



المدرسة الوطنية للذكاء الاصطناعي والرقمنة - بركان
ÉCOLE NATIONALE DE L'INTELLIGENCE ARTIFICIELLE ET DU DIGITAL - BERKANE
ἡλᾱοοο. ἡ8ε1ᾱ. ἡ8ἡἈΚ. ἡ8ο8ε1.ἡ8ε ἡ8ἡο8ε1. - ἡοΚ.ἡ

MACHINE LEARNING 2

Pr. BOUTAHIR Mohamed Khalifa



2024 - 2025

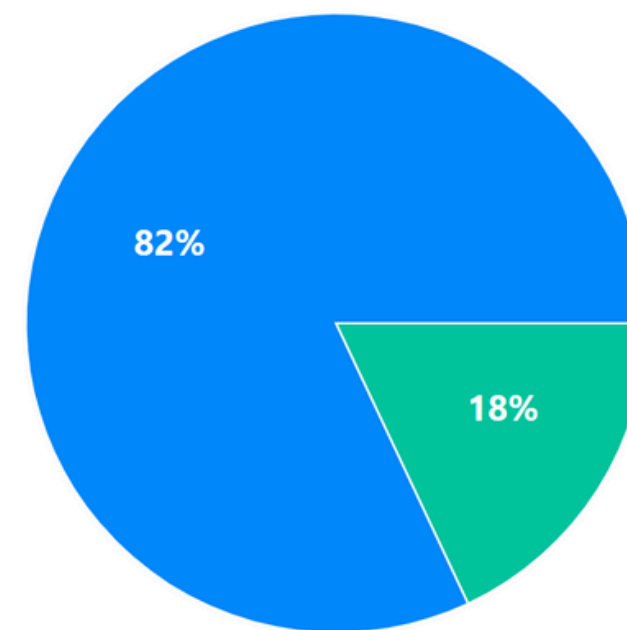
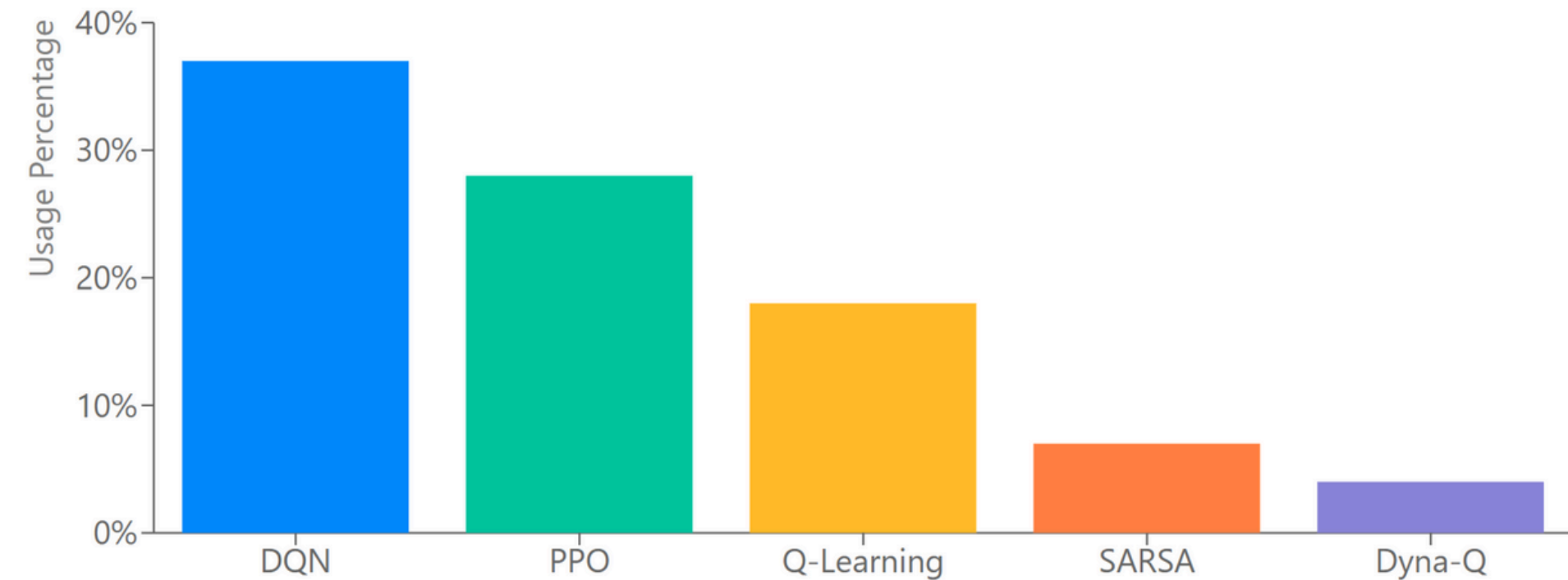
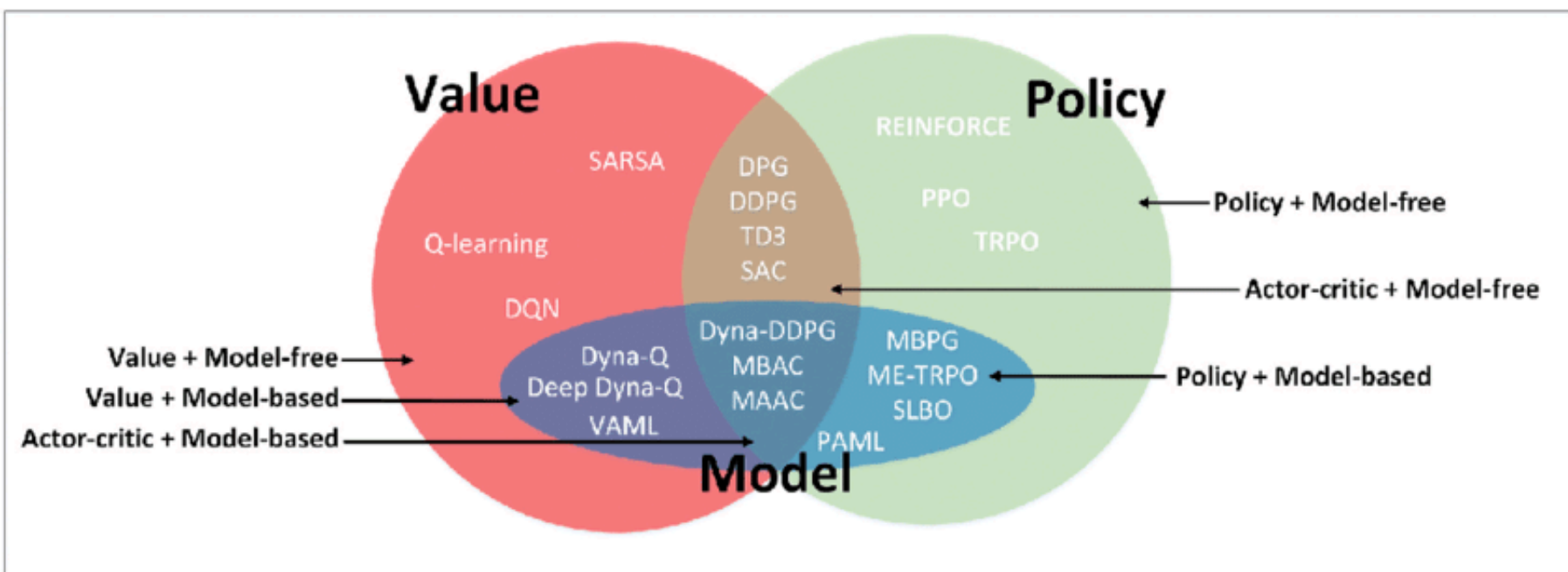
○ OBJECTIFS DE LA SESSION

Sur cette partie, nous allons explorer les principaux algorithmes de l'apprentissage par renforcement (RL) et comprendre comment ils fonctionnent et dans quels contextes les utiliser.

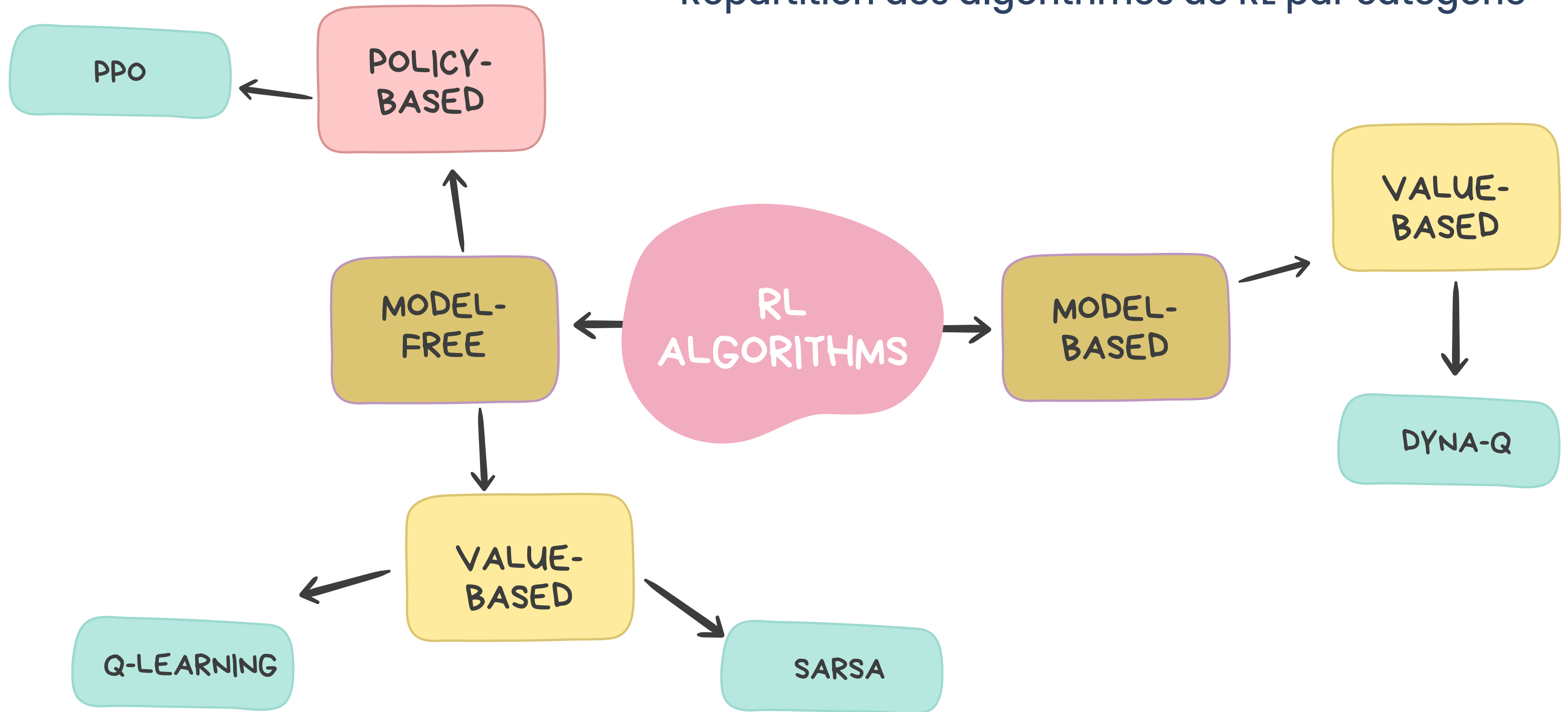


- Répartition des algorithmes de RL par categorie (Value-Based, Policy-Based, Model-Based).
- Expliquer les concepts clés associés : On-Policy vs Off-Policy, TD Learning vs Monte Carlo, Epsilon-Greedy.
- Découvrir 4 algorithmes fondamentaux et les plus utilisés : Q-Learning, SARSA, PPO, Dyna-Q.
- Comparer les approches et savoir quel algorithme choisir selon le problème.

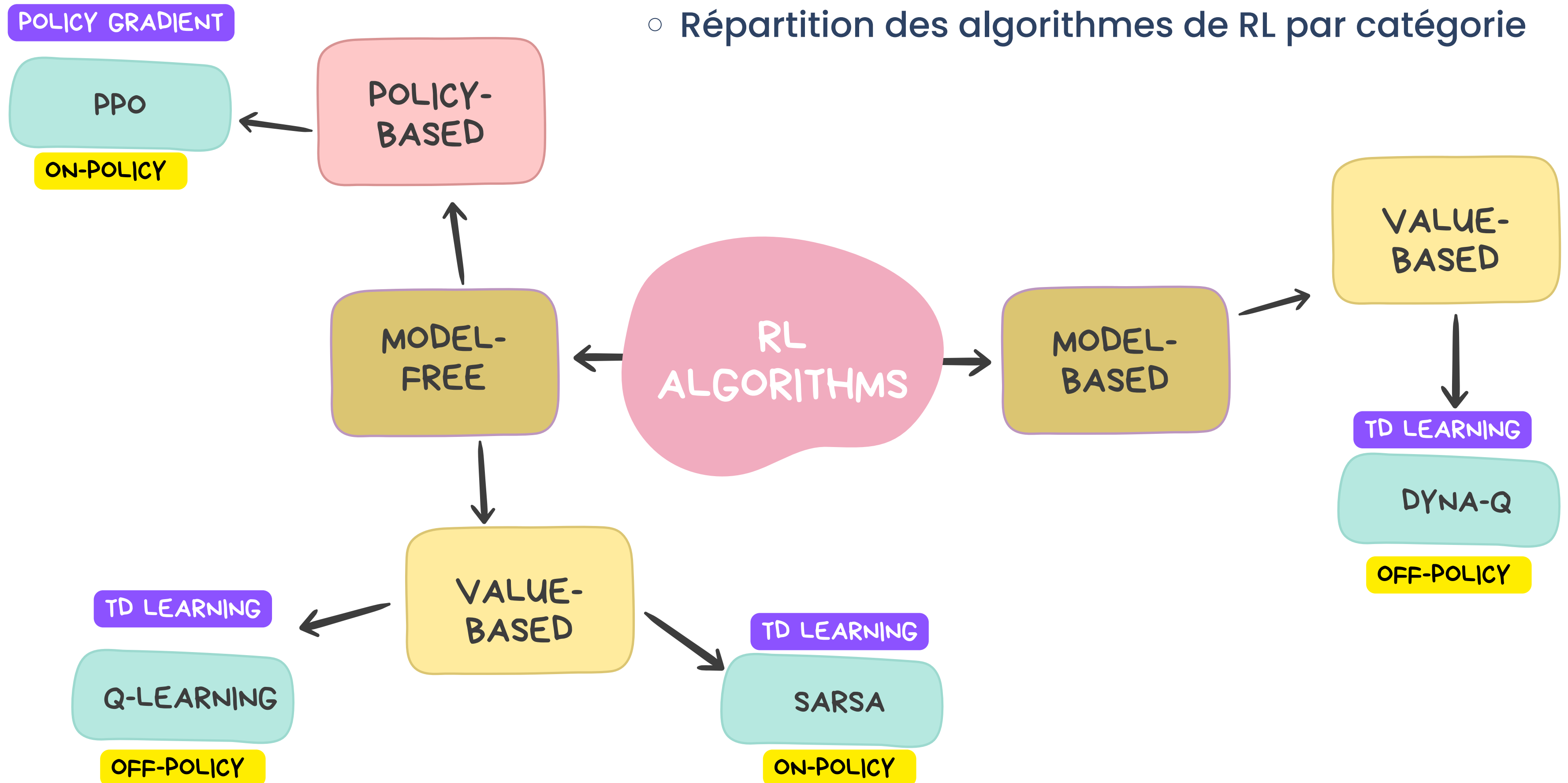
- Répartition des algorithmes de RL par catégorie



- Répartition des algorithmes de RL par catégorie



- Répartition des algorithmes de RL par catégorie



- Méthodes d'Apprentissage des Politiques : On-Policy vs Off-Policy

Un algorithme **On-Policy** apprend strictement à partir des **actions** qu'il exécute réellement avec sa **politique** actuelle $\pi(s)$.

L'apprentissage est aligné avec l'exploration, ce qui signifie que l'agent apprend en fonction des choix qu'il fait vraiment.

📌 **Formule (SARSA – On-Policy)**

$$Q(s, a) \leftarrow Q(s, a) + \alpha (R + \gamma Q(s', a') - Q(s, a))$$

- On utilise $Q(s', a')$, qui correspond à l'action réellement prise dans l'état suivant.

○ Méthodes d'Apprentissage des Politiques : On-Policy vs Off-Policy

Un algorithme **Off-Policy** apprend sans suivre strictement sa **politique** actuelle.

Il met à jour ses **valeurs** en fonction de **la meilleure action possible**, même si **l'agent** a pris une autre **action** en pratique.

📌 Formule (Q-Learning - Off-Policy)

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left(R + \gamma \max_{a'} Q(s', a') - Q(s, a) \right)$$

- On utilise $\max_{a'} Q(s', a')$ c'est-à-dire **la meilleure action** possible dans **l'état** suivant, même si **l'agent** a pris une **action** différente.

- Méthodes d'Apprentissage des Politiques : On-Policy vs Off-Policy

	On-Policy	Off-Policy
Mise à jour	Basée sur l'action réellement prise.	Basée sur la meilleure action possible.
Exploration	L'agent suit strictement sa politique actuelle.	L'agent peut explorer librement sans impacter l'apprentissage.
Stabilité	Plus stable, car il suit une seule stratégie.	Peut être plus instable, car il ne suit pas toujours sa propre politique.
Exemples	SARSA, PPO.	Q-Learning, DQN.

- Les techniques d'estimation de valeur dans RL

- **Monte Carlo : Mise à Jour à la Fin de l'Épisode**

Monte Carlo met à jour la **valeur** d'un **état** après la fin d'un **épisode** en utilisant la différence entre le retour G_t et la **valeur** estimée actuelle $V(S_t)$.

L'actualisation dépend de l'erreur entre la **valeur** estimée et le retour réel. C'est une méthode simple mais qui peut être lente pour des épisodes très longs.

$$V(S_t) \leftarrow V(S_t) + \alpha[G_t - V(S_t)]$$

- **G_t** : Retour calculé après avoir atteint la fin de l'épisode (où $R(t)$).
- **$G_t - V(S_t)$** : Erreur entre l'estimation actuelle et la réalité (qu'on appelle erreur de Monte Carlo).
- **α** : Taux d'apprentissage, qui contrôle à quelle vitesse on met à jour la valeur $V(S_t)$.

- Les techniques d'estimation de valeur dans RL

- **TD Learning : Mise à Jour après Chaque Étape**

Temporal Difference Learning met à jour les **valeurs** immédiatement après chaque **action**, sans attendre la fin de l'**épisode**, comme dans **Monte Carlo**.

$$V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$$

- **V(St)** : Valeur estimée actuelle de l'état St.
- **Rt+1** : Récompense immédiate obtenue après avoir pris une action dans St.
- **V(St+1)** : Valeur estimée de l'état suivant St+1.
- **γ** : Facteur d'actualisation ($0 < \gamma \leq 1$) qui contrôle l'importance des récompenses futures.
- **α** : Taux d'apprentissage (définit dans quelle mesure la valeur est mise à jour).

- Les techniques d'estimation de valeur dans RL

- **Policy Gradient (PG) – Une Sous-Approche de Monte Carlo**

- **L'approche Policy Gradient (PG)** est une méthode basée sur la politique en RL. Contrairement aux méthodes basées sur la valeur, PG apprend directement la politique optimale plutôt que d'estimer la valeur des états.
- **Policy Gradient** fait partie des méthodes **Monte Carlo**, car il attend la fin d'un épisode pour calculer les gradients et ajuster la politique.
- L'objectif de **(PG)** est de trouver une politique optimale, qui maximise la récompense cumulative attendue.

L'algorithme suit trois étapes principales :

- 1** Exécution de l'agent dans l'environnement pour collecter des expériences.
- 2** Calcul du gradient de la politique :
 - L'algorithme évalue les actions prises et ajuste les probabilités pour favoriser les bonnes actions.
- 3** Mise à jour des paramètres de la politique dans la direction qui maximise la récompense future.

- Les techniques d'estimation de valeur dans RL

	TD Learning	Monte Carlo	Policy Gradient
Mise à jour	Après chaque action	Après la fin de l'épisode	Mise à jour sur la politique π après la fin de l'épisode
Vitesse	Rapide, mise à jour progressive	Lent, attend la fin de l'épisode	Plus lent, mais s'adapte aux grandes actions
Précision	Approximatif	Plus précis (sauf variance élevée)	Peut être instable
Utilisation	Valeurs $Q(s,a)$	Récompenses cumulées G_t	Directement sur π
Exemples	Q-Learning, SARSA, DQN	REINFORCE, Monte Carlo Control	PPO, TRPO, A3C

○ La Stratégie epsilon ϵ -Greedy

L'epsilon-greedy strategy est une méthode permettant de gérer le compromis **exploration/exploitation** en RL. Elle est utilisée dans plusieurs algorithmes pour aider l'agent à explorer de nouvelles **actions** au début, puis à exploiter ses connaissances plus tard.

1 Au début de l'apprentissage ($\epsilon=1.0$) :

- L'agent choisit des actions au hasard \rightarrow 100% exploration.
- Objectif : découvrir comment l'environnement fonctionne.

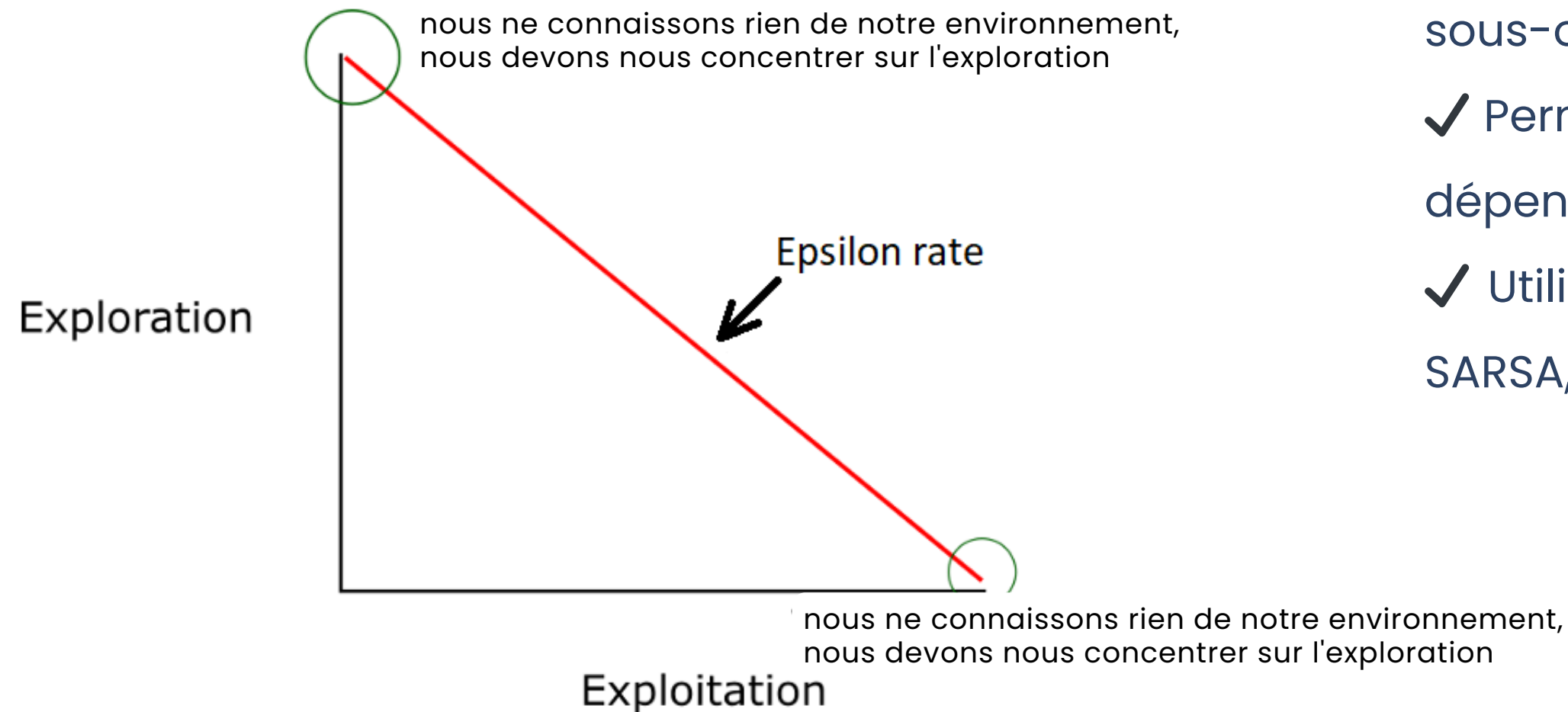
2 Progressivement, ϵ diminue :

- On réduit ϵ à chaque épisode pour favoriser l'exploitation des bonnes stratégies apprises.
- Exemple : Après plusieurs itérations, $\epsilon=0.1$, l'agent choisit encore 10% du temps au hasard, mais exploite 90% du temps.

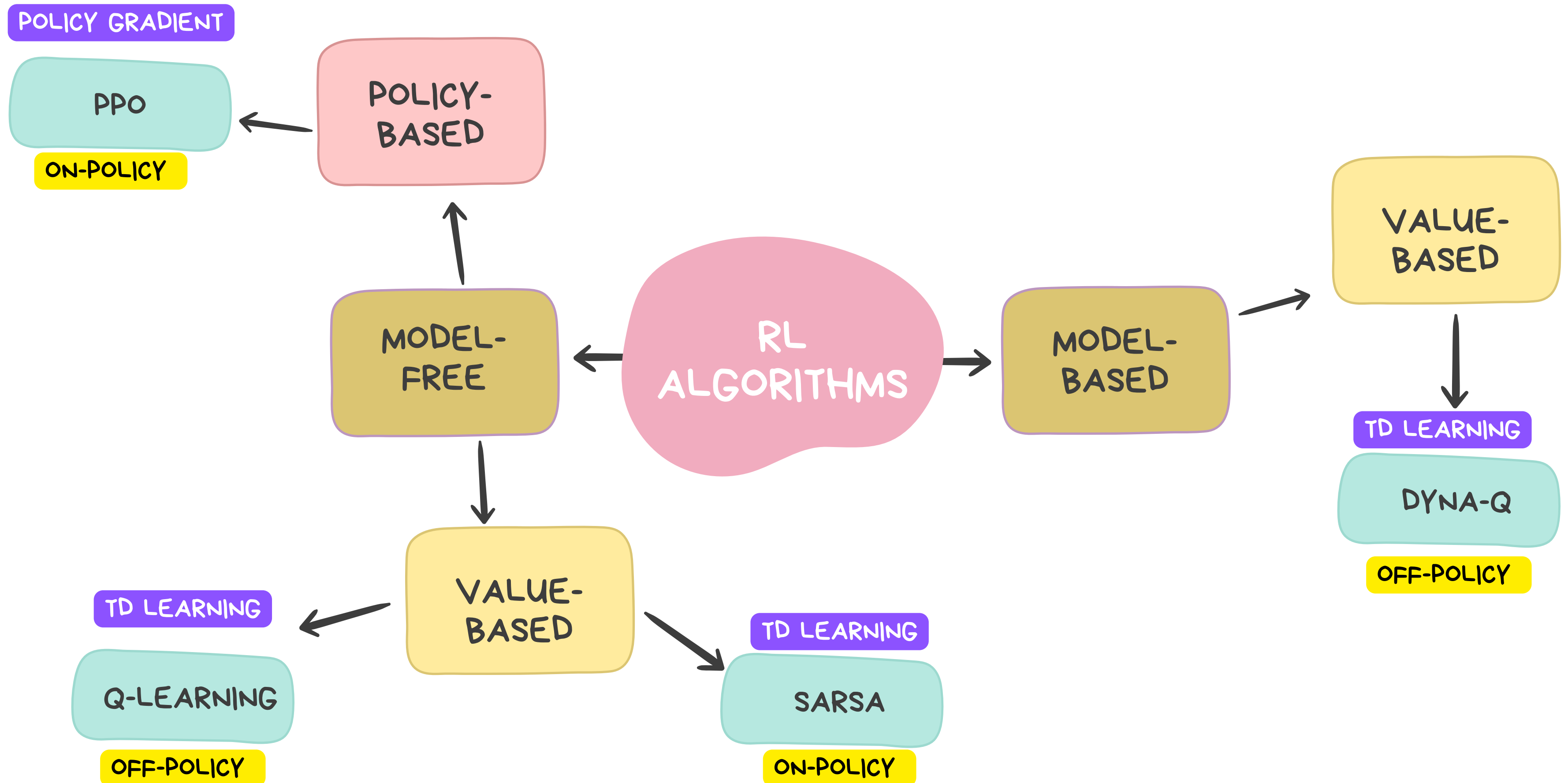
3 À la fin de l'apprentissage ($\epsilon \approx 0$) :

- L'agent sélectionne toujours l'action optimale trouvée.
- Plus besoin d'exploration car l'agent a appris la meilleure politique.

- La Stratégie epsilon ϵ -Greedy



- ✓ Évite que l'agent reste bloqué dans une stratégie sous-optimale.
- ✓ Permet d'apprendre progressivement sans être trop dépendant des premières expériences.
- ✓ Utilisé dans de nombreux algorithmes (Q-Learning, SARSA, DQN, etc.).



Model-Free

Value-Based

Off-Policy

1- Q-Learning

Q-Learning est un algorithme **d'apprentissage par renforcement** sans modèle (**model-free**) et hors-politique (**off-policy**) utilisé pour trouver la meilleure stratégie qu'un **agent** doit suivre afin de maximiser sa **récompense**.

Il entraîne une **fonction Q** qui estime la **valeur** d'effectuer une **action** donnée dans un **état** spécifique.



Le Q vient de la « **Qualité** » (la valeur) de cette **action** à cet **état**.

Model-Free











Value-Based

Off-Policy

1- Q-Learning



- La **Q-function** est codée par une **Q-table**, une table où chaque cellule correspond à une valeur de la paire **état-action**.
- La **Q-table** est initialisée, c'est-à-dire que toutes les valeurs sont = 0 au début. Ce tableau contient, pour chaque état et chaque action, les valeurs état-action correspondantes.

				
	0	0	0	0
	0	0	0	0
	0	0	0	0
	0	0	0	0
	0	0	0	0
	0	0	0	0






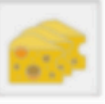
Model-Free

Value-Based

Off-Policy

1- Q-Learning

- Au début, notre **Q-table** est inutile puisqu'elle donne des valeurs arbitraires pour chaque paire état-action (la plupart du temps, nous initialisons la Q-table à 0). Au fur et à mesure que l'agent explore l'environnement et que nous mettons à jour la Q-table, celle-ci nous donnera une approximation de plus en plus précise de la politique optimale.

	←	→	↑	↓
	0	0	0	0
	0	0	0	0
	0	0	0	0
	0	0	0	0
	0	0	0	0
	0	0	0	0



	←	→	↑	↓
	0	10.8	0	0
	0	9.9	0	-10
	0	0	0	10
	0	-10	0	0
	0	0	0	0
	0	0	0	0

Model-Free

Value-Based

Off-Policy

1- Q-Learning

L'essentiel du Q-Learning est :

- Entraîne une **Q-function** (une fonction action-valeur), qui en interne est une **Q-table** qui contient toutes les valeurs de la paire état-action.
- Compte tenu d'un état et d'une action, notre **Q-function** recherchera la valeur correspondante dans sa **Q-table**.
- Lorsque l'entraînement est terminé, nous avons une **Q-function** optimale, ce qui signifie que nous avons une table **Q optimale**.
- Et si nous avons une fonction Q optimale, nous avons une politique optimale puisque nous connaissons la meilleure action à entreprendre pour chaque état.

$$\pi^*(s) = \arg \max_a Q^*(s, a)$$

La recherche d'une fonction de valeur optimale conduit à une politique optimale.

Model-Free

Value-Based

Off-Policy

1- Q-Learning

Pseudo-code de Q-Learning:

1. Initialiser **Q-table** avec des valeurs arbitraires (ex : 0)
2. Pour chaque **épisode** :
 - Initialiser **l'état** s
 - Tant que **l'épisode** n'est pas terminé :
 - I. Sélectionner une action a avec la stratégie **epsilon-greedy**
 - II. Exécuter **l'action** a et observer la **récompense** R et le nouvel **état** s'
 - III. Mettre à jour la **Q-table** : $Q(s, a) = Q(s, a) + \alpha * [R + \gamma * \max_{a'}(Q(s', a')) - Q(s, a)]$
 - IV. Mettre $s = s'$ - Fin de **l'épisode**
3. Répéter jusqu'à convergence (les valeurs de Q ne changent plus significativement)

Model-Free

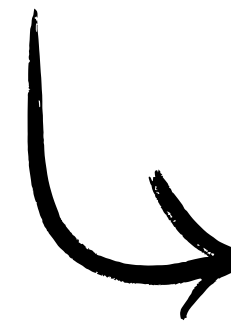
Value-Based

ON-Policy

2- SARSA (State-Action-Reward-State-Action)

SARSA (State-Action-Reward-State-Action) est un algorithme d'apprentissage par renforcement **model-free** et **on-policy**, qui apprend une politique optimale en mettant à jour sa Q-Table en fonction de la séquence d'état-action-récompense-état-action. Contrairement au **Q-Learning**, **SARSA** suit toujours **la politique actuelle** lorsqu'il met à jour ses valeurs Q.

$$Q(s, a) = Q(s, a) + \alpha [R + \gamma Q(s', a') - Q(s, a)]$$



SARSA met à jour ses valeurs en fonction de l'action a' réellement prise suivant la politique actuelle.

Model-Free

Value-Based

ON-Policy

2- SARSA (State-Action-Reward-State-Action)

Pseudo-code de SARSA:

1. Initialiser **Q-table** avec des valeurs arbitraires (ex : 0)
2. Pour chaque **épisode** :
 - Initialiser **l'état** s
 - Tant que **l'épisode** n'est pas terminé :
 - I. Sélectionner une action a avec la stratégie **epsilon-greedy**
 - II. Exécuter **l'action** a et observer la **récompense** R et le nouvel **état** s'
 - III. Mettre à jour la **Q-table** : $Q(s, a) = Q(s, a) + \alpha * [R + \gamma * Q(s', a) - Q(s, a)]$
 - IV. Mettre $s = s'$ - Fin de **l'épisode**
3. Répéter jusqu'à convergence (les valeurs de Q ne changent plus significativement)

Model-Free

Value-Based

ON-Policy

2- SARSA (State-Action-Reward-State-Action)

Critère	SARSA (On-Policy)	Q-Learning (Off-Policy)
Type d'apprentissage	On-policy (suit la politique en cours)	Off-policy (utilise la meilleure action possible)
Mise à jour des valeurs Q	Utilise l'action réellement prise a'	Utilise l'action optimale $\max Q(s',a)$
Exploration vs Exploitation	Plus conservateur (réduit le risque)	Plus agressif (prend des décisions optimales plus rapidement)
Exemple d'application	Navigation d'un robot avec incertitudes	Jeux où l'agent doit apprendre la meilleure stratégie

- SARSA est plus prudent et s'adapte mieux aux environnements instables.
- Q-Learning apprend plus vite mais peut être risqué dans des environnements changeants.

Model-Free

Policy-Based

ON-Policy

3- PPO (Proximal Policy Optimization)

Proximal Policy Optimization (PPO) est un algorithme **policy-based**, développé par **OpenAI**, permettant d'optimiser directement la politique d'un agent dans un environnement.

PPO est une avancée majeure qui remplace des méthodes plus complexes comme **Trust Region Policy Optimization (TRPO)** en offrant une meilleure stabilité et une plus grande facilité d'implémentation.

PPO devenu l'algorithme standard chez **OpenAI** parce qu'il est plus simple et plus robuste.

📌 Pourquoi PPO a-t-il été créé ?

- ✓ Améliorer la stabilité des algorithmes policy-based
- ✓ Éviter les mises à jour trop brutales qui déstabilisent l'apprentissage
- ✓ Faciliter l'entraînement des agents dans des environnements complexes

📌 Principaux domaines d'application :

- ◆ Jeux vidéo (ex: agents intelligents dans des simulations)
- ◆ Robotique (contrôle précis des mouvements des robots)
- ◆ Modèles de langage (ex: ChatGPT)

Model-Free

Policy-Based

ON-Policy

3- PPO (Proximal Policy Optimization)

L'algorithme maximise une fonction de perte clippée pour éviter des mises à jour trop importantes :

$$L^{CLIP}(\theta) = \mathbb{E} [\min(r_t(\theta)A_t, \text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon)A_t)]$$

avec :

- **$r_t(\theta)$** : le rapport entre l'ancienne et la nouvelle politique $r_t(\theta) = \frac{\pi_{\theta}(a_t|s_t)}{\pi_{\theta_{\text{ancien}}}(a_t|s_t)}$
 - Si r_t est proche de 1 ($0.8 < r_t < 1.2$) → peu de changement dans la politique.
 - Si r_t est trop élevé → la nouvelle politique diffère trop de l'ancienne.
- **$\text{clip}(\dots, 1-\epsilon, 1+\epsilon)$** : Si r_t dépasse un certain seuil $[1-\epsilon, 1+\epsilon]$. la mise à jour est bloquée, cela évite que l'agent oublie ce qu'il a appris trop rapidement.
- **A_t** : l'avantage estimé d'une action $A_t = Q(s_t, a_t) - V(s_t)$
 - Si A_t est positif → l'action était meilleure que prévu.
 - Si A_t est négatif → l'action était moins bonne que prévu.
- **ϵ** : une marge de tolérance (ex: $\epsilon=0.2$) qui limite les mises à jour excessives.

Model-Free

Policy-Based

ON-Policy

3- PPO (Proximal Policy Optimization)

TRPO impose une contrainte stricte sur la mise à jour de **la politique** en utilisant la **KL-Divergence**.

- Cela signifie que pour chaque **mise à jour**, il faut résoudre une **équation d'optimisation** sous contrainte, ce qui est un problème mathématique compliqué.
- Ce type d'optimisation s'appelle "optimisation quadratique" car elle implique des calculs avancés pour trouver la solution de l'équation.
- **Conséquence : TRPO** est lent, difficile à régler et très exigeant en calculs.

$$D_{KL}(P||Q) = \sum P(x) \log \frac{P(x)}{Q(x)} \quad \text{L'équation de KL-Divergence}$$

- $P(x)$: La distribution de l'ancienne politique.
- $Q(x)$: La distribution de la nouvelle politique.
- Si $D(kl)$ est grand, cela signifie que la nouvelle politique est très différente de l'ancienne.
- Si $D(kl)$ est petit, cela signifie que la nouvelle politique est similaire à l'ancienne.

Model-Free

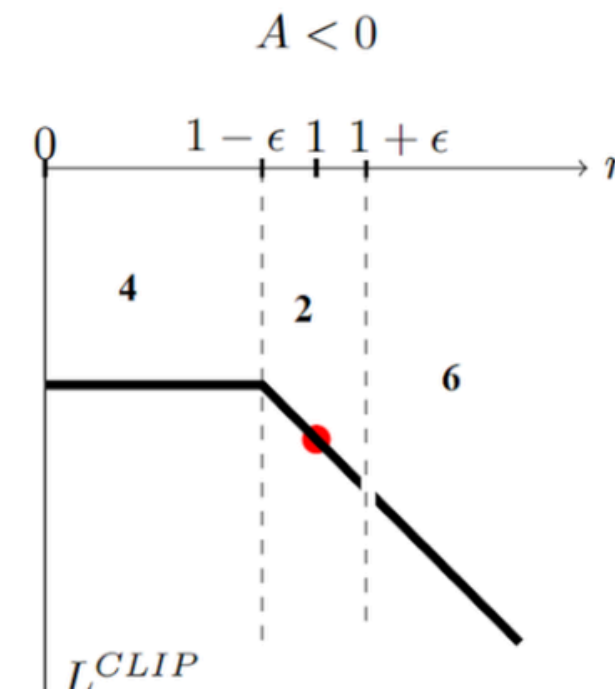
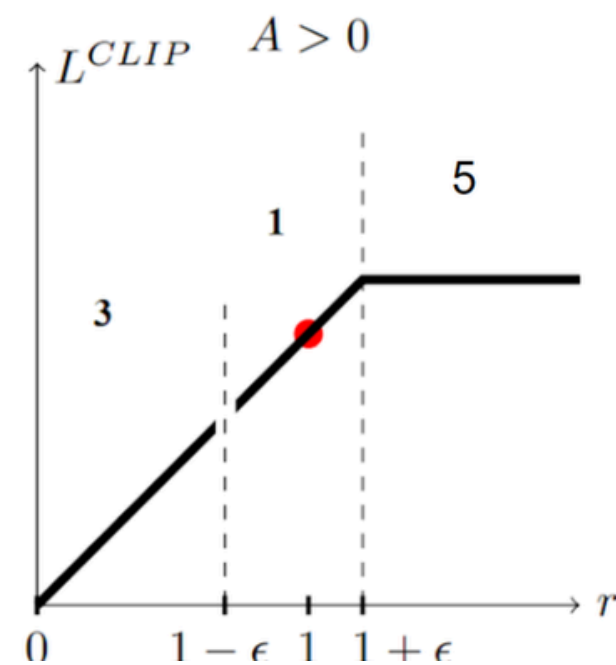
Policy-Based

ON-Policy

3- PPO (Proximal Policy Optimization)

- **PPO** corrige le problème d'instabilité de **TRPO** en introduisant **une fonction de perte clippée** qui limite les mises à jour trop importantes de la politique.

$$\text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon)$$



- Si r_t est dans l'intervalle $[1-\epsilon, 1+\epsilon]$, l'apprentissage continue normalement.
- Si r_t est hors de cette plage, la mise à jour est limitée, empêchant une modification de la politique.

Model-Free

Policy-Based

ON-Policy

3- PPO (Proximal Policy Optimization)

Pseudo-code de PPO:

- 1 Initialisation** : Définition des paramètres de la politique π et de la fonction de valeur V .
- 2 Collecte des expériences** : L'agent explore l'environnement et stocke des transitions (s,a,r,s') sur plusieurs épisodes pour l'apprentissage.
- 3 Calcul des avantages** : Utilisation de Generalized Advantage Estimation (GAE) pour mesurer la qualité des actions.
- 4 Mise à jour de la politique** : Optimisation avec la perte CLIP pour stabiliser l'apprentissage.
- 5 Mise à jour de la fonction de valeur** : Ajustement de $V(s)$ pour mieux estimer les valeurs des états.
- 6 Répétition** : Mise à jour de la politique π et reprise du processus jusqu'à convergence.

Model-Based

Value-Based

OFF-Policy

4- DYNA-Q

Dyna-Q est une extension du **Q-Learning**, qui combine apprentissage direct et modélisation de l'environnement pour améliorer l'efficacité de l'apprentissage. Contrairement aux algorithmes **model-free**, **Dyna-Q** est **model-based**, ce qui signifie qu'il essaie de reconstruire une version approximative de **l'environnement** à partir des **interactions passées**.

- **Pourquoi Dyna-Q ?**

- Améliore la vitesse d'apprentissage en combinant expériences réelles et simulées.
- Utilise un modèle interne pour générer de nouvelles expériences fictives.
- Diminue la dépendance à l'exploration physique, utile lorsque les interactions sont coûteuses.

- **Exemple d'application :**

- Dans un jeu vidéo, au lieu d'attendre qu'un joueur rencontre un ennemi 100 fois pour apprendre la meilleure stratégie, Dyna-Q peut utiliser un modèle interne pour simuler des rencontres et accélérer l'apprentissage.

Model-Based

Value-Based

OFF-Policy

4- DYNA-Q

Dyna-Q repose sur la mise à jour de la **Q-Table** comme dans le **Q-Learning** :

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left[R + \gamma \max_{a'} Q(s', a') - Q(s, a) \right]$$

Mais ajoute un modèle interne $P(s' \mid s, a)$ et $R(s, a)$, qui permettent de simuler des transitions.

s	a	s' (prédit)	R (prévu)
1	Gauche	2	-1
2	Droite	3	5
3	Haut	1	0

- Dyna-Q fait la mises à jour en deux types :
 - Mise à jour avec des expériences réelles (identique à Q-Learning).
 - Mise à jour avec des expériences simulées à partir du modèle interne

- Chaque fois que **l'agent** interagit avec **l'environnement**, il ajoute ces informations à sa mémoire.
- Puis, plutôt que d'exécuter de nouvelles actions dans l'environnement, il peut réutiliser ces expériences en les simulant mentalement.

Model-Based

Value-Based

OFF-Policy

4- DYNA-Q

Critère	Model-Free (Q-Learning, SARSA)	Model-Based (Dyna-Q)
Interaction avec l'environnement	Nécessaire à chaque mise à jour.	Peut utiliser des expériences simulées.
Apprentissage	L'agent apprend uniquement en interagissant avec l'environnement. Il met à jour sa Q-Table à partir des expériences directes.	L'agent construit un modèle de l'environnement et l'utilise pour simuler des expériences fictives.
Adaptabilité	Réagit aux nouvelles situations en explorant.	Peut anticiper grâce à son modèle.
Retour en arrière ?	il ne se base que sur ce qu'il vit en direct, sans mémoire interne de l'environnement.	il retourne en arrière dans sa propre mémoire (modèle) pour générer des expériences et accélérer l'apprentissage.

Model-Based

Value-Based

OFF-Policy

4- DYNA-Q

Processus de Dyna-Q :

1. Prendre une action a dans l'environnement réel, observer s', R .
2. Mettre à jour la Q-Table avec cette expérience réelle.
3. Ajouter cette transition à la mémoire (s, a, s', R) .
4. Générer et jouer des expériences simulées à partir de cette mémoire :
5. Choisir un (s, a) passé.
6. Prédire s' et R avec le modèle.
7. Mettre à jour $Q(s, a)$ avec ces valeurs simulées.
8. Répéter ce processus jusqu'à convergence.