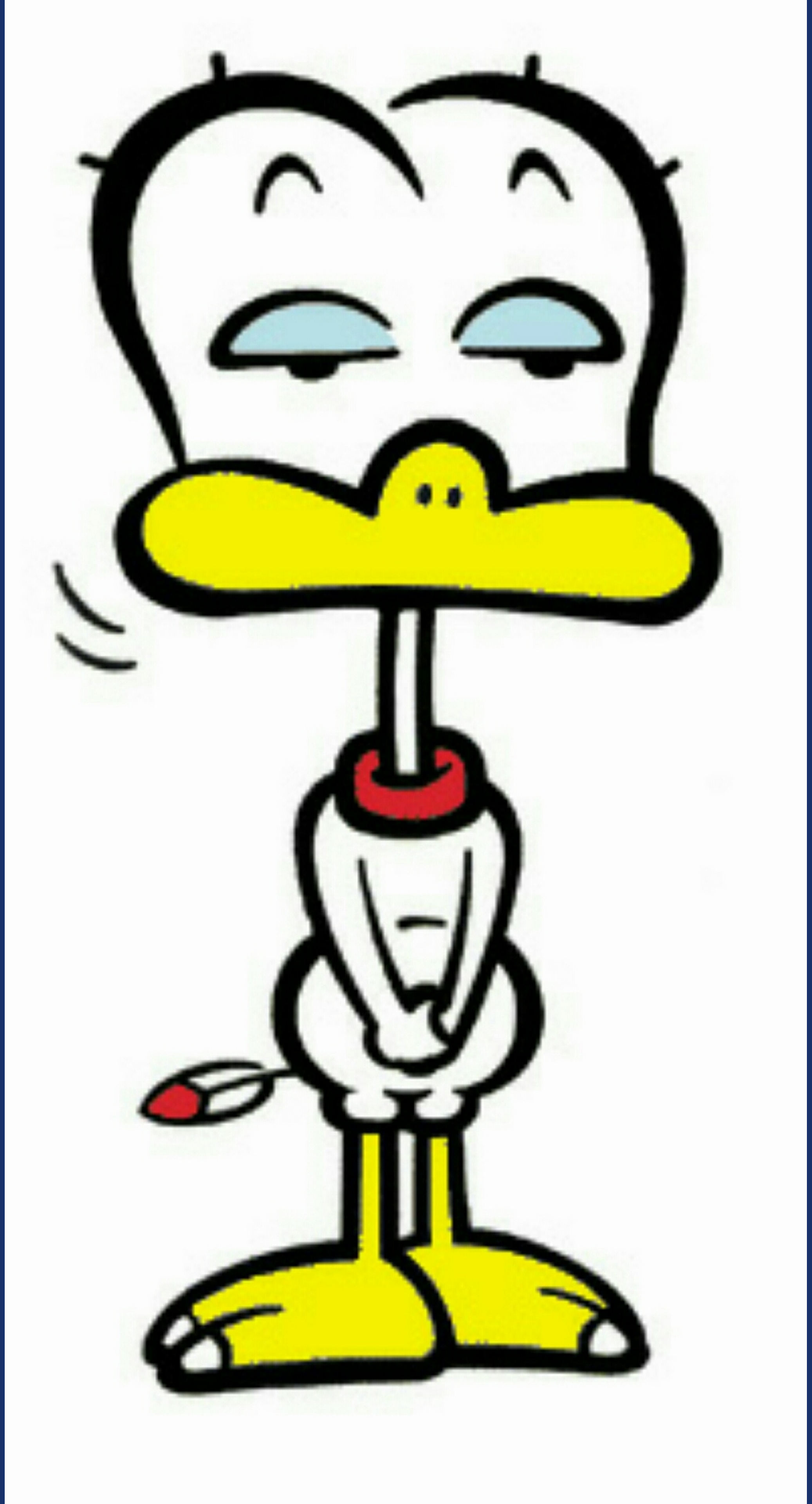
BEAUTIFUL DOMAIN LOGIC

SCALA MATSURI - MARCH 2022

美しいドメインロジック



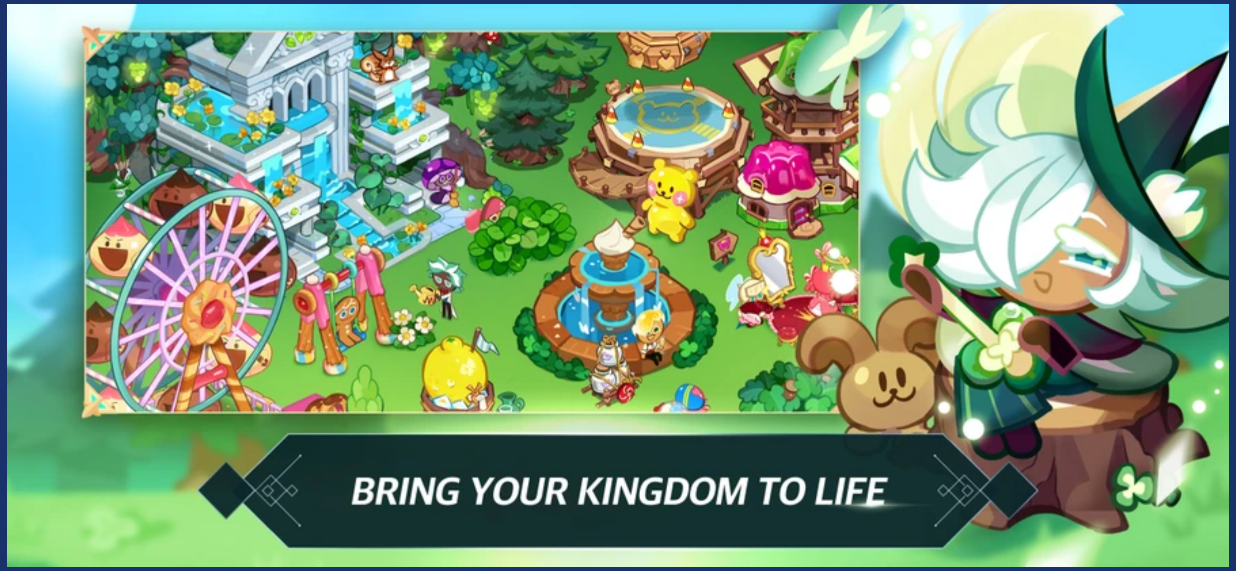
WHO AM I?

- PIERRE RICADAT AKA @GHOSTDOGPR
- 🇫🇷 EXILED TO 🇰🇷
- DEVELOPER AT DEVSISTERS
- CONTRIBUTOR TO ZIO
- CREATOR OF CALIBAN

🇰🇷に長く住んでいます
DEVSISTERS 勤務、CALIBAN 作者

DEVSISTERS

- KOREAN GAME COMPANY FOUNDED IN 2007
- LAUNCHED SOCIAL RPG COOKIERUN: KINGDOM IN JANUARY 2021



2007年設立の韓国のゲーム会社

昨年 COOKIE RUN: KINGDOM をローンチ

COOKIERUN: KINGDOM

- OVER 40 MILLIONS DOWNLOADS SINCE LAST YEAR
- 350,000+ CONCURRENT PLAYERS
- 50,000+ REQUESTS/SEC

- SERVER CODE ENTIRELY WRITTEN IN SCALA 🤩

同時プレーヤー35万人以上

サーバーのコードは全てSCALA

PLAN

1. OVERALL ARCHITECTURE
2. DOMAIN LOGIC CHARACTERISTICS
3. IMPLEMENTATION CHOICES
4. PRACTICAL EXAMPLES

全体のアーキテクチャ
ドメイン·ロジックの特徴などを紹介

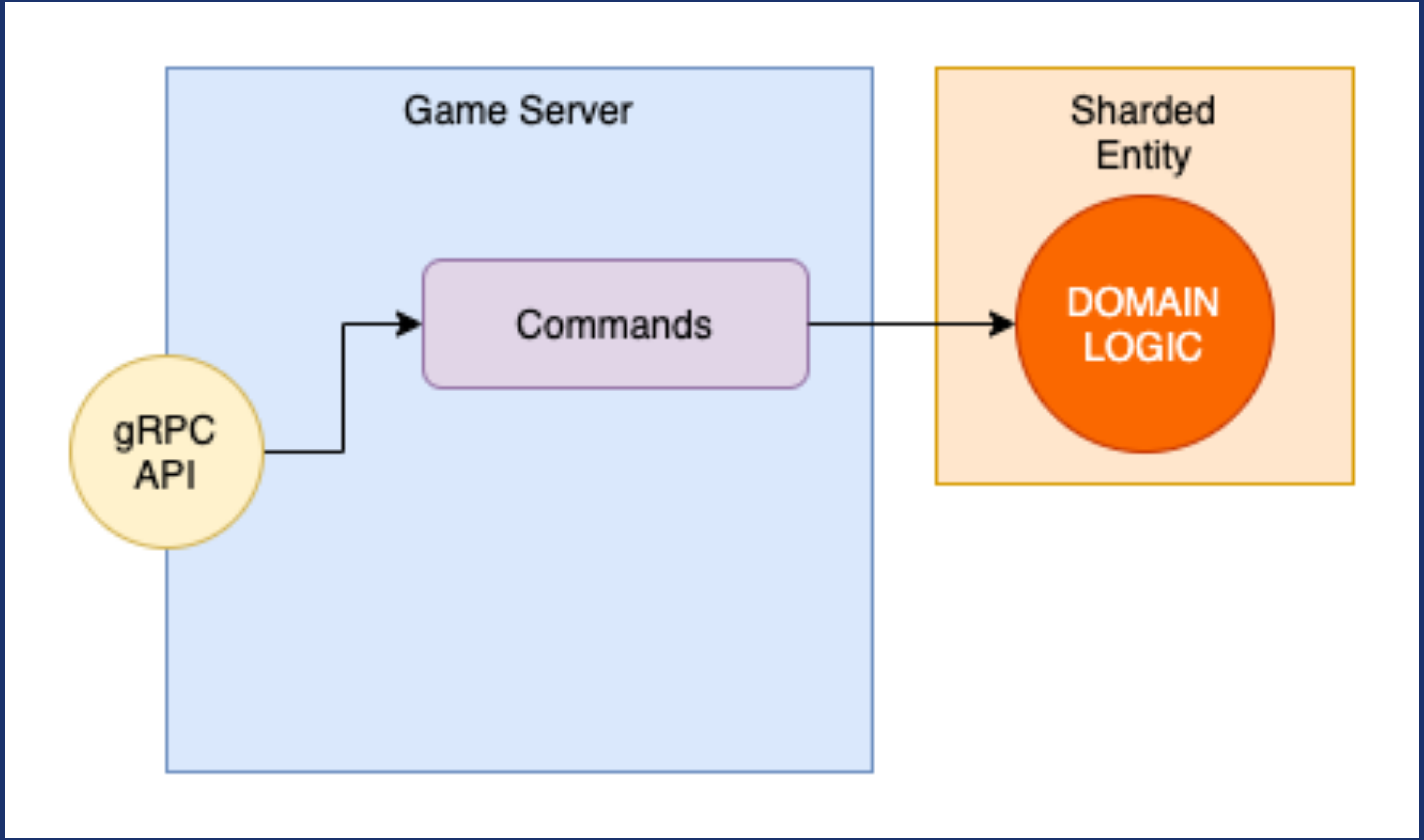
CQRS

- COMMAND QUERY RESPONSIBILITY SEGREGATION
  - COMMANDS MODIFY THE STATE
  - QUERIES ARE READ-ONLY

- SINGLE WRITER PRINCIPLE

コマンドは状態の変更
クエリはリード·オンリー

CQRS: COMMANDS



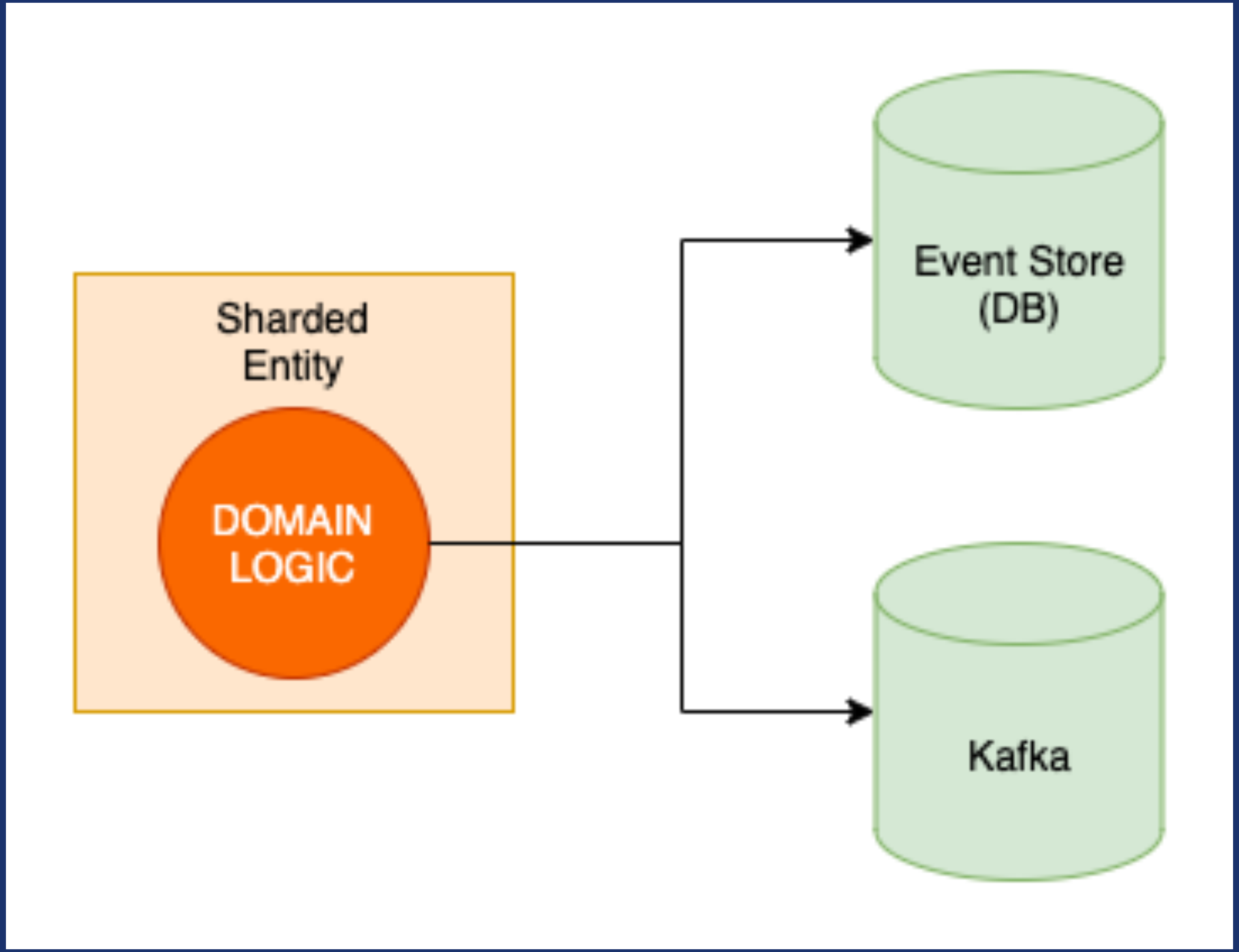
EVENT SOURCING

- CRUD
  - ADD/MODIFY/DELETE DATA DIRECTLY
- EVENT SOURCING
  - ONLY SAVE EVENTS
  - BUILD STATE BY REPLAYING EVENTS

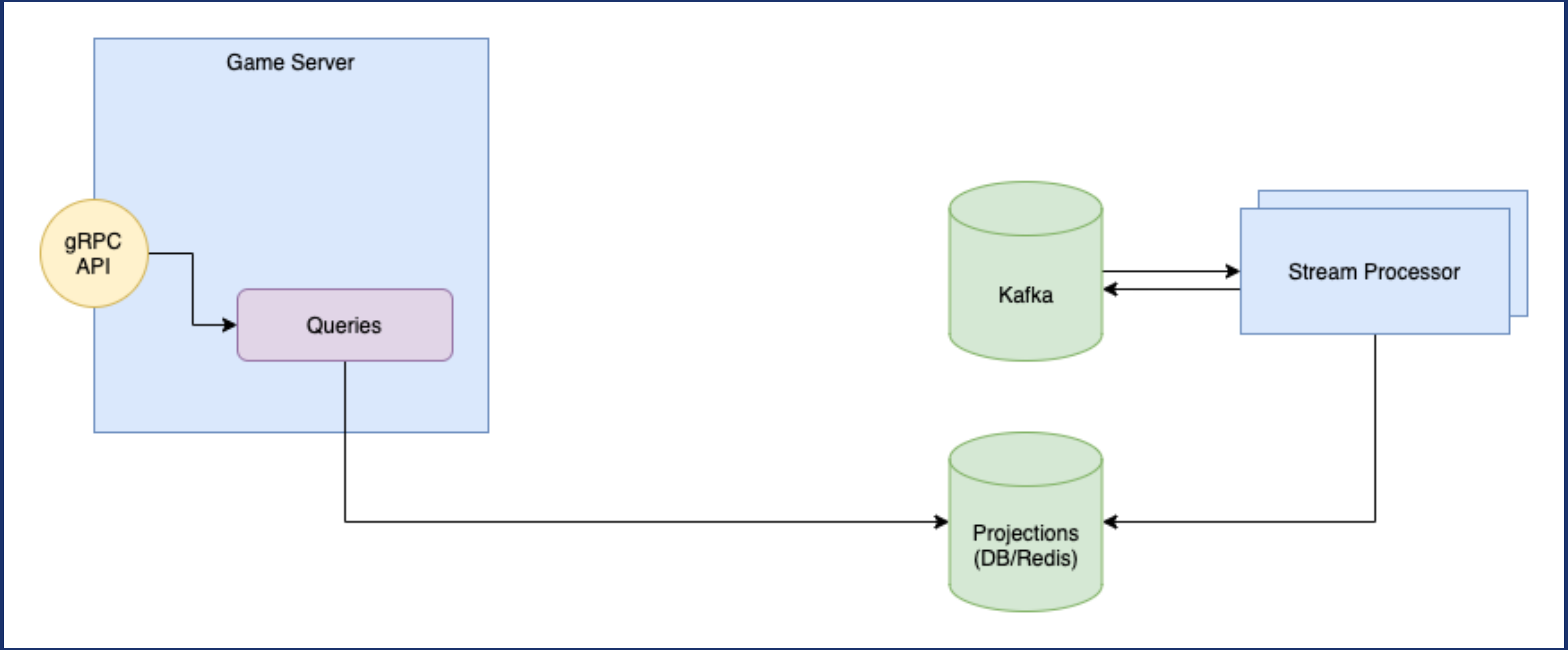
イベント·ソーシングはイベントのみ保存

イベントをリプレイすることで状態を構築

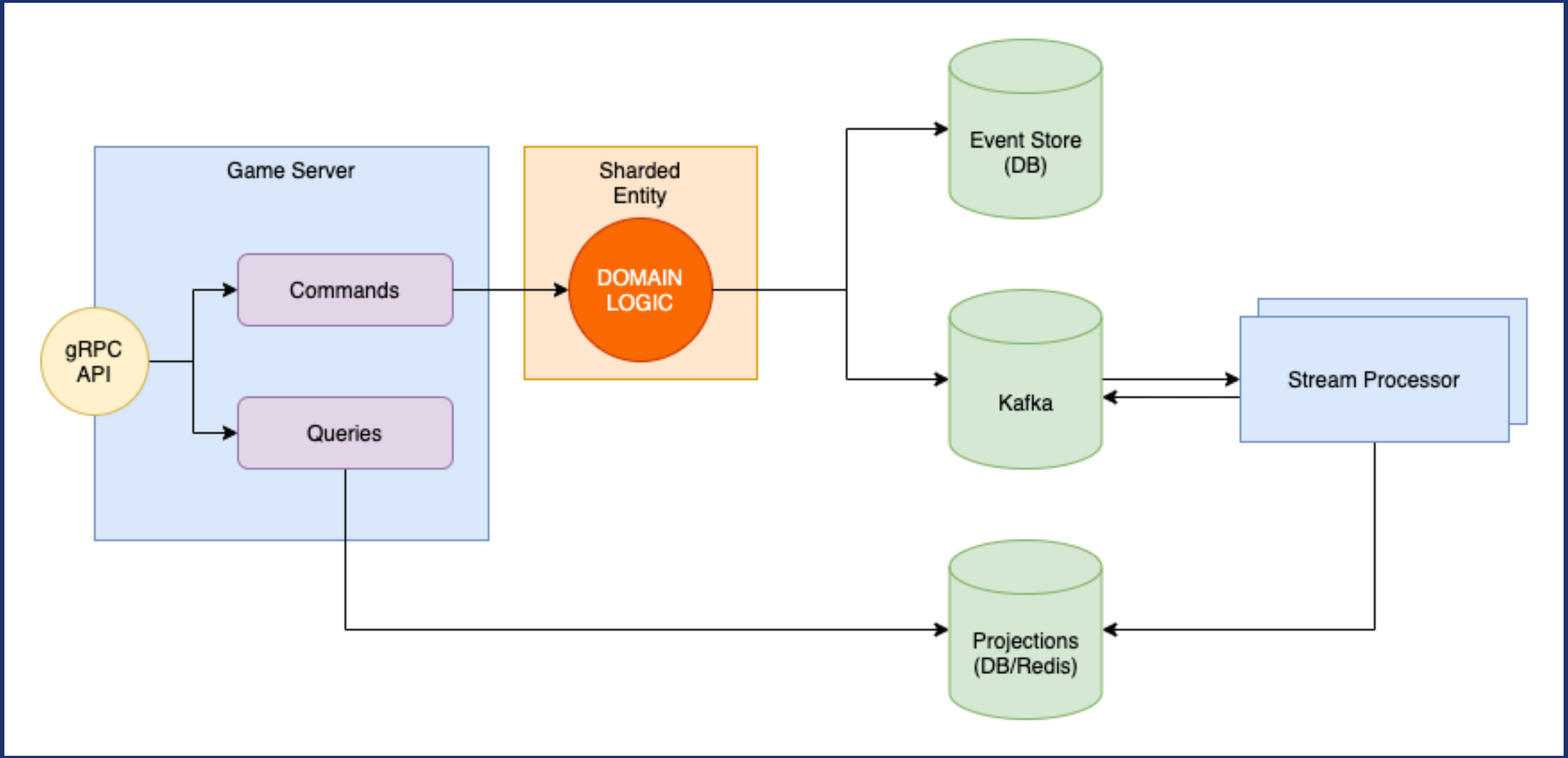
EVENT SOURCING



CQRS: QUERIES



OVERALL ARCHITECTURE



DOMAIN LOGIC

- **TRANSITION**: STATE + EVENT => STATE
- **PROGRAM**: STATE + COMMAND => STATE + EVENTS (+ RESULT)

状態遷移: 状態 + イベント => 状態
プログラムはコマンドを処理してイベントや結果を返す

CONSTRAINTS

> MAY RETURN SOME RESULT
> MAY FAIL WITH A DOMAIN ERROR
> MAY REQUIRE SOME CONFIGURATION (AKA ENVIRONMENT)

制約: 結果を返すかも
失敗してドメイン·エラーを返すかも

MORE CONSTRAINTS

- NO SIDE EFFECTS!
  - REPLAYABLE
  - ERRORS
  - TRANSACTIONS ACROSS ENTITIES

副作用は禁止
リプレイできるように

MORE CONSTRAINTS

- FAST!
  - IT IS OUR GAME SERVER'S MAIN TASK (>50% CPU)
  - DOMAIN LOGIC CAN GET PRETTY COMPLEX
    - E.G. VALIDATING HUNDREDS OF QUEST REQUIREMENTS

高速でなくてはいけない
ゲームサーバの主要なタスクであるため

DOMAIN LOGIC IN SCALA

- TRANSITION:

```
(Event, State) => Either[Error, State]
```

- PROGRAM:

```
(Env, State) => Either[Error, (State, Chain[Event], Result)]
```

ドメイン・ロジックをSCALAで書いてみる

HOW TO IMPLEMENT PROGRAMS?

- FOR COMPREHENSION WITH THE FOLLOWING OPERATIONS:
  - SUCCEED WITH A RESULT
  - FAIL WITH AN ERROR
  - GET STATE
  - GET ENVIRONMENT
  - LIFT AN EVENT

実装はFOR 内包表記を使う

PLAIN FUNCTIONS

```scala
type Program[A] =
  (Env, State, Chain[Event]) => Either[Error, (State, Chain[Event], A)]

implicit val programMonad: Monad[Program] = ???
```

- USE FLATMAP FROM CATS/SCALAZ/ZIO-PRELUDE

素の関数を使う場合

CATS や ZIO-PRELUDE の FLATMAP を使う

PLAIN FUNCTIONS 😈

- NOT CONVENIENT
- FLATMAP NOT STACK SAFE
- NEED TO UNLIFT/LIFT FUNCTION, EITHER AND TUPLE AT EACH OPERATION

いちいち関数を持ち上げるのが面倒

FLATMAP はスタックセーフじゃない

CATS IRWST

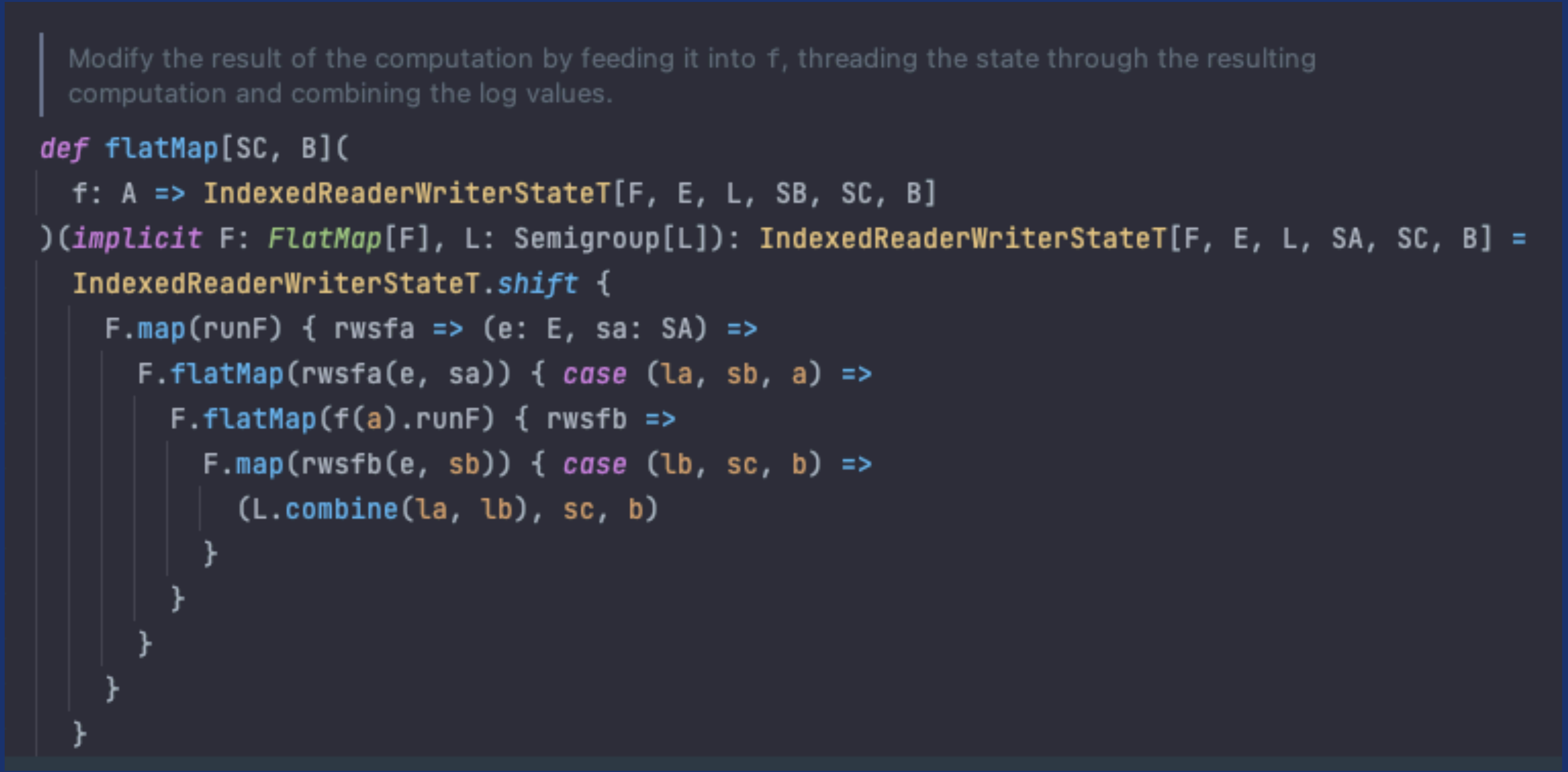
```
type Program[A] =
  IndexedReaderWriterStateT[F, Env, Chain[Event], State, State, A]

class IndexedReaderWriterStateT[F[_], E, L, SA, SB, A](
  val runF: F[(E, SA) => F[(L, SB, A)]]
)
```

- F?
  - CAN'T USE EVAL
  - COULD BE EITHER OR IO

ここでF に何を置くか? EITHER か IO の 2択

CATS IRWST 😈

> SO MANY LAYERS TO UNLIFT/LIFT



Modify the result of the computation by feeding it into f, threading the state through the resulting computation and combining the log values.

```scala
def flatMap[SC, B](
  f: A => IndexedReaderWriterStateT[F, E, L, SB, SC, B]
)(implicit F: FlatMap[F], L: Semigroup[L]): IndexedReaderWriterStateT[F, E, L, SA, SC, B] =
  IndexedReaderWriterStateT.shift {
    F.map(runF) { rwsfa => (e: E, sa: SA) =>
      F.flatMap(rwsfa(e, sa)) { case (la, sb, a) =>
        F.flatMap(f(a).runF) { rwsfb =>
          F.map(rwsfb(e, sb)) { case (lb, sc, b) =>
            (L.combine(la, lb), sc, b)
          }
        }
      }
    }
  }
```

何層にも渡った持ち上げ

ZIO

```
type Program[A] = ZIO[Has[Env]          with
                      Has[Ref[State]] with
                      Has[Ref[Chain[Event]]], Error, A]
```

- Single data type holding all information
- No unnecessary unlift/lift

1つのデータ型で全ての情報を持つ

ZIO 😈

- ALLOW YOU RUNNING ANY EFFECT
- DISCIPLINE
  - NOT RUNNING ANY EFFECT OUTSIDE OF ZIO ✅
  - NOT RUNNING ANY EFFECT WHEN THE TYPE IS ZIO? 😬
- THE TEMPTATION IS TOO HIGH

ZIO のエフェクトなら何でも実行できてしまう

CATS MTL

```
trait Program[F[_]] extends
  MonadError[F, Error] with // raise error
  Ask[F, Env]          with // read environment
  Stateful[F, State]   with // read/write state
  Tell[F, Chain[Event]]     // write events
```

> CAN ONLY USE FUNCTIONS FROM THESE INTERFACES

> CAN USE IRWST OR ZIO AS F

これらのインターフェイスからの関数のみ使える

F は IRWST か ZIO か選べる

CATS MTL 😈

- TYPE INFERENCE
- EVERYTHING RELIES ON IMPLICITS (BOILERPLATE, LESS DISCOVERABLE)
- CAN'T ELIMINATE ERRORS

型推論が効かない

全てが IMPLICITS 頼り

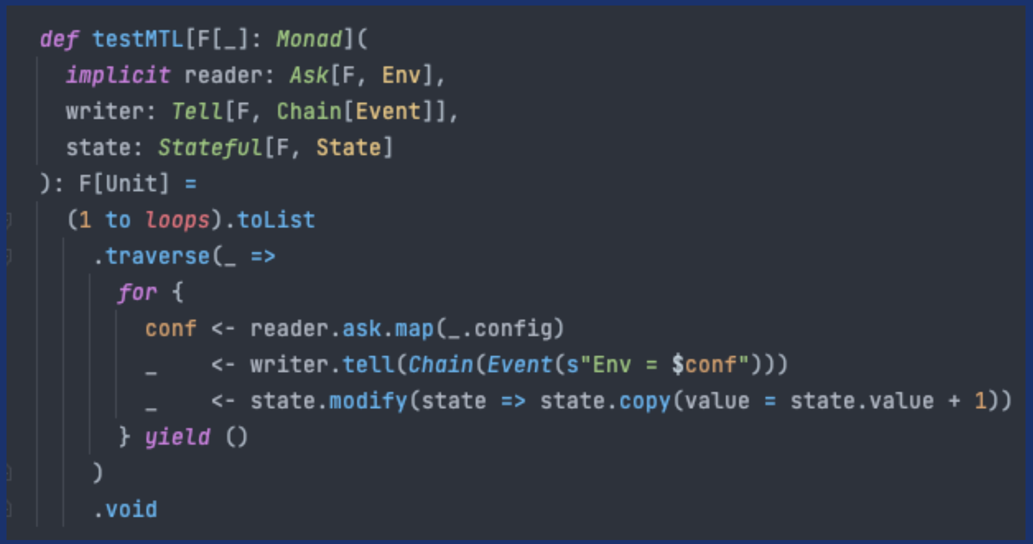
ZPURE (ZIO-PRELUDE)

```
type Program[A] =
  ZPure[Event, State, State, Env, Error, A]
```

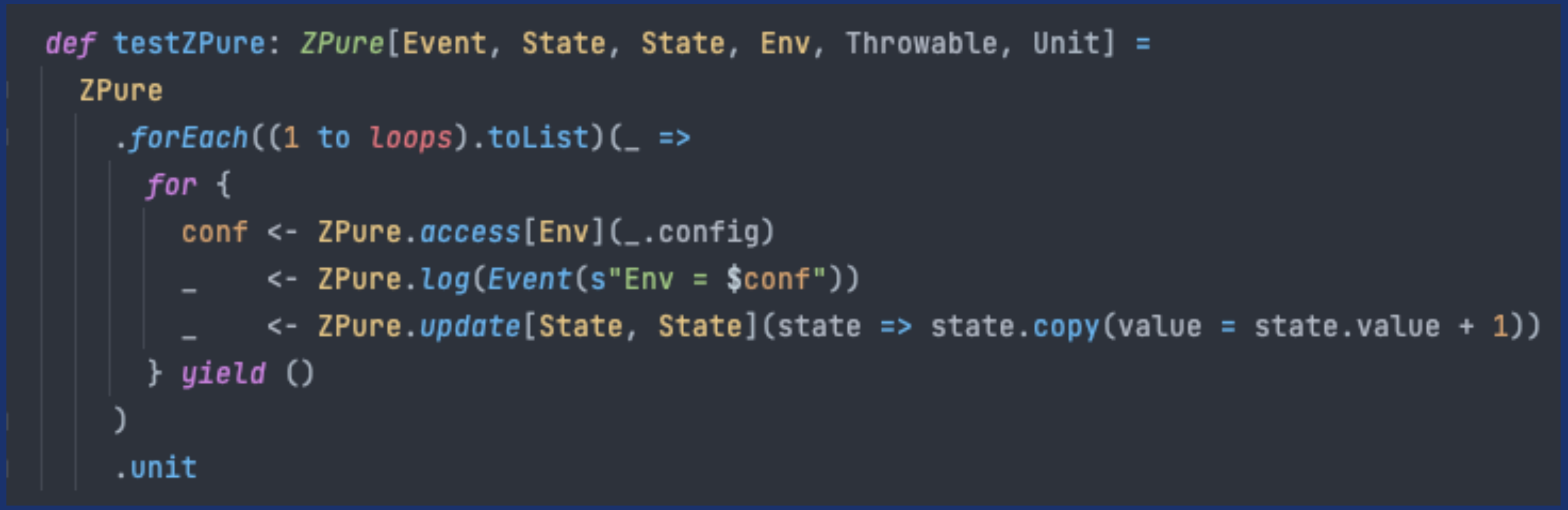
- SINGLE DATA TYPE HOLDING ALL INFORMATION
- NO UNNECESSARY UNLIFT/LIFT
- NO EFFECTS

1つのデータ型で全ての情報を持つ

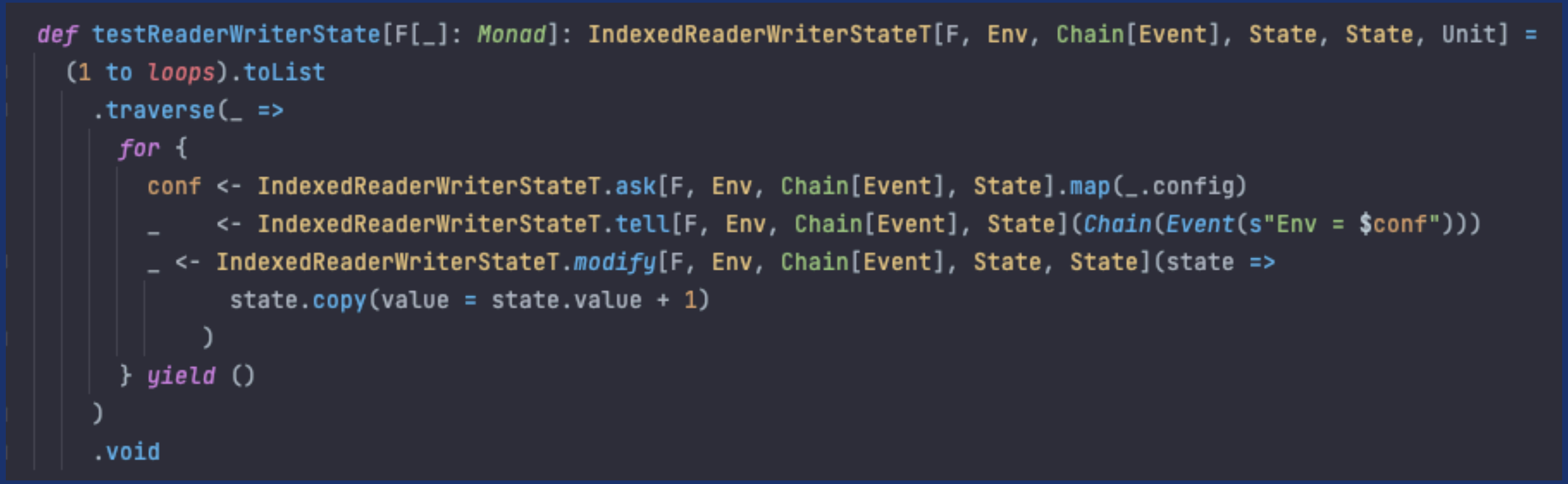
PERFORMANCE



```scala
def testMTL[F[_]: Monad](
  implicit reader: Ask[F, Env],
  writer: Tell[F, Chain[Event]],
  state: Stateful[F, State]
): F[Unit] =
  (1 to loops).toList
    .traverse(_ =>
      for {
        conf <- reader.ask.map(_.config)
        _    <- writer.tell(Chain(Event(s"Env = $conf")))
        _    <- state.modify(state => state.copy(value = state.value + 1))
      } yield ()
    )
    .void
```



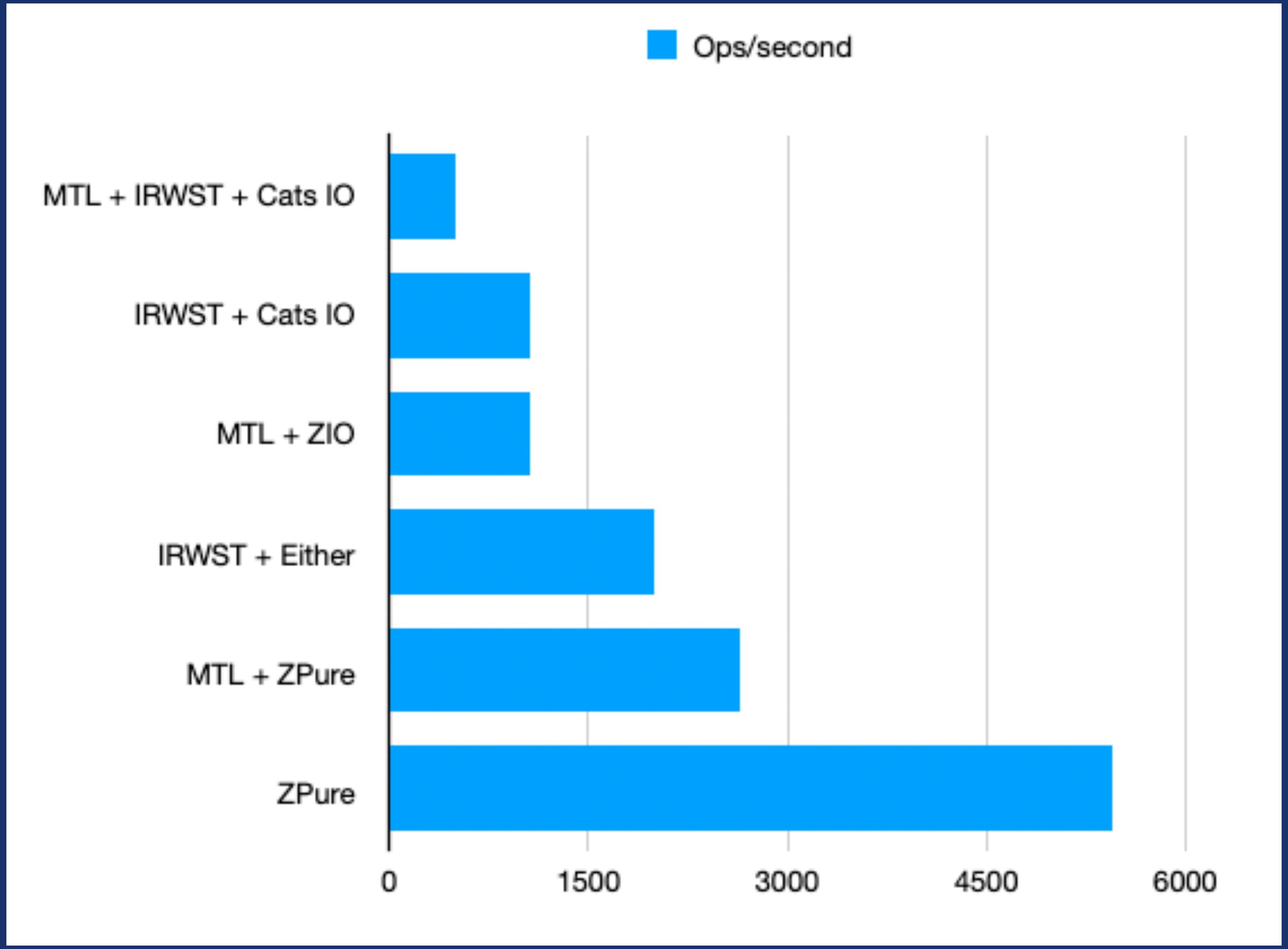
```scala
def testZPure: ZPure[Event, State, State, Env, Throwable, Unit] =
  ZPure
    .forEach((1 to loops).toList)(_ =>
      for {
        conf <- ZPure.access[Env](_.config)
        _    <- ZPure.log(Event(s"Env = $conf"))
        _    <- ZPure.update[State, State](state => state.copy(value = state.value + 1))
      } yield ()
    )
    .unit
```



```scala
def testReaderWriterState[F[_]: Monad]: IndexedReaderWriterStateT[F, Env, Chain[Event], State, State, Unit] =
  (1 to loops).toList
    .traverse(_ =>
      for {
        conf <- IndexedReaderWriterStateT.ask[F, Env, Chain[Event], State].map(_.config)
        _    <- IndexedReaderWriterStateT.tell[F, Env, Chain[Event], State](Chain(Event(s"Env = $conf")))
        _ <- IndexedReaderWriterStateT.modify[F, Env, Chain[Event], State, State](state =>
               state.copy(value = state.value + 1)
             )
      } yield ()
    )
    .void
```

性能比較

PERFORMANCE



HTTPS://GITHUB.COM/GHOSTDOGPR/MTL-BENCHMARKS

IN PRACTICE

- CUSTOM DSL FOR BASIC OPERATIONS
- BUILDING BLOCKS ON TOP OF IT
- ASSEMBLING DOMAIN LOGIC LIKE LEGOS

実際の所は、カスタム DSL で基礎を作り
その上に再利用できるブロックを作っていく

CUSTOM DSL

```scala
def pure[A](a: A): Program[A]

def raiseError[A](t: => E): Program[Nothing]

def assertThat(cond: => Boolean, e: => E): Program[Unit]

def extractOption[A](a: Option[A], t: => E): Program[A] =
  a match {
    case Some(v) => pure(v)
    case None    => raiseError(t)
  }
```

CUSTOM DSL

```
// State
def get: Program[S]
def inspect[A](f: S => A): Program[A]

// Environment
def inquire[A](f: Env => A): Program[A]

// Events
def liftEvent(e: Evt): Program[Unit]
```

MORE BUILDING BLOCKS

```scala
type GuildProgram[+A] =
  ZPure[GuildEvent, GuildState, GuildState, GuildEnv, ValidationError, A]

lazy val inquireGuildMetadata: GuildProgram[GuildMetadata] =
  inquire(_.guildMetadata)

lazy val inquireRequesterId: GuildProgram[UserId] =
  inquire(_.requesterId).flatMap(
    extractOption(_, InvalidInput("There is no requester."))
  )
```

BEAUTIFUL DOMAIN LOGIC

```
lazy val joinGuild: GuildProgram[Unit] =
  for {
    metadata    <- inquireGuildMetadata
    requesterId <- inquireRequesterId
    memberCount <- getGuild.map(_.members.size)
    _           <- assertThat(
                     memberCount < metadata.maxMemberCount,
                     ValidationError.GuildFull()
                   )
    _           <- liftEvent(GuildMemberJoined(requesterId))
  } yield ()
```

美しいドメイン·ロジック

WHY BEAUTIFUL?

- ZERO NOISE, FOCUS 100% ON DOMAIN
- EASY TO READ, MAINTAIN AND ONBOARD NEW DEVELOPERS
- INDEPENDENT FROM PROGRAM IMPLEMENTATION
- FAST!

ノイズ無し、読みやすい

PROGRAM実装からの独立、高速!

THANKS!

- TWITTER: @GHOSTDOGPR
- TRY COOKIERUN: KINGDOM AT COOKIERUN-KINGDOM.COM

ご清聴ありがとうございます

COOKIE RUN: KINGDOM も試してね

QUESTIONS?