The Gate of the AOSP #3 : Externals & Extras

QEMU and Valgrind : Emulator & Memory Technology

2012. 10. 26.

www.kandroid.org 운영자 : 양정수 (yangjeongsoo at gmail.com), 닉네임: 들풀

QEMU and Valgrind : Emulator & Memory Technology

1. **Background History and Knowledge**
   - Android Emulator vs. IPhone Simulator
   - Dynamic Binary Translation and Instrumentation
   - Memory Issues : Fragmentation and Leakage

2. **QEMU : Emulation vs. Performance**
   - What is QEMU?
   - Goldfish Linux Kernel and Supported ABI(armeabi,armeabi-v7a,x86,mips)
   - QEMU Build System : Android Build System, Cygwin, MinGW
   - Emulation + Simulation(?) : OpenGL GPU Emulation
   - Emulation + H/W Virtualization : x86 and Google TV Emulator
   - Some Advanced Topics
     - LLVM Backend for QEMU
     - Can we dream about real s/w phone or cloud phone ?

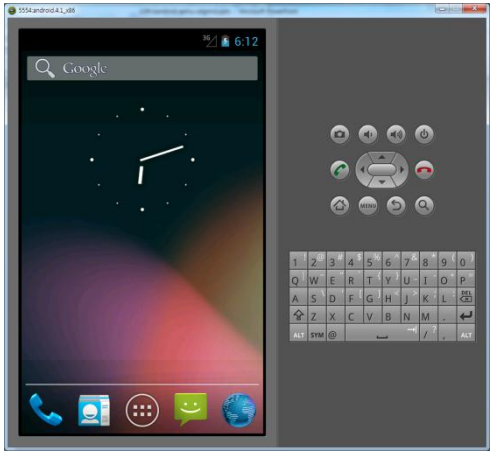
3. **Valgrind : Memory Leak vs. Memory Management**
   - What is Valgrind?
   - Valgrind Build and Installation for Real Target
   - Valgrind Issues : No Swap and Low Memory Killer
   - Memory Analysis Tools & Memory Management Methods (Java:MAT, JNI:Golbal/Local Ref, C++:sp,wp, SysV Shmem)
   - Some Advanced Topics
     - Valgrind and LLVM's AddressSanitizer
     - What Every Android Programmer Should Know About Memory?

Emulation versus simulation

The word "emulator" was coined in 1963 at IBM[15] during development of the NPL (IBM 360) product line, using a "new combination of software, microcode, and hardware".[16] They discovered that using microcode hardware instead of software simulation, to execute programs written for earlier IBM computers, dramatically increased simulation speed. Earlier in 1957, IBM provided the IBM 709 computer with an interpreter program (software) to execute legacy programs written for the IBM 704 to run on the IBM 709 and later on the IBM 7090[17] In 1963, when microcode was first used to speed up this simulation process, IBM engineers coined the term "emulator" to describe the concept.

It has recently become common to use the word "emulate" in the context of software. However, before 1980, "emulation" referred only to emulation with a hardware or microcode assist, while "simulation" referred to pure software emulation.[18] For example, a computer specially built for running programs designed for another architecture is an emulator. In contrast, a simulator could be a program which runs on a PC, so that old Atari games can be simulated on it. Purists continue to insist on this distinction, but currently the term "emulation" often means the complete imitation of a machine executing binary code while "simulation" often refers to Computer simulation, where a computer program is used to simulate an abstract model. Computer simulation is used in virtually every scientific and engineering domain and Computer Science is no exception, with several projects simulating abstract models of computer systems, such as Network simulation.

source : http://en.wikipedia.org/wiki/Emulator#Emulation_versus_simulation



Dynamic Binary Translation and Instrumentation

Why dynamic, not static ?

- Static method is very difficult (ex, indirect branches)

Dynamic Binary Translation :

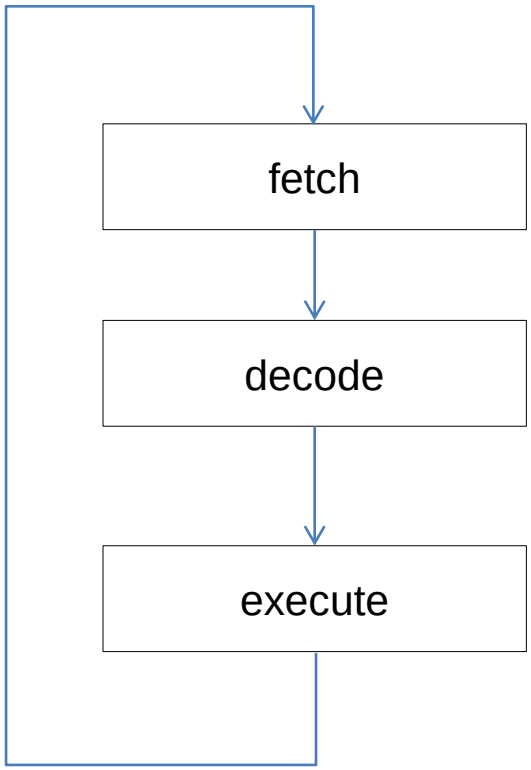
- ex) Translating arm executable into x86 executable

Dynamic Binary Instrumentation :

- Injecting arbitrary code into binary at runtime.

Dynamic Binary Translation	Dynamic Binary Instrumentation
Simple Translation • fetch-decode-execute loop Advanced Translation • Caching • Recompilation (ex, JIT Compiler)	Decompilation & Recompilation • Valgrind Using Trap Instruction • Dtrace • SystemTap • Frysk • GDB

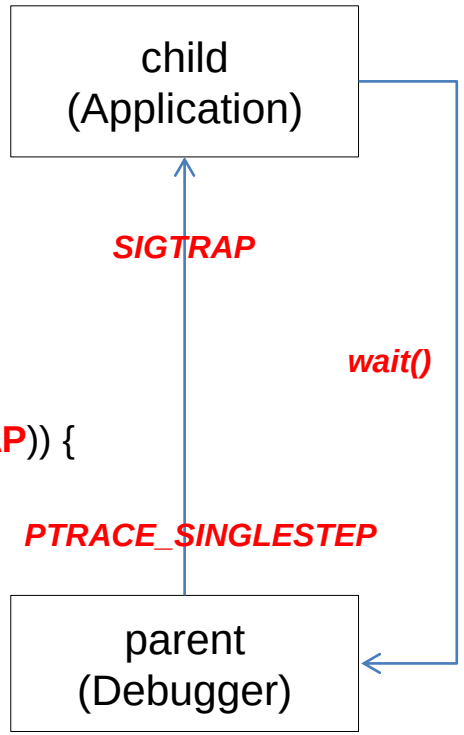
Dynamic Binary Translation and Instrumentation : fetch-decode-execute loop




```
0xc0083420:  e0861005


add  r1, r6, r5

switch(opcode) {

case OP_ADD:
        add(r1, r6, r5);
        break;
}

void add(int r1, int r6, int r5)
{
        context->reg[r1] = context->reg[r6]
                                + context->reg[r5];
}
```

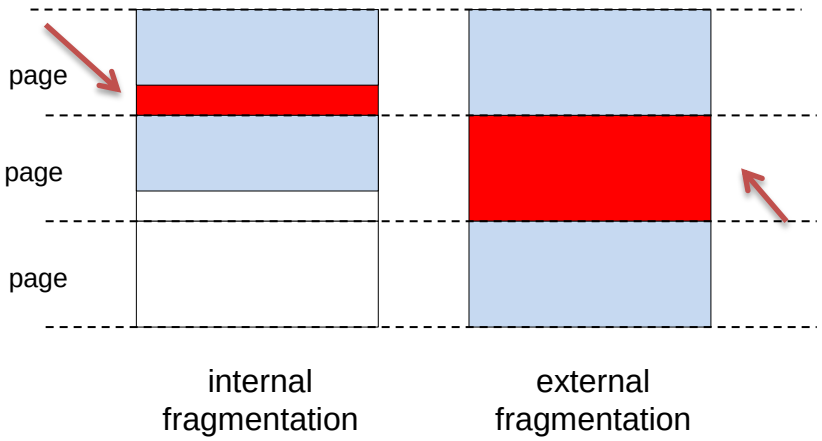
Dynamic Binary Translation and Instrumentation : trap instruction

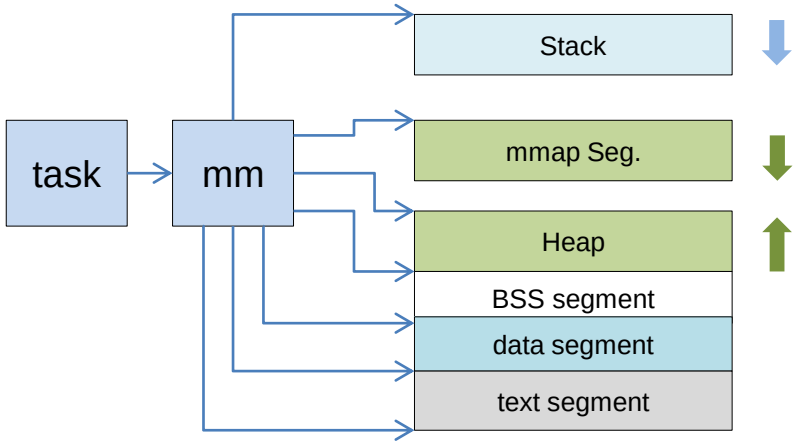
```
int main(int argc, char **argv)
{
    pid_t child;

    child = fork();
    if(child == 0) {
        ptrace(PTRACE_TRACEME, 0, NULL, NULL);
        execl(argv[1], argv[1], NULL);
    }
    else {
        int status;
        ptrace(PTRACE_SINGLESTEP, child, NULL, NULL);
        while(1) {
            wait(&status);
            if (WIFSTOPPED(status) && (WSTOPSIG(status) == SIGTRAP)) {
                ptrace(PTRACE_SINGLESTEP, child, NULL, NULL);
            }
            else {
                break;
            }
        }
    }
    return 0;
}
```



Memory Issues : Fragmentation and Leakage

Memory Fragmentation	Memory Leakage
Internal Fragmentation • Linux kernel : slab allocator • Libc : dlmalloc, ptmalloc External Fragmentation • Linux kernel : buddy system	Memory Allocation • Static allocation • Stack allocation • Heap(Dynamic) allocation Garbage Collection Explicit collection Automatic collection





QEMU and Valgrind : Emulator & Memory Technology

1. **Background History and Knowledge**
   - Android Emulator vs. IPhone Simulator
   - Dynamic Binary Translation and Instrumentation
   - Memory Issues : Fragmentation and Leakage

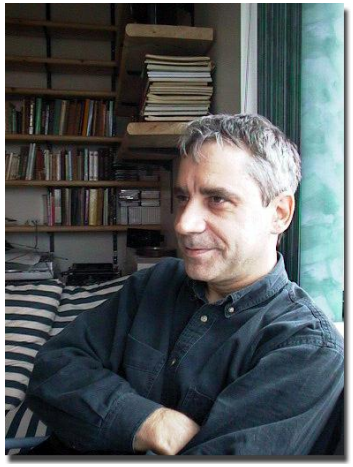
2. **QEMU : Emulation vs. Performance**
   - What is QEMU?
   - Goldfish Linux Kernel and Supported ABI(armeabi,armeabi-v7a,x86,mips)
   - QEMU Build System : Android Build System, Cygwin, MinGW
   - Emulation + Simulation(?) : OpenGL GPU Emulation
   - Emulation + H/W Virtualization : x86 and Google TV Emulator
   - Advanced Topics
     - LLVM Backend for QEMU
     - Can we dream about real s/w phone or cloud phone ?

3. **Valgrind : Memory Leak vs. Memory Management**
   - What is Valgrind?
   - Valgrind Build and Installation for Real Target
   - Valgrind Issues : No Swap and Low Memory Killer
   - Memory Analysis Tools & Memory Management Methods (Java:MAT, JNI:Golbal/Local Ref, C++:sp,wp, SysV Shmem)
   - Advanced Topics
     - Valgrind and LLVM's AddressSanitizer
     - What Every Android Programmer Should Know About Memory?

What is QEMU ?

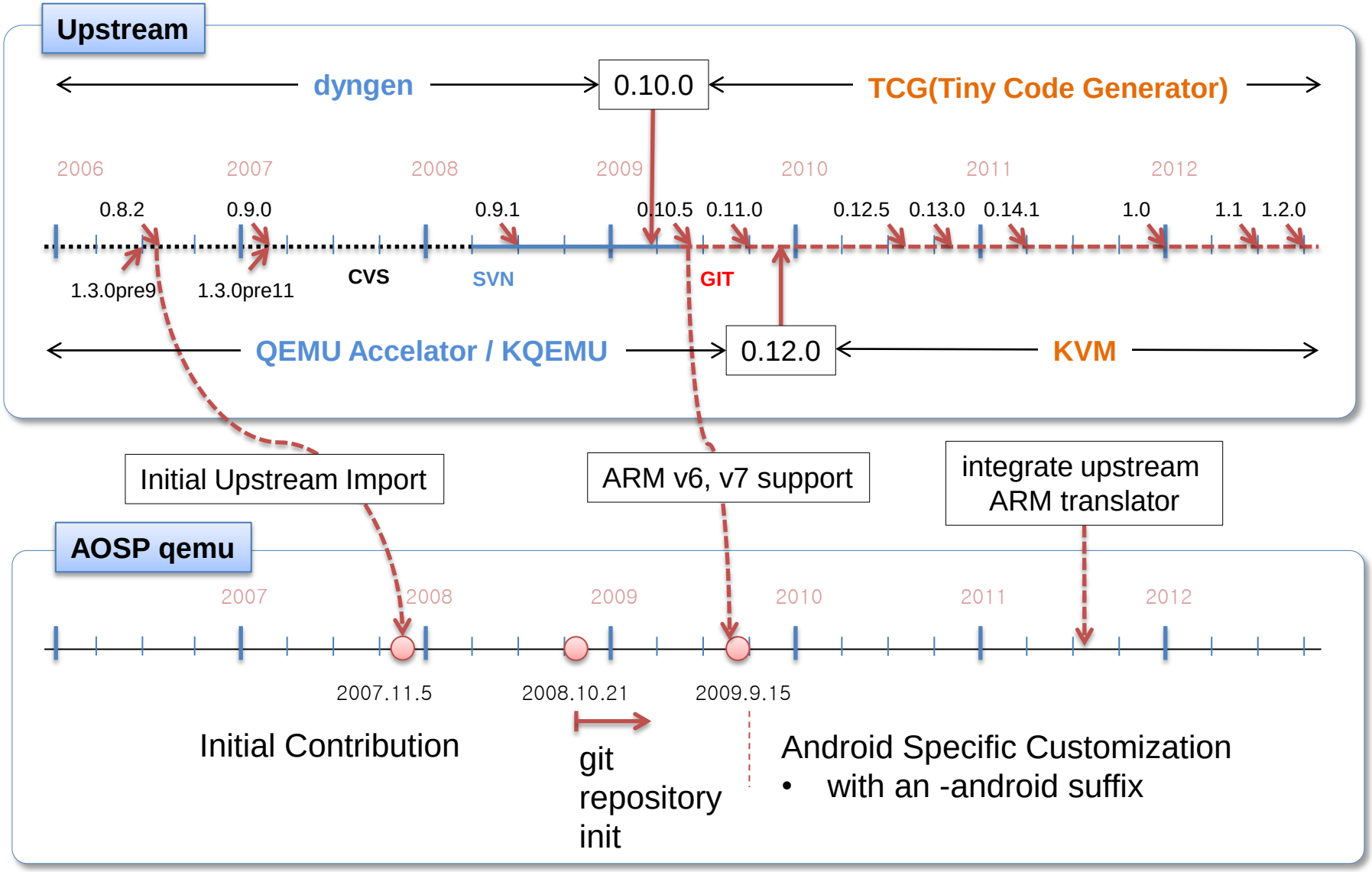
- Open source library for hardware **emulation** and **virtualization**
- Fast CPU and device emulator based on **dynamic binary translation**
- Execution of SW binaries of a guest instruction set on host PC
  - Development of drivers and application SW on host PC
  - Debugging of guest binaries
  - Development of SW tool chain for guest SW
- Different operation modes
  - **full system emulation**
  - user mode emulation
  - **KQEMU/KVM**

Source : First QEMU Users' Forum - 1st International QEMU Users' Forum, 2011

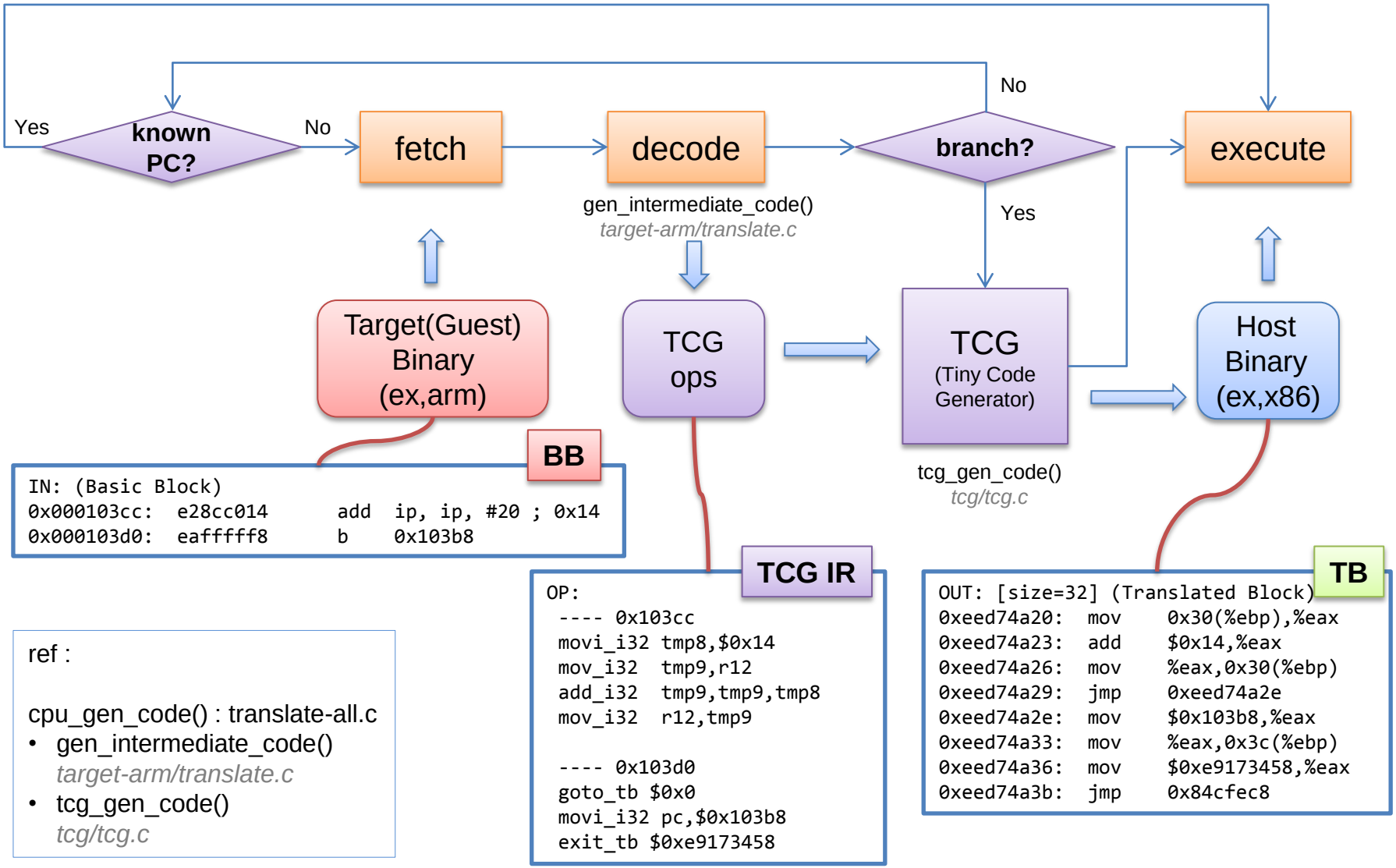


Fabrice Bellard
born :1972
Grenoble, France

What is QEMU ?


Upstream
dyngen
0.10.0
TCG(Tiny Code Generator)
2006
2007
2008
2009
2010
2011
2012
0.8.2
0.9.0
0.9.1
0.10.5
0.11.0
0.12.5
0.13.0
0.14.1
1.0
1.1
1.2.0
1.3.0pre9
1.3.0pre11
CVS
SVN
GIT
QEMU Accelator / KQEMU
0.12.0
KVM
Initial Upstream Import
ARM v6, v7 support
integrate upstream
ARM translator
AOSP qemu
2007
2008
2009
2010
2011
2012
2007.11.5
2008.10.21
2009.9.15
Initial Contribution
git
repository
init
Android Specific Customization
• with an -android suffix

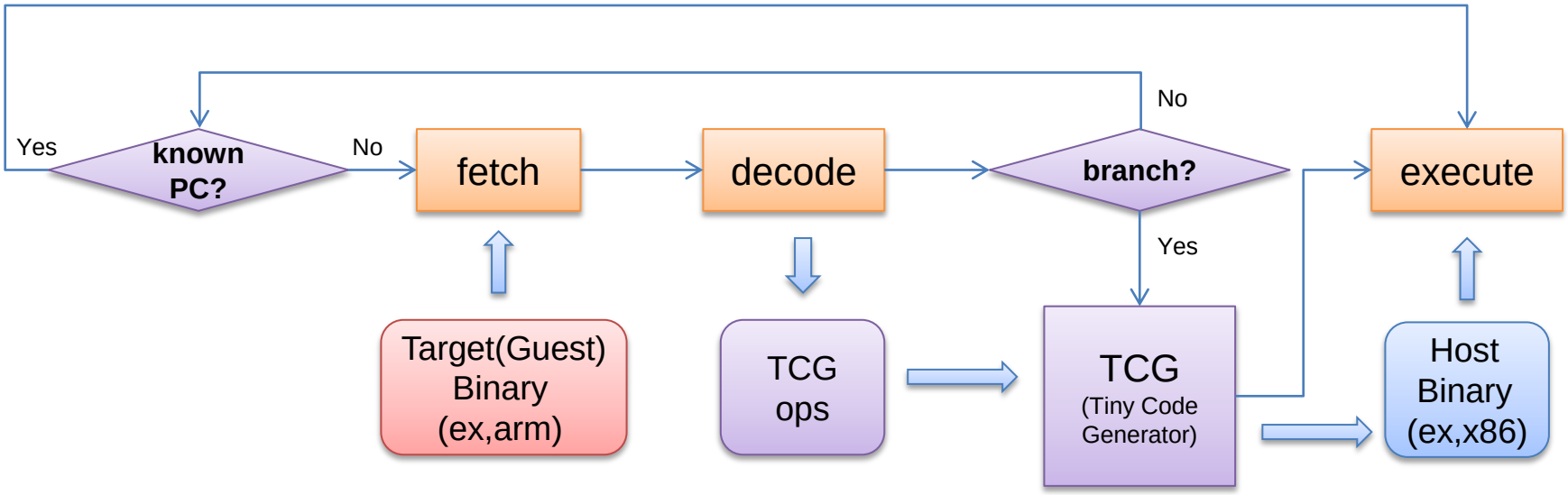
What is QEMU ?



ref :

cpu_gen_code() : translate-all.c
- gen_intermediate_code()
  target-arm/translate.c
- tcg_gen_code()
  tcg/tcg.c

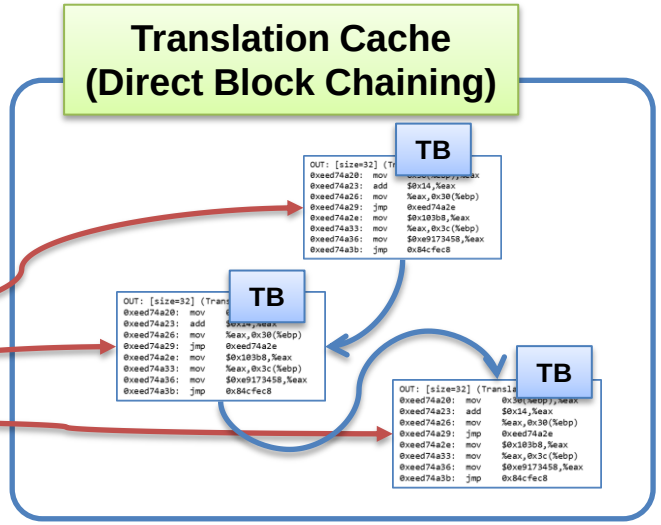
What is QEMU ?



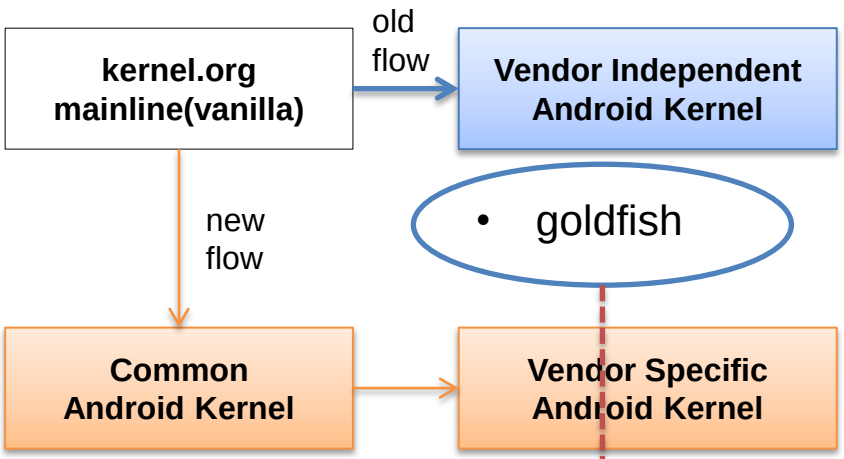

```
for (;;) {
        tb = tb_find_fast();
}
```

Hash Table (tb_jmp_cache)

0xa7a1
0xd37c
0x4389
…



Goldfish Linux Kernel and Supported ABI(armeabi,armeabi-v7a,x86,~~mips~~)



Android Kernel Enhancements

- Binder
- Ashmem
- Pmem
- Logger
- PowerManager
- Low Memory Killer
- RAM Console
- Paranoid network
- Timed GPIO
- …

Architecture Dependent Kernel

- goldfish
- msm
- omap
- samsung
- tegra
- exynos

For the first android SDK,

1. Initial qemu Upstream Import
2. Initial contribution

What is the initial contribution?

1. Android-specific modification to qemu
2. Create virtual soc : goldfish
   - Support CPU : ARM926EJS
   - Support Device : fb,audio,tty,mmc,nand, etc

What is the add-on features?

1. Support additional ABI : armv7, x86
2. OpenGLES h/w acceleration
3. Emulator with h/w assist for x86

Android Build System

- AOSP Build System : GNU Make + Toolchain + Custom Build Tools
- NDK Build System : **Cygwin** + Toolchain + Custom Build Tools

How can we build android emulator?

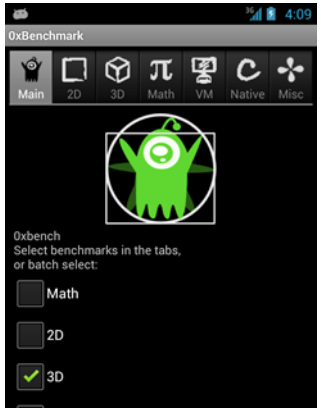
$ cd ~/asop/external/qemu; ./android-configure.sh

- for Linux
  $ ./android-rebuild.sh
- for Windows
  $ ./android-rebuild.sh --static --**mingw**

Cygwin vs. MinGW

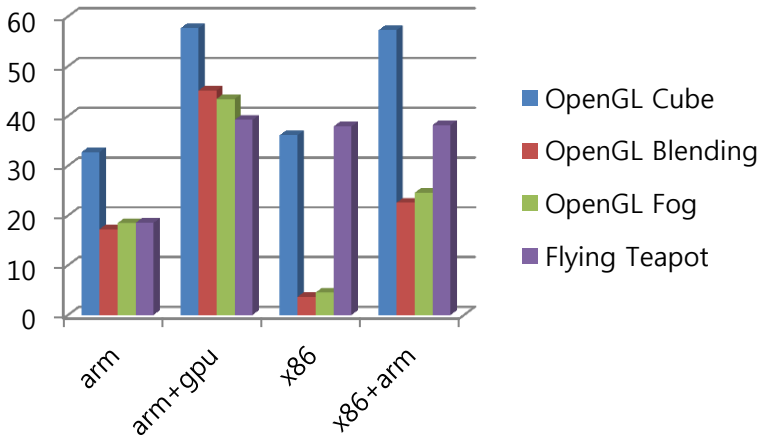
- The Cygwin tools are ports of the popular GNU development tools for Microsoft Windows. They run thanks to the Cygwin library which provides the POSIX system calls and environment these programs expect.
- MinGW ("Minimalistic GNU for Windows") is a collection of freely available and freely distributable Windows specific header files and import libraries combined with GNU toolsets that allow one to produce native Windows programs that do not rely on any 3rd-party C runtime DLLs.

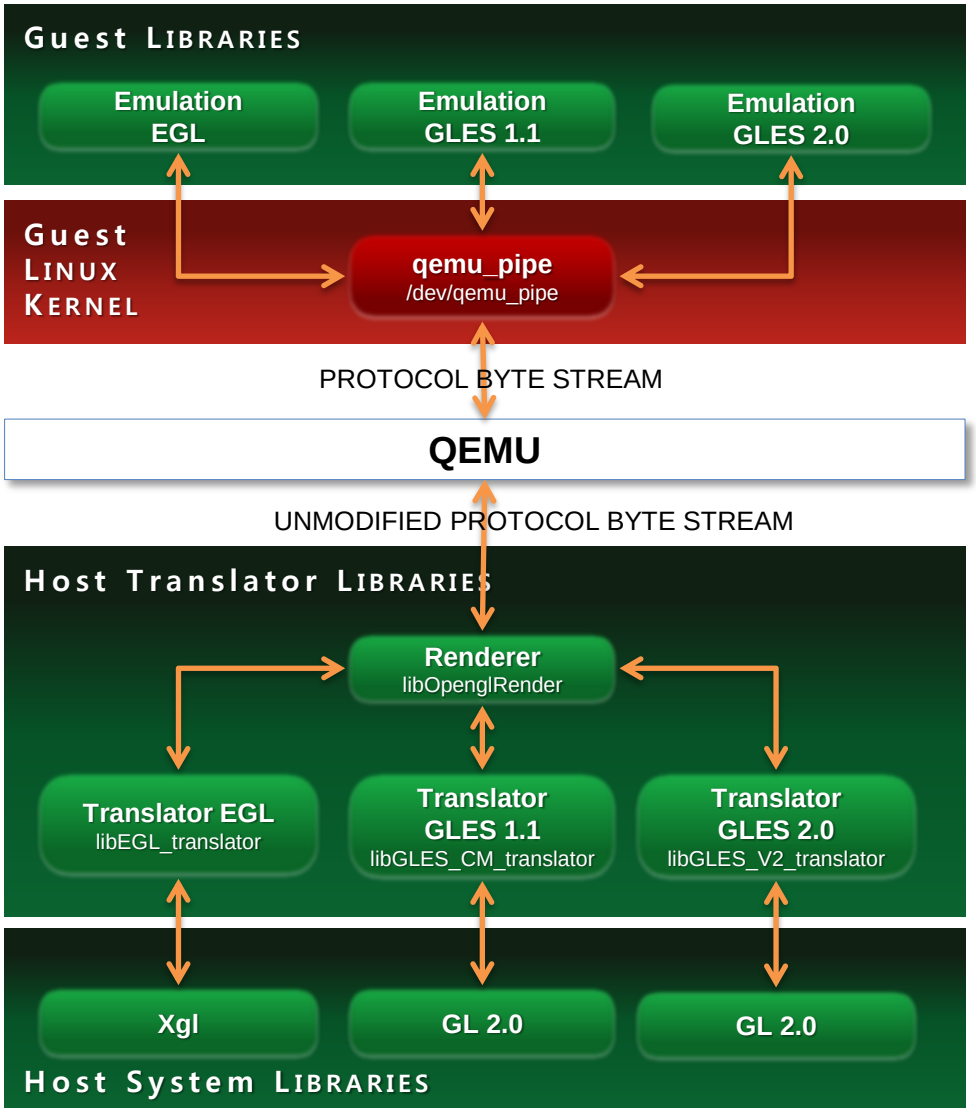
Emulation + Simulation(?) : OpenGL GPU Emulation



	arm	arm+gpu	x86	x86+arm
OpenGL Cube	32.726646	57.63191	36.11239	57.23642
OpenGL Blending	17.244913	45.1264	3.632293	22.54601
OpenGL Fog	18.457346	43.40562	4.50357	24.5354
Flying Teapot	18.511148	39.20031	37.88435	38.16494

http://code.google.com/p/0xbench/





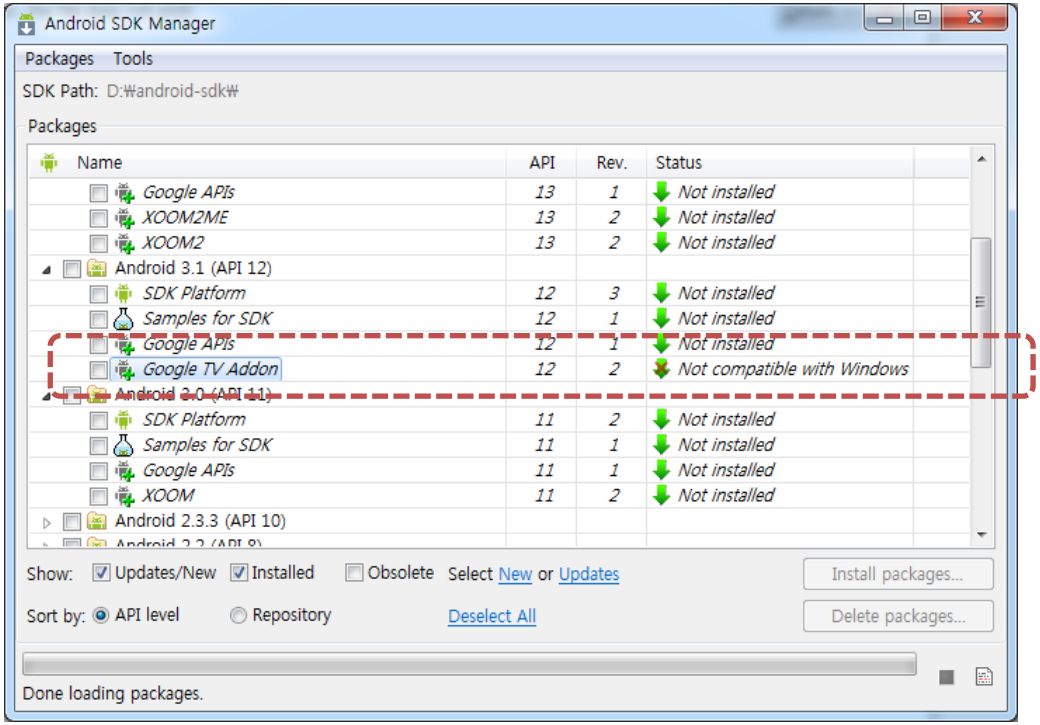
Google TV Emulator install process

1. Check System Requirement : Linux OS with KVM

```
$ cat /proc/cpuinfo
$ egrep '^flags.*(vmx|svm)' /proc/cpuinfo
$ sudo modprobe kvm-intel (or sudo modprobe kvm-amd)
```

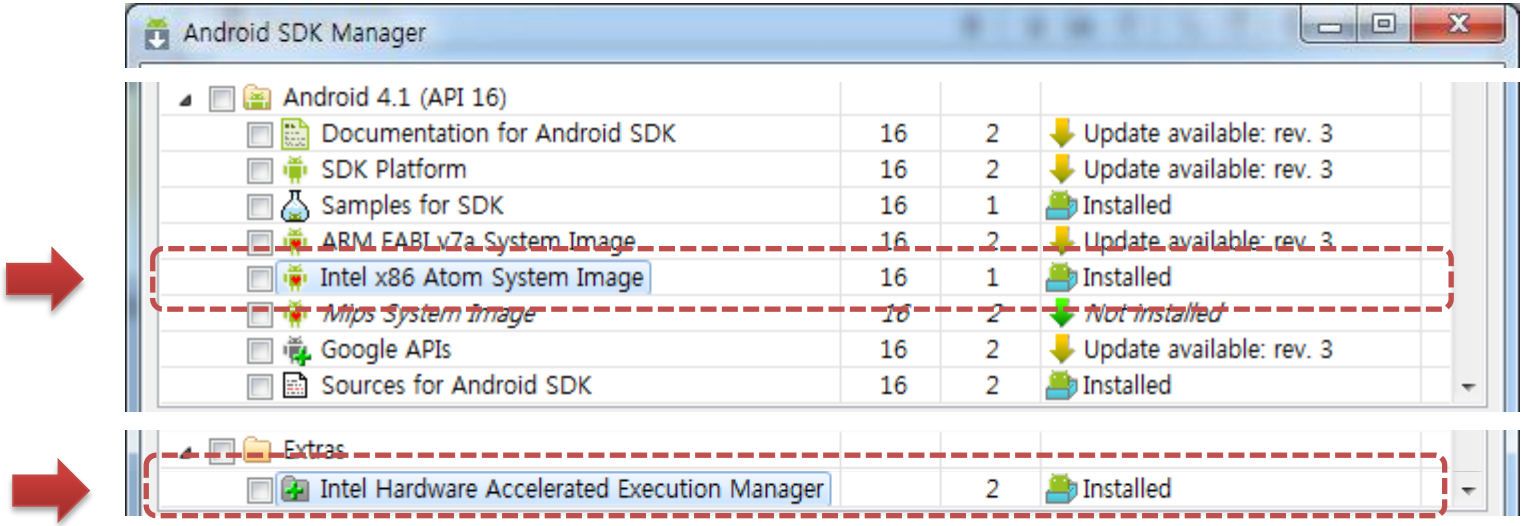
2. Linux KVM Install Process

```
$ make
$ make modules
$ make modules_install

$ sudo modprobe kvm-intel
$ ls -l /dev/kvm
$ sudo chmod a+rw /dev/kvm
$ sudo addgroup kvmusers
$ sudo addgroup you kvmusers
$ sudo chgrp kvmusers /dev/kvm
$ sudo chmod g+rw /dev/kvm
$ ls -l /dev/kvm
```



Emulation + H/W Virtualization : x86 and Google TV Emulator

Intel HAX based Android Emulator Acceleration

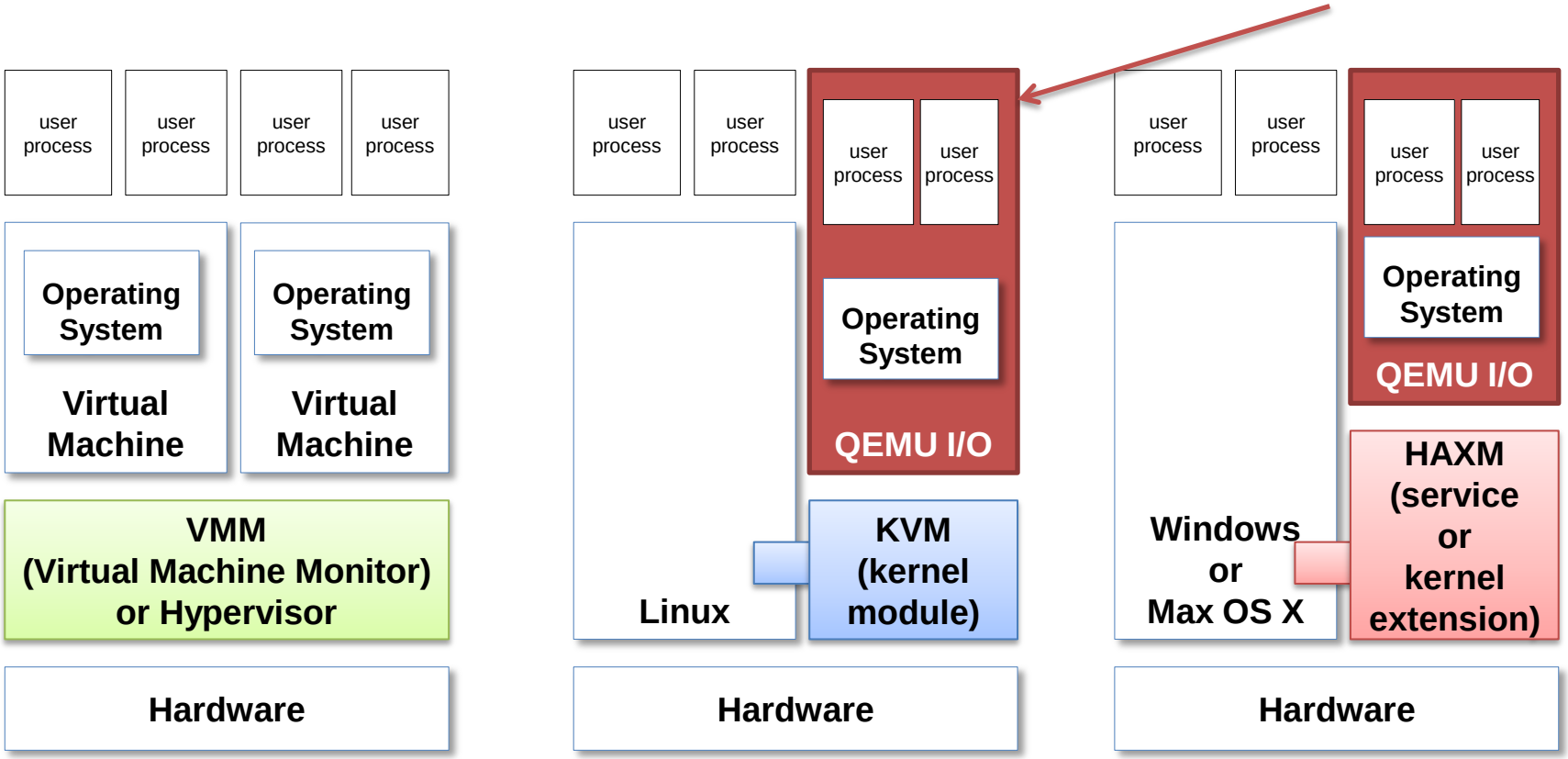



```
D:₩android-sdk₩tools>sc query intelhaxm

SERVICE_NAME: intelhaxm
        종류               : 1  KERNEL_DRIVER
        상태               : 4  RUNNING
                                (STOPPABLE, NOT_PAUSABLE, IGNORES_SHUTDOWN)
        WIN32_EXIT_CODE    : 0  (0x0)
        SERVICE_EXIT_CODE  : 0  (0x0)
        검사점             : 0x0
        WAIT_HINT          : 0x0

D:₩android-sdk₩tools>
```

Emulation + H/W Virtualization : x86 and Google TV Emulator

What is KVM, Intel HAXM?

- KVM is a Hypervisor/VMM based on Linux
- HAXM is a Hypervisor/VMM based on Windows or Max OS X
- KVM adds a third mode to the standard linux **kernel** and **user** modes, the **guest** mode



LLVM Backend for QEMU

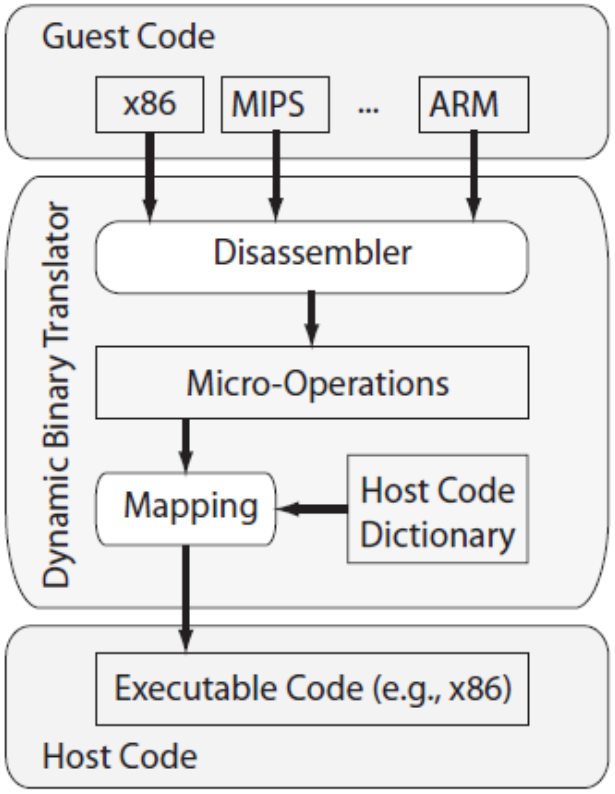


Figure 1: The QEMU dynamic binary translator

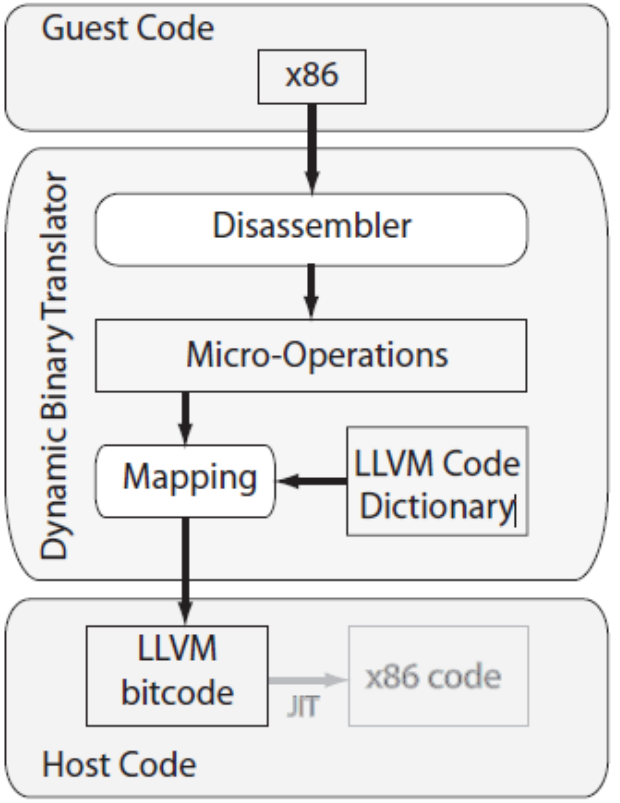
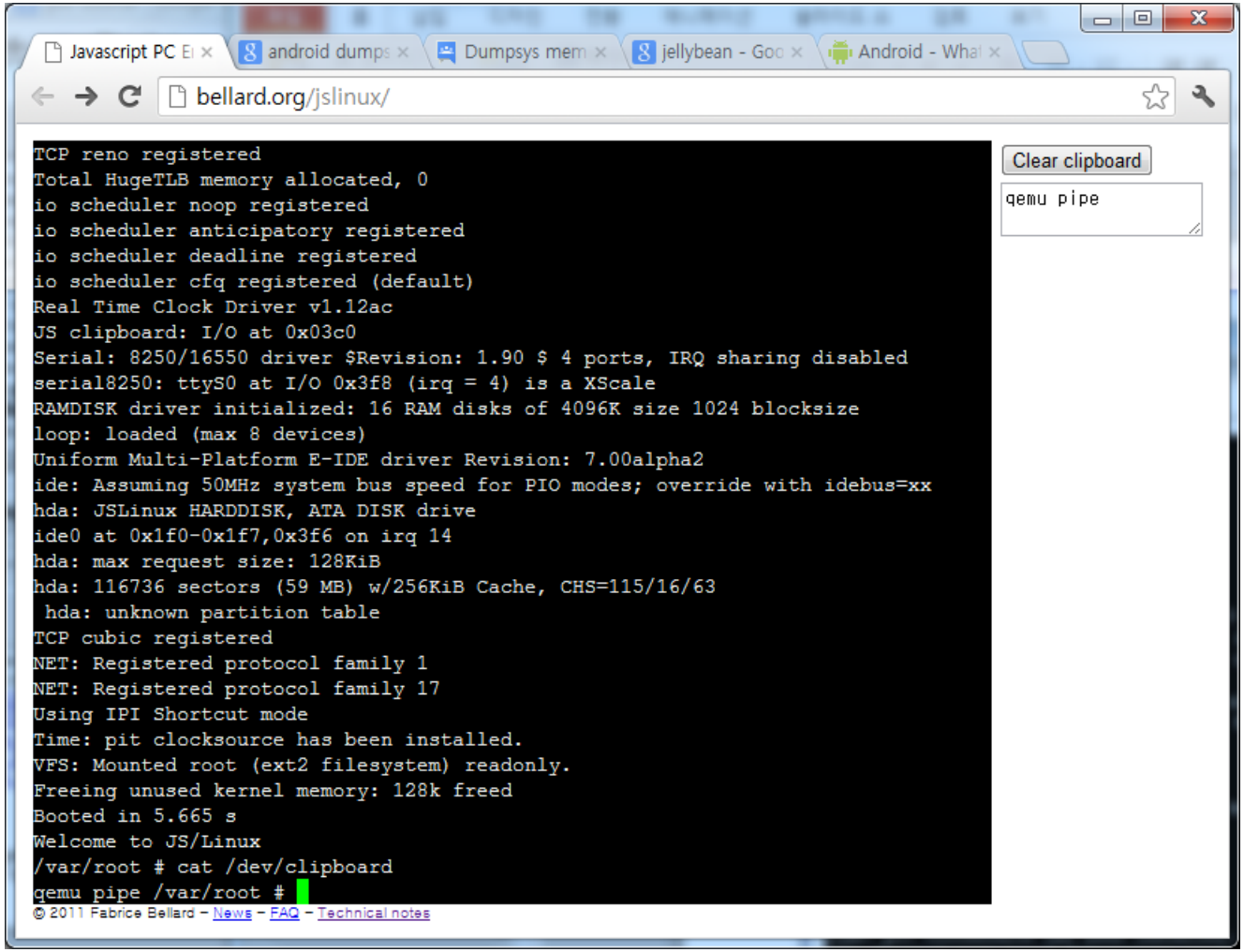


Figure 2: An LLVM backend for QEMU

source : http://infoscience.epfl.ch/record/149975/files/x86-llvm-translator-chipounov_2.pdf

Can we dream about real s/w phone or cloud phone?

source : http://bellard.org/jslinux/tech.html



QEMU and Valgrind : Emulator & Memory Technology

1. **Background History and Knowledge**
   - Android Emulator vs. IPhone Simulator
   - Dynamic Binary Translation and Instrumentation
   - Memory Issues : Fragmentation and Leakage

2. **QEMU : Emulation vs. Performance**
   - What is QEMU?
   - Goldfish Linux Kernel and Supported ABI(armeabi,armeabi-v7a,x86,mips)
   - QEMU Build System : Android Build System, Cygwin, MinGW
   - Emulation + Simulation(?) : OpenGL GPU Emulation
   - Emulation + H/W Virtualization : x86 and Google TV Emulator
   - Advanced Topics
     - LLVM Backend for QEMU
     - Can we dream about real s/w phone or cloud phone ?

3. **Valgrind : Memory Leak vs. Memory Management**
   - What is Valgrind?
   - Valgrind Build and Installation for Real Target
   - Valgrind Issues : No Swap and Low Memory Killer
   - Memory Analysis Tools & Memory Management Methods (Java:MAT, JNI:Golbal/Local Ref, C++:sp,wp, SysV Shmem)
   - Advanced Topics
     - Valgrind and LLVM's AddressSanitizer
     - What Every Android Programmer Should Know About Memory?

What is Valgrind?

pronounce : “**Val-grinned**”, not “Val-grined”

- Valgrind is a “program-execution monitoring framework”.
- Valgrind comes with many tools,
  the tool you will use most often is the memcheck tool.
- Memcheck will detect and report the following types of memory errors:
  - Use of uninitialized memory.
  - Reading/writing to memory after it has been freed.
  - Reading/writing off the end of malloc’d blocks.
  - Reading/writing inappropriate areas on the stack.
  - Overlapping src and dest pointers in memcpy() and related functions.
  - other stuff...

Source : http://ugweb.cs.ualberta.ca/~c201/F06/resources/valgrind.pdf

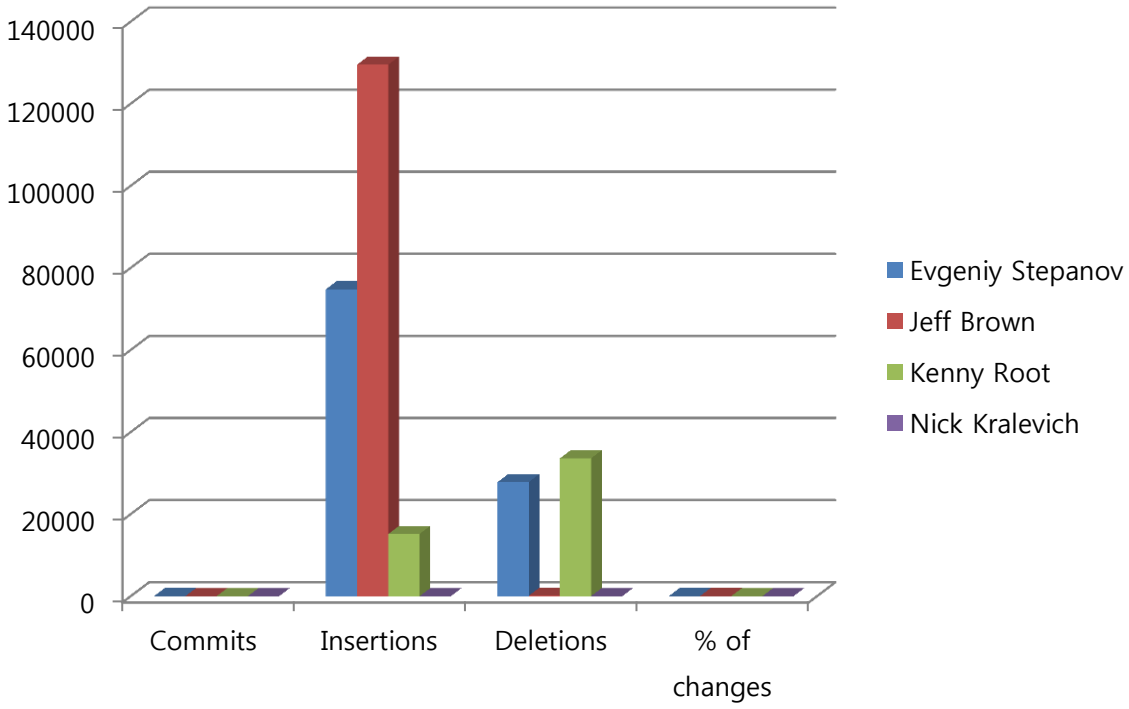
Julian Seward is a recycled compiler hacker. he founded the Valgrind project in 2000 and is the project lead and a full time developer. He's also the author of bzip2, a data compression program.

Valgrind, in Norse mythology, is the sacred gate to Valhalla through which only the chosen slain can pass.

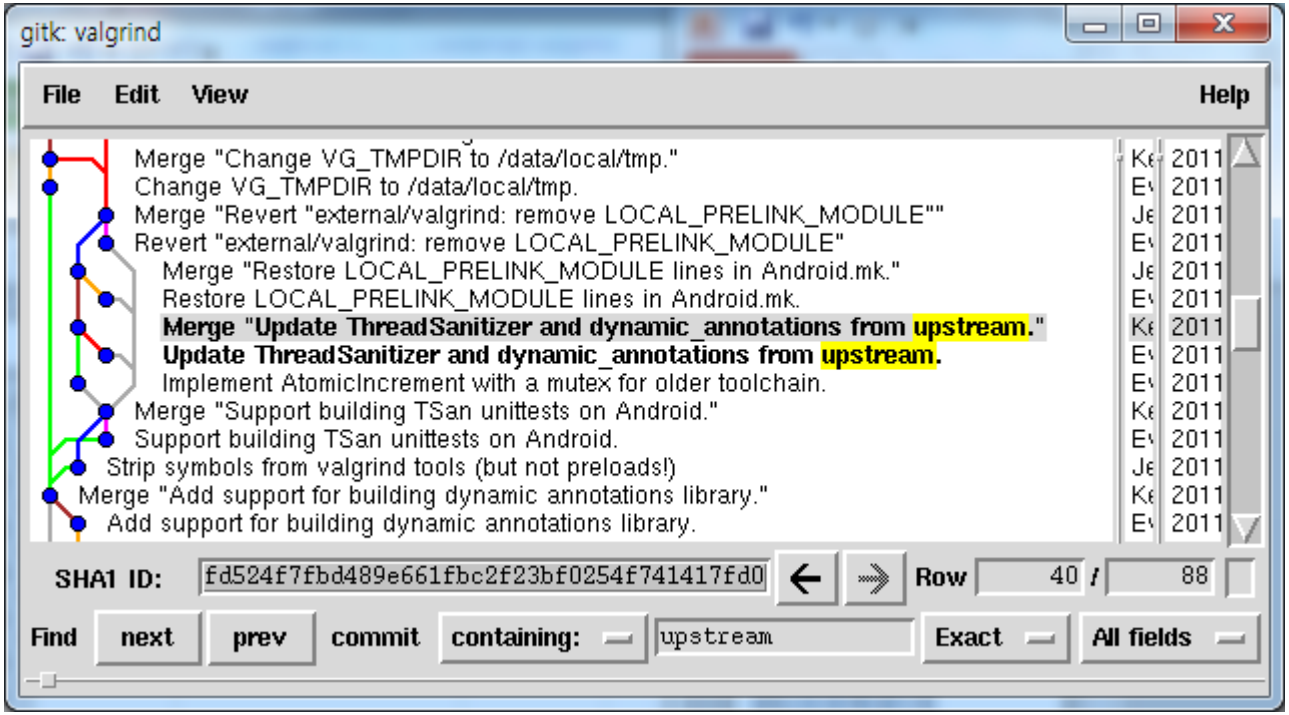
What is Valgrind?

Initial upstream import : valgrind 3.6.0 - 2011.2.4

Author	Commits	Insertions	Deletions	% of changes
Evgeniy Stepanov	24	74792	27816	36.51
Jeff Brown	4	129524	123	46.13
Kenny Root	12	15163	33595	17.35
Nick Kralevich	2	8	14	0.01



What is Valgrind?



Finding races and memory errors with LLVM instrumentation
AddressSanitizer, ThreadSanitizer

Timur Iskhodzhanov, Alexander Potapenko, Alexey Samsonov,
Kostya Serebryany, Evgeniy Stepanov, Dmitriy Vyukov

LLVM Dev Meeting
November 18, 2011

source : http://llvm.org/devmtg/2011-11/Serebryany_FindingRacesMemoryErrors.pdf

Valgrind Build and Installation for Real Target

$ cd ~/android-4.1.1_r1/external/valgrind/

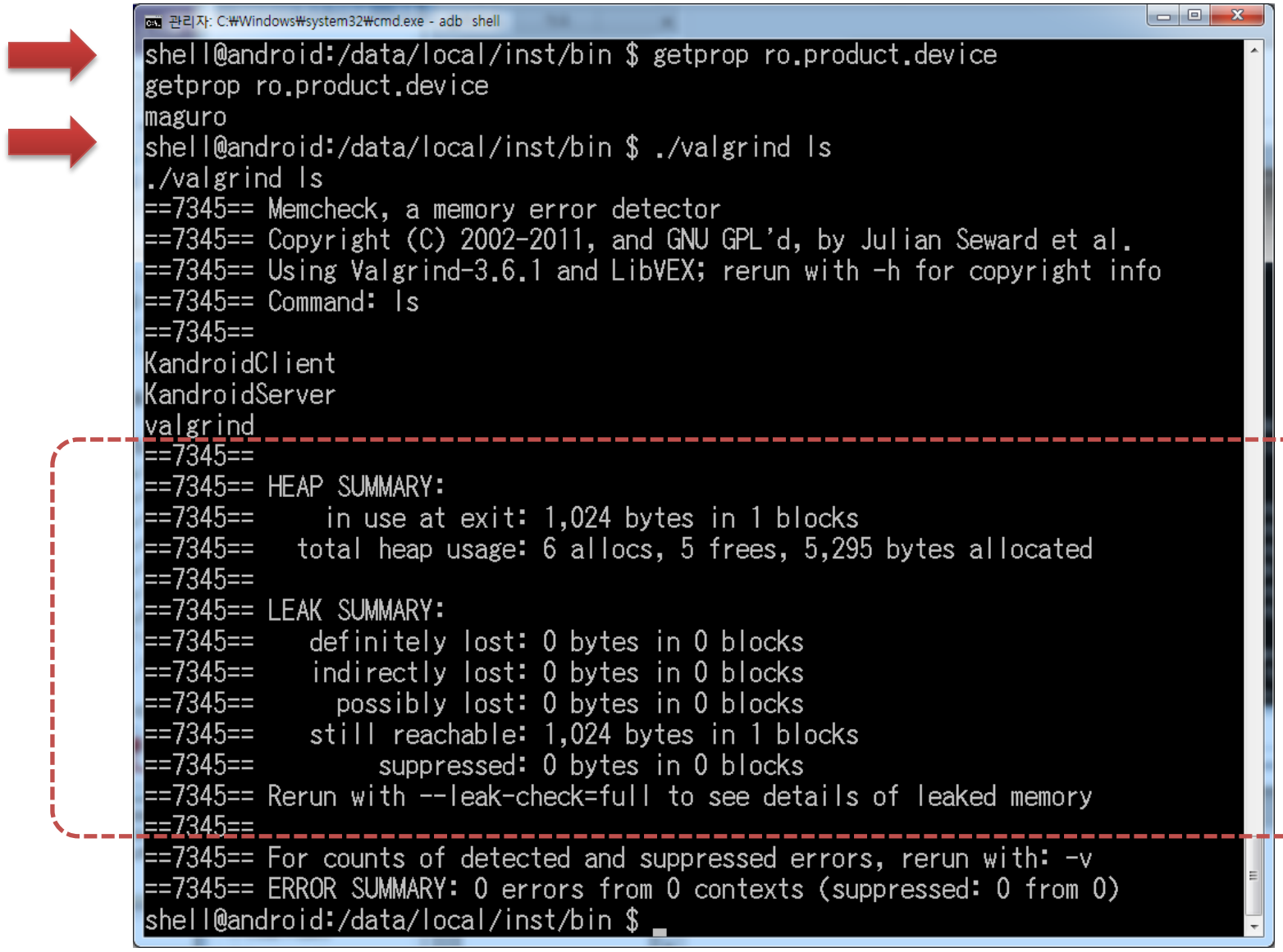
Modify VG_LIBDIR cflag.
$ vi ~/android-4.1.1_r1/external/valgrind/main/Android.mk
- -DVG_LIBDIR=\"/system/lib/valgrind\"
+ -DVG_LIBDIR=\"/data/local/inst/lib/valgrind\")

$ mm -B

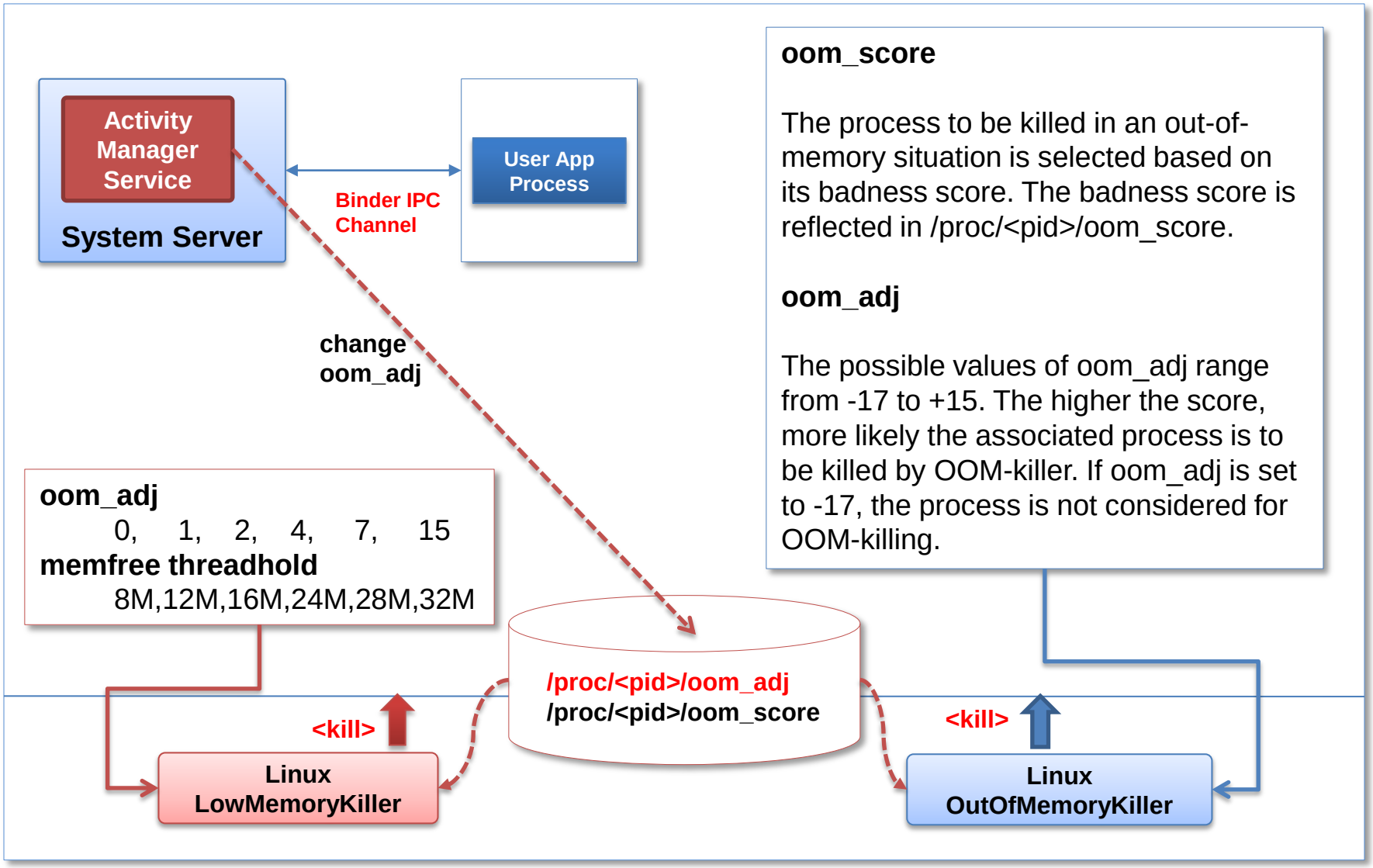
```
|-- bin
|   |-- racecheck_unittest
|   `-- valgrind
`-- lib
    `-- valgrind
        |-- cachegrind-arm-linux
        |-- callgrind-arm-linux
        |-- default.supp
        |-- drd-arm-linux
        |-- helgrind-arm-linux
        |-- massif-arm-linux
        |-- memcheck-arm-linux
        |-- none-arm-linux
        |-- tsan-arm-linux
        |-- vgpreload_core-arm-linux.so
        |-- vgpreload_drd-arm-linux.so
        |-- vgpreload_helgrind-arm-linux.so
        |-- vgpreload_massif-arm-linux.so
        |-- vgpreload_memcheck-arm-linux.so
        `-- vgpreload_tsan-arm-linux.so
```

```
{
  bionic_dlopen
  Memcheck:Addr4
  fun:__dl_strlen
  ...
  fun:__dl_find_library
  fun:__dl_dlopen
}

{
  bionic_dlopen_c
  Memcheck:Cond
  fun:__dl_strlen
  ...
  fun:__dl_find_library
  fun:__dl_dlopen
}

{
   sha1_block_data_order-reads-below-sp
   Memcheck:Addr4
   fun:sha1_block_data_order
}

{
   dvmPlatformInvoke-misinterpretation-1
   Memcheck:Addr4
   fun:dvmCallJNIMethod_virtualNoRef
}

{
   dvmPlatformInvoke-misinterpretation-2
   Memcheck:Addr4
   fun:dvmCallJNIMethod_staticNoRef
}
```

Valgrind Build and Installation for Real Target



Valgrind Issues : No Swap and OOM Killer



Memory Analysis Tools & Memory Management Methods

	General	Android + External Tools
system	• /proc/meminfo • top	• Low memory Killer • DDMS SysInfo
process	• /proc/<pid>/statm, maps, smaps • Vss, Rss, Pss, Uss	• procrank • dumpsys meminfo • smem
Java	Automatic Garbage Collection • Memory Leak Issues	• MAT • Android GC - Native Bitmap Heap • android.os.Debug.MemoryInfo • ActivityManager.getProcessMemoryInfo
JNI	Native to java object reference • Global Ref vs. Local Ref	• setprop dalvik.vm.checkjni true • setprop dalvik.vm.jniopts forcecopy • setprop debug.checkjni 1
C++	Smart Pointer • auto_ptr, scoped_ptr, unique_ptr, • shared_ptr, weak_ptr	• valgrind • sp, wp
C	glibc memory manager • ptmalloc (dlmalloc + pthread)	• valgrind • bionic libc (dlmalloc) • libc.debug.malloc

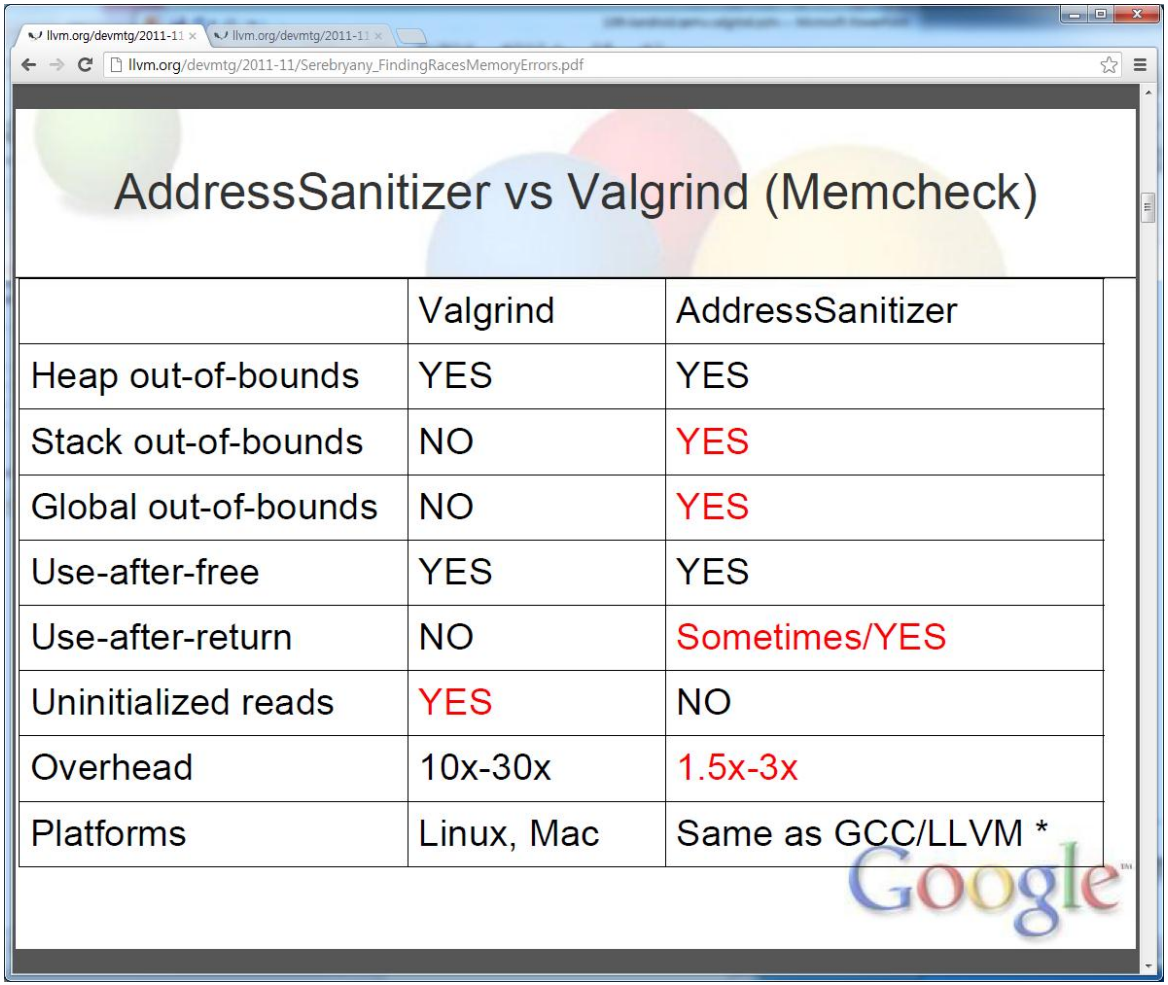
AddressSanitizer vs. Valgrind (Memcheck)

	AddressSanitizer	Valgrind/Memcheck	Dr. Memory	Mudflap	Guard Page
technology	CTI	DBI	DBI	CTI	Library
ARCH	x86	x86,ARM,PPC	x86	all(?)	all(?)
OS	Linux, Mac	Linux, Mac	Windows, Linux	Linux, Mac(?)	All (1)
Slowdown	**2x**	20x	10x	2x-40x	?
Detects:					
Heap OOB	**yes**	yes	yes	yes	some
Stack OOB	**yes**	no	no	some	no
Global OOB	**yes**	no	no	?	no
UAF	**yes**	yes	yes	yes	yes
UAR	**some**	no	no	no	no
UMR	no	yes	yes	?	no
Leaks	not yet	yes	yes	?	no

DBI: dynamic binary instrumentation / **CTI**: compile-time instrumentation
UMR: uninitialized memory reads / **UAF**: use-after-free (aka dangling pointer)
UAR: use-after-return / **OOB**: out-of-bounds
x86: includes 32- and 64-bit.
Guard Page: a family of memory error detectors (Electric fence or DUMA on Linux, Page Heap on Windows, Guard Malloc in Mac)

source : http://code.google.com/p/address-sanitizer/wiki/ComparisonOfMemoryTools

AddressSanitizer vs. Valgrind (Memcheck)



	Valgrind	AddressSanitizer
Heap out-of-bounds	YES	YES
Stack out-of-bounds	NO	YES
Global out-of-bounds	NO	YES
Use-after-free	YES	YES
Use-after-return	NO	Sometimes/YES
Uninitialized reads	YES	NO
Overhead	10x-30x	1.5x-3x
Platforms	Linux, Mac	Same as GCC/LLVM *

source : http://llvm.org/devmtg/2011-11/Serebryany_FindingRacesMemoryErrors.pdf

Advanced Topics (1) – LLVM AddressSanitizer

```
$ svn co http://llvm.org/svn/llvm-project/llvm/trunk llvm
$ cd llvm/tools ; svn co http://llvm.org/svn/llvm-project/cfe/trunk clang
$ cd ../projects ; svn co http://llvm.org/svn/llvm-project/compiler-rt/trunk compiler-rt
$ svn co http://llvm.org/svn/llvm-project/test-suite/trunk test-suite
$ cd ../.. ; mkdir build ; cd build/ ; ../llvm/configure ; make
$ export PATH=<your_path>/Release+Asserts/bin/:$PATH:.
```

```
$ cd ../tools/clang
$ svn co http://src.chromium.org/svn/trunk/src/tools/clang/scripts
$ cd ../../
$ tools/clang/scripts/update.sh
```

```
$ cat > test/use-after-free.c
#include <stdlib.h>
int main() {
  char *x = (char*)malloc(10 * sizeof(char*));
  free(x);
  return x[5];
}

$ clang -faddress-sanitizer -O1 -fno-omit-frame-pointer -g   tests/use-after-free.c
```

Advanced Topics (1) – LLVM AddressSanitizer

```
$ ./a.out
=================================================================
==8069== ERROR: AddressSanitizer:
heap-use-after-free on address 0x7f466f3ebf45 at pc 0x409284 bp 0x7fff3dd175f0 sp 0x7fff3dd175e8
READ of size 1 at 0x7f466f3ebf45 thread T0
    #0 0x409283 (/home/jsyang/llvm/llvm/a.out+0x409283)
    #1 0x7f466e85ec4c (/lib/libc-2.11.1.so+0x1ec4c)
0x7f466f3ebf45 is located 5 bytes inside of 80-byte region [0x7f466f3ebf40,0x7f466f3ebf90)
freed by thread T0 here:
    #0 0x4092d0 (/home/jsyang/llvm/llvm/a.out+0x4092d0)
    #1 0x40924a (/home/jsyang/llvm/llvm/a.out+0x40924a)
    #2 0x7f466e85ec4c (/lib/libc-2.11.1.so+0x1ec4c)
previously allocated by thread T0 here:
    #0 0x409390 (/home/jsyang/llvm/llvm/a.out+0x409390)
    #1 0x40923f (/home/jsyang/llvm/llvm/a.out+0x40923f)
    #2 0x7f466e85ec4c (/lib/libc-2.11.1.so+0x1ec4c)
```

What Every Android Programmer Should Know About Memory?

- Paging : Contiguous Virtual Address vs. Noncontiguous Physical Address
- MMU : hardware vs. software
- Page : Clean/Dirty/Locked, File Mapping vs. Anonymous Mapping
  Reference Counted,
- Page Table and Protection Bit :
  PRESENT, PROTNONE, RW, USER, DIRTY, ACCESSED
- OOM Killer vs. Low Memory Killer
- Shared Memory vs. Shared Memory Virtual File System
- SYS V Shm, POSIX Shm, Berkeley Shm vs. Anonymous Shared Memory
- Process Address Segment : Code, Data, BSS, Heap, Stack, Mmap
- Memory Allocation : Static vs. Dynamic, Global vs. Stack
- malloc is a library function, not a system call
- Various malloc library implementation : dlmalloc, ptmalloc, …
- Android C++ reference management :
  sp, wp – template + operator overloading
- Android Java GC includes the native Bitmap Heap.

Q & A

www.kandroid.org