# LLVM, Clang & VMKit:
## Nouvelles générations de compilateur

Nicolas Geoffray
Université Pierre et Marie Curie, France

nicolas.geoffray@lip6.fr

# Gros grain

- LLVM (Apple): Low Level Virtual Machine
  - Framework pour compilateurs
- Clang (Apple): Parser C/ObjectiveC/C++
  - Utilise LLVM comme back-end
- VMKit (Lip6): VMs Java et .Net
  - Utilise LLVM comme back-end

# Programme

- Qu'est-ce que LLVM?
- Qu'est-ce que Clang?
- Qu'est-ce que VMKit?
  - Design
  - Performance
- Recherche Système avec VMKit

# LLVM

# Qu'est-ce que LLVM?

- Un ensemble d'instructions
  - Orienté registres
- Un ensemble d'optimisations
  - Optimisations compilateur standard (propagation de constantes, déroulement de boucles, ...)
- Un générateur de code
  - X86, X86_64, PPC, PPC64, ARM, Thumb, Sparc, Alpha, IA64, MIPS

# llvm-gcc:
## Frontend C/C++/ObjC...

- Port de GCC vers LLVM
  - De Gimple au bytecode LLVM
- make CC=llvm-gcc
- Présent dans XCode 3.1

# Pourquoi un nouveau compilateur?

- gcc -O3 MonProg.c -o main.exe
  - *Et si je connais mieux comment optimiser mon programme?*
  - *Et si je connais mieux la gestion de mes “alias”?*
- ld MonProg.o AutreProg.o -o main.exe
  - *Et si je veux inliner des fonctions de AutreProg.o dans MonProg.o ?*
- ./main.exe 4 2
  - *Et si la connaissance des inputs du programme peut influer les performances?*

# Différences avec GCC

- Insertion de passes utilisateur
  - Aujourd'hui plugins GCC
- Link-time Optimization
  - Aujourd'hui branche dans GCC
- Compilateur JIT
  - Pas d'équivalent dans GCC
- Licence BSD
  - GCC est GPL

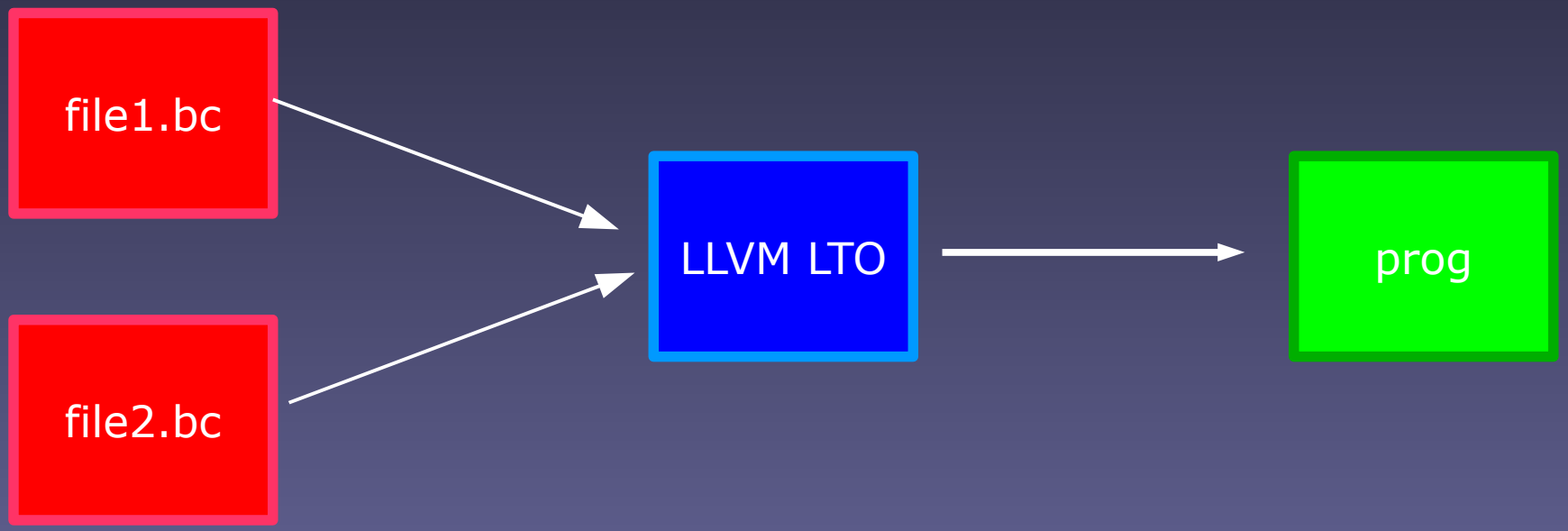
# IR manipulable

- Format de fichier
  - .ll file: similaire à de l'assembleur
  - .bc file: fichier bytecode

```
define i32 @add(i32 %a, i32 %b)  {
entry:
    %tmp3 = add i32 %b, %a
    ret i32 %tmp3
}
```

# Link-time optimization

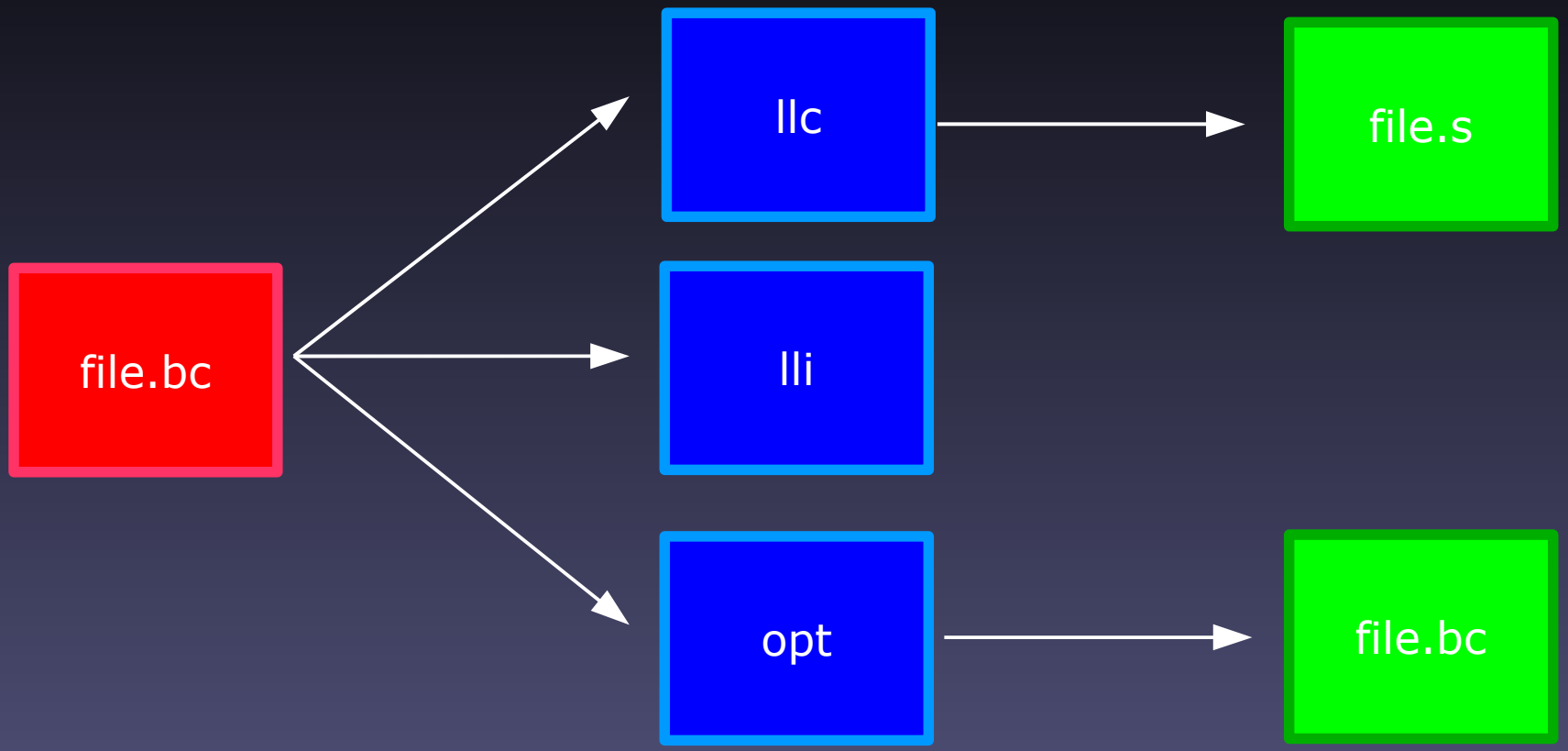
```
nicolas-geoffrays-macbook-pro:~ varth$ gcc -c file1.c
nicolas-geoffrays-macbook-pro:~ varth$ gcc -c file2.c
nicolas-geoffrays-macbook-pro:~ varth$ gcc -o prog file1.o file2.o
```



# Compilateur Just-in-time

- Génération de code en mémoire, pas en fichier
  - Utilise les même bibliothèques
- Spécialisation de code
  - En fonction des inputs
- Fonctions compilées juste-à-temps
  - “Lazy compilation”

# Design modulaire




Pour exécuter votre optimisation:

```
> opt -load apass.so file.bc
```

# Utilisateurs

- Industrie
  - Cray: Optimisations de back-end
  - Adobe: JIT pour flash
  - Apple: JIT pour OpenGL dans MacOS X 10.5
  - Sun: vérifier statique de code
  - Siemens: génération de code VLIW
- Recherche
  - Secure Virtual Architecture
  - Allocation Registres
  - Transactions
  - Programmation orientée Aspects

# Clang

# Qu'est-ce que Clang?

- Front-end C, ObjectiveC, C++ pour LLVM
- En remplacement direct de GCC
- Motivation
  - Meilleurs temps de compilation et utilisation mémoire
  - Messages d'erreur plus verbeux
  - Outils pour programmeurs

# Outils pour programmeurs

- Clang dans les IDEs
- Analyse statique
  - bad argument, bad dereference, dead increment, dead initialization, dead store, divide-by-zero, null dereference, uninitialized value, ...
  - Beaucoup de bugs trouvés dans FreeBSD:

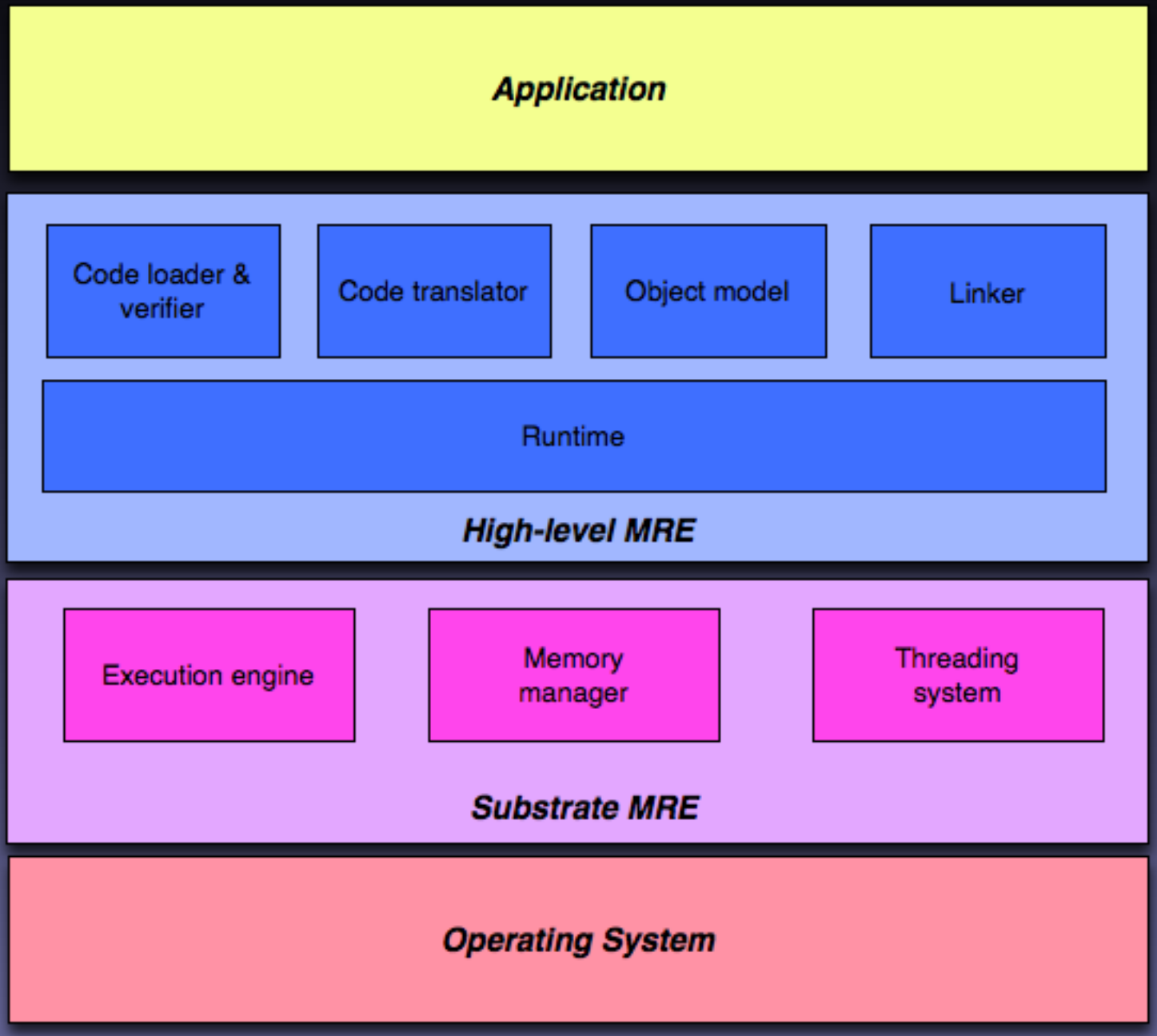
    http://www.daemonology.net/tmp/2008-08-03-1/

# VMKit

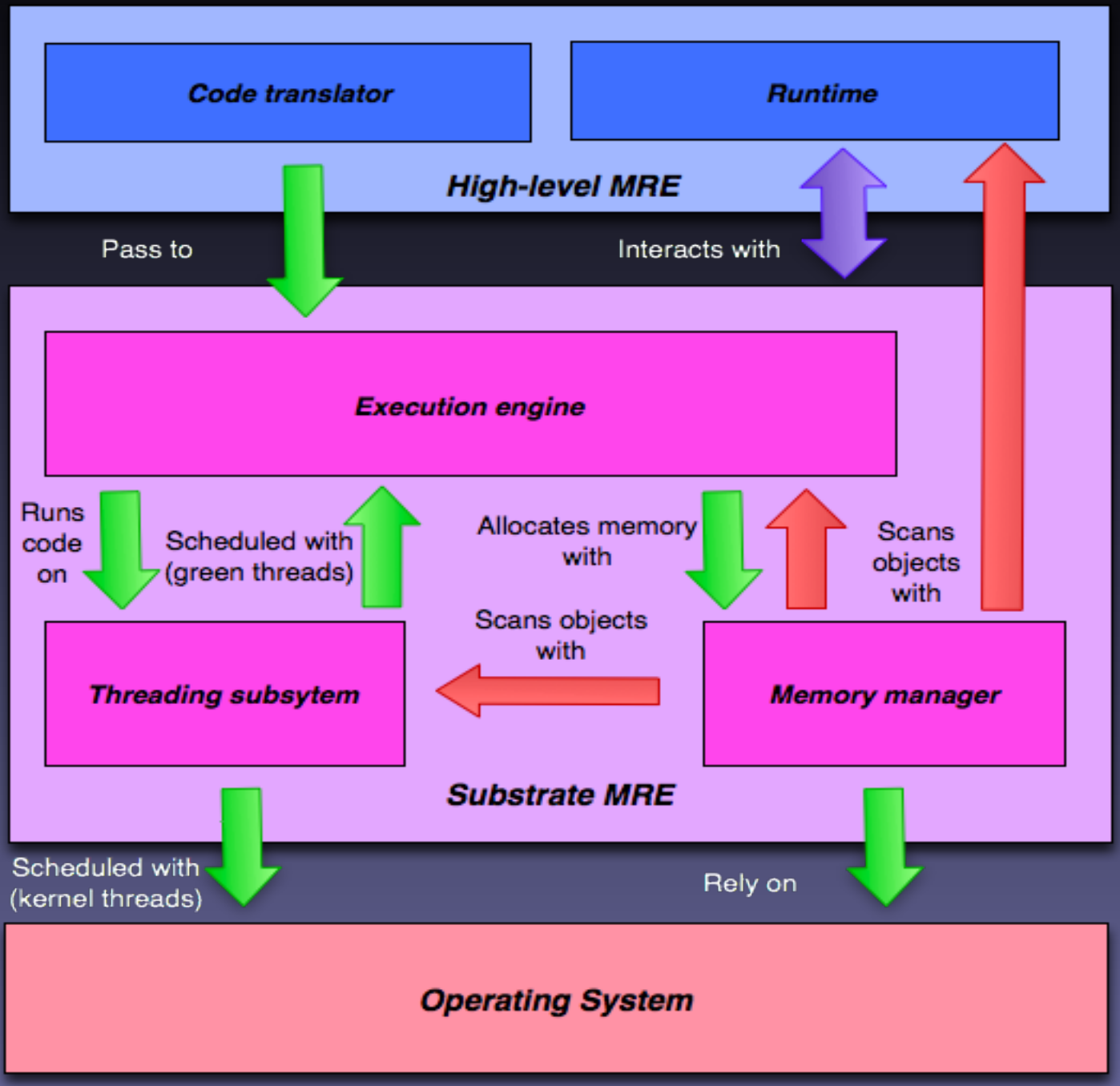
# Qu'est-ce que VMKit?

- Une VM bas-niveau qui facilite le développement de VMs haut-niveau
- Une implémentation de la JVM et de .Net
- Utilisation:
  - vmkit -java HelloWorld
  - vmkit -net HelloWorld.exe

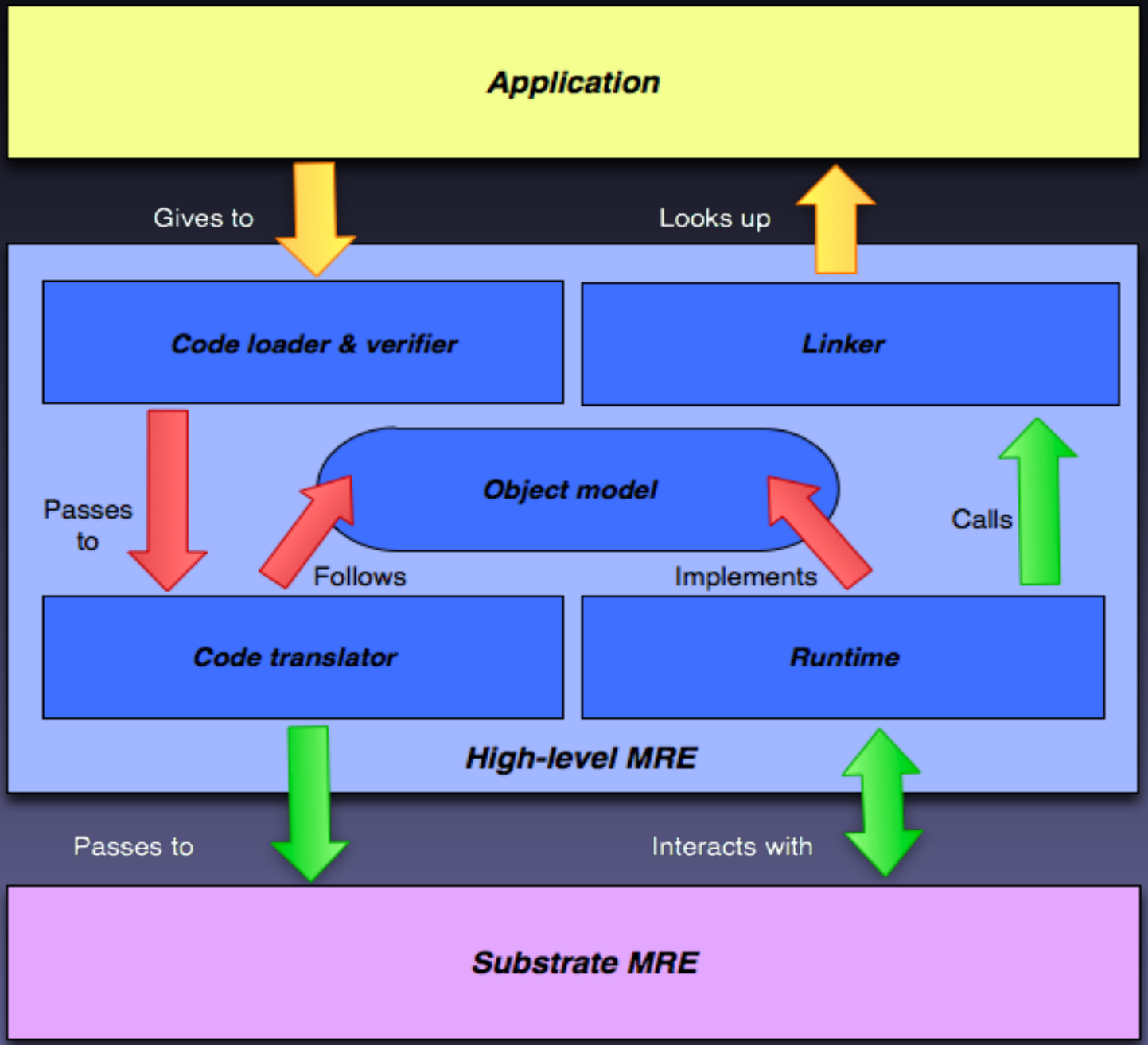
# Vue d'ensemble



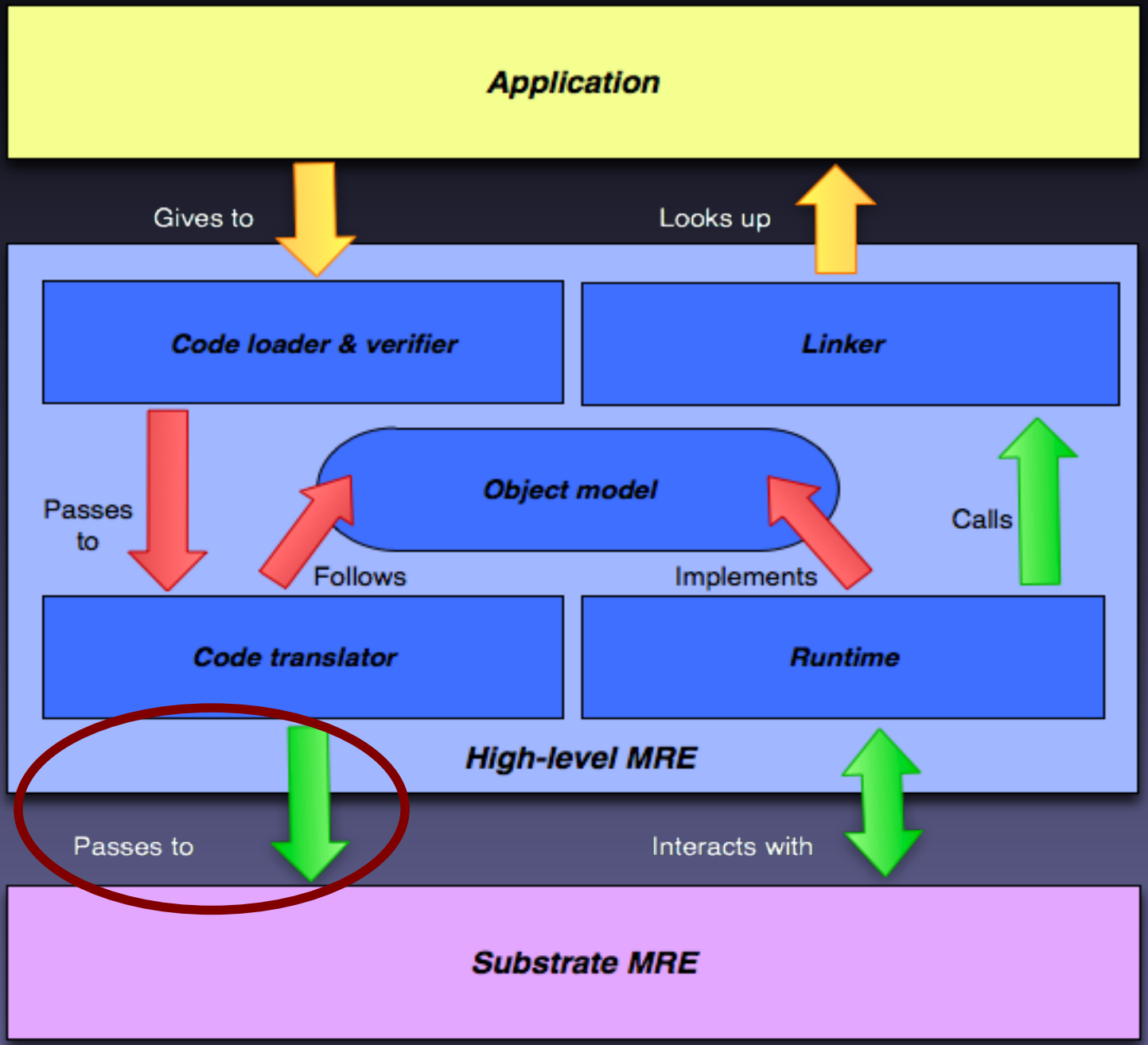
# Dans VMKit


Code translator
Runtime
High-level MRE
Pass to
Interacts with
Execution engine
Runs code on
Scheduled with (green threads)
Allocates memory with
Scans objects with
Scans objects with
Threading subsytem
Memory manager
Substrate MRE
Scheduled with (kernel threads)
Rely on
Operating System

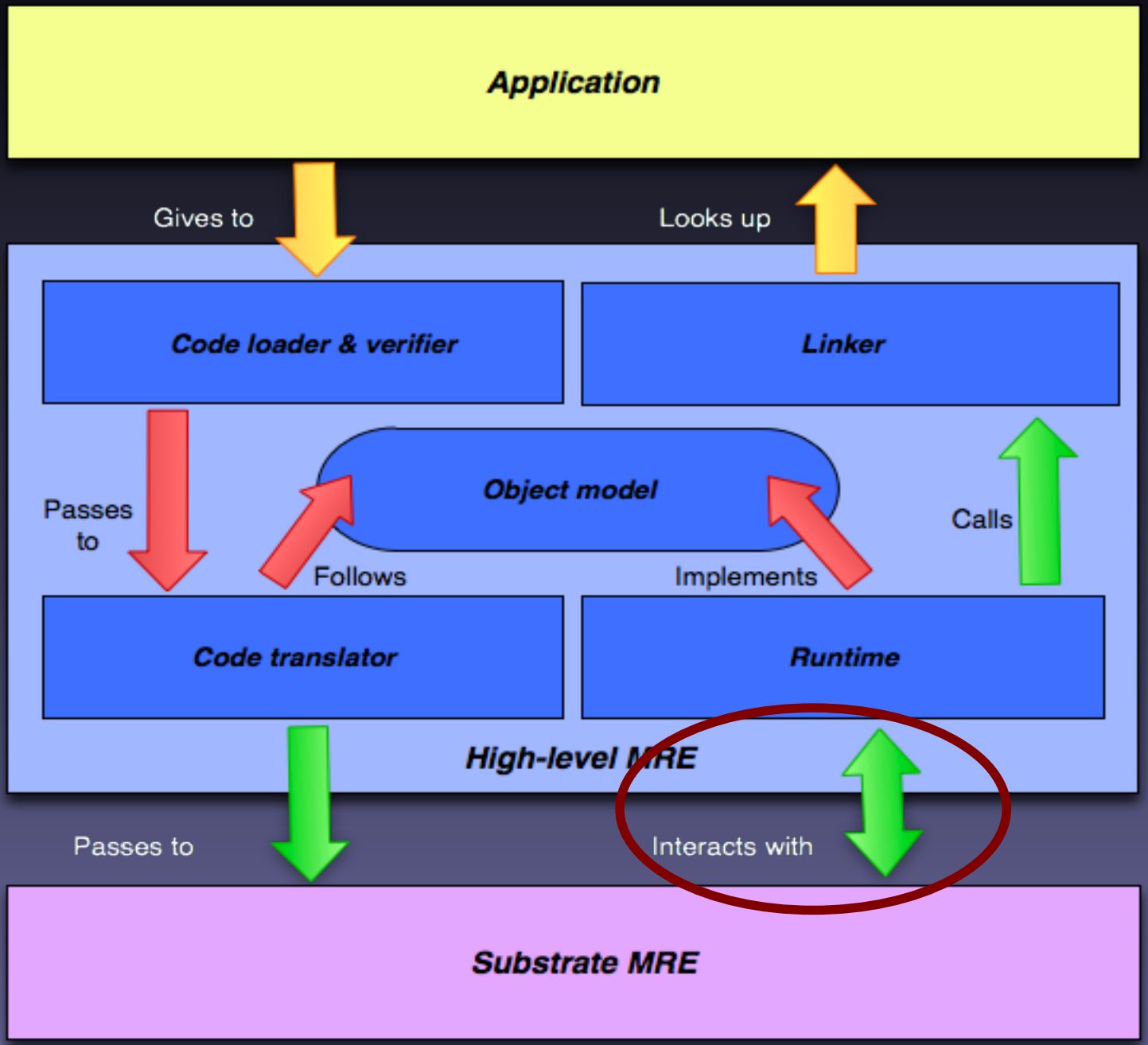
# Une VM haut-niveau



# Une VM haut-niveau



# Une VM haut-niveau



# Interactions
# VM haut-niveau <--> VM Bas-niveau

- La VM haut-niveau
  - Alloue avec la VM bas-niveau
  - Crée des threads avec la VM bas-niveau
  - Exécution de l'appli. avec VM bas-niveau
- La VM bas-niveau
  - Résoud les symboles avec la VM haut-niveau
  - Trace la mémoire avec la VM haut-niveau
  - Appelle la VM haut-niveau pour les fonctions du runtime

# Implémentation de VMKit

**VMKit link des composants existants**

- Execution Engine: LLVM
- Memory Manager: Boehm
- Threads: Posix threads

# LLVM comme “execution engine”

## Deux possibilités

- LLVM JIT --> Génère en-mémoire
- LLVM AOT --> Génère .so/.exe

## Fonctionnalités manquantes

- Interprète
- Optimisations dynamiques
- Scan précis de la pile

# PosixThreads: Threads noyaux

## Utilisé pour:

- Verrouillage
- Scheduler

## Fonctionnalités manquantes

- Thin locks (implémenté dans VMKit)
- TLS efficace (implémenté dans VMKit)

# Boehm comme gestionnaire mémoire

## Garbage Collector

- Conservatif (tas + pile)
- Non intrusif

## Fonctionnalités manquantes

- Scan précis du tas (implémenté dans VMKit)
- Scan précis de la pile
- Algos pour GC non-conservatifs

# Exécution type (JVM, .Net)

- Charge .class and .jar (.exe)
- JIT main()
  - Insère des *stubs* pour les méthodes (“lazy compilation”) et les champs
- Exécute main()
  - Stubs appellent la VM haut-niveau
  - Chargement de class (assembly) dynamique

# Translation de  bytecode

Tous les bytecode JVM (MSIL) sont traduits vers LLVM

- Translation un --> un (e.g. add, sub, div, affectations et lectures de variable, appels static ...)
- Translation un --> plusieurs (e.g. opérations sur tableaux, opérations sur les champs, appels virtuels ...)
- Translation un --> appel runtime (e.g. exceptions, héritage, synchronisations)

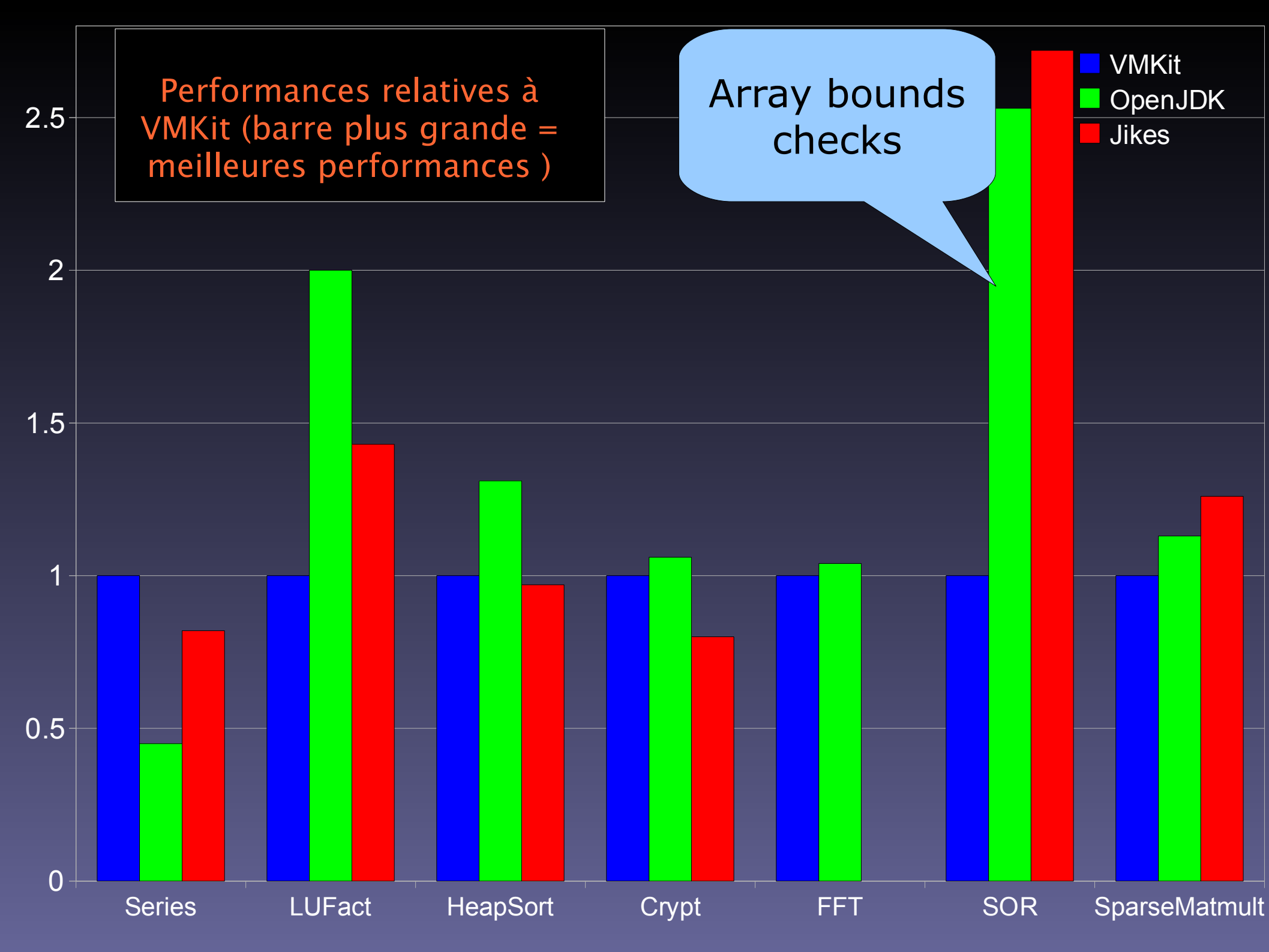
# Portabilité

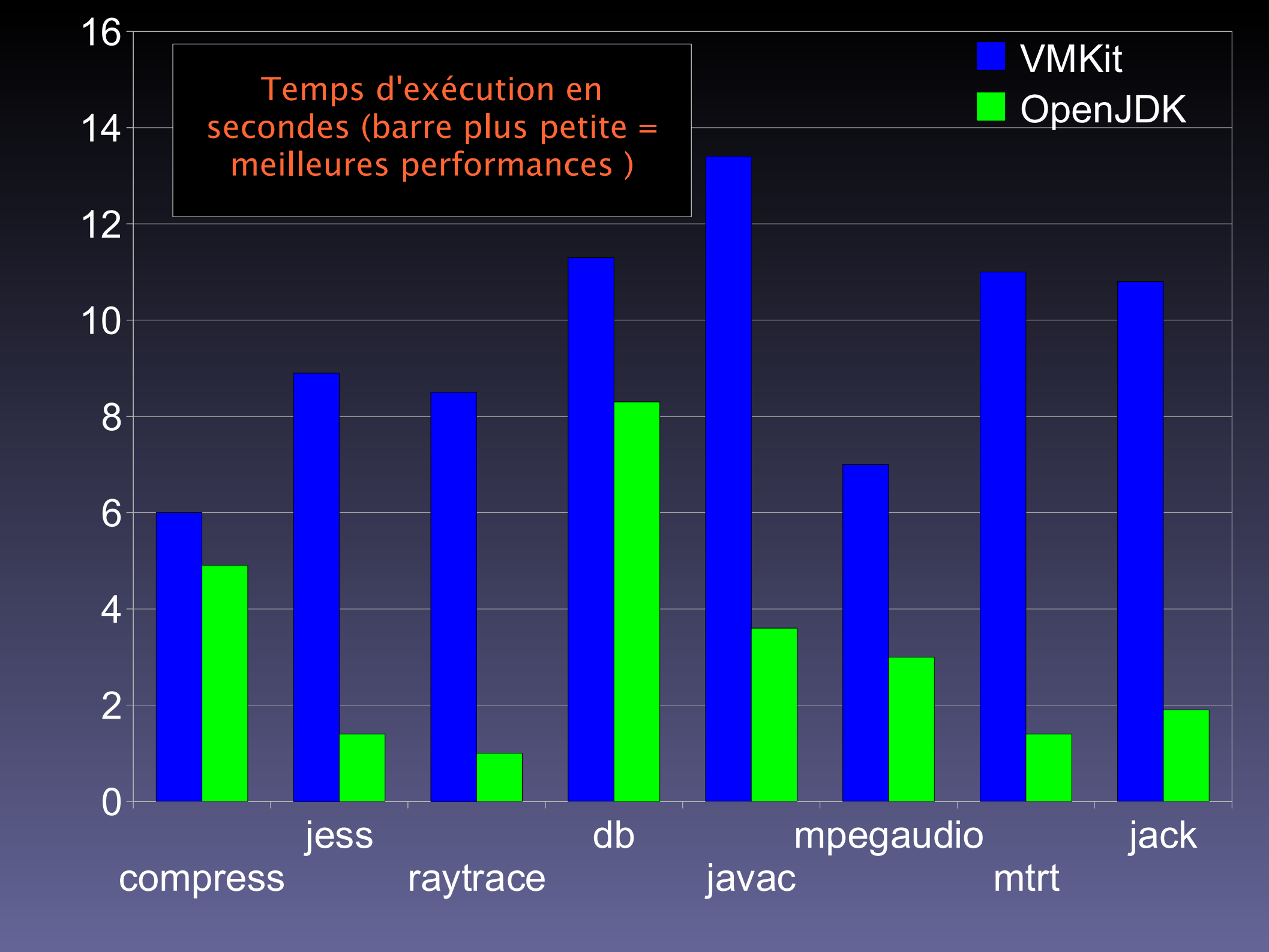
| OS/Arch | JIT | AOT | GC |
|---|---|---|---|
| Linux/x86 | ☑ | ☑ | Boehm |
| MacOSX/x86 | ☑ | ☒ | Boehm |
| Linux/x64 | ☑ | ☒ | Malloc |
| Linux/ppc32 | ☑ | ☒ | Boehm |
| MacOSX/ppc32 | ☑ | ☒ | Boehm |
| ../ARM, PPC64 | ? | ? | ? |
| .../IA64, Mips, ... | ☒ | ? | ? |

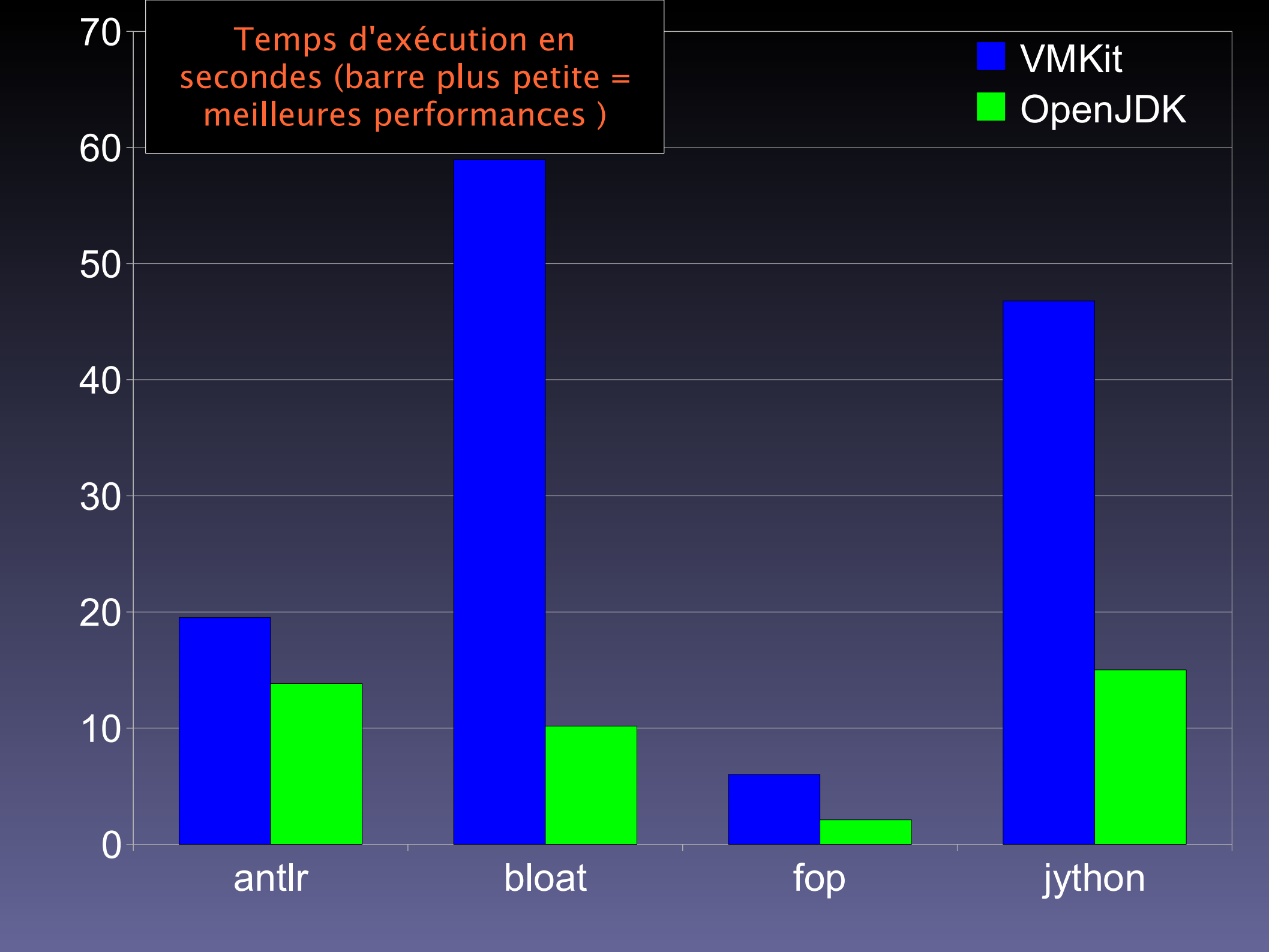
# Performances de VMKit

# Benchmarks JVM

- Pentium D 3GHz (x86_32), 3Go, Linux
- 2 JVMs
    - OpenJDK, Jikes RVM
- Java Grande Forum Benchmark
    - Section2: benchmarks CPU-intensifs
- SPEC JVM98
    - Benchmarks CPU- et Mémoire- intensifs
- Dacapo
    - Benchmarks VM-intensifs

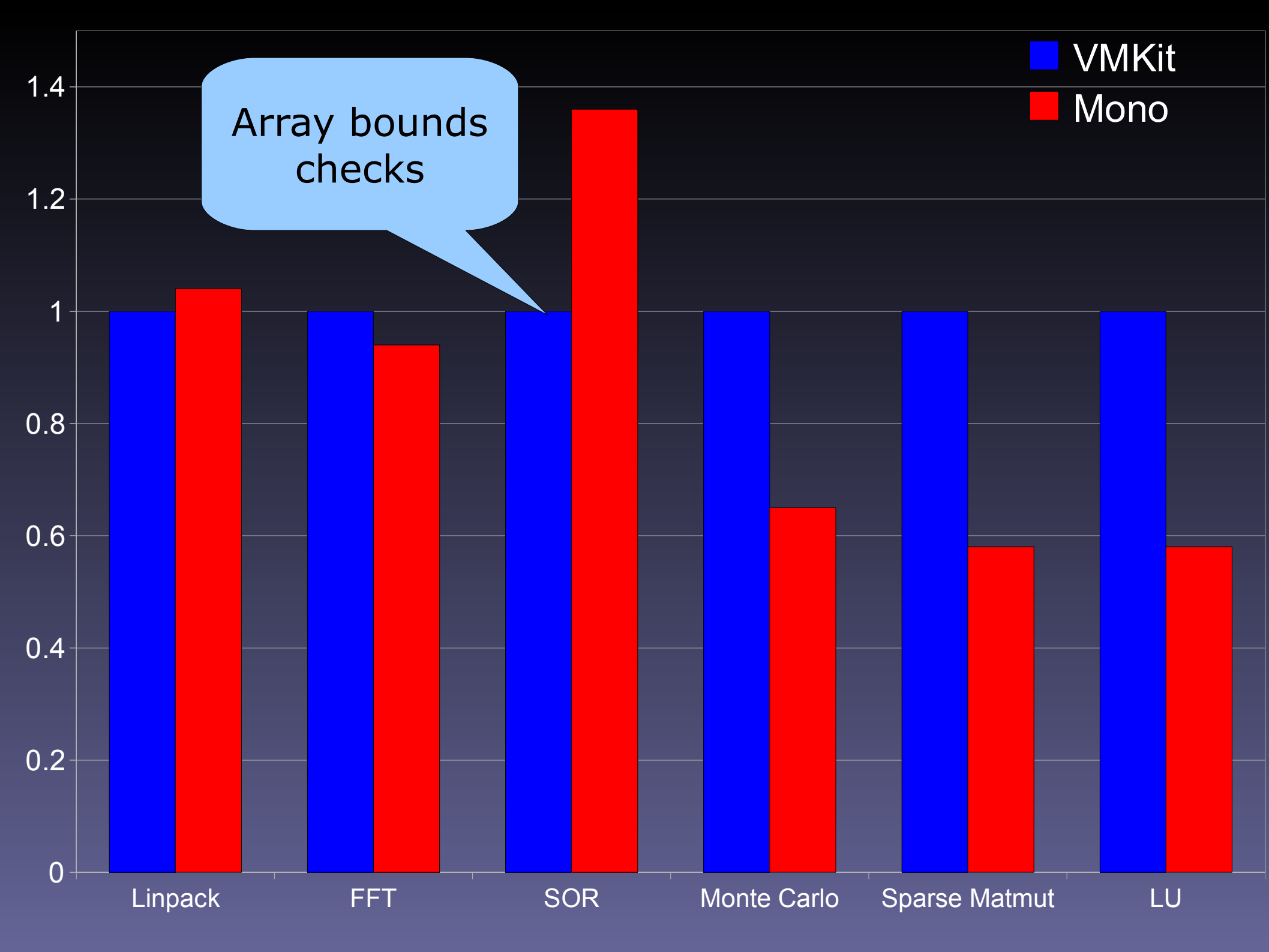

Performances relatives à VMKit (barre plus grande = meilleures performances )
Array bounds checks
VMKit
OpenJDK
Jikes
2.5
2
1.5
1
0.5
0
Series
LUFact
HeapSort
Crypt
FFT
SOR
SparseMatmult


Temps d'exécution en secondes (barre plus petite = meilleures performances )
VMKit
OpenJDK
16
14
12
10
8
6
4
2
0
compress
jess
raytrace
db
javac
mpegaudio
mtrt
jack


Temps d'exécution en secondes (barre plus petite = meilleures performances )
VMKit
OpenJDK
70
60
50
40
30
20
10
0
antlr
bloat
fop
jython

# Benchmarks .Net

- Athlon XP 1800+, 512M, Linux
- Comparaisons avec Mono
- Pas de comparaisons avec Microsoft
- PNetMark
  - Applications CPU-intensives


VMKit
Mono
Array bounds checks
1.4
1.2
1
0.8
0.6
0.4
0.2
0
Linpack
FFT
SOR
Monte Carlo
Sparse Matmut
LU

# Temps de compilation

- VMKit compile toutes les méthodes
  - Pas d'interprétation
  - Pas d'optimisations dynamiques

Influe sur le temps de chargement

# Démarrage Tomcat (OpenJDK)

Jul 31, 2008 7:17:39 PM org.apache.coyote.http11.Http11Protocol init
INFO: Initializing Coyote HTTP/1.1 on http-8080
Jul 31, 2008 7:17:39 PM org.apache.catalina.startup.Catalina load
INFO: Initialization processed in 1367 ms
Jul 31, 2008 7:17:40 PM org.apache.catalina.core.StandardService start
INFO: Starting service Catalina
Jul 31, 2008 7:17:40 PM org.apache.catalina.core.StandardEngine start
INFO: Starting Servlet Engine: Apache Tomcat/6.0.16
Jul 31, 2008 7:17:40 PM org.apache.coyote.http11.Http11Protocol start
INFO: Starting Coyote HTTP/1.1 on http-8080
Jul 31, 2008 7:17:40 PM org.apache.jk.common.ChannelSocket init
INFO: JK: ajp13 listening on /0.0.0.0:8009
Jul 31, 2008 7:17:40 PM org.apache.jk.server.JkMain start
INFO: Jk running ID=0 time=0/44  config=null
Jul 31, 2008 7:17:40 PM org.apache.catalina.startup.Catalina start
INFO: Server startup in 1142 ms

# Démarrage Tomcat (VMKit avec Opt)

Jul 31, 2008 6:35:51 PM org.apache.coyote.http11.Http11Protocol init
INFO: Initializing Coyote HTTP/1.1 on http-8080
Jul 31, 2008 6:35:51 PM org.apache.catalina.startup.Catalina load
INFO: Initialization processed in 15020 ms
Jul 31, 2008 6:35:54 PM org.apache.catalina.core.StandardService start
INFO: Starting service Catalina
Jul 31, 2008 6:35:54 PM org.apache.catalina.core.StandardEngine start
INFO: Starting Servlet Engine: Apache Tomcat/6.0.16
Jul 31, 2008 6:36:11 PM org.apache.coyote.http11.Http11Protocol start
INFO: Starting Coyote HTTP/1.1 on http-8080
Jul 31, 2008 6:36:13 PM org.apache.jk.common.ChannelSocket init
INFO: JK: ajp13 listening on /0.0.0.0:8009
Jul 31, 2008 6:36:13 PM org.apache.jk.server.JkMain start
INFO: Jk running ID=0 time=19/738  config=null
Jul 31, 2008 6:36:13 PM org.apache.catalina.startup.Catalina start
INFO: Server startup in 22660 ms

# Démarrage Tomcat (VMKit sans Opt)

Jul 31, 2008 6:51:42 PM org.apache.coyote.http11.Http11Protocol init
INFO: Initializing Coyote HTTP/1.1 on http-8080
Jul 31, 2008 6:51:42 PM org.apache.catalina.startup.Catalina load
INFO: Initialization processed in 10219 ms
Jul 31, 2008 6:51:44 PM org.apache.catalina.core.StandardService start
INFO: Starting service Catalina
Jul 31, 2008 6:51:44 PM org.apache.catalina.core.StandardEngine start
INFO: Starting Servlet Engine: Apache Tomcat/6.0.16
Jul 31, 2008 6:51:57 PM org.apache.coyote.http11.Http11Protocol start
INFO: Starting Coyote HTTP/1.1 on http-8080
Jul 31, 2008 6:51:59 PM org.apache.jk.common.ChannelSocket init
INFO: JK: ajp13 listening on /0.0.0.0:8009
Jul 31, 2008 6:52:00 PM org.apache.jk.server.JkMain start
INFO: Jk running ID=0 time=16/679  config=null
Jul 31, 2008 6:52:00 PM org.apache.catalina.startup.Catalina start
INFO: Server startup in 17729 ms

# VMKit: Temps de compilation avec Opt

| ---User Time--- | --System Time-- | --User+System | --Wall Time--- | --- Name --- |
|---|---|---|---|---|
| 1.2840 ( 8.0%) | 17.5490 ( 52.8%) | 18.8331 ( 38.4%) | 19.6628 ( 39.1%) | X86 DAG->DAG Instruction Selection |
| 3.2201 ( 20.2%) | 0.7200 ( 2.1%) | 3.9402 ( 8.0%) | 4.2851 ( 8.5%) | Unswitch loops |
| 3.3322 ( 20.9%) | 0.7800 ( 2.3%) | 4.1122 ( 8.3%) | 4.1940 ( 8.3%) | Predicate Simplifier |
| 1.1120 ( 7.0%) | 0.8240 ( 2.4%) | 1.9361 ( 3.9%) | 1.9569 ( 3.8%) | Linear Scan Register Allocator |
| 1.0640 ( 6.6%) | 0.8800 ( 2.6%) | 1.9441 ( 3.9%) | 1.9235 ( 3.8%) | Live Variable Analysis |
| 0.8200 ( 5.1%) | 0.9280 ( 2.7%) | 1.7481 ( 3.5%) | 1.5910 ( 3.1%) | Live Interval Analysis |
| 0.6920 ( 4.3%) | 0.6760 ( 2.0%) | 1.3680 ( 2.7%) | 1.3666 ( 2.7%) | Global Value Numbering |
| 0.6640 ( 4.1%) | 0.5880 ( 1.7%) | 1.2520 ( 2.5%) | 1.2098 ( 2.4%) | Simple Register Coalescing |
| 0.3480 ( 2.1%) | 0.4520 ( 1.3%) | 0.8000 ( 1.6%) | 0.7449 ( 1.4%) | Combine redundant instructions |
| 0.1640 ( 1.0%) | 0.3400 ( 1.0%) | 0.5040 ( 1.0%) | 0.5229 ( 1.0%) | Combine redundant instructions |
| 15.8850 (100.0%) | 33.1980 (100.0%) | 49.0830 (100.0%) | 50.2414 (100.0%) | TOTAL |

# VMKit: Temps de compilation sans Opt

| ---User Time--- | --System Time-- | --User+System-- | --Wall Time--- | --- Name --- |
|---|---|---|---|---|
| 2.0081 ( 24.9%) | 24.4535 ( 79.4%) | 26.4616 ( 68%) | 27.2189 ( 68%) | X86 DAG->DAG Instruction Selection |
| 1.6041 ( 19.9%) | 0.9960 ( 3.2%) | 2.6001 ( 6.7%) | 2.5746 ( 6.5%) | Live Variable Analysis |
| 1.2680 ( 15.7%) | 0.9920 ( 3.2%) | 2.2601 ( 5.8%) | 2.3452 ( 5.9%) | Live Interval Analysis |
| 1.2320 ( 15.3%) | 0.8440 ( 2.7%) | 2.0761 ( 5.3%) | 2.0690 ( 5.2%) | Linear Scan Register Allocator |
| 1.0000 ( 12.4%) | 0.6280 ( 2.0%) | 1.6280 ( 4.1%) | 1.5355 ( 3.8%) | Simple Register Coalescing |
| 0.3680 ( 4.5%) | 0.4320 ( 1.4%) | 0.8000 ( 2.0%) | 0.7780 ( 1.9%) | Control Flow Optimizer |
| | | | | |
| 8.0444 (100.0%) | 30.7618 (100.0%) | 38.8063 (100.0%) | 39.5548 (100.0%) | TOTAL |

# Conclusion sur VMKit:

## Encore beaucoup d'optimisations à faire!

- Au niveau de LLVM
  - Scan précis de la pile
  - Optimisations dynamiques
- Au niveau de VMKit
  - Utiliser un GC précis (en cours: portage de MMTk)

# Recherche ***Système*** avec VMKit

# Sécurité dans OSGi

## Qu'est-ce que OSGi?

- Une plateforme orientée services
- *Pour les nuls*: un modèle de chargement de classe dans la JVM meilleur: un service = un classloader
- Les services s'exécutent dans une même JVM et communiquent par appels de méthode

**Problème**: ***quid des modules malicieux dans les plateformes ouvertes (set-top-box)?***

# Sécurité dans OSGi

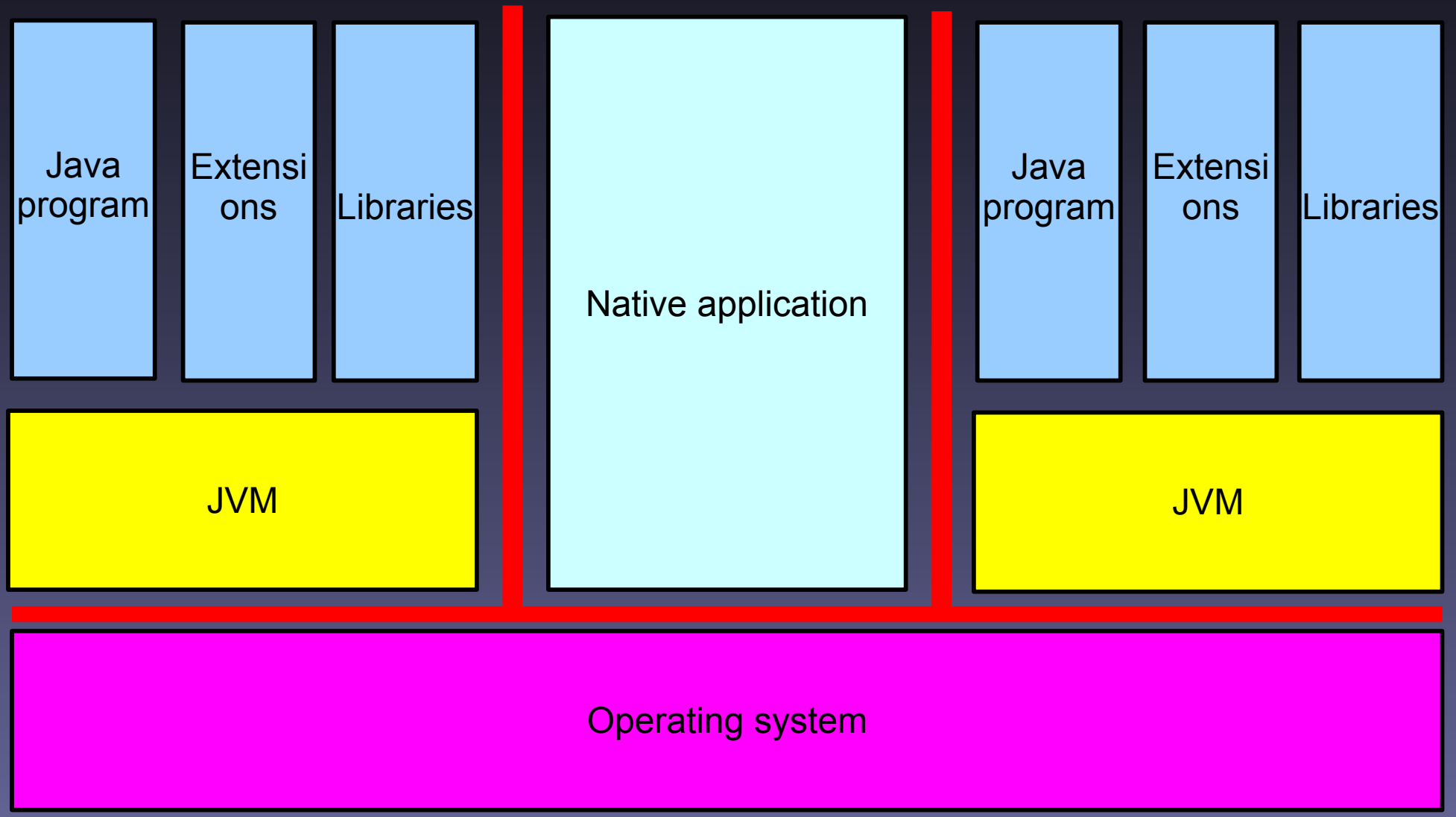
## Notre contribution:

- Isoler chaque class loader, similaire à la spécification Java *Isolates*
- Comptage de ressources pas service
- Support pour la terminaison de services

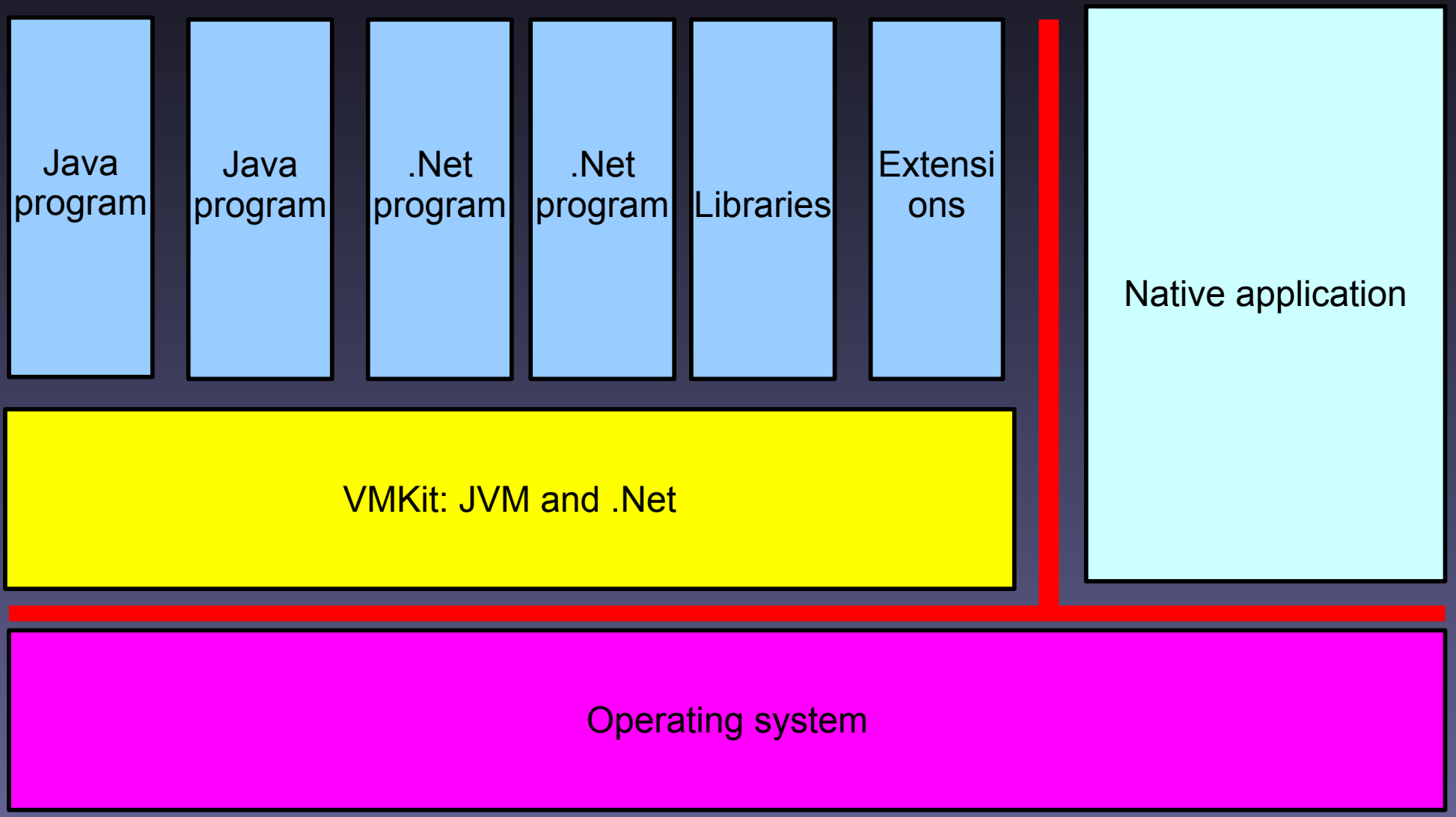
## Travaux futurs:

- Plateforme autonome

# Isolation actuelle



# Avec VMKit


Java program
Java program
.Net program
.Net program
Libraries
Extensions
Native application
VMKit: JVM and .Net
Operating system

# Conclusion

- LLVM, Clang, VMKit
  - Beaucoup d'améliorations à venir
  - Apple contribue beaucoup à LLVM (pour remplacer gcc?)
- Recherche système avec VMKit
  - Expérience montre sa simplicité
  - Performances homogènes

Pour plus d'informations

**http://vmkit.llvm.org**

**http://llvm.org**
**http://clang.llvm.org**