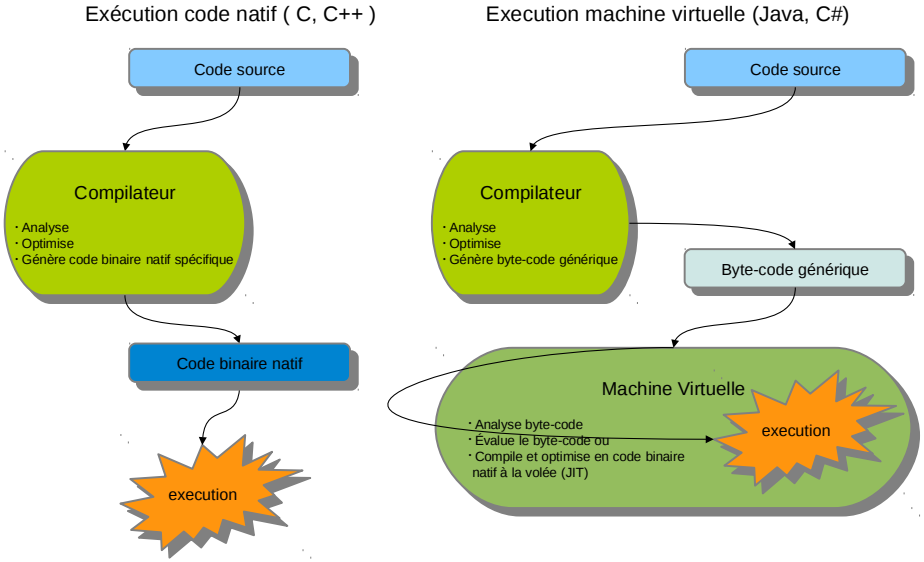
Introduction aux Machines Virtuelles avec VMKit

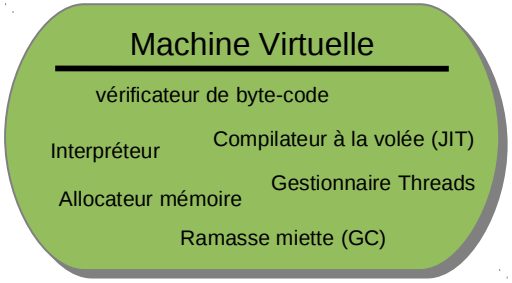
Ingénieur : Harris Bakiras,
Responsables : Gaël Thomas, Gilles Müller
EQUIPE REGAL LIP6 – INRIA (Paris/France)

VMKit
un substrat de machine virtuelle

Les environnements d'exécution



Les machines virtuelles



Problèmes

Développement extrêmement long à réaliser !

Comment tester une idée avec différents langages ?

Comment implémenter une nouvelle VM efficace pour de nouveaux langages ?

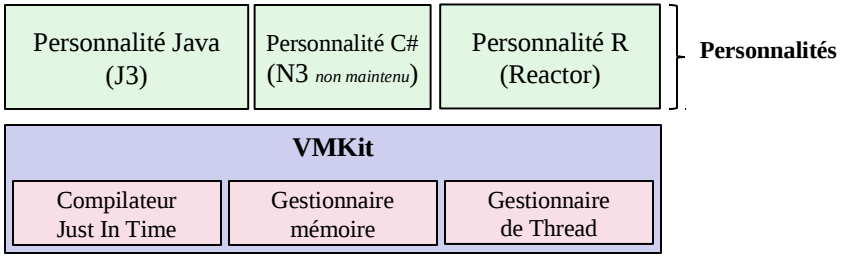
Comment étendre rapidement des langages existant ?

VMKit : un substrat de machine virtuelle

But de VMKit : aider à expérimenter dans les VM

Objectif : factoriser les composants communs des VM

- Compilateur Just In Time : génération de code natif à la volée
- Gestionnaire mémoire : alloue et collecte automatiquement la mémoire libre
- Gestionnaire de Thread : créé et synchronise les threads

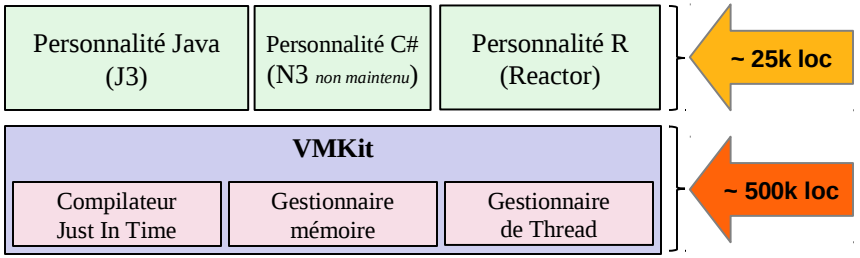


VMKit : un substrat de machine virtuelle

But de VMKit : aider à expérimenter dans les VM

Objectif : factoriser les composants communs des VM

- Compilateur Just In Time : génération de code natif à la volée
- Gestionnaire mémoire : alloue et collecte automatiquement la mémoire libre
- Gestionnaire de Thread : créé et synchronise les threads

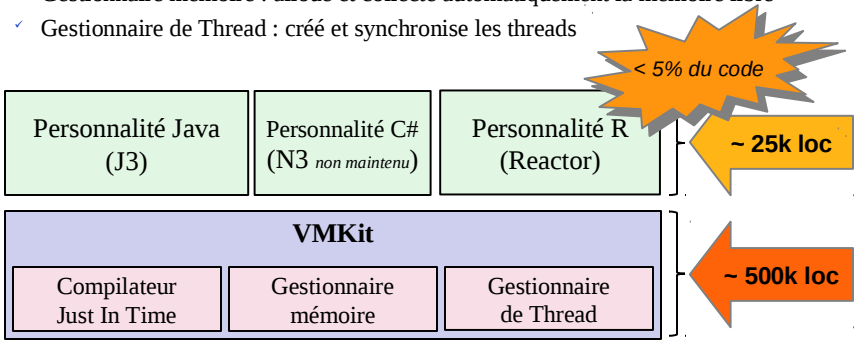


VMKit : un substrat de machine virtuelle

But de VMKit : aider à expérimenter dans les VM

Objectif : factoriser les composants communs des VM

- Compilateur Just In Time : génération de code natif à la volée
- Gestionnaire mémoire : alloue et collecte automatiquement la mémoire libre
- Gestionnaire de Thread : créé et synchronise les threads



VMKit
d'un point de vue technique

Implementation de VMKit

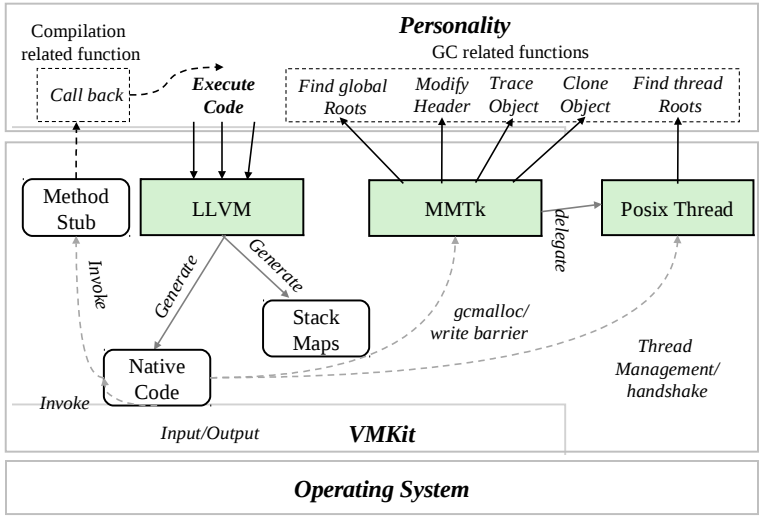
Choix d'implémentation: **repose sur des composants externes**

- ✓ Compilateur à la volée (JIT) : LLVM [Lattner & Adve – CGO'04]
- ✓ Gestionnaire mémoire : MMTk [Blackburn et Al. – ICSE'04]
- ✓ Gestionnaire de Thread : Posix

VMKit = glue entre les différents composants

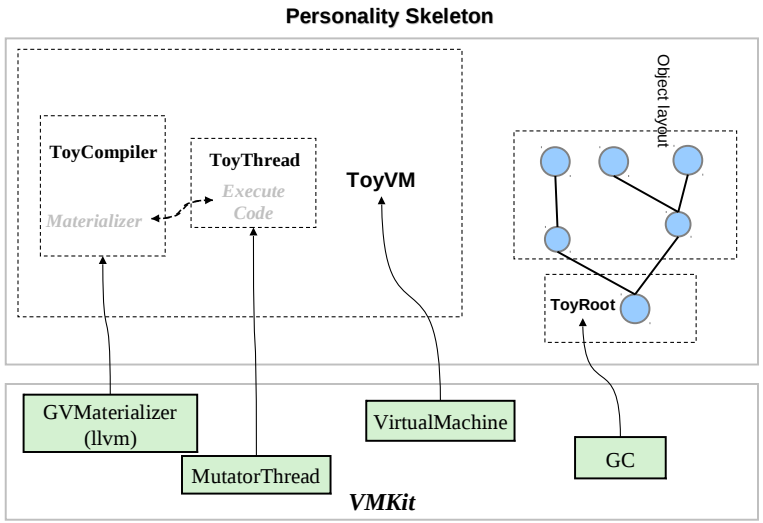
- Entre JIT-C et gestionnaire mémoire = GC précis
- Entre le gestionnaire de Thread et le gestionnaire mémoire = GC multi-threadé

Architecture détaillée

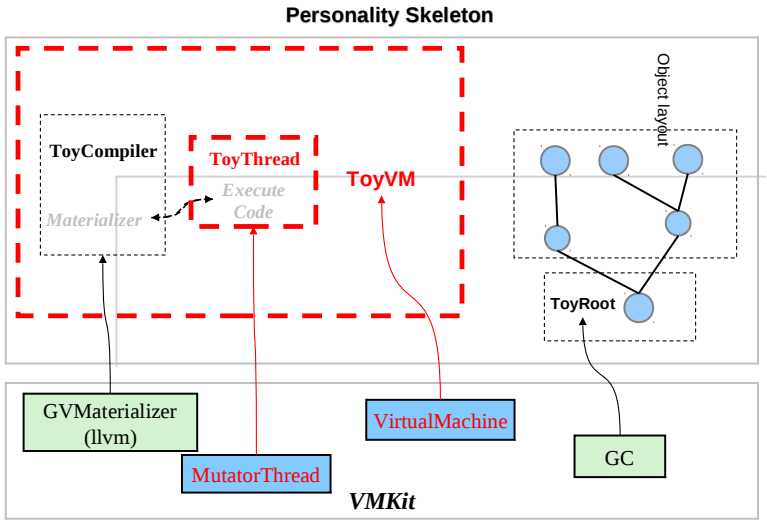


VM Minimal (ToyVM) pour le tutoriel

Architecture de la ToyVM



Architecture de la ToyVM



ToyVM et ToyThread

ToyVM ← VirtualMachine

- *Gestion des Threads*
- *Point d'entrée du Garbage collector*
- *Backtrace (parcours de la pile d'exécution)*

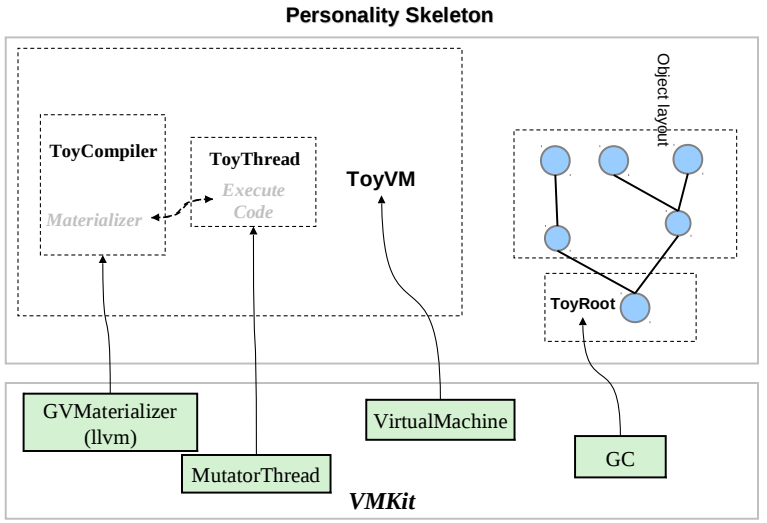
- Traçage des variables globales
- Gestion des exceptions

ToyThread ← MutatorThread ← Thread

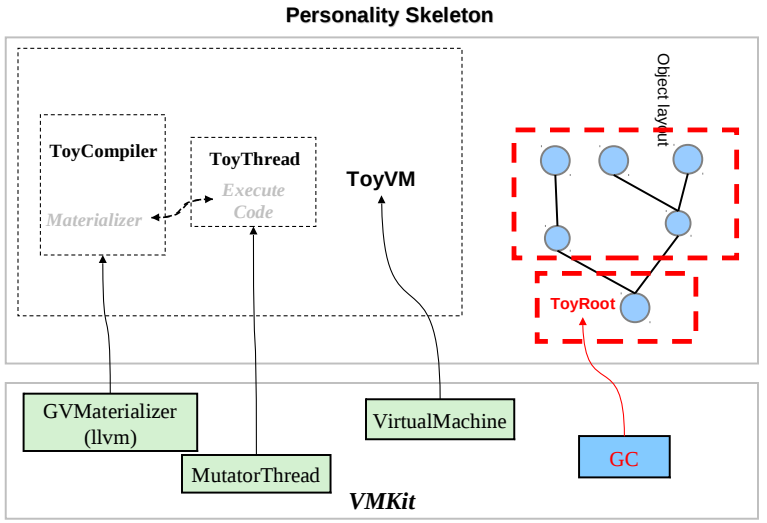
- *Rendez-vous du garbage collector*
- *Scan de la pile d'exécution lors du GC*

- Traçage des variables locales
- Méthode principale d'exécution

Architecture de la ToyVM



Architecture de la ToyVM



ToyRoot (tag)

ToyRoot ← vmkit::gc

- Tagger les objets (stack maps)

```
[ ..... ]

ToyRoot* F (ToyRoot* param) {
 TOY_PARAM(param) ;
 TOY_VAR(ToyRoot, val) ;
       [ init val ... ]
 val = g(param, val) ;
 return val ;
}

ToyRoot* G (ToyRoot* a, ToyRoot* b){
 TOY_PARAM(a) ;
 TOY_PARAM(b) ;
 TOY_VAR(ToyRoot, res) ;
 res = a.doSomething(b) ;
 return res ;
}

[ ..... ]
```


ToyRoot (tag)

ToyRoot ← vmkit::gc

- Tagger les objets (stack maps)

```
[ ..... ]

ToyRoot* F (ToyRoot* param) {
 TOY_PARAM(param) ;
 TOY_VAR(ToyRoot, val) ;
       [ init val ... ]
 val = g(param, val) ;
 return val ;
}

ToyRoot* G (ToyRoot* a, ToyRoot* b){
 TOY_PARAM(a) ;
 TOY_PARAM(b) ;
 TOY_VAR(ToyRoot, res) ;
 res = a.doSomething(b) ;
 return res ;
}

[ ..... ]
```

Stack Maps

ToyRoot (tag)

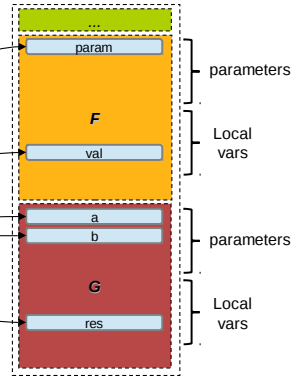
ToyRoot ← vmkit::gc

- Tagger les objets (stack maps)

```
[ ..... ]

ToyRoot* F (ToyRoot* param) {
 TOY_PARAM(param) ;
 TOY_VAR(ToyRoot, val) ;
       [ init val ... ]
 val = g(param, val) ;
 return val ;
}

ToyRoot* G (ToyRoot* a, ToyRoot* b){
 TOY_PARAM(a) ;
 TOY_PARAM(b) ;
 TOY_VAR(ToyRoot, res) ;
 res = a.doSomething(b) ;
 return res ;
}

[ ..... ]
```

Stack Maps

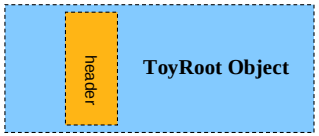
Call stack



ToyRoot (header gc)

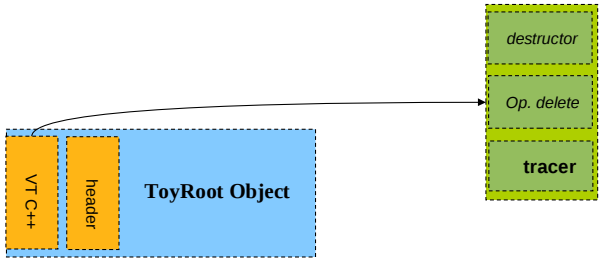
ToyRoot ← vmkit::gc

- Tagger les objets (stack maps)
- header GC
  - Bits de marquage
  - Bits de hash
  - Bits de lock (J3)
  - Bits personnalisables



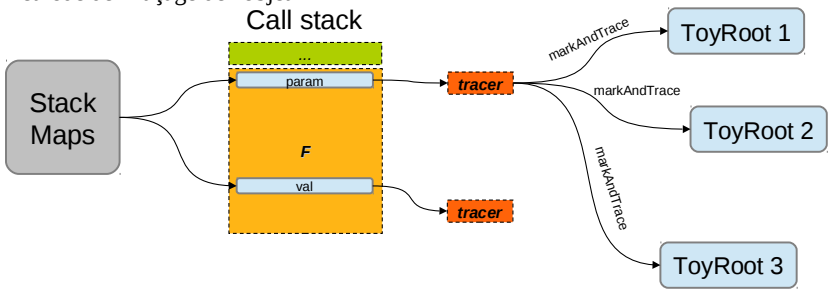
ToyRoot (header gc)

ToyRoot ← vmkit::gc
- Tagger les objets (stack maps)
- header GC
  - Bits de marquage
  - Bits de hash
  - Bits de lock (J3)
  - Bits personnalisables
- Méthode de Traçage de l'objet



ToyRoot (tracer)

ToyRoot ← vmkit::gc
- Tagger les objets (stack maps)
- header GC
  - Bits de marquage
  - Bits de hash
  - Bits de lock (J3)
  - Bits personnalisables
- Méthode de Traçage de l'objet

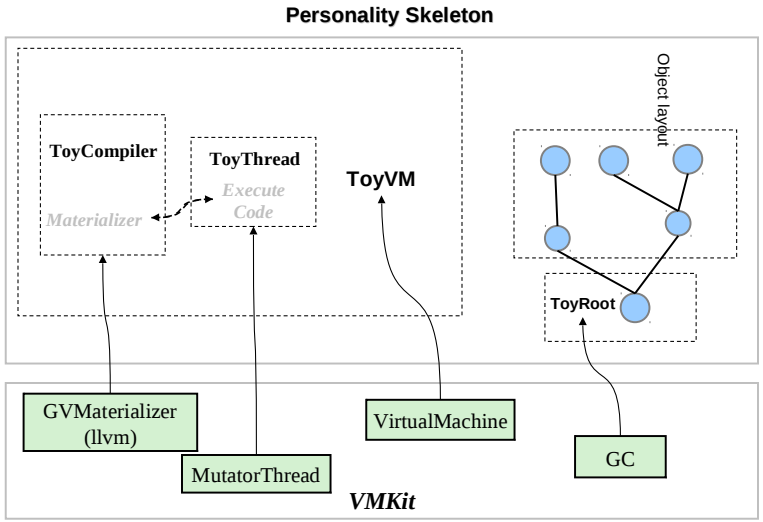


ToyRoot (tracer)

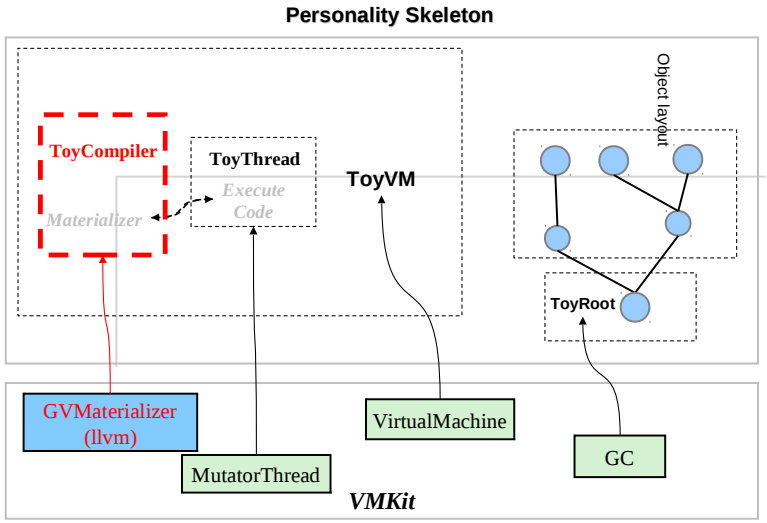
ToyRoot ← vmkit::gc
- Tagger les objets (stack maps)
- header GC
  - Bits de marquage
  - Bits de hash
  - Bits de lock (J3)
  - Bits personnalisables
- Méthode de Traçage de l'objet

- Allocateur GC
  - Surcharge *operator new*
  - Appel à *new* interdit pour gc (paramètre opaque)

Architecture de la ToyVM



Architecture de la ToyVM



ToyCompiler (load IR)

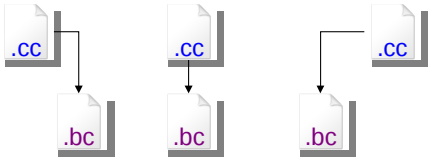
ToyCompiler ← GVMaterializer

- Récupération du code IR (représentation intermédiaire LLVM)

ToyCompiler (load IR)

ToyCompiler ← GVMaterializer

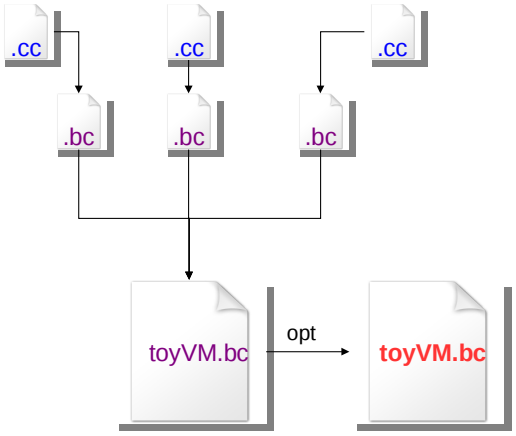
- Récupération du code IR (représentation intermédiaire LLVM)



ToyCompiler (load IR)

ToyCompiler ← GVMaterializer

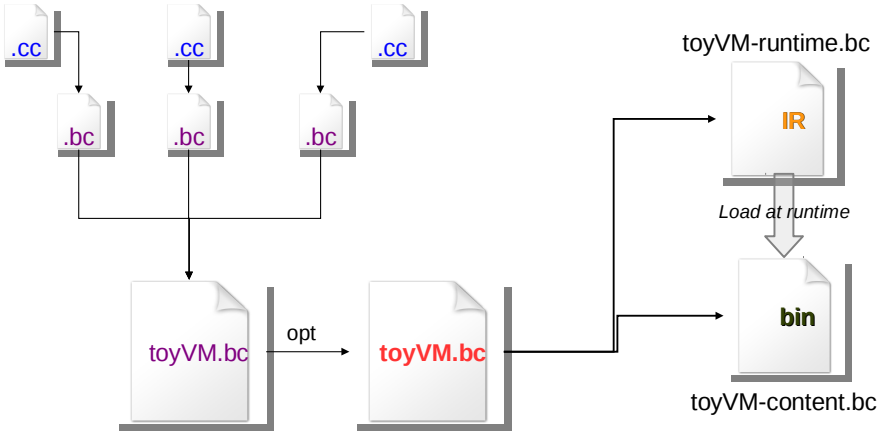
- Récupération du code IR (représentation intermédiaire LLVM)



ToyCompiler (load IR)

ToyCompiler ← GVMaterializer

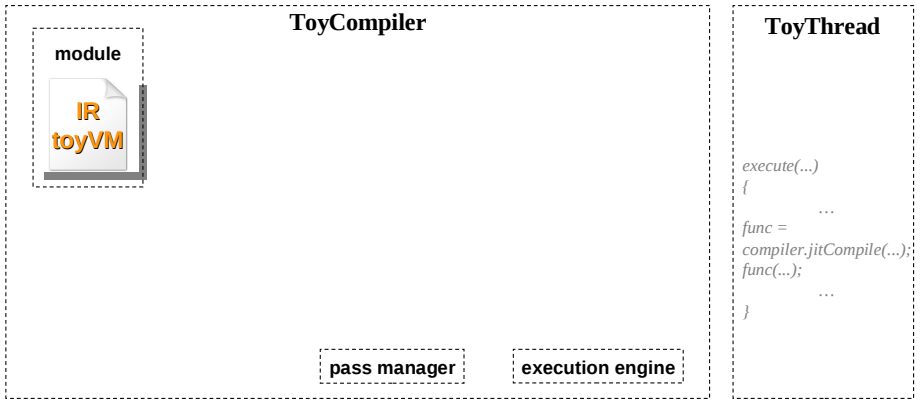
- Récupération du code IR (représentation intermédiaire LLVM)



ToyCompiler (generate IR)

ToyCompiler ← GVMaterializer

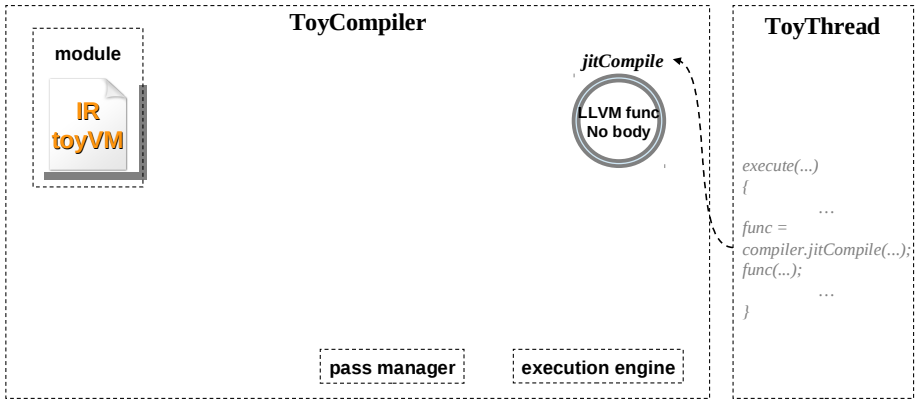
- Récupération du code IR (représentation intermédiaire LLVM)
- JIT compiling



ToyCompiler (generate IR)

ToyCompiler ← GVMaterializer

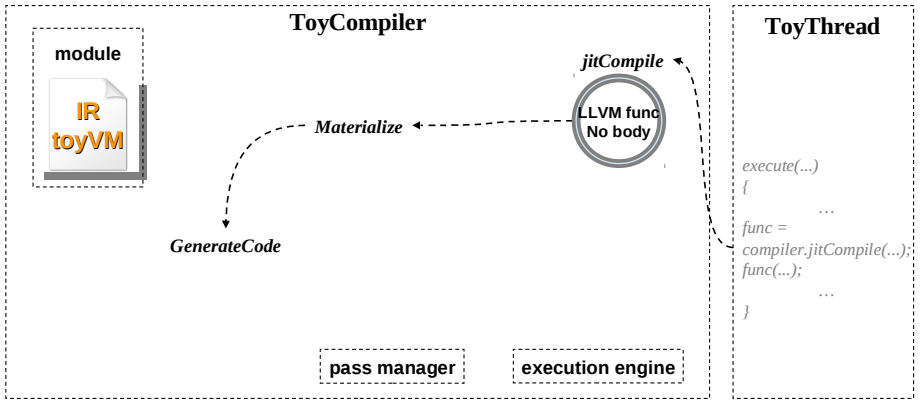
- Récupération du code IR (représentation intermédiaire LLVM)
- JIT compiling



ToyCompiler (generate IR)

ToyCompiler ← GVMaterializer

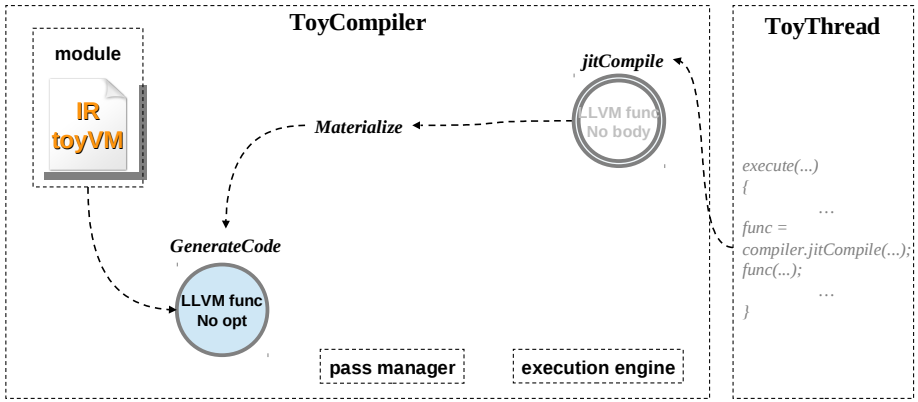
- Récupération du code IR (représentation intermédiaire LLVM)
- JIT compiling



ToyCompiler (generate IR)

ToyCompiler ← GVMaterializer

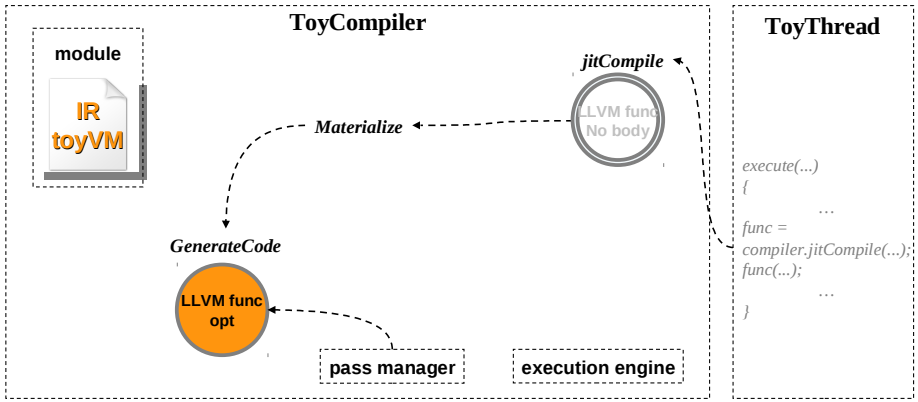
- Récupération du code IR (représentation intermédiaire LLVM)
- JIT compiling



ToyCompiler (optimize IR)

ToyCompiler ← GVMaterializer

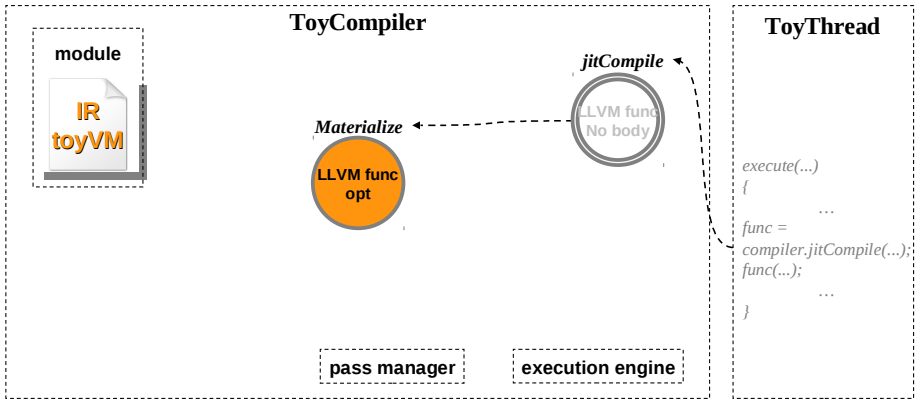
- Récupération du code IR (représentation intermédiaire LLVM)
- JIT compiling



ToyCompiler (optimize IR)

ToyCompiler ← GVMaterializer

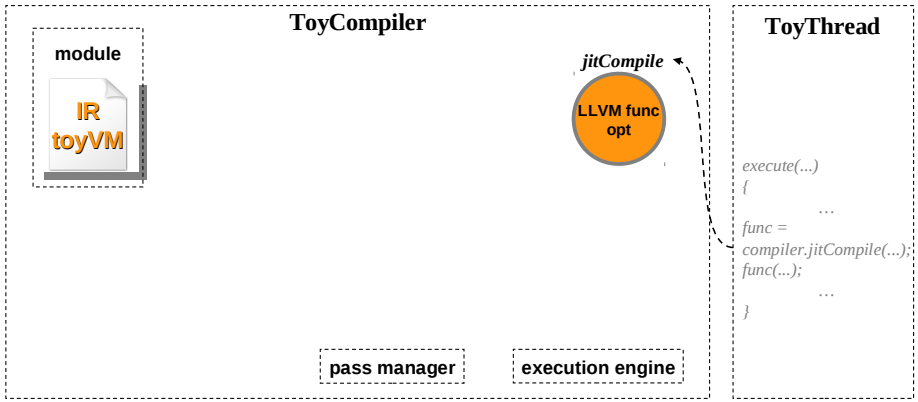
- Récupération du code IR (représentation intermédiaire LLVM)
- JIT compiling



ToyCompiler (optimize IR)

ToyCompiler ← GVMaterializer

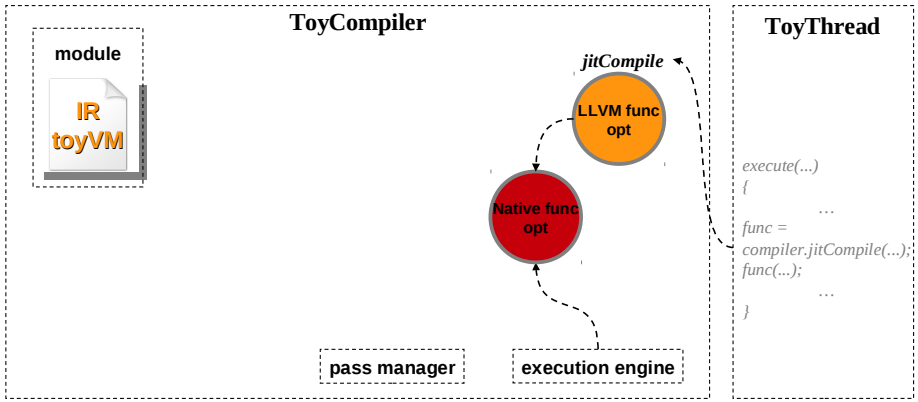
- Récupération du code IR (représentation intermédiaire LLVM)
- JIT compiling



ToyCompiler (IR to native)

ToyCompiler ← GVMaterializer

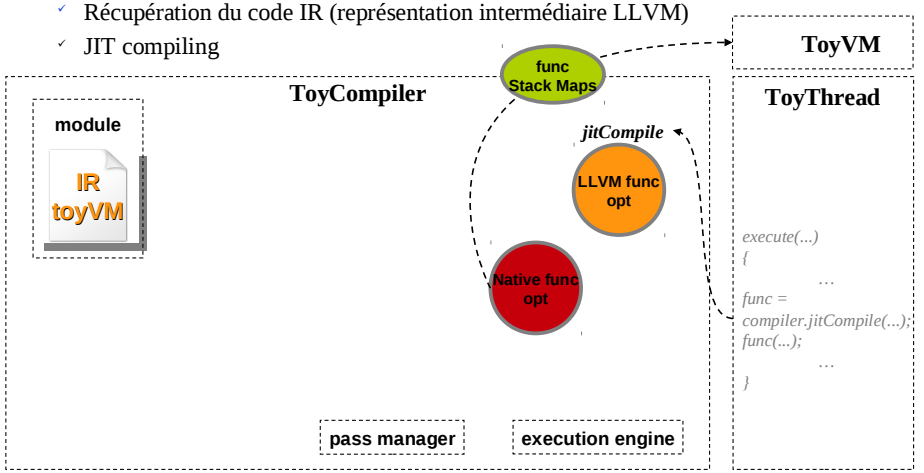
- Récupération du code IR (représentation intermédiaire LLVM)
- JIT compiling



ToyCompiler (GC Infos)

ToyCompiler ← GVMaterializer

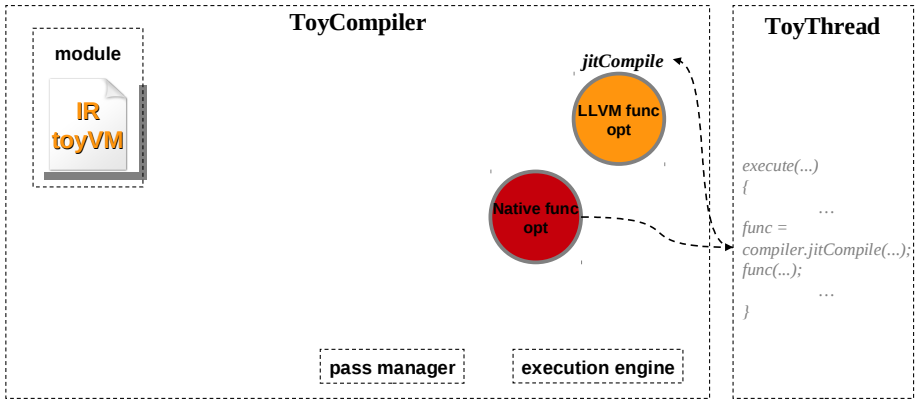
- Récupération du code IR (représentation intermédiaire LLVM)
- JIT compiling



ToyCompiler (IR to native)

ToyCompiler ← GVMaterializer

- Récupération du code IR (représentation intermédiaire LLVM)
- JIT compiling



ToyCompiler

ToyCompiler ← GVMaterializer

- Récupération du code IR (représentation intermédiaire LLVM)
- JIT compiling
  - Generation d'IR
  - Optimisation de l'IR
  - Conversion de l'IR vers code natif
- Transmission des StackMaps (information variables locales) au GC

Questions ?

A vos claviers !