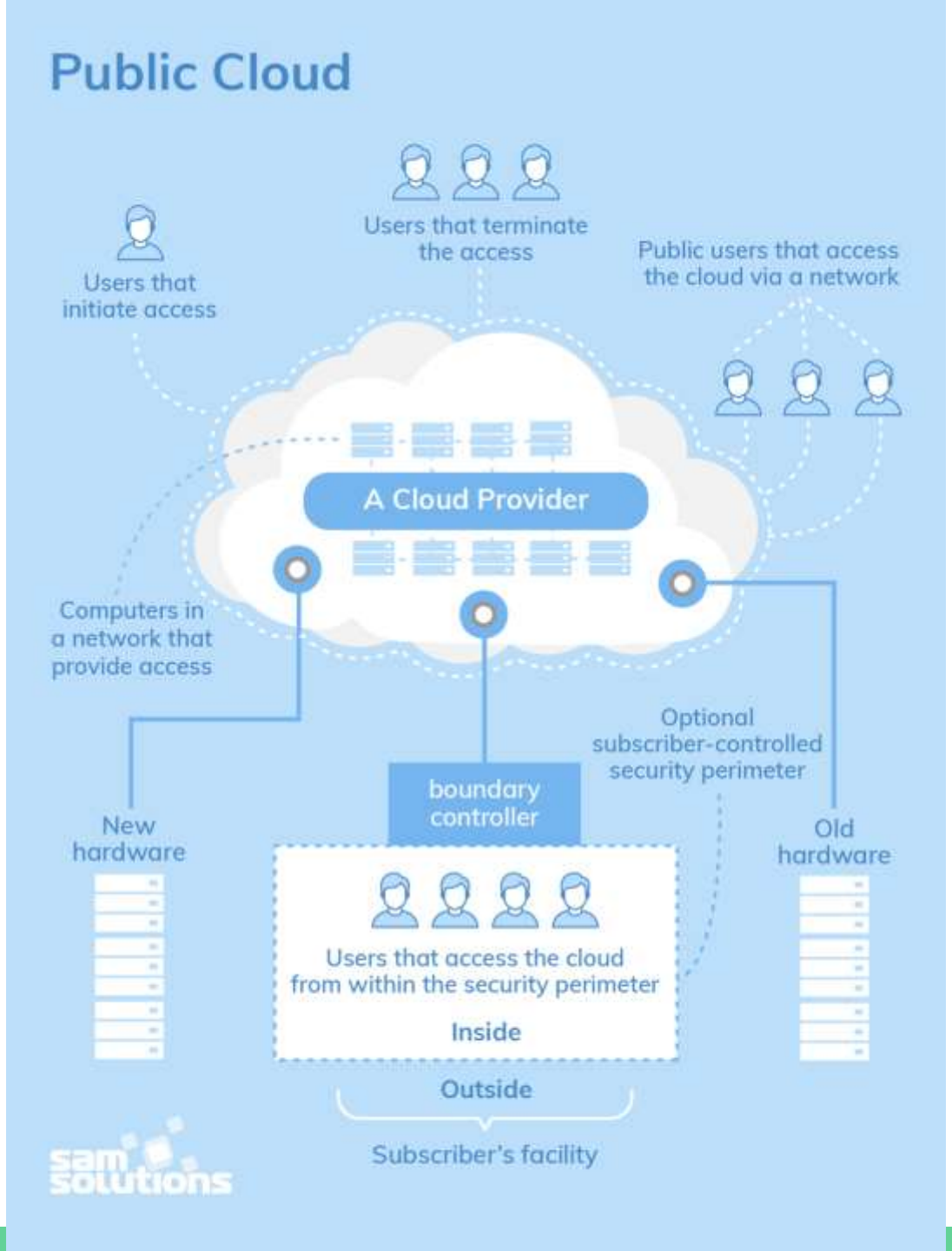
Deployment Models

- Public
- Private
- Hybrid
- Community



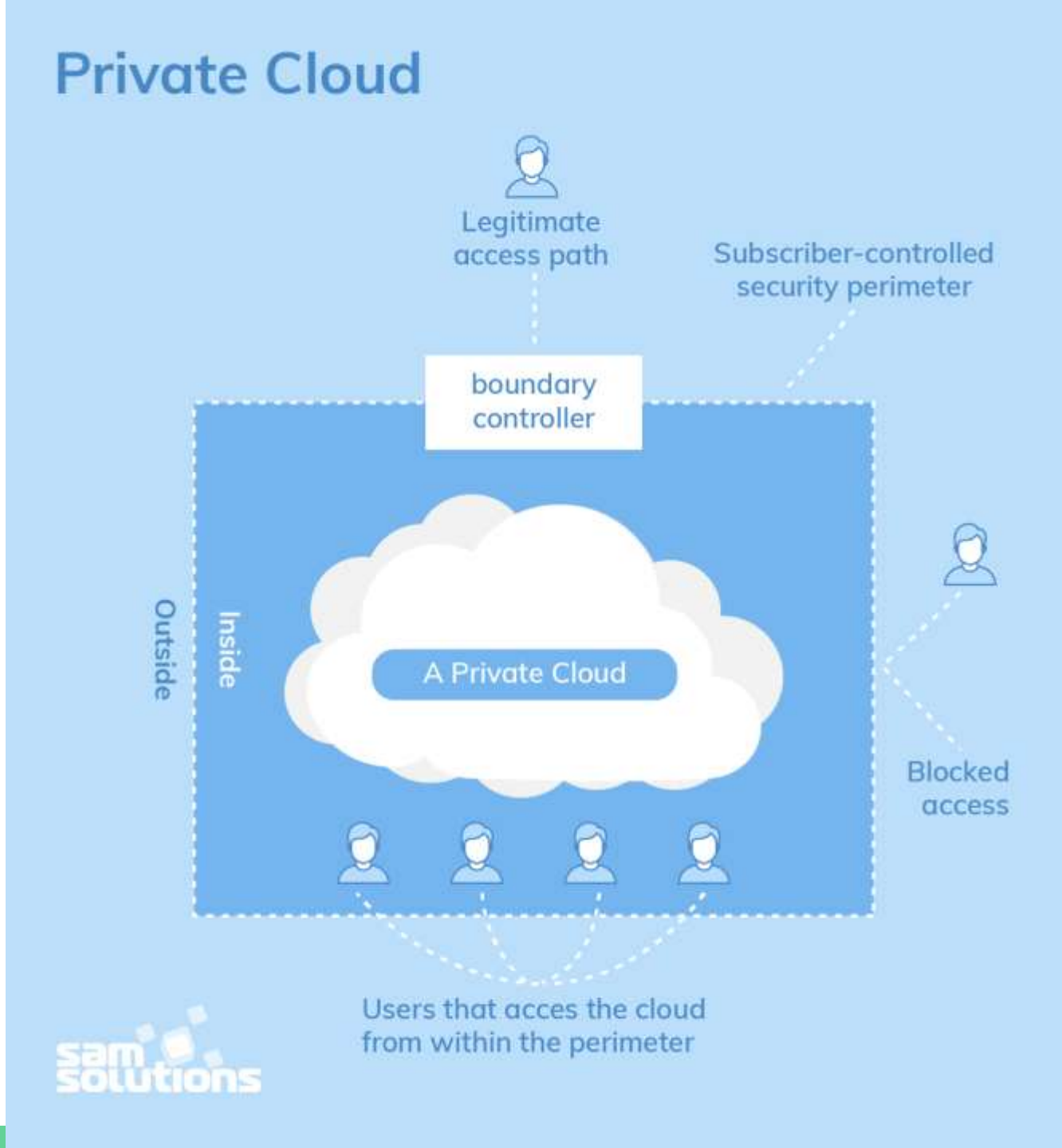
Public Cloud

- Tersedia untuk umum
- Seluruh infrastruktur dikelola oleh pihak penyedia
- Layanan tersedia untuk umum, mungkin tidak memuaskan untuk beberapa kebutuhan spesifik



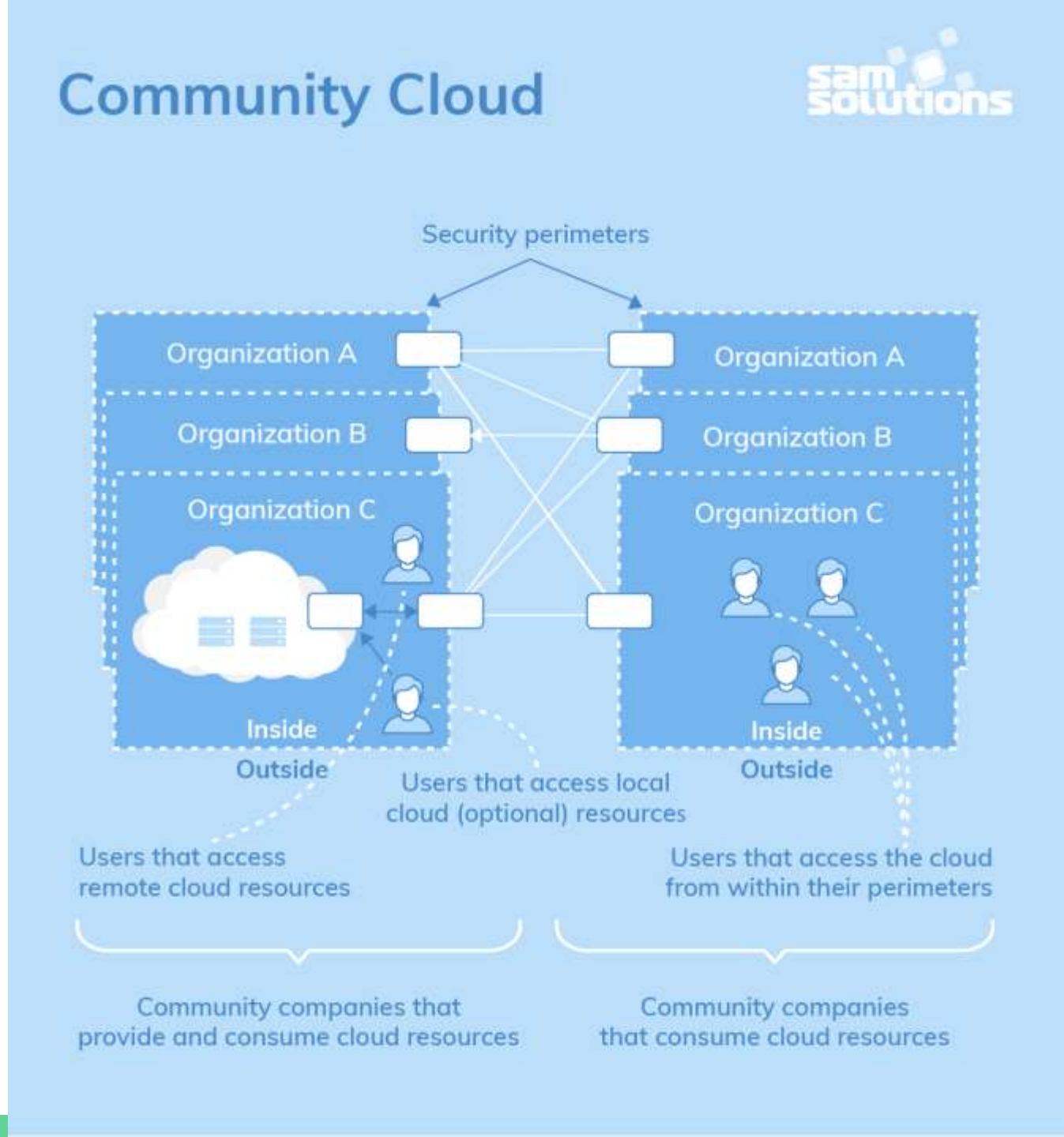
Private Cloud

- Kegunaan sama dengan cloud lainnya
- Hanya bisa diakses dengan jaringan internal (VPN / L2TP)
- Bisa dikelola internal atau vendor



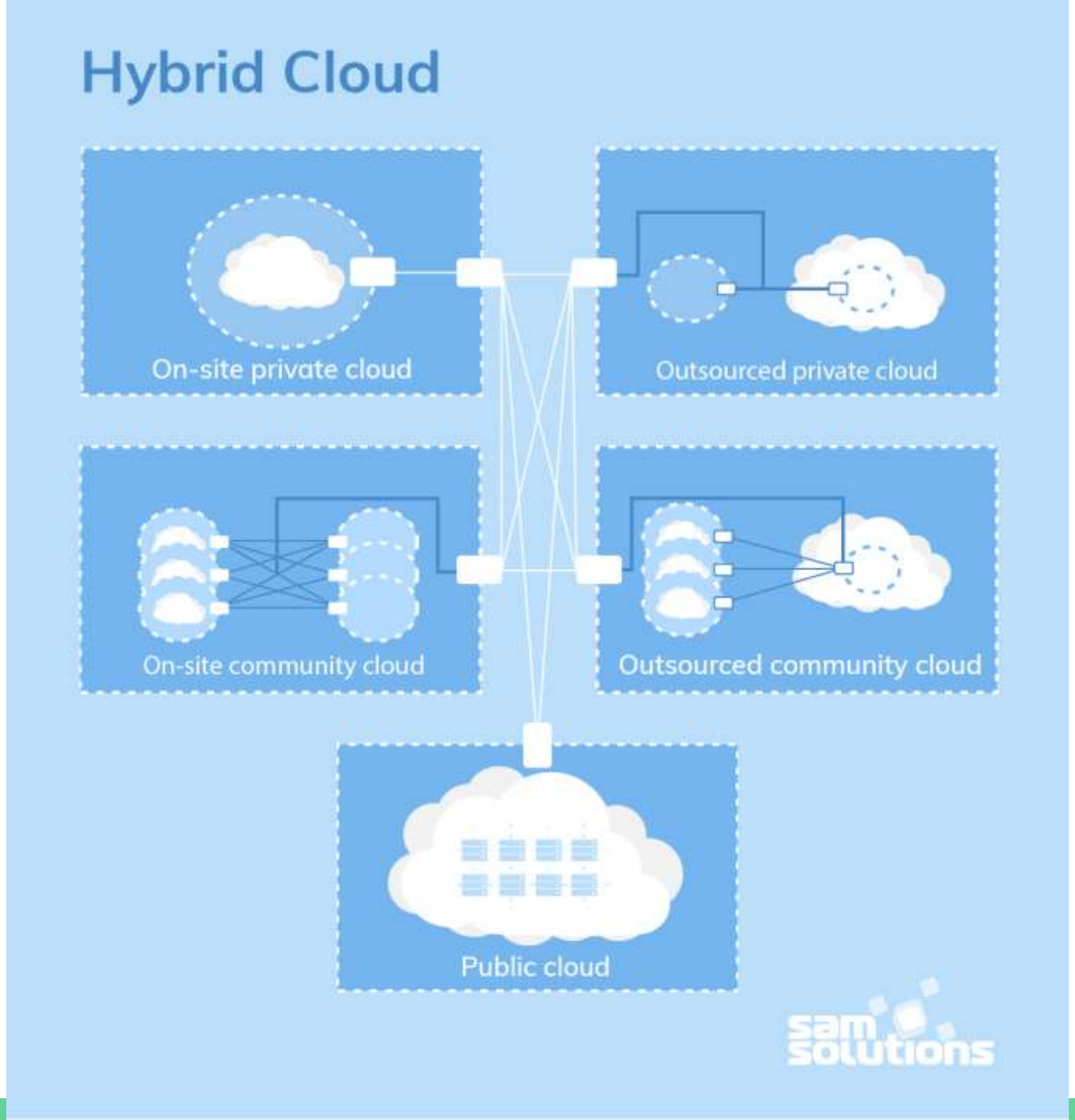
Community Cloud

- Private Cloud yang dikelola bersama dan digunakan bersama



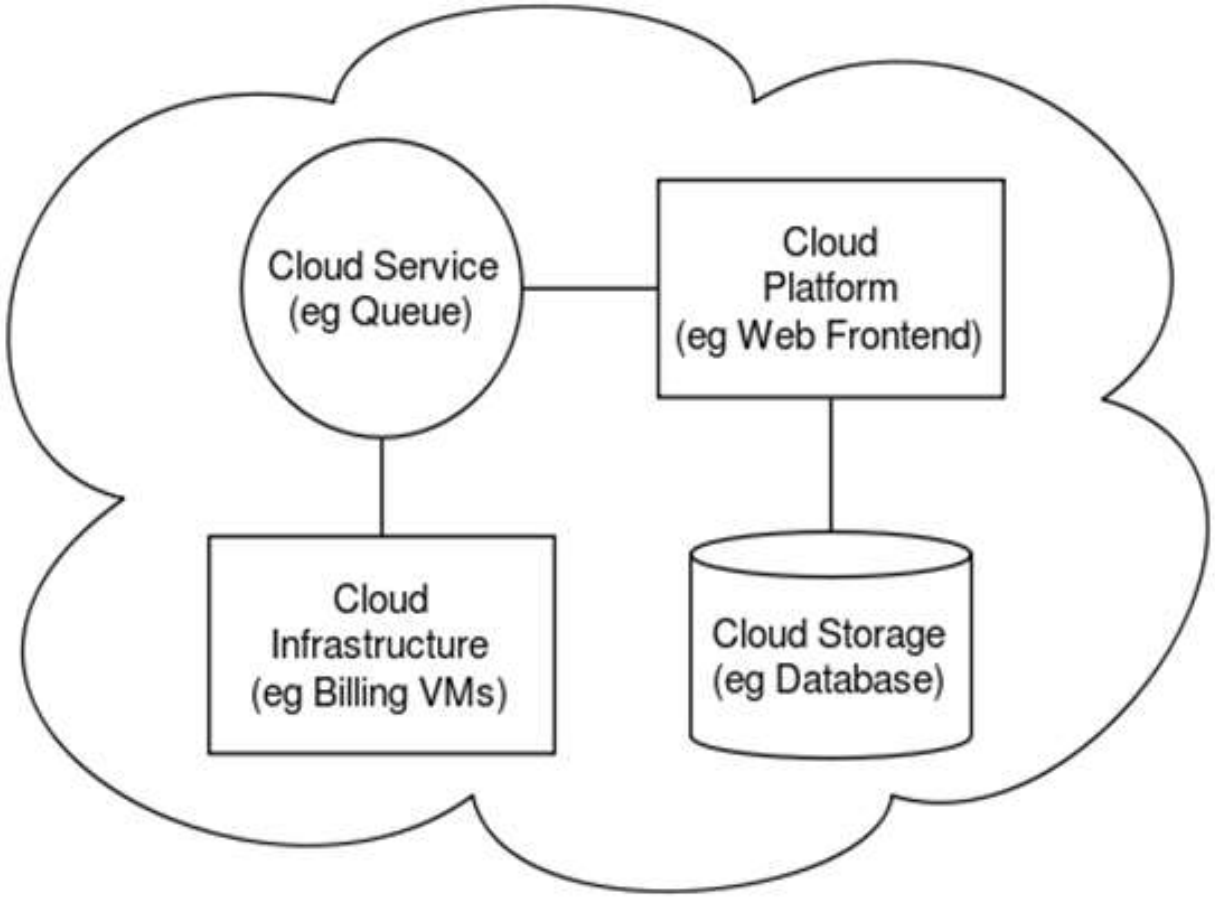
Hybrid Cloud

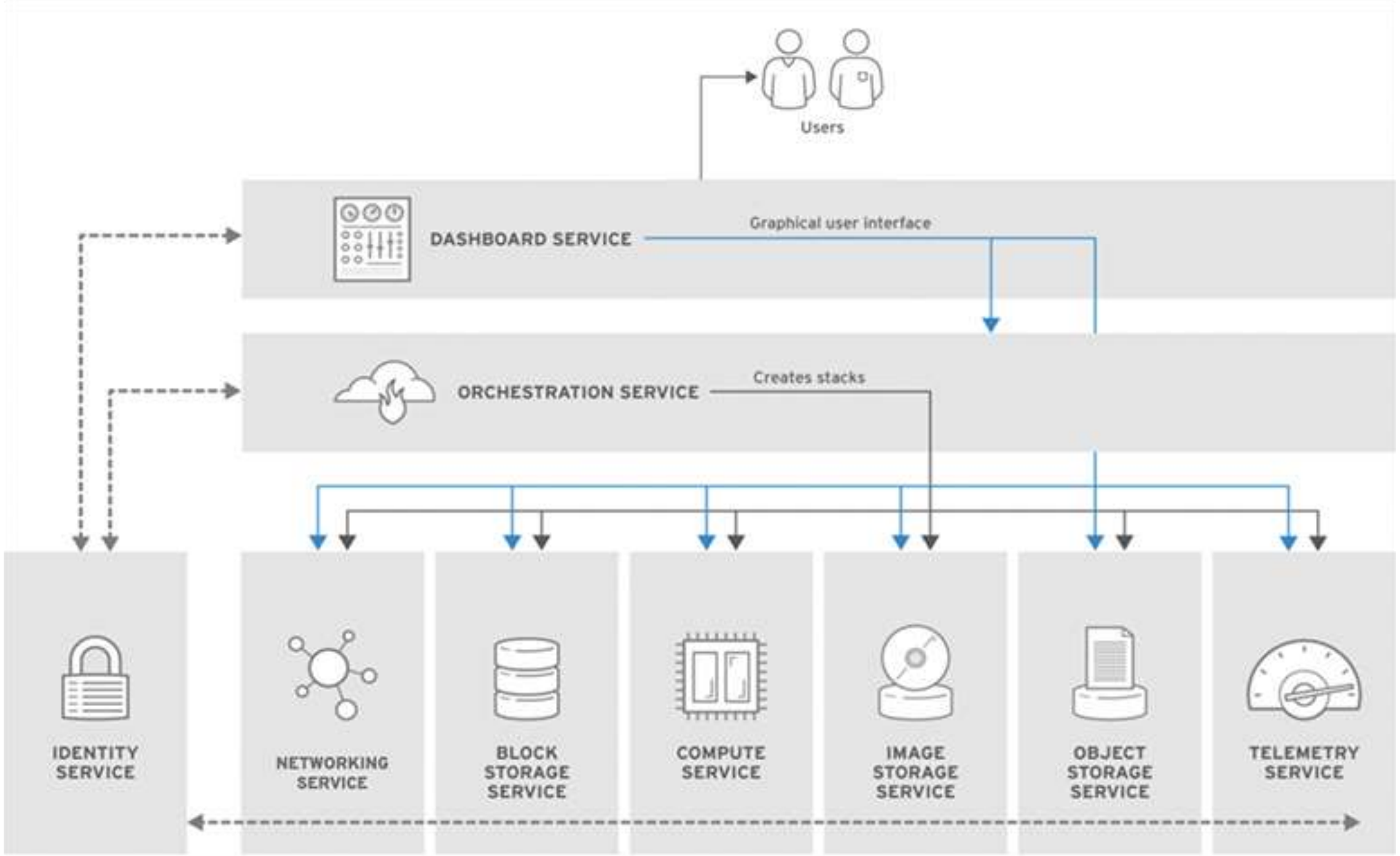
- Gabungan dari public, private, dan community cloud
- Alokasi deployment tergantung dari keamanan yang diperlukan serta kegunaan
- Perusahaan SaaS akan deploy aplikasinya di public cloud, namun aplikasi akuntansi akan deploy di private, dst



Cloud Architecture

- Biasanya terbagi menjadi beberapa tingkatan
- Tiap tingkatan memiliki kekhususan masing masing (dan tim tersendiri)
- Seorang data scientist dapat fokus ke cloud service maupun dashboard tampilan




Users
DASHBOARD SERVICE
Graphical user interface
ORCHESTRATION SERVICE
Creates stacks
IDENTITY SERVICE
NETWORKING SERVICE
BLOCK STORAGE SERVICE
COMPUTE SERVICE
IMAGE STORAGE SERVICE
OBJECT STORAGE SERVICE
TELEMETRY SERVICE

Cloud Security

- Beberapa concern yang diperhatikan:
  - Pembatasan hak akses ke root IaaS (SSH, security token, VPN)
  - Keamanan fisik data
  - Pengelolaan data redundancy (fault tolerance, backups)
  - Legal issues terkait negara tempat sebuah data disimpan / dikelola / digunakan
  - Manajemen identitas
- Sejumlah agen menggunakan virtual cloud server untuk dijadikan zombie / bot DDoS
- Virtual cloud server yang dijadikan bouncing point untuk hacker

Pertanyaan? Diskusi?

Software Architecture

Pertemuan 5
MK Algoritma Pemrograman II

Ika Qutsiati Utami, S.Kom., M.Sc.
M. N. Fakhruzzaman, S.Kom., M.Sc.
Program Studi S1 Teknologi Sains Data
Fakultas Teknologi Maju dan Multidisiplin
Universitas Airlangga Indonesia

Highlight

- Why do we need a software architecture?
- What is a Monolithic Architecture?
- What is a Layered Architecture?
- What is a Microservices Architecture?

Background

- Enterprise architects get paid lots of money because ***architecting a quality software is difficult and requires experience*.**
- How to architect a solution ***with little prior experience***
- There are so many architectures and design patterns. ***What if I choose the wrong one?***
- How can I create an entire architecture ***without knowing all the details about the code I'm going to write?***

Why do we need Software Architecture?

"***Any intelligent fool can make things bigger, more complex, and more violent. It takes a touch of genius—and a lot of courage to move in the opposite direction***" (E.F. Schumacher's book, Small is Beautiful)

As a developer:

- It is always more fun to solve a complex problem in a complex way.

BUT, unfortunately ...

- You get no additional bonus for indirectly solving problems ☹
- You make the real money by solving a complex problem in a **simple way**.

What is Software Architecture?

Designing software architecture is about arranging components of a system to best fit the desired quality attributes of the system.

There is no way to please everyone without sacrificing the quality of the system !!!

Therefore, when designing software architecture, you must decide which quality attributes matter most for the given business problem.

System Quality Attributes

- **Performance** - how long do you have to wait before that spinning "loading" icon goes away?
- **Availability** - what percentage of the time is the system running?
- **Usability** - can the users easily figure out the interface of the system?
- **Modifiability** - if the developers want to add a feature to the system, is it easy to do?
- **Interoperability** - does the system play nicely with other systems?
- **Security** - does the system have a secure fortress around it?
- **Portability** - can the system run on many different platforms (i.e. Windows vs. Mac vs. Linux)?
- **Scalability** - if you grow your userbase rapidly, can the system easily scale to meet the new traffic?
- **Deployability** - is it easy to put a new feature in production?
- **Safety** - if the software controls physical things, is it a hazard to real people?

System Quality Attributes

- **Users:** system is fast, reliable, and available
- **Project manager:** system is delivered on time and on budget
- **CEO:** system contributes incremental value to his/her company
- **Head of security:** system is protected from malicious attacks
- **App support team:** system is easy to understand and debug

System Quality Attributes

- Depending on ***what software you are building or improving***
- Example:
    1. Financial services company
        a. **Security** (prevent clients to lose millions of dollars)
        b. **Availability** (clients need to always have access to their assets)
    2. Gaming/video streaming company (i.e. Netflix)
        a. **Performance** (if games/movies freeze up all the time, nobody will play/watch)

Software Architecture

- So..

> **Building software architecture** is not about finding the best tools and the latest technologies. It's about ***delivering a system that works effectively***.

- There are many architectures to choose, but not all of them are "***beginner friendly***" and sometimes ***require years of experience to implement correctly***.
- Example:
  - Effective peer-to-peer architecture (i.e. Bitcoin, Bittorrent)

Why do we need an architecture?

Software Architecture is important for the success of a project.

1. Architecture enables quality attributes
2. Architecture enables communication among stakeholders
3. Architecture focuses on the assembly rather than creation of components
4. Architecture restricts design choices, enabling creativity in other areas

3 Common Architectures:

1. Monolithic
2. Layered or n-tier
3. Microservice

Why 3 common architectures

- Beginner friendly
- Usually suffice no matter what technology you use
- Come up the most often in software communities

Monolithic

- **A monolithic architecture** describes an architecture where all of the following components are bunched into one codebase:
  - **Views** (ex: HTML, CSS, Javascript)
  - **Application/Business Logic** (ex: ExpressJS)
  - **Data Access/Database** (ex: MongoDB)
- May seem ineffective BUT not all industry believe it is useless.
- ***Jika baru memulai dan sistem belum kompleks, bisa pilih monolithic***

Monolithic

- **Application Structure: CENTRALIZED** (kode untuk akses ke database, ke server, dan API endpoint jadi satu codebase tanpa dipisah-pisah)
- Centralization is not sustainable into the future.
- ***BUT, gak masalah kalau baru belajar dan memulai, setelah lebih jago bisa mencoba refactoring kodenya menjadi arsitektur yang lebih mudah dikelola misalnya menggunakan layered architecture dsb.***

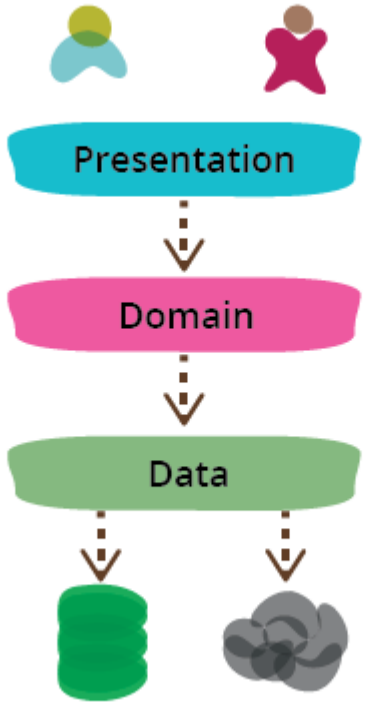
```
// ============================================
// ==========  DATABASE CONNECTION  ===========
// ============================================
// Connect to running database
mongoose.connect(
  `mongodb://${process.env.DB_USER}:${process.env.DB_PW}@127.0.0.1:27017/monolithic_app_db`,
  { useNewUrlParser: true }
);

// User schema for mongodb
const UserSchema = mongoose.Schema(
  {
    name: { type: String },
    email: { type: String },
  },
  { collection: "users" }
);

// Define the mongoose model for use below in method
const User = mongoose.model("User", UserSchema);

function getUserByEmail(email, callback) {
  try {
    User.findOne({ email: email }, callback);
  } catch (err) {
    callback(err);
  }
}
```

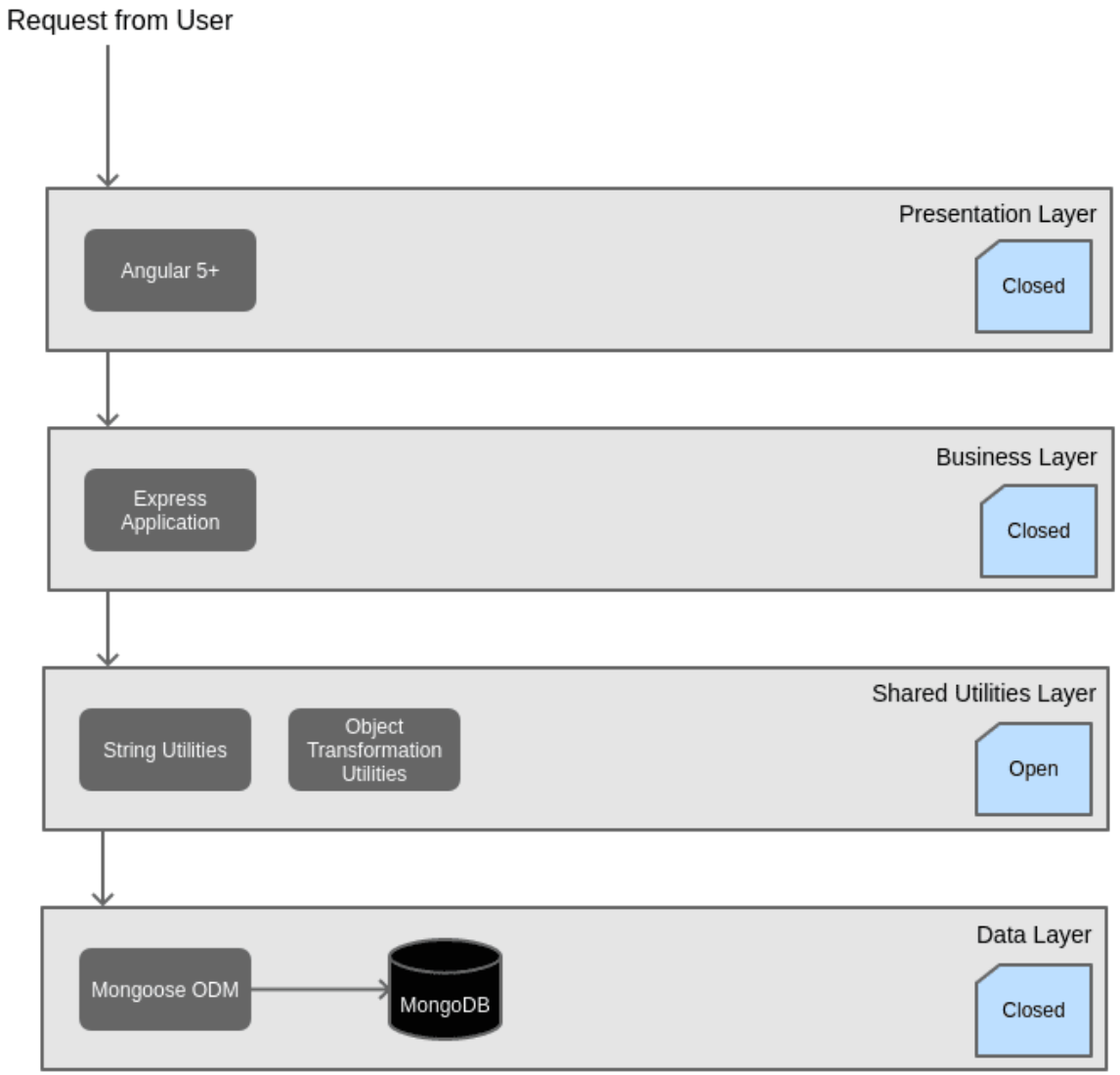
```
// set the view engine to ejs
app.set("view engine", "ejs");

// index page
app.get("/", function (req, res) {
  res.render("home");
});

// ============================================
// ============  API ENDPOINT  ================
// ============================================
app.post("/register", function (req, res) {
  const newUser = new User({
    name: req.body.name,
    email: req.body.email,
  });

  newUser.save((err, user) => {
    res.status(200).json(user);
  });
});

// ============================================
// ==============  SERVER =====================
// ============================================
app.listen(8080);
console.log("Visit app at http://localhost:8080");
```

Layered or n-tier

- Your monolithic application will start **getting big** -> you will start **hiring people** -> it will quickly **become a mess** -> you need **refactoring your monolith**
- The layered architecture splits your application into common layers:
    1. **Presentation Layer** (User Interface)
    2. **Business Layer** (Logika Bisnis)
    3. **Data Access Layer** (Database)
- The point is to create a **SEPARATION OF CONCERNS**

Layered or n-tier





Layered or n-tier

- **Application flow (example):**
    1. Presentation layer makes a call from an HTML user form
    2. Presentation layer javascript processes the form and executes a call to the business layer
    3. Business layer processes the form info and makes a call to the data access layer
    4. Data access layer processes the information and makes a query to the database for the user
    5. Data access layer returns the information to the business layer
    6. Business layer returns the information via HTTP to the presentation layer
    7. Presentation layer renders the view with the new information

Layered or n-tier

1. Presentation layer makes a call from an HTML user form

```
<!-- File: home.ejs -->

<!-- On form submit, home.ejs executes the getDataFromBusinessLayer() function -->

<form id="emailform" onsubmit="getDataFromBusinessLayer()">
  <input name="email" id="email" placeholder="Enter email..." />
  <button type="submit">Load Profile</button>
</form>
```

2. Presentation layer javascript processes the form and executes a call to the business layer

```
// File: presentation-layer-user.js

function getDataFromBusinessLayer() {
  event.preventDefault();
  const email = $("#email").val();

  // Perform the GET request to the business layer
  // ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  $.ajax({
    url: `http://localhost:8081/get-user/${email}`,
    type: "GET",
    success: function (user) {
      // Render the user object on the page
      // Ommitted for brevity
    },
    error: function (jqXHR, textStatus, ex) {
      console.log(textStatus + "," + ex + "," + jqXHR.responseText);
    },
  });
}
```

3. Business layer processes the form info and makes a call to the data access layer

```
// File: business-layer-user.js

app.get("/get-user/:useremail", function (req, res) {
  // Makes a call to the data access layer
  // ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  const user = User.getUserByEmail(req.params.useremail, (error, user) => {
    res.status(200).json({
      name: user.name,
      email: user.email,
      profileUrl: user.profileUrl,
    });
  });
});
```

4. Data access layer processes the information and makes a query to the database for the user

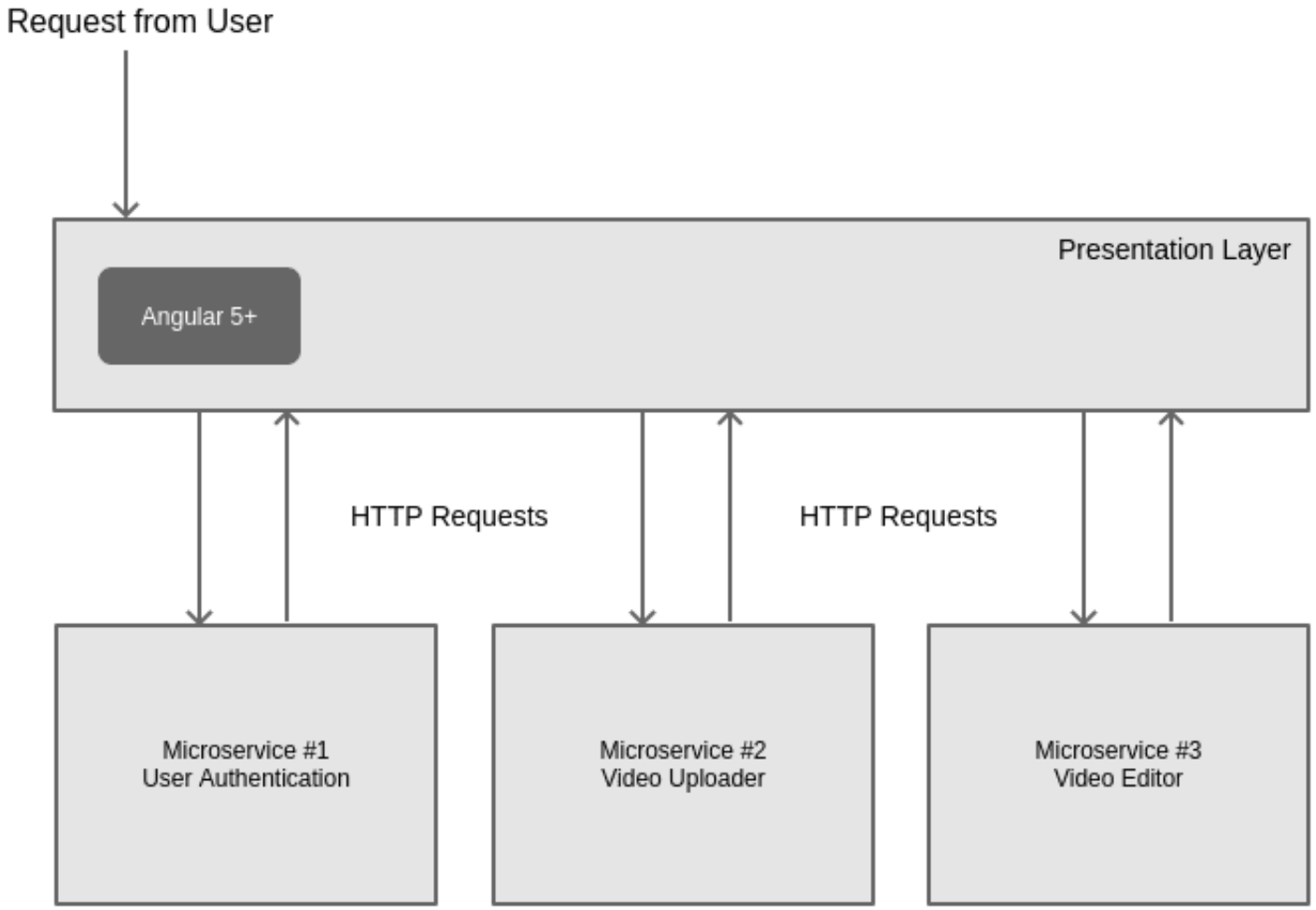
```
// File: data-layer-user.js

module.exports.getUserByEmail = (email, callback) => {
  try {
    // Makes a call to the database
    // ^^^^^^^^^^^^^^^^^^^^^^^^^^^^
    User.findOne({ email: email }, callback);
  } catch (err) {
    callback(err);
  }
};
```

5. Data access layer returns the information to the business layer
6. Business layer returns the information via HTTP to the presentation layer
7. Presentation layer renders the view with the new information

Microservices

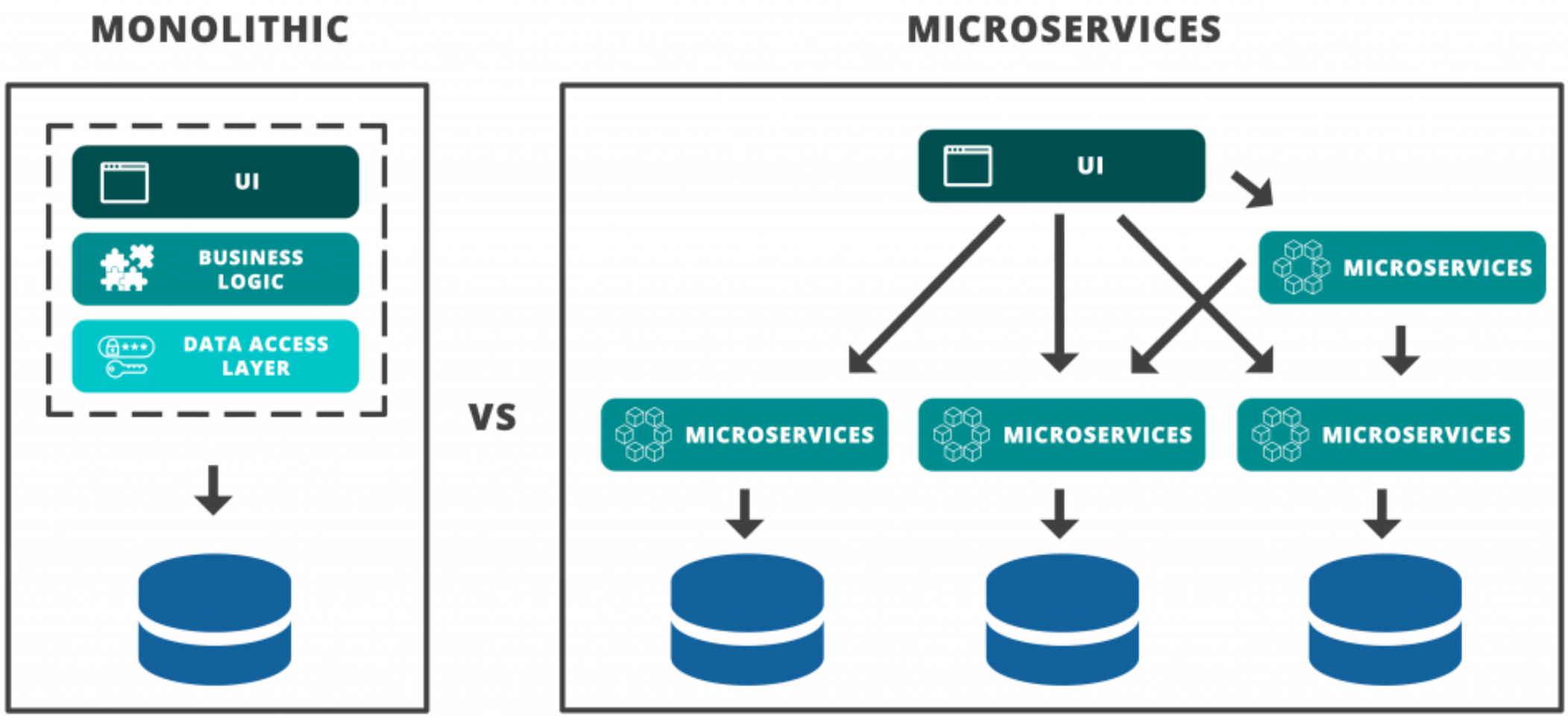
- Architecture untuk aplikasi berbasis cloud dan **bersifat terdistribusi** (perubahan pada program yang di lakukan oleh satu tim developer tidak menganggu keseluruhan aplikasi).
- Aplikasi di bangun sebagai **sekumpulan service** dan setiap layanan berjalan dalam processnya sendiri.
- Setiap fungsi disebut sebagai service
- Cara berkomunikasi melalui **API (Application Programing Interface).** masing-masing app

Microservices



Microservices

Cepat tanggap kebutuhan pasar	Siklus pengembangan dipersingkat, penerapan dan *update* lebih cepat.
Scalable	Jika ada permintaan untuk service tertentu, dapat menerapkan di beberapa server, infrastruktur, untuk memenuhi kebutuhan.
Handal	*Service Independent:* jika salah satu bagian gagal, seluruh aplikasi tidak akan mati, tidak seperti model aplikasi *Monolithic*.
Aksesibilitas	Karena aplikasi yang lebih besar dipecah menjadi bagian-bagian yang lebih kecil, developer dapat lebih mudah memahami, memperbarui, dan menyempurnakan bagian sehingga menghasilkan siklus pengembangan yang lebih cepat.
Lebih terbuka	Penggunaan API sehingga bebas untuk memilih Bahasa pemrograman dan teknologi yang diperlukan.



https://datacommcloud.co.id/

Microservices

- Three parts to our microservices architecture (example):
    1. **View Server (localhost:8080)** - This server runs all of the front-end application logic which includes the main index.html file that utilizes multiple microservices.
    2. **User Authentication Server (localhost:8081)** - This server manages all user authentication.
    3. **Game Server (localhost:8082)** - This server controls the game that is played on the screen.
- **Notice how each of the servers run independently on different ports. This means you could host them on completely different servers and still make the application work.**

Microservices

- We are talking about **API endpoints** (i.e. the communication between the microservices).
- **Goal of microservices?**
  - To understand why a user authentication microservice might be useful, imagine a large company that offers a wide variety of services to its users.
  - **Ex: Google,** because you not only use your login credentials for Gmail and other core Google services but you also use it to log into YouTube and many other applications.

Microservices (cont..)

- **Imagine if Google implemented a user authentication scheme in each individual application!!**
- This is highly **inefficient**, so instead, Google created a **"microservice"** that functions like user authentication for not only Google applications, but an increasingly large number of 3rd party applications.
- This is made possible because the authentication microservice is decoupled from the underlying infrastructure with **robust APIs**.

Other architectures

1. **Microkernel architecture** (ex: Wordpress)
2. **Peer-to-peer architecture** (ex: Bitcoin)
3. **Event-Driven Architecture** (ex: instant messaging system or chat application)

Conclusion

- Whatever your situation, **there is an architecture out there for you.**
- **BUT.. Remember..** the ultimate goal with architecting software solutions is:
    1. Solve your problems in the **SIMPLEST WAY POSSIBLE**
    2. Design your architecture to address **THE QUALITY ATTRIBUTES** you desire in your system
- **If you can meet these two requirements, you have succeeded.**

Terima Kasih

Referensi:
Zach Gollwitzer

Pemrograman Web Part 1

Pertemuan 6

MK Algoritma Pemrograman II

M. N. Fakhruzzaman, S.Kom., M.Sc.

Program Studi S1 Teknologi Sains Data

Fakultas Teknologi Maju dan Multidisiplin

Universitas Airlangga Indonesia

Tujuan Pembelajaran

1. Mampu memahami konsep dasar Pemrograman Web
2. Mampu memahami istilah-istilah yang ada dalam pemrograman web
3. Mampu menggunakan text editor
4. Mampu mengimplementasikan Struktur Navigasi

Pengertian Website

Website adalah kumpulan halaman digital yang berisi informasi berupa teks, animasi, gambar, suara dan video atau gabungan dari semuanya yang terkoneksi oleh internet, sehingga dapat dilihat oleh siapapun yang terkoneksi dengan jaringan internet.

Statis vs Dinamis

- **Statis:**
  - Halaman tidak/jarang berubah (statis)
  - Perubahan suatu halaman dilakukan secara manual dengan mengedit codingan website tersebut
  - Minim interaksi dari pengunjung atau admin
  - Contoh: *company profile, landing page* (dengan catatan hanya menampilkan informasi saja tanpa fitur lain)
- **Dinamis:**
  - Sering update (dinamis)
  - Ada halaman *backend* untuk melakukan perubahan konten dari website tersebut
  - Menyediakan fitur-fitur untuk interaksi dengan pengguna
  - Contoh: toko online, web berita, forum, web media sosial, dll.

Statis vs Dinamis

Website Statis	Website Dinamis
Konten jarang berubah	Konten selalu update
Cukup dibuat menggunakan HTML saja atau ditambah CSS agar lebih bagus	Biasanya dibangun menggunakan bahasa pemrograman tertertu seperti: PHP, Javascript, Ruby, dan lainnya. Jika ingin cepat bisa menggunakan framework (laravel, codeigninter, nodejs) atau CMS (Wordpress, Magento, dan Pretashop)
User biasanya hanya melihat-lihat saja	User bisa melakukan sesuatu seperti berkomentar, mendaftar, tanya jawab, dan lainnya
Tidak menggunakan database	Menggunakan database
Loading lebih cepat karena halaman biasanya hanya sedikit, tidak ada database, dan fitur-fitur yang memberatkan server.	Loading bisa lebih lambat dari web statis, karena meload banyak halaman dan fitur

Pemrograman Web

- **Pemrograman web** adalah pembuatan aplikasi/program menggunakan Bahasa pemrograman yang dapat diakses/dijalankan menggunakan web browser.
- **Bahasa:**
  1. ***Client side:*** HTML, CSS, JavaScript, XML.
  2. ***Server side:*** PHP, ASP, ASP.Net, Java, Ruby, Perl, Python dll

Terminologi Penting

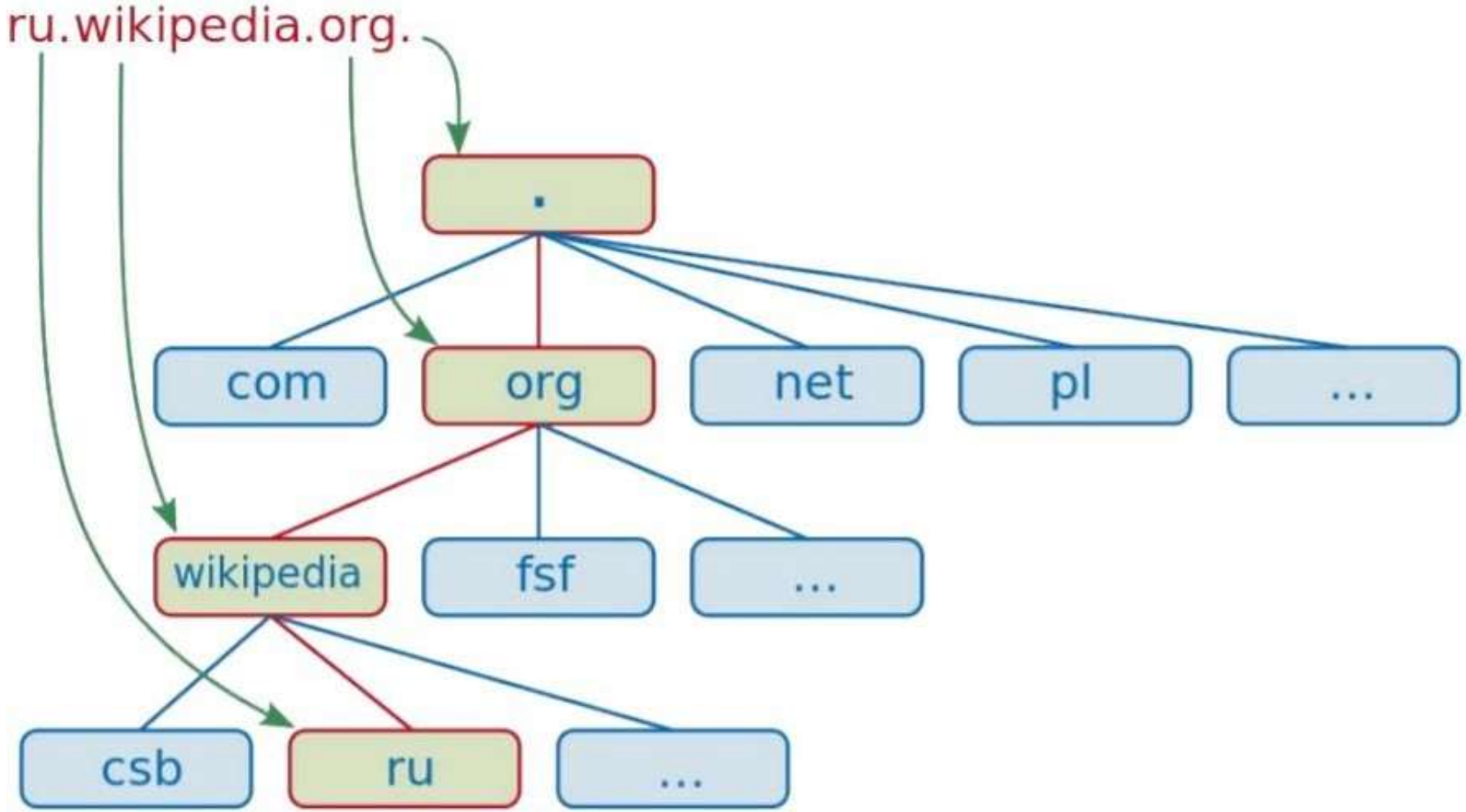
1. Internet
2. World Wide Web (WWW)
3. Web Server
4. URL (Universal Resource Locator)
5. HTTP (Hypertext Transfer Protocol)
6. DNS (Domain Name System)
7. IP (Internet Protocol)

Why Domain Name

1. **User friendly**
   › No need to remember IP address
2. **Load balancing**
   › Same name maps to changing IP address
3. **Decoupling**
   › Can move server to different network, ISP, etc

194.31.53.144/openlearningftmm/

Struktur Hirarki



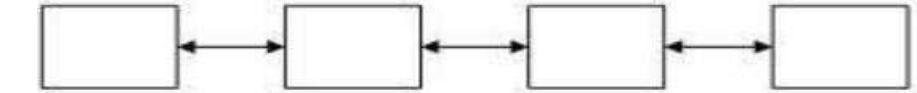
Struktur Navigasi Website

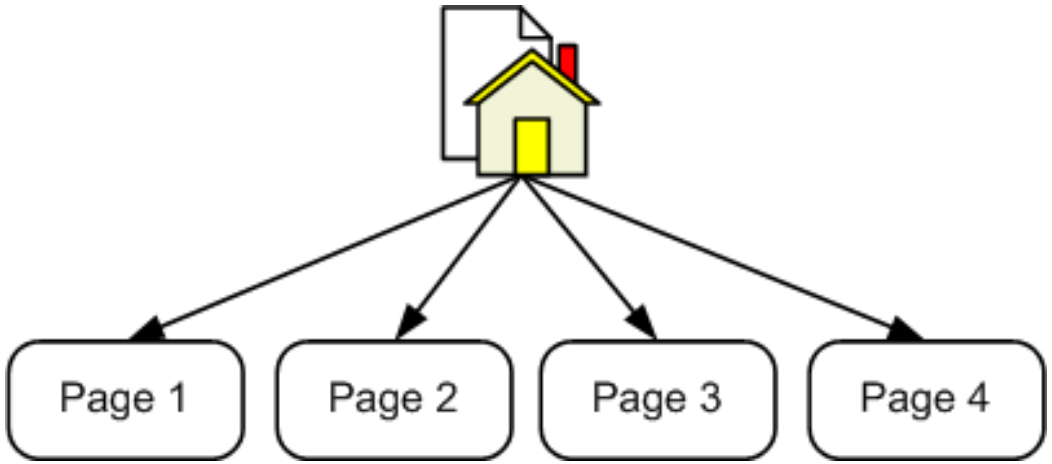
- **Struktur navigasi:** bagan hirarki dari suatu website yang menggambarkan isi dari setiap halaman dan link.
- **Struktur navigasi:** gambaran dari hubungan atau rantai kerja dari seluruh elemen yang akan digunakan dalam aplikasi.
- **Kriteria pengelompokan struktur navigasi:**
  - Kebutuhan
  - Kemudahan pemakaian
  - Kemudahan pembuatan
  - Interaktif

Struktur Navigasi Website

4 Struktur dasar:

- Linear: berurut sesuai urutan

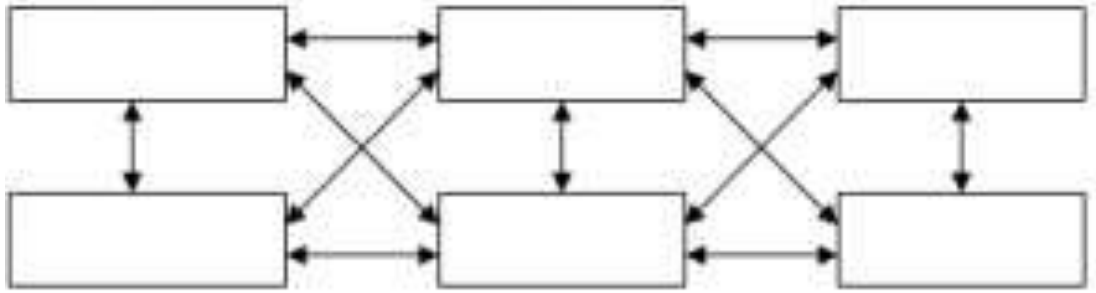




Struktur Navigasi Website

4 Struktur dasar:

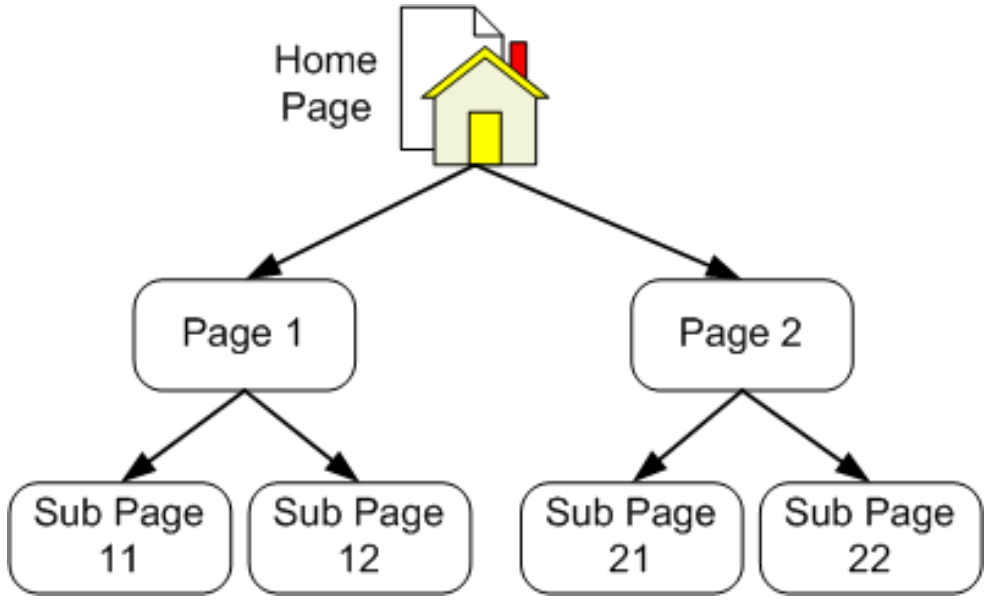
- Non-linear: pengembangan dari struktur navigasi linier dan bisa membuat navigasi bercabang. Pencabangan struktur non linier berbeda dengan struktur hirarki, karena tiap tampilan mempunyai kedudukan yang sama yaitu tidak ada parent atau child.



Struktur Navigasi Website

4 Struktur dasar:

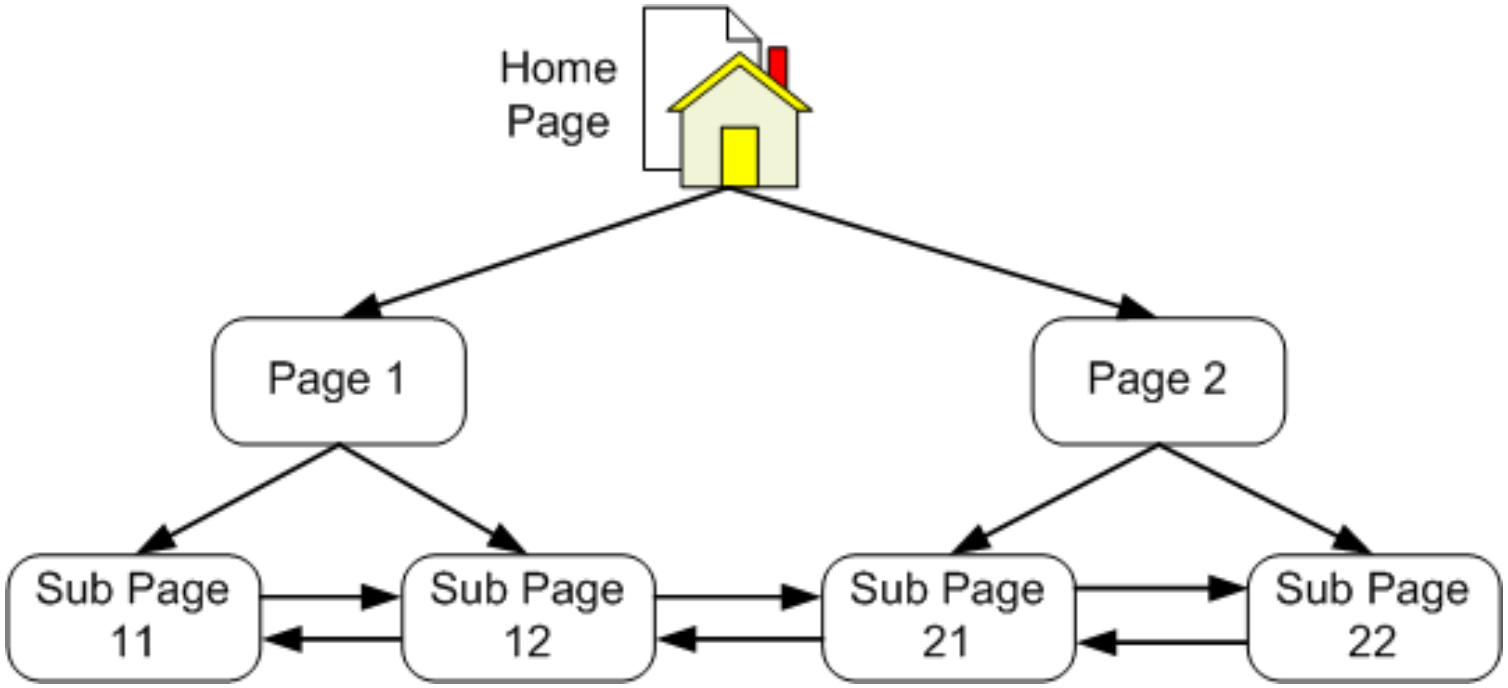
- Hirarki: hubungan antara satu kategori peringkat tinggi dengan subkategori bermakna pengkhususan kategori peringkat tinggi. Pergerakan ke atas hierarki bermakna perluasan kategori tersebut, sementara pergerakan ke bawah hierarki bermakna pengkhususan kategori tersebut



Struktur Navigasi Website

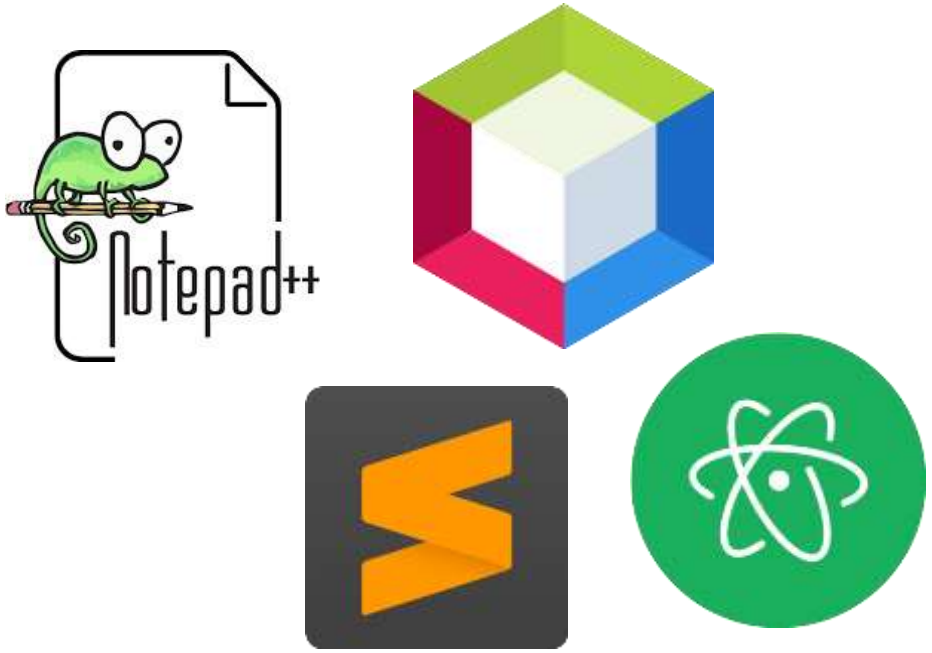
4 Struktur dasar:

- Campuran: bebas disesuaikan dengan kebutuhan customer.



Text Editor

- Dalam membuat sebuah halaman web dibutuhkan **text editor**.
- **Contoh:**
  - Notepad
  - Notepad++
  - Sublime
  - Atom
  - Netbeans
  - Dll.



Pengenalan HTML

Hypertext Markup Language (HTML)

- **HTML adalah bahasa pemrograman yang digunakan untuk menampilkan sebuah halaman website.**

- File HTML dapat diedit oleh editor teks apapun.
- Disimpan dengan extension **.html atau .htm**
- Kode-kode dalam HTML biasanya disebut **TAG**.
- Tag dituliskan diapit oleh tanda lebih kecil (<), tanda lebih besar (>), dan garis miring (/).

Struktur Dasar HTML

- Default dimulai dengan tag **<HTML>** dan diakhiri dengan **</HTML>.**
- Tag **<HEAD> …</HEAD>** adalah tag kepala dimana akan terlebih dulu dieksekusi sebelum tag badan.
- Di dalam tag HEAD berisi tag **<META>** dan **<TITLE>.**
- Tag **<META>** merupakan informasi atau header suatu dokumen HTML.
- Tag **<TITLE> …</TITLE>** adalah tag judul dan muncul pada titlebar browser.
- Tag **<BODY> …</BODY>** adalah tag berisi content dari suatu halaman web.

```
<html>
<head>
        <title> Judul Web </title>
</head>
<body>


</body>
</html>
```

HTML Element

- The HTML element is everything from the start tag to the end tag:

```
<tagname>Content goes here...</tagname>
```

- Contoh:

```
<h1>My First Heading</h1>

<p>My first paragraph.</p>
```

Start tag	Element content	End tag
<h1>	My First Heading	</h1>
	My first paragraph.	
 	none	*none*

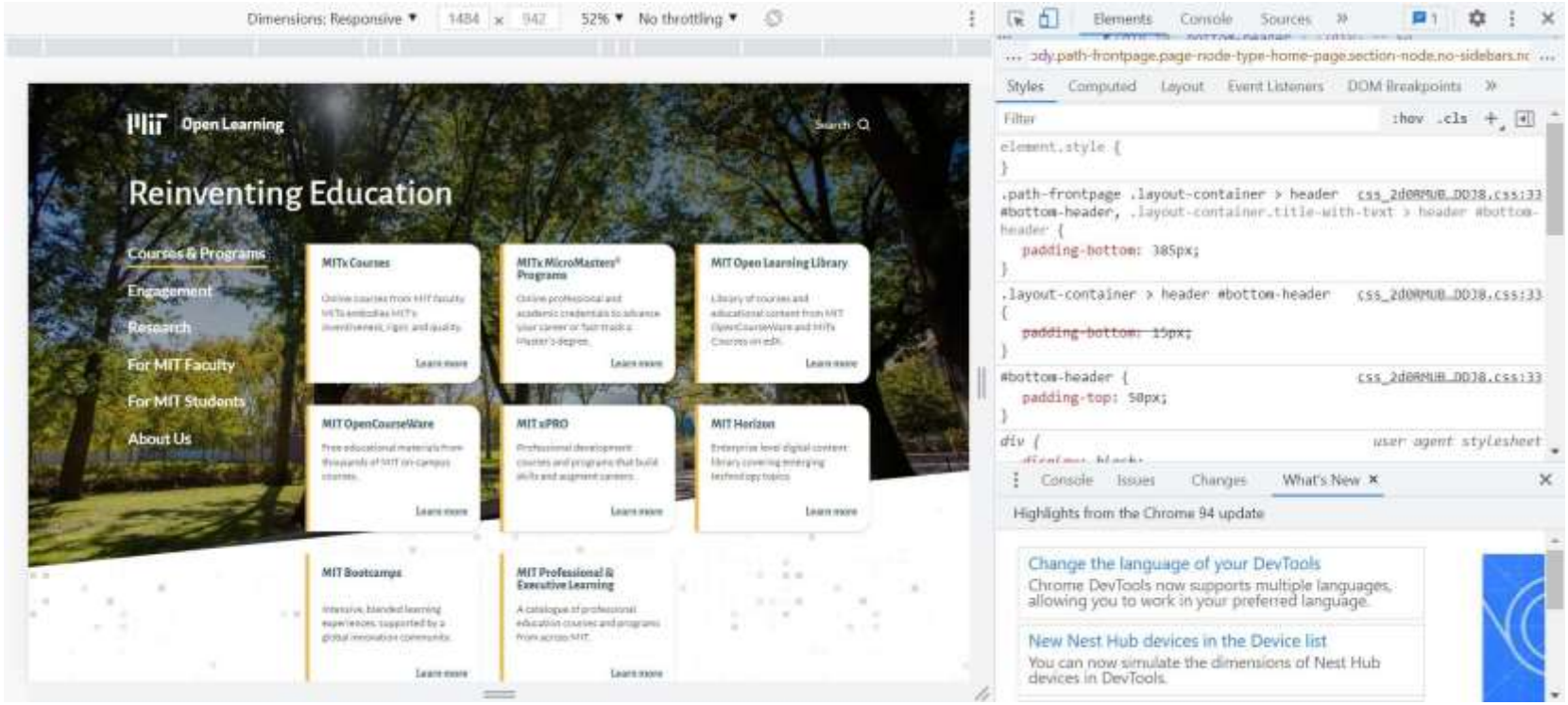
Scripting

```
<!DOCTYPE html>
<html>
<body>

<h1>My First Heading</h1>

<p>My first paragraph.</p>

</body>
</html>
```

1. Buka text editor
2. Tuliskan kode disamping
3. Simpan file dengan ekstensi .htm atau .html
4. Jalankan file dengan double click
5. File akan tampil di web browser

View HTML Source

- Right-click in an HTML page and select **"View Page Source“ atau**
- Right-click on an element (or a blank area) and choose **"Inspect"** or **"Inspect Element"** to see HTML elements (and CSS).



Tag Dasar

No	Nama Tag	Fungsi
1	<! DOCTYPE html>	Deklarasi untuk mendefinisikan dokumen menjadi HTML
2	<html>	Tag pembuka untuk membuat dokumen HTML
3	<head>	Informasi meta tentang dokumen
4	<title>	Membuat judul halaman yang nantinya akan ditampilkan di browser
5	<body>	Tempat dibuatnya semua konten website menggunakan HTML

Tag Dasar

Nama Tag	Fungsi
<h1> to <h6>	Tag untuk membuat heading
<p>	Tag untuk membuat paragraf
 	Memasukan satu baris putus
<hr>	Tag untuk membuat perubahan dasar kata didalam isi
<!--...-->	Tag untuk membuat komentar
<a>	Tag untuk membuat hyperlink
<link>	Tag untuk membuat hubungan antara dokumen dan sumber daya eksternal

Tag Dasar: Formatting

No	Nama Tag	Fungsi
1	<small>	Tag untuk membuat teks kecil
2	<strike>	Tag untuk membuat teks yang di coret tengah (tidak disupport lagi di HTML5)
3		Tag untuk membuat teks penting
4	<b>	Tag untuk membuat huruf bercetak tebal
5	<abbr>	Tag untuk membuat sebuah singkatan

Tag Dasar: Form

Nama Tag	Fungsi
<form>	Tag untuk membuat sebuah form HTML untuk input pengguna
<input>	Tag untuk membuat sebuah kontrol input
<textarea>	Tag untuk membuat sebuah kontrol input multibaris (text area)
<button>	Tag untuk membuat sebuah tombol yang dapat diklik
<select>	Tag untuk membuat sebuah daftar drop-down
<option>	Tag untuk membuat pilihan dalam daftar drop-down
<label>	Tag untuk membuat sebuah label untuk sebuah elemen <input>

Tag Dasar: Images

Nama Tag	Fungsi
<img>	Tag untuk membuat gambar
<map>	Tag untuk membuat gambar-peta
<area>	Tag untuk membuat area dalam gambar-peta
<canvas>	Digunakan untuk menggambar grafik, melalui scripting (JavaScript) (tag baru HTML5)
<figcaption>	Tag untuk membuat sebuah caption untuk elemen <figure> (tag baru HTML5)
<figure>	Menentukan konten mandiri (tag baru HTML5)

Tag Dasar: List

Nama Tag	Fungsi
	Tag untuk membuat daftar dengan selain nomor
	Tag untuk membuat daftar dengan nomor
	Tag untuk membuat sebuah item daftar
<dl>	Tag untuk membuat sebuah daftar definisi
<dt>	Tag untuk membuat istilah (item) dalam daftar definisi
<menu>	Tag untuk membuat deskripsi dari item dalam daftar definisi

Tag Dasar: Tables

Nama Tag	Fungsi
<table>	Tag untuk membuat tabel
<caption>	Tag untuk membuat sebuah caption tabel
<th>	Tag untuk membuat sebuah sel header tabel
<tr>	Tag untuk membuat baris dalam sebuah tabel
<td>	Tag untuk membuat sel dalam sebuah tabel
<thead>	Mengelompokan isi header dalam sebuah tabel
<tbody>	Mengelompokanisi tubuh dalam sebuah tabel
<col>	Menentukan properti kolom untuk setiap kolom dalam elemen <colgroup>
<colgroup>	Menentukan kelompok dari satu atau lebih kolom dalam sebuah tabel untuk diformat

Pengenalan CSS

Cascading Style Sheets (CSS)

- **CSS:** bahasa yang digunakan untuk mengatur tampilan suatu dokumen yang ditulis dalam bahasa markup/markup language.
- Dalam konteks web, CSS adalah bahasa yang digunakan untuk mengatur **tampilan/desain sebuah halaman HTML**.

Cara Penggunaan CSS

- **External Style Sheet:** CSS disimpan pada suatu file sehingga terpisah dari dokumen HTML. Cara menggunakan tinggal melakukan pemanggilan file CSS dalam dokumen HTML.

```
<html>
<head>
        <title>CSS Secara Internal</title>
        <link rel=”stylesheet” type=text/css” href=”style.css” />
</head>
```

- **Internal Style Sheet:** CSS ditulis pada bagian HEAD dokumen HTML menggunakan tag <style>

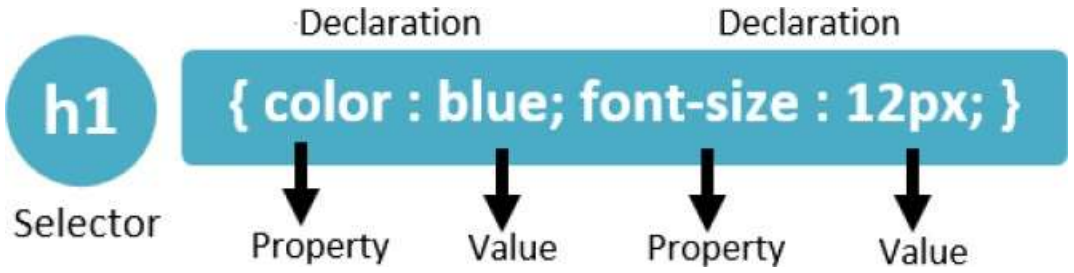
```
<html>
<head>
        <title>CSS Secara Internal</title>
        <style type="text/css">P{text-align:justify;}
</style>
</head>
```

Cara Penggunaan CSS

- **Inline Style Sheet:** CSS ditulis langsung pada tag HTML yang akan diatur tampilannya menggunakan atribut style

```
<html>
<head>
        <title>CSS Secara Internal</title>
        <style type="text/css">P{text-align:justify;}
</style>
</head>
<body>
<p style =”text-align:justify;”> Paragraph yang diatur
CSS Secara Internal</p>
</body>
</html>
```

Penulisan CSS



Struktur CSS:

1. **Selector:** biasanya berupa tag HTML, id, class
2. **Declaration:** berisi aturan-aturan css yang terdiri dari properti dan nilainya yang dipisahkan oleh tanda titik dua. Setiap aturan css harus diakhiri dengan tanda titik koma.

Penulisan CSS: Selector

- **Biasanya berupa tag HTML, id, class**
  - id menggunakan tanda # didepan nama selector
  - class menggunakan tanda titik didepan nama selector
- **Contoh:**

```
h1 { color : blue ; } ➔ tag html h1
#teks { color :green; } ➔ id
.warna { color : red; } ➔ class
```

Penulisan CSS: Declaration

- **Selector ID pada CSS**

```
<body>
   <p id=”teks”>TEST
</p>
</body
```

```
#teks
{
Color : blue;
Font-family: Calibri;
}
```

- **Aturan ID:**
  - Sebuah elemen HTML hanya boleh memiliki 1 id
  - Setiap halaman hanya boleh memiliki 1 elemen dengan id tersebut
  - Sebaiknya tidak digunakan untuk css (lebih baik gunakan class)

Penulisan CSS: Declaration

- **Selector Class pada CSS**

```
<body class=”warna”>

</body>
```

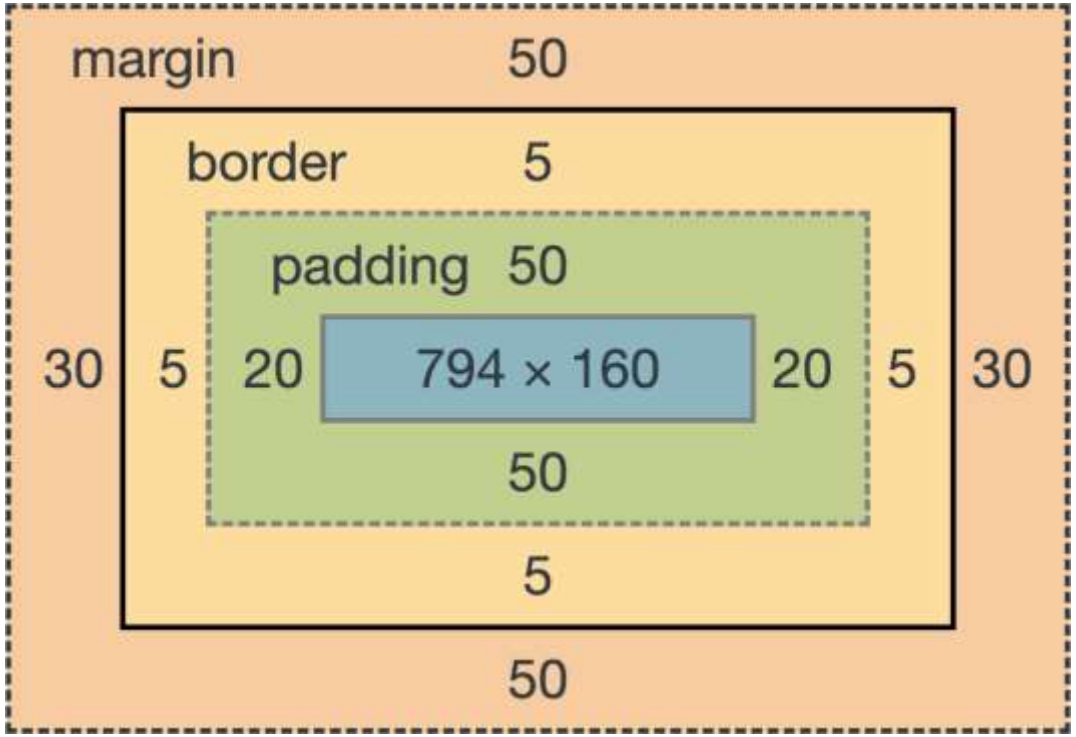
```
.warna
{ background-color: lightgreen;  }
```

Properti CSS

Properti	Fungsi	Nilai	Contoh penulisan :
Color	Mengatur warna pada Teks	Nama warna, kode hexa warna	Color : blue; Color:#ffffff;
Background-color	Mengatur warna latar	Nama warna, kode hexa warna,rgb	Background-Color:rgb(200,0,55);
Background-image	Mengatur gambar latar	Nama file gambar	Background-image:url(header.jpeg);
Text-align	Mengatur perataan teks	Left,Center, right, justify	Text-align:left;
Font-family	mengatur jenis font	Nama-nama jenis huruf, co: arial, times new roman, georgia	Font-family:calibri;
Dll.			

Padding, Margin, dan Border

- Dalam CSS dikenal istilah **'Box Model'.**
- Berguna untuk **tata letak elemen**



Responsive Web

- Teknik atau metode bagi web designer untuk membuat suatu layout website yang dapat menyesuaikan diri sesuai dengan ukuran layar pengguna.
- Baik dari ***ukuran huruf, user interface, gambar*** dan ***tata letak*** akan menyesuaikan dengan lebar layar dan resolusi device yang digunakan.
- **Popular CSS framework:** Bootstrap, Materialize CSS, Foundation, Bulma, Tailwind CSS, UIkit, Milligram, Pure.CSS, etc.
- Keuntungan:
  - Kenyamanan
  - Dapat diakses oleh berbagai device
  - Hemat biaya
  - Dll.

Contoh Responsive Web

- Media queries
- Fluid grids
- Flexible visuals
- Dribbble
- GitHub
- Klientboost
- Magic Leap
- Dll.

Trend website design 2021: https://99designs.com/blog/trends/web-design-trends/

Tips Interface Design bagus: https://www.webstyleguide.com/7-interface-design.html

Pengenalan JavaScript

Programming the browser

- **Java**
  - › 1990: project at Sun to replace C++
  - › 1994: “Oak”retargeted to the web for “applets”
  - › Java takes off, first safe language in widespread use
- **Javascript**
  - › 1995: “Mocha” project at Netscape
  - › Javascript takes off, included in Microsoft’s IE
  - › 1996: submitted to Ecma as standard
- **today**
  - › Java alive and well server-side
  - › but JSdominates client-side
  - › making inroads server-side too (eg, node.js)

Syntax

- **statements like Java**
  - while, for, if, switch, try/catch, return, break, throw
- **comments**
  - **use //, avoid /**/**
- **semicolons**
  - inserted if omitted
- **declarations**
  - function scoping with var

```
var MAX = 10;
var line = function (i, x) {
    var l = i + " times " + x
          + " is " + (i * x);
    return l;
}
var table = function (x) {
    for (var i = 1; i <= MAX; i += 1) {
        console.log(line(i, x));
    }
}
// display times table for 3
table(3);
```

```
1 times 3 is 3
2 times 3 is 6
3 times 3 is 9
4 times 3 is 12
5 times 3 is 15
6 times 3 is 18
7 times 3 is 21
8 times 3 is 24
9 times 3 is 27
10 times 3 is 30
undefined
>
```

Basic Types

- **primitive types**
  - › strings, numbers, booleans
  - › operators auto convert
- **arrays**
  - › can grow, and have holes
- **funny values**
  - › undefined: lookup non-existent thing
  - › null: special return value
- **equality**
  - › use ===, !==

```
> 1 + 2
3
> 1 + '2'
"12"
> 1 * 2
2
> 1 * '2'
2
```

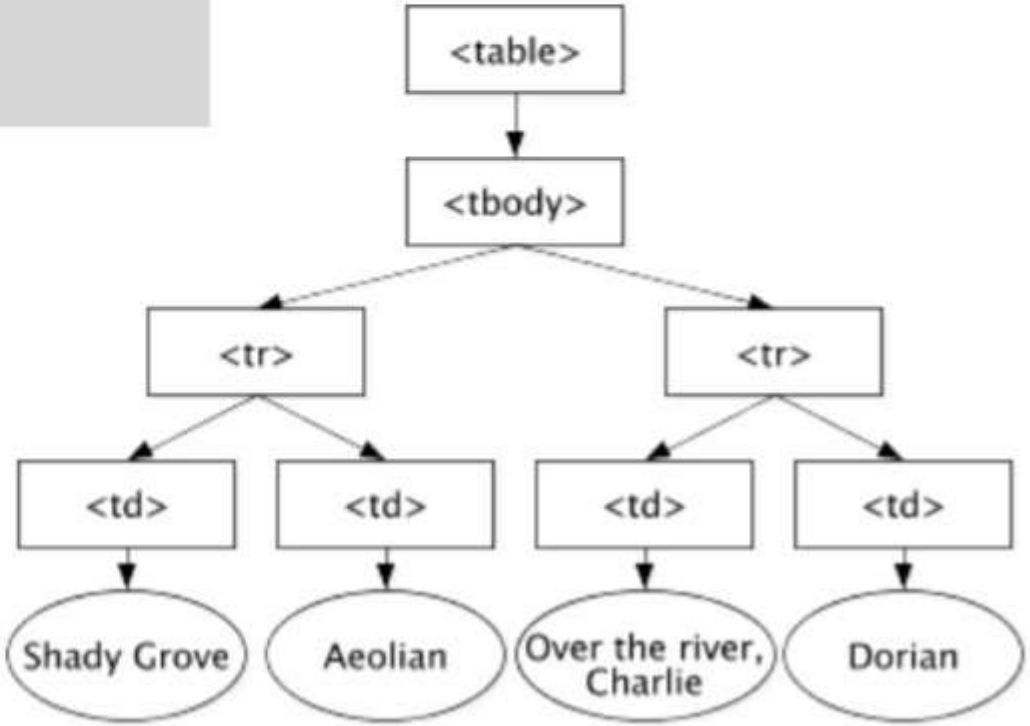
```
> a = []
[]
> a[2] = 'hello'
"hello"
> a.length
3
> a[1]
undefined
> a[2]
"hello"
> a[3]
undefined
```

```
> 1 === 1
true
> 1.0 === 1
true
>'hello' === 'hello'
true
> [] === []
false
```

```
<table>
  <tbody>
    <tr>
      <td>Shady Grove</td>
      <td>Aeolian</td>
    </tr>
    <tr>
      <td>Over the River, Charlie</td>
      <td>Dorian</td>
    </tr>
  </tbody>
</table>
```

Document Object Model (DOM)

DOM: *inteíface* yang memungkinkan *developeí* untuk memanipulasi konten, stíuktuí, dan *style* situs web.



Client Side Programming

- **where's the code?**
  - › JavaScript
- **manipulating the DOM**
  - › traverse the elements
  - › add and delete elements
  - › modify properties (eg position)
  - › modify styles (eg color)
- **also**
  - › registering events (against timers, buttons)
  - › making calls to webservers

Modifying Styles

```
<!DOCTYPE html>
<head>
    <style>
    .plain {color: black}
    .highlight {color: red}
    </style>
</head>
<body>
    <h1 id="header" class="">
        My new webpage
    </h1>
    <div id="content" class="plain">Welcome!
    </div>
</body>
</html>
```

page

before

after

```
function highlight_content() {
    var e = document.getElementById('content');
    e.className = 'highlight';
}
highlight_content();
```

code executed in console

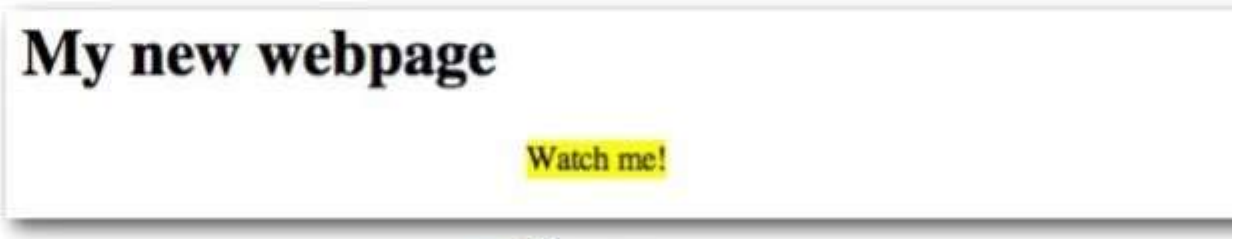
Animating Element

```
<!DOCTYPE html>
<head>
    <style>
        #object {
        position:absolute;
        background:yellow;
        left:0px;
        }
    </style>
</head>

<body>
    <h1 id="header" class="">
        My new webpage
    </h1>
    <div id="object">Watch me!
    </div>
</body>
</html>
```

page

```
var obj = document.getElementById('object');
var i = 0;
function animate() {
    obj.style.left = i + 'px';
    i = i + 10;
    setTimeout(animate,20);
    }
animate();
```

code executed in console



effect

Inserting Element

```html
<!DOCTYPE html>
<head>
    <style>
        #content {
            font-size: 20px;
        }
        .popup {
        position:relative;
        left: 10px;
        width: 100px;
        top: -40px;
        opacity: 0.5;
        background:yellow;
        }
    </style>
</head>
<body>
    <h1 id="header" class="">
        My new webpage
    </h1>
    <div id="content">Content
    </div>
</body>
</html>
```

page

```
function popup(elt) {
    var box = document.createElement('div');
    box.innerHTML = "comment";
    box.className = 'popup';
    elt.appendChild(box);
}
var elt = document.getElementById('content');
popup(elt);
```

code executed in console

before

after

Latihan

- Buatlah sebuah halaman web berbasis HTML CSS dan JS dengan fitur sebagai berikut:
  - Halaman wajib live di internet (pakai alwaysdata / lainnya)
  - 1 buah text field
  - 1 buah button dengan aksi:
    - Mengambil URL yang diinput dalam text field
    - Mengambil data yang diberikan URL tersebut
    - Ubah dan tampilkan data tersebut dalam tabel berbasis HTML dengan lokasi bawah button tersebut

    - Clue:
      - gunakan AJAX/Axios
      - URL yang dipakai https://testingalpro.alwaysdata.net/api/getcoffee.php

Terima Kasih

Referensi:

https://www.w3schools.com/html/
https://www.w3schools.com/css/
https://www.webstyleguide.com/7-interface-design.html
Web programming karya Ani Oktarini Sari et.al.
https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-170-software-studio-spring-2013/lecture-notes/