
# L'ÉVOLUTION DE L'ACTIVITÉ DE PERFORMANCE

# CE QUE NOUS ALLONS PRÉSENTER

- Comment garder la performance applicative au centre de l’expérience client ?
- Pourquoi la stratégie de test de performance doit évoluer ?
- Quelles sont les solutions ?
- Quelles nouvelles problématique cela soulève-t-il ?

# ÉVOLUTION DE L'ARCHITECTURE DES SI

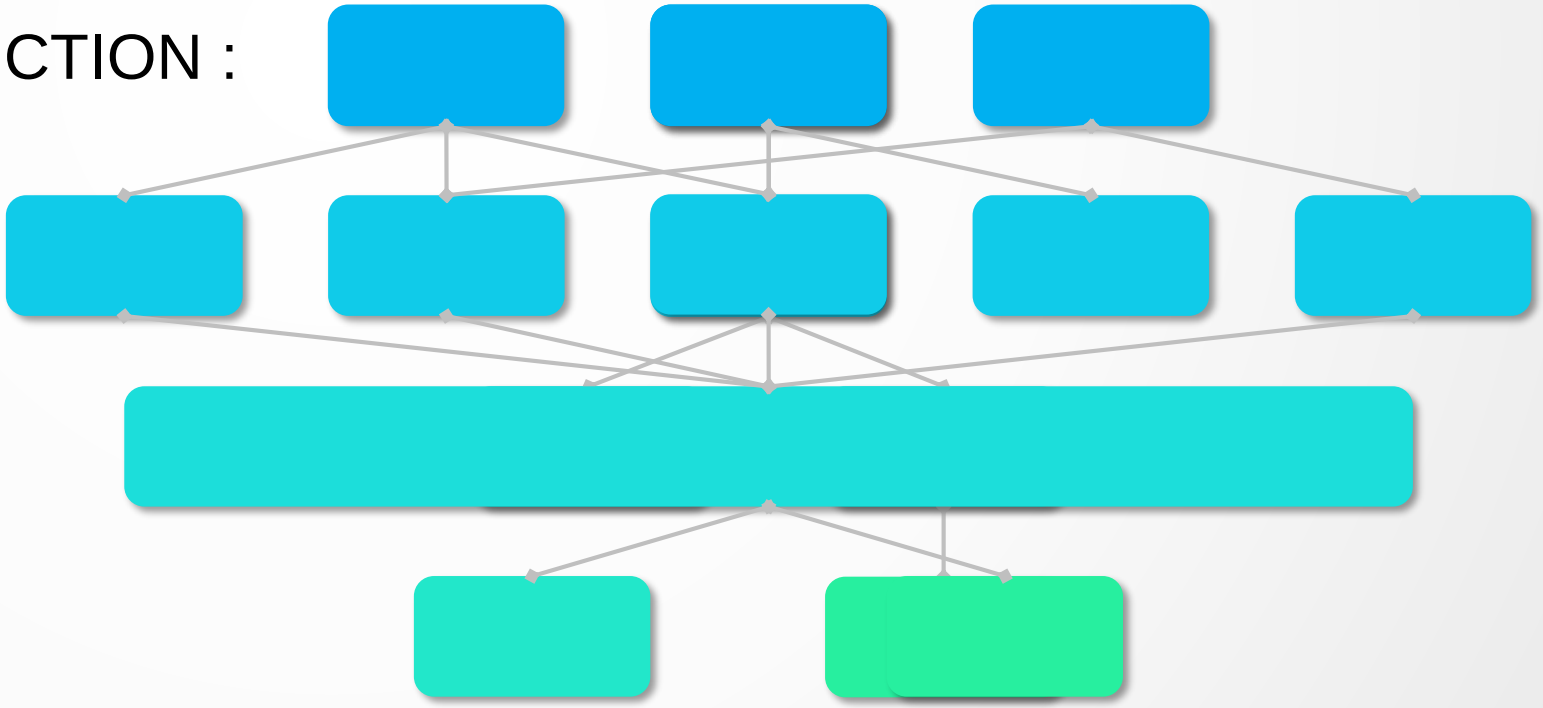
## MODULAIRE – DISTRIBUÉ - MULTICANAL

(monolithique-centralisé)

ARCHITECTURE DE COMPOSANTS :

ARCHITECTURE PAR FONCTION :

- Compte client
- Compte vendeur
- Moteur de recherche
- Panier
- Inventaire
- Paiement



# ÉVOLUTION DU MODÈLE DE DÉVELOPPEMENT

RECUEIL DU BESOIN

ADOPTION UTILISATEURS

TMA

ANALYSE

ARCHITECTURE ET SPÉCIFICATIONS

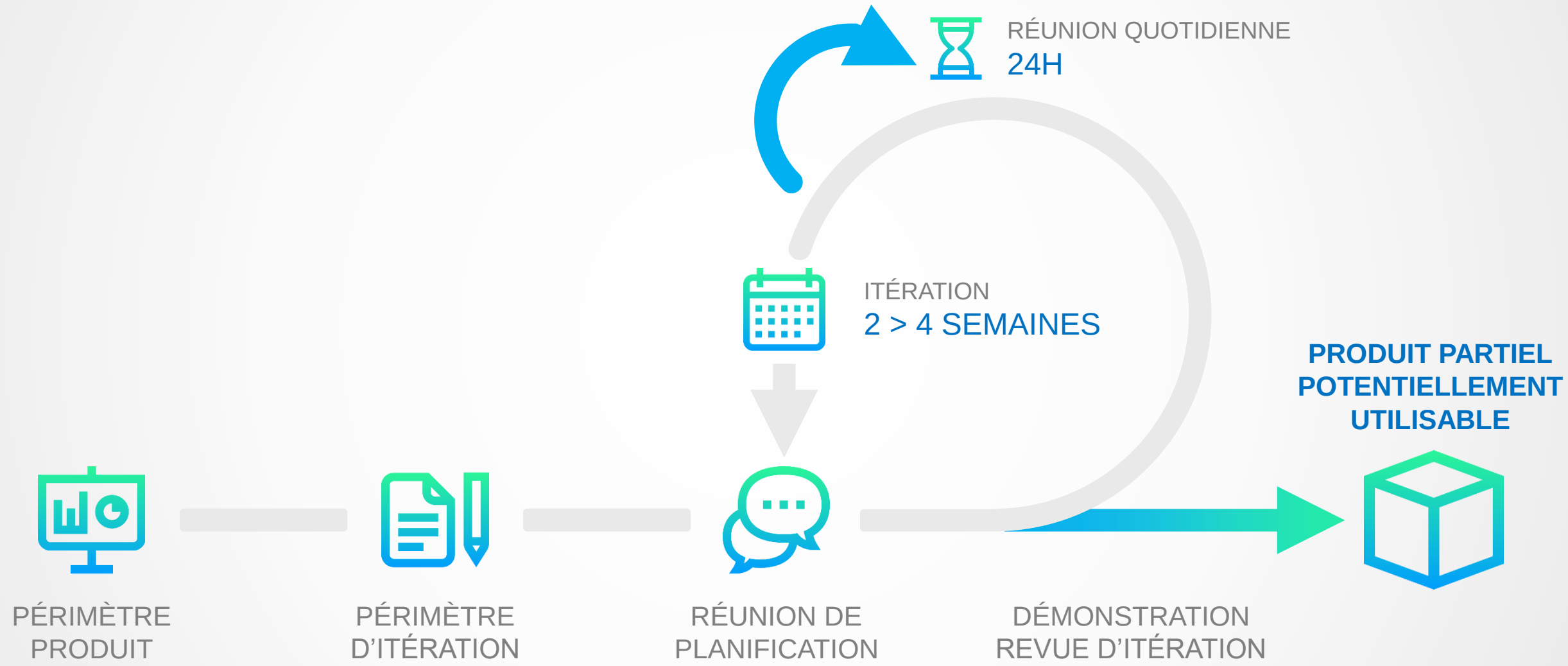
CONCEPTION

DÉVELOPPEMENT

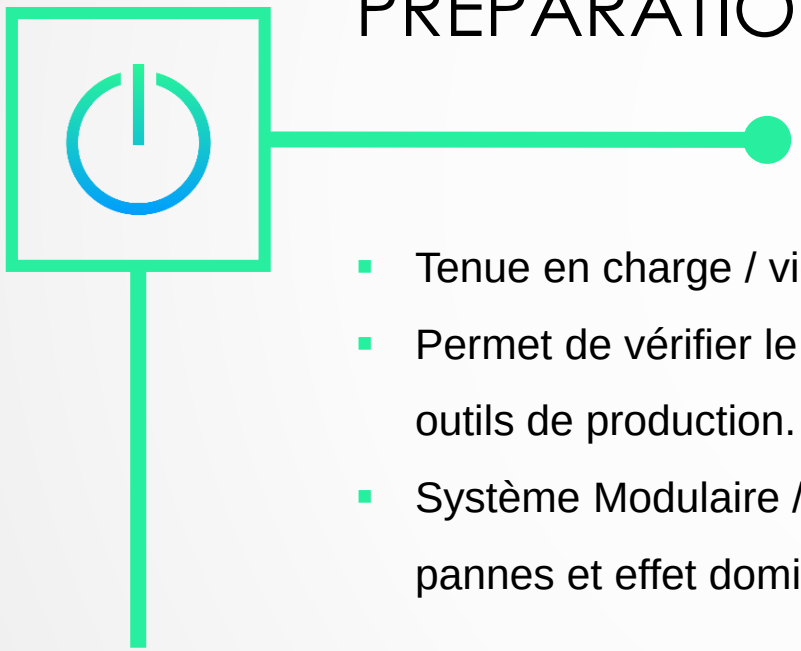
MISE EN PRODUCTION

INTÉGRATION

PLAN DE TEST ET RECETTE


RÉUNION QUOTIDIENNE
24H
ITÉRATION
2 > 4 SEMAINES
PRODUIT PARTIEL POTENTIELLEMENT UTILISABLE
PÉRIMÈTRE PRODUIT
PÉRIMÈTRE D'ITÉRATION
RÉUNION DE PLANIFICATION
DÉMONSTRATION REVUE D'ITÉRATION

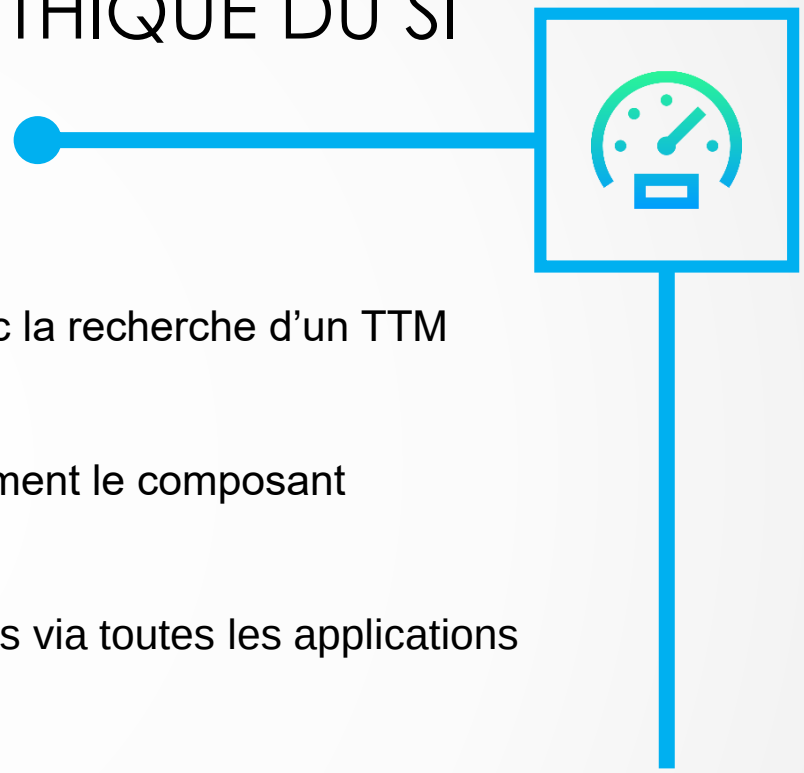
# POSITIONNEMENT DES TESTS DE PERFORMANCE BOUT-EN-BOUT



## PRÉPARATION DU PASSAGE EN PRODUCTION

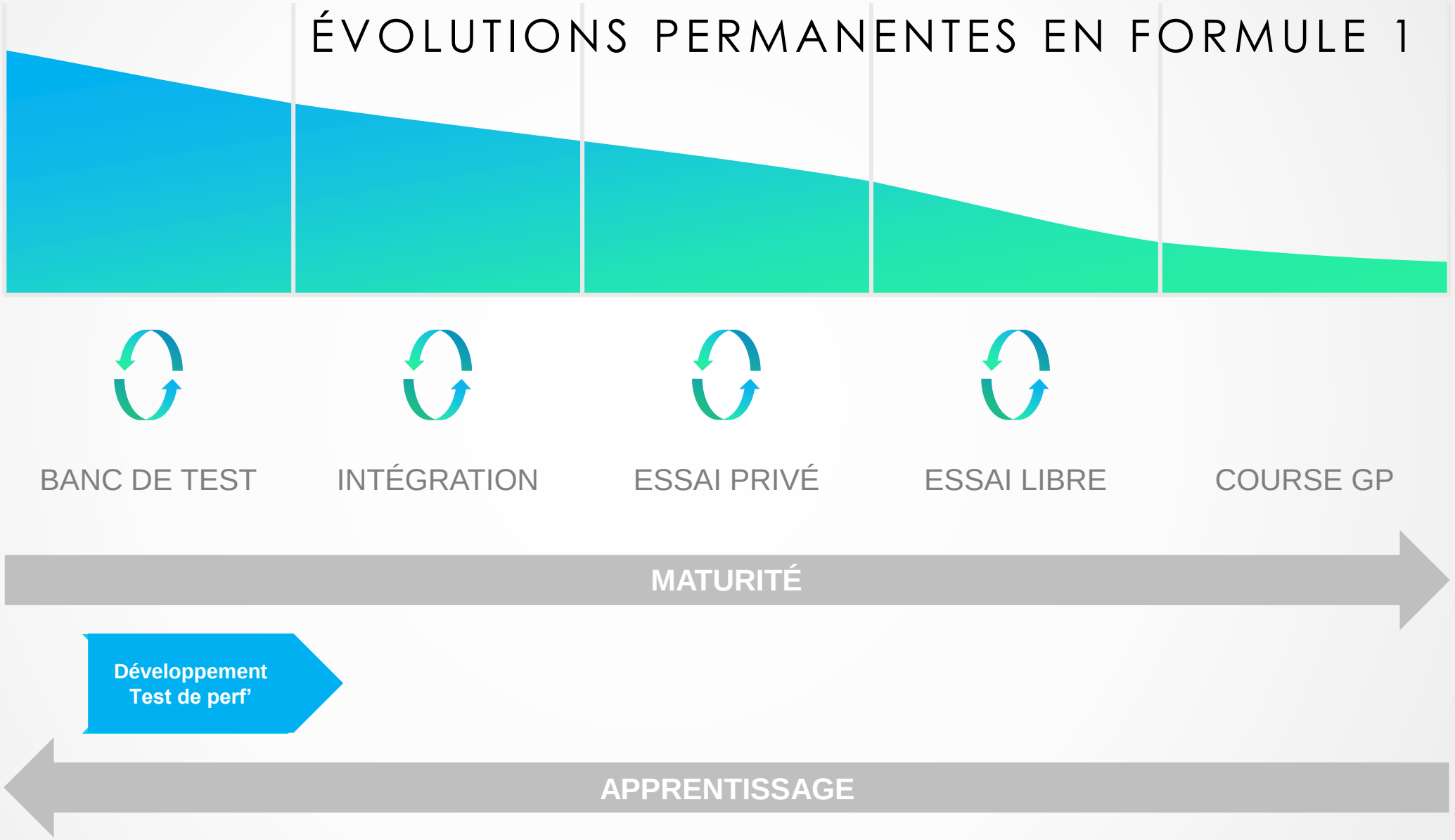
- Tenue en charge / vieillissement / pic de charge / impact d'un batch.
- Permet de vérifier le monitoring / suivi / alerting de l'application avec les outils de production.
- Système Modulaire / Distribué : Répartition de charge, Tolérance aux pannes et effet domino.

# LES LIMITES D'UNE VUE MONOLITHIQUE DU SI



- Impossibilité de tester le code dormant.
- Effet tunnel (découverte séquentielle) incompatible avec la recherche d'un TTM le plus court possible.
- Difficulté de comparer des résultats et d'identifier facilement le composant « défaillant » si le SI change très vite.
- Difficulté de tester des composants/modules transverses via toutes les applications en même temps.

# ÉVOLUTIONS PERMANENTES EN FORMULE 1



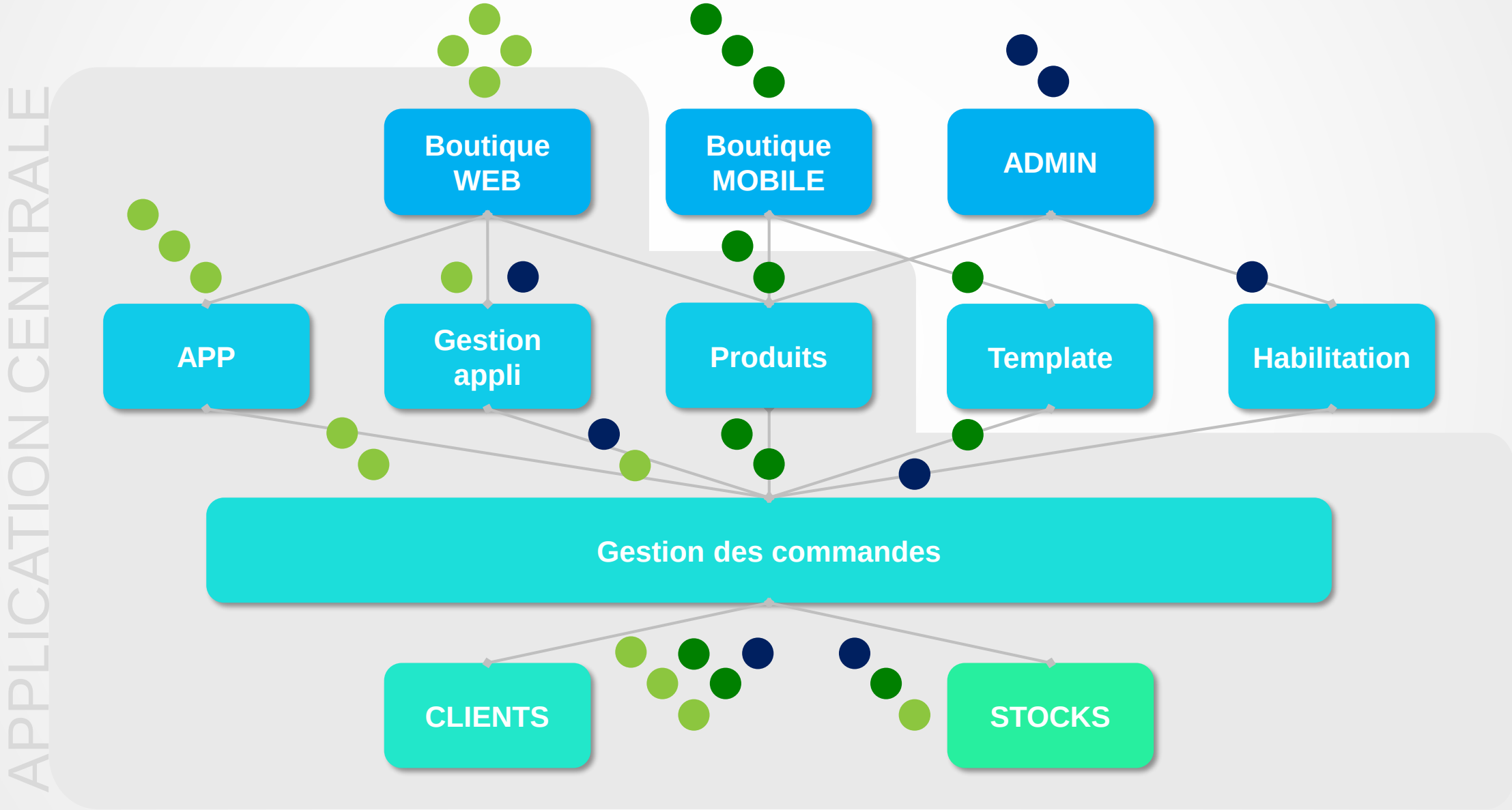
# GÉRER LES PÉRIMÈTRES ET LES RISQUES

- Comprendre le périmètre adressé par chaque type de test :
  - Test de performance en continu des composants
    - Test du « moteur », et suivi de la nature et du nombre d'interactions en amont
    - Jeu de données restreint (surtout avec l'utilisation de bouchon)
  - Test de performance bout-en-bout
    - Tests de la cohérence générale du SI en condition réelle, avec des données réelles
    - Problème de performance traité séquentiellement (un bottleneck peut en cacher beaucoup d'autres)
- Accepter de traiter des sujets « à risque » dans des environnements dédiés pour permettre la parallélisassions des sujets

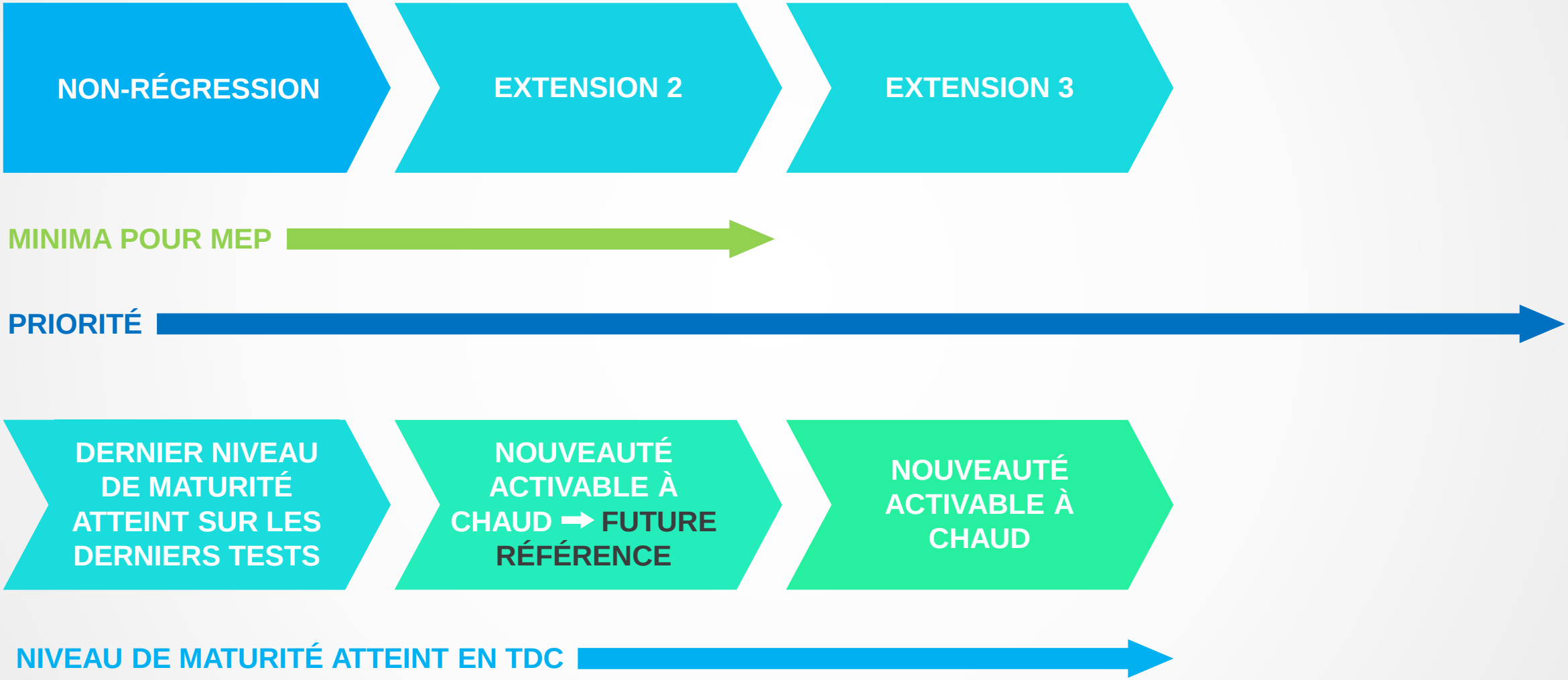
# LA MODÉLISATION D'USAGE ET LES OBJECTIFS

- Qu'est ce que la modélisation d'usage ?
- Comment identifier les chiffres permettant cette modélisation ?
- Qu'est ce qui a changé dans les nouvelles architectures ?
- Comment garantir que notre modèle de charge soit toujours bon ?
- Comment fixer des objectifs de temps de réponse par module ?

# LA MODÉLISATION D'USAGE – SCHÉMATISÉE



# L'APPROCHE ITÉRATIVE



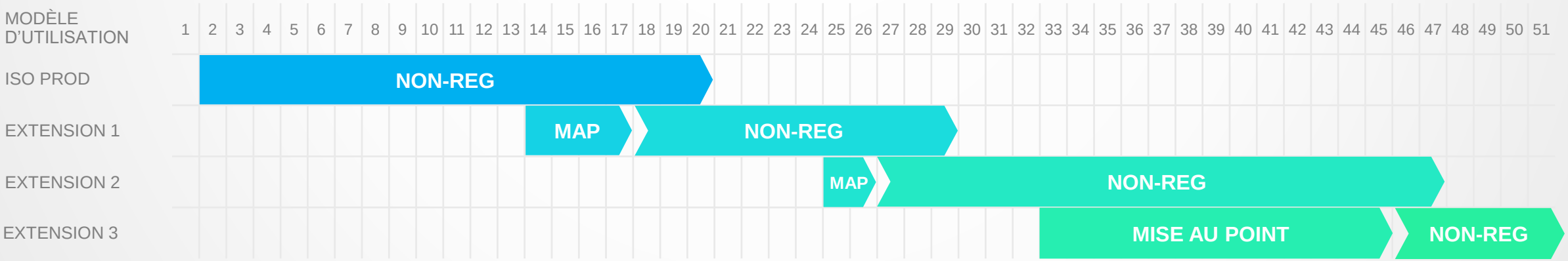
# TESTS DE COMPOSANTS

## OBJECTIFS

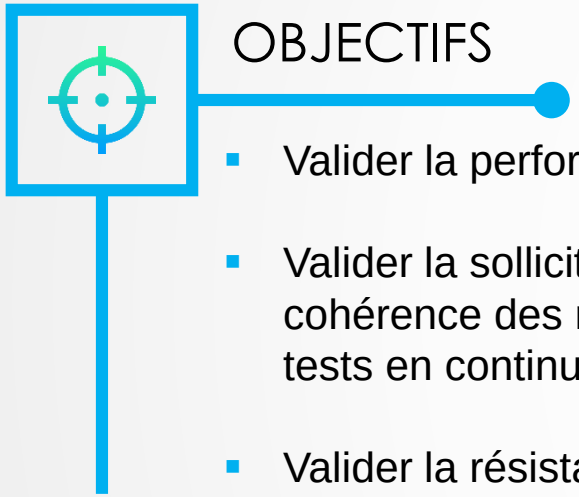
- Avoir un module testé au plus tôt
- Anticiper les tests sur le code « dormant »
  non utilisable sur l'application bout-en-bout
- Ne pas alourdir les non-reg et accepter les ruptures dans l'historisation des résultats

## STRATÉGIE

- 1 modèle avec un bon niveau de maturité, sur lequel est effectué la non-reg
- 1 modèle avec extension de périmètre



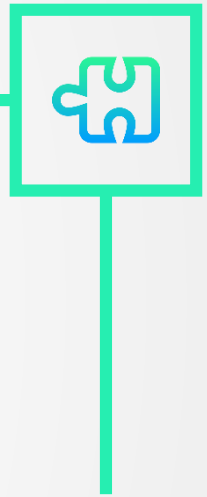
# TESTS DE PERFORMANCE BOUT-EN-BOUT



## OBJECTIFS

- Valider la performance en bout-en-bout
- Valider la sollicitation des modules et la cohérence des modèles de charge des tests en continu
- Valider la résistance aux pannes
- Valider la supervision applicative

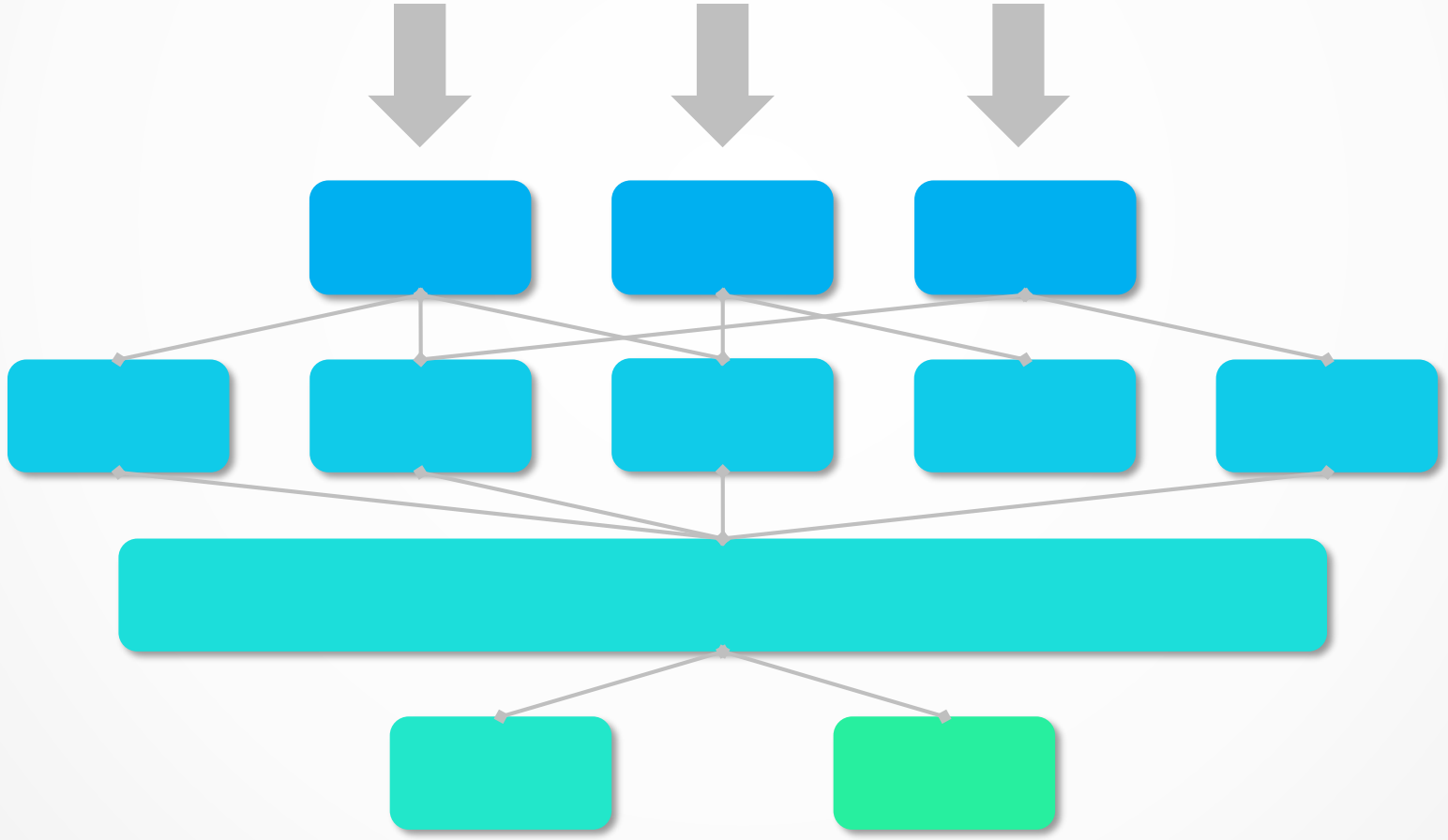
## STRATÉGIE



- Modèle itératif « classique » à la demande adapté aux différents backlogs produits / backlog sprints

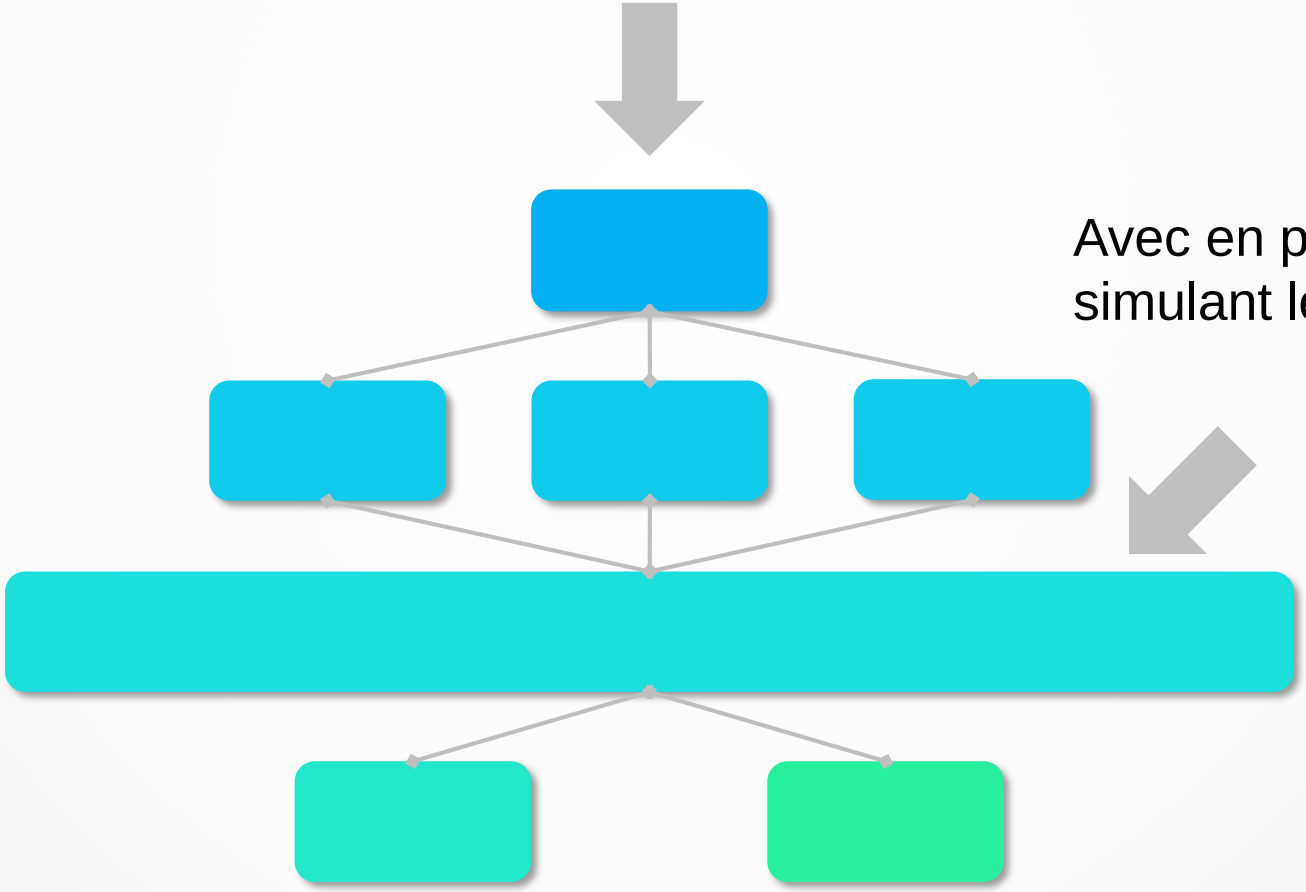
# LA PERFORMANCE BOUT-EN-BOUT
## COMMENT GÉRER SUR LES MODULES TRANSVERSES ?

Tester en parallèle avec toutes les applications sollicitant ce composant transverse



# LA PERFORMANCE BOUT-EN-BOUT
## COMMENT GÉRER SUR LES MODULES TRANSVERSES ?

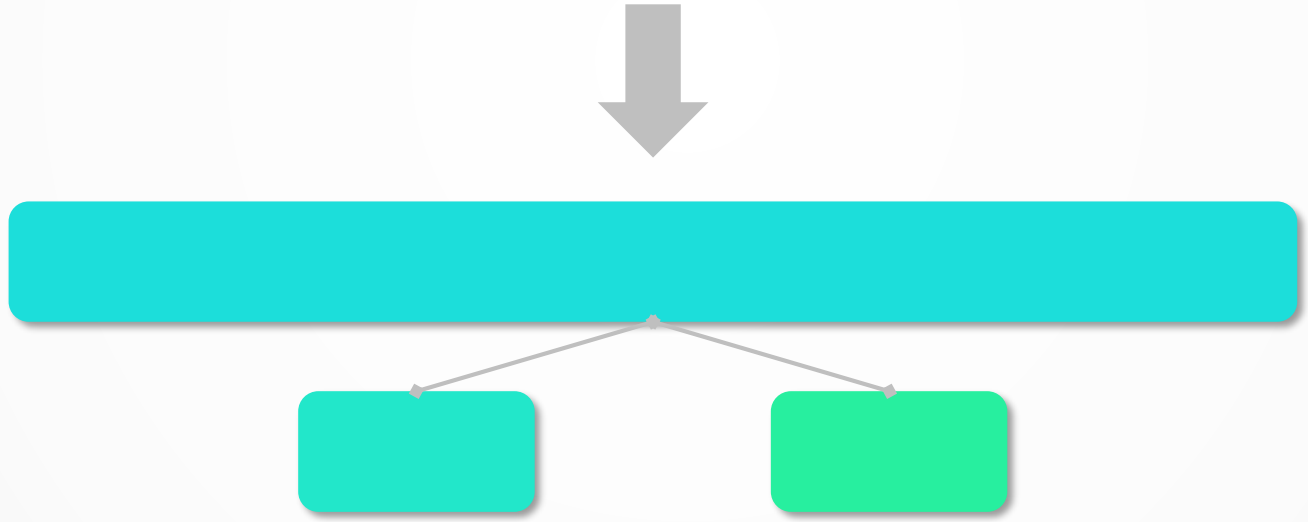
Tester avec l'application nécessitant des évolutions en bout-en-bout



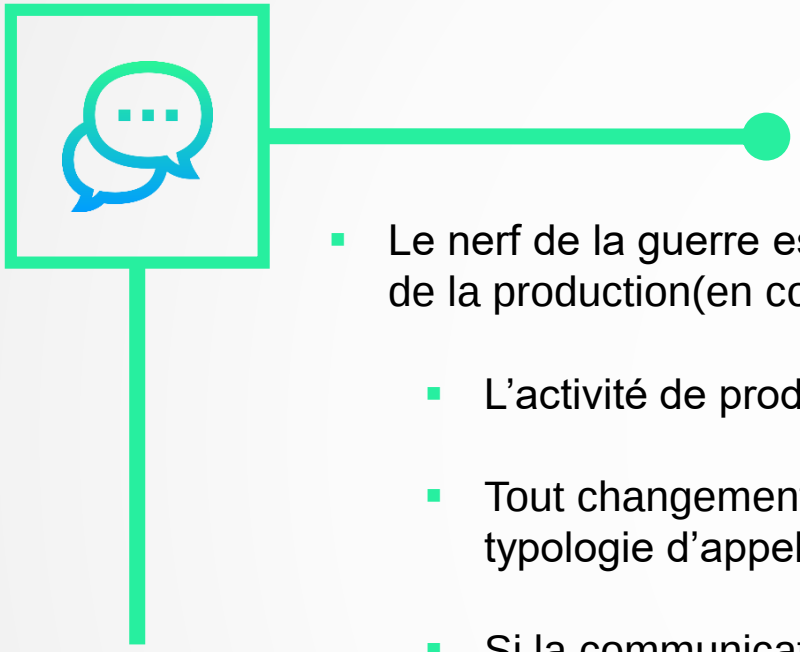
Avec en parallèle une injection simulant les flux des autres clients

# LA PERFORMANCE BOUT-EN-BOUT
## COMMENT GÉRER SUR LES MODULES TRANSVERSES ?

Tester les applications en bout-en-bout séparément et tester les briques transverses lors de test simulant tous ses clients

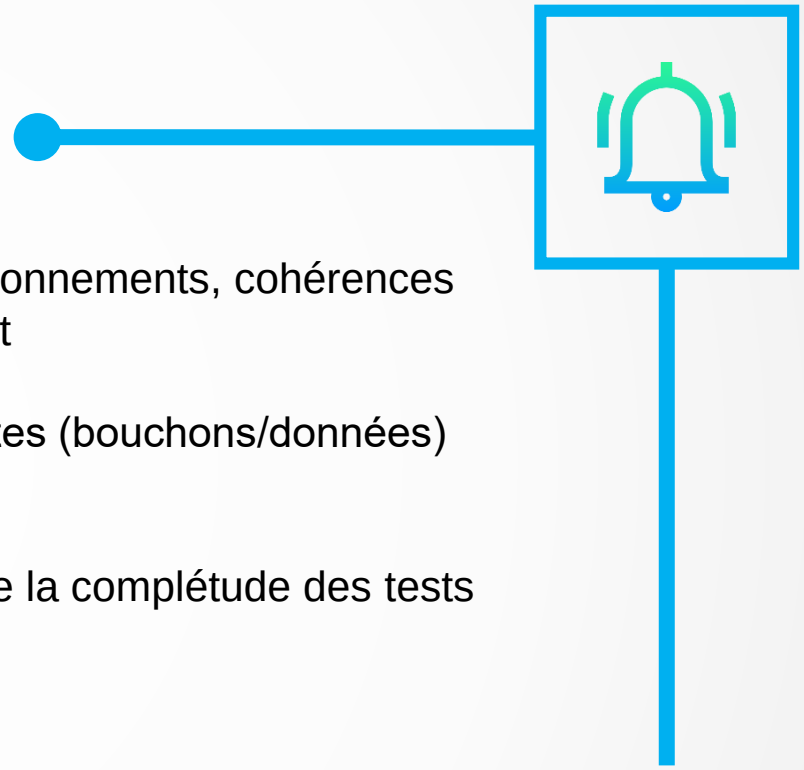


# LA COMMUNICATION ET LA PRODUCTION



- Le nerf de la guerre est d'assurer la cohérence entre les différents tests et l'activité de la production(en cours et prévisionnelle) :
  - L'activité de production évolue, le modèle d'usage doit donc évoluer
  - Tout changement en amont ou en aval (évolution du comportement, nombre et typologie d'appel) constaté dans un environnement doit être communiqué
  - Si la communication n'est pas transparente, les tests ne seront plus pertinent et cohérent entre eux.
- Un bon moyen d'échanger est d'utiliser des moyens communs (dashboard /monitoring applicatifs).

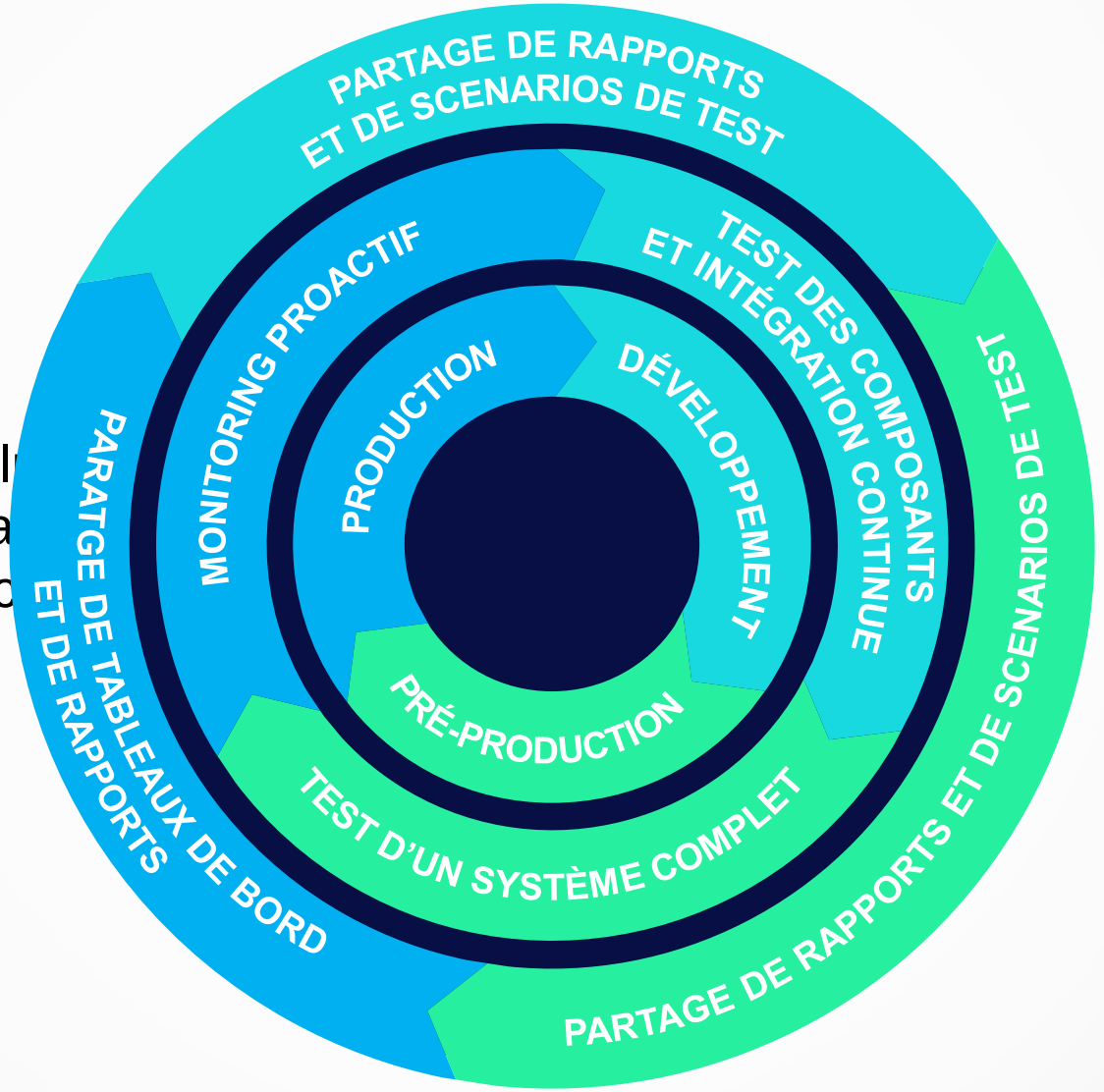
# LES NOUVELLES DIFFICULTÉS



- Les environnements de test : multiplications des environnements, cohérences des versions et des données entre chaque composant
- L'intégration de la performance et des besoins implicites (bouchons/données) dans les phases de développement
- La communication est au centre de la cohérence et de la complétude des tests

# CONCLUSION

Désormais vous n’avez pl
ne pas mettre la performa
au cœur de l’expérience c




** Source du schéma Neotys*

? QUESTIONS
RÉPONSES !