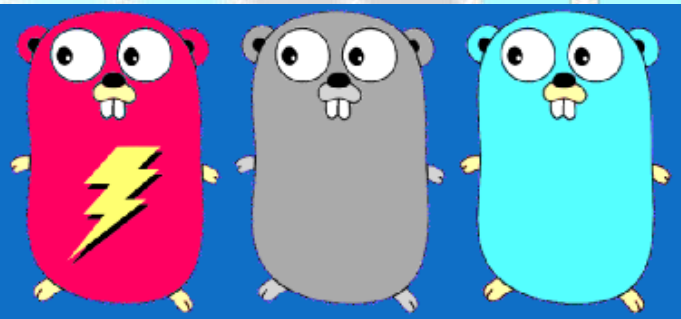
UNIVERSITÉ DE VERSAILLES ST-QUENTIN-EN-YVELINES

université PARIS-SACLAY

Génie Logiciel pour le Calcul Scientifique



#3

06/12/2024

jean-michel.batto@cea.fr

cea

https://gogs.eldarsoft.com/M2_IHPS

CSP – Aout 1978 – Tony Hoare

Fichier : tony_hoare-CSP1978.pdf

Programming Techniques
S. L. Graham, R. L. Rivest Editors

Communicating Sequential Processes

C.A.R. Hoare
The Queen's University
Belfast, Northern Ireland

This paper suggests that input and output are basic primitives of programming and that parallel composition of communicating sequential processes is a fundamental program structuring method. When combined with a development of Dijkstra's guarded command, these concepts are surprisingly versatile. Their use is illustrated by sample solutions of a variety of familiar programming exercises.

Key Words and Phrases: programming, programming languages, programming primitives, program structures, parallel programming, concurrency, input, output, guarded commands, nondeterminacy, coroutines, procedures, multiple entries, multiple exits, classes, data representations, recursion, conditional critical regions, monitors, iterative arrays
CR Categories: 4.20, 4.22, 4.32

grams, three basic constructs have received widespread recognition and use: A repetitive construct (e.g. the **while** loop), an alternative construct (e.g. the conditional **if..then..else**), and normal sequential program composition (often denoted by a semicolon). Less agreement has been reached about the design of other important program structures, and many suggestions have been made: Subroutines (Fortran), procedures (Algol 60 [15]), entries (PL/I), coroutines (UNIX [17]), classes (SIMULA 67 [5]), processes and monitors (Concurrent Pascal [2]), clusters (CLU [13]), forms (ALPHARD [19]), actors (Hewitt [1]).

The traditional stored program digital computer has been designed primarily for deterministic execution of a single sequential program. Where the desire for greater speed has led to the introduction of parallelism, every attempt has been made to disguise this fact from the programmer, either by hardware itself (as in the multiple function units of the CDC 6600) or by the software (as in an I/O control package, or a multiprogrammed operating system). However, developments of processor technology suggest that a multiprocessor machine, constructed from a number of similar self-contained processors (each with its own store), may become more powerful, capacious, reliable, and economical than a machine which is disguised as a monoprocessor.

In order to use such a machine effectively on a single task, the component processors must be able to communicate and to synchronize with each other. Many methods of achieving this have been proposed. A widely adopted method of communication is by inspection and updating of a common store (as in Algol 68 [18], PL/I, and many machine codes). However, this can create severe problems in the construction of correct programs and it may lead to expense (e.g. crossbar switches) and

Questions

- ➔Qui est l'auteur et quelles sont ses qualités/reconnaissances à la date de l'article?
- ➔Quels sont les problèmes adressés par l'article?
- ➔Quelles sont les solutions?

Les 7 propositions

Exécutions parallèles de commandes avec démarrage synchrone et arrêt synchrone (au dernier élément)

Echanges entre processus avec des E/S élémentaires via des **channels** bloquants

Ceux-ci ont des actions déterminées en E/S sur l'état du processus

Le concept de barrière non-déterministe

Commande d'E/S avec barrière

Commande d'E/S dans les boucles

Capacité à choisir l'action en fonction du message

Les 7 propositions selon une IA

INTERACTIF

nvidia/llama-3.1-nemotron-70b-instruct

https://build.nvidia.com/nvidia/llama-3_1-nemotron-70b-instruct

- Quelles sont les 7 propositions de Tony Hoare concernant CSP ?

Séquentialité des Processus

Communication par Canaux

Synchronisation Implicite

Absence de Partage de Mémoire

Détermination du Récepteur

Input Guard (Gardes d'Entrée)

Décomposition Hiérarchique

CSP : Communicating Sequential Process

par Tony Hoare, 1978

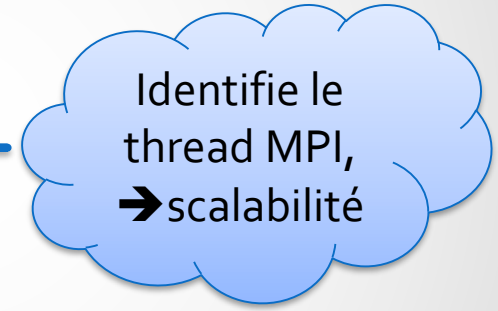
- Communicating : MPI_Init+MPI_Ssend
- Sequential process →objets concurrents
  - pthread en C, class thread en C++11
- Sequential process
- →blocage sur lecture/écriture dans une file // buffer
  - MPI_Recv / MPI_Ssend

CSP : Communicating Sequential Process

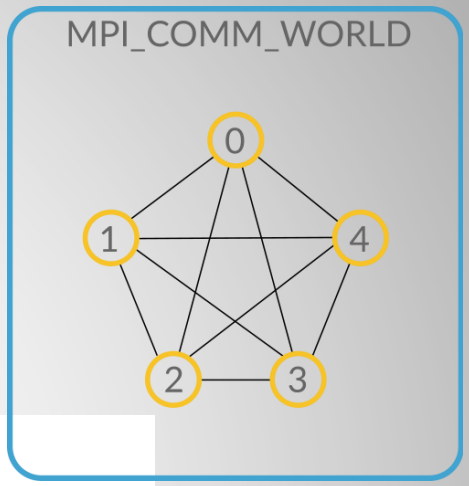
- Fondamental dans les machines multicoeurs.
- Notion de canal de liaison en MPI.
- Permet de faire des actions 'en tâche de fond'.
- Ne précise pas le code de traitement exécuté sur chaque nœud.
- Dans les codes MPI – on utilise, par usage, le même code sur chaque nœud ➔ scalabilité.

Quelques fonctions MPI à connaitre

- API synchrone&asychrones avec processus numérotés (=connus)
  - MPI_Init()
  - MPI_Abort()
  - MPI_Get_processor_name()
    - MPI_Comm_size()
    - MPI_Comm_rank()
  - MPI_Send(), MPI_Ssend()
  - MPI_Recv()
  - MPI_Finalize()
  - MPI_Bcast() / MPI_Scatter()
  - MPI_Reduce()
  - MPI_Allreduce()



MPI 1/4



- MPI_Init() /MPI_Finalize()
- MPI_Abort()

```c
#include <mpi.h>
int main(int argc, char *argv[])
{
    MPI_Init(NULL, NULL);
    MPI_Abort(MPI_COMM_WORLD, 112); // in Europe 112, for emergency
    /* No further code will execute */
    MPI_Finalize();
    return 0;
}
```

- MPI_Get_processor_name()
  - ➔résultat de gethostname, uname, ou sysinfo

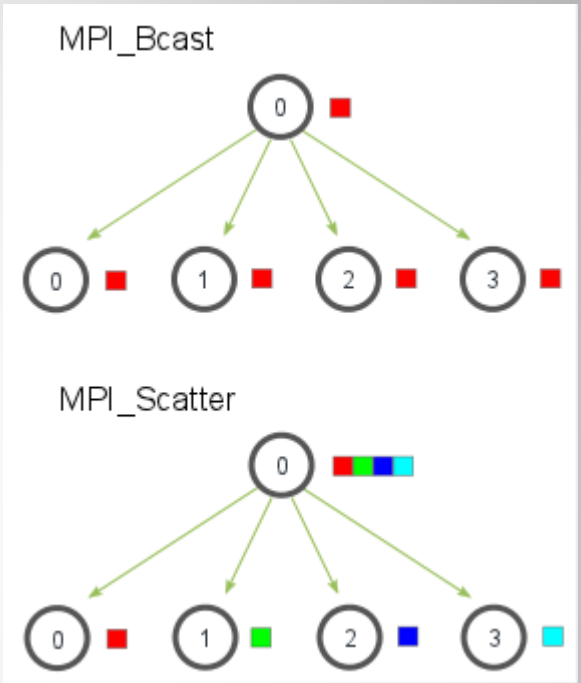
MPI 2/4

- MPI_Comm_size() ➔ effectif des processus
- MPI_Comm_rank() ➔ rang du processus

- MPI_Send() ➔ communication point à point
- MPI_Ssend() ➔ communication point à point, synchrone
- MPI_Recv() ➔ réception point à point, responsable du buffer data

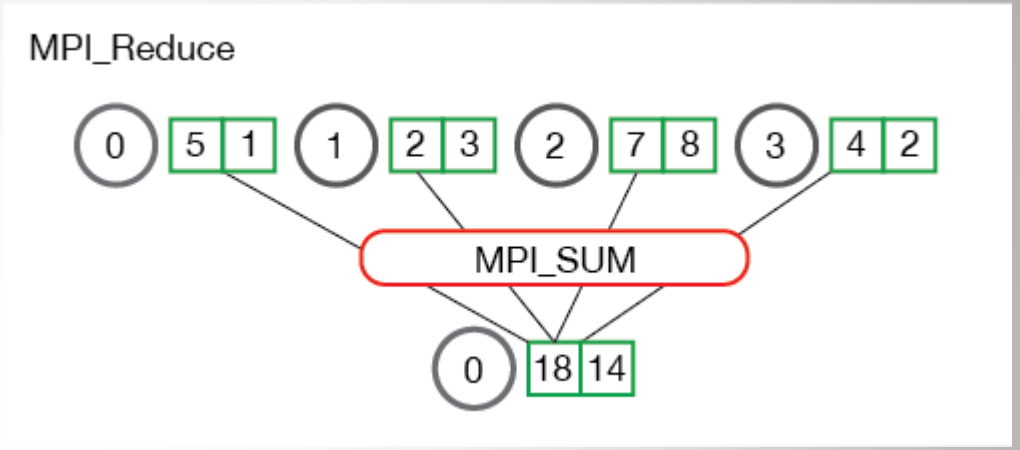
MPI 3/4

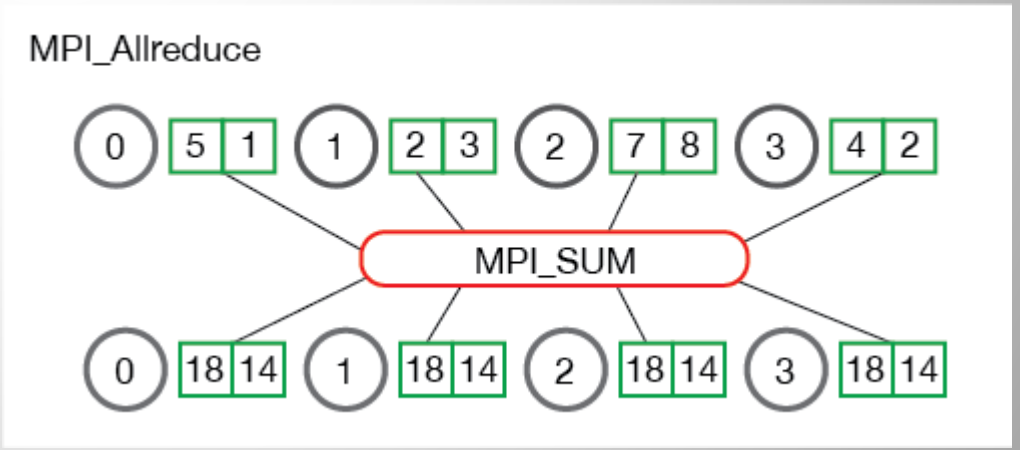
- MPI_Bcast() / MPI_Scatter()
- Instructions avec fonctions
  - MPI_Reduce() ➔root
  - MPI_Allreduce() ➔partage



MPI 4/4

- MPI_Bcast() / MPI_Scatter()
- Instructions avec fonctions
  - MPI_Reduce() ➔root
  - MPI_Allreduce() ➔partage





Les 23 patterns du GoF

- Elements of Reusable Object-Oriented Software
- – Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides
- Chez Addison-Wesley, 1995.
- Ces patterns trés célèbres ont été conçus par 4 informaticiens surnommés le "Gang of Four" (d'où le terme GoF pour ces patterns).
- Ils proposent des solutions élégantes, et toujours différentes pour résoudre différents problèmes récurrents rencontrés par les architectes logiciels.

Utilité de l'approche du GoF

- Patterns et conception : standardiser la conception.
- Formalisation de savoir-faire.
- Augmenter le niveau d'abstraction ➔ Eviter de construire des objets trop adhérents du réel ➔ simplifie le réemploi.
- Arbitrage entre flexibilité et performance ➔ l'adaptation aux évolutions.

GoF

- En 3 séances (7, 7, 9)
- Objectif : apprendre à bien utiliser 2 patterns

Les patrons / patterns

Les 23 patterns décrits :

<table>
<tr><th></th><th>Creational</th><th>Structural</th><th>Behavioral</th></tr>
<tr><td rowspan="2">Class</td><td rowspan="2">Factory Method</td><td rowspan="2">Adapter (class)</td><td>Interpreter</td></tr>
<tr><td>Template Method</td></tr>
<tr><td rowspan="9">Object</td><td>Abstract Factory</td><td>Adapter (object)</td><td>Chain of Responsibility</td></tr>
<tr><td>Builder</td><td>Bridge</td><td>Command</td></tr>
<tr><td>Prototype</td><td>Composite</td><td>Iterator</td></tr>
<tr><td rowspan="5">Singleton</td><td>Decorator</td><td>Mediator</td></tr>
<tr><td>Facade</td><td>Memento</td></tr>
<tr><td>Flyweight</td><td>Observer</td></tr>
<tr><td rowspan="3">Proxy</td><td>State</td></tr>
<tr><td>Strategy</td></tr>
<tr><td>Visitor</td></tr>
</table>

Typologie

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

Réemploi des objets

Spécifier la classe d'un objet explicitement	Abstract Factory
	Factory Method
	Prototype
Coder explicitement les comportements	Command
	Chain of Responsibility
Prise en compte des dépendances	Abstract Factory
	Bridge
Dépendance sur les représentations et les objets	Abstract Factory
	Memento
	Bridge
	Proxy
Dépendance algorithmique	Strategy
	Builder
	Iterator
	Template Method
	Visitor
Couplage léger	Facade
	Mediator
	Observer
	Command
	Abstract Factory
	Bridge
Sous-classe pour étendre le comportement	Bridge
	Composite
	Decorator
	Chain of Responsibility
	Strategy
Incapacité à altérer la classe ancêtre	Visitor
	Decorator
	Adapter

Dans notre étude : le réemploi

Spécifier la classe d'un objet explicitement	Abstract Factory
	Factory Method
	Prototype
Coder explicitement les comportements	Command
	Chain of Responsibility
Prise en compte des dépendances	Abstract Factory
	Bridge
Dépendance sur les représentations et les objets	Abstract Factory
	Memento
	Bridge
	Proxy
Dépendance algorithmique	Strategy
	Builder
	Iterator
	Template Method
	Visitor
Couplage léger	Facade
	Mediator
	Observer
	Command
	Abstract Factory
	Bridge
Sous-classe pour étendre le comportement	Bridge
	Composite
	Decorator
	Chain of Responsibility
	Strategy
Incapacité à alterer la classe ancêtre	Visitor
	Decorator
	Adapter

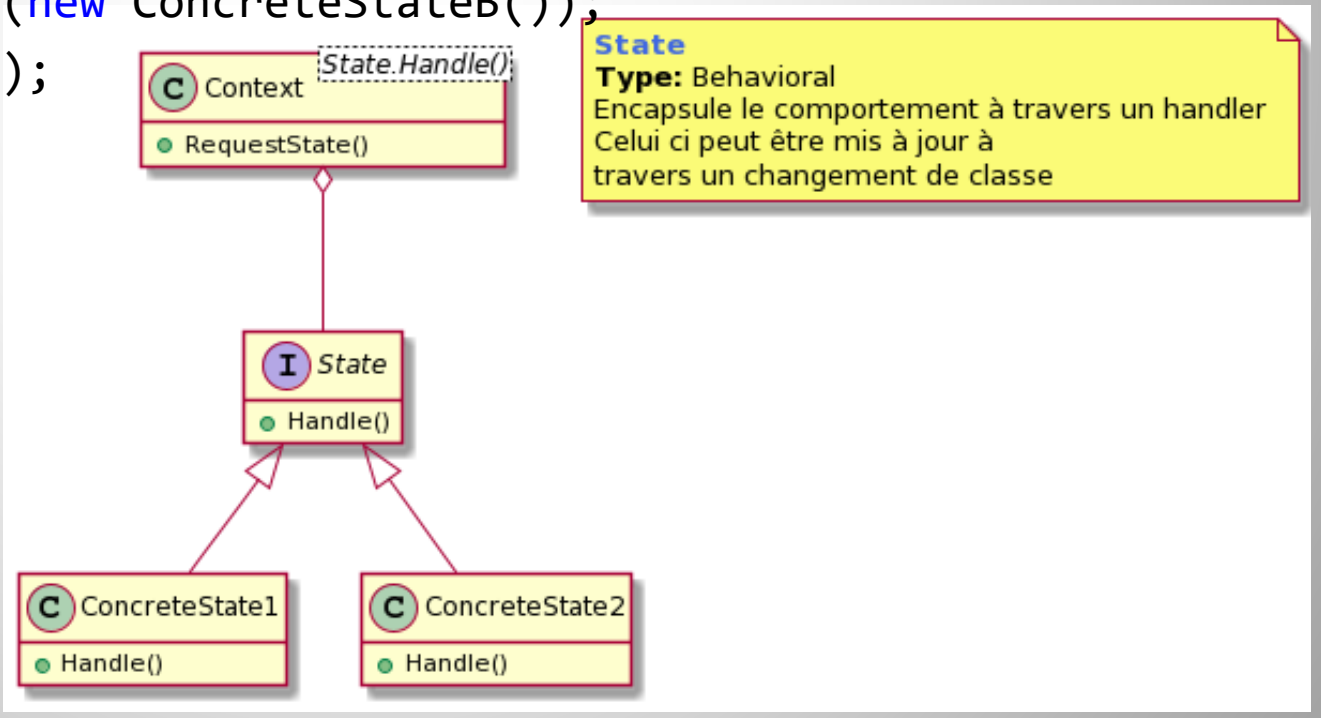
Objectif de ce cours

Typologie	Nom du Design Pattern	Ce qui doit être ajusté	verbe	composition sous jacente	mélange de classes dynamique ?
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	état=verbe	structure	filtrage vtab
	Strategy	un algorithme indépendant du client	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

Behavioral / State

- https://godbolt.org/z/TxojreTaY

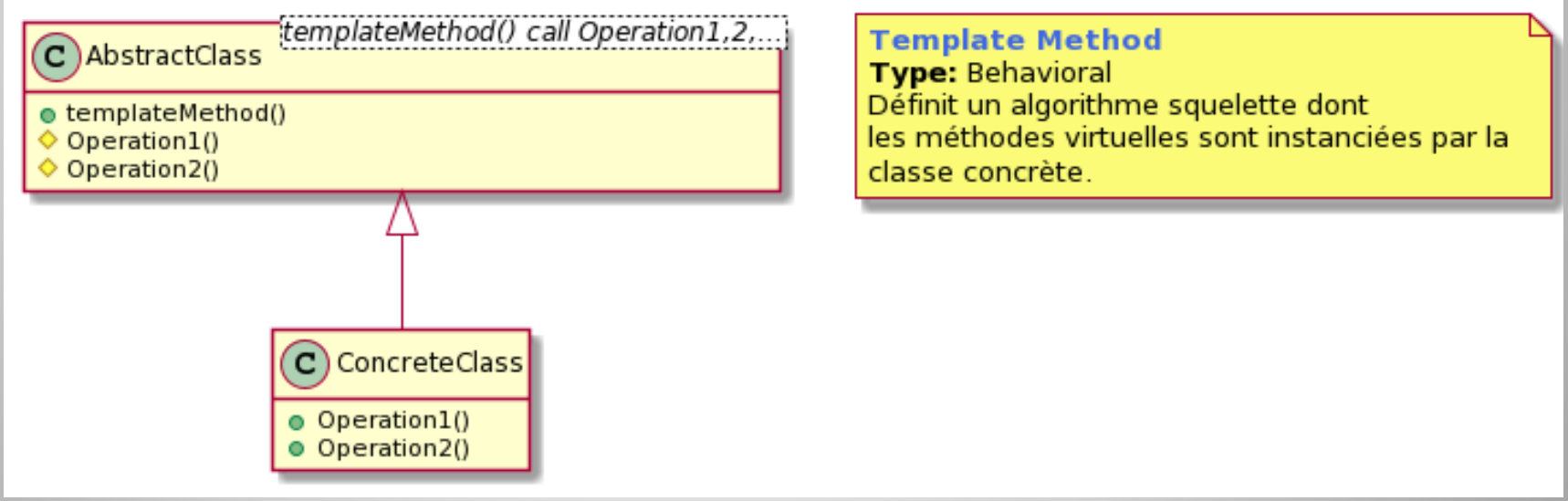
```
int main() {
    Context* context = new Context();

    context->setState(new ConcreteStateA());
    context->Request();

    context->setState(new ConcreteStateB());
    context->Request();
}
```



Behavioral / Template Method

- https://godbolt.org/z/8vhM5PaoG

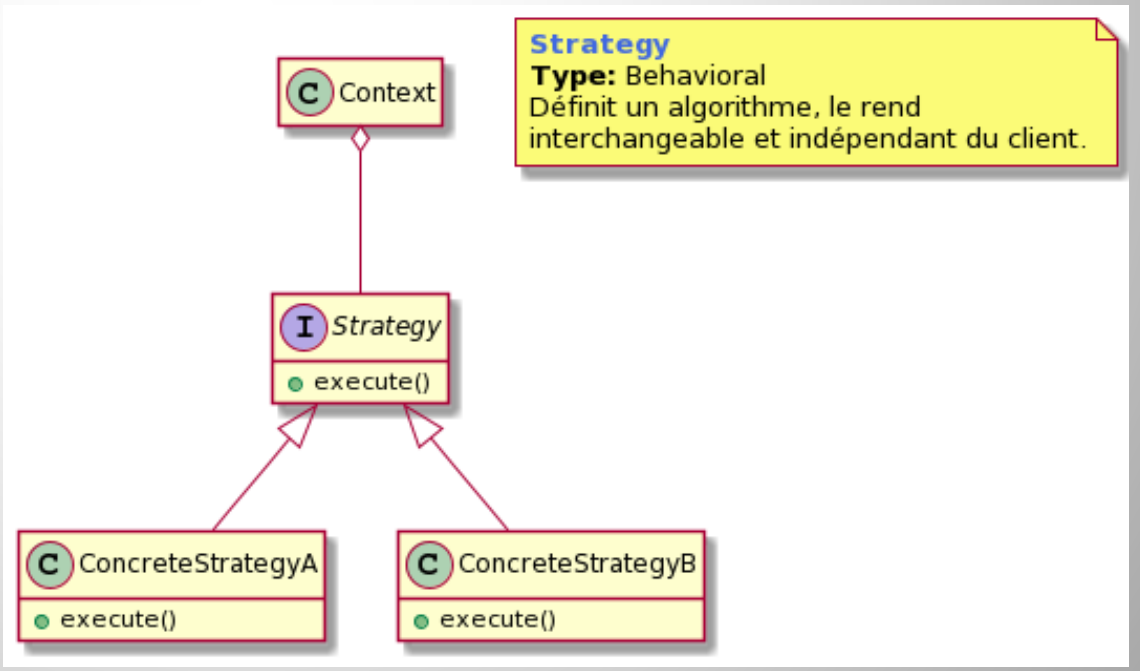
```
int main() {
  AbstractClass *templateClass = new ConcreteClass();
  templateClass->templateMethod();
}
```



Behavioral / Strategy

- https://godbolt.org/z/zbaK3Wr5T

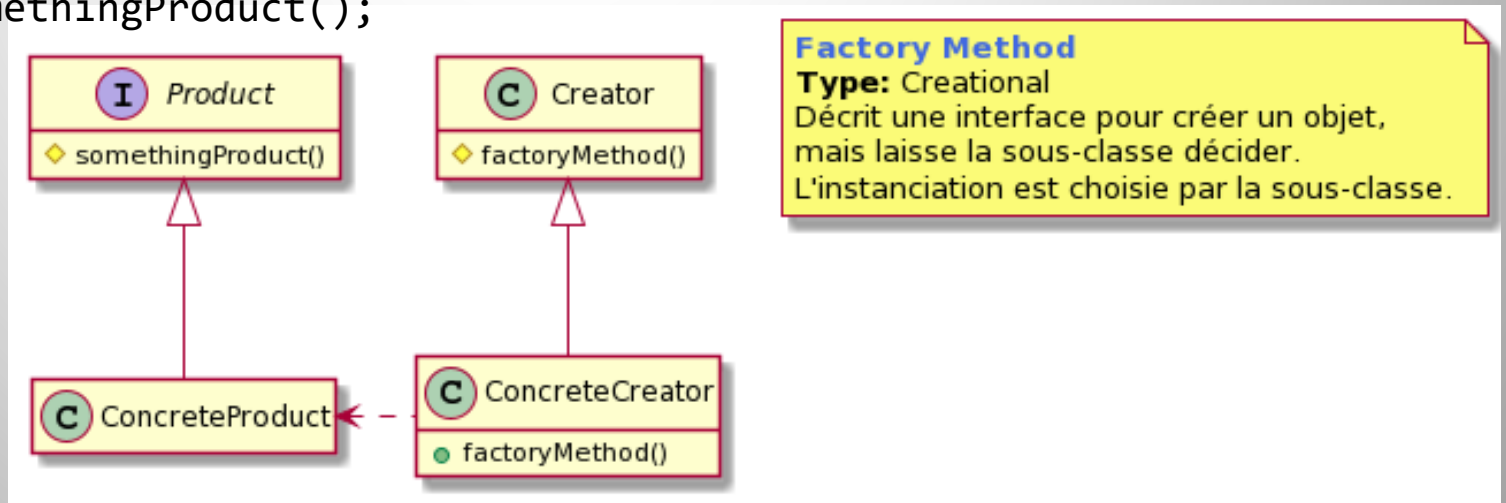
```
int main() {
    Context context(new ConcreteStrategyB());
    context.execute();
}
```



Creational / Factory Method

- Permettre à une classe de créer des objets dont elle ne connaît pas la classe
- https://godbolt.org/z/7WPYfzvsa

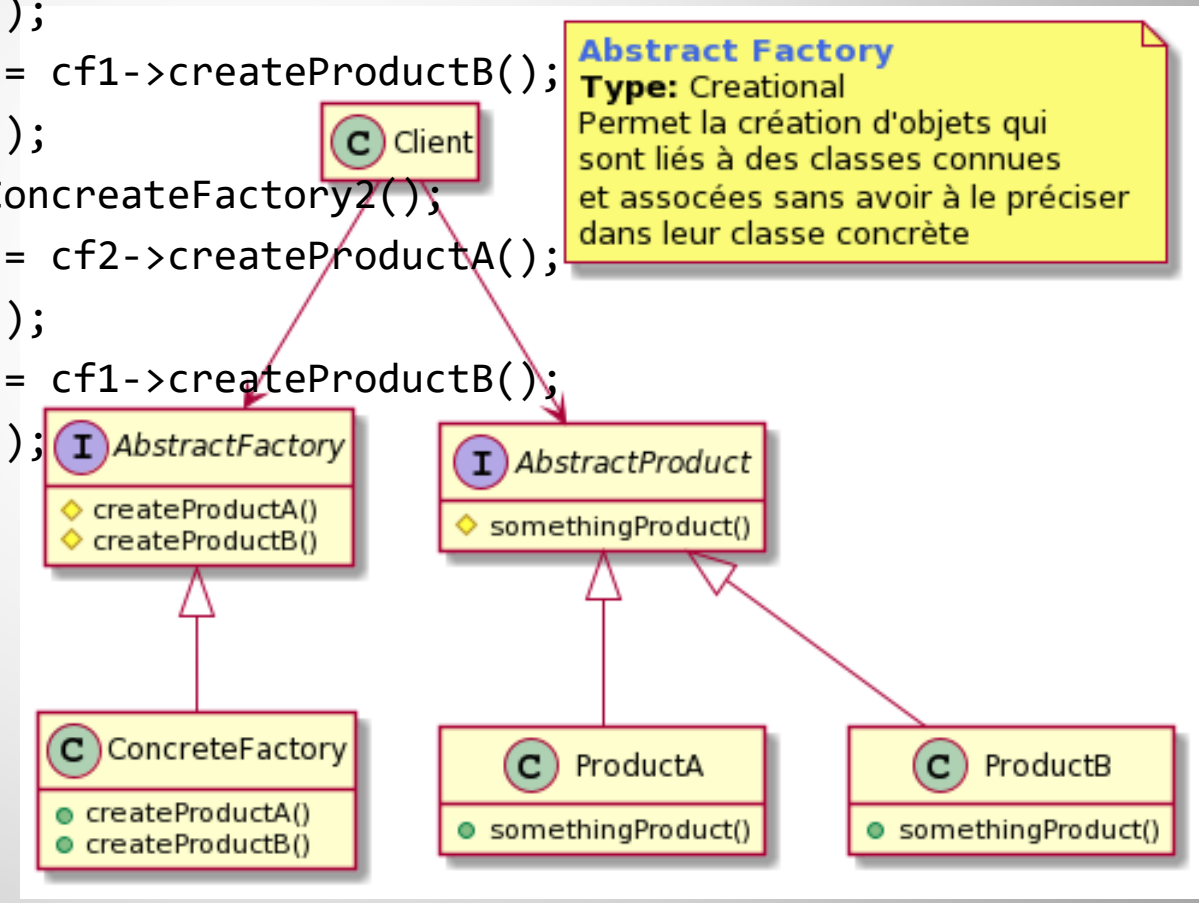
```
int main(int argc, char* argv[]) {
  Creator *creator = new ConcreateCreator(ConcreateCreator::A);

     Product *productA = creator->factoryMethod();
  productA->somethingProduct();

     Product *productB = creator->factoryMethod();
  productB->somethingProduct();

   }
```



Creational / Abstract Factory

- https://godbolt.org/z/hh5se4o7j

```
int main(int argc, char* argv[]) {
  AbstractFactory* cf1 = new ConcreateFactory1();
  AbstractProductA* productA1 = cf1->createProductA();
  productA1->somethingProduct();
  AbstractProductB* productB1 = cf1->createProductB();
  productB1->somethingProduct();
  AbstractFactory* cf2 = new ConcreateFactory2();
  AbstractProductA* productA2 = cf2->createProductA();
  productA2->somethingProduct();
  AbstractProductB* productB2 = cf1->createProductB();
  productB2->somethingProduct();
   }
```

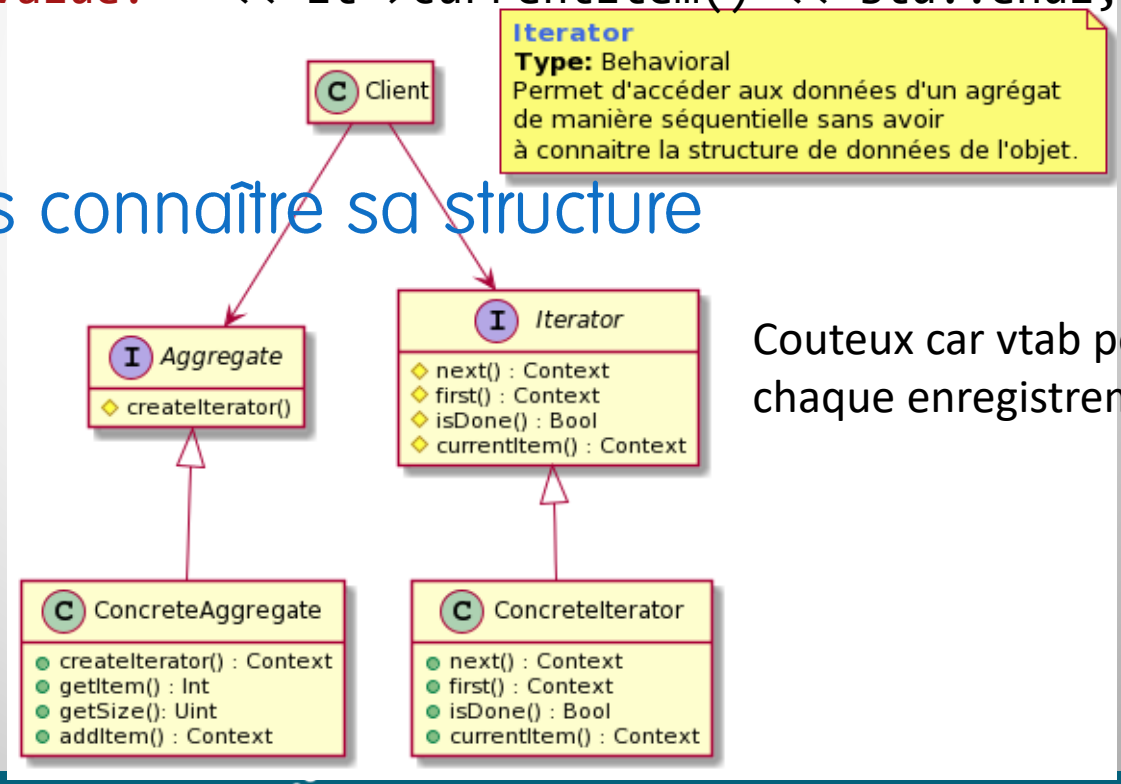


Behavioral / Iterator

- https://godbolt.org/z/bqxMrsr6K

```
int main() {
  ConcreteAggregate *concreteAggregate = new ConcreteAggregate(10);
  concreteAggregate->addItem(123);
     Iterator* it = concreteAggregate->createIterator();
  for ( ; !it->isDone(); it->next()) {
    std::cout << "Item value: " << it->currentItem() << std::endl;
  }
   }
```

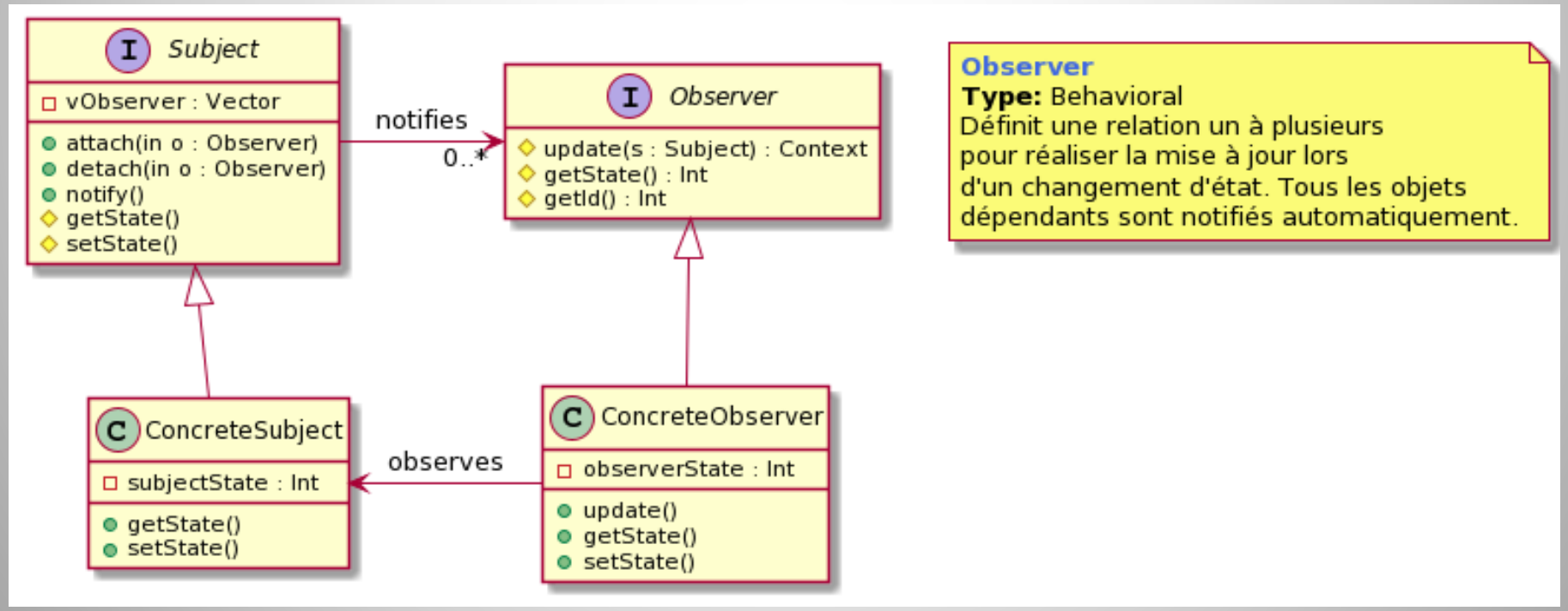
Visiter un objet sans connaître sa structure



Couteux car vtab pour chaque enregistrement

Behavioral / Observer

- https://godbolt.org/z/qMKrsoY7M
- Dépendance Publish-Subscribe



```
int main() {
    ConcreteObserver observer1(1000, 1);
    ConcreteObserver observer2(2000, 2);
    std::cout << "Observer1 state: " << observer1.getState() << std::endl;
    std::cout << "Observer2 state: " << observer2.getState() << std::endl;

    Subject* subject = new ConcreteSubject();
    subject->attach(&observer1);
    subject->attach(&observer2);

    subject->setState(10);
    subject->notify();

    std::cout << "Observer1 state: " << observer1.getState() << std::endl;
    std::cout << "Observer2 state: " << observer2.getState() << std::endl;

    subject->detach(1);
    subject->setState(100);
    subject->notify();

    std::cout << "Observer1 state: " << observer1.getState() << std::endl;
    std::cout << "Observer2 state: " << observer2.getState() << std::endl;
}
```

Retour sur Docker

Kernel Linux	Conteneur Docker	Fonction
init -S	tini	Devient le processus parent (PID=1), accepte les signaux et les dispatche aux descendants
init	supervisord	Idem à tini, écrit en Python, Permet d'avoir plusieurs process avec des uid différents

Une solution HA (cours #1)

- Le micro-service = panacée ?
  - Pour :
    - Simplicité de livraison
    - Facilité de montée en charge
    - Facilité de restauration
  - Contre :
    - Pas de hiérarchie fonctionnelle (priorité?)
    - Introduit le « nouveau » métier de Devops
    - Pousse à la consommation

Par définition, Docker pousse à n'avoir que 1 processus par conteneur

Lancement microservice multiprocessus

- CMD = la commande lancée dans le Dockerfile
- RUN = créé le conteneur à partir de l'image
- EXEC = s'attache à un conteneur existant

```
CMD cp -R /home/mpiuser/.ssh-source/* /home/mpiuser/.ssh \
   && chmod 700 /home/mpiuser/.ssh/authorized_keys \
   && chmod 700 /home/mpiuser/.ssh/id_rsa \
   && chown -R mpiuser:mpiuser /home/mpiuser/.ssh \
   && exec /usr/bin/supervisord -c /etc/supervisor/conf.d/supervisord.conf
```

- Copie les fichiers de clés RSA, applique les droits puis lance supervisord.

Paramétrage des clés SSH

Dans docker-compose.yml

`shm_size: '512m'` ➔ MPI utilise de la shared memory pour la communication (par défaut Docker alloue 64m)

```
secrets:
  - source: "id_rsa"
    target: "/home/mpiuser/.ssh-source/id_rsa"
  - source: "id_rsa_mpi_pub"
    target: "/home/mpiuser/.ssh-source/id_rsa.pub"
  - source: "authorized_keys"
    target: "/home/mpiuser/.ssh-source/authorized_keys"
```

Utilisation de la fonctionnalité « secrets », uniquement en swarm → permet un partage sécurisé

```
secrets:
  id_rsa_mpi_pub:
    file: ssh/id_rsa.mpi.pub
  id_rsa:
    file: ssh/id_rsa.mpi
  authorized_keys:
    file: ssh/id_rsa.mpi.pub
networks:
  mpinet:
    external: true
    name : yml_mpinet
    driver: overlay
volumes:
  usrlocalvarmpi-foo:
  usrlocalgrafana-foo:
  usrlocalinfluxdb-foo:
```

On décrit les secrets en fin de fichier docker-compose.yml

Les réseaux et les volumes partagés sont décrits en fin de fichier

Paramétrage de supervisord.conf

```
[supervisord]
nodaemon=true
user=root
pidfile = /tmp/supervisord.pid
logfile = /tmp/supervisord.log
logfile_maxbytes = 10MB
logfile_backups=10
loglevel = debug

[program:sshd]
user=mpiuser
umask=022
environment=HOME="/home/mpiuser",USER="mpiuser"
chown=mpiuser:mpiuser
stdout_logfile=/tmp/sshd.out
stderr_logfile=/tmp/sshd.err
stdout_logfile_maxbytes=10MB
autostart=true
autorestart=true
stdout_logfile_backups=10
command=/usr/sbin/sshd -D -f /home/mpiuser/ssh/sshd_config -E /tmp/sshd.log

[program:telegraf]
user=root
stdout_logfile_maxbytes=10MB
autostart=true
autorestart=false
stdout_logfile_backups=10
command=/usr/bin/telegraf -config /etc/telegraf/telegraf.conf -config-directory /etc/telegraf/telegraf.d
```

http://supervisord.org/configuration.html

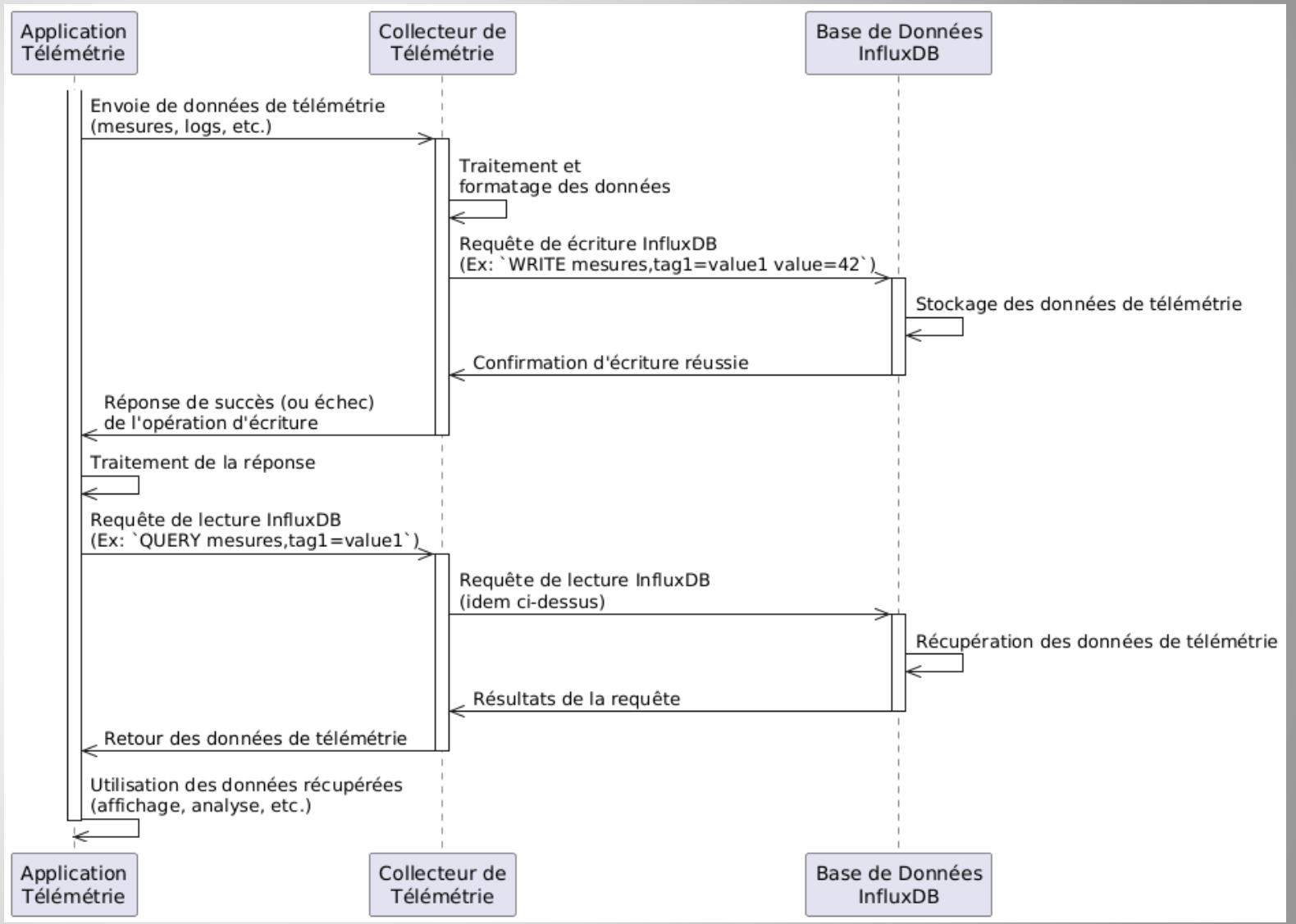
Supervision

- La fonction de supervision ➔ télémétrie

Supervision technique

Supervision applicative

- Permettre la détection des défauts du système (technique)
- Aider à diagnostiquer les problèmes du point de vue de l'utilisateur (applicative)
- Vérifie la bonne allocation des ressources (la saturation ou la sous utilisation) et lance des alertes (technique et applicative) ➔ l'application peut avoir un problème de conception (ie consommation disque).



Le concept de « Time Series »

- La métrologie a une dimension de projection : le temps.
- Il s'agit d'une Database spécialisée pour les données indexées sur la dimension temps.
- DB Spécialisée pour aggréger les données sur un intervalle de temps.

- Les 3 principaux (2024)

  InfluxDB (Go), Timescaledb (PostgreSQL- C), Prometheus (Go)

INTERACTIF

Quelles sont les principales DB Time Series et en quels langages sont elles écrites ? Peux-tu faire une réponse concise ?

TICK Stack

- Influxdb a lancé le concept de TICK Stack
- *T : Telegraf, *I : InfluxDB, *C : Chronograf, *K : Kapacitor
- Il s'agit d'un ensemble cohérent pour la métrologie, le stockage, la visualisation et l'alerte.

  Alternative Prometheus

TD Image Docker avec Telegraf

- Objectifs
  - Déployer une métrologie sur 4 nœuds
  - Pouvoir faire de la télémétrie applicative ➔ une télémétrie « banalisée », ie l'utilisateur n'a pas à spécifier le collecteur de la télémétrie ni à s'identifier.

Création des nœuds, connexion

INTERACTIF

- D'abord créer un swarm

```
docker swarm init
```

- Puis y attacher un réseau

```
docker network create --driver=overlay --attachable yml_mpinet
```

- Dans le répertoire du cours CM3

```
docker-compose up --scale mpihead=1 --scale mpinode=3---scale
   grafana=1 -d
```

→ utilise le fichier docker-compose.yml

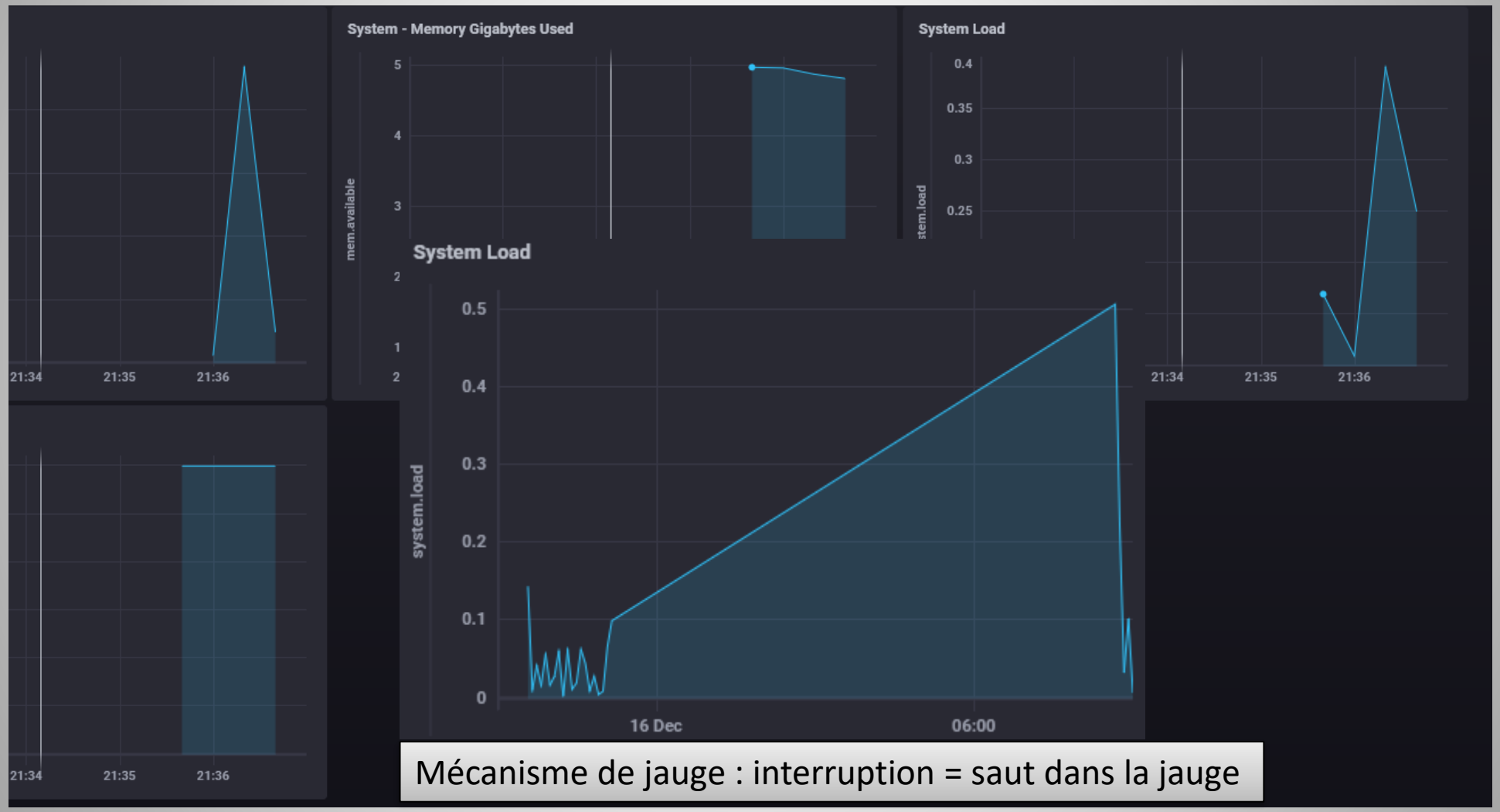
→ L'image docker n'est pas la même (jmbatto/**m2chps-mpi41-xmp**)

Se connecter à un shell docker (utiliser le bash) : (root car supervisor), ssh localhost:2022 → mpiuser, utiliser la clé ppk

Vision de la télémétrie technique

INTERACTIF

Chaque nœud a son service telegraf – qui peut suivre des applications et des services



Mécanisme de jauge : interruption = saut dans la jauge

Paramétrage telegraf

INTERACTIF

- ➔fichier telegraf.conf
- Le paramétrage est « anonyme » - on ne précise pas le nom du nœud
- Le service telegraf peut encrypter la communication (confidentialité)

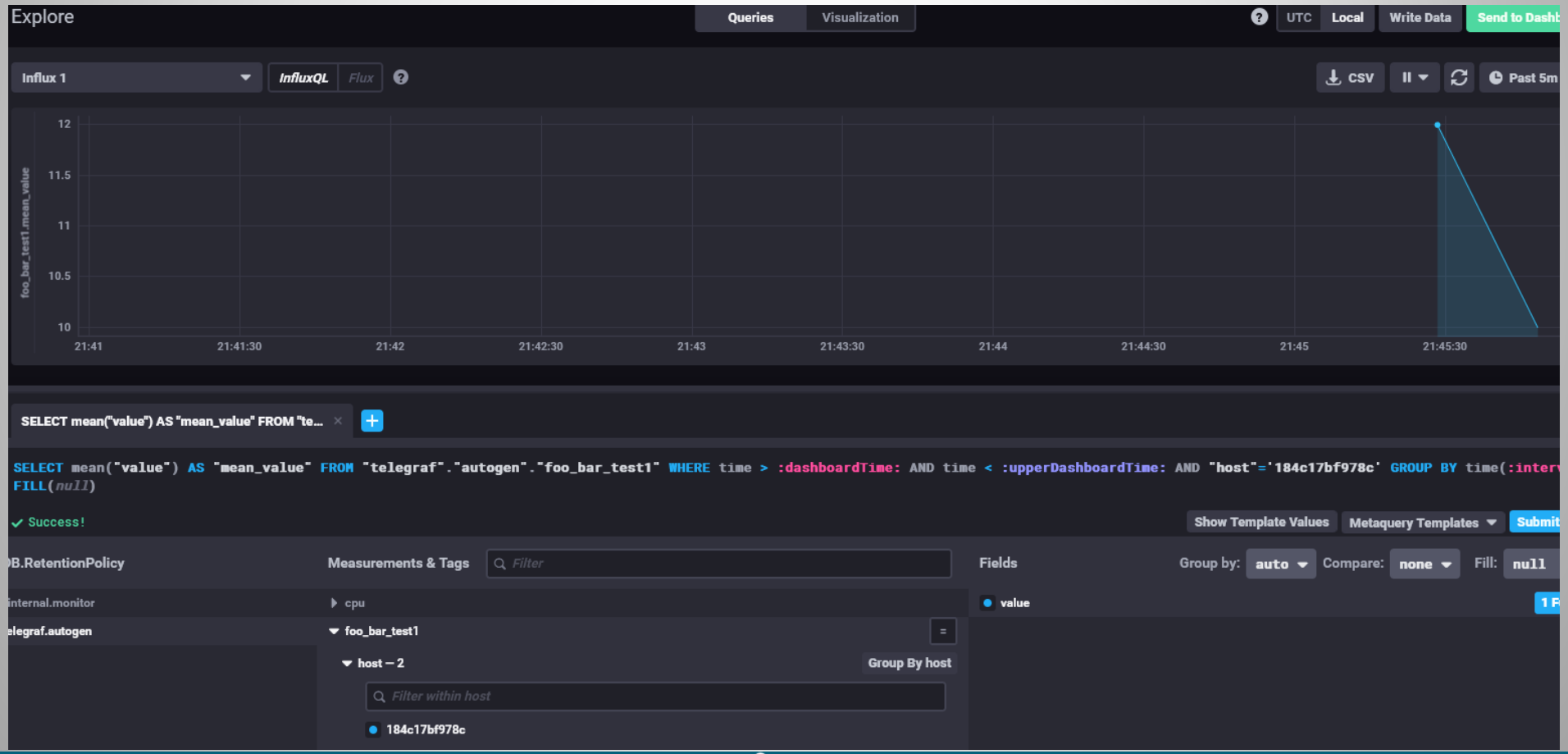
Télémétrie applicative

INTERACTIF

- statsd (2011) est un service d'audit répandu qui écoute sur un port (socket)
- Écriture avec netcat sur le port 8125 → redirection vers influxdb

```
echo "foo.bar.test1:+1|g" | nc -u localhost 8125
```

Telegraf ajoute une encapsulation https et redirige localhost vers un collecteur



Question : comment faire un netcat applicatif?

INTERACTIF

- 2 pistes (git clone) :
- https://github.com/guzlewski/netcat.git

- https://github.com/romanbsd/statsd-c-client.git
- (attention à LD_LIBRARY_PATH)

- Vous pouvez écrire dans un socket (et demander de l'aide à une IA)

TD2 individuel pour le 15 dec 2024

INTERACTIF

- EB : faire la télémétrie applicative de 2 nœuds MPI (osu_bibw.c)
- SFD : voici un exemple de fonction en langage Go, réalisé une fonction en C et utilisez là pour mesurer la durée de chaque benchmark.

```
//transaction : le nom de la portion de code instrumentée
//since une durée
func Chrono(since time.Duration, transaction string) {
        var client *statsd.Client
        client = statsd.NewClient("localhost:8125", statsd.MaxPacketSize(1400),
   statsd.MetricPrefix(ChronoGeneralTag+"."))← le point est supprimé
        client.PrecisionTiming("requestTime",
                since,
                statsd.StringTag("transaction", transaction))
                        client.Close()
   }
}
```

TD individuel pour le 15 dec 2024 minuit

- A me rendre **pour le 15 décembre 2024 minuit** :
- Mettre dans la Forge Gogs un rapport du TD2 (**1 page** PDF – présente un benchmark dans un tableau avec une introduction qui donne le contexte, avec le nom de l'étudiant et la date)
- Avec le diagramme de séquence PlantUML qui décrit votre test et le **prompt utilisé** (+ref IA).
- Avec la/les captures « chronograf » commentées
- +code source commenté (dedans : date, nom étudiant)

Exemple de prompt :

Peux-tu m'écrire le code pour un diagramme de séquence dans PlantUML qui présente une télémétrie applicative avec l'envoie d'une requête vers une base de télémétrie InfluxDB ?