Benchttp ⚡

Tests end-to-end de la performance d'endpoints HTTP

Contexte :
lors du cycle de vie d'un produit...

nouvelle **feature**

refactoring

changement de **spécifications**

→ risques d'introduire des **régressions**

→ nouveaux **standards** à atteindre

Contexte :
lors du cycle de vie d'un produit...

nouvelle **feature**

refactoring

changement de **spécifications**

→ risques d'introduire des **régressions**

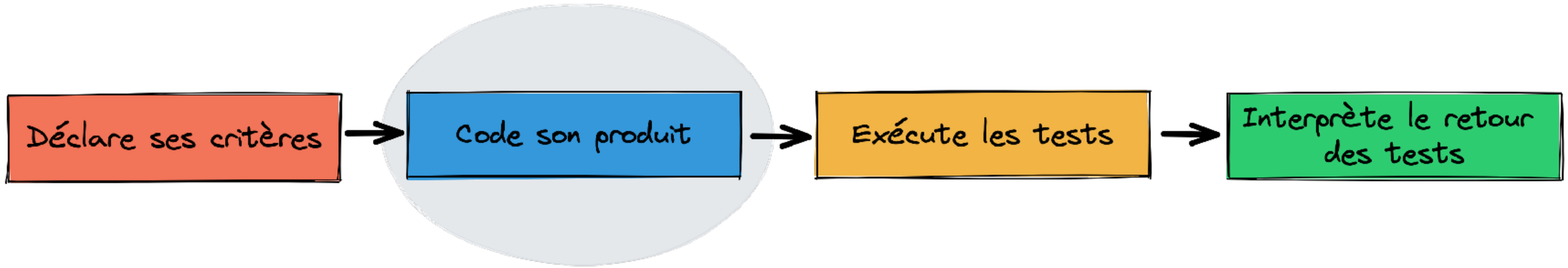
→ nouveaux **standards** à atteindre

→ **adressé pour le front-end peu ou pas adressé pour le back-end**

Quels besoins adresser ? Spécifications du problème

- spécialisé dans l'évaluation de **performance d’endpoints HTTP**
- offre des **statistiques avancées** pour analyser la performance
- permet de définir ce qui passe et ne passe pas un **test d'acceptation**
- utilisable de manière **déclarative** et à **haut niveau**, pas via le code source utilisateur

User journey :
le flux de travail qu'un utilisateur veut suivre



Quelles solutions aujourd’hui ?

Outils généralistes

Puissants mais ne couvrent pas exactement nos besoins

- **Postman** ou **Insomnia**
- testent le **contenu** des réponses

- écrire ses propres tests
- implique de la **maintenance**
- demande de s'éloigner du **métier**

Outils spécialisés

Correspondent mais complexes et/ou invasifs dans le workflow

- **Gatling** ou **Blackfire**
- flexibles mais complexes en configuration
- écrire du code via un SDK
- ouvrir un compte utilisateur

Outil adéquate

Benchttp
= Cypress x (Insomnia + Postman)

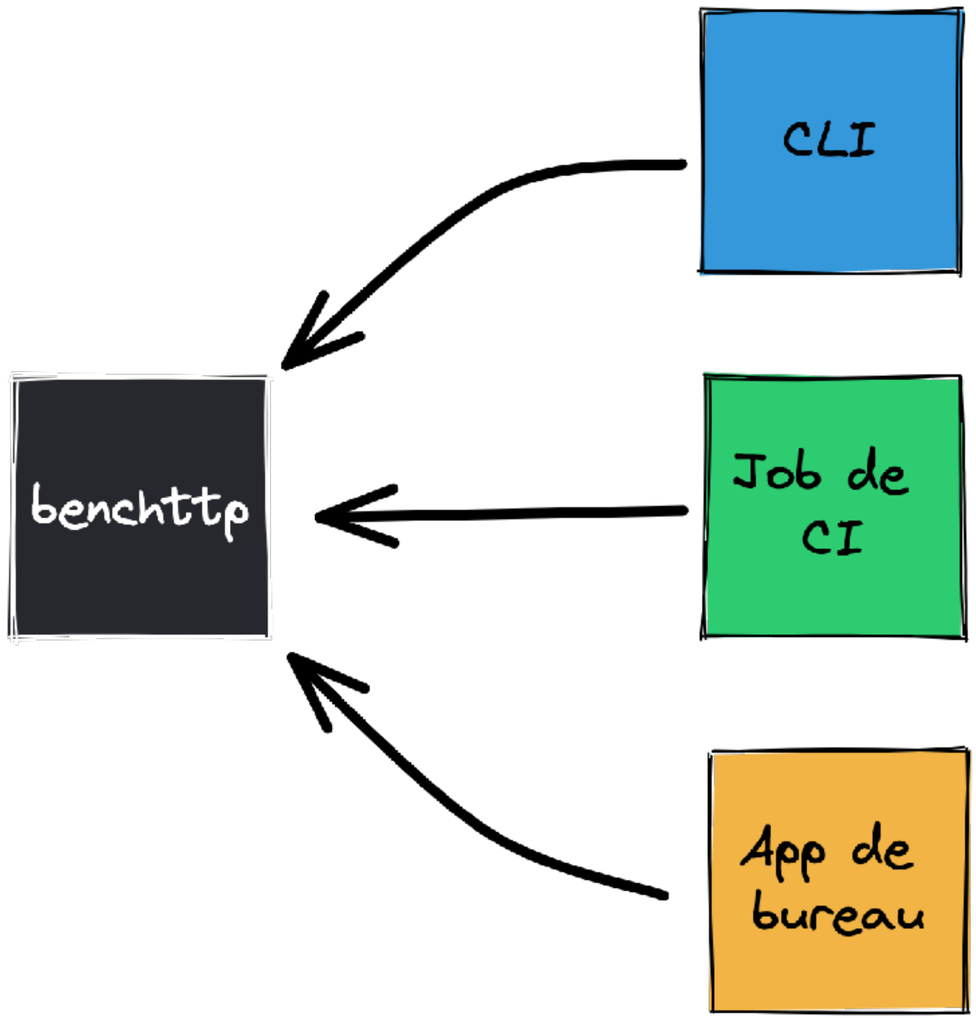
La solution Benchttp : les cas d'utilisation

tester pendant la **phase de développement**

faire des **tests d'acceptation** et de **non-régression en CI**

visualiser les **statistiques avancées** qui sont générées

1 moteur → 3 cas d'utilisation → 3 applications

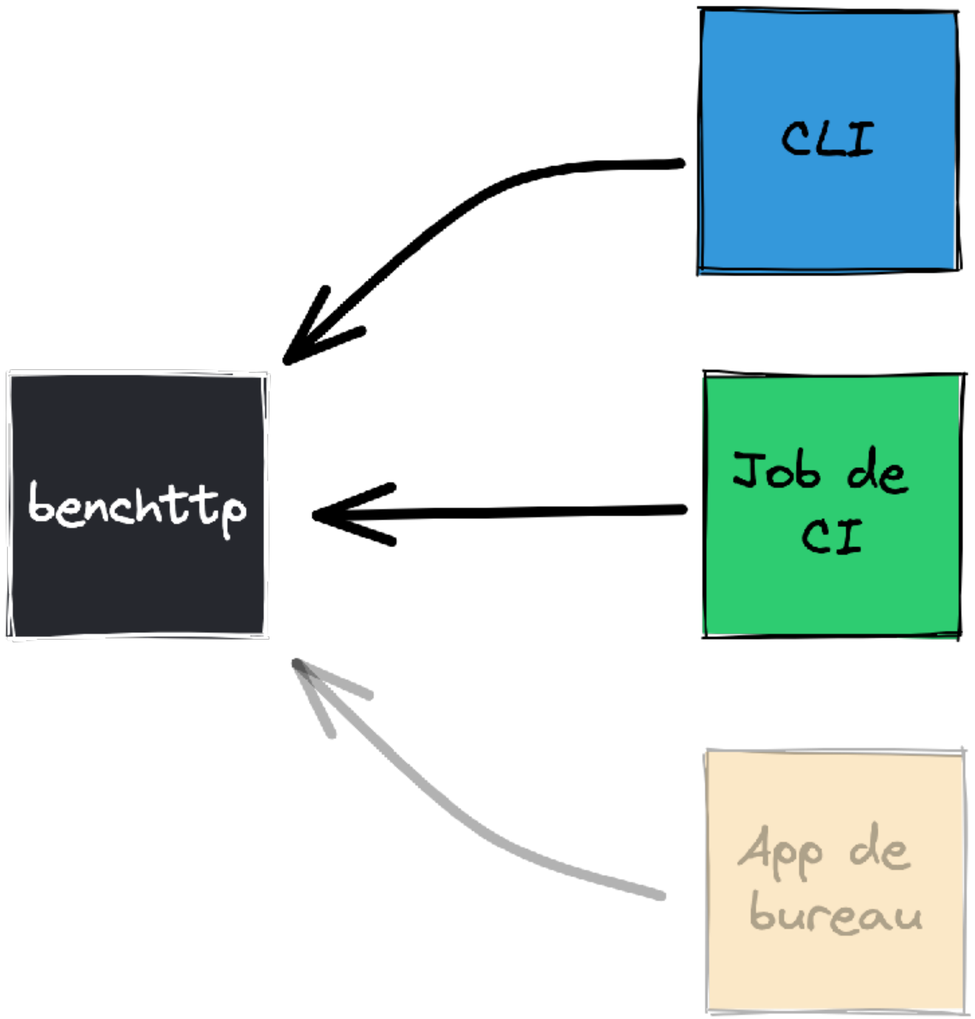


configurer et tester
feedback loop pendant le développement

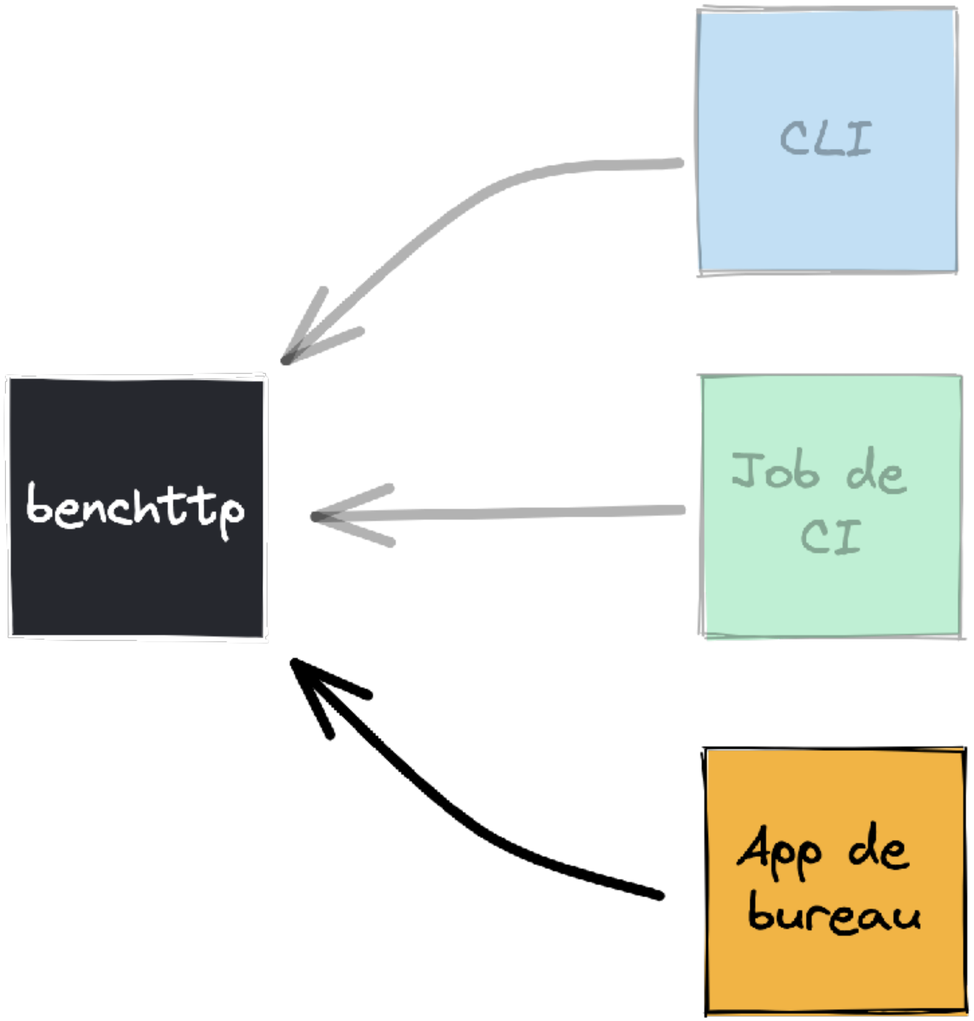
configurer, tester et sécuriser
tests d'acceptation et de non-régression automatiques

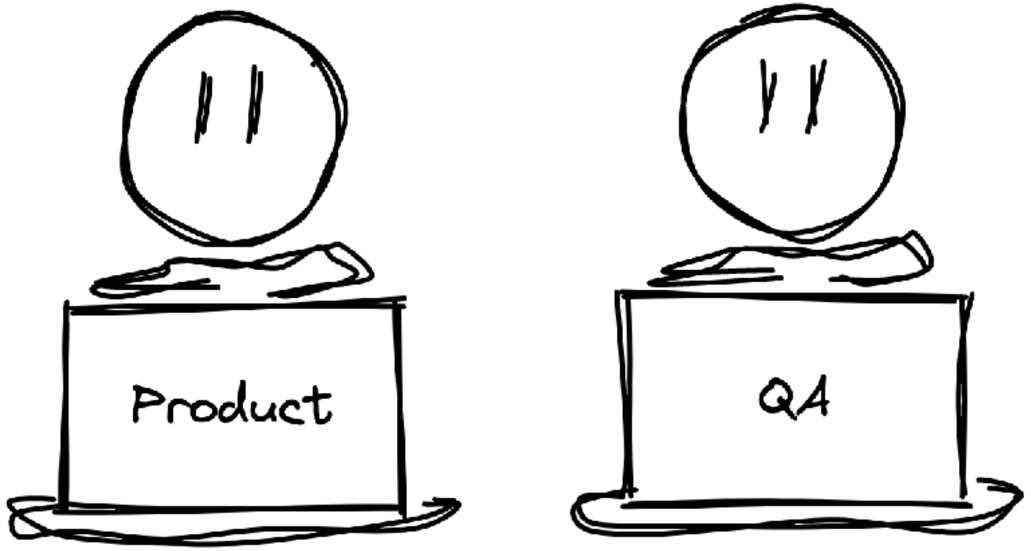
configurer, tester et visualiser/explorer
données statistiques et détails des tests

Utilisateurs cibles



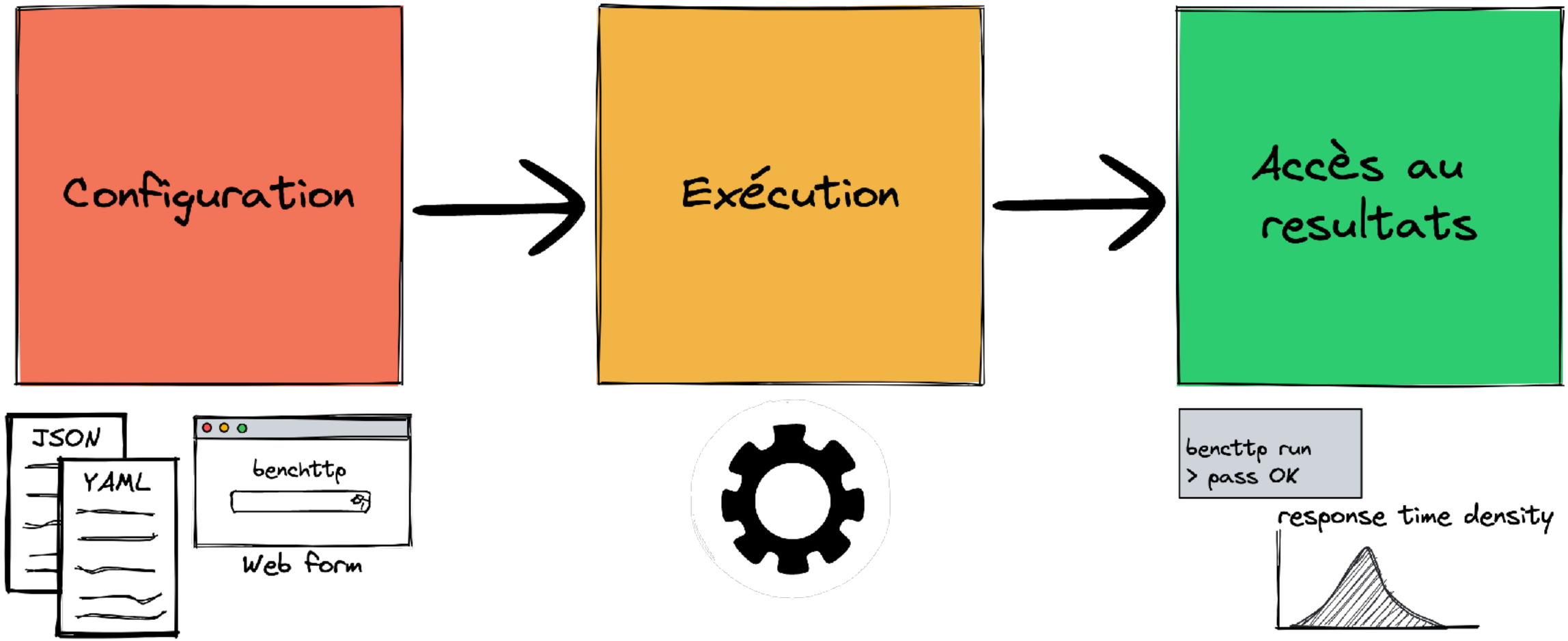
Utilisateurs cibles





Périmètre du POC commercial : un parcours d'exécution complet

Un parcours d'exécution complet



Un parcours d'exécution complet

moteur **opérationnel à 100%**

gestion des **erreurs prévisibles**

UX **claire**

→ la persistence de la donnée générée est considérée hors scope !

Démo live 

Démo → benchttp/cli

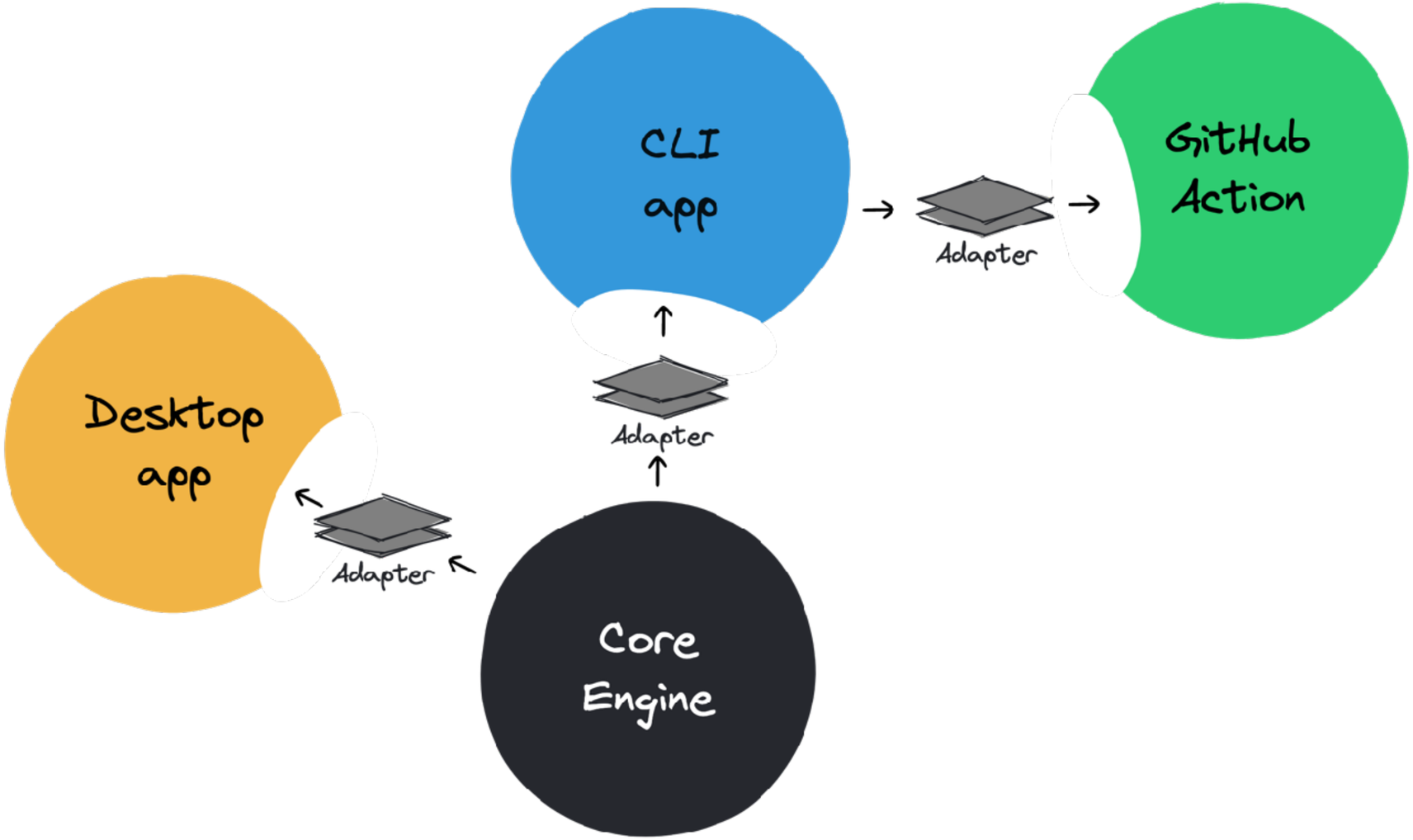
Démo → benchttp/action

github.com/benchttp/cobaye/pulls

pass ✓ fail ✗

Démo → benchttp/desktop

Architecture de la solution (à haut niveau)



Choix technique :
exécution entièrement locale

Exécution entièrement locale due à des contraintes incontournables

intégration au flow de **développement**

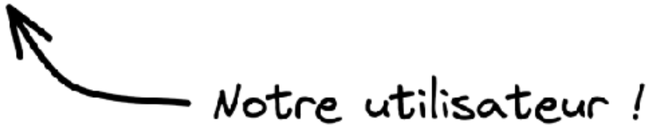
besoins de **scalabilité imprévisibles** (milliers de requêtes concurrentes !)

failles de **sécurité** et enjeux de **responsabilité** (endpoints inconnus à l'avance !)

→ supprime les coûts d'infrastructure et de déploiement
→ délègue au client la sécurité et la responsabilité

Challenge technique : contourner les CORS en local

le navigateur ne peut pas faire des requêtes vers d'autres origines que celle du site actuel sans modification du code serveur



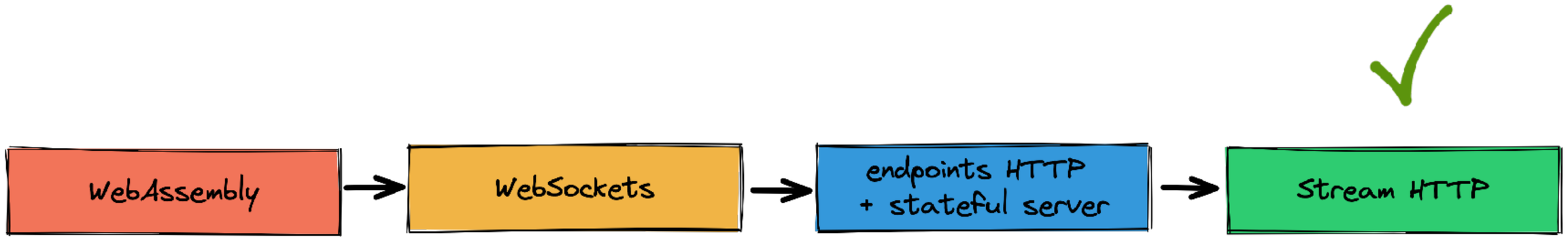
Challenge technique : contourner les CORS en local

le navigateur ne peut pas faire des requêtes vers d'autres origines que celle du site actuel sans modification du code serveur

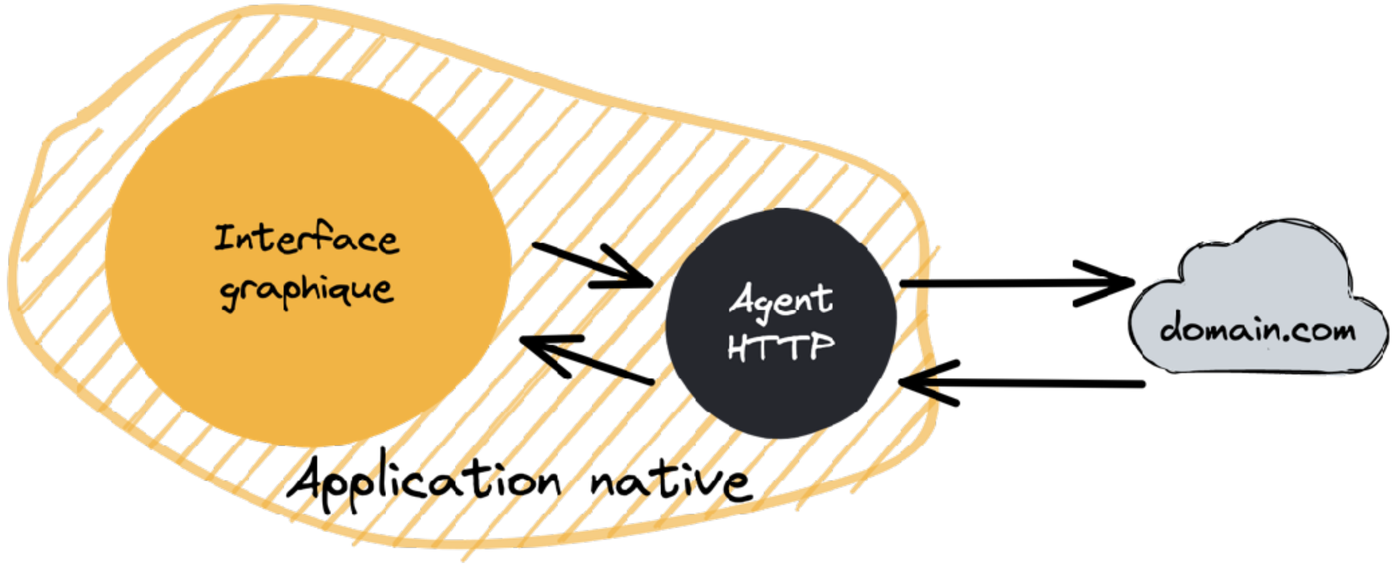
Contourner les restrictions CORS en local

Comment réussir à faire communiquer le core et l'interface graphique via un agent HTTP local ?

→ POC successifs



Conclusion technique :
app desktop + serveur HTTP embarqué



→ cross-platform et installable sans aucune dépendance

Roadmap 

benchttp@next

- pouvoir comparer différentes exécutions
- pouvoir écrire des scénarios de tests
- pouvoir setup des niveaux de sévérités dans la suite de tests (ex. : warn)
- pouvoir profiler l’utilisation hardware en direct
- avoir des statistiques plus poussées
- pouvoir importer/exporter des configurations dans l’app desktop

Annexe :
Architecture détaillée

Solution modulable et faiblement couplée

Annexe :
Stack technique

Core et CLI

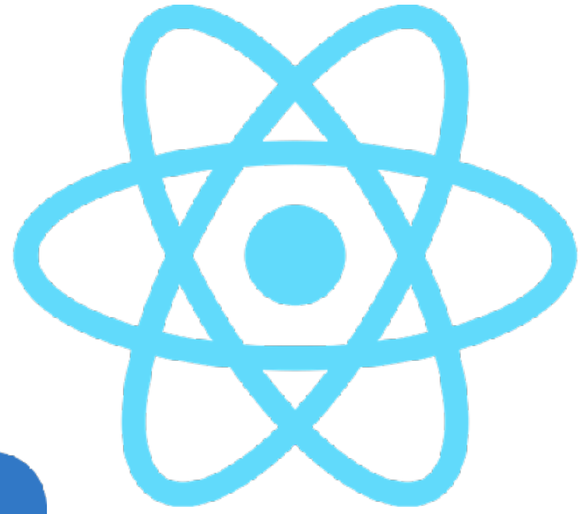
Performance et portabilité

- performant sur les cas d’usage
- accès bas niveau sur le protocol HTTP et hardware
- programmation concurrente incluse
- portatif via la compilation en binaire

Front-end : creation d’interfaces

Outils solides et écosystème riche

- compétences de l’équipe
- écosystème vaste, activement maintenu, largement éprouvé
- portabilité conséquente (navigateur, app bureau)



Construction de l'application desktop

Compatible et transparent

- compatible React pour l'interface
- compile cross-platform
- embarquement d'un executable binaire via bindings Rust

Annexe 3 :
Workflow d'équipe et tooling

Workflow et tooling