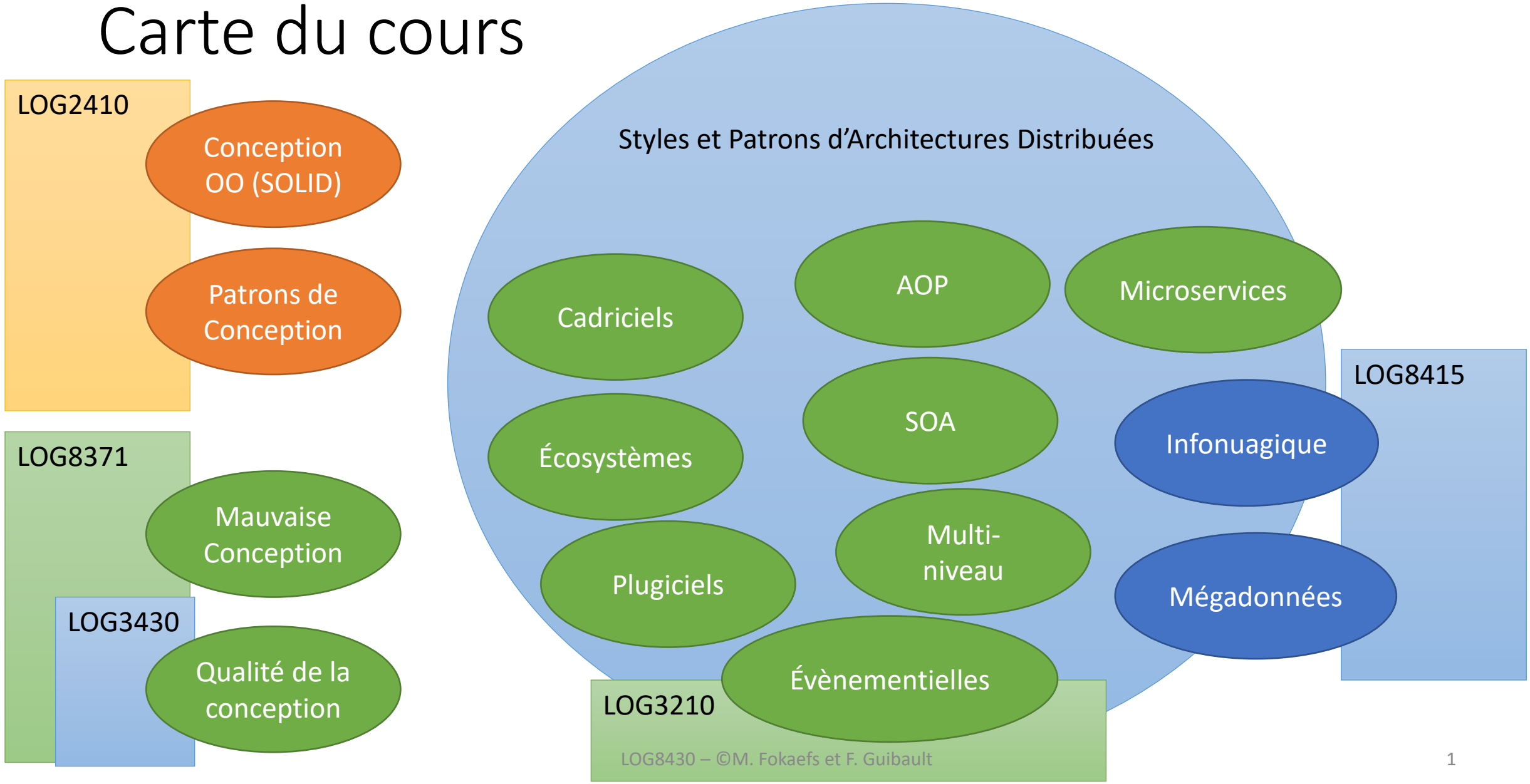
Carte du cours



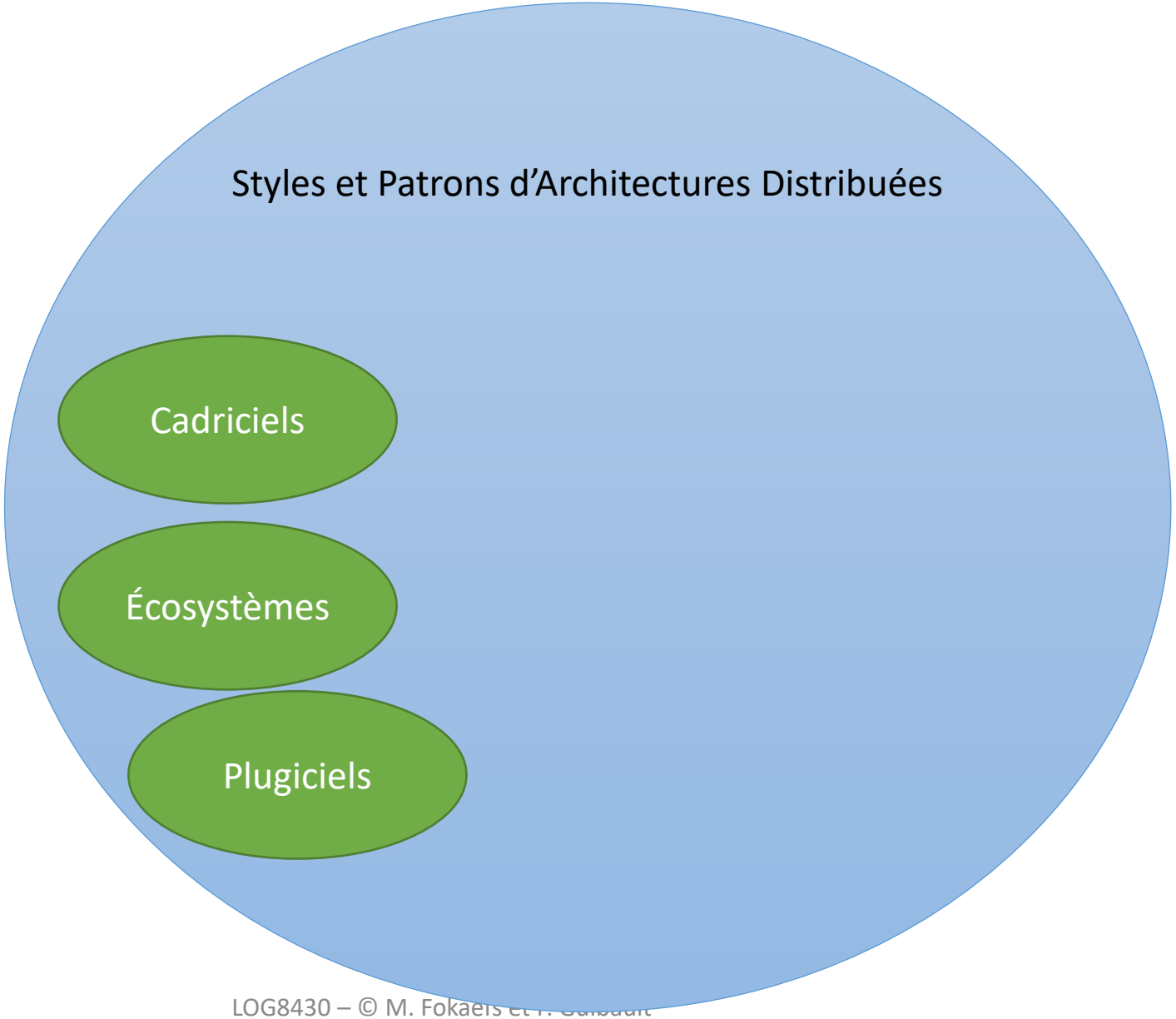
POLYTECHNIQUE MONTRÉAL
UNIVERSITÉ D'INGÉNIERIE

LOG8430: Cadriciels et Systèmes Distribuées

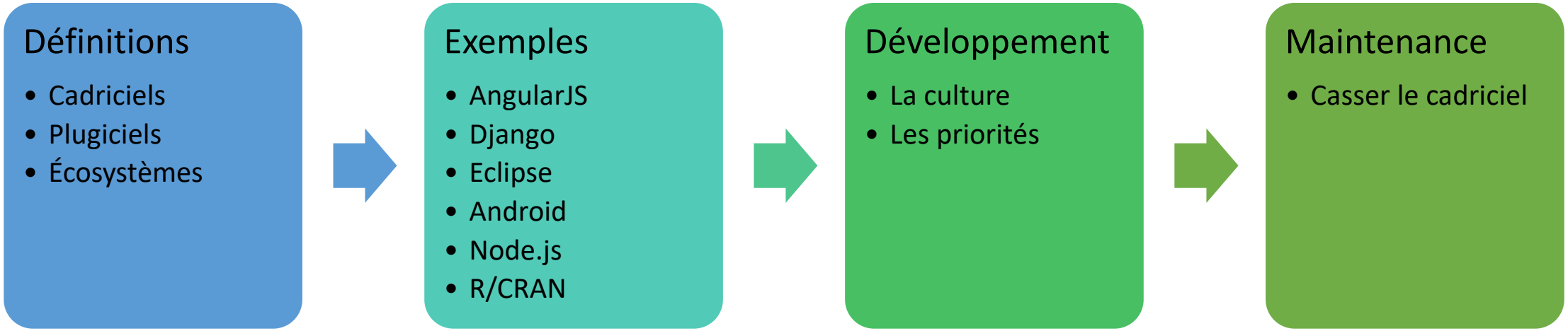
Aujourd'hui

LOG2410

- Conception OO (SOLID)
- Patrons de Conception



Les systèmes distribués…dans la nature!



Cadriciels : Définition

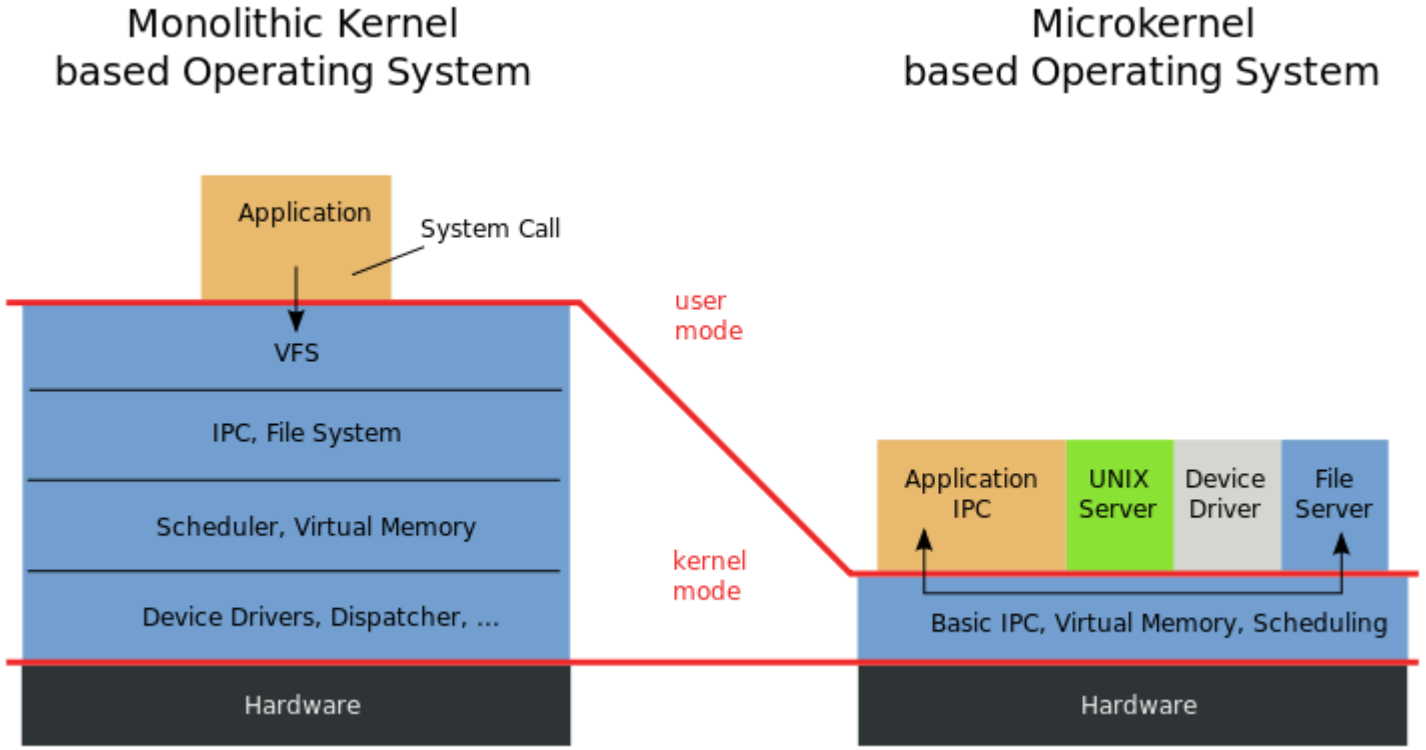
- Les cadriciels sont des architectures à moitié implémentées et génériques (agnostiques à l'application).
- Ils fournissent un squelette avec une fonctionnalité de base prévue pour être étendue.
- Dans un cadriciel, il y a des modules abstraits (*hot spots*), que les développeurs doivent étendre, et des modules concrets (*frozen spots*), qu'il ne faut pas changer.
  - Donc, les cadriciels implémentent le principe OCP par définition.
- Un bon exemple est le patron Template Method.
  - La méthode template est la partie concrète. Les méthodes abstraites appelées par la méthode template constituent la partie qui sera étendue.
  - Pour d'autres exemples d'utilisation de patrons dans les cadriciels, consultez l'article: « Meta Patterns—A Means For Capturing the Essentials of Reusable Object-Oriented Design » par Wolfgang Pree.

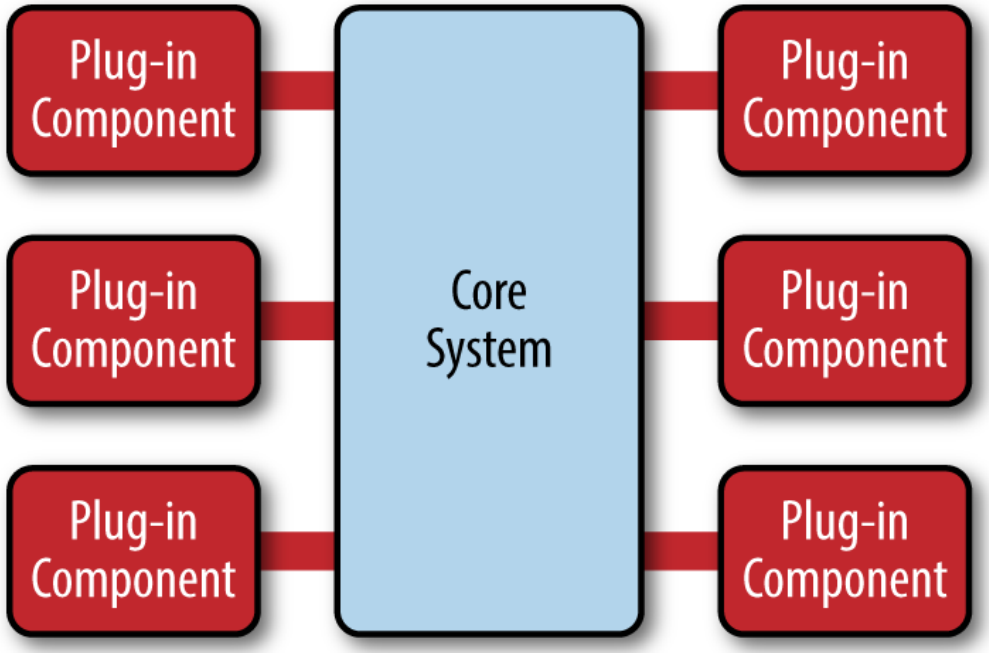
Cadriciels : Propriétés

- Inversion de contrôle : Les modules concrets du cadriciel dictent le flux de contrôle et non les modules étendus par le programme.
  - Connu aussi comme le principe Hollywood : « Ne nous appelez pas, nous vous appellerons »
  - Une implémentation du principe DIP.
  - Exemple 1 : ActionListeners. Le cadriciel gère les évènements de l'interface graphique et le code du développeur est notifié pour fournir la réaction à ces évènements.
  - Exemple 2 : Le patron Stratégie.
- L'extensibilité et la réutilisabilité sont augmentées, mais la modifiabilité est réduite.

Plugiciels : Définitions

- Les plugiciels suivent le type d'architecture « microkernel » (comme pour les systèmes d'exploitation).





Plugiciels : Propriétés

- Le système de base est atomique et indépendant. Ça veut dire que le système peut fonctionner sans plug-in.
- Les plug-ins ajoutent des fonctionnalités supplémentaires.
- Contrairement aux cadriciels, les plug-ins améliorent l'expérience de l'utilisateur plutôt que celle du développeur.
- Les plug-ins dépendent certainement du système de base, mais il est possible qu'ils dépendent aussi de d'autres plug-ins.
- SRP est le principe le plus pertinent pour les plugiciels.
  - La fonctionnalité est séparée et la logique est isolée.
- La flexibilité et l'extensibilité sont augmentées, mais la maintenabilité peut souffrir.
  - Les plug-ins dépendent fortement de la version du système de base (forme de vendor lock-in).

Les écosystèmes logiciels

- En fait, un écosystème logiciel est une collection de modules qui ont quelque chose en commun, p.ex. un langage, un cadriciel etc.
  - Il y a plusieurs définitions qui incluent aussi des aspects d'affaires et des relations économiques. Souvent, un écosystème est simplement un dépôt de modules logiciels partagé par une communauté de développeurs.
- La présence d'une plateforme, comme dans le cas des cadriciels ou des plugiciels, n'est pas nécessaire.
  - ATTENTION! Ça ne veut pas dire que les plugiciels ou les cadriciels n'ont pas d'écosystème, mais on peut classifier ces systèmes de façon plus concrète.
- Les modules, ou les « paquets », contribuent des fonctionnalités.
- Les développeurs soumettent des paquets dans un dépôt.
- La base des paquets est juste le langage ou un environnement d'exécution (comme JRE).
- Comme les cadriciels, mais contrairement aux plugiciels, les paquets facilitent le travail des développeurs.

Écosystèmes : Propriétés

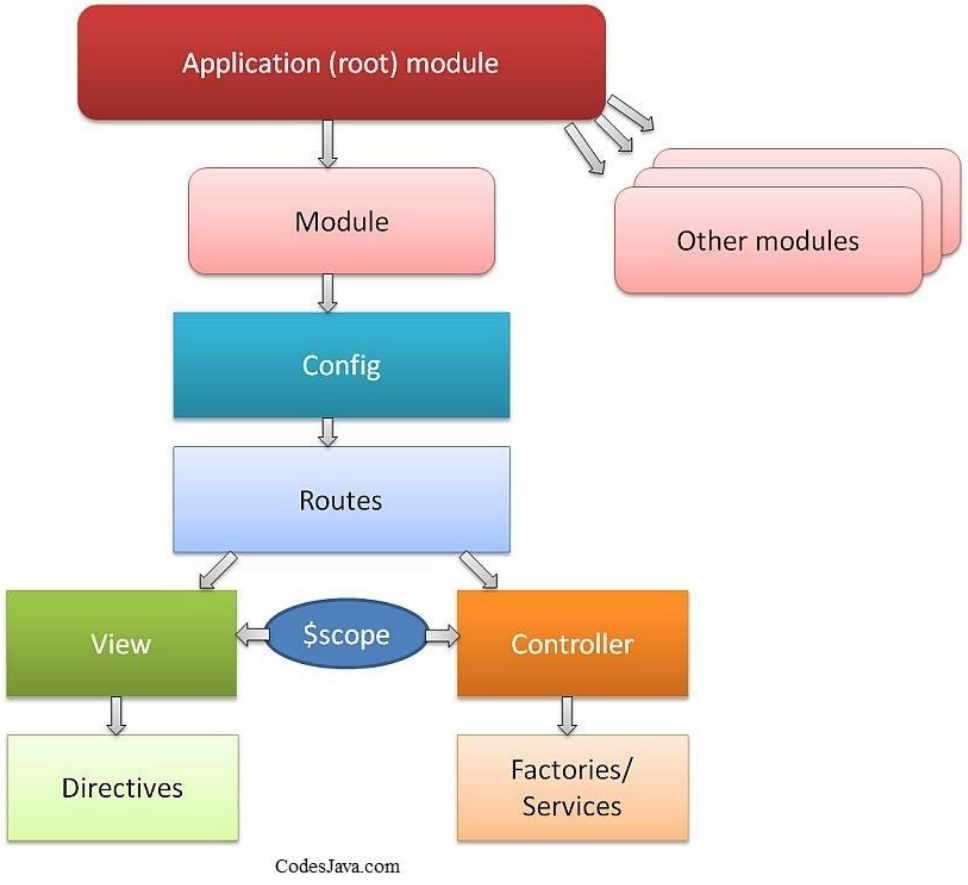
- Le modèle principal pour les systèmes à code source ouvert.
- Les contributions sont gérées par la communauté, qui spécifie les règles et les habitudes de développement et de contribution.
- N'importe quelle contribution est possible, si elle est conforme aux règles.
- Les fonctionnalités sont séparées et isolées. entre les paquets
- La modularité, la testabilité et la réutilisabilité sont augmentées.
- La maintenabilité souffre comme dans le cas des plugiciels.

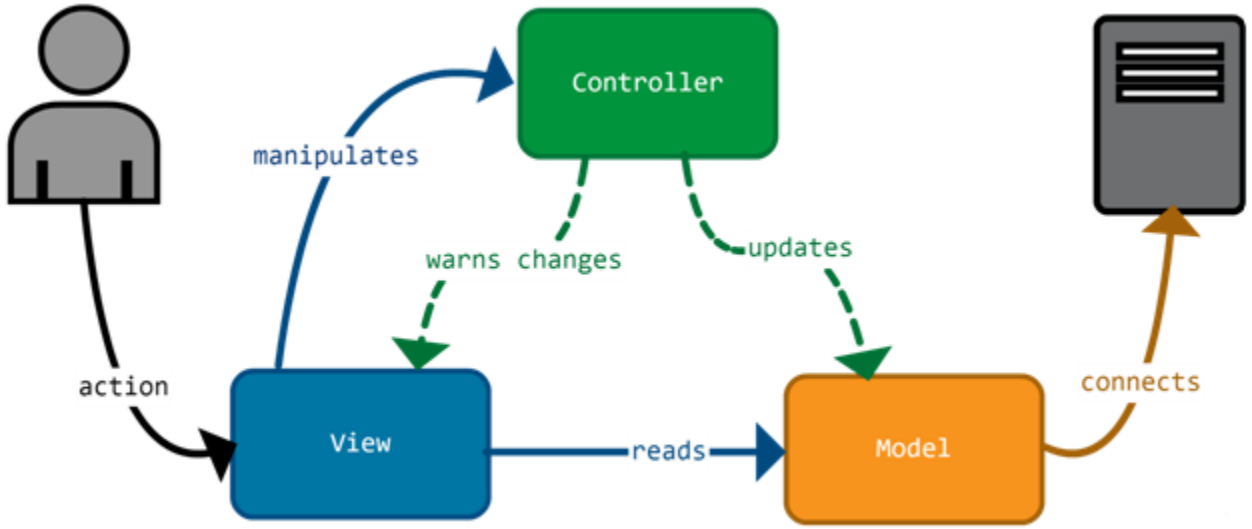
Exemples

Angular

- Angular est un cadriciel pour le développement d'applications web en javascript/Typescript.
- Le cadriciel suit l'architecture MV*.
  - Il est possible d'implémenter n'importe quelle variante du style : MVC, MVP, MVVM
- Les principes d'Angular sont
  - Moins de code.
  - Automatisation augmentée : association des données, transition des pages.
  - Testabilité et réutilisabilité augmentée.
- Les désavantages :
  - Complexe à apprendre pour un débutant.

Architecture d'AngularJS



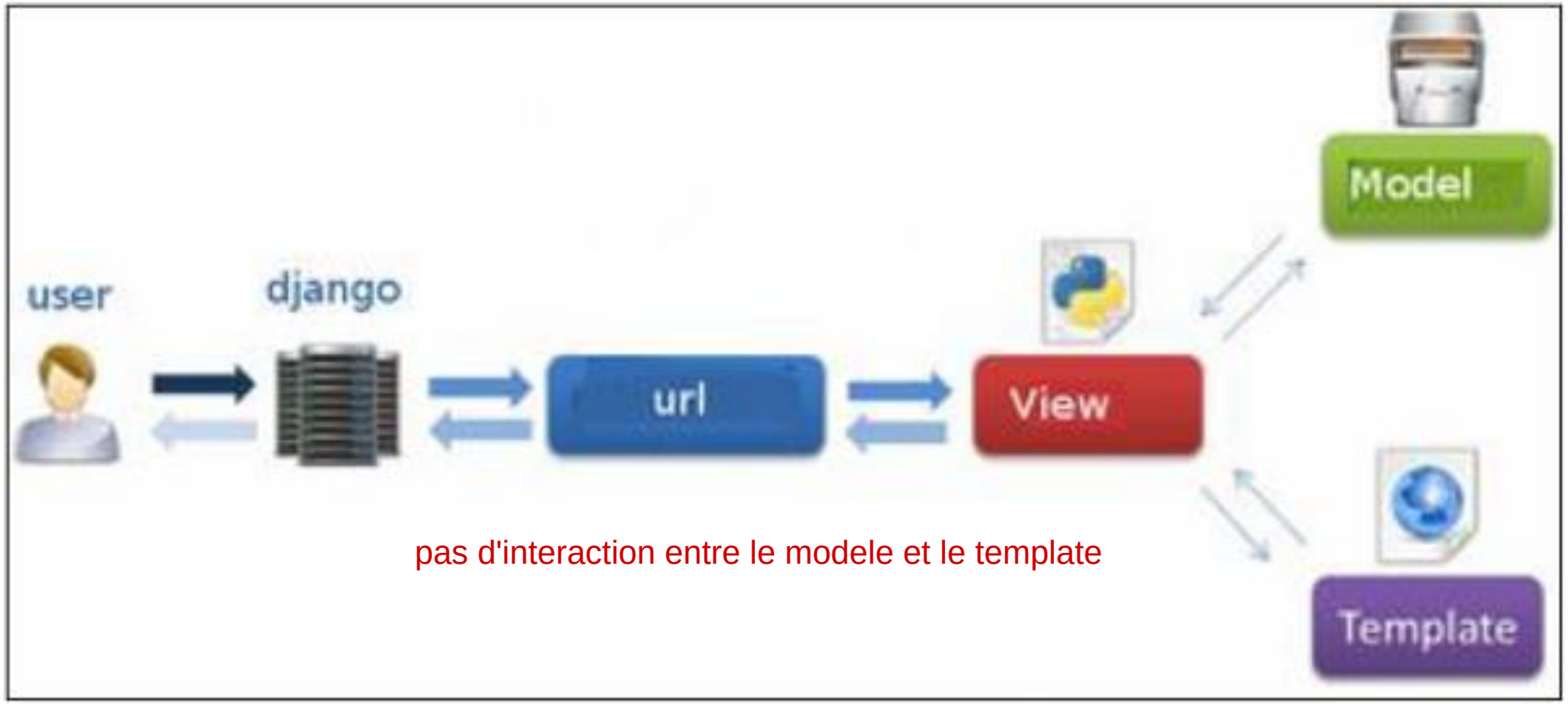


Django

- Django est un cadriciel pour le développement d'applications web en python.
- Le cadriciel suit une architecture MVT (Model-View-Template).
  - Le contrôleur fait partie de Django et pas de l'application
  - Les Templates sont un mélange de HTML et de DTL (Django Template Language). Le DTL spécifie des actions ou des variables dans le code HTML de la page.
    Django template language
- Les principes de Django
  - Faible couplage, forte cohésion
  - Développement rapide
  - Moins de code
  - SRP garanti, clonage de code minimisé.
  - Sécuritaire, extensible, cohérent.

Le principe SOLID Faible couplage, bcp de dependances à l'interne de la classe mais pas d'utilisation dans d'autres classes. Une classe a une seule responsabilité!!!

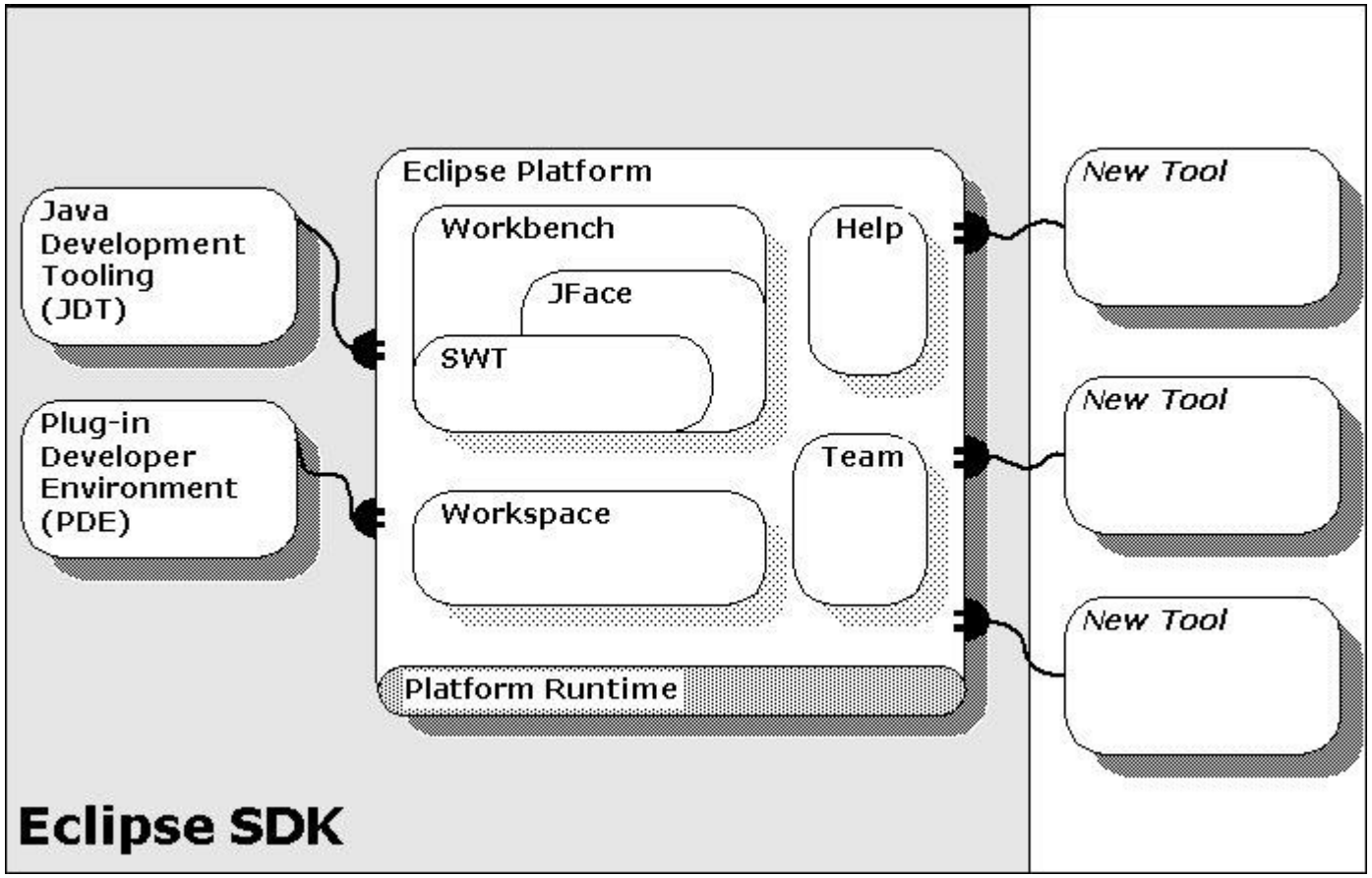
Architecture de Django



Eclipse

- Eclipse est un environnement de développement intégré (IDE).
- À la base, c'est un éditeur Java sophistiqué.
- Eclipse est un plugiciel.
  - Des plug-ins supplémentaires offrent beaucoup plus de fonctionnalités, parmi lesquelles la programmation dans d'autres langages.
- Les plug-ins utilise le SDK (Standard Development Toolkit) pour augmenter la fonctionnalité, l'interface ou les deux.
- L'objectif principal d'Eclipse est l'unification des outils de programmation sous une seule plateforme.
- Eclipse est officiellement appuyé par IBM, mais il y a plusieurs compagnies et développeurs qui s'engagent avec la plateforme.
  - Pour la première version, 490 contributeurs de 40 compagnies se sont engagés.
  - Aujourd'hui, il y a 1000 contributeurs de 170 compagnies.

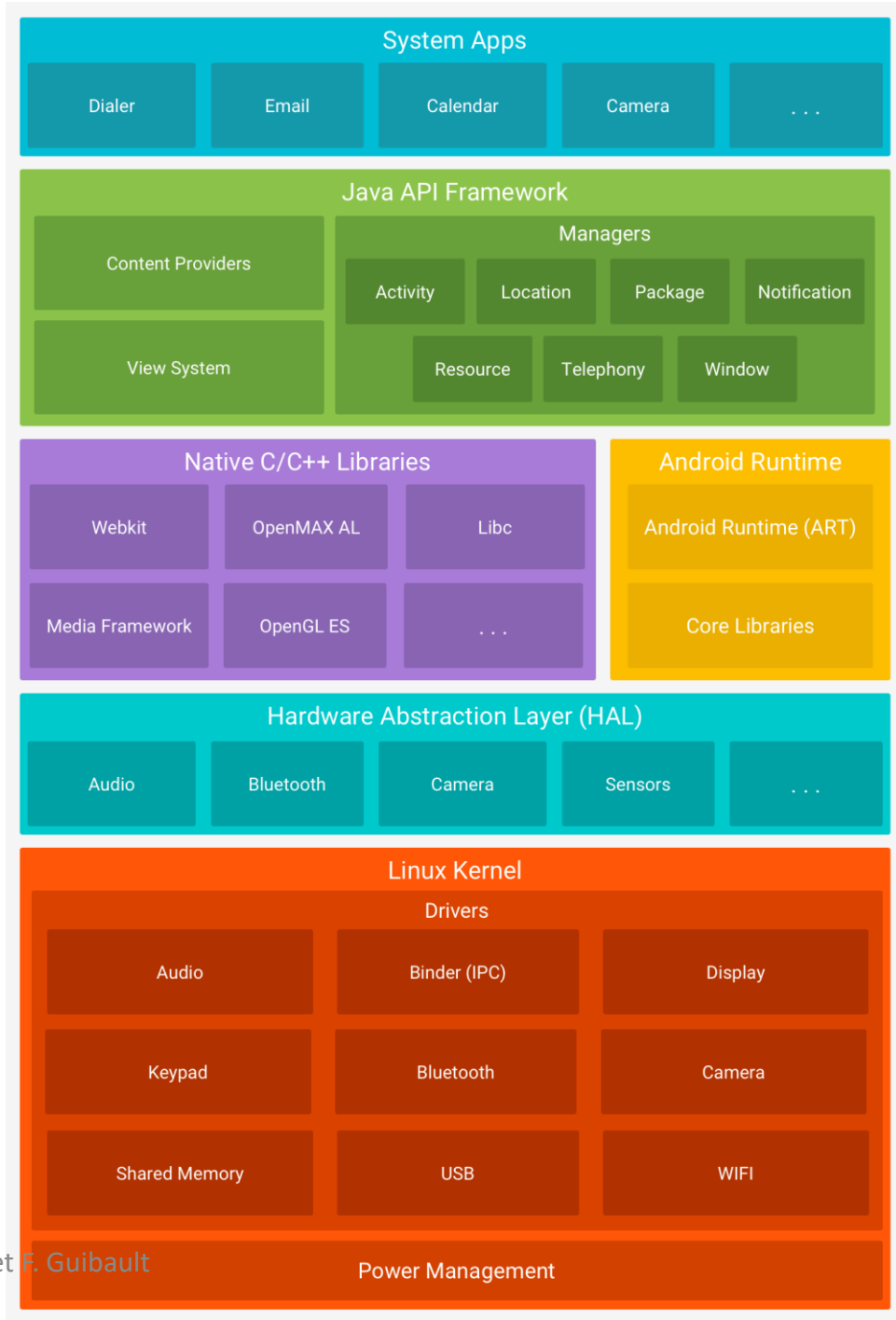
L’architecture d’Eclipse



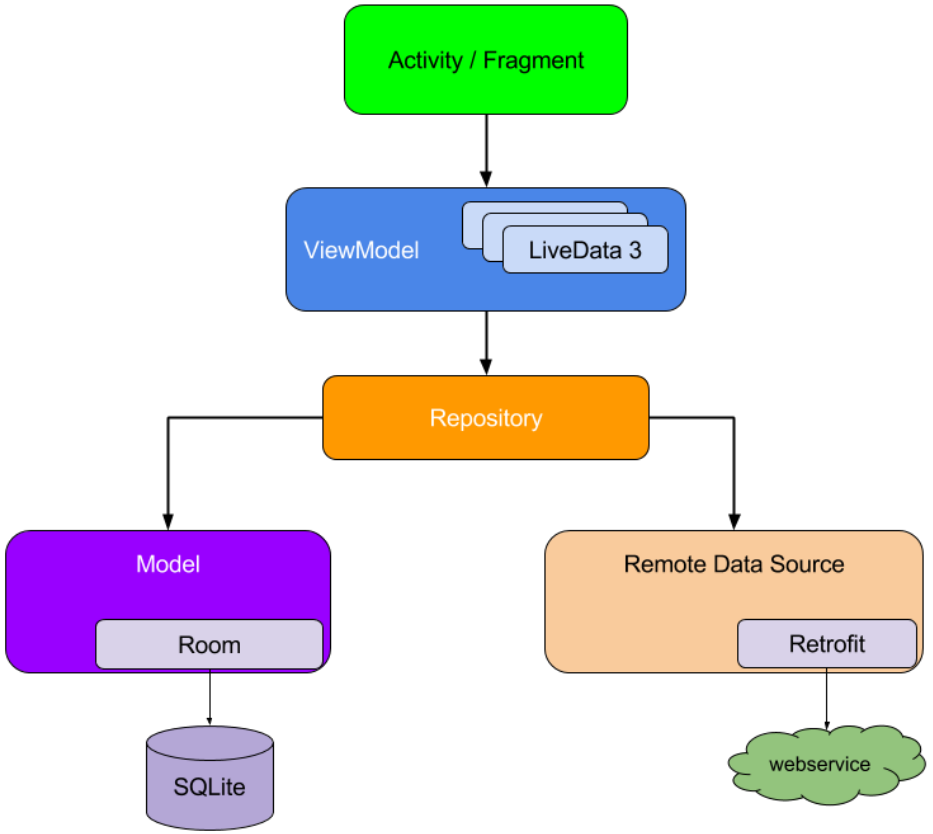
Android

- Android est une plateforme pour le développement d'applications mobiles.
- Android est aussi un système d'exploitation pour les dispositifs mobiles.
- En fait, c'est un cadriciel, un plugiciel et un écosystème en même temps.
- Les applications d'Android suivent une architecture MVVM
  - Modèle – Room/Retrofit
  - ViewModel – Lifecycle, LiveData
  - View - IU

L’architecture d’Android



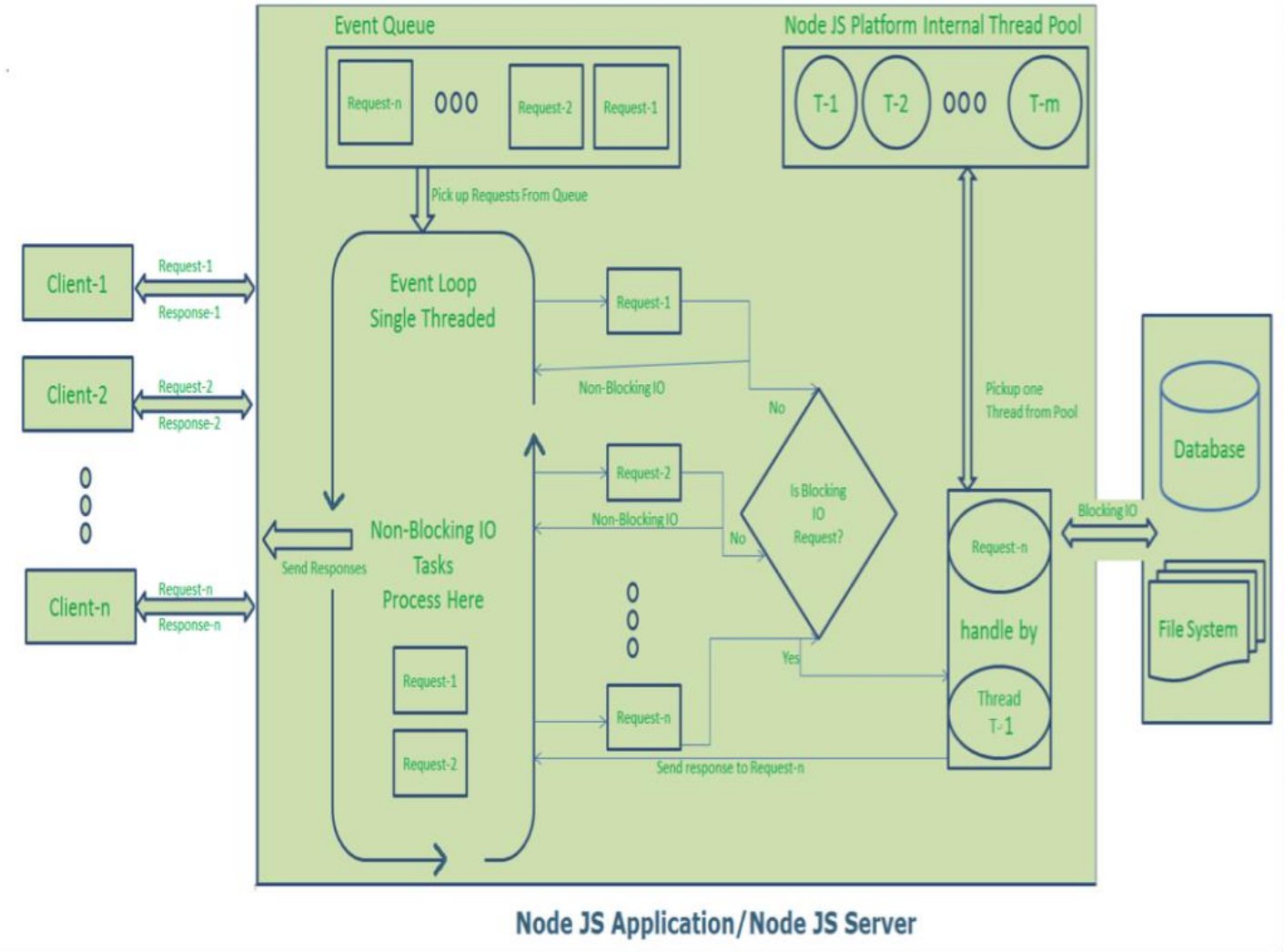
L’architecture des Applications Android



Node.js

- Node.js est un environnement d'exécution de JavaScript.
- Les développeurs peuvent contribuer des modules à Node.js sous forme de paquets.
- Les paquets sont gérés, installés et configurés par npm (Node Package Manager).
- Node.js est une plateforme pour le développement d'applications du côté serveur, et il est aussi un écosystème.
- Il suit une architecture évènementielle.

L’architecture de Node.js



R/CRAN – The Comprehensive R Archive Network

- CRAN est un dépôt distribué pour les paquets de R
  - R est un langage pour les calculs scientifiques et statistiques.
- Les développeurs peuvent contribuer leurs paquets dans CRAN.
- La communauté a des règles pour la soumission des paquets.
- CRAN suit une architecture peer-to-peer.
  - Des serveurs distribués ont des fichiers dupliqués entre eux.
  - La sélection du serveur pour télécharger un paquet est basée sur la proximité.

POLYTECHNIQUE MONTRÉAL
UNIVERSITÉ D'INGÉNIERIE

Développement et maintenance des systèmes distribués

La maintenance des systèmes distribuées

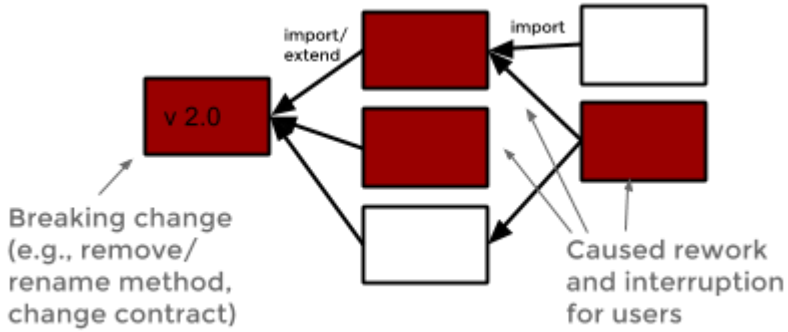
- Affirmation : La qualité de la conception d'un logiciel ne joue aucun rôle dans sa maintenance fonctionnelle.
- Vrai? Deux systèmes conçus différemment (p.ex. monolithique vs modulaire) peuvent avoir une fonctionnalité identique.
- Faux! La qualité de la conception affecte la capacité de maintenir efficacement la fonctionnalité du système.
  - Une conception plate ou complexe peut compliquer la maintenance, spécialement en cas d'un changement externe.
  - Alors, on fait le « ménage » à l'avance, afin d'être prêts à accueillir nos « visiteurs ».

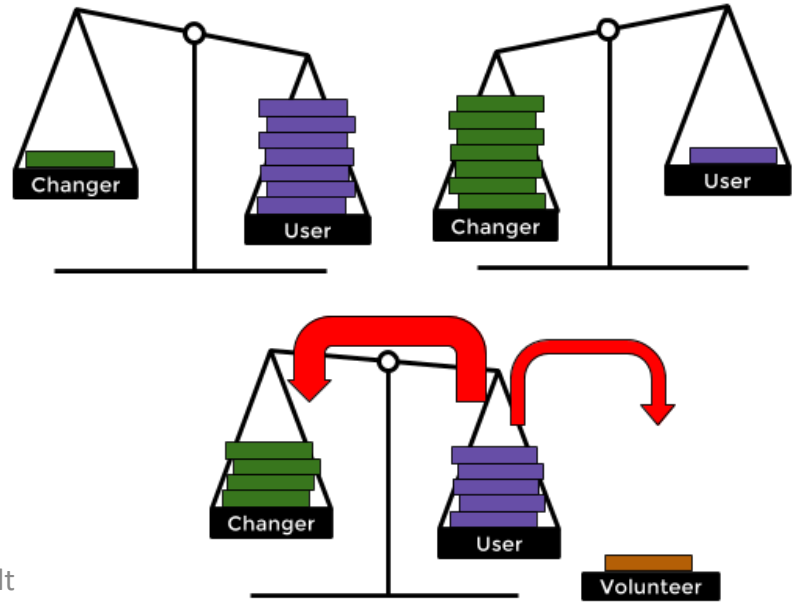
How to break an API?

- Des entrevues avec des développeurs de trois écosystèmes (Eclipse, R, Node.js)
- Des sondages auxquels ont répondu 2000 développeurs de 18 écosystèmes.

Définition de l'étude

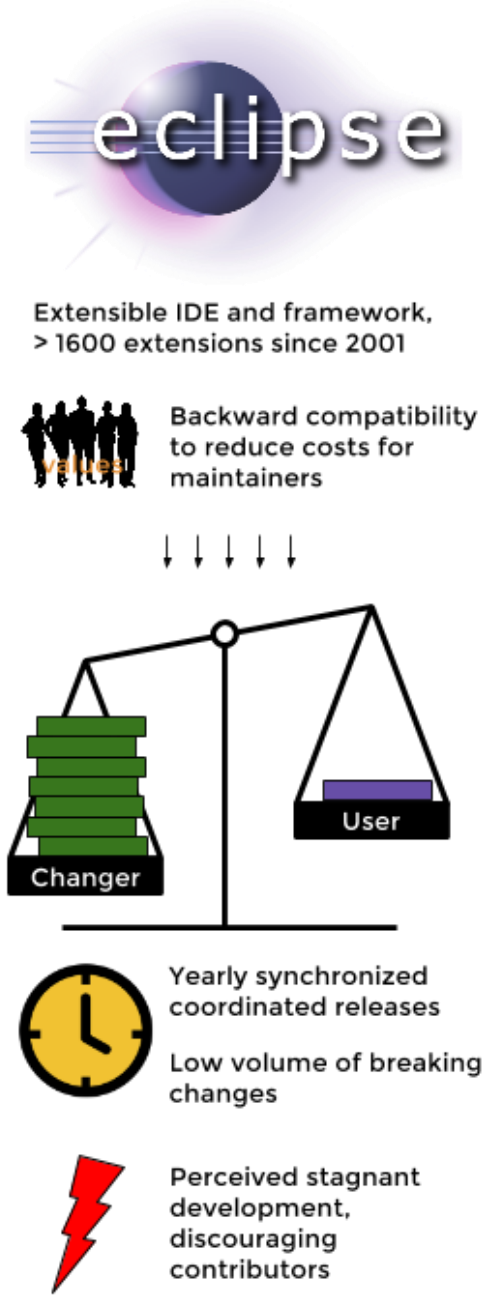
- **Changements de rupture** : À cause des dépendances entre des modules qui sont développés indépendamment, des changements peuvent se propager à travers l'écosystème.
- **Négociations de l'impact des changements de rupture** : La stratégie des développeurs par rapport à qui serra responsable de s'adapter aux changements.





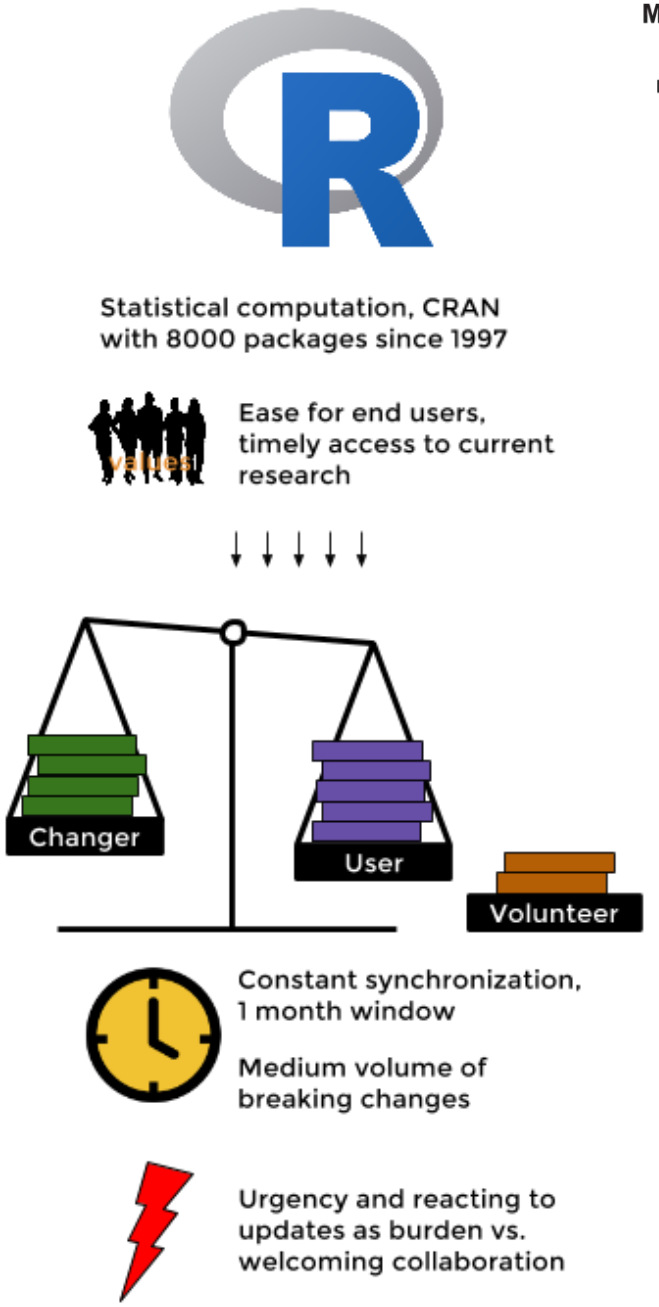
Eclipse

- Valeur : La rétrocompatibilité pour réduire les coûts pour les utilisateurs.
- Eclipse se met à jour chaque année avec peu de changements de rupture.
- Couteux pour les mainteneurs.
- Effort et frais minimaux pour les utilisateurs
- Le développement est perçu comme stagnant.
- Les nouveaux contributeurs sont découragés.



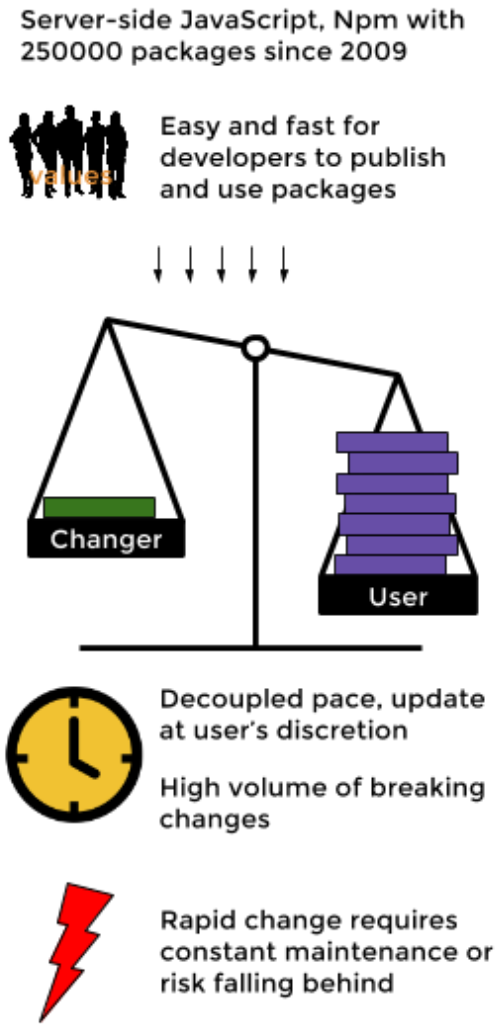
R/CRAN

- Valeurs : Facile pour les utilisateurs, accès rapide à la recherche courante.
- Les paquets dans CRAN se synchronisent chaque mois.
- Changements de rupture en nombre moyen.
- Les frais sont partagés entre les mainteneurs et les utilisateurs.
- Des collaborations avec des mainteneurs externes sont encouragées.
- L'urgence de réagir aux mises à jour peut être perçue comme un obstacle.



Node.js

- Valeur : La publication et l'utilisation des paquets doit être facile et rapide pour les développeurs.
- La mise à jour des paquets est indépendante et découplée.
- C'est la responsabilité de l'utilisateur de mettre à jour son paquet basé sur les changements d'autres paquets.
- Beaucoup de changements de rupture.
- L'évolution rapide requiert une maintenance constante ou de confronter le risque d'être déphasé.



Alors, comment casser l'écosystème?

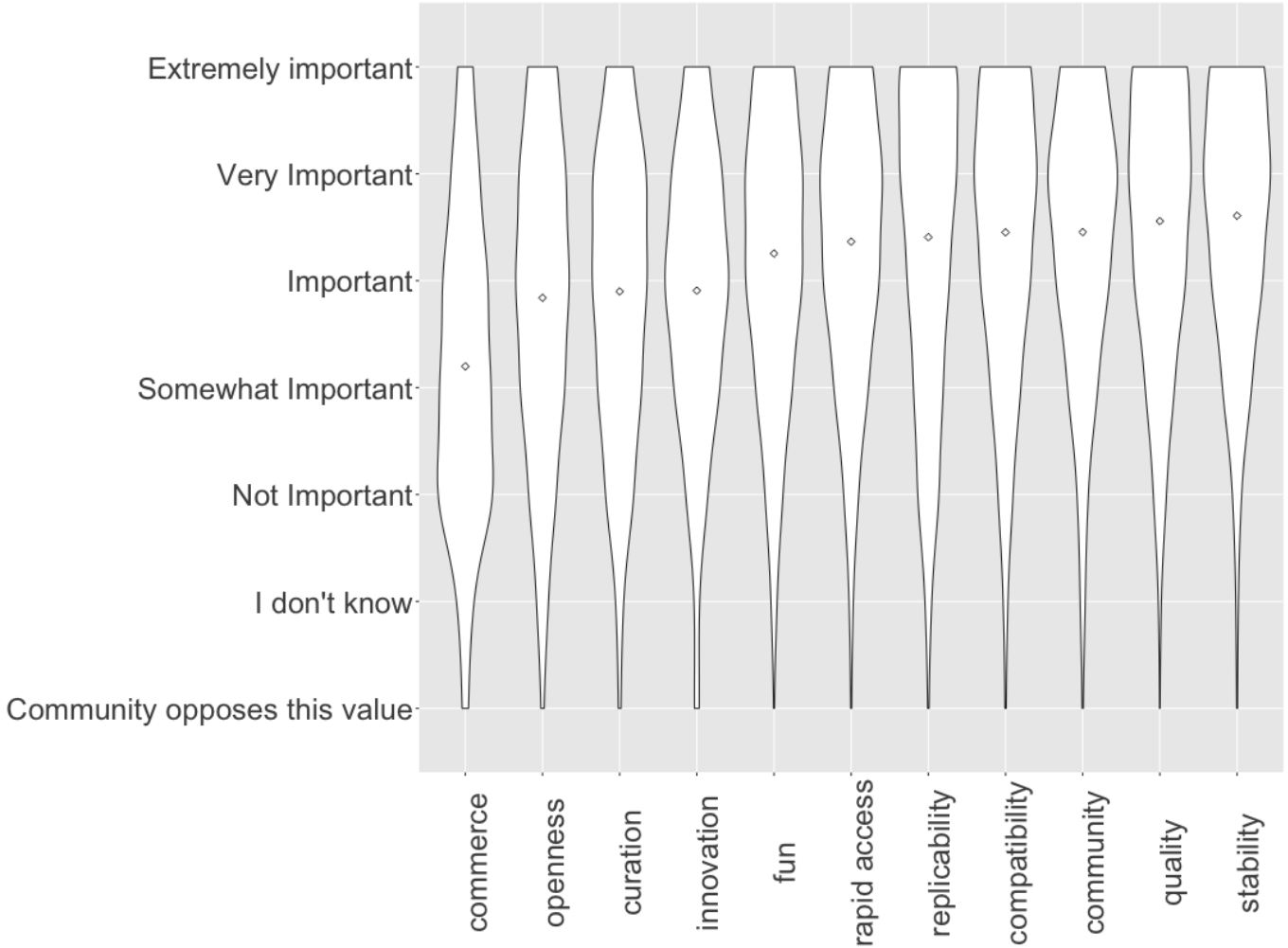
- Eclipse : pas possible.
- R/CRAN : communiquer avec les développeurs affectés.
- Node.js : incrémenter le numéro majeur de la version.

car processus rigoureux pour maintenance et màj des modules

notifier les utilisateurs

on est allé de version 2.0 à 3.0

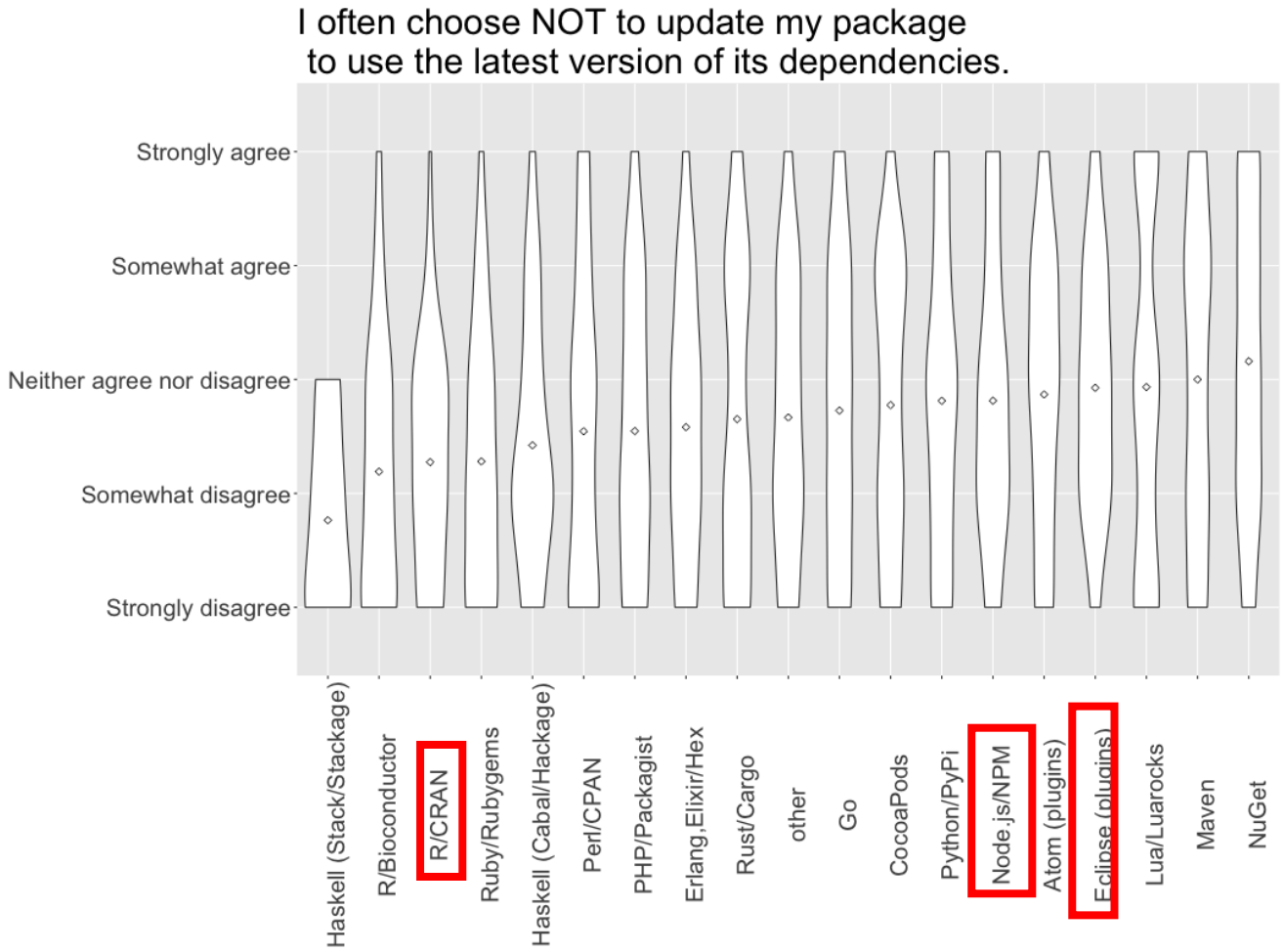
Quelle valeur est la plus importante pour chaque écosystème?



La fréquence des changements de rupture

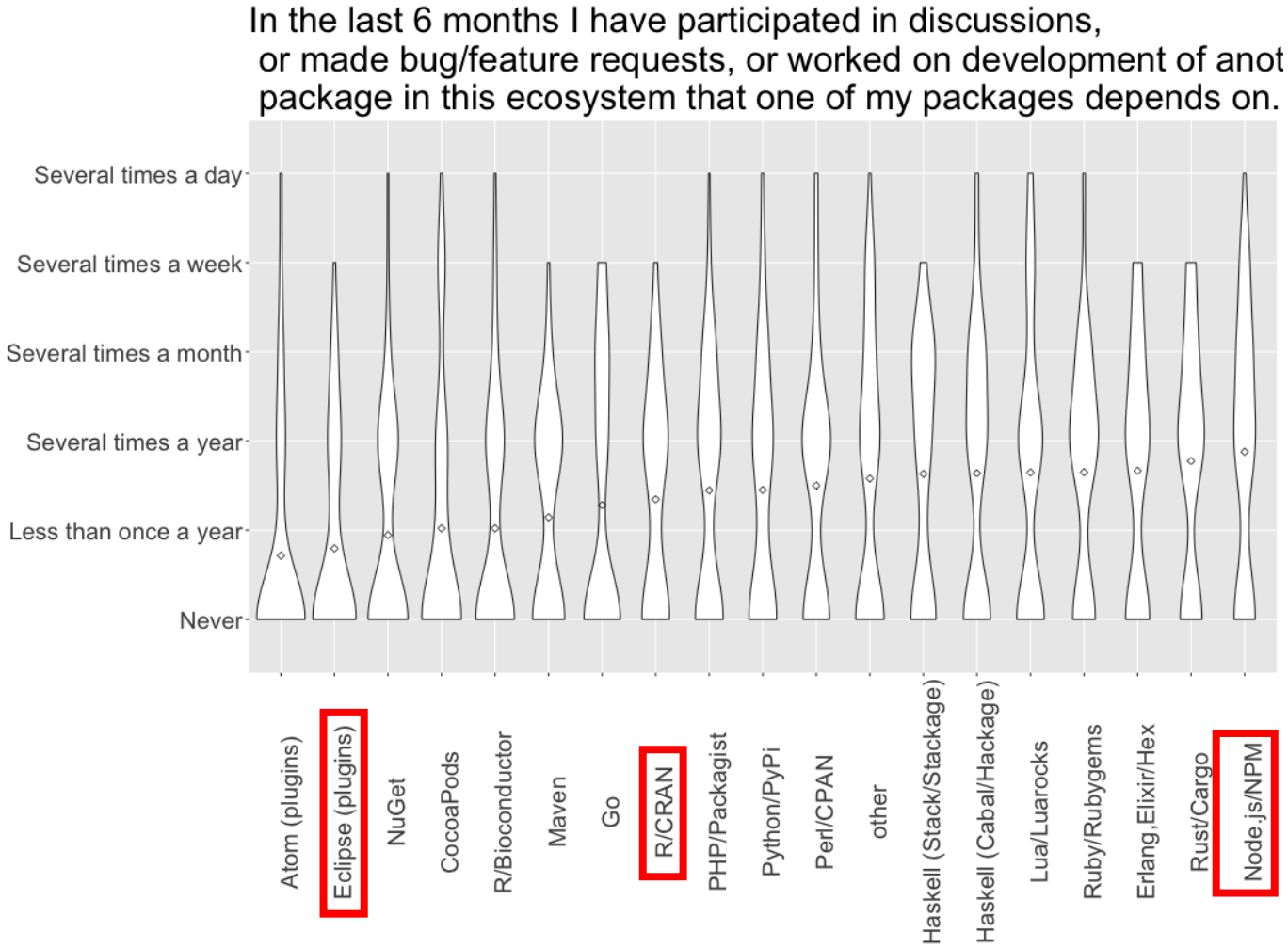
La coordination entre les mainteneurs

Réticence à évoluer à cause des dépendances



Ça vaut la peine d'ajouter une dépendance si elle ajoute une valeur importante.

La fréquence de la collaboration et de la communication entre les mainteneurs



La prochaine fois

LOG2410

Conception OO (SOLID)

Patrons de Conception

LOG8371

LOG3430

Qualité de la conception