UNIVERSITÉ DE VERSAILLES ST-QUENTIN-EN-YVELINES

université PARIS-SACLAY

#7

23/01/2026

jean-michel.batto@cea.fr

cea

https://gogs.eldarsoft.com/M2_IHPS

Les patrons / patterns

Les 23 patterns :

	Creational	Structural	Behavioral
Class	Factory Method	17 Adapter (class)	21 Interpreter
			Template Method
Object	Abstract Factory	17 Adapter (object)	13 Chain of Responsibility
	14 Builder	Bridge	22 Command
	16 Prototype	18 Composite	Iterator
	Singleton	10 Decorator	23 Mediator
		11 Facade	12 Memento
		19 Flyweight	Observer
		20 Proxy	State
			Strategy
			15 Visitor

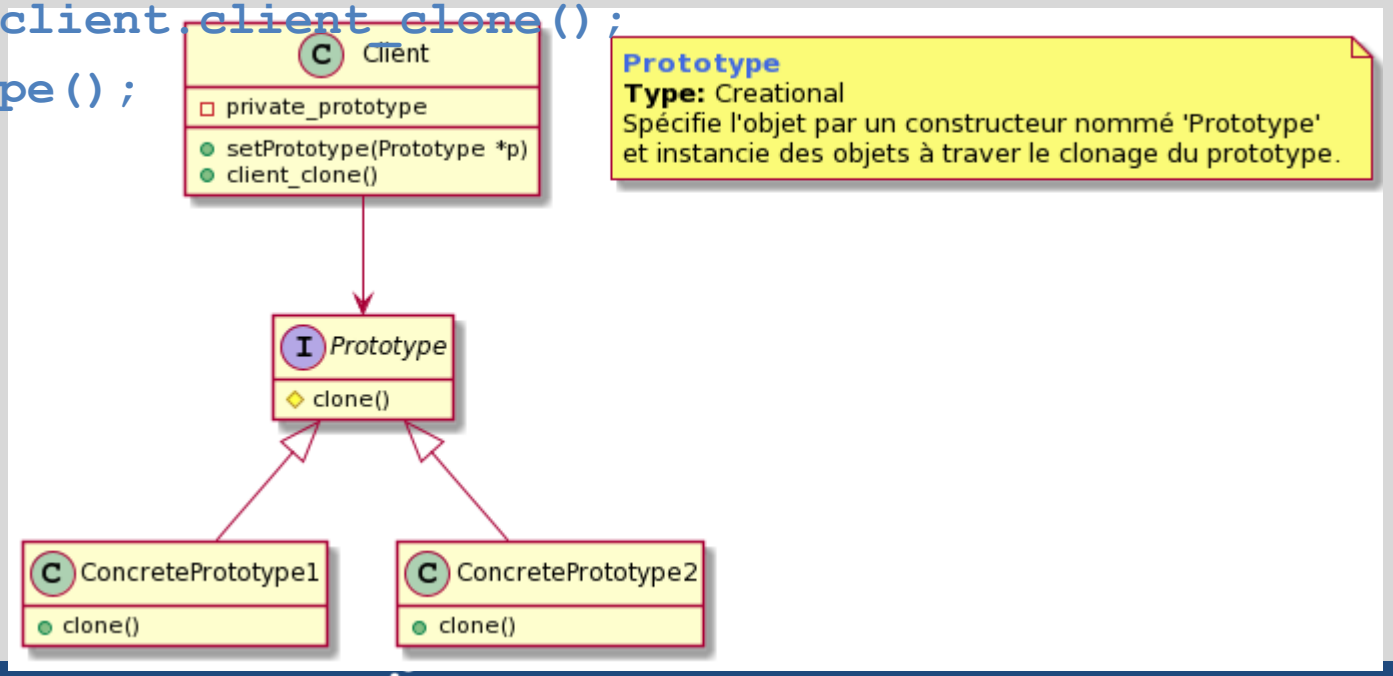
Typologie	Nom du Design Pattern	Ce qui doit être ajuster	verbe	composition sous jacente	mélange de classes ?
Creational	Abstract Factory	famille d'objets dépendants		structure	non
	Builder	Comment créer un objet composite dont la structure du composite est indépendante		liste	non
	Factory Method	Sous classe d'un objet qui est instanciée sans connaitre la classe ancêtre (connaissance retardée)			filtrage vtab
	Prototype	Classe d'objet qui est instanciée grâce à un constructeur de copie	copie=verbe	liste	non
	Singleton	La seule instance d'une classe	copie=o=verbe		non
Structural	Adapter	accède à un objet en modifiant l'interface			non
	Bridge	Fait l'implémentation d'un objet par découplage	découplage		non
	Composite	structure et composition d'un objet vue de manière uniforme		arbre	non
	Decorator	Responsabilité d'un objet sans héritage - ajout dynamique			filtrage vtab
	Facade	Exposer une interface à un sous-système			filtrage vtab
	Flyweight	cout de stockage d'un objet, partage de l'état	état=verbe	liste	non
	Proxy	Comment un objet est accédé, son emplacement (mémoire, disque) - effet miroir		queue	non
Behavioral	Chain of Responsibility	Un objet qui peut répondre à une demande avec découpplage	découplage	queue	filtrage vtab
	Command	quand et comment une commande peut être faite - la commande devient un objet		structure	non
	Interpreter	grammaire et interprétation d'un objet			non
	Iterator	se déplacer dans une structure d'objet sans en connaitre le détail		liste	filtrage vtab
	Mediator	Comment et avec quels objets sont décrites les interactions			non
	Memento	Quelles sont les informations privées qui sont stockée à part et quand?	état=verbe	état (queue=infinie)	non
	Observer	l'effectif des objets observés et quand s'effectue la mise à jour		queue	non
	State	les états d'un objet sont des variables, avec un handler()	état=verbe	structure	filtrage vtab
	Strategy	un algorithme	extension=verbe	structure	filtrage vtab
	Template Method	les étapes/squelette d'un algorithme		structure	non
	Visitor	les opérations élémentaires sont appliquées à un objet sans modifier sa classe		liste	non

Creational / Prototype

Création de nouveaux objets par copie d'un modèle

https://godbolt.org/z/M87PrKxrh

```
int main(int argc, char* argv[]) {
  Client client;
  client.setPrototype(new ConcretePrototype1);
  Prototype *p1 = client.client_clone();
  p1->checkPrototype();
  client.setPrototype(new ConcretePrototype2);
  Prototype *p2 = client.client_clone();
  p2->checkPrototype();
}
```

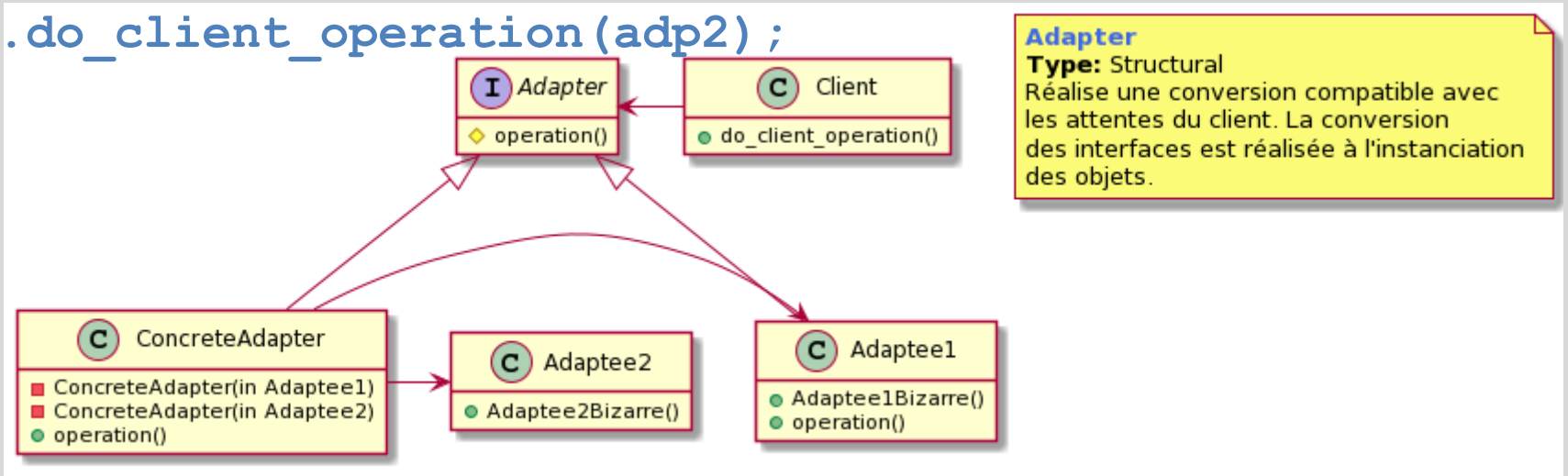


Structural / Adapter

Convertir l'interface d'une classe

https://godbolt.org/z/xj6reoaGd

```
int main() {
    Client client;
    std::cout << "new Adaptee1" << std::endl;
    ConcreteAdapter adp1(new Adaptee1());
    client.do_client_operation(adp1);
    std::cout << "new Adaptee2" << std::endl;
    ConcreteAdapter adp2(new Adaptee2());
    client.do_client_operation(adp2);
}
```

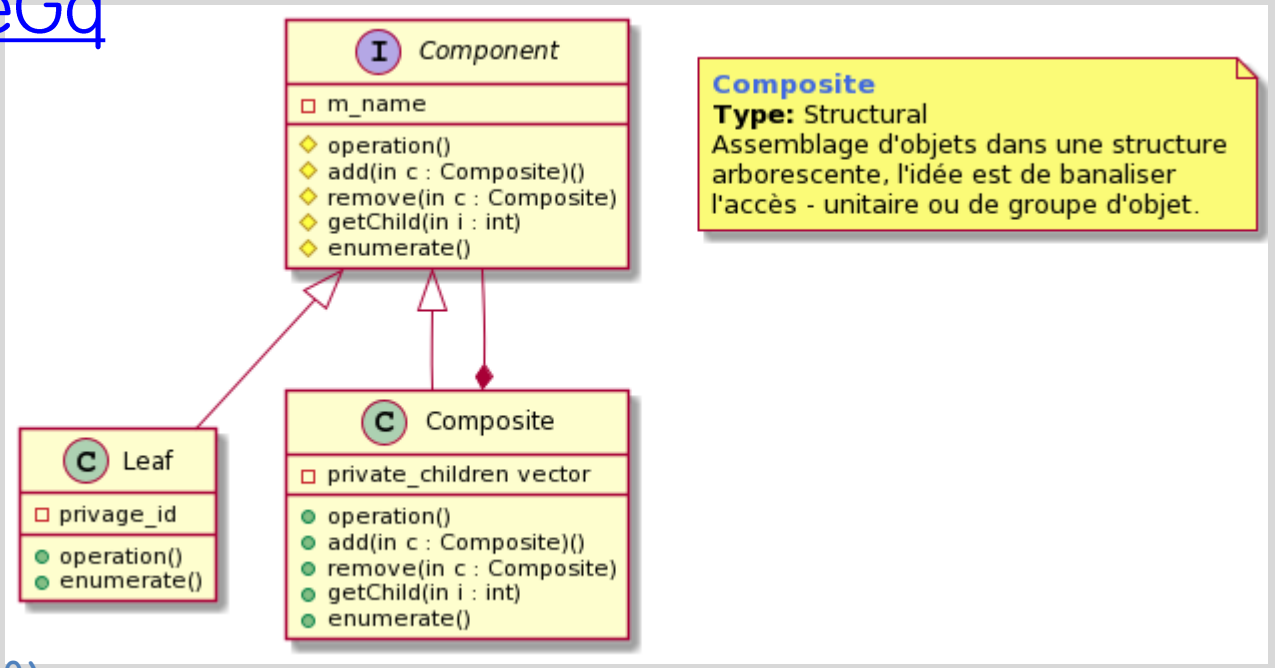


Structural / Composite

Représentation de hiérarchies d'objets vus de manière uniforme par le client

https://godbolt.org/z/ooK67xeGq

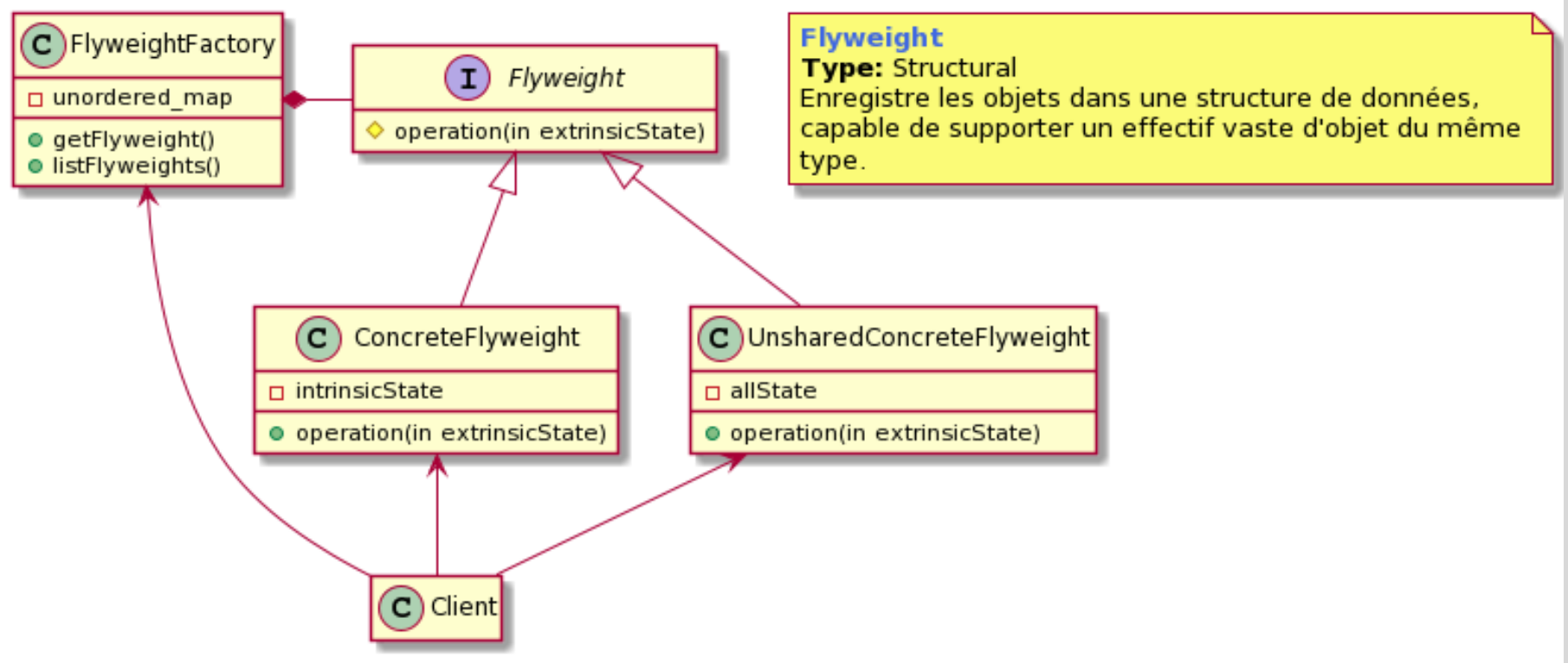
```
int main() {
  Composite composite;
  composite.group("principal");
  for (unsigned int i = 0; i < 3; ++i) {
    composite.add(new Leaf(i));
  }
  Composite composite2;
  composite2.group("secondaire");
  composite.add(&composite2);
  composite.remove(0);
  composite.operation();
  Component *component1 = composite.getChild(0);
  component1->operation();
  Component *component2 = composite.getChild(3);
  component2->operation();
  composite.enumerate();
}
```



Structural / Flyweight

Partager l'état extrinsèque - factorisation

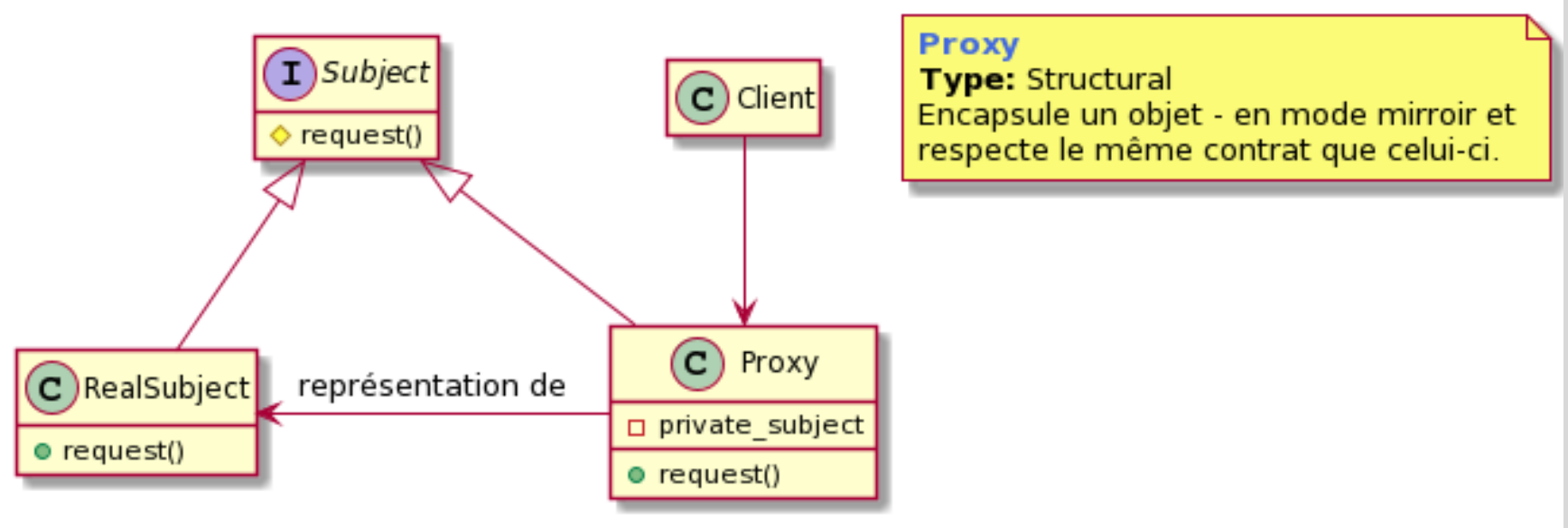
https://godbolt.org/z/xEfWhT7zc



Structural / Proxy

Objet miroir d'un autre objet plus lointain

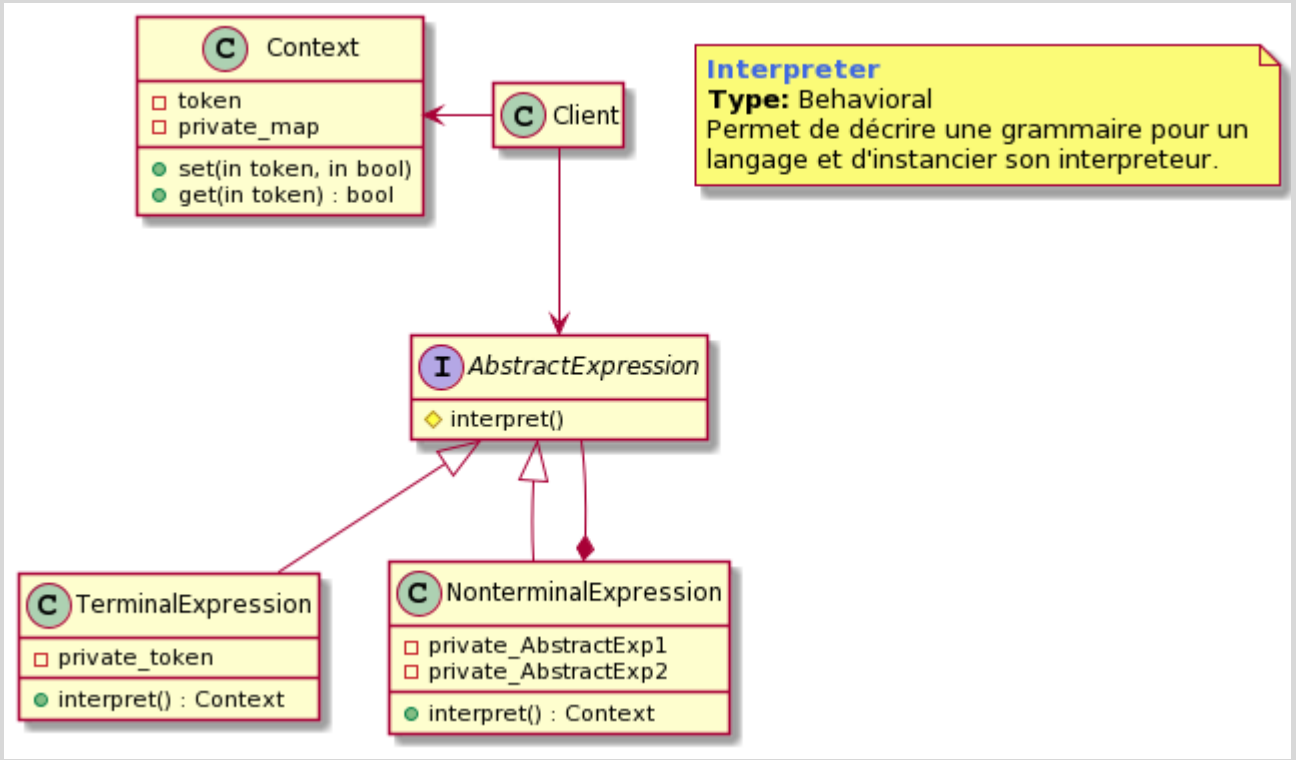
https://godbolt.org/z/cTbP35sd4



Behavorial / Interpreter

Décrit un interpreteur – utile pour un moteur d'état

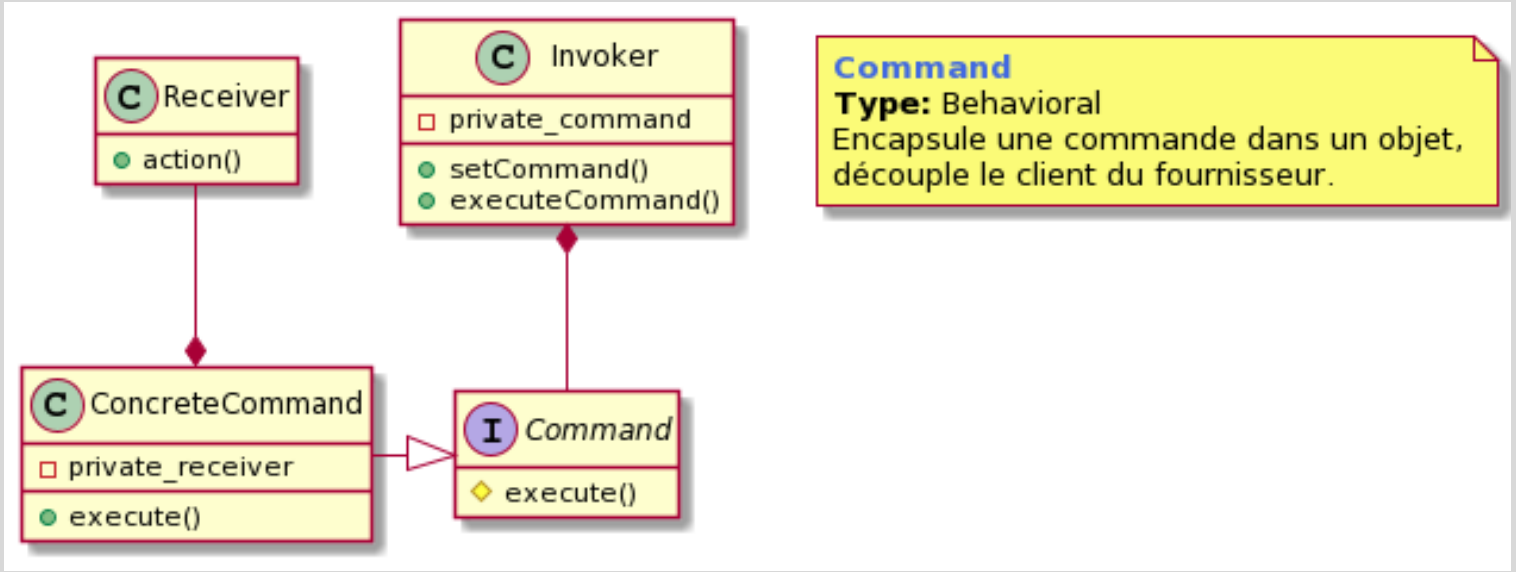
https://godbolt.org/z/oes9M9bbT



Behavorial / Command

Range une commande dans un objet, découple le client du fournisseur

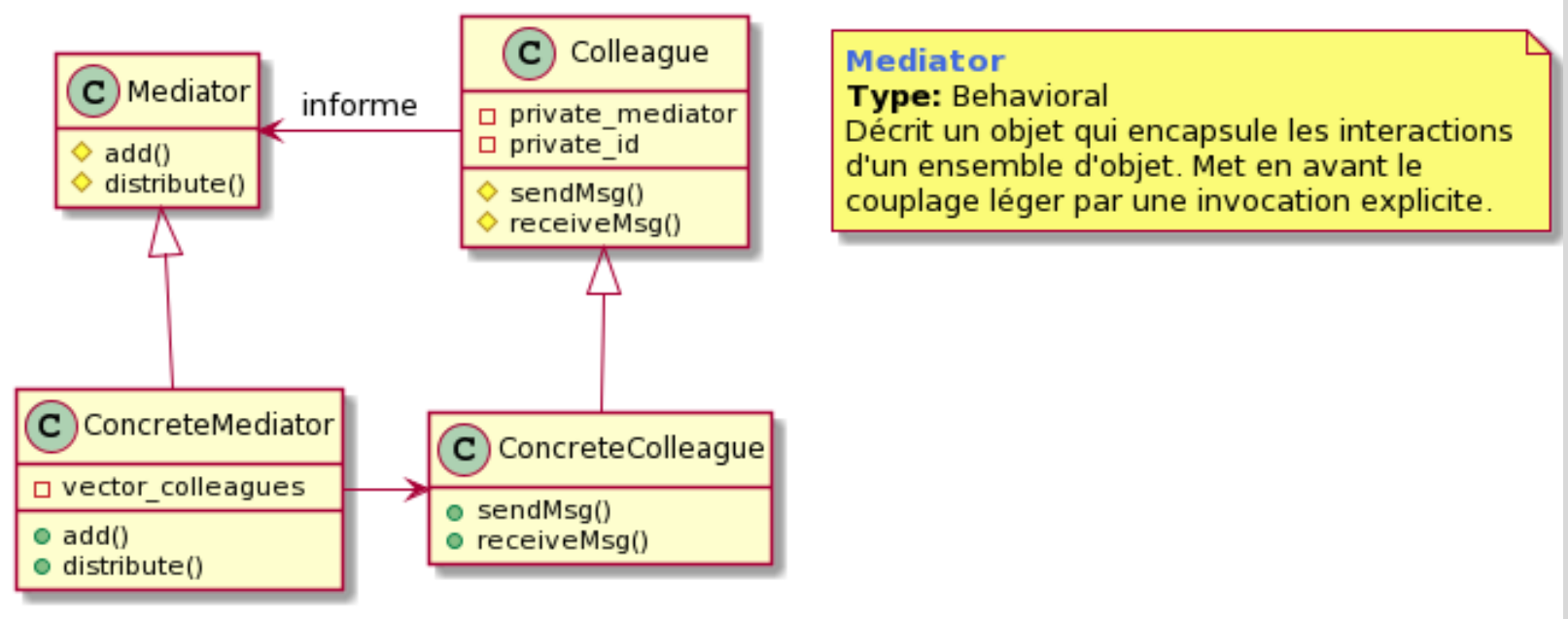
https://godbolt.org/z/GMcKodP9G



Behavorial / Mediator

Permet le couplage léger

https://godbolt.org/z/o5Eneo1sG



Comment appliquer les 23 patterns?

Biblio➔Exploring Game Architecture Best-Practices 2011.pdf

https://doi.org/10.1145/1984674.1984682

Exploring Game Architecture Best-Practices with Classic Space Invaders – 2011

➔Quels sont les patterns utiles d'après l'article?

Quelques patterns à connaitre

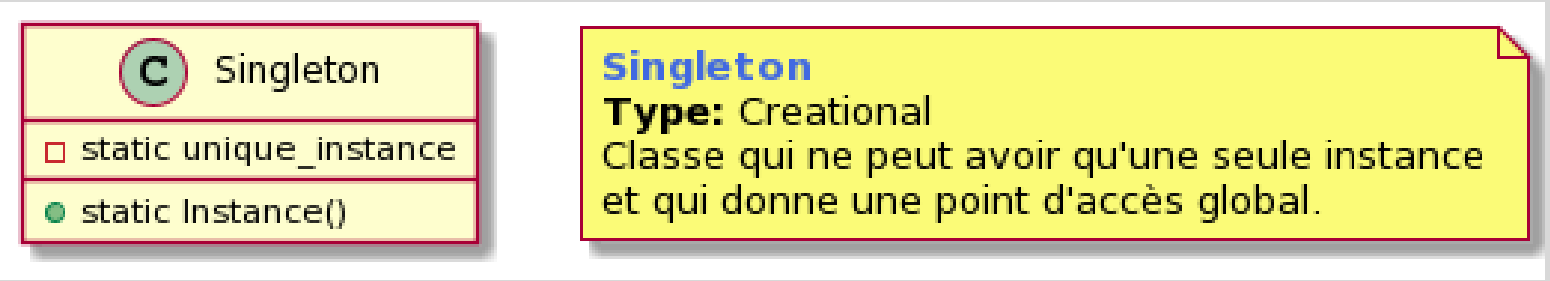
- Singleton
- Observer
- Strategy
- Adapter
- Facade

Creational / Singleton

Classe qui ne donne qu'une seule instance

https://godbolt.org/z/5916Mc6Ez

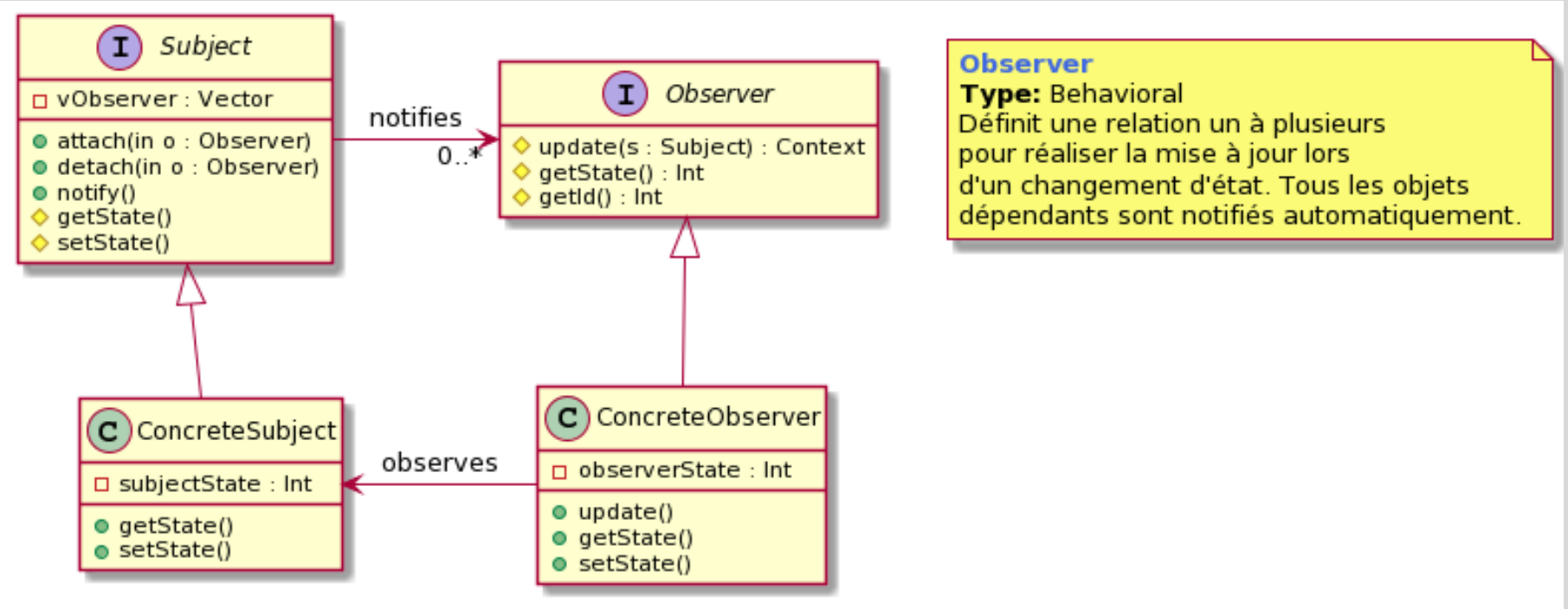
```
int main(int argc, char* argv[]) {
  Singleton *singleton = Singleton::Instance();
  singleton->checkSingleton();
  //we create a new singleton2 but...
  Singleton *singleton2 = Singleton::Instance();
  singleton2->checkSingleton();
}
```



Behavioral / Observer

https://godbolt.org/z/qMKrsoY7M

Dépendance Publish-Subscribe, idée de queue

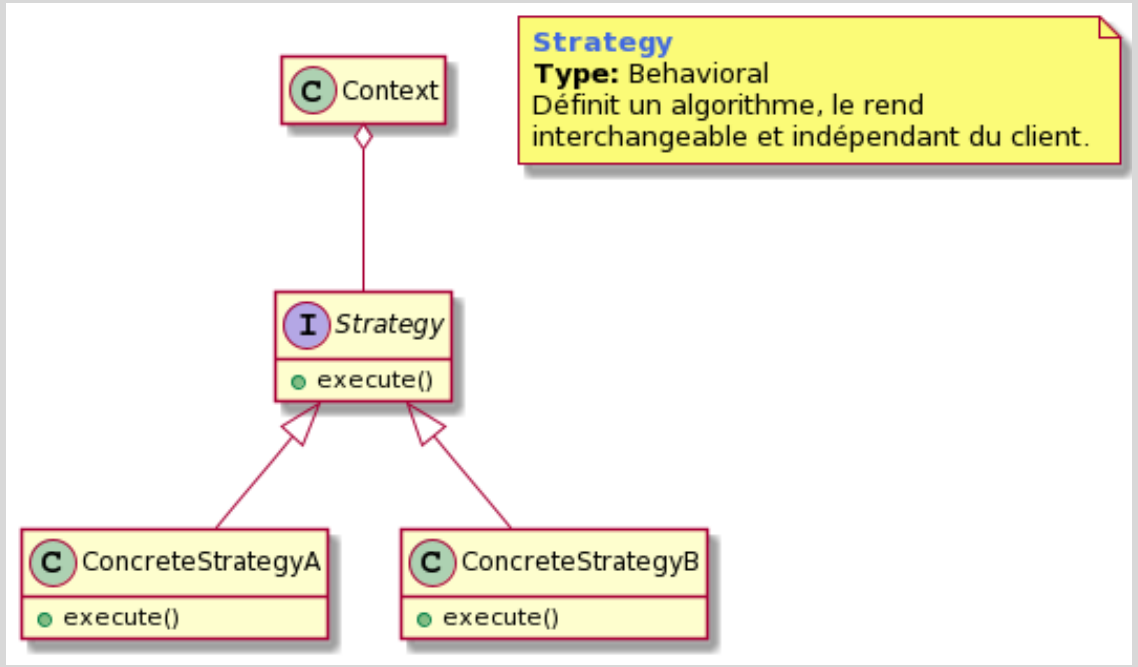


Behavioral / Strategy

https://godbolt.org/z/zbaK3Wr5T

```
int main() {
    Context context(new ConcreteStrategyB());
    context.execute();
}
```

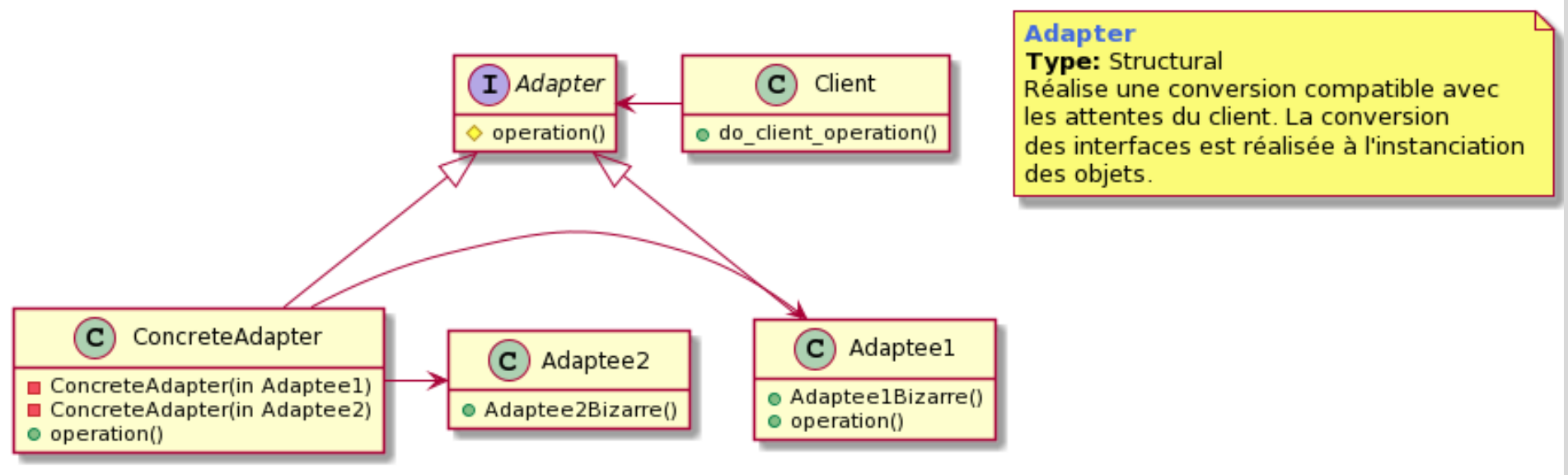
➔algorithme



Structural / Adapter

Convertir l'interface d'une classe

https://godbolt.org/z/xj6reoaGd

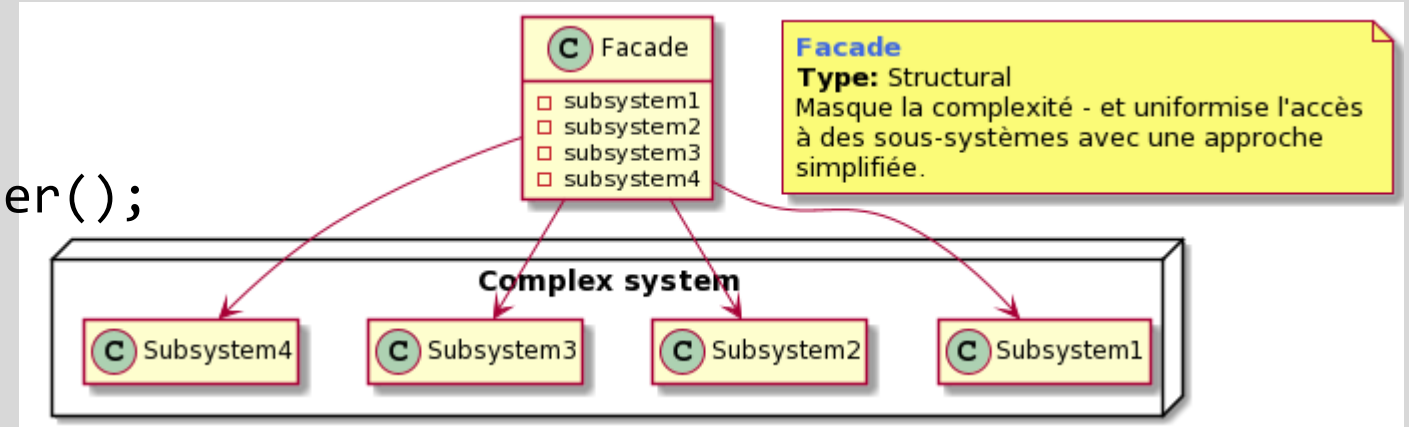


Structural / Facade

Interface de sous-système simplifiée

https://godbolt.org/z/oWTWTaPc3

```
  private:
    SubSystem1 *subsystem1;
    SubSystem2 *subsystem2;
    SubSystem3 *subsystem3;
    SubSystem4 *subsystem4;
};
int main() {
    Facade facade;
    facade.operationWrapper();
}
```



Cycle de vie du code

Build

- SFD - Recette
- Livraisons Dev / Préprod / Prod

Run

- PRA

MCO

MCS

Codage et bonnes pratiques

Penser sa communication (comment partager ses idées d'architecture)

Organiser son code / Organisation fonctionnelle

Stocker ses données

Organiser ses données

Techniques de mise en cache

Patrons de conception cloud

Configurations du code

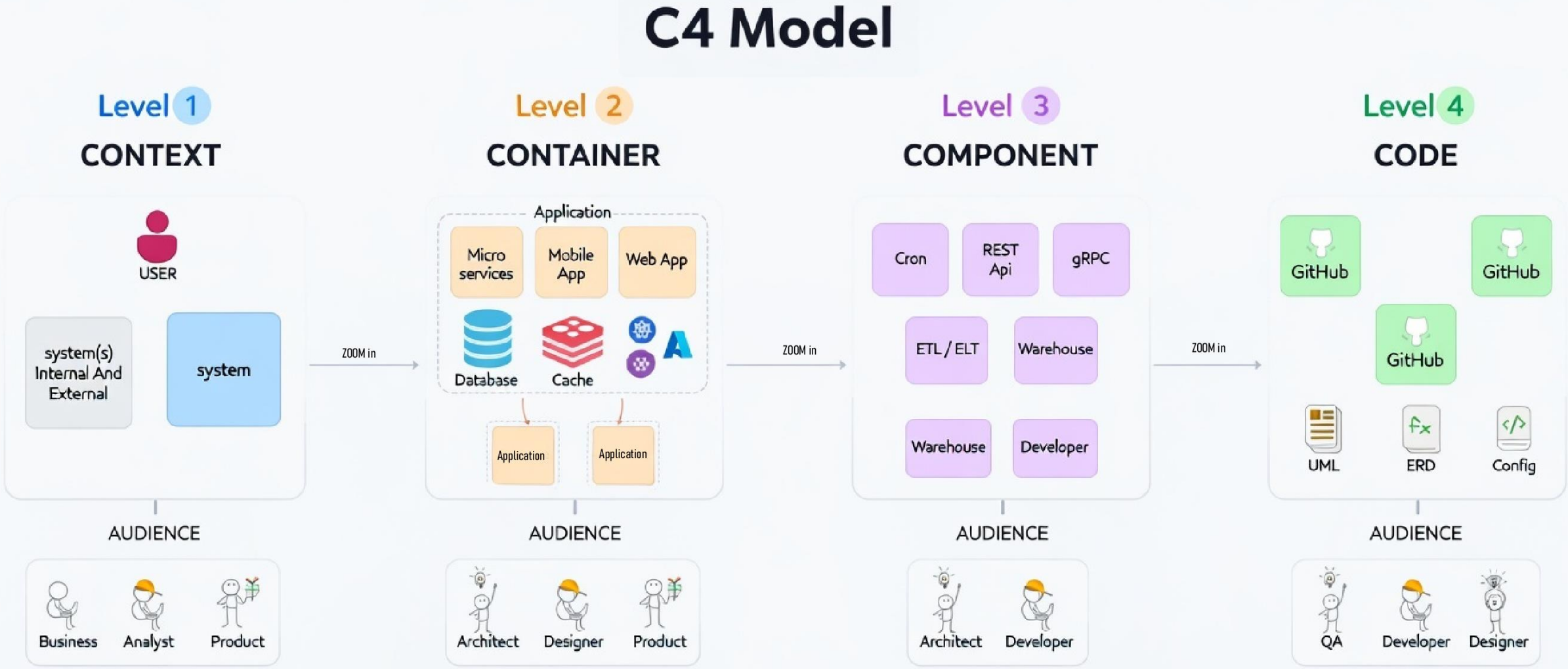
Observabilité / Journalisation / Log / Traces

Cocomo

Chiffrage / Devisage

La représentation d'architecture C4

Le modèle C4 est une technique de modélisation graphique allégée pour les architectures logicielles, basée sur une décomposition hiérarchique en quatre niveaux : contexte, conteneurs, composants et code.



Organiser son code

Organisation ses répertoires:

- Avoir une règle de nommage
- Eviter les déséquilibres
- Identifier les tests et la documentation
- Penser à l'impact de l'arborescence dans le git

Bonnes pratiques de créations de paquets

Organisation fonctionnelle : le modèle hexagonal

Bonnes pratiques de créations de paquets

Organiser en paquets nommés par le service rendu, pas le contenu

Publier le strict minimum nécessaire aux utilisateurs

Privilégier la simplicité d'API à la multiplicité de réglages nécessaires

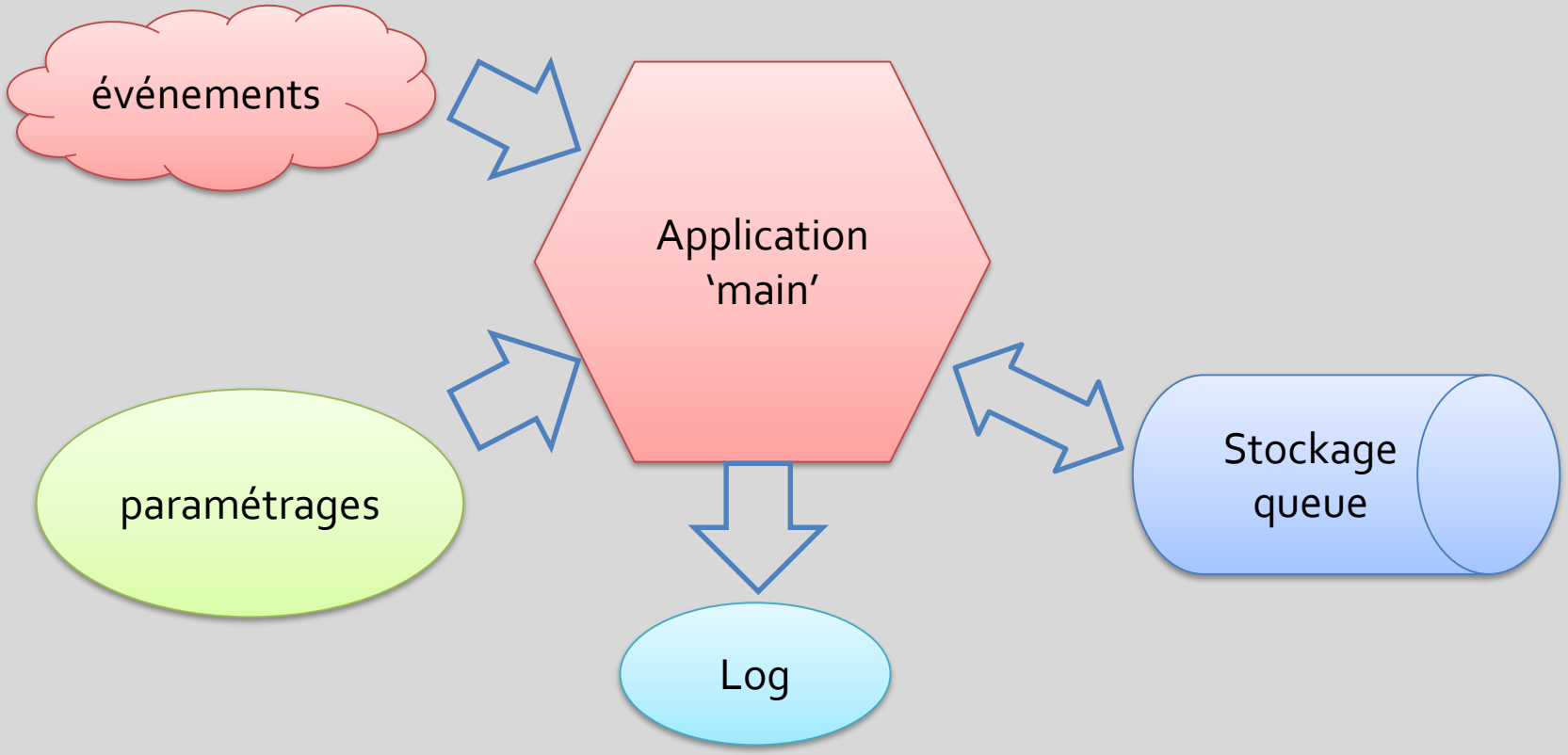
Fournir des erreurs typées et documentées

Prévoir une version des paquets pour l'utilisation concurrente – avec un nom type HPC.

Mettre un fichier README.md par paquet, avec une date de mise en développement, les auteurs éventuels, et une explication de l'usage du paquet.

Organisation fonctionnelle, le modèle hexagonal

La vision hexagonale – le « main » est au centre



Principes SOLID (acronyme de 2004)

Single responsibility

Une classe ne s'occupe que d'1 chose à la fois

Open–closed

Héritage, composition ➔ ajout mais pas de retrait, et mécanisme de protection

Liskov substitution

Un type spécialisé (heritage) doit pouvoir remplacer un type ancêtre

Interface segregation

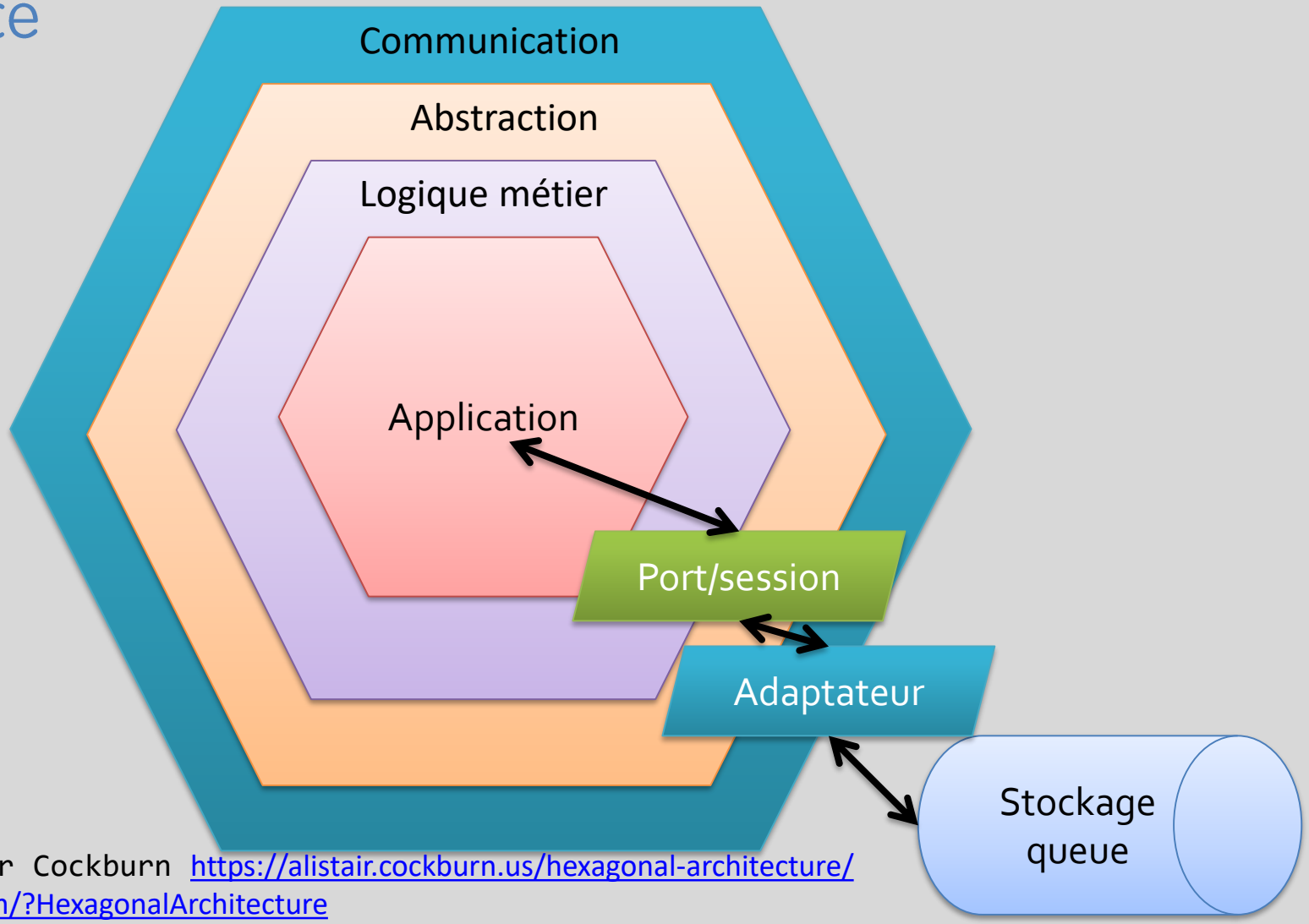
Un objet de depend pas d'une interface qui ne le concerne pas

Dependency inversion

Vision hexagonale (l'objet le plus conceptuel ne connait pas le detail de l'objet concret), tout est passé par l'interface, pas de surcharge de classe concrete

Organisation fonctionnelle, le modèle hexagonal

Le modèle hexagonal préfigure l'architecture par microservice



2005, Alistair Cockburn https://alistair.cockburn.us/hexagonal-architecture/
http://wiki.c2.com/?HexagonalArchitecture

Organisation fonctionnelle, le modèle hexagonal

Les couches externes importent les couches internes, jamais le contraire

Chaque couche définit ses points d'intégration, dits "port", sous formes de types/classe – qui vont permettre la mise en place d'objets de service.

On sépare les paquets.

- pour les consommateurs – avec le point d'exposition (par ex. http)
- pour les fournisseurs – avec la session (exemple: database/sql/driver)

Chaque couche peut se composer de plusieurs paquets, en particulier les hexagones

Un paquet ne fournit jamais plusieurs couches ou adaptateurs

Données

Le stockage est défini par la logique intrinsèque des données (*data-driven*)

Les autres stockages sont définis par les usages (*query-driven*)

On sépare les services avec un mode CQRS =
Command Query Responsibility Segregation

Par ex:

service d'acquisition et modification: intégrité, cohérence, normalisation

services de restitution: performance, scalabilité ⇒ dénormalisation

service de transformation: passer de l'un à l'autre, *streaming*

Event Sourcing: la source d'autorité est le flux d'événements complets multi-services

Données structurées (SQL)

Les modèles de données sont transactionnelles. ➔ moniteur transactionnel

La donnée est décrite à travers une relation et une entité (E/R) – avec des formes normales pour décrire les relations (Boyce-Codd formes normales).

1NF: tous les attributs de toute relation, ayant par définition une clé, sont atomiques (isolés)

pas de listes, pas de NULL

une adresse peut être non atomique: numéro, rue, code postal

2NF: la clef entière peut être un composite - une jointure. Aucun attribut n'appartenant pas à la clef ne dépend transitivement d'un sous ensemble de la clef – on obtient une table étoile.

3NF: on factorise par rapport à 2NF – il s'agit de faire une table flocon.

Limitations:

Le principe de la normalisation n'est pas associée à la performance

La normalisation permet de décrire les contraintes d'intégrités

Données déstructurées (NoSQL)

Dans une base relationnelle (il y a un moniteur transactionnel)

- ACID: Atomic updates, Consistency, Isolation, Durability
- Des répliques en lecture contiennent des copies potentiellement non intègres, et toujours en (léger) décalage avec la version de référence ➔ contrainte Causale prise en charge par le moniteur transactionnel

Dans une base non-relationnelle, généralement répartie, les répliques peuvent être toutes incohérentes, et c'est assumé

Avec les systèmes de services indépendant, l'intégrité est limitée à chaque service. Le système global n'est plus que *eventually consistent*

Les efforts sont à porter sur la réduction de la fenêtre d'incohérence ➔ contrainte Causale

Théorème CAP de Brewer (Consistency / Availability / Partition tolerance): *aucun système réparti ne peut garantir à la fois pour tous ses noeuds la cohérence, la disponibilité, et la résistance au partitionnement)* https://en.wikipedia.org/wiki/CAP_theorem.

Deux à la fois sont possibles, choisir la solution selon la paire choisie

Configurations du code ➔ décrit l'évolution

Comment déployer des versions différentes de code simultanément ?

Feature flag (aka Feature Toggle): un mécanisme externe à l'application qui définit des variables lors de l'exécution des requêtes entrantes. A/B ou MVT (multivariant)

Composants:

Un stockage des règles de détermination des variables, voire une UI

stockage des règles de classement des requêtes, de l'historique des évolutions, etc

activation/désactivation, politique de rollout

conservation des révisions, lien vers l'observabilité, documentation

Un proxy, interne ou externe, qui répartit les requêtes en groupes, ajoutant des **headers appropriés**: géociblage, authentification, tirage statistique, etc

Une bibliothèque qui détermine les flags actifs et la configuration de l'appli:

au démarrage, puis en cours de fonctionnement par sondage du stockage,

pour le support de la reconfiguration sans déploiement

L'application des configurations – modulaire et progressive : interne, beta, puis rampe jusqu'à 100%. Désactivation en cas de problème.

https://martinfowler.com/articles/feature-toggles.html

Cache

Les caches servent à limiter la consommation de ressources dans les applications, pour les protéger et permettre leur scalabilité horizontale.

Caches de données

- Hors application: dans la base de données, répliques en lecture pour décharger le serveur d'écriture. Problème: le retard de mise à jour.
- Hors processus: serveurs dédiés: Memcached, Redis, AWS Elasticache
  - Résilience et scalabilité par **sharding** et réplication. Notion de hachage cohérent
- En processus: cache statique

Filesystem : avoir une vision des accès aux fichiers (et décider où mettre le cache applicatif)

Il existe une hiérarchie des caches – on peut construire un benchmark théorique (Amdahl).

Patron de conception Cloud : RPC

Débuts du calcul réparti: RPC – Remote Procedure Call

- L'appelant sérialise son message dans un format réseau
- Le bibliothèque gère l'appel et la réception des résultats retour
- Elle désérialise les résultats et revient, le tout synchrone
- Peut être explicite (RPC) ou transparent avec des stubs générés (Corba)

Avantage: concept très simple, implémentations nombreuses: XML-RPC, SOAP, JSON-RPC

Problèmes:

- Compromis lisibilité (formats texte) vs performance (formats binaires)
- Résistance aux pannes si centralisation du service?
- Couplage fort entre appelant et appelé

Le choix actuel gRPC (car http/2). (google RPC)

https://app.swaggerhub.com ➔ équivalent fonctionnel de github pour la définition des API/REST

Patron de conception Cloud : bus événementiel

Modèle adapté aux applications découplées, asynchrone

Chaque service publie des messages et n'attend pas de réponse par défaut

- Possibilité de surcouche implémentant un modèle requête/réponse
- Mais plus de modèle "fonction distante"

Chaque service s'abonne (pub/sub) à des files d'attente (des "sujets") de son choix

Les formats de message sont définis extérieurement aux services, souvent en protobuf

Gestion de la garantie de délivrance: *at-least-once, at-most-once, exactly-once, ordered*

- Les consommateurs ont plus ou moins de charge selon les garanties

Gestion de la tolérance aux pannes et de la persistance

- Quelques files Apache Kafka, NATS, NSQ, RabbitMQ
- Possible sans serveur séparé, avec ZeroMQ

Patron de conception fonctionnelle

Un micro-service est un service traitant un seul contexte borné au sens DDD (data driven design).

- Il est la source de vérité sur ses données
- Lorsqu'il a des représentations de données d'autres contextes, la structure est la sienne

Intégration de micro-services

- Le paramétrage fin de la connectivité: IP, port, règles d'accès, etc, ne fait pas partie du "métier" d'un service
- Définir des paramètres HTTP par défaut, limités à une seule connexion
- Associer au service un *sidecar proxy* qui sera le seul à communiquer avec lui, et aura tous les accès, et sera en coupure avec l'extérieur
- Le groupe de conteneurs (pod) associe les deux
- L'orchestrateur (*service mesh*) agence les proxies sans connaître les applications
- Il peut même fournir le coupe-circuit (cf *supra*) et un degré d'observabilité sans code applicatif

Observabilité

Les 4 piliers de l'observabilité

Journaux: enregistrer du texte avec des paramètres

Métriques: compter, mesurer, répartir des grandeurs

Traces: enregistrer une pile d'appels virtuelle au-travers de services multiples

Et un quatrième: les alertes, transversales aux 3 autres.

Observabilité et supervision (*monitoring*)

La supervision vise à observer des indicateurs d'un système, de toutes natures, avec pour but d'en détecter les anomalies à partir de manifestations extérieures (ex. charge CPU, espace disque saturé)

"Qu'est-ce qui ne va pas ? // Cybersécurité

L'observabilité vise à identifier l'état d'un système pour comprendre la raison des anomalies

Journalisation

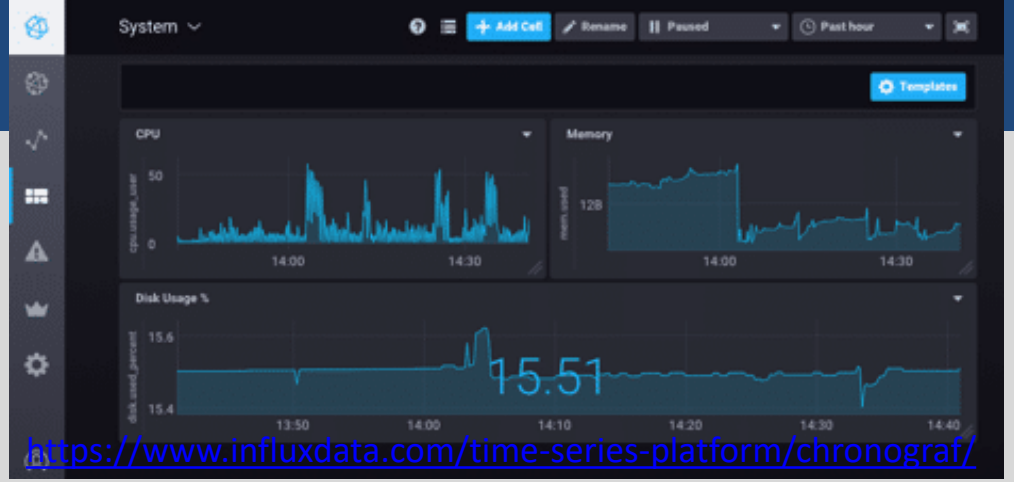
Déclenchement: lors d'un événement 2 grandes approches: texte non structuré sur stdout ou stderr, typique des applications conteneurisées en modèle *12-factor*

journaux structurés, inspirés par syslog (RFC 5424), au départ avec des niveaux (*severity*) et catégories (*facility*), depuis étendus avec des étiquettes

Problèmes liés à la journalisation:

- Volume
- Sérialisation
- Données nominatives (RGPD)
- Couplage de charge et rétropression
- Cybersécurité

Les métriques



https://www.influxdata.com/time-series-platform/chronograf/

Déclenchement: événementiel ou périodique

Dans les API modernes (graphite, statsd), le code émet une donnée numérique (échantillon) avec un chemin (ex. cpu.0.load) avec une valeur et des étiquettes

- Le collecteur construit automatiquement les séries à partir des chemins
- Dans les API plus anciennes, les séries doivent être prédéfinies

3 principaux types de données:

- Gauge: c'est un niveau. Le collecteur conserve un agrégat, généralement la dernière valeur obtenue sur une période
- Compteur: suite monotone pouvant seulement être réinitialisée. Le collecteur conserve normalement la dernière valeur obtenue sur une période
- Histogramme / distribution: c'est une valeur pour laquelle le collecteur conserve des séries statistiques, pour extraire des agrégats: moyenne, médiane, mode, minimum, p95, p99, etc. Tous les systèmes ne supportent pas tous les agrégats

Les traces

Déclenchement sur requête

Notion de span (=profiler=)

- À l'entrée dans une période observable (une fonction), début d'une étendue (*span*)
- À la sortie, fin de l'étendue et collecte de sa durée, et de données comme étiquettes (*defer*).

Les étendues forment un arbre depuis une étendue racine

- Les appels de fonction/méthode créent des enfants successifs
- La propagation de l'étendue racine permet et son inclusion dans les requêtes sortantes permet de créer une trace répartie dans une archi micro-services

Corrélation: injecter l'identification de l'étendue dans le logger et/ou le client de métriques permet de corréler journaux, métriques, et traces, pour l'observation la plus utilisable

Le stockage des données d'observation

Les données temporelles de l'observabilité posent un problème de montée en charge dans les bases de données classiques:

- La PK naturelle est la séquence temporelle ou l'incrément monotone des enregistrements
  - ⇒ les mises à jour ont toujours lieu dans les mêmes pages actives
  - ⇒ en cas de sharding, utilisation d'un seul shard
  - ⇒ contention de verrouillage et limitation des performances
- Beaucoup plus d'écritures que de lectures
- Besoin d'agrégation (*rollup*) des métriques, et pb d'expiration des données (effacement des anciennes observations)

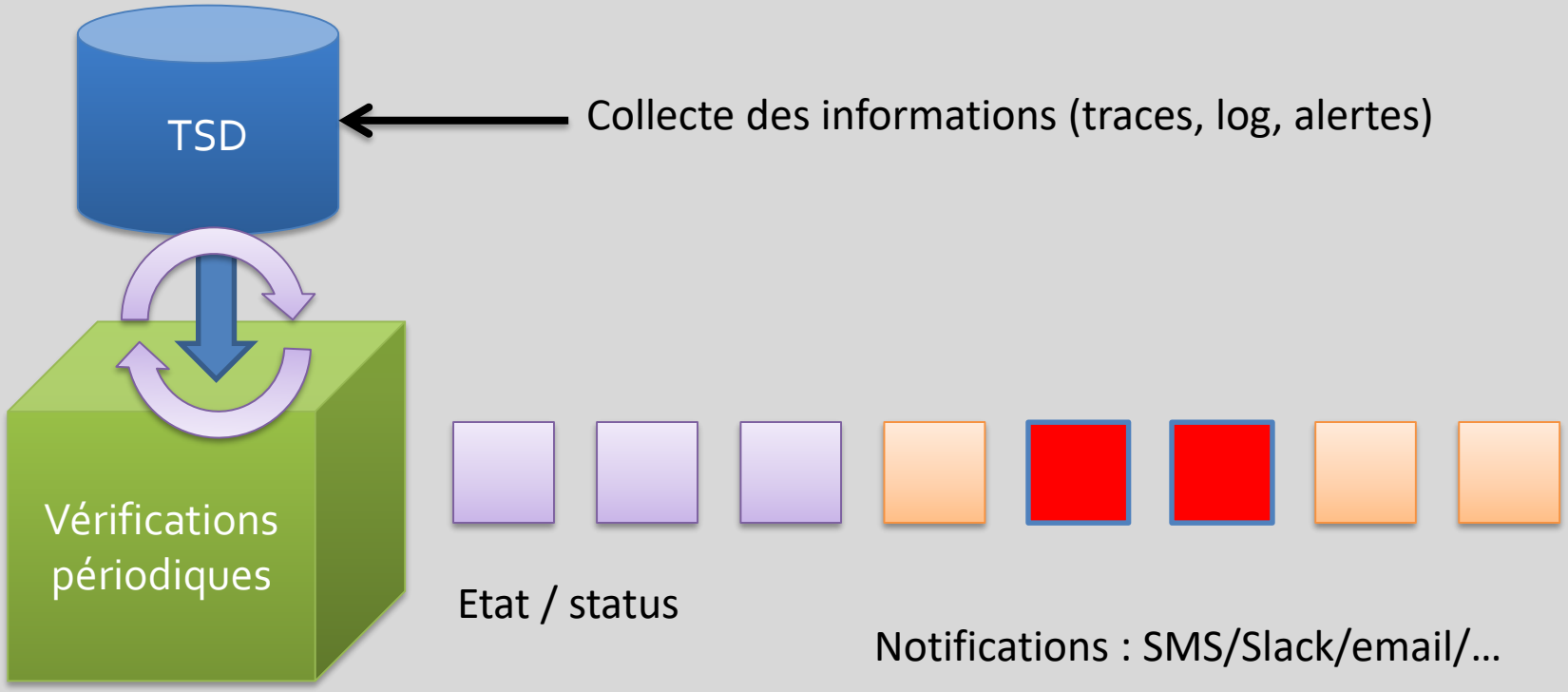
Les bases spécialisées sont optimisées – avec une capacité à gérer les tranches de temps:

- TSD = Time Series DataBase
- Outils libres: **InfluxDB,** Prometheus, Graphite, RRDtool, OpenTSDB, Druid, M3DB, Victoria Metrics, Akumuli TimescaleDB (extension Postgres), FaunaDB, DolphinDB
- Choisir plutôt la simplicité d'intégration, le niveau de support, etc.

Les alertes – avec une TSD

Il s'agit de pouvoir notifier de manière ciblée

Grammaire de l'observabilité



Chiffrage / Devisage

Dans les activités du futur consultant HPC, il peut y avoir celle du chiffrage/devisage

Normalement, il faut une expérience de 3 ans (au moins), quand on est junior dans le domaine d'intérêt, pour construire un budget

Objectifs Prix / Réalisation

Construire un prix ➔ élément déterminant de la vente !

Limiter les risques : connaitre la durée, l'effectif et la contingence.

Durée / Effectif = en JH

Qq métriques = 1 mois 35h = 20 JH en moyenne

La qualification est décrite en 4 niveaux :

- Expert - Xpert
- Senior - P
- Confirmé - Mid
- Junior - A level

Les qualifications

Junior

Sortie d'école

Rôle/Responsabilité : les activités simples/non engageantes contractuellement/pas relations externes

Confirmé

1 a 2 ans

Rôle/Responsabilité : toutes les activités simples sans relations externes/activité moyennement complexes

Senior

3 à 5 ans

Rôle/Responsabilité : toutes les activités avec acteurs internes/externes, supervise les Juniors

Expert

›6 ans

Rôle/Responsabilité : toutes les activités internes/externes, très complexes, supervise Juniors/Confirmés

Les Risques

Outil Jalons

- Permet de quantifier la réussite (ou l'échec)
- Permet d'avoir une approche pragmatique (go/no-go – réorientation)

Outils Pilotage

- Comité de pilotage / Comité de suivi
- Méthodologie SCRUM Agile, Cycle en V

Outil RACI

- Décrire les responsabilités

Contingence

- Intégrer le risque dans le prix

Jalons

Les jalons fixes = tautologie - gabarit date/niveau de fonction

Par ex : capacité à lire un fichier test

Les jalons flottants = idiot - =des objectifs

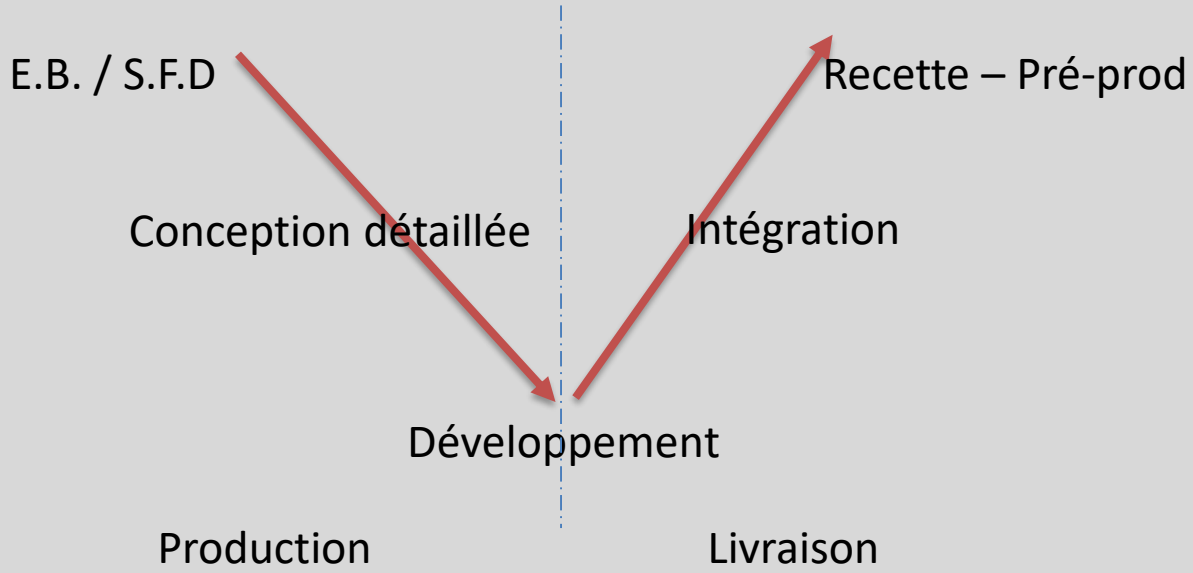
La backlog mixe les activités et les jalons

Ils sont nécessaires au pilotage

Pilotage

Le cycle en V :

Chaque jalon de production est en correspondance avec un jalon de livraison.



Agile – SCRUM

Méthode itérative – construire « en réaction » au cycle en V

Le cycle repose sur 1 EB claire – avec un client qui valide ses besoins.

➔le client peut avoir du mal à définir ses besoins

➔le coût de cette définition peut être ventilé en plusieurs étapes

Dans le cycle en V : le développement « consomme » 1 CP

➔pourquoi ne pas auto-organisé un consensus à la place du CP?

➔concept de Story / Backlog / Sprint

Dans le cycle en V : l'identité du produit n'est pas « pilotée » (car il n'y a pas de jalon pour ça) – en Agile il y a 1 PO

➔le poste de Directeur de Produit ou Product Owner (PO) est une réponse à ce défaut du cycle en V

Les responsabilités

Matrice RACI = Tableau nominatif

Définir au début du projet le périmètre précis de l'intervention et les zones de responsabilité

RACI = **Responsible, Accountable, Consulted, and Informed**. ➔ en français : Responsable(s), Autorité Superviseur (1 seul par ligne), Consulté (Expert), Informé (non décisionnaire)

Attention au faux ami français A et R peuvent être inversé !

RACI français avec : Responsable (1 seul) / Acteur / Consulté / Informé

RACI (anglais)	Jean	Youcef	Léa	Pierre
EB	R	R	A	
SFD	R	A		R
Recette		A	C	R

Prise en compte de la livraison

Si livraison échelonnée = micro-service avec ou sans BUS

Si Livraison totale = monolithe (le micro-service coute plus cher)

il y a une hybridation possible → plusieurs monolithes (➔micro services)

➔attention les micro-services amènent le métier d'urbaniste (Définition du POS)

Les coûts

Si définition d'un format de fichier ➔définir le format = SFD

Si usage d'une DB (SQL ou NoSQL) ➔définir les modèles d'initialisation (=schéma), la reprise des données, les sauvegardes, les rétentions ➔ coût

Si usage d'une bibliothèque (OpenSource ou pas) ➔coût d'évolution, si la bibliothèque évolue avant la fin du projet?

Test/Recette ➔ce qui coute est la confrontation négative au client

La documentation

Utilité

- Support de conviction pour le projet
- Aide à l'exploitation
- Aide à la maintenance
- Capitalisation pour des futures ventes

Coût

- Ecriture
- Correction
- Validation
- Archivage

➔C'est l'élément nécessaire à la capitalisation

Contingence / Garantie / Marge

La contingence : un surcout pour un Plan B – Risque interne

La Garantie (SLA = Service Level Agreement) est Contractuelle

Marge = le bénéfice de l'opération (Secret)

Ces surcout sont cumulatifs

Capitalisation

Comment capitaliser?

Si le produit est architecturé en Micro-Service

Si les plans de tests sont lisibles

Si le code est commenté

Si les bibliothèques sont maintenues

Si les schéma de DB / Format de fichier sont « équilibrés » (pas d'effet de GOD class)

Si les codeurs sont disponibles

Le mode de livraison

3 environnements :

DEV / Pré-prod / PROD : stricte séparation des environnements

➔DEV : chaque DEV a la responsabilité de son environnement, les fichiers de test sont supervisés par le client

➔Pré-prod : espace de qualification, est « similaire à la PROD » - données d'intérêts anonymisées – ici le DEV ne peut pas faire ce qu'il veut // espace qui peut servir de plan B pour la PROD

➔PROD : espace de production – reçoit les livraisons

~~Cycle de vie du code~~ Ce qu'on livre

Build

- SFD – Recette (=qualification en préprod? En prod?)
- Livraisons Dev / Préprod / Prod

~~Run~~

- ~~PRA~~

~~MCO~~

~~MCS~~

~~On peut livrer plus si le contrat le précise (ie contrat de performance)~~