Benchttp

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

Gregory Albouy, Clara Gonnon, Damien Mathieu, Alex Mongeot, Thomas Moreira, Kérian Pelat, Hicham Sbihi

Au cours du cycle de vie d'un produit, il faudra ...

intégrer nouvelle **feature**

faire du **refactoring**

réécrire après un **changement de spécifications**

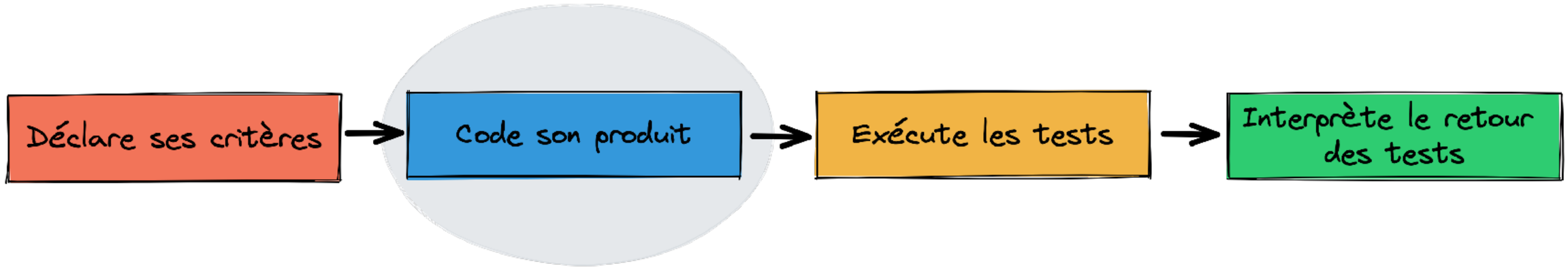
→ risques d'introduire des **régressions de performance**

→ nouveaux **standards** à atteindre

Quels besoins adresser ? Spécifications du problème

- pouvoir **évaluer** la **performance d'endpoints HTTP**
- pouvoir **analyser** la performance avec des **statistiques adaptées**
- pouvoir **définir** ce qui passe et ne passe pas un **test d'acceptation**
- pouvoir être **déclaratif à haut niveau**, ne pas toucher au code source

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**
- sont à exécuter **manuellement**
- testent le **contenu** des réponses

Outils spécialisés

Correspondent mais complexes et/ou invasifs dans le workflow

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

Benchttp

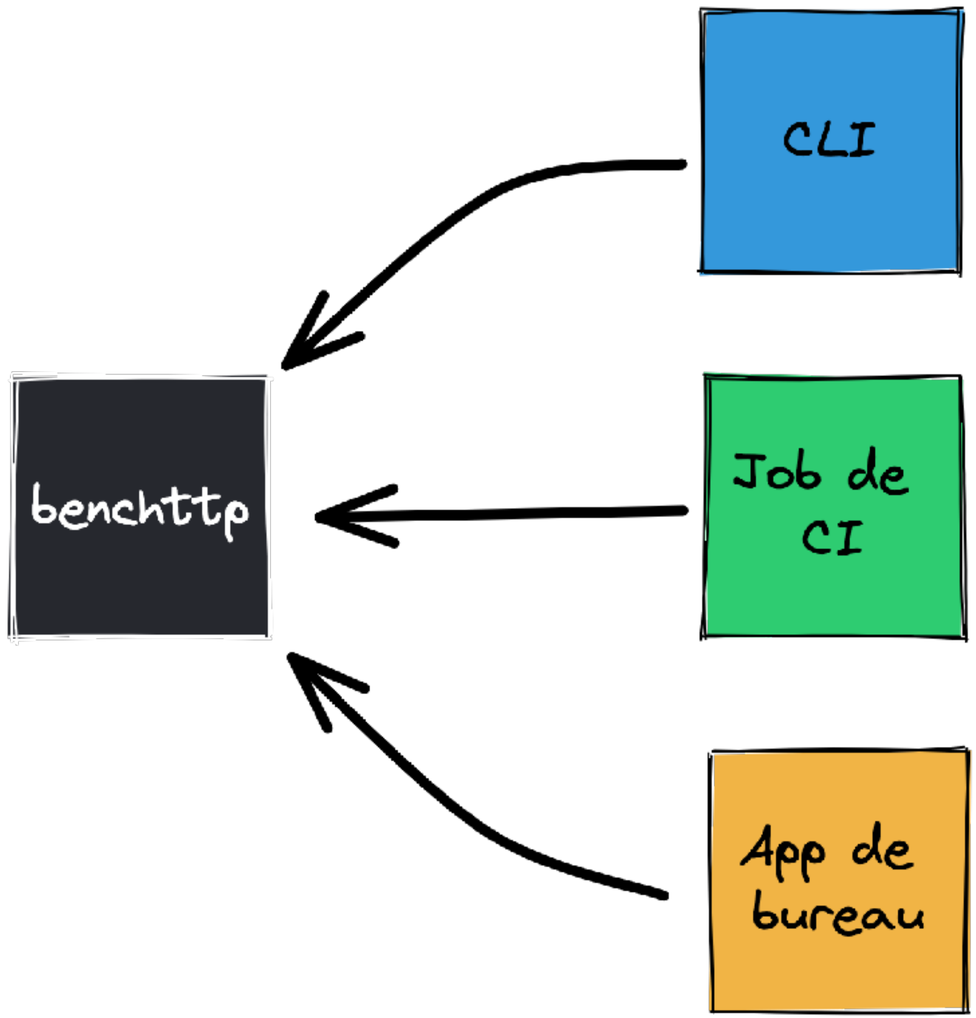
Benchttp :
la solution et les cas d’utilisation

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

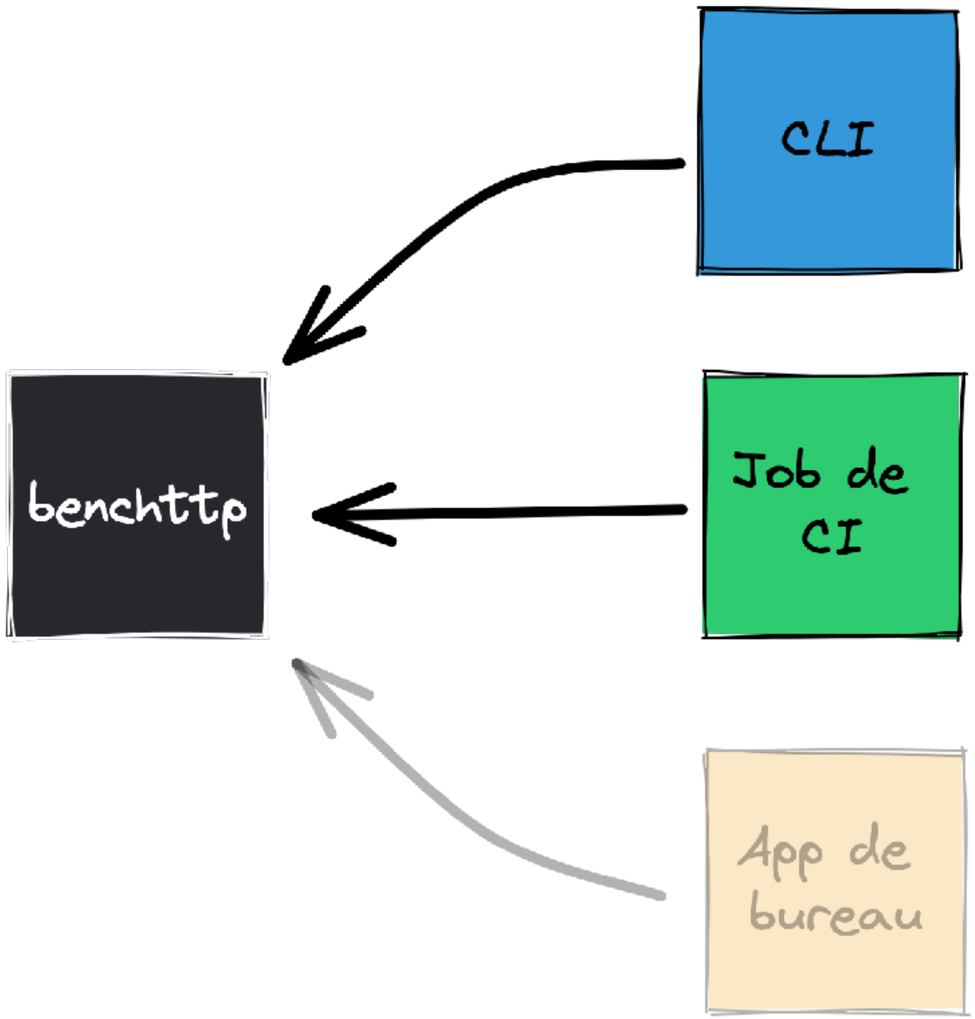


configurer et tester
feedback loop pendant le développement

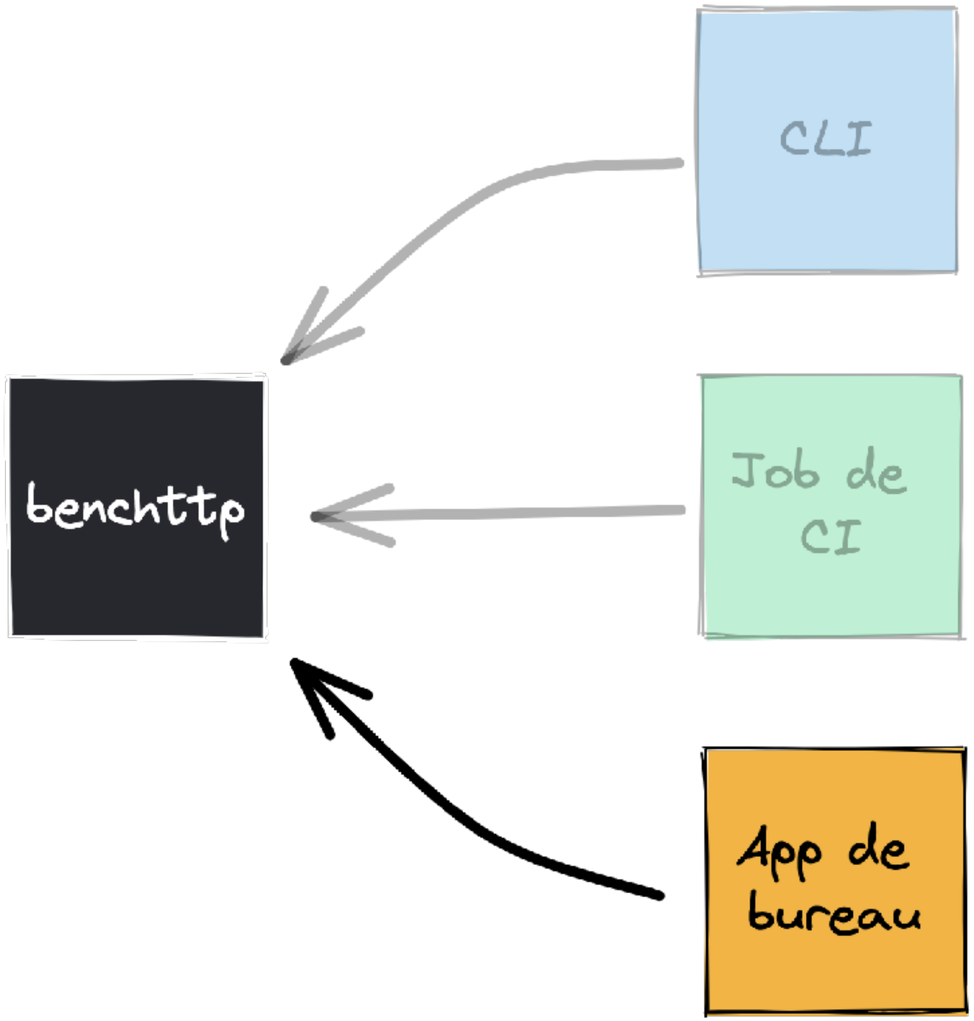
configurer, tester et sécuriser
tests d'acceptation et de non-régression automatiques

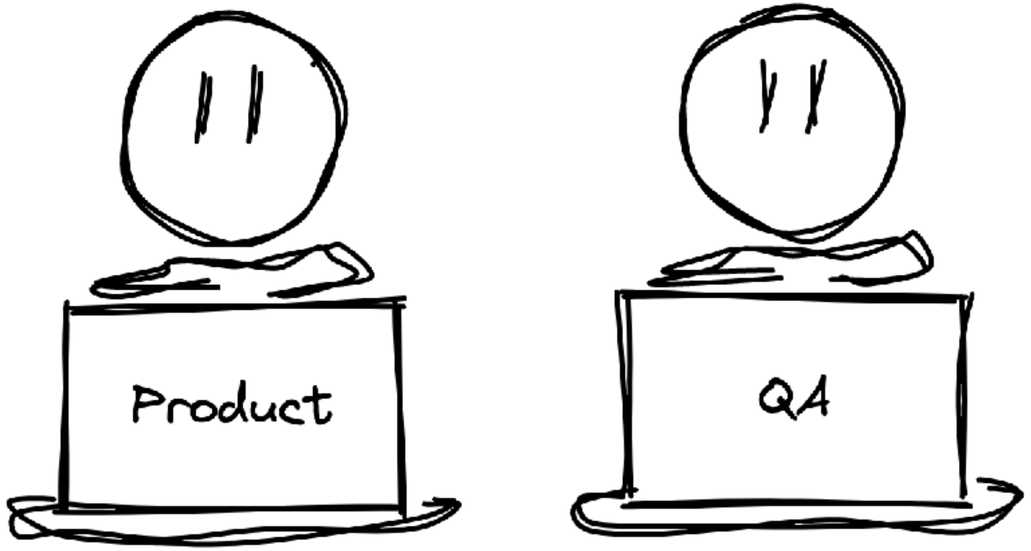
configurer, tester et visualiser/explorer
données statistiques et détails des tests

Utilisateurs cibles



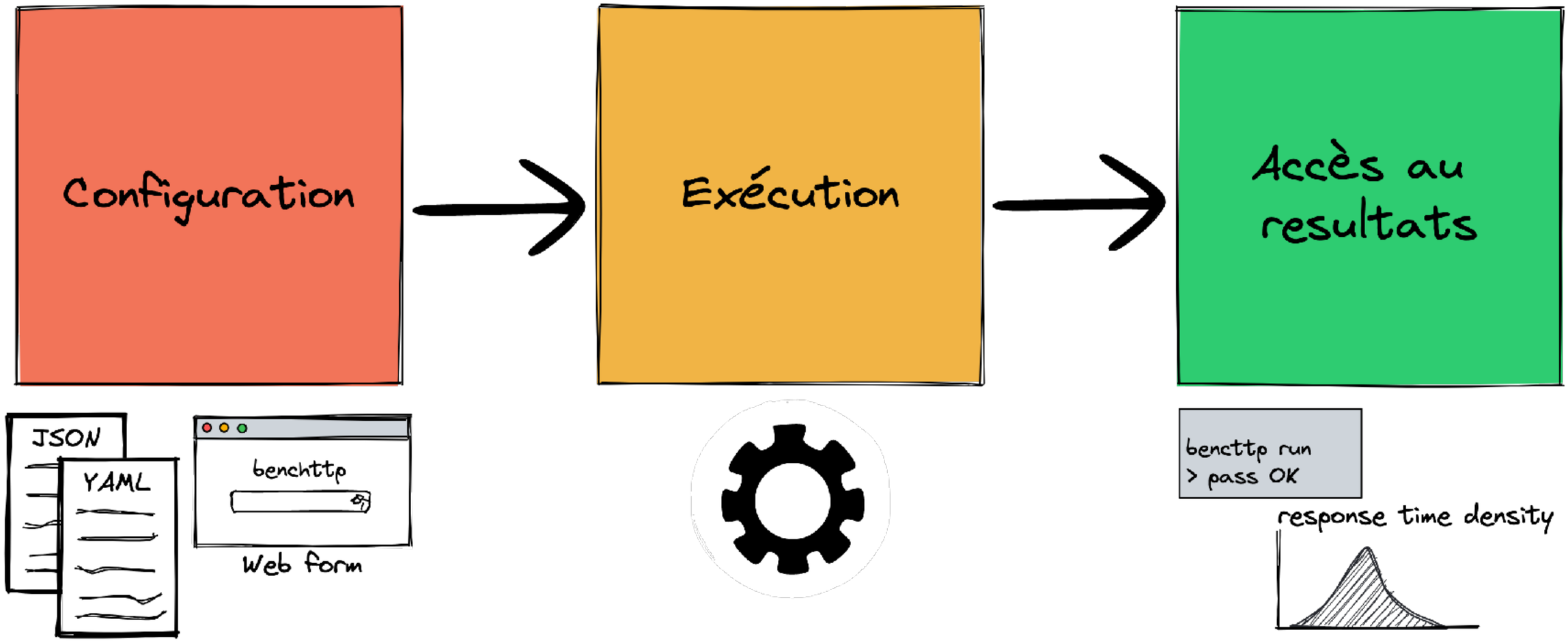
Utilisateurs cibles





Périmètre du POC commercial : un parcours d'exécution complet

Un parcours d'exécution complet



Les implications techniques du POC

moteur **opérationnel à 100%**

gestion des **erreurs prévisibles**

UX **claire**

minimum de supposition sur le contexte d'utilisation : **non-opinonated**

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

Démo live 

Démo → benchttp/cli

Démo → benchttp/action

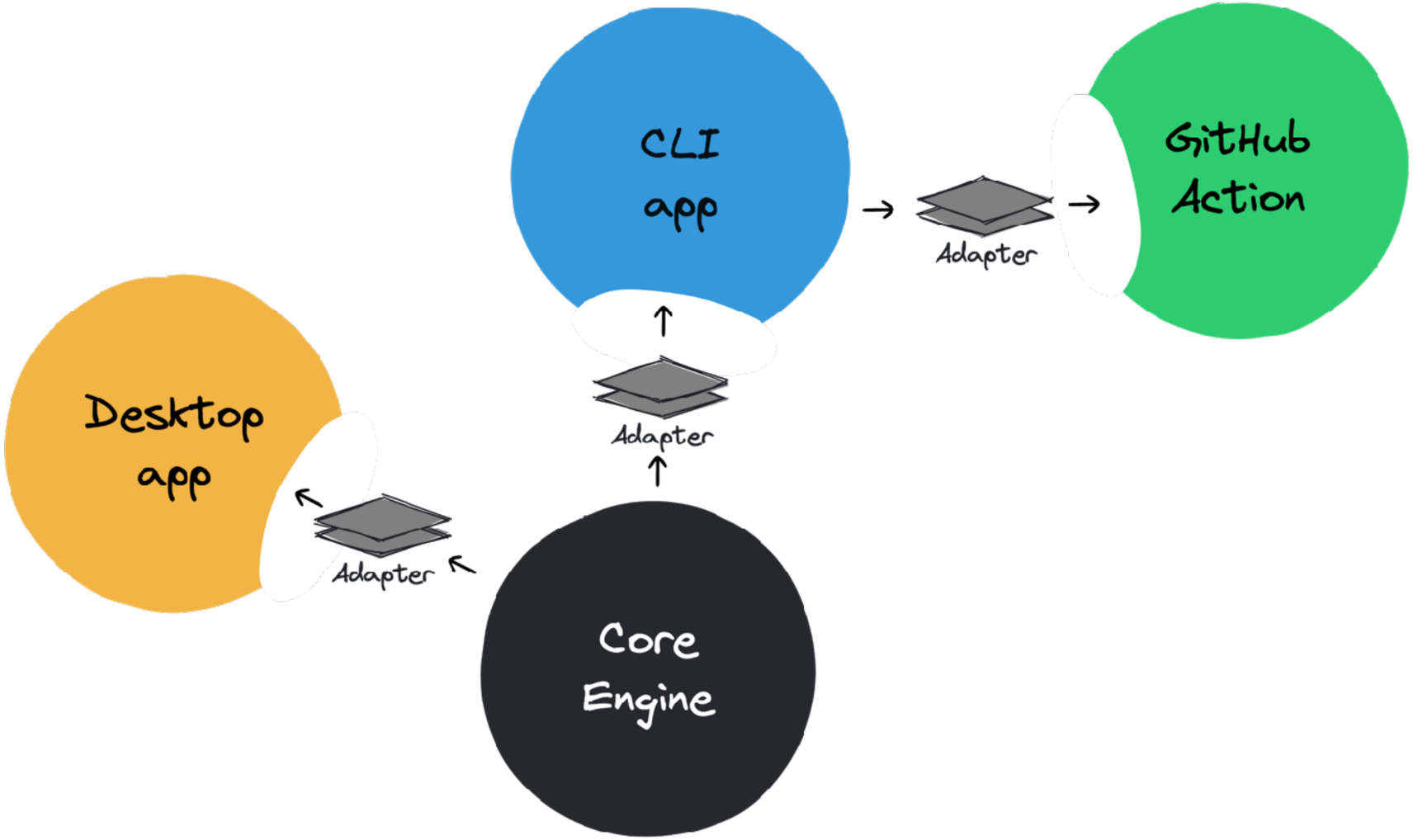
github.com/benchttp/cobaye/pulls

pass ✓ fail ✗

Démo → benchttp/desktop

Technique ⚙️

Architecture globale de la solution



Choix technique :
exécution entièrement locale

Exécution entièrement locale due à des contraintes incontournables

intégration au flow de **développement**

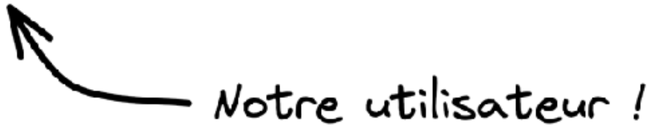
besoins de **scalabilité imprévisibles** (milliers de requêtes concurrentes !)

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 restrictions CORS

impossible de faire des requêtes depuis un navigateur vers un serveur qui n'accepte pas l'origine de Benchttp



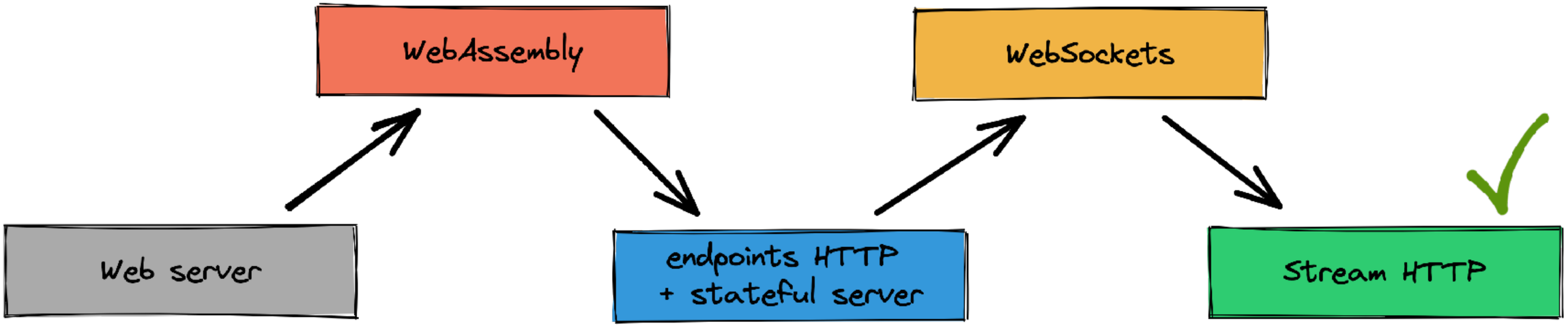
Challenge technique : contourner les restrictions CORS

impossible de faire des requêtes depuis un navigateur vers un serveur qui n'accepte pas l'origine de Benchttp

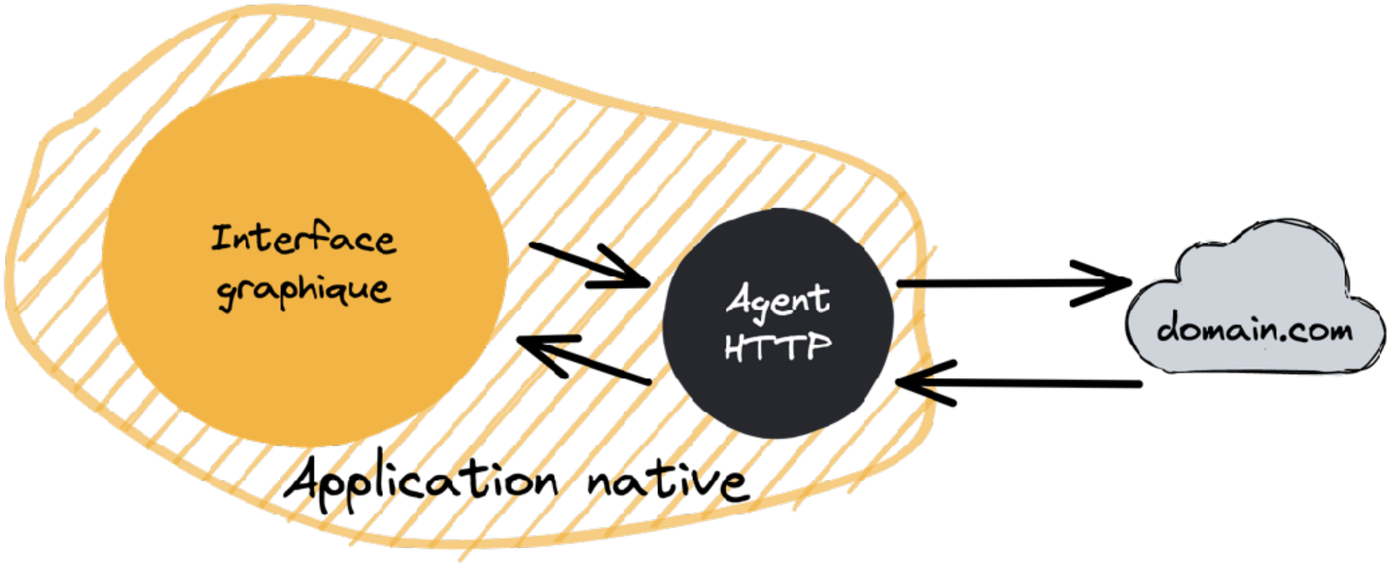
→ un agent HTTP doit faire les requêtes localement !

Comment faire communiquer l'application front-end avec l'agent HTTP local ?

→ POC successifs → affiner la solution



Conclusion technique :
app desktop + serveur HTTP embarqué



→ installable sans aucune dépendance et cross-platform

Roadmap 🪧 benchttp@next

- persistance des résultats
- pouvoir comparer différentes exécutions
- pouvoir setup des niveaux de sévérité dans la suite de tests (ex. : warn)
- pouvoir profiler l’utilisation hardware
- pouvoir importer/exporter des configurations dans l’app desktop

Annexe :
Architecture détaillée

Solution modulable et faiblement couplée

Annexe :
Stack technique

Core et CLI

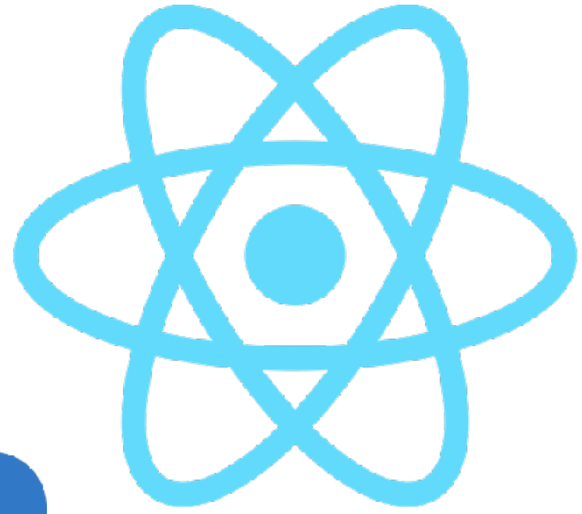
Performance et portabilité

- 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 :
Workflow d'équipe et tooling

Workflow et tooling