



# Plan

- 1 Introduction
- 2 Solutions existantes
- 3 Architecture
- 4 Technologies
- 5 Réalisation
- 6 Discussion
- 7 Conclusion

h e p i a

Haute école du paysage, d'ingénierie  
et d'architecture de Genève

# Problématique

- Nombre de fichiers énorme.

# Problématique

- Nombre de fichiers énorme.
- Difficulté à retrouver des fichiers.

# Problématique

- Nombre de fichiers énorme.
- Difficulté à retrouver des fichiers.
- Plusieurs emplacements logiques pour un seul fichier.

# Problématique

- Nombre de fichiers énorme.
- Difficulté à retrouver des fichiers.
- Plusieurs emplacements logiques pour un seul fichier.

## Systeme de tags de fichiers et répertoires avec possibilité de recherche par tags.

# Cahier des charges

- Étudier et s'approprier le langage Rust.

# Cahier des charges

- Étudier et s'approprier le langage Rust.
- Répertorier les applications existantes permettant d'étiqueter les fichiers.



# Cahier des charges

- Étudier et s'approprier le langage Rust.
- Répertorier les applications existantes permettant d'étiqueter les fichiers.
- Étudier les attributs étendus (XATTR) lors des manipulation courantes sur les fichiers.

# Cahier des charges

- Étudier et s'approprier le langage Rust.
- Répertorier les applications existantes permettant d'étiqueter les fichiers.
- Étudier les attributs étendus (XATTR) lors des manipulation courantes sur les fichiers.
- Analyser les moyens d'indexer une arborescence de fichiers.

# Cahier des charges

- Étudier et s'approprier le langage Rust.
- Répertorier les applications existantes permettant d'étiqueter les fichiers.
- Étudier les attributs étendus (XATTR) lors des manipulation courantes sur les fichiers.
- Analyser les moyens d'indexer une arborescence de fichiers.
- Explorer les méthodes de surveillance du système de fichiers.

# Cahier des charges

- Étudier et s'approprier le langage Rust.
- Répertorier les applications existantes permettant d'étiqueter les fichiers.
- Étudier les attributs étendus (XATTR) lors des manipulation courantes sur les fichiers.
- Analyser les moyens d'indexer une arborescence de fichiers.
- Explorer les méthodes de surveillance du système de fichiers.
- Concevoir et implémenter un système performant.

# Cahier des charges

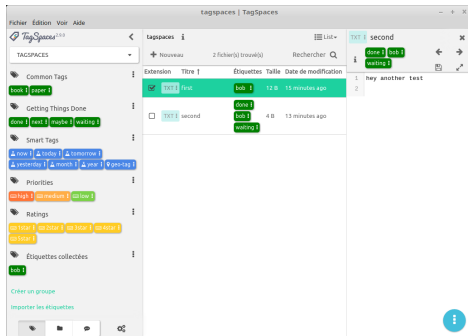
- Étudier et s'approprier le langage Rust.
- Répertorier les applications existantes permettant d'étiqueter les fichiers.
- Étudier les attributs étendus (XATTR) lors des manipulation courantes sur les fichiers.
- Analyser les moyens d'indexer une arborescence de fichiers.
- Explorer les méthodes de surveillance du système de fichiers.
- Concevoir et implémenter un système performant.
- Mesurer les performances de ce système.

# TMSU, Tagsistant et TagSpaces

- Gestion des tags.
- Liste des fichiers liés à des tags.
- Usage de commandes usuelles pour manipuler les tags (Tagsistant).

# TMSU, Tagsistant et TagSpaces

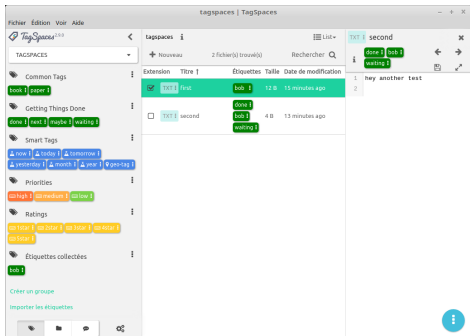
- Gestion des tags.
- Liste des fichiers liés à des tags.
- Usage de commandes usuelles pour manipuler les tags (Tagsistant).



# TMSU, Tagsistant et TagSpaces

- Gestion des tags.
- Liste des fichiers liés à des tags.
- Usage de commandes usuelles pour manipuler les tags (Tagsistant).

Points positifs	Points négatifs
Simple (CLI)	Dépendance à FUSE
Rapides et efficaces	Dépendance à une BDD externe
Open source	Modification et accès uniquement par l'app (Tagsistant et TagSpaces)
	Non écrits en Rust





# macOS

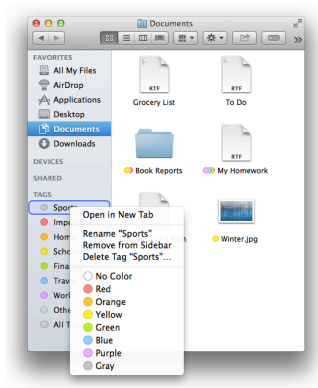
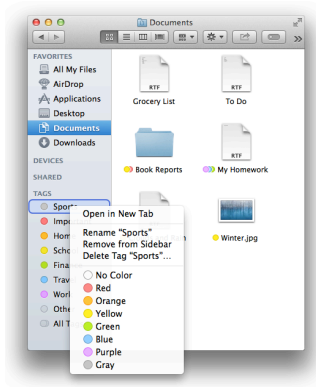


Figure – Gestion d'un tag dans le Finder [1]

h e p i a

Haute école du paysage, d'ingénierie  
et d'architecture de Genève

# macOS



Points positifs	Points négatifs
Système de tags intégré à l'explorateur de fichiers	Code propriétaire
Stocke les tags dans les attributs des fichiers	Seulement pour macOS
Performant	Non écrit en Rust

Figure – Gestion d'un tag dans le Finder [1]

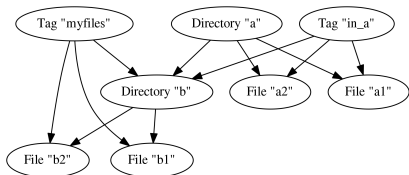
h e p i a

Haute école du paysage, d'ingénierie  
et d'architecture de Genève

# Gestion des tags

- Stockage des tags dans les attributs étendus (XATTR).
- Outil dédié plutôt que reprendre les commandes existantes.
- Confort d'utilisation.

# Indexation des fichiers et des tags



# Indexation des fichiers et des tags

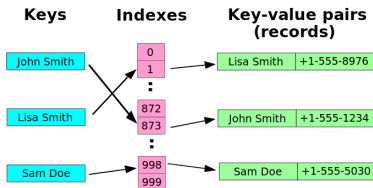
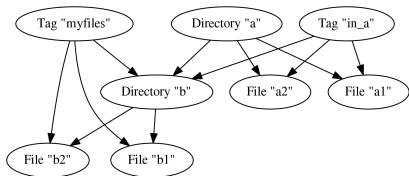


Figure – Un annuaire représenté comme une table de hachage - [2]

# Surveillance du système de fichiers

- Mise à jour du graphe lors des événements suivants :
  - Changement sur les tags.
  - Création de fichiers/répertoires.
  - Suppression de fichiers/répertoires.
  - Déplacement/renommage de fichiers/répertoires.
- Pattern producer-consumer.

## Requêtes de tags et fichiers

- Lister les fichiers et répertoires associés à des tags (expressions logiques).
- Lister les tags existants.
- Renommer un tag.

# Rust

## Généralités (1)

- Langage moderne, performant, fiable, compilé, et fortement typé.
- Disponible sur Linux, Windows et macOS.
- Cargo : système de compilation et d'exécution et gestionnaire de paquets intégré à Rust.
- Structures, collections, énumérations et pattern matching.

```
1 enum Direction { North, South, East, West }
2 match direction {
3     Direction::North => println!("Go North"),
4     Direction::South => println!("Go South"),
5     _ => println!("Go East or West") //clause par défaut
6 }
```



# Rust

## Généralités (2)

- Tests.
- Gestion des erreurs.

```
1 enum Result<T, E> { Ok(T), Err(E), }  
2 match value {  
3     Ok(data) => println!("Value : {}", data),  
4     Err(error) => println!("Error : {}", error)  
5 }
```

# Rust

## Généralités (3)

- Unsafe Rust.
- Concurrency et Threads.

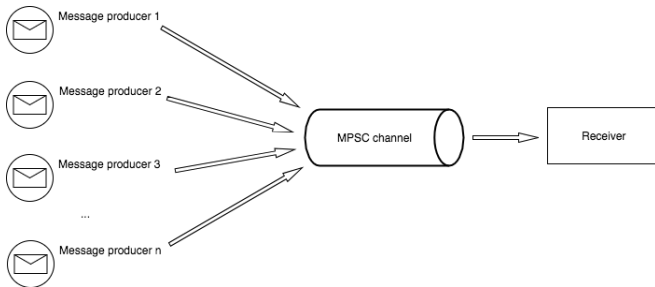


Figure – Canal de communication entre threads - [3]

# Rust

## Ownership et Borrowing (1)

### Ownership

- Chaque variable est dite le "possesseur" (*owner*) d'une valeur.
- Il ne peut y avoir qu'un seul *owner* pour une valeur.
- Lorsque l'*owner* est détruit ou change de portée, la valeur est détruite.

### Borrowing

- 1 À tout moment, il ne peut y avoir soit une seule référence mutable, soit plusieurs références immutables, mais pas les deux en même temps.
- 2 Les références doivent toujours être valides.

# Rust

## Ownership et Borrowing (2)

```
1 let a = 10;
2 let b = a;
3 let mut my_vec = vec![3, 2, 1];
4 let other_vec = my_vec;
5 my_vec.push(42); // Erreur, la valeur a été déplacée
```

```
1 fn main() {
2     let mut my_vec = vec![3, 2, 1];
3     ref_immutable(&my_vec);
4     ref_mutable(&mut my_vec);
5 }
6 fn ref_immutable(v : &Vec<i32>) { println!("{:?}", v); }
7 fn ref_mutable(v : &mut Vec<i32>) { v.push(42); }
```

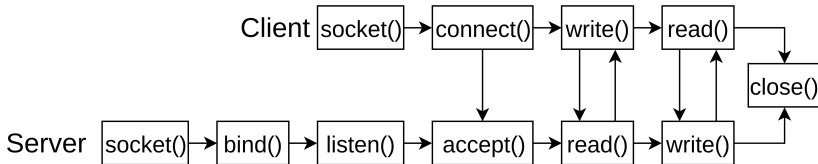
# Attributs étendus (XATTR)

- Métadonnées sous forme de paire `espace.nom:valeur`.
- Nom = chaîne de caractères, valeur = chaîne de caractères ou données binaires.
- Existent sous ext2-3-4, XFS, Btrfs, UFS1-2, NTFS, HFS+, ZFS.

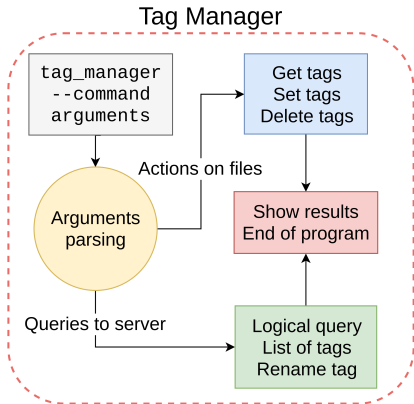
# Inotify

- API de notifications d'événements sur le système de fichiers.
- Trois appels système : initialisation, ajout de surveillance sur un chemin de fichiers donné et suppression de cette surveillance.
- Lecture d'un événement avec `read()`.

# Sockets

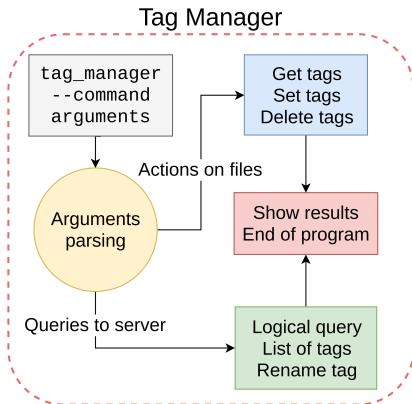


# Tag Manager



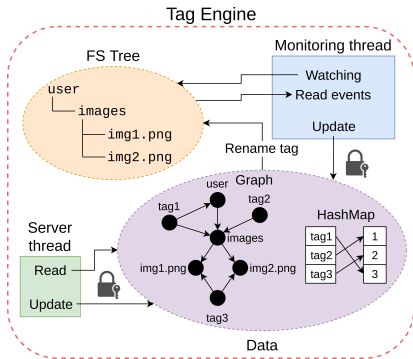


# Tag Manager

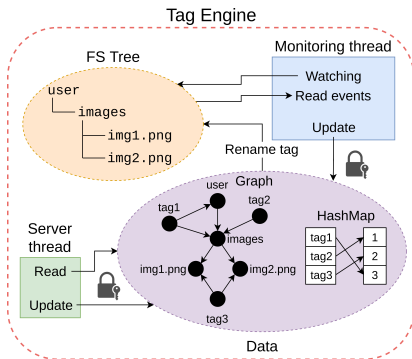


- CLI.
- Gestion des tags.
- Requêtes sur les tags et fichiers.
- Manipule les XATTR des fichiers.
- Programme "client".

# Tag Engine

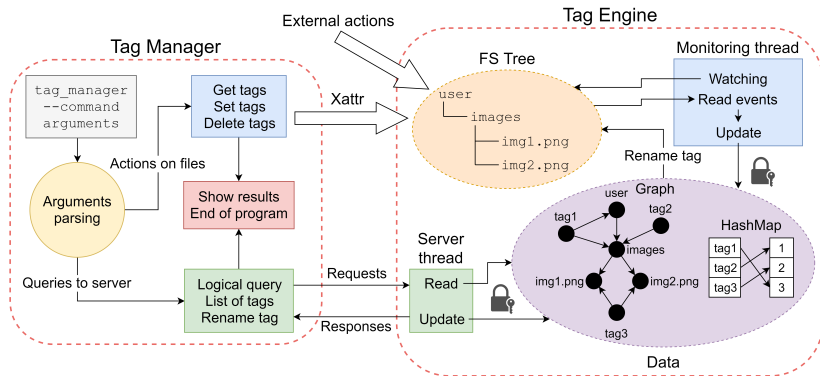


# Tag Engine

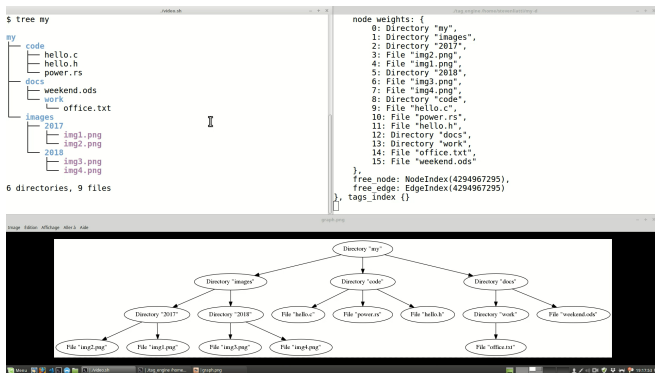


- Surveille l'arborescence des fichiers.
- Écoute sur une socket les requêtes provenant de Tag Manager.
- Maintient la relation entre tags, fichiers et répertoires à l'aide d'un graphe orienté et d'une hashmap.
- Les changements sur le FS sont répercutés sur le graphe.
- Multithread.

# TagFS



# Démo



## Vidéo

h e p i a

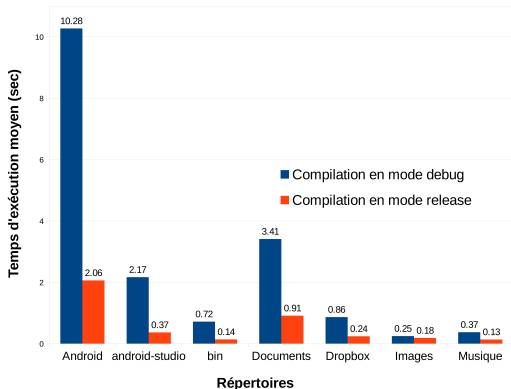
Haute école du paysage, d'ingénierie  
et d'architecture de Genève

# Mesures de performances

Répertoire	Sous-répertoires	Fichiers
Android	15'172	112'046
android-studio	3'331	13'287
bin	553	9'306
Documents	15'442	64'486
Dropbox	2'377	8'659
Images	5	863
Musique	135	1'352

# Mesures de performances

Répertoire	Sous-répertoires	Fichiers
Android	15'172	112'046
android-studio	3'331	13'287
bin	553	9'306
Documents	15'442	64'486
Dropbox	2'377	8'659
Images	5	863
Musique	135	1'352



# Rust vs C

Avantages par rapport à C	Inconvénients par rapport à C
Garanties sécurité mémoire	Courbe d'apprentissage plus longue
Détection des erreurs à la compilation	Contraintes du langage parfois handicapantes
Compilateur verbeux	Moins répandu
Performances égales ou très proches	Manque de soutien global
Gestion des erreurs	
Absence de NULL	
Cargo et Crates.io	
Librairie standard	
Généricité	



- Étude du langage Rust.
- Conception d'un moteur de gestion de tags.
- Diverses technologies et approches.

## Remerciements

- Florent Glück
- Orestis Malaspinas
- Joël Cavat

# Références I



Apple team.

Os x : Tags help you organize your files.

<https://support.apple.com/en-us/HT202754>, février 2015.

Consulté le 08.05.2018.



Wikipédia.

Un annuaire représenté comme une table de hachage.

[https://fr.wikipedia.org/wiki/Table\\_de\\_hachage#/media/File:HASHTB08.svg](https://fr.wikipedia.org/wiki/Table_de_hachage#/media/File:HASHTB08.svg), juin 2015.

Consulté le 23.06.2018.



Marcin Baraniecki.

Multithreading in rust with mpsc (multi-producer, single consumer) channels.

<https://bit.ly/2AbJELg>, novembre 2017.

Consulté le 18.06.2018.