How applications are run on Android ?

LSE
Security System
Laboratory of Epita

Jean-Loup Bogalho & Jérémy Lefaure
clippix@lse.epita.fr
blatinox@lse.epita.fr

Table of contents

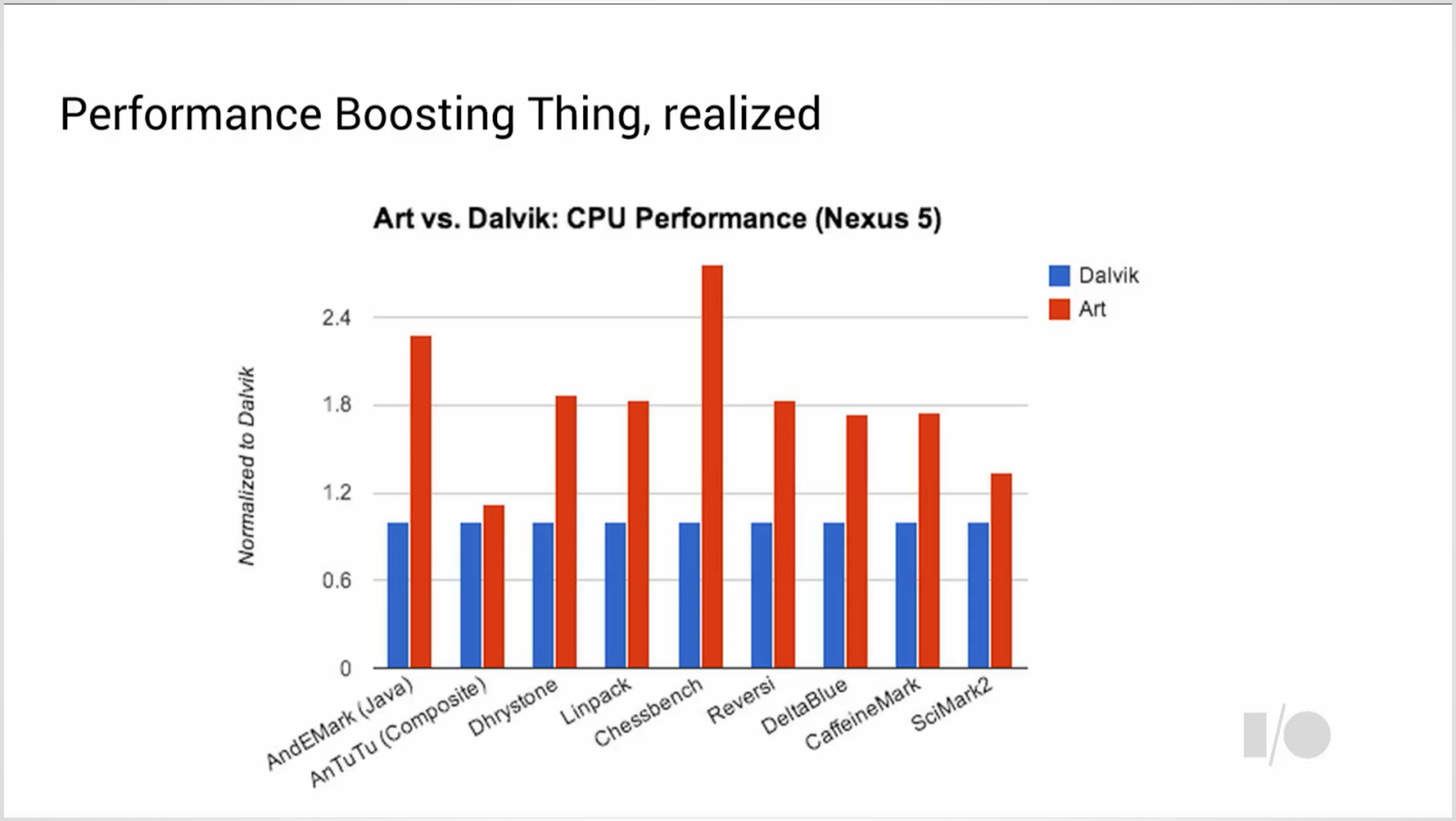
What is Dalvik ?

- Android’s Virtual Machine
- Designed to run on embedded systems
- Register-based (lower memory consumption)
- Run Dalvik Executable (.dex) files

What is ART ?

- Android RunTime
- Dalvik’s successor
- ART Is Not a JVM
- Huge performance gain thanks to ahead-of-time (AOT) compilation
- Available in Android 4.4

What is ART ?

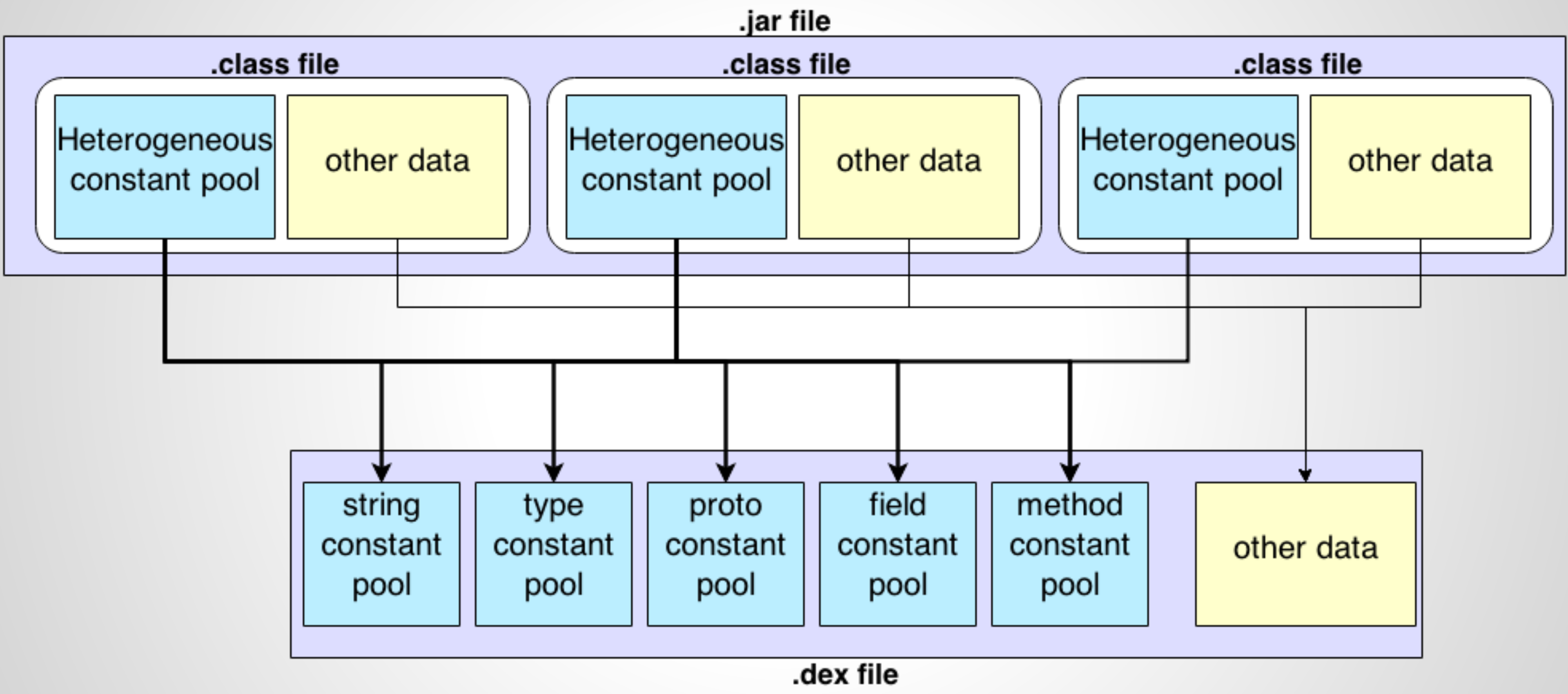

Performance Boosting Thing, realized
Art vs. Dalvik: CPU Performance (Nexus 5)
Dalvik
Art
Normalized to Dalvik
2.4
1.8
1.2
0.6
0
AndEMark (Java)
AnTuTu (Composite)
Dhrystone
Linpack
Chessbench
Reversi
DeltaBlue
CaffeineMark
SciMark2

Executable files

Dalvik: .dex files

- Not the same bytecode as classical Java bytecode
- .class files are converted in .dex files at build time
- Optimized for minimal memory footprint

Dalvik: .dex files



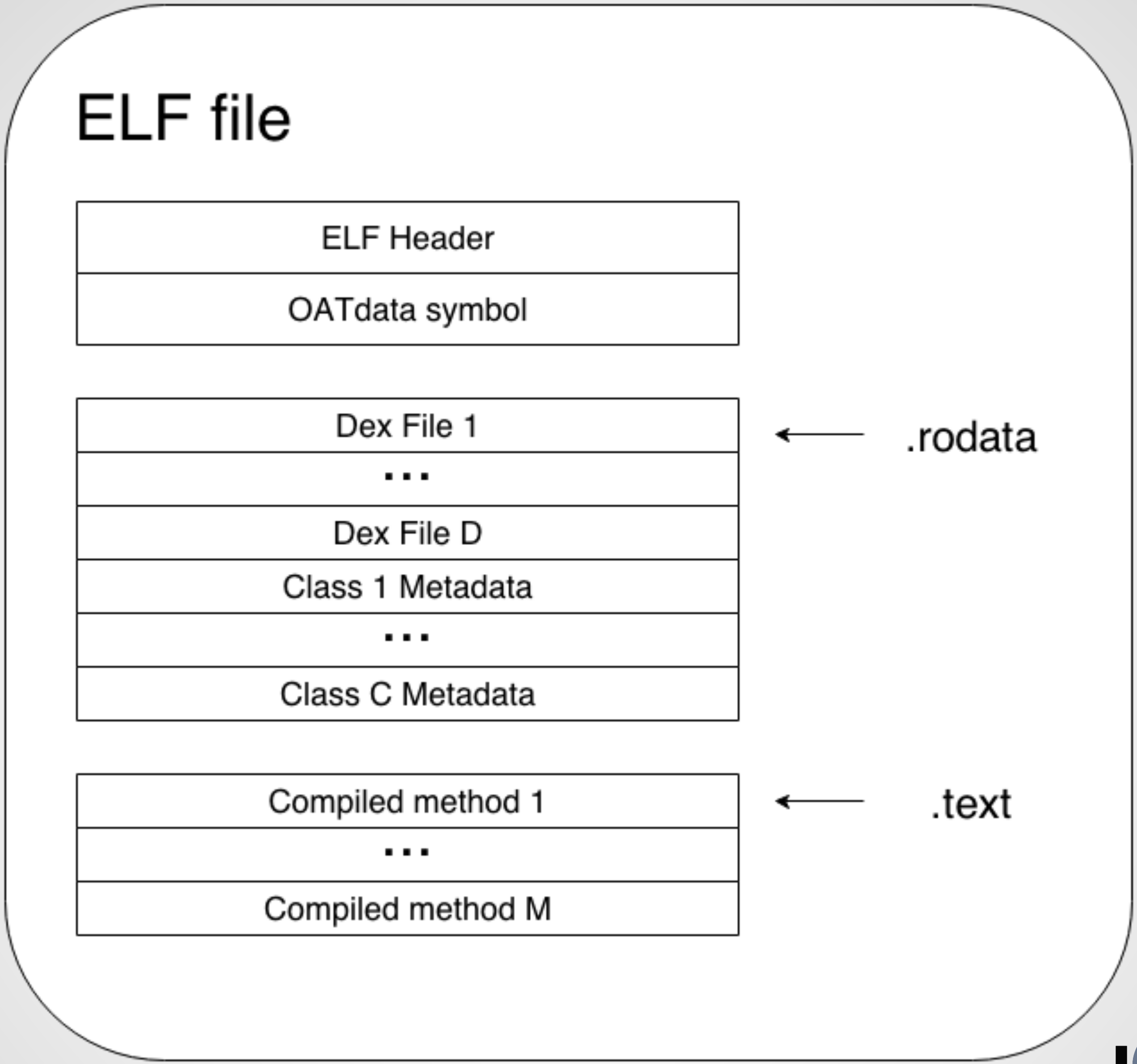
Dalvik: application installation

- Verification:
  - bytecode check (illegal instructions, valid indices,...)
  - checksum on files
- Optimization:
  - method inlining
  - byte swapping and padding
  - static linking

LSE Security System
Laboratory of Epita

ART: OAT file

- Generated during installation (dex2oat)
- ELF format
- Classes metadata


ELF file
ELF Header
OATdata symbol
Dex File 1
...
Dex File D
Class 1 Metadata
...
Class C Metadata
Compiled method 1
...
Compiled method M
.rodata
.text

Memory management

LSE Security System Laboratory of Epita

Zygote

- Daemon started at boot time
- Loads and initializes core libraries
- Forks to create new Dalvik instance
- Startup time of new VM is reduced
- Memory layouts are shared across processes

Dalvik: memory management

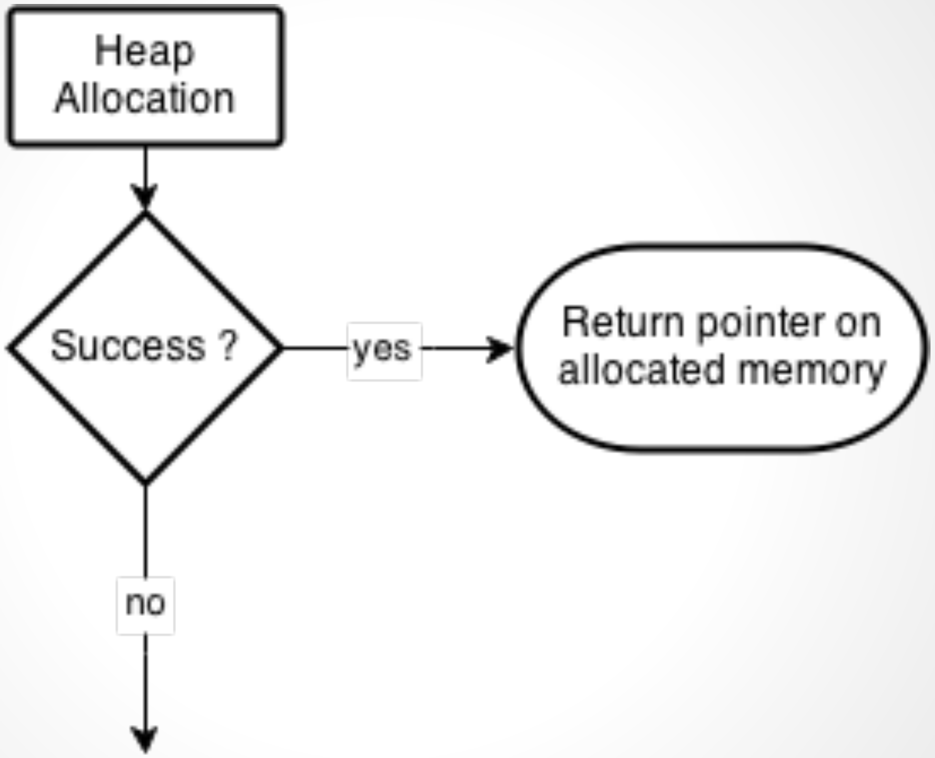
- Memory is garbage collected
- Automatic management avoids programming errors
- Objects are not freed as soon as they become unused

Dalvik: memory allocation

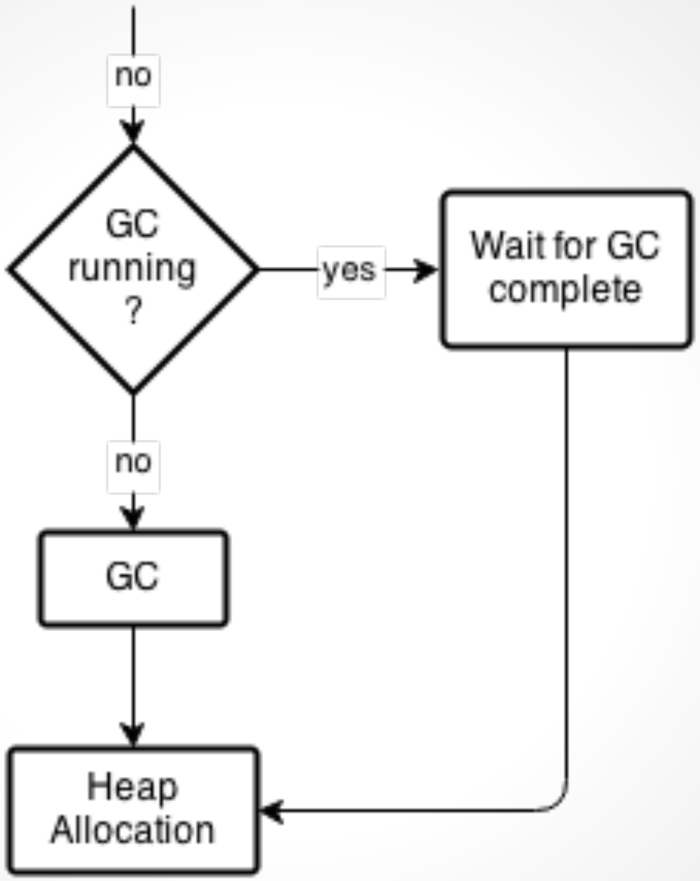
- Allocation profiling:
  - allocation count (succeeded or failed)
  - total allocated size (succeeded or failed)
- malloc function is more complex since memory is garbage collected

LSE Security System Laboratory of Epita

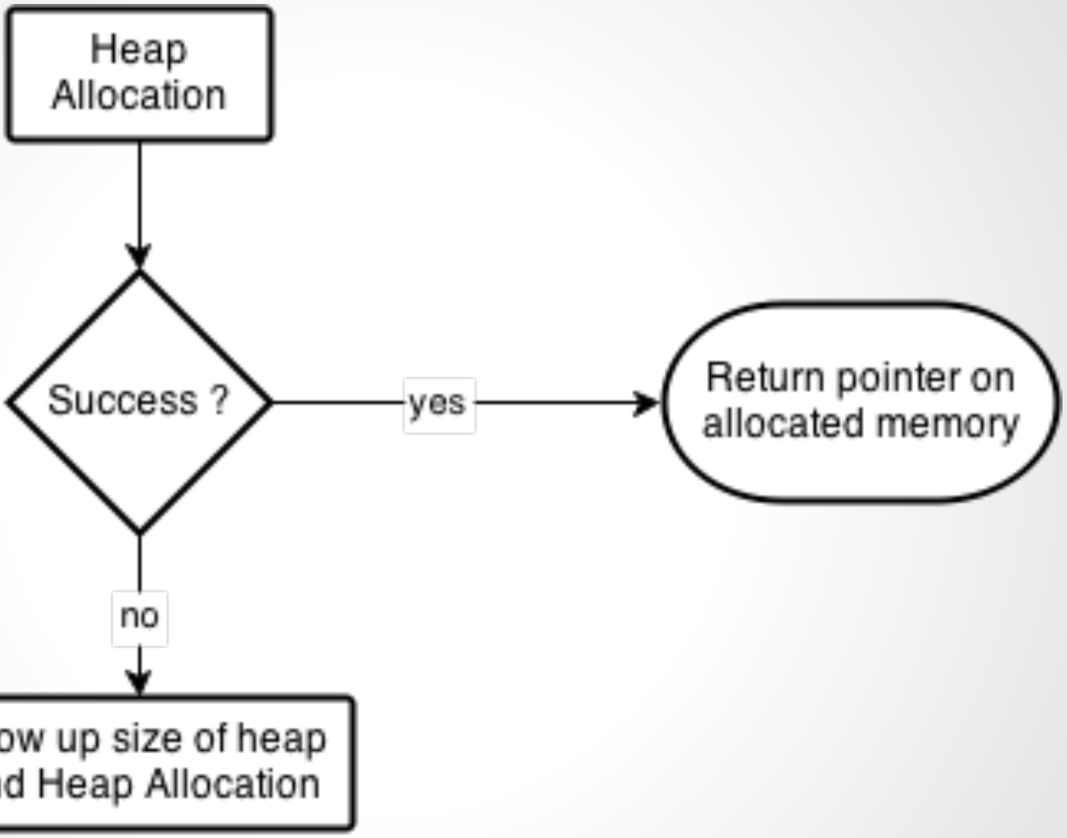
Dalvik: memory allocation



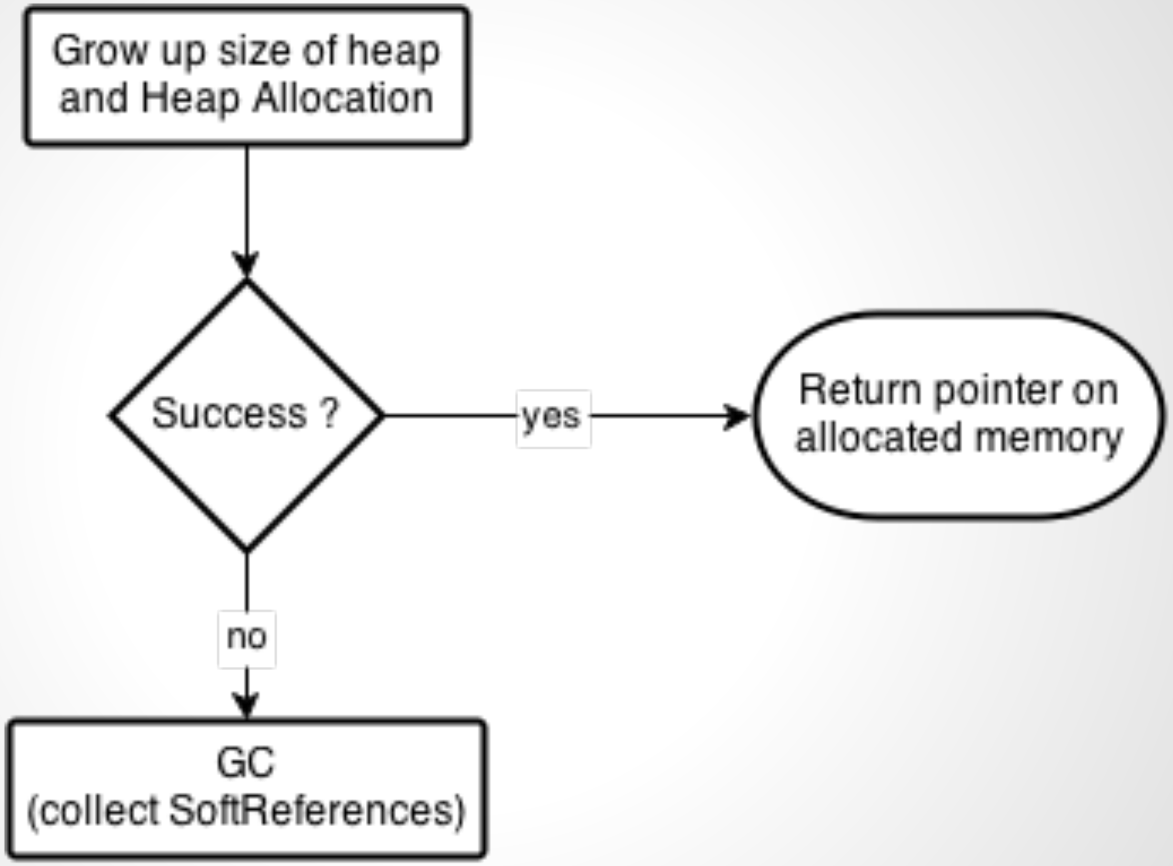
Dalvik: memory allocation



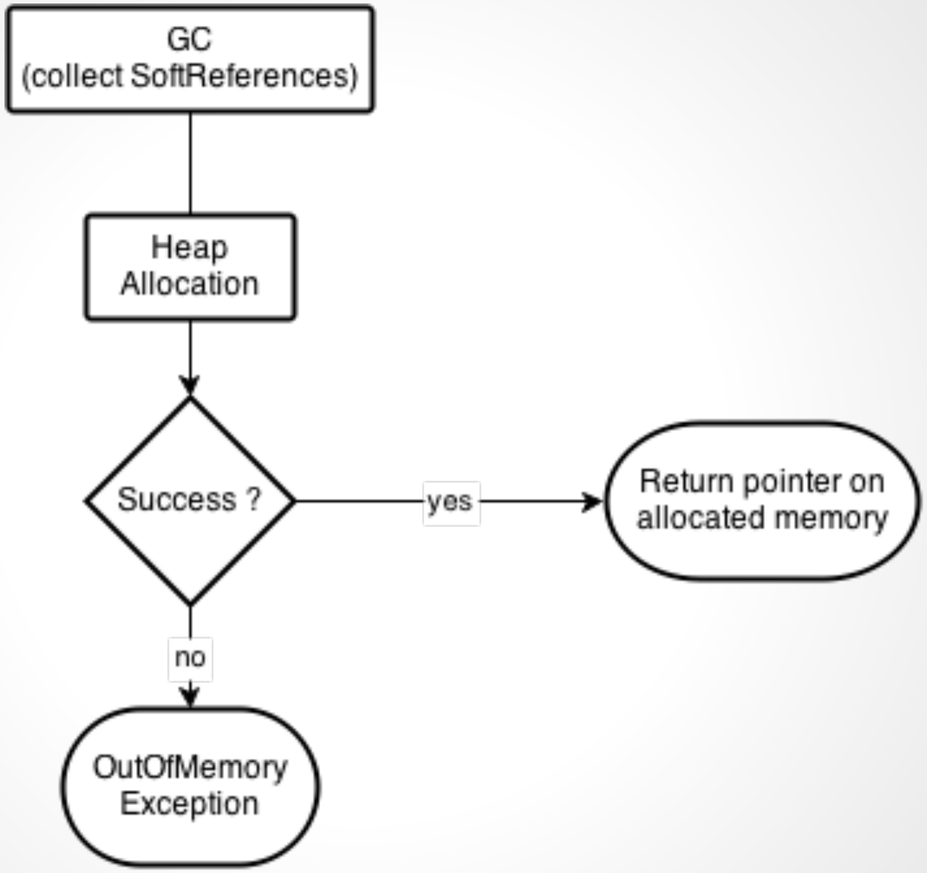
Dalvik: memory allocation


Heap Allocation
Success ?
yes
Return pointer on allocated memory
no
Grow up size of heap and Heap Allocation

Dalvik: memory allocation


Grow up size of heap and Heap Allocation
Success ?
yes
Return pointer on allocated memory
no
GC (collect SoftReferences)

Dalvik: memory allocation

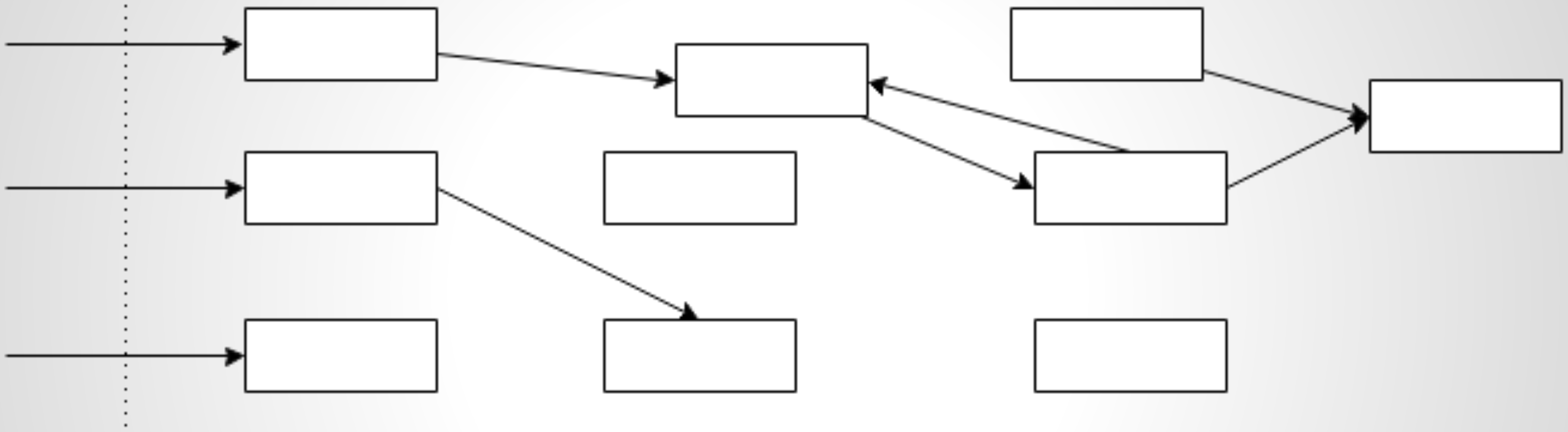


Dalvik: garbage collection

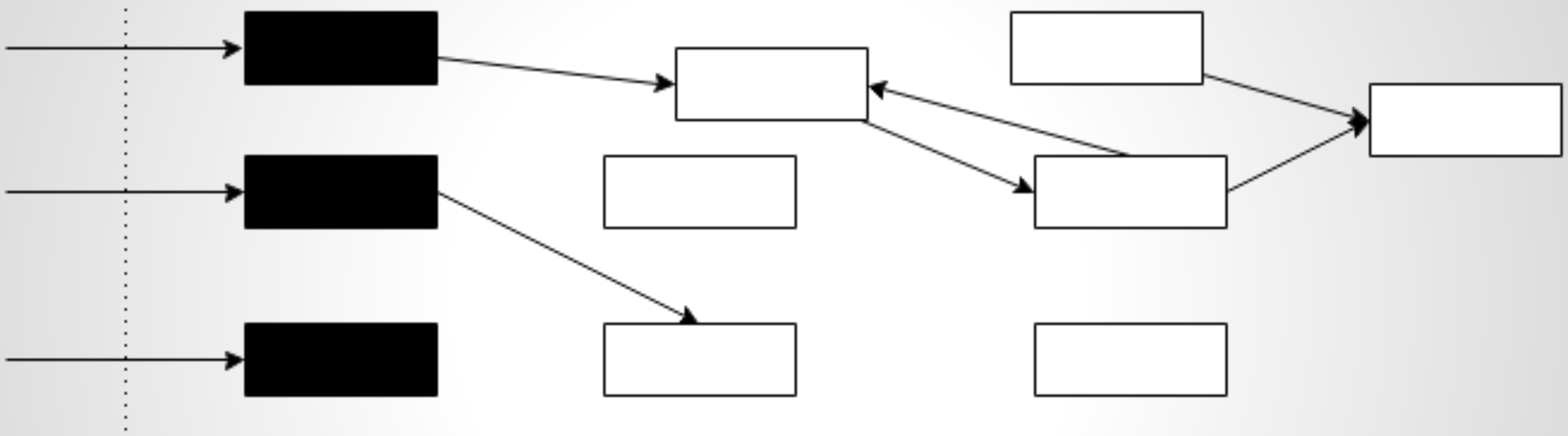
- Mark and Sweep algorithm
  - depends on the size of the heap
  - collects all garbage
- Stop the world before Android 2.3
- Mostly concurrent (2 pauses)

LSE Security System
Laboratory of Epita

Mark and Sweep

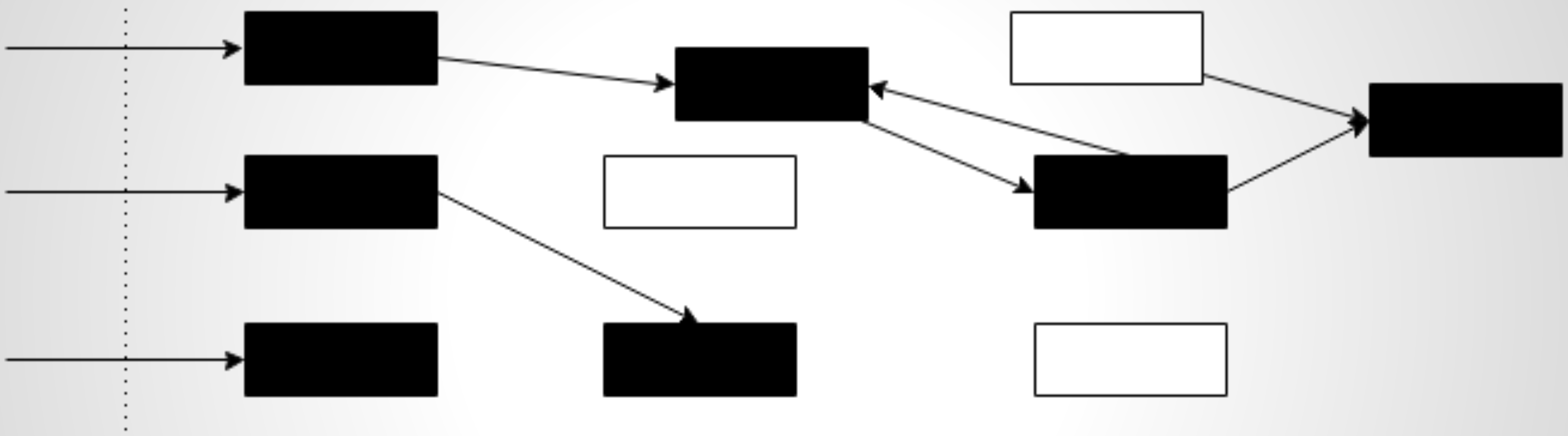


Mark and Sweep



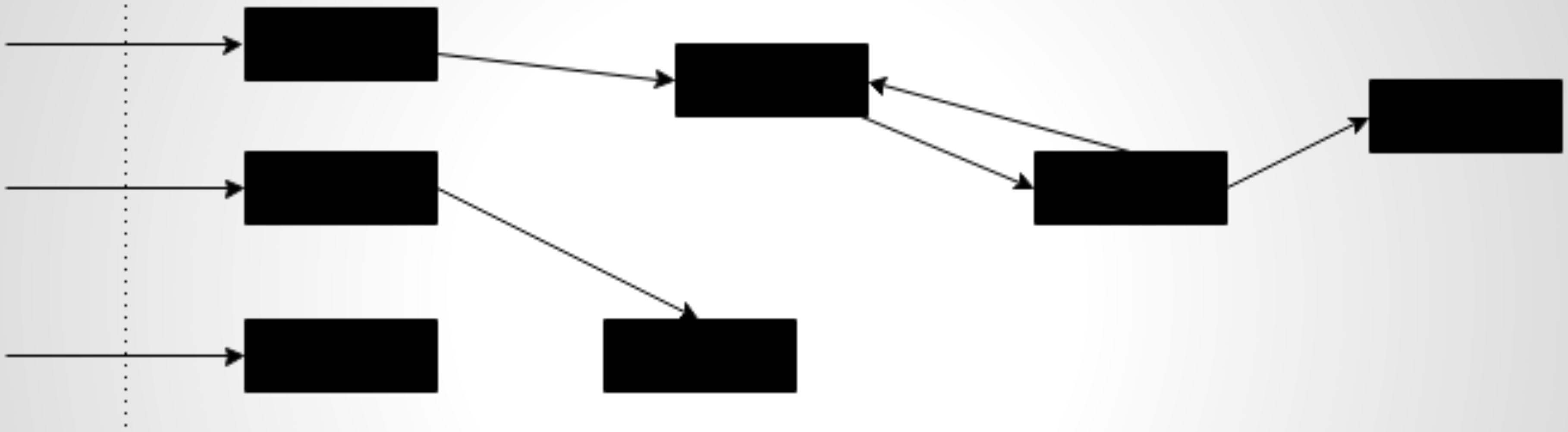
Step 1: Mark the roots

Mark and Sweep



Step 2: Recursively mark reachable objects

Mark and Sweep



Step 3: Sweep unmarked objects

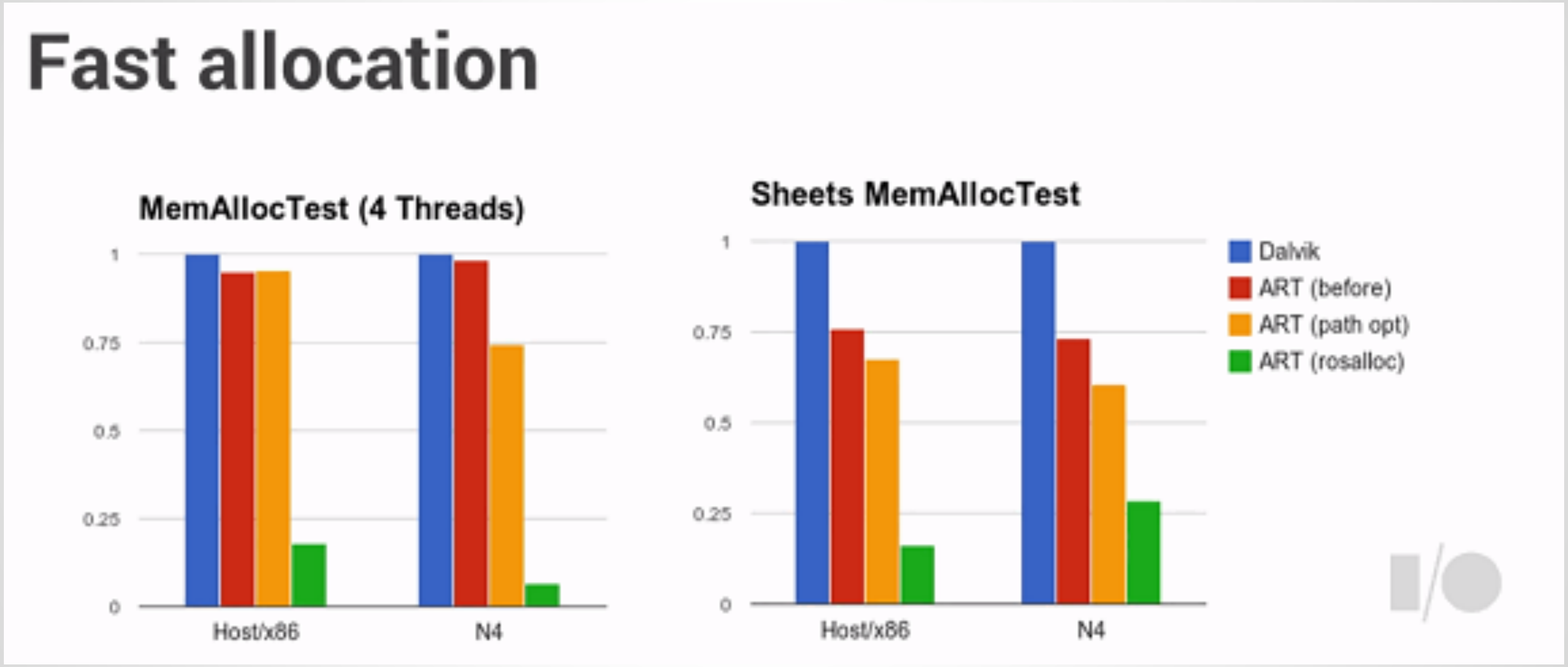
ART: garbage collectors

- GC faster
- Less fragmentation: moving collectors
- Concurrent, only one pause

ART: Rosalloc

- new allocator()
- Scales better for multithreaded applications

ART: Rosalloc



JIT and AOT compilation

LSE Security System Laboratory of Epita

JIT and AOT compilation

- Vocabulary:
  - Just In Time compilation
  - Ahead Of Time compilation
  - Hot code / Cold code
  - Granularity
- Purpose
  - Better performance

JIT and AOT compilation

- Granularity
  - Bigger:
    - Performance (optimizations)
    - Less context switches, synchronizations
    - Less re-usability
  - Smaller:
    - The opposite

JIT and AOT compilation

- When should we compile?
  - When you can accept latencies
  - Later compilation allows more optimizations
  - Coarse grained:
    - Installation
    - Launching
    - Execution (1 more thread to run)

JIT and AOT compilation

- Drawbacks:
  - CPU time (compilation)
  - Memory (results of compilation, tables)
  - Mostly: time

Dalvik: JIT compilation

- Operate on traces (~100 instructions)
- During program’s execution
- Why:
  - Hottest portions are compiled
  - Small translation cache
  - Performance boost is early perceived
  - Ignore jumps and method calls
  - Good trade-off between speed and memory

LSE Security System
Laboratory of Epita

Dalvik: JIT compilation

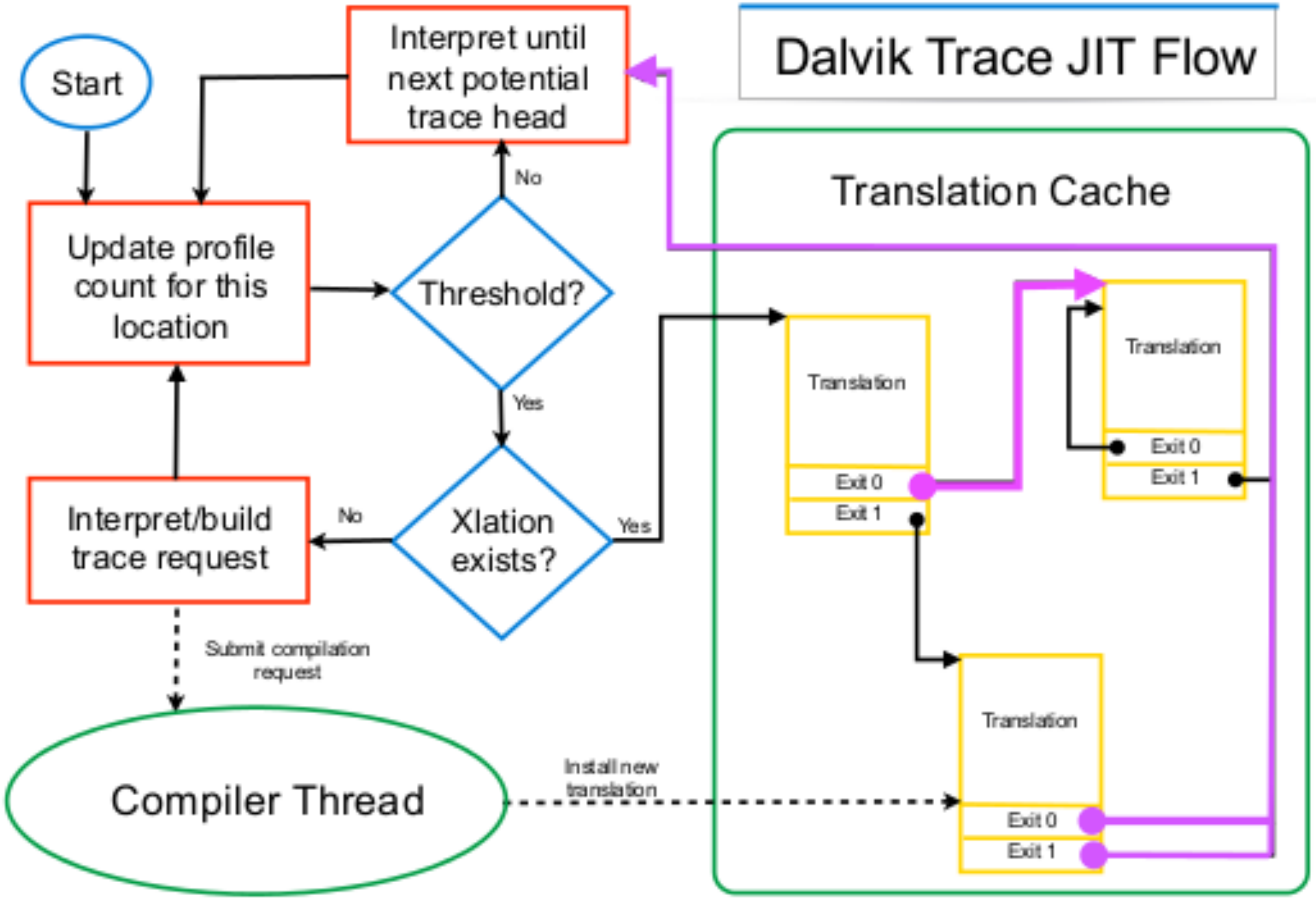
- One thread by Java application
  - Shared between every threads
  - Not shared between processes
  - Use private pages
- Re-done at every run of the application
- Several target architectures
  - ARM, MIPS, x86
  - Values and code generation that differs (performance, instructions set)

LSE Security System Laboratory of Epita

Dalvik: JIT compilation

- Stages:
  - Profile traces
  - Trace is considered hot:
    - Compiled version ?
      - Yes: use it
      - No: ask for a compilation
  - Repeat
- Compilation:
  - Task queue full => flush or block every other threads

Dalvik Trace JIT Flow



Dalvik: Tuning and debugging

- Debug options enables:
  - Statistics
  - Debug information
- Types of profiling:
  - Continuous polling
  - periodic polling (user defined)

Dalvik: Tuning and debugging

- Statistics:
  - Traces
  - Compiled traces
  - Calls to compiler
  - Number of traces profiled
  - Number of chained translated blocks
  - Time spent in compilation
  - Time during which the GC was blocked

Dalvik: Tuning and debugging

- Tunning:
  - Size of translation cache
  - Threshold to compile a trace
  - Maximal length of a trace
  - Layers and filters for hotness
- Debugging:
  - Comparison of the results of interpreted and compiled versions

ART: AOT compilation

- Compile at install-time
- Use llvm

ART: AOT compilation

- Stages (dex2oat):
  - Resolution
  - Verification
  - Initialisation
  - Compilation

Conclusion

- http://blog.lse.epita.fr
- #lse on rezosup
- blatinox@lse.epita.fr
- clippix@lse.epita.fr

QUESTIONS?