Le design des systèmes de machine learning 3

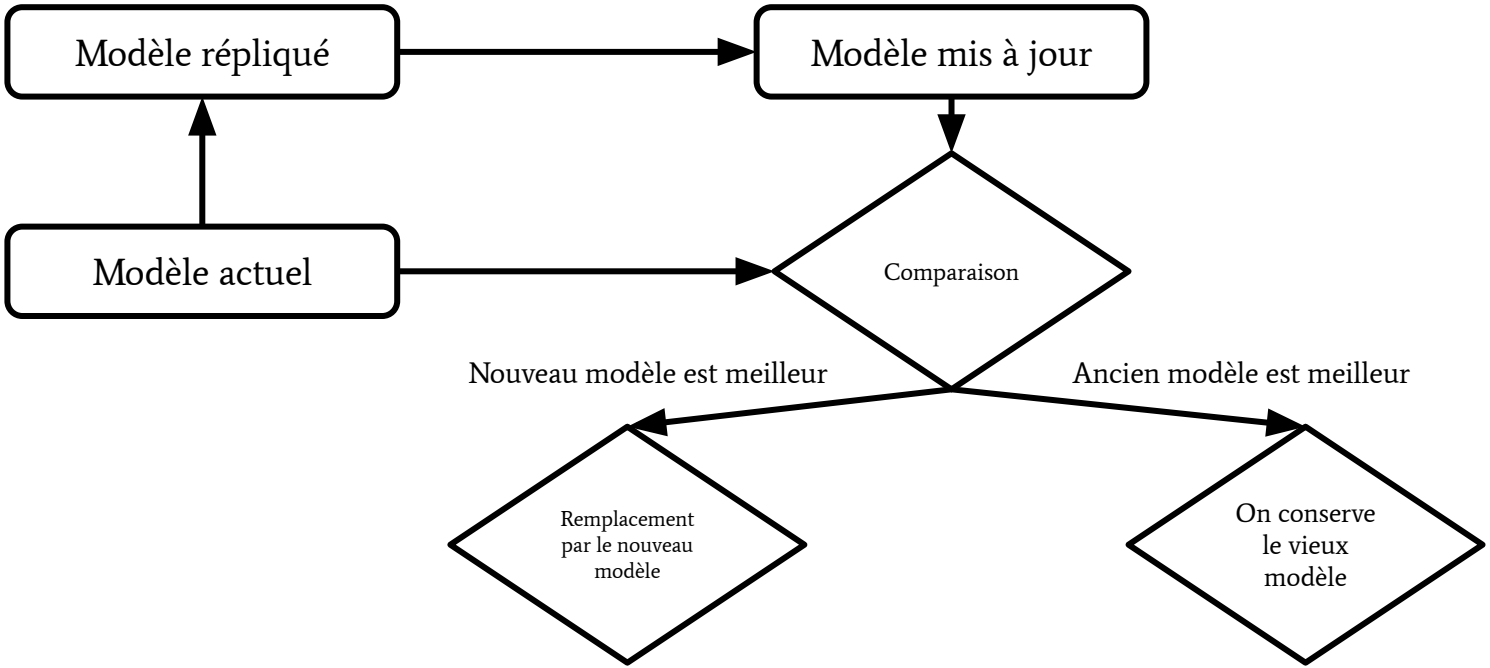
Julien Romero

Apprentissage en continu

Comment adapter notre modèle aux data shifts ?

- Apprentissage continu : on veut mettre à jour notre modèle fréquemment
- À chaque nouveau point de données ?
  - Oubli catastrophique : peut causer un oubli de ce qui était appris précédemment
  - Très cher à exécuter : Hardware fait pour du traitement en batch
- En général, on attend au moins un batch de donnée
  - Et on entraîne à intervalle régulier
- En pratique, on crée un réplica et on vérifie que le réplica est bien meilleur

Comment adapter notre modèle aux data shifts ?



Réentraînement stateful ou stateless

- Finetuning ou from scratch
- Pourquoi faire du stateful/finetuning ?
  - Demande moins de puissance de calcul
  - Converge plus rapidement
  - Pas besoin de stocker les données (place + vie privée)
- On peut combiner les deux approches

L'apprentissage continu est un problème d'infrastructure

- Il faut designer votre infrastructure pour :
  - Mettre à jour vos modèles quand c'est nécessaire
  - Déployer les modèles rapidement
- Deux types de mises à jour
  - Itération de modèle : on change l'architecture ou on rajouter une feature
  - Itération de données : on ne change pas le modèle, mais on utilise de nouvelles données

Pourquoi faire de l'apprentissage continu ?

- Lutte contre les data shift
  - Surtout événements soudain
  - Ex : Uber dans un quartier tranquille doit pouvoir se mettre à jour si un gros événement s'y produit
- Adaptation aux événements rares
  - Ex : le Black Friday se produit rarement et les habitudes d'achats changent par rapport aux autres jours et Black Friday précédents
- Mitigation du cold start problem
  - Nouveaux utilisateurs, produits, nouveau comportement (téléphone au lieu d'un ordinateur), site sans identification, usage peu fréquent du site (dernières données sur l'utilisateur périmées)
  - TikTok peut s'adapter en quelques minutes

Les défis de l'apprentissage continu

- Accéder aux données récentes
  - Souvent, on utilise des données dans des data warehouses, qui sont lentes à se mettre à jour
  - Utilisation de Kafka nécessaire
  - Il faut annoter ces données
    - Étiquettes naturelles de préférence
    - Parfois, il faut joindre le comportement de l'utilisateur (click) avec la prédiction du modèle. Ces deux événements sont séparés. C'est de la label computation.
      - Batch processing plus facile, mais latence élevée.
    - Si pas le choix, on peut faire du programmatic labeling (algorithme non ML) ou du crowdsourcing (demander à des humains)

Les défis de l'apprentissage continu

- Accéder aux données récentes
- Évaluation
  - Il faut que le modèle soit meilleur
  - Risque d'erreurs très coûteuses
  - Risque d'attaque antagonistes : manipulation du modèle en générant de faux comportements
  - Ex : Modèle Tay de Microsoft devenu raciste en quelques heures sur Twitter
  - L'évaluation prend du temps

Les défis de l'apprentissage continu

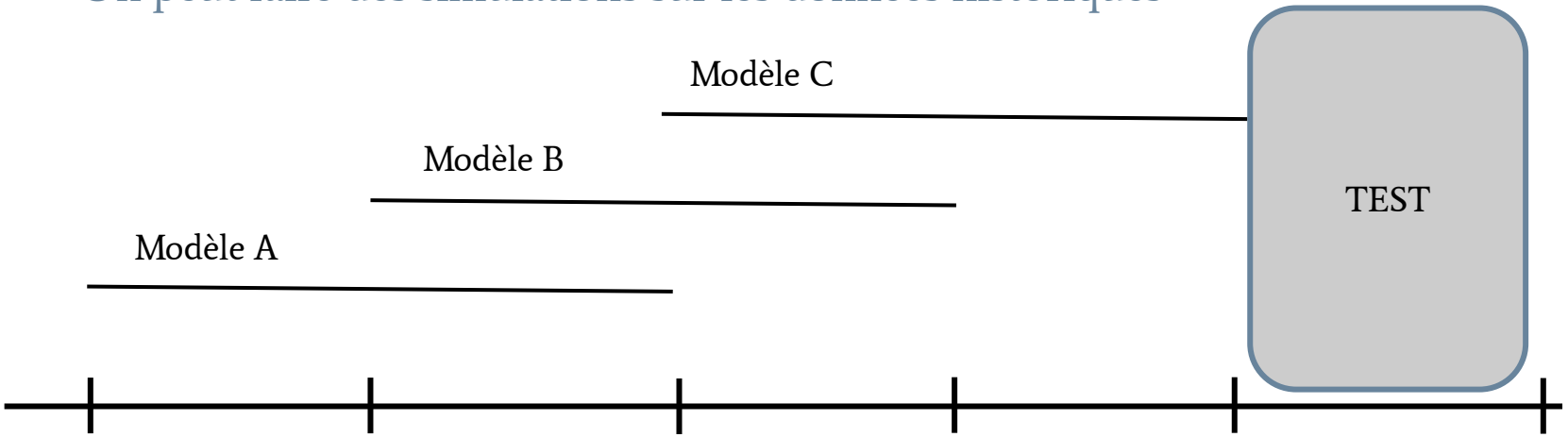
- Accéder aux données récentes
- Évaluation
- Algorithmes
  - Difficile de mettre à jour fréquemment certaines approches, surtout celle basées sur des matrices et des arbres
    - Pour le réseau de neurones, facile, mais faire de la factorisation demande beaucoup de temps et toutes les données
  - Calcul de statistiques en ligne pour mettre à l'échelle par exemple

Les quatre niveaux de l'apprentissage continu

1. Entraînements manuels et stateless
   a. Au début, une entreprise cherche à développer pleins de modèles
   b. Mettre à jour un modèle peut être difficile si les choses ont changé depuis la dernière fois
      i. Requête base de données, extraction des features, réentraînement, export du modèle, déploiement
2. Entraînements automatisés
   a. On a un script qui fait tout pour nous : récupération des données, downsample/upsample, extraction de features, création du jeu d'entraînement avec des étiquettes, entraînement, évaluation, déploiement
   b. Demande un scheduler, des données accessibles, et un stockage pour le modèle
3. Entraînements automatisés et stateful
   a. Il faut bien tracker la version de son modèle, par exemple X.Y signifie modèle stateless X après Y mises à jours
4. Apprentissage continu
   a. Réentraînement déclenché par des événements : temporels (toutes les 5 minutes), performance (perte de X%), volume (+X% de données d'entraînement), data shift

À quelle fréquence mettre à jour les modèles ?

- On peut faire des simulations sur les données historiques



À quelle fréquence mettre à jour les modèles ?

- On peut faire des simulations sur les données historiques
- Évaluer les gains
  - Peut-être passer du temps à faire une itération sur le modèle est plus avantageuse

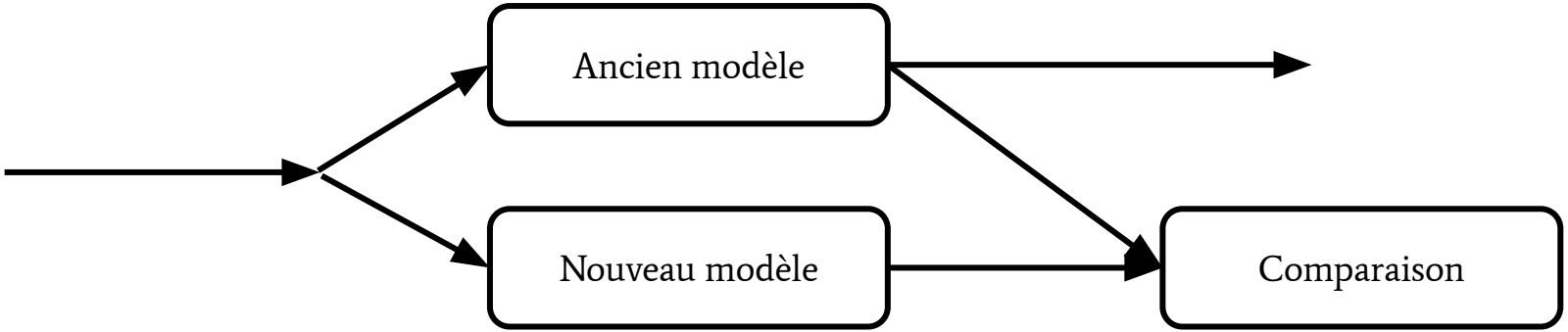
Tests en production

Tests en production

- Souvent, on fait un train/val/test split statique
  - Permet de comparer des baselines
  - Ne rend pas compte des changements dans les données
- Offline, on peut aussi faire des backtest
  - Entraîne sur une période de temps spécifique et on teste sur les données après cette période
  - Attention ! Certaines périodes temporelles peuvent contenir des données corrompues, rendant l'évaluation caduque
- Le mieux : tester en production

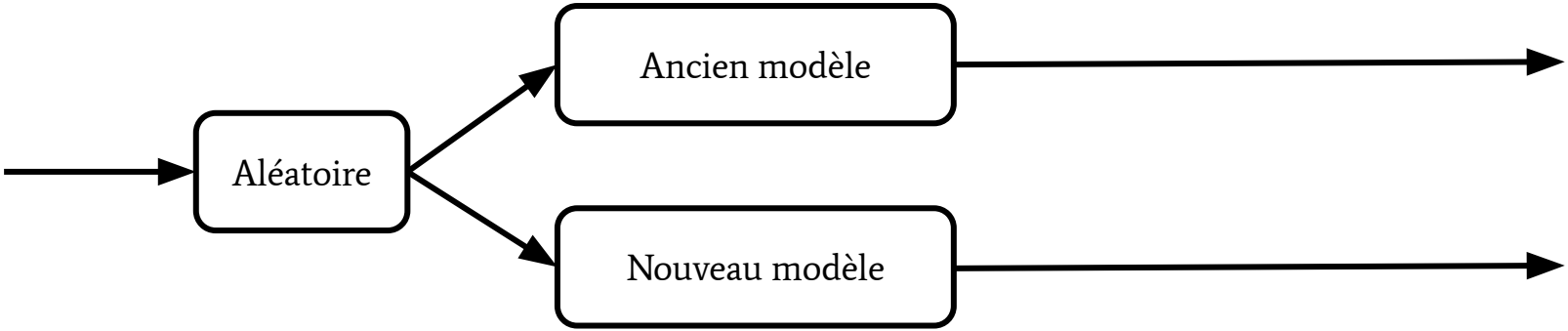
Shadow Deployment

1. On déploie le nouveau modèle en parallèle de l'ancien.
2. Toutes les requêtes passent par les deux modèles, mais seuls les résultats de l'ancien modèle sont retournés à l'utilisateur.
3. On compare les résultats des deux modèles.



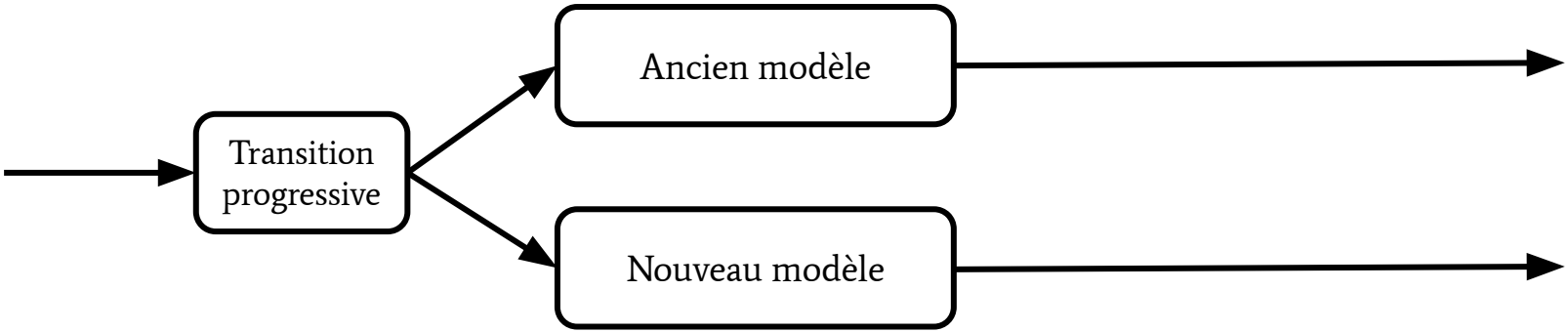
A/B Testing

1. On déploie le nouveau modèle en parallèle de l'ancien.
2. On redirige aléatoirement les requêtes vers un des modèles
   a. Parfois, à cause dépendances entre les modèles, on peut prendre un modèle aléatoire par utilisateur ou par jour
3. Comparaison des performances (tests statistiques)



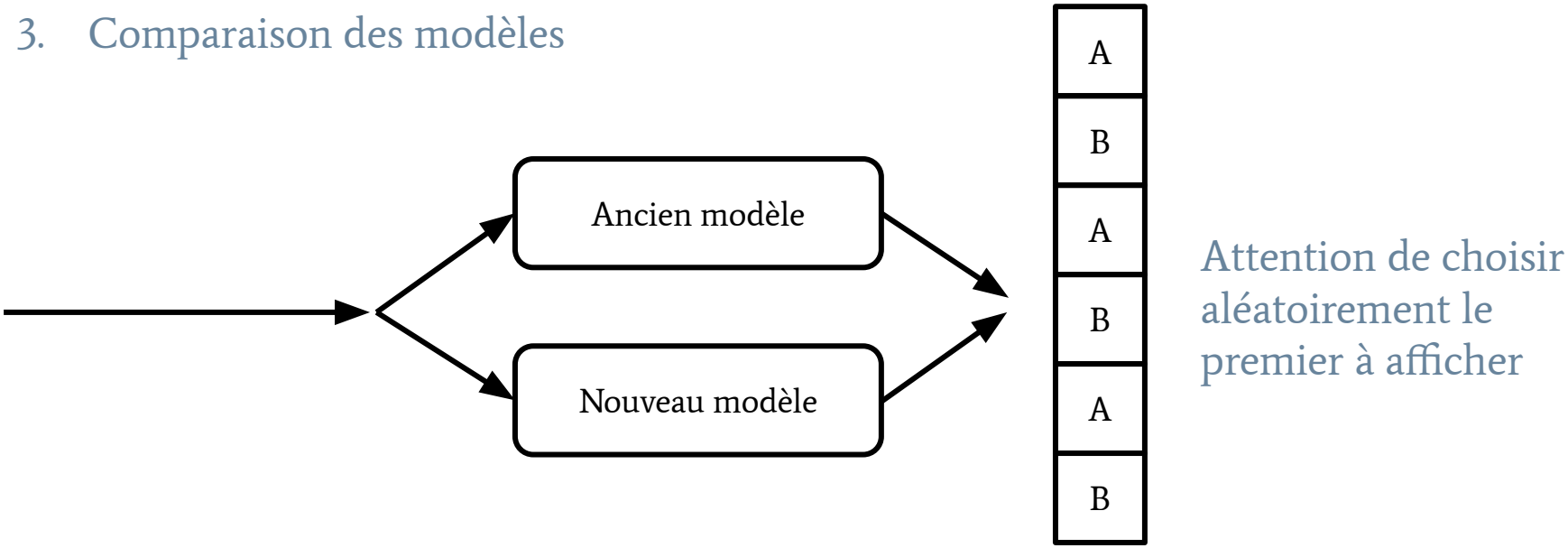
Canary Release

1. On déploie le nouveau modèle en parallèle de l'ancien.
2. Si tout se passe bien, on accroît progressivement le trafic vers le nouveau modèle
3. Sinon, on arrête



Entrelacement

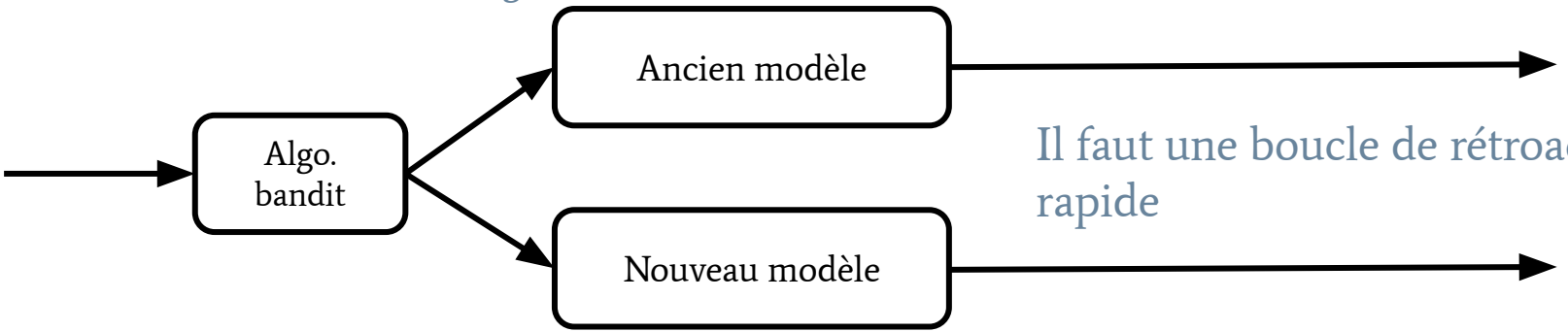
1. On déploie le nouveau modèle en parallèle de l'ancien.
2. Si possible (système de recommandation), on mélange la sortie des deux modèles
3. Comparaison des modèles



Attention de choisir aléatoirement le premier à afficher

Bandit

1. On déploie le nouveau modèle en parallèle de l'ancien.
2. On utilise un algorithme pour balancer l'exploration des modèles et l'exploitation du meilleur
   a. Bandit souvent relié à l'apprentissage par renforcement
   b. Stateful != A/B testing



Il faut une boucle de rétroaction rapide

Infrastructure et outils pour le MLOps

L'infrastructure dépend beaucoup des besoins

- Peu de besoin : business analytics
  - Des notebooks suffisent
- De nombreux besoins uniques : véhicule autonome, Google search
  - Infrastructures ultra-spécialisées
- Au milieu : la plupart des entreprises traitant au maximum quelques terabytes de données, avec une centaines d'ingénieurs
  - Peuvent bénéficier d'outils génériques

Stockage et capacité de calculs

- Il faut définir de combien de stockage et de puissance de calcul notre application a besoin
  - Taille du stockage, type de stockage (HDD/SSD)
  - Nombre de CPU/GPU
  - Bande passante de la mémoire
  - Nombre d'opération par secondes
- On peut opter pour un solution interne ou externe sur le cloud
  - AWS, GCP
  - Pour le cloud, il faut bien connaître ce qui est disponible

Sur la vitesse de calcul

- Souvent donnée en FLOPS (Floating point operation per second)
- Ce nombre est un maximum, mesuré sur des benchmark très spécifiques
- En pratique, on est intéressé par maximiser l'utilisation
  - Ratio nombre de FLOPS exécuté par rapport au nombre maximal théorique
  - Très dépendant de la vitesse de chargement des données

Cloud publique ou data center privé

- Le cloud publique permet :
  - Grande flexibilité
  - Facilité d'utilisation
  - Demande peu d'ingénieurs
  - Très utile au début du développement d'un projet
- Par contre :
  - Problème des données personnelles
  - Très coûteux
  - Difficile de s'en séparer, crée une dépendance
- Certaines entreprises réduisent les coûts en faisant du multi-cloud
  - Compatibilité avec plusieurs plateforme pour sélectionner la moins chère

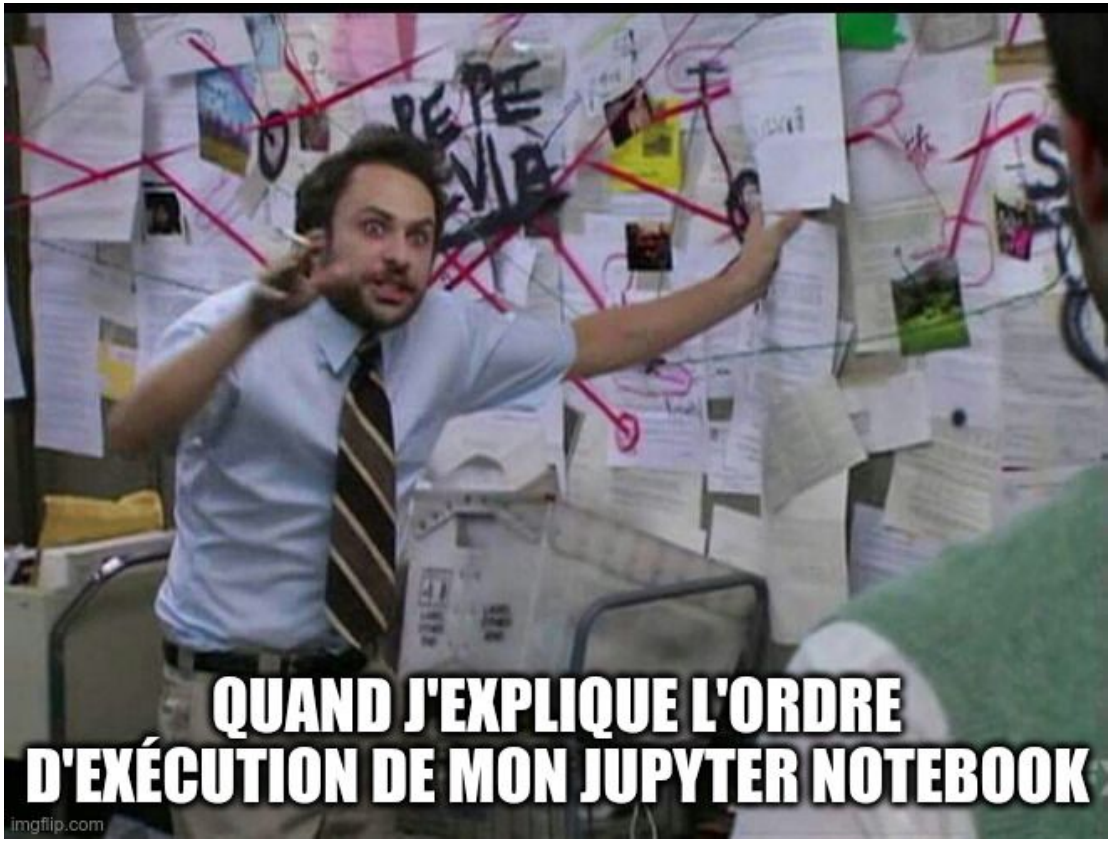
L'environnement de développement

- Environnement sur lequel les ingénieurs ML développent le code, lancent des expérience et interagissent avec l'environnement de production.
- Trois éléments principaux :
  - IDE
  - Versionning (Git)
  - CI/CD (Continuous Integration/Continuous Delivery)
- Sûrement un des éléments de votre architecture les plus importants
  - Rend les ingénieurs plus efficaces

IDE

- De nombreuses options : VSCode, JetBrains, Vim
- Certains IDE sur le cloud, accessibles dans le navigateur
  - AWS Cloud9, GitHub
- Notebooks très populaires en data science
  - Stateful : conserve des résultats intermédiaires, images, graphes, même si erreur
  - Par contre, on peut exécuter les cellules dans n'importe quel ordre, ce qui rend la reproducibilité compliquée

IDE


QUAND J'EXPLIQUE L'ORDRE
D'EXÉCUTION DE MON JUPYTER NOTEBOOK
imgflip.com

Les environnements de développement partagés

- L'environnement de développement devrait être partagé par l'équipe
  - Gestion des versions des bibliothèques
  - Reproducibilité
  - Proche de l'environnement de production
  - Support de la DSI plus facile (pas 1000 machines différentes)
  - Sécurité
  - Télétravail facilité
- Par exemple, environnement virtuel Python accédé en SSH
- Existe des solutions avec tout intégré : AWS Cloud9, Amazon SageMaker Studio, GitHub Codespaces

Du développement à la production : les conteneurs

- En production, on a besoin de déployer rapidement et sur de nombreuses machines notre application
- Comment recréer un environnement équivalent sur toutes les machines ?
  - Utilisation de conteneurs. Solution la plus connue : Docker
  - Principe : on crée une *image* partagée qui représente un configuration. Cette image est générée à partir d'un DockerFile qui décrit comment configurer l'image (installation de bibliothèques, copie de fichiers, exécution de scripts, télécharge des données, ...)
  - Quand on exécute une image, on obtient un conteneur
- Les images peuvent être stockées dans des container registry comme Docker Hub
- Quand on a de nombreuses instances à gérer, on a besoin d'un orchestrateur pour tout automatiser
  - Docker compose sur une seule machine, Kubernetes (K8s) sur un cluster

Gestion des ressources

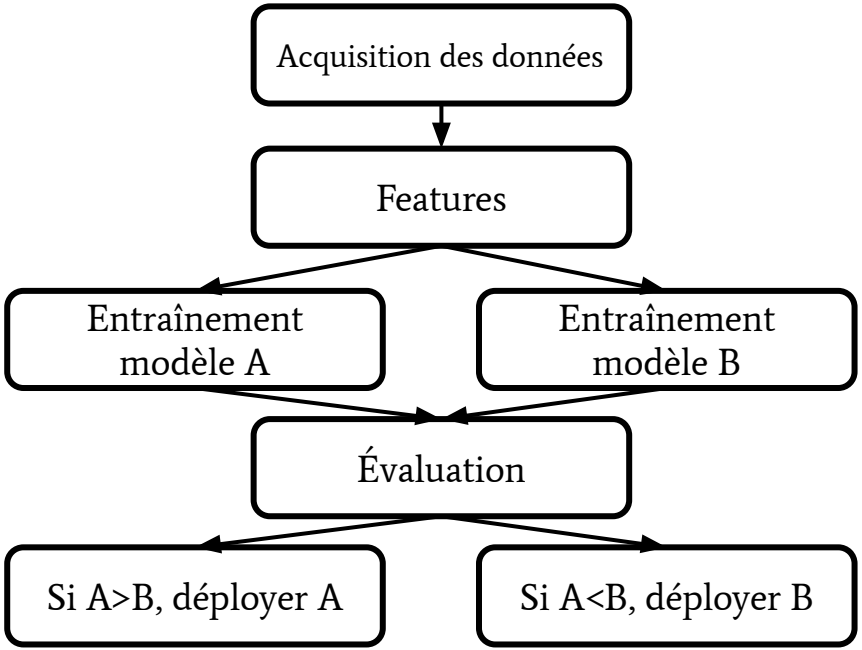
- Gérer les ressources de stockage et de calcul est crucial
- Dans un data center privé
  - Ressources limités
  - Maximisation de l'utilisation des ressources
  - Demande beaucoup de travail d'ingénieurs
- Dans un cloud public
  - Ressources élastiques
  - On ne cherche pas à maximiser l'utilisation des ressources, mais plutôt à trouver la solution la moins chère
- Compromis entre plus de calculs et moins de ressources humaines

Crons

- Dans les workflow ML, on exécute beaucoup de tâches répétitives avec des dépendances entre elles
- **Cron** = programmation de tâches répétitives à intervalles de temps réguliers
  - Pas de dépendances entre les tâches
- On a souvent des dépendances entre les tâches :
  - 1. Récupérer les dernières données
  - 2. Extraire les features
  - 3a. Entraîner un modèle A
  - 3b. Entraîner un modèle B
  - 4. Comparer les modèles
  - 5. Déployer le meilleur modèle

Schedulers

- Les dépendances forment des graphes orientés acycliques



Schedulers

- Les dépendances forment des graphes orientés acycliques
- Un **scheduler** est un programme cron qui gère les dépendances
  - Souvent déclenché par un événement
  - Doit connaître les ressources disponibles et les ressources nécessaires pour chaque tâche (soit données par l'utilisateurs, soit apprises)
  - Optimisation de l'utilisation des ressources
- Exemple du cours : Slurm

Orchestrateurs

- Les schedulers gèrent quand il faut exécuter une tâches et quelles sont les ressources nécessaires
- Un orchestrateur gère où obtenir ces ressources
  - Plus bas niveau
  - Plus souvent utilisés pour des tâches longues comme un site web
- Dans Hadoop, utilisation de YARN
- Le plus connu est Kubernetes

Gestion des workflows en data science

- En data science, nous avons beaucoup de workflows pour gérer les différentes étapes de la vie d'un modèle de ML
  - Chargement des données, création des features, entraînement, test, ...
- Ils peuvent être décrit avec des graphes orientés acycliques
- Il existe de nombreux outils spécifiques :
  - Airflow, Argo, Perfect, Kubeflow, Metaflow, ...
  - On y décrit des tâches et comment elles sont connectées (en Python ou YAML)
  - Pour choisir, se renseigner sur les fonctionnalités (graphes statiques ou dynamiques avec créations de nouvelles étapes à l'exécution, graphes pouvant prendre des paramètres comme les hyperparamètres, possibilité de découper en plusieurs conteneurs, ...)

Les plateformes ML

- Les plateformes ML sont des logiciels rassemblant des outils pour de déploiement
- Souvent plusieurs composantes :
  - *Déploiement de modèle* : mise en production des modèles, de leurs dépendances, et création des points d'accès (MLflow Models Seldon, Cortex, Ray Serve, AWS SageMaker, GCP Vexter AI, Azure ML, ...)
  - *Stockage des modèles (model store)* : solution spécialisée pour enregistrer des modèles et y accéder facilement. On enregistre la définition du modèles, les paramètres, les fonctions pour créer les features et pour faire une prédiction, les dépendances (version Python, bibliothèques), données utilisées (version), code ayant généré le modèle, artéfacts (logs, courbes), tags
  - *Stockage de features (feature store)*: enregistrement d'informations liées aux features. Gestion des features (documentation, partage de features à travers un catalogue), calcul des features (script pour calculer les features et si le calcul est coûteux, pour stocker le résultat), cohérance des features (entre la production et le développement)

En résumé

- Apprentissage en continu
  - Défi et choix à prendre
  - Fréquence des mises à jours
- Tests en production
  - Shadow deploiement, A/B testing, canary release, experiences entrelancées, bandits
- Infrastructures et outils
  - Pour le stockage et le calcul
  - Pour les développeurs
  - Pour la gestion des ressources
  - Pour le déploiement des modèles