olibc: Another C Library optimized for embedded Linux

Jim Huang (黃敬群) <**jserv**@0xlab.org>

Developer, 0xlab

Feb 22, 2013 / Embedded Linux Conference

Rights to copy

http://0xlab.org/

contact@0xlab.org

Corrections, suggestions, contributions and translations are welcome!

Latest update: Feb 22, 2013

What I will discuss about...

- I learned a bit about the toolchain and system library optimizations while developing Android based projects.
- It might be a good idea to “reuse” them in ordinary embedded Linux projects.
- Therefore, I plan to emphasize on how to leverage the engineering efforts originally from Android world.
  - Review C library characteristics
  - Toolchain optimizations
  - Build configurable runtime
  - Performance evaluation

Related talks

- Android Builders Summit 2013
  - LLVMLinux: Compiling Android with LLVM - Behan Webster, Converse in Code - Behan Webster
- Embedded Linux Conference 2013
  - Toybox: Writing a new Linux Command Line from Scratch - Rob Landley
  - System-wide Memory Management without Swap - Howard Cochran
  - Bringing kconfig to EGLIBC - Khem Raj

Agenda

(1) Take from Android

(2) olibc: configurable

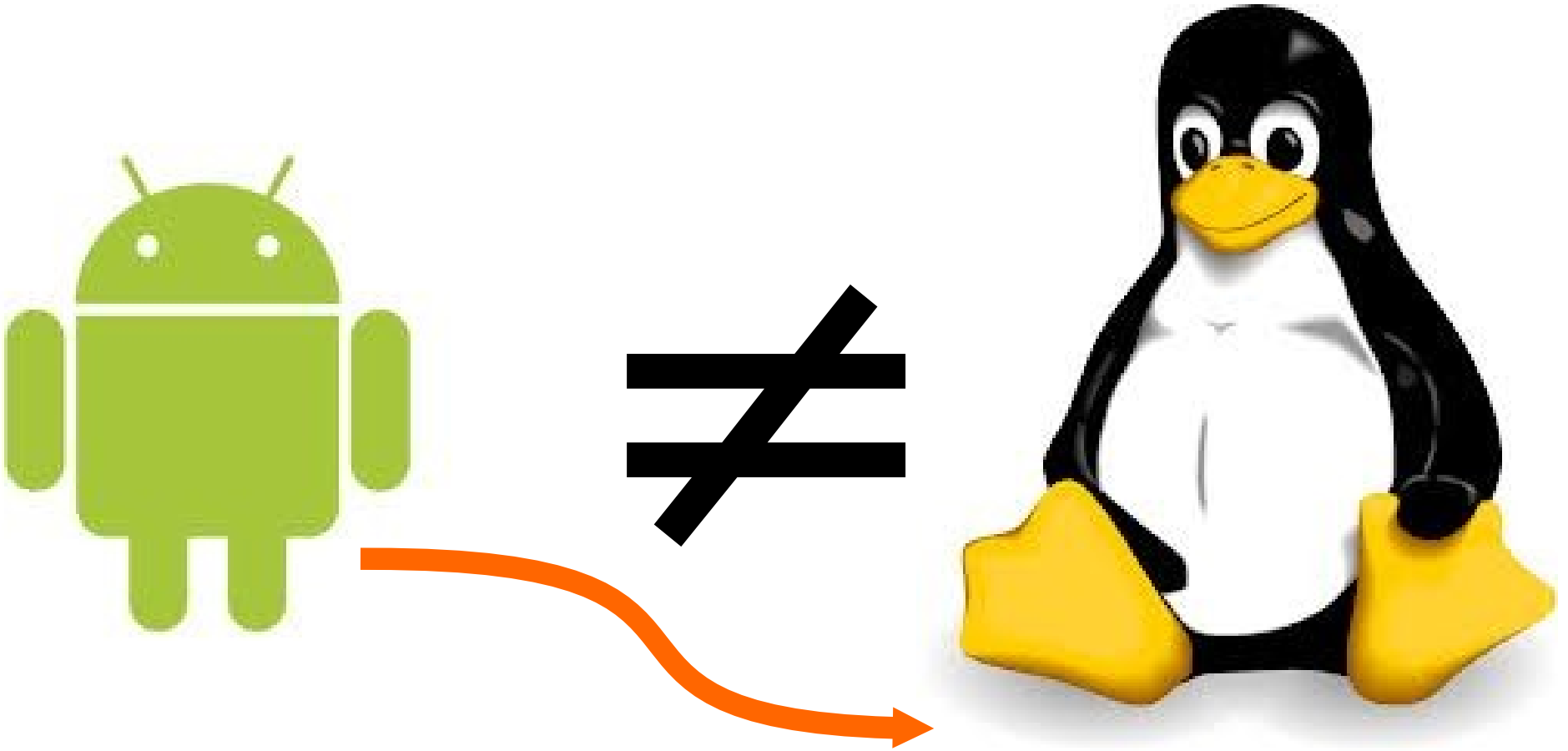
(3) Optimizations

(4) Action Items

Take from Android

bionic libc, dynamic linker, debugging facilities

We know, Android is not Linux, but...



We're __taking__ someting useful back to embedded Linux.

from Rob Landley's talk

- “Dalvik is the new ROM basic"
- ...
- “why not extend toolbox/bionic instead of replace?
  - just enough to run dalvik. (The new ROM BASIC.)"

Our "something useful" is the base to launch Dalvik Virtual Machine and the above Android Framework

Source: http://www.landley.net/talks/celf-2013.txt

Why build bionic-derived libc?

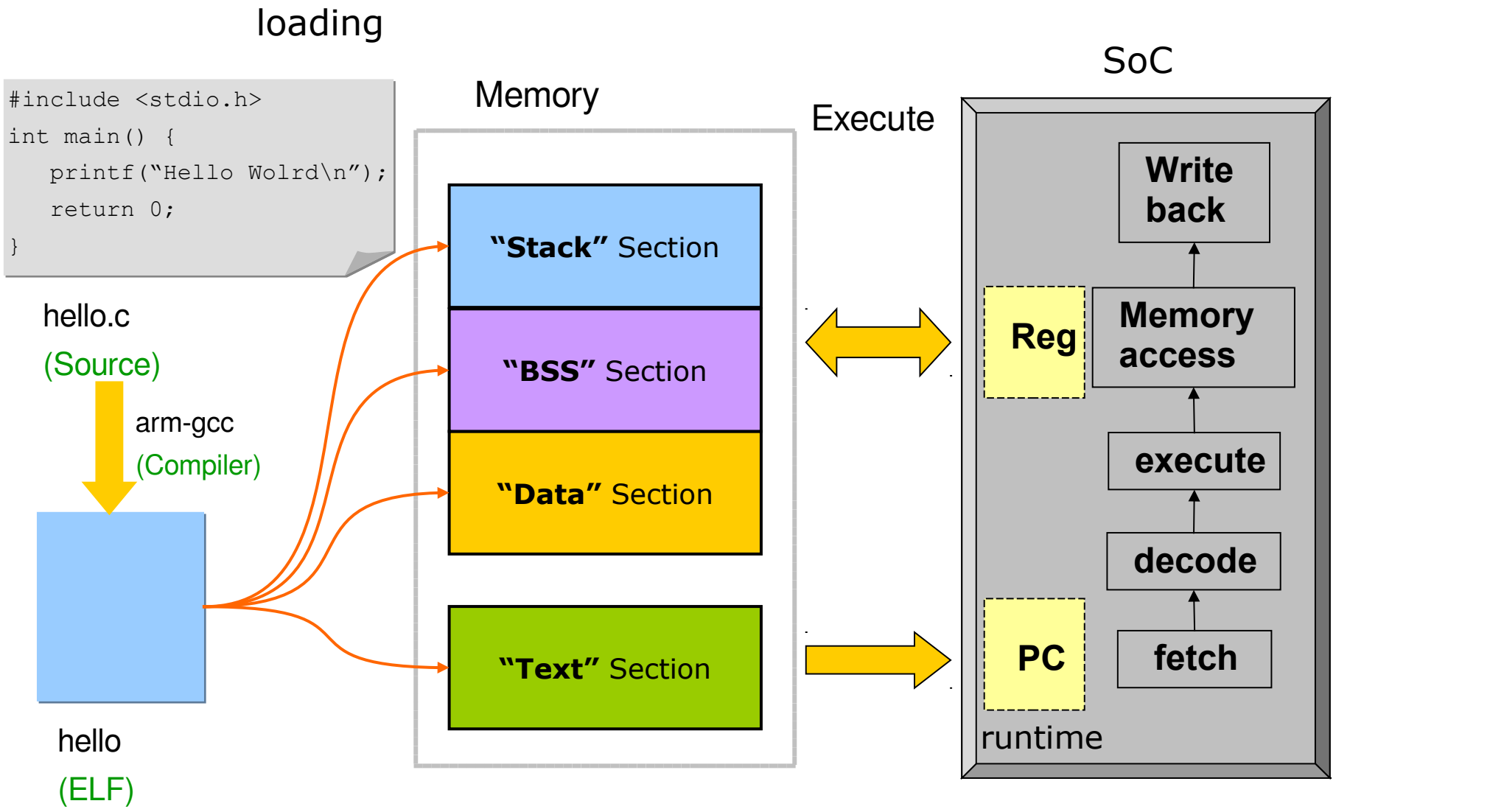
- License
  - glibc (LGPL), uClibc (LGPL), dietlibc (GPL), musl (MIT)
- Optimized for major (consumer) targets
  - ARMv7 + MIPS + Intel Atom optimizations
  - glibc (good arm/mips/atom), uClibc (simpler arm/mips/x86), dietlbc (N/A), musl (simple x86)
- API coverage is getting more complete by versions.
- Catch up with latest SoC technologies
  - Contributors: Google, Intel, Qualcomm, Texas Instruments, NVIDIA, ST-Ericsson, Linaro, MIPS, etc.
- The problem is, Android is not a community project.

"Copyleft is dying.
GPLv2 was category killer, synonymous with copyleft." – Rob Landley

Goals of olibc

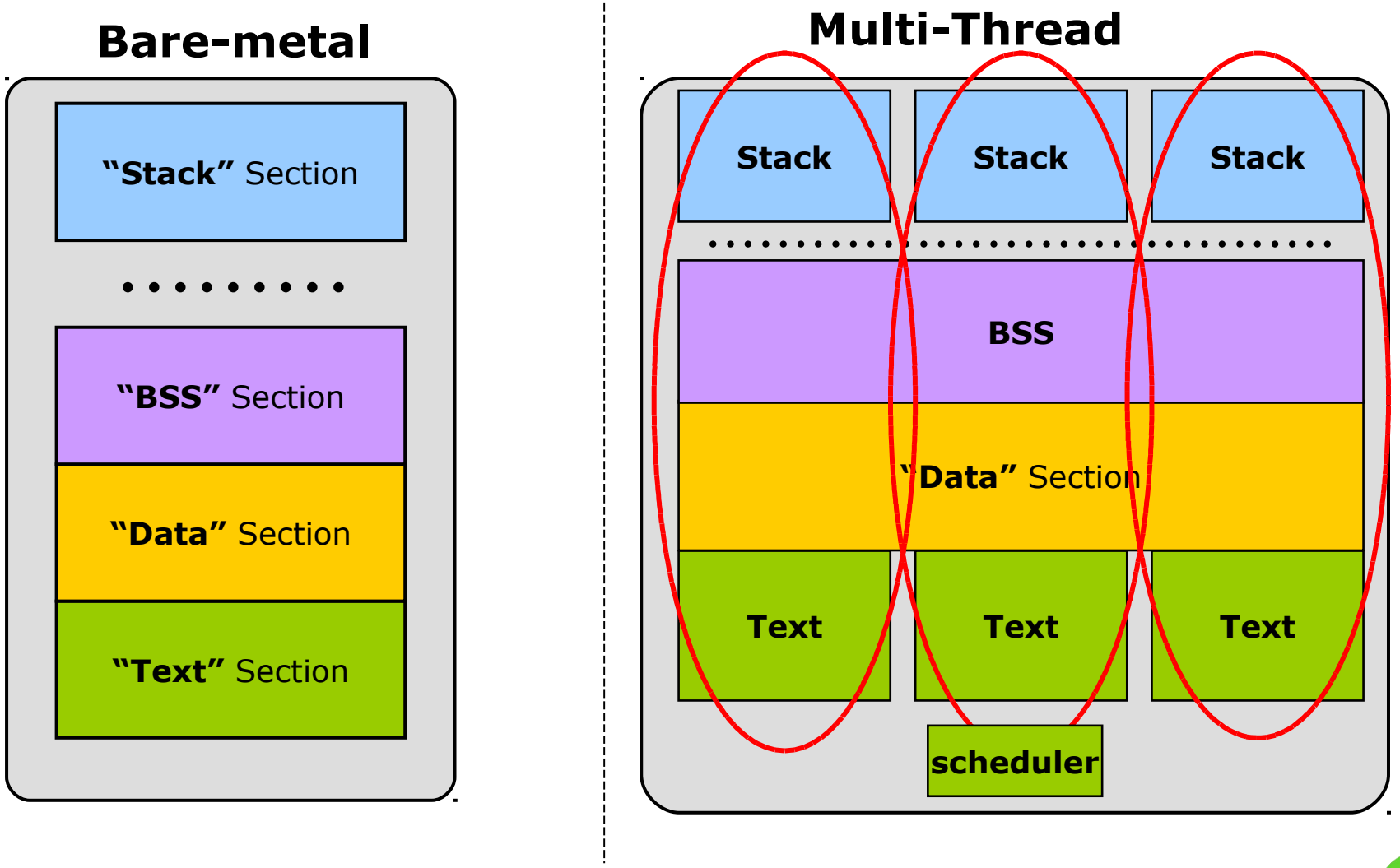
- Create small, fast, free, and standard-compliant implementation for C Library.
- Offer configurable levels of functionality and should scale from tiny embedded Linux up to general purpose environments such as Android-powered smart devices.
- Provide system utilities around C library
  - benchmarking, validation, prelinker, ...
- Introduce steady security, performance, and comformance fixes.

Programming Model

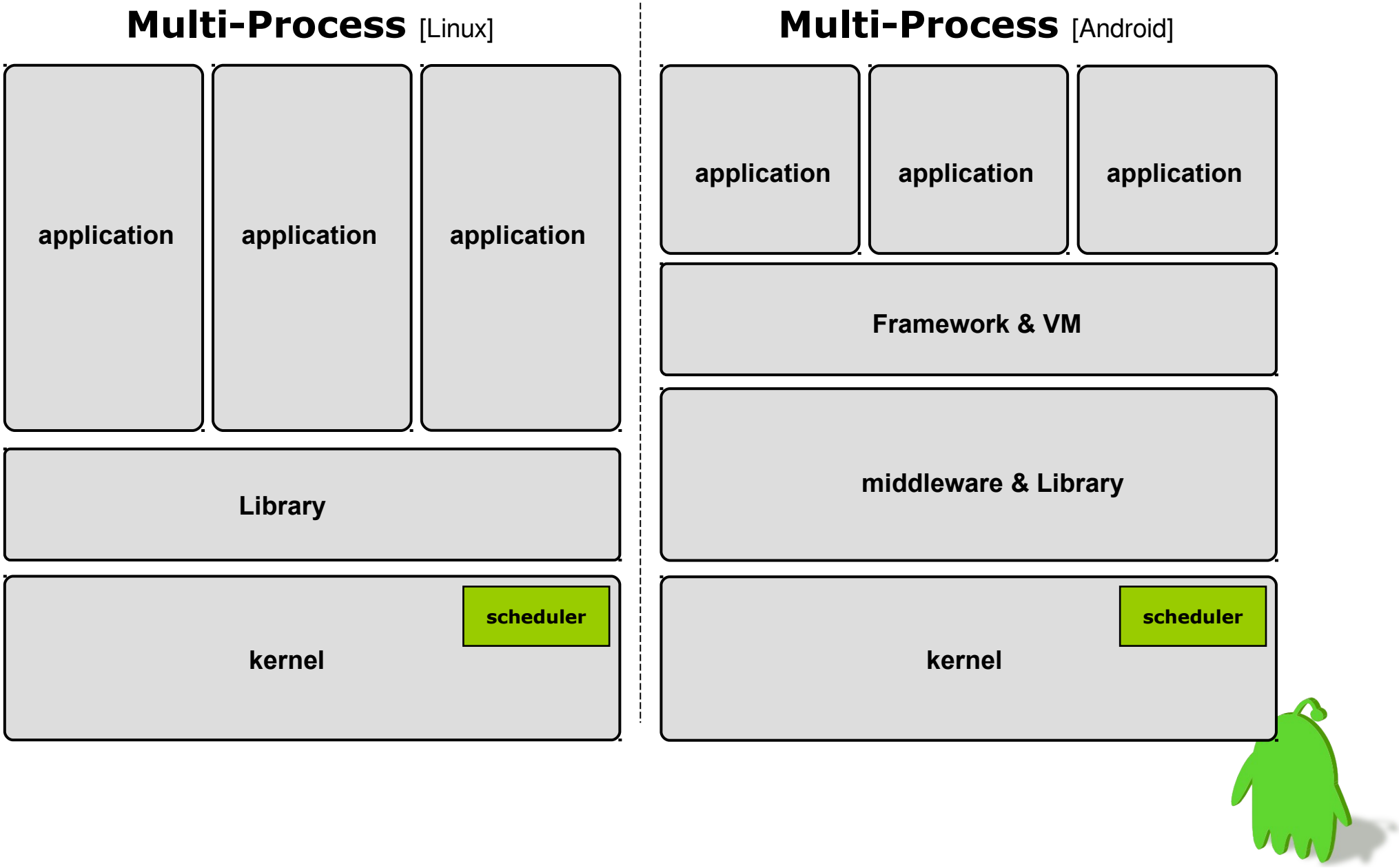


Let's review the programming model...

Programming Model (multi-threaded)



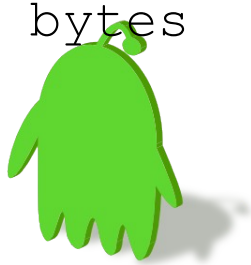
Programming Model (multi-process)



bionic libc

- Small C Library implementation
  - mixture of NetBSD (libc) and FreeBSD (libm)
- BSD license
- Partially POSIX compatible; not compatible with glibc
- No SysV IPC support (due to Binder IPC)
- Support for ARM (w/ Thumb), x86, MIPS
- Fast pthread implementation (based on futexes)
- Small footprint

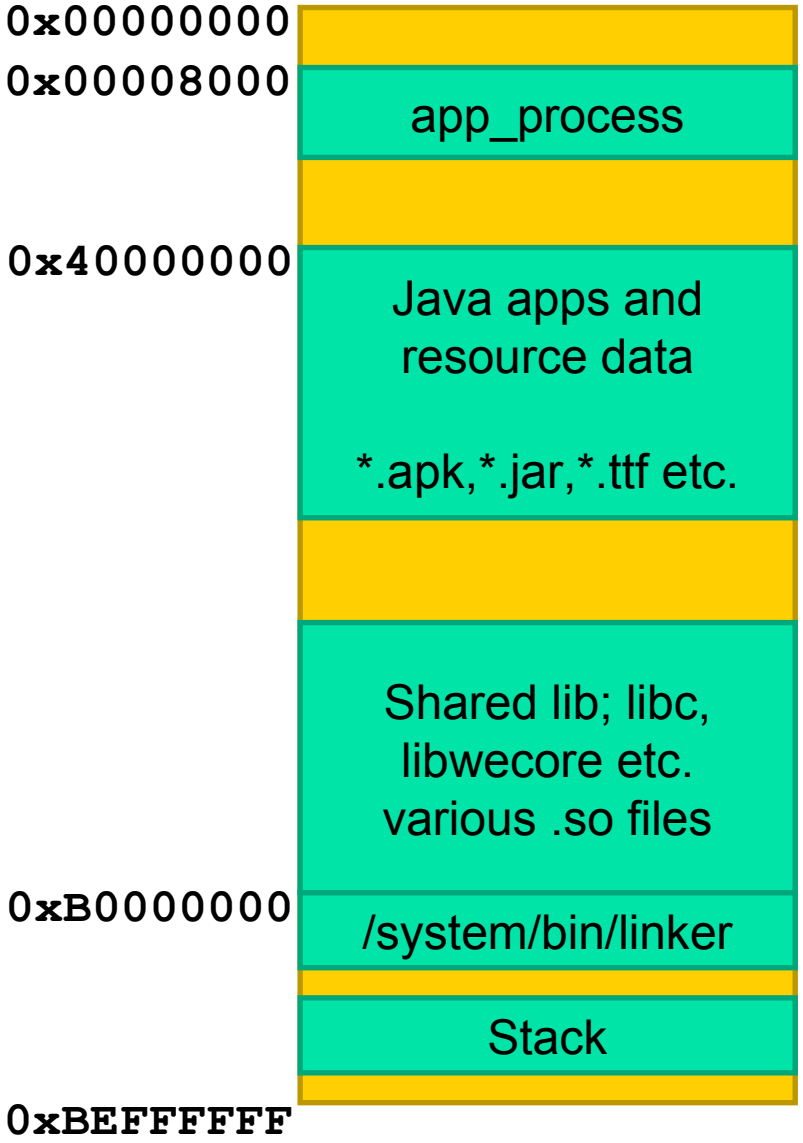
```
glibc   2.11    : /lib/libc.so          → 1,208,224 bytes
uClibc  0.9.30  : /lib/libuClibc.so     →   424,235 bytes
bionic  2.1     : /system/lib/libc.so   →   243,948 bytes
```



Not in bionic libc

- Complete wide chars
- C++ exceptions (limited since NDKr5)
- Full C++ STL
- Full POSIX Thread

Memory Map [Android pre-4.x]



No memory with execution attribute

File mapping direction

Some memory are of execution attribute

File mapping direction

Memory related changes

- ASLR (Address space layout randomization) since Android 4.0
  - help protect system and third party applications from exploits due to memory-management issues
  - PIE (Position Independent Executable) is added since Android 4.1
  - original ELF prelinker was removed
- AddressSanitizer since 4.1

AddressSanitizer vs. Valgrind

	Valgrin	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
Host platform	Linux, Mac OS X	where (latest) GCC/LLVM runs

AddressSanitizer

```
=================================================================
==7161== ERROR: AddressSanitizer global-buffer-overflow on address 0x2a002194 at
pc 0x2a00051b bp 0xbeeafb0c sp 0xbeeafb08
READ of size 4 at 0x2a002194 thread T0
    #0 0x40022a4b (/system/lib/libasan_preload.so+0x8a4b)
    #1 0x40023e77 (/system/lib/libasan_preload.so+0x9e77)
    #2 0x4001c98f (/system/lib/libasan_preload.so+0x298f)
    #3 0x2a000519 (/system/bin/global-out-of-bounds+0x519)
    #4 0x4114371d (/system/lib/libc.so+0x1271d)
0x2a002194 is located 4 bytes to the right of global variable 'global_array
(external/test/global-out-of-bounds.cpp)' (0x2a002000) of size 400
Shadow byte and word:
  0x05400432: f9
  0x05400430: 00 00 f9 f9
More shadow bytes:
  0x05400420: 00 00 00 00
  0x05400424: 00 00 00 00
  0x05400428: 00 00 00 00
  0x0540042c: 00 00 00 00
=>0x05400430: 00 00 f9 f9
  0x05400434: f9 f9 f9 f9
  0x05400438: 00 00 00 00
  0x0540043c: 00 00 00 00
  0x05400440: 00 00 00 00
Stats: 0M malloced (0M for red zones) by 35 calls
Stats: 0M realloced by 0 calls
```

```
int global_array[100] = {-1};
int main(int argc, char **argv) {
    return global_array[argc + 100];  /* BOOM */
}
```

Shared library issues

- Older Android dynamic linker has an arbitrary low (for larger applications such as GStreamer, LibreOffice) limit on number of shared libs: 128
  - Until Sep 12, 2012, dynamically allocating soinfo-structs in linker is implemented.
- Mozilla: Use a hacked-up copy of the bionic linker in our own code, in addition to the system one.
  - two run-time linkers not aware of each others ended up a failure

C++ Integrations

	C++ Exception	C++ RTTI	Standard Library
system	No	No	No
gabi++	No	Yes	No
stlport	No	Yes	Yes
gnustl	Yes	Yes	Yes

- olibc provides stlport, which depends on wchar support in libc.

Debuggerd

- Nice embedded-specific crash handler
  - used on all Android devices
- Crash report data placed in log/tombstone
- Debuggerd also facilitates connecting debugger to dying process
  - Can halt and wait for gdb to attach to the process
- Apache license

How Debuggerd works

- Debuggerd acts as a crash-handling daemon
- Adds default signal handler to each process, which handles any signals that generate core
  - included in bionic, thus every application gets it
- Signal handler captures deadly signal and contacts debuggerd
- Debuggerd records information using ptrace (registers, stack, memory areas), and /proc
- Has builtin ARM stack unwinder for generating a backtrace
- Automatically rotates a fixed number of crash reports
- Reference:

  https://wiki.linaro.org/WorkingGroups/ToolChain/Outputs/LibunwindDebuggerd

unwinding

- Unwinding = processing stack and memory image to create a backtrace
- Backtrace is very compact - summarizes stack information nicely
- Local variables usually not available
- Different methods available, depending on compilation flags

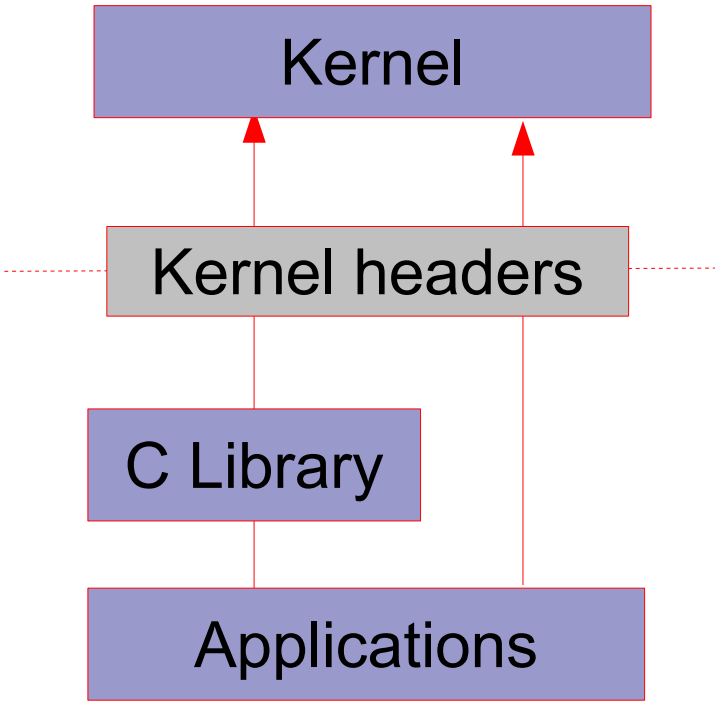
Crash Handler

- New crash handler written by Tim Bird of Sony
  – Based on debuggerd from Android
- Implemented as a core file handler
- Writes crash report to a “tombstone_0x” file in /tmp/tombstones
- Writes information from /proc, ptrace, and kernel log buffer
- Also writes some information to the kernel log
- Information: http://elinux.org/Crash_handler

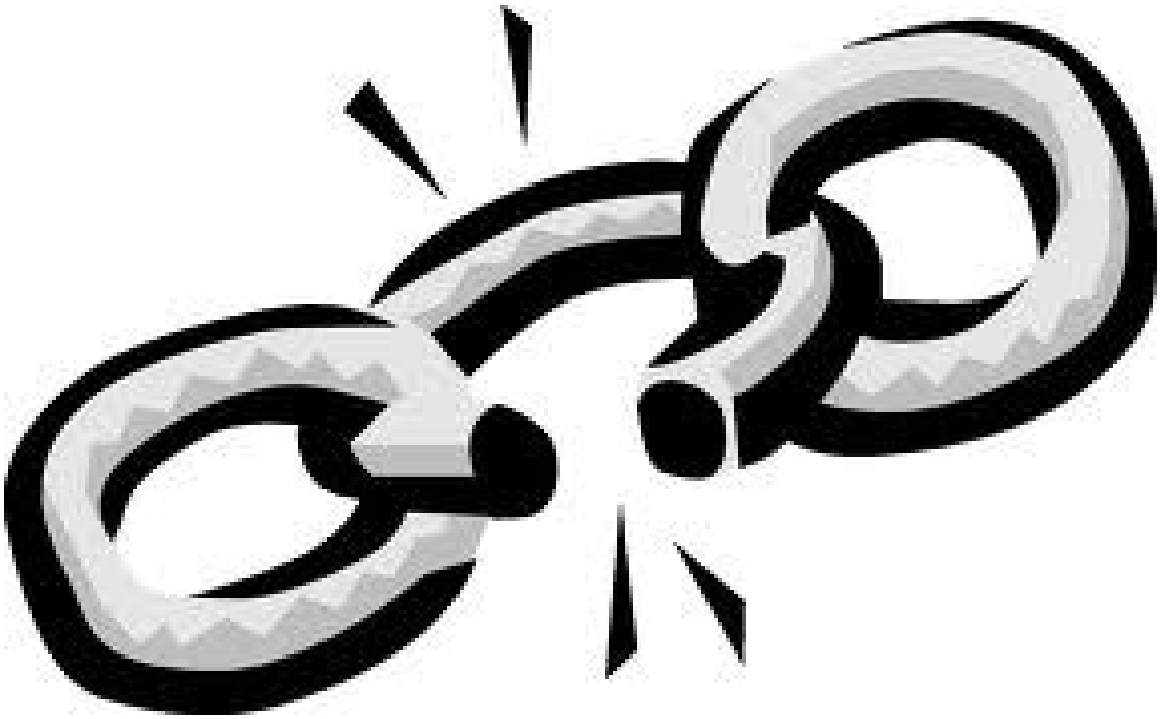
License Issue

- THE BIONIC LIBRARY: DID GOOGLE WORK AROUND THE GPL? brownrudnick, Mar 2011
- Bionic Revisited: What the Summary Judgment Ruling in Oracle v. Google Means for Android and the GPL, brownrudnick, Nov 2011
  - Google tries to “clean” Linux kernel headers to avoid the GPL

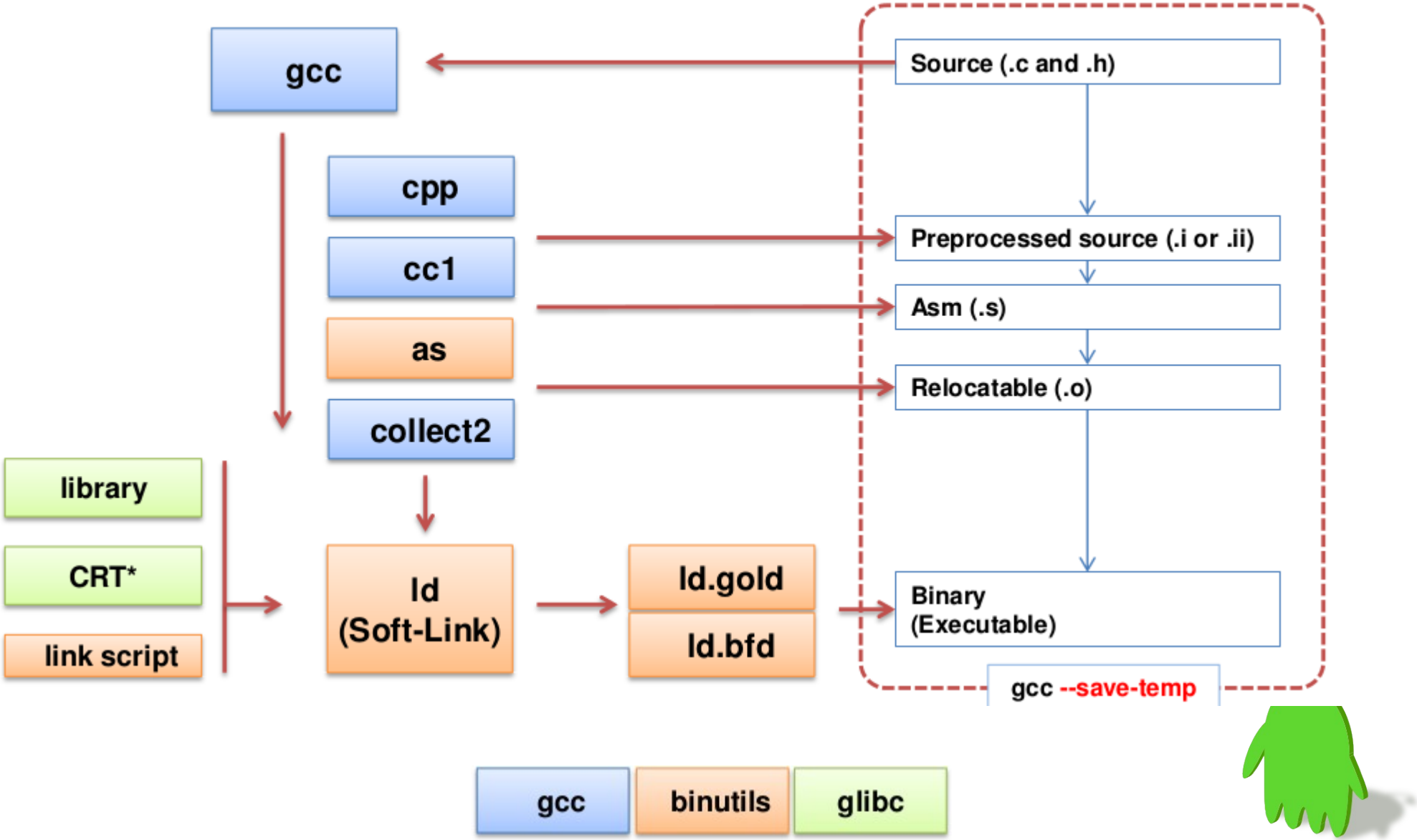
olibc: configurable



Major problem: broken toolchain



How Toolchain works



External Toolchain Issues

- CodeSourcery Toolchain doesn't use gold linker, and Android's original build flags are a bit aggressive.
  - e.g. ICF (Identical Code Folding), which is gold only redundancy elimination
  - Option: **`--icf`** (known to bring 5% size reduction)
- Default link script doesn't take olibc into consideration.
- Sometimes, toolchains have optimization bugs

Build Android compatible toolchain

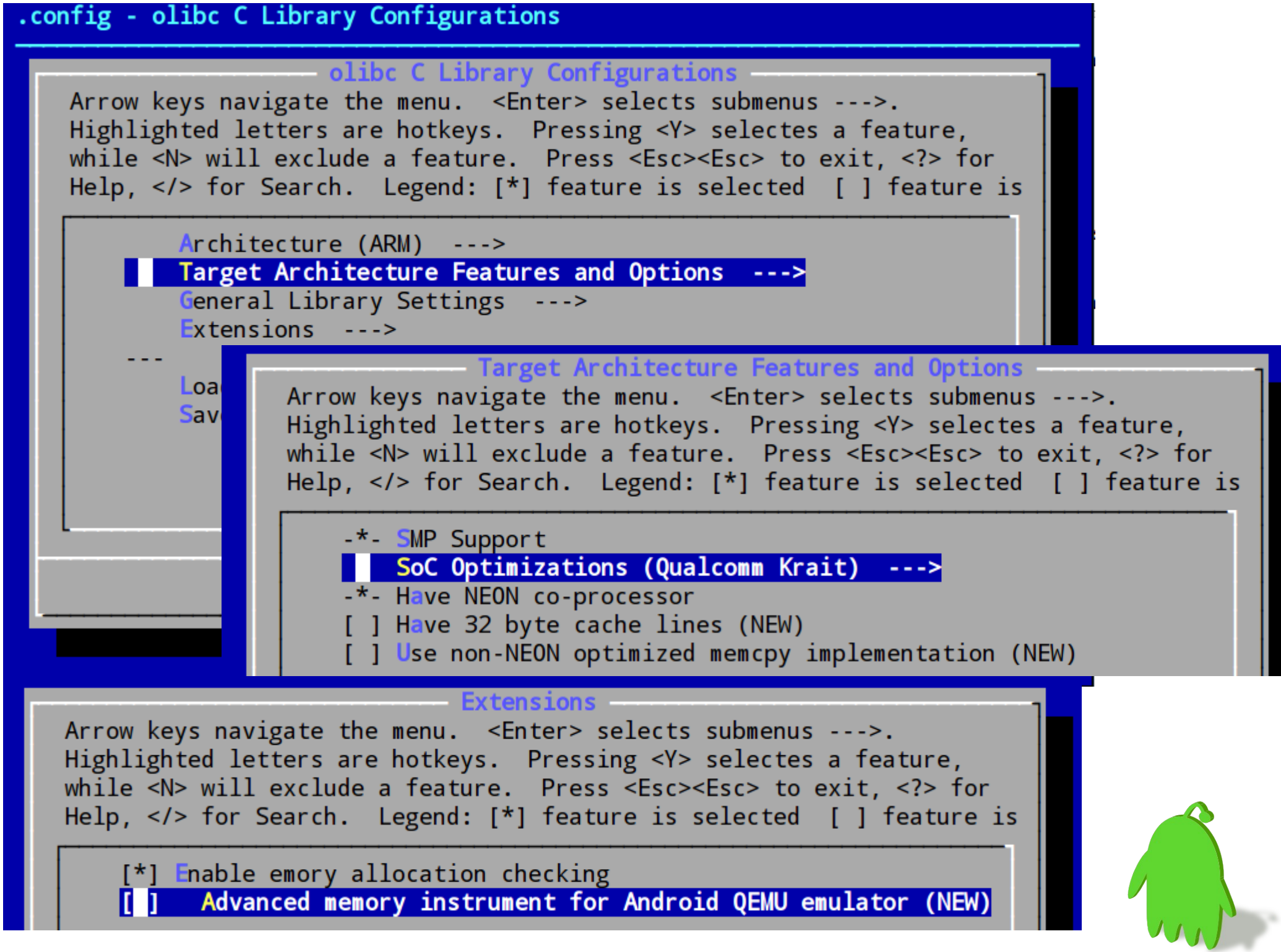
- Barebone-style building:
  - Inside Android tree
  - Specify all system and bionic header file paths, shared library, paths, libgcc.a, crtbegin_*.o, crtend_*.o, etc.
- Standalone-style building:
  - Convenient for native developers:

    arm-xxx-eabi-gcc -mandroid --sysroot=<path-to-sysroot > hello.c -o hello

    (<path to sysroot> is a pre-compiled copy of Bionic)

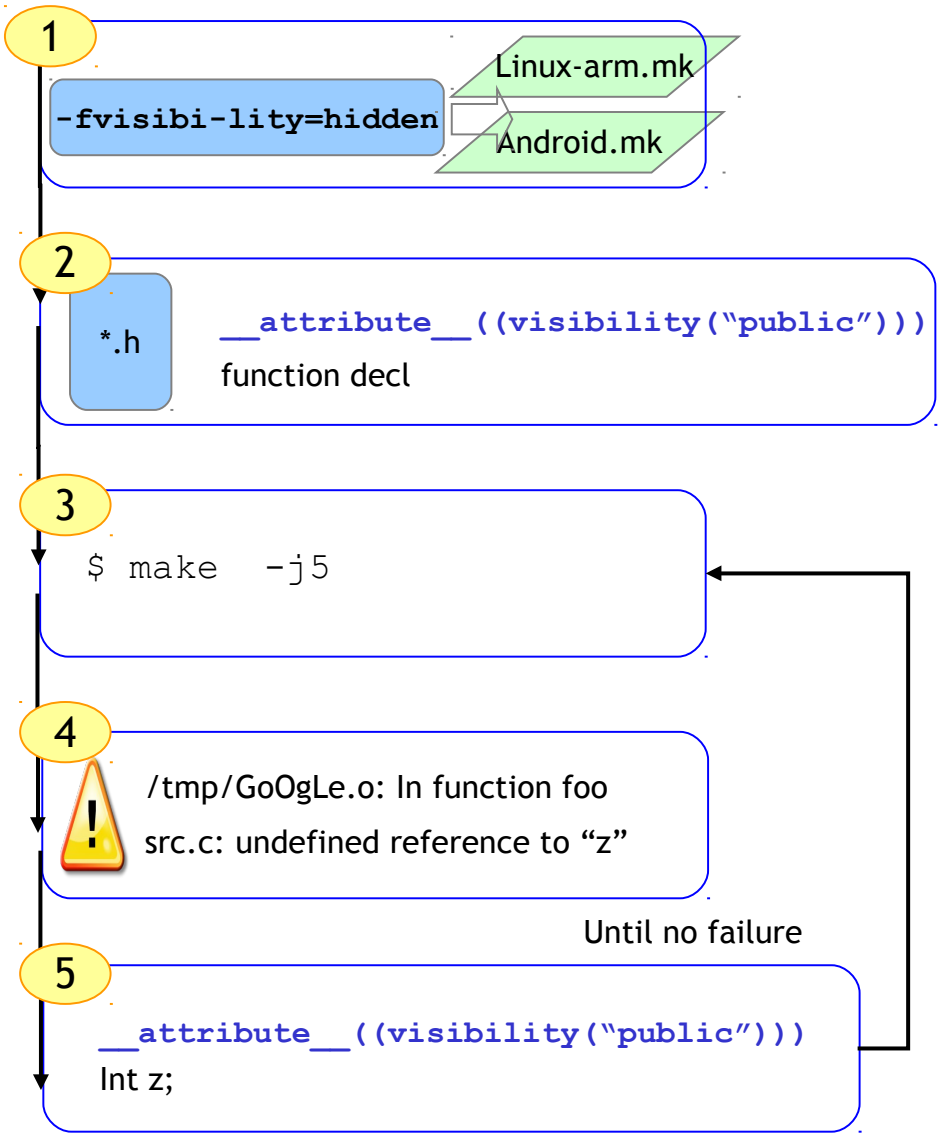
olibc: Configurable and Optimized

- Configured using the Kconfig language pioneered by the Linux kernel
  - Extensions, Library settings, crash handler, ...
- Encapulate the Android build system to become simpler and dedicated one.
- Allow full optimization techniques originally applied by Android including implementation and toolchain
  - SoC enhancements
- Use repo to manage source tree
  - `repo init -u https://github.com/olibc/manifest.git`
  - `repo sync`


.config - olibc C Library Configurations
olibc C Library Configurations
Arrow keys navigate the menu. <Enter> selects submenus --->.
Highlighted letters are hotkeys. Pressing <Y> selectes a feature,
while <N> will exclude a feature. Press <Esc><Esc> to exit, <?> for
Help, </> for Search. Legend: [*] feature is selected [ ] feature is
Architecture (ARM) --->
Target Architecture Features and Options --->
General Library Settings --->
Extensions --->
Target Architecture Features and Options
Arrow keys navigate the menu. <Enter> selects submenus --->.
Highlighted letters are hotkeys. Pressing <Y> selectes a feature,
while <N> will exclude a feature. Press <Esc><Esc> to exit, <?> for
Help, </> for Search. Legend: [*] feature is selected [ ] feature is
-*- SMP Support
SoC Optimizations (Qualcomm Krait) --->
-*- Have NEON co-processor
[ ] Have 32 byte cache lines (NEW)
[ ] Use non-NEON optimized memcpy implementation (NEW)
Extensions
Arrow keys navigate the menu. <Enter> selects submenus --->.
Highlighted letters are hotkeys. Pressing <Y> selectes a feature,
while <N> will exclude a feature. Press <Esc><Esc> to exit, <?> for
Help, </> for Search. Legend: [*] feature is selected [ ] feature is
[*] Enable emory allocation checking
[ ] Advanced memory instrument for Android QEMU emulator (NEW)

Optimizations

Build Tweaks: Symbol Visibility



1. Goal: Visibility of a function should match the API spec in programmer's design.

2. Solution:
First, systematically applying the 5 steps.
Fundamentally, need to go through the APIs of each library:
   - ✓ Consciously decide what should be "public" and what shouldn't.

3. Result: ~500 KB savings for opencore libs

4. Key: The whole hidden functions can be garbage collected if unused locally:

5. Toolchain's options:
   `-ffunction-sections,`
   `-Wl,--gcsections,`

Build Tweaks: code size

- Android default inline options:

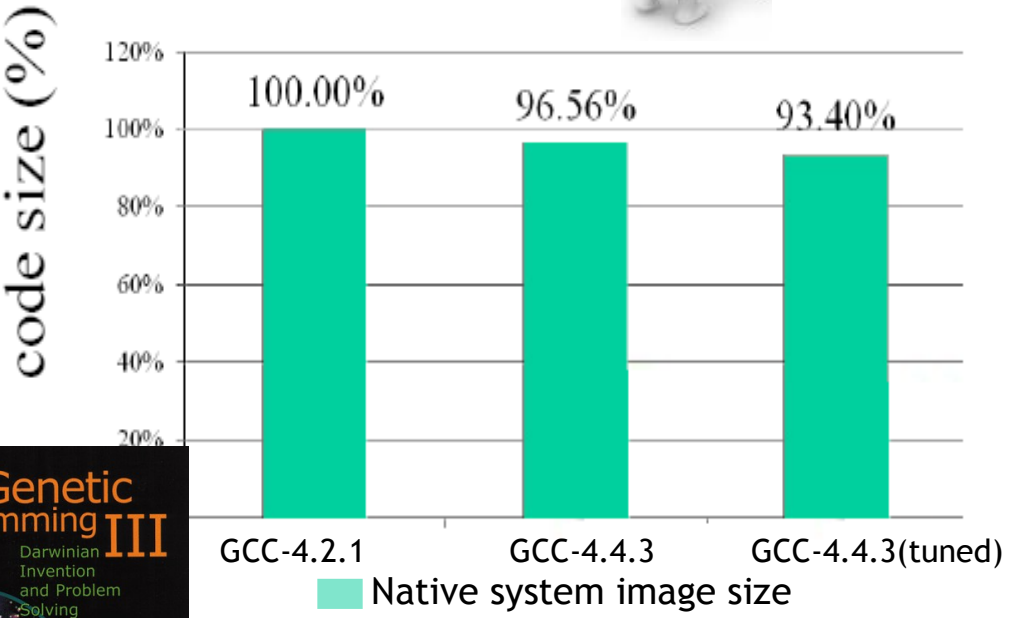
```
-finline-functions
-fno-inline-functions-called-once
```

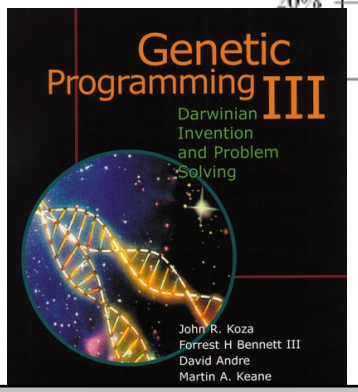
(unit: byte)

	GCC-4.2.1	GCC-4.4.3	GCC-4.4.3 (tuned inline options)
Native system image	23,839,291	23,027,032	22,087,436

Use genetic algorithm to find best-fit:

```
-finline
-fno-inline-functions
-finline-functions-called-once
--param max-inline-insns-auto=62
--param inline-unit-growth=0
--param large-unit-insns=0
--param inline-call-cost=4
```

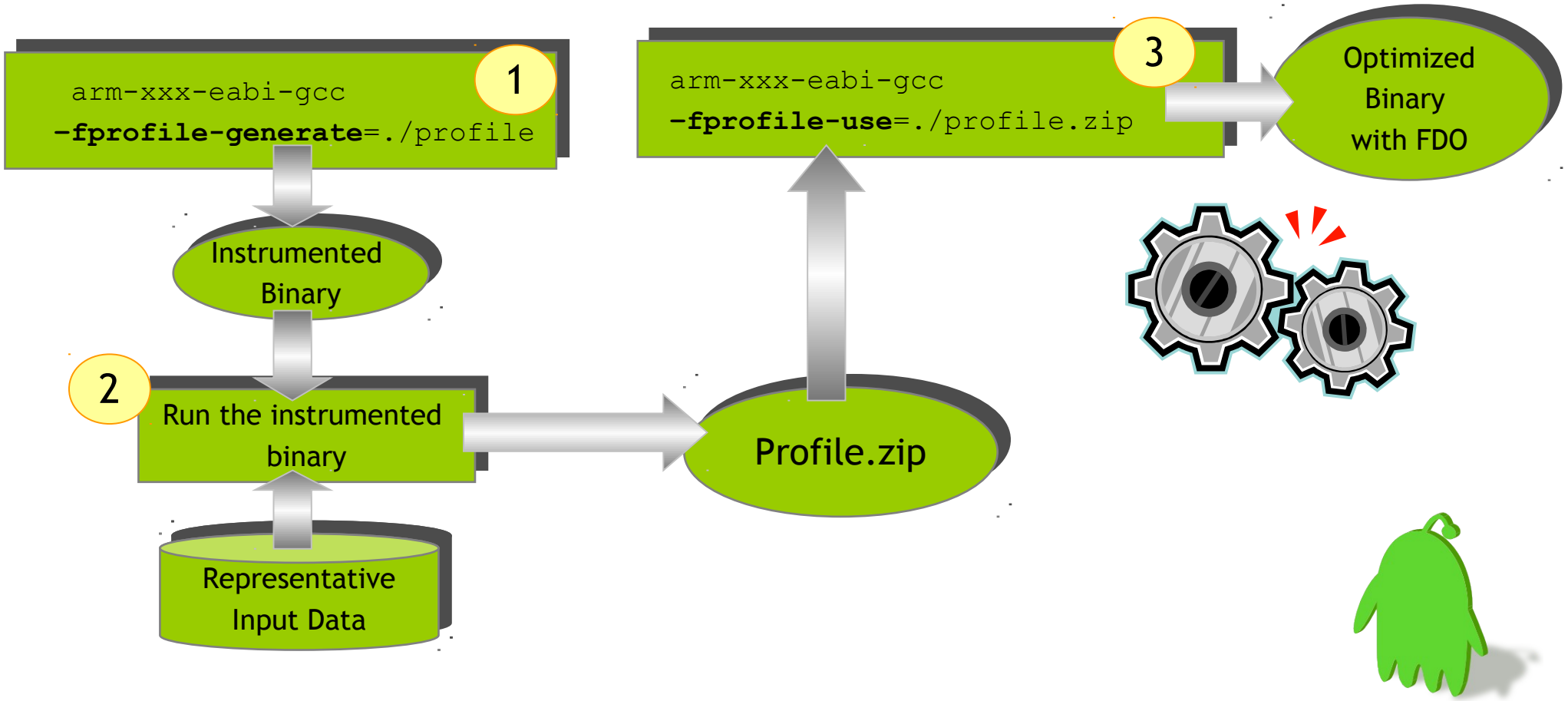




Source: Smaller and Faster Android, Shih-wei Liao, Google Inc.

Instrumentation-based FDO (Feedback-Directed Optimization)

1. Build twice.
2. Find representative input
3. Instrumentation run: 2~3X slower but this perturbation is OK, because threading in Android is not that time sensitive
4. 1 profile per file, dumped at application exit.



FDO Performance

- Global hotness for ARM (HOT_BB_COUNT_FRACTION, Branch prediction routine for the GNU compiler, gcc-4.4.x/gcc/predict.c)
  - 1% improvement on Android's skia library as belows.
  - smaller effects on smaller Android benchmarks.

(unit: bytes)

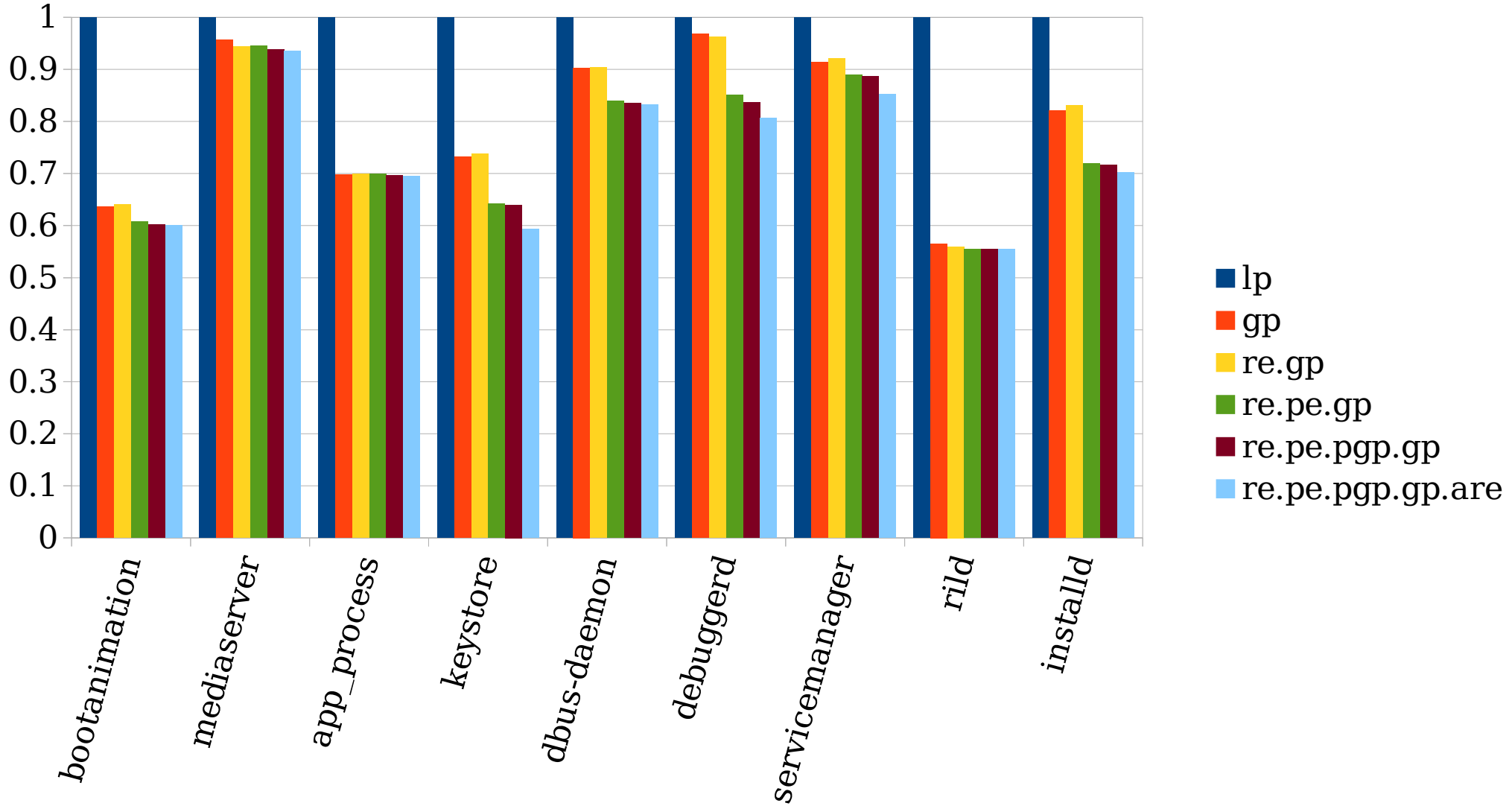
Content Work	default	fdo-default	fdo-modified
Size of libskia	7,879,646	7,396,032	7,319,668
Size reduction	0.00%	6.14%	7.11%
Stdev (over 100 runs)	0.28	0.63	0.26
Speedup	1	0.98	0.97

Source: Smaller and Faster Android, Shih-wei Liao, Google Inc.

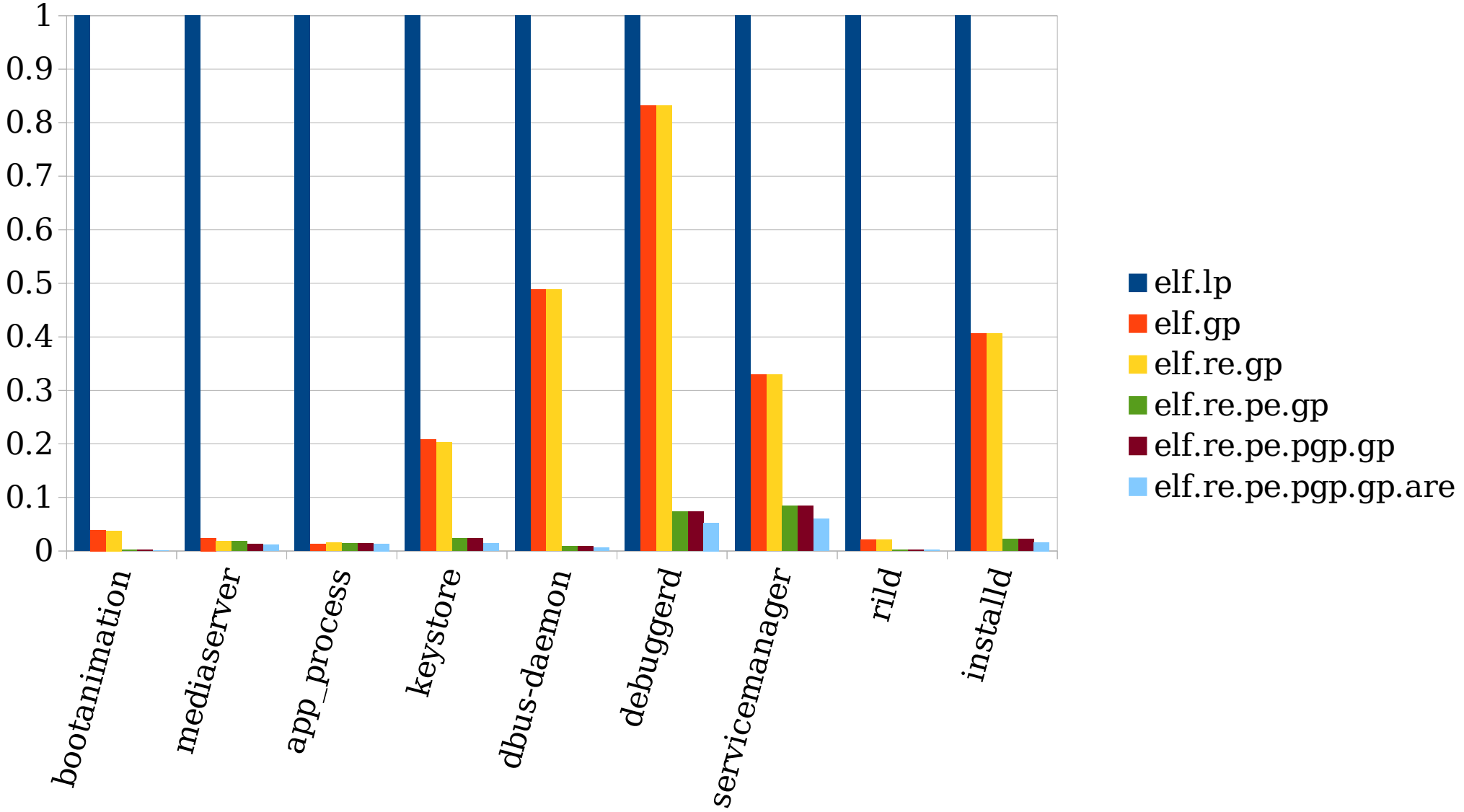
Dynamic Linker Optimizations

- Why?
  - The major reason to optimize dynamic linker is to speed up application startup time.
- How?
  - Implement GNU style hash support for bionic linker
  - Prelinker improvements: incremental global prelinking

(normalized) Dynamic Link time



(normalized) Symbol Lookup number

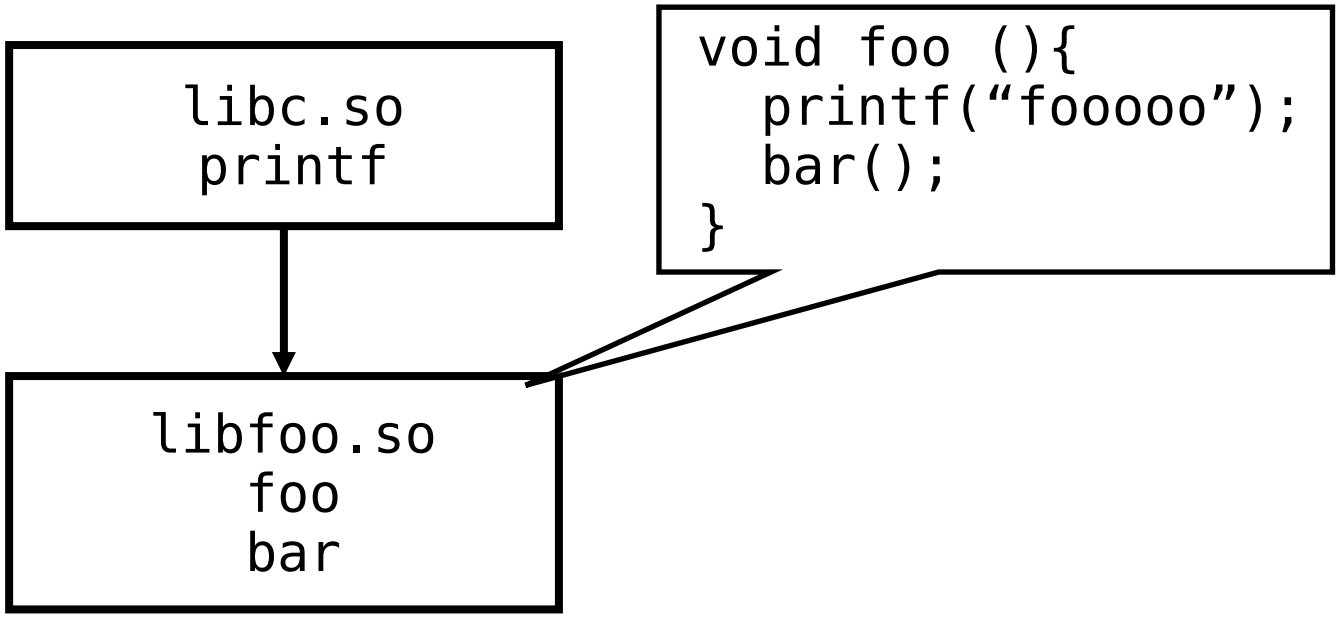


DT_GNU_HASH: visible dynamic linking improvement =

Better hash function (few collisions)

+ Drop unnecessary entry from hash

+ Bloom filter



libfoo.so

DT_GNU_HASH	DT_HASH
foo	foo
bar	bar
	printf

GNU-style Hash

SysV ELF hash

name	2sym collision #	3sym collision #	3+sym collision #
sysv	1749	5	
libiberty	42		
dcache	37		
djb2	29		
sdbm	39		
djb3	31		
rot	337	39	61
sax	34		
fnv	40		
oat	30		

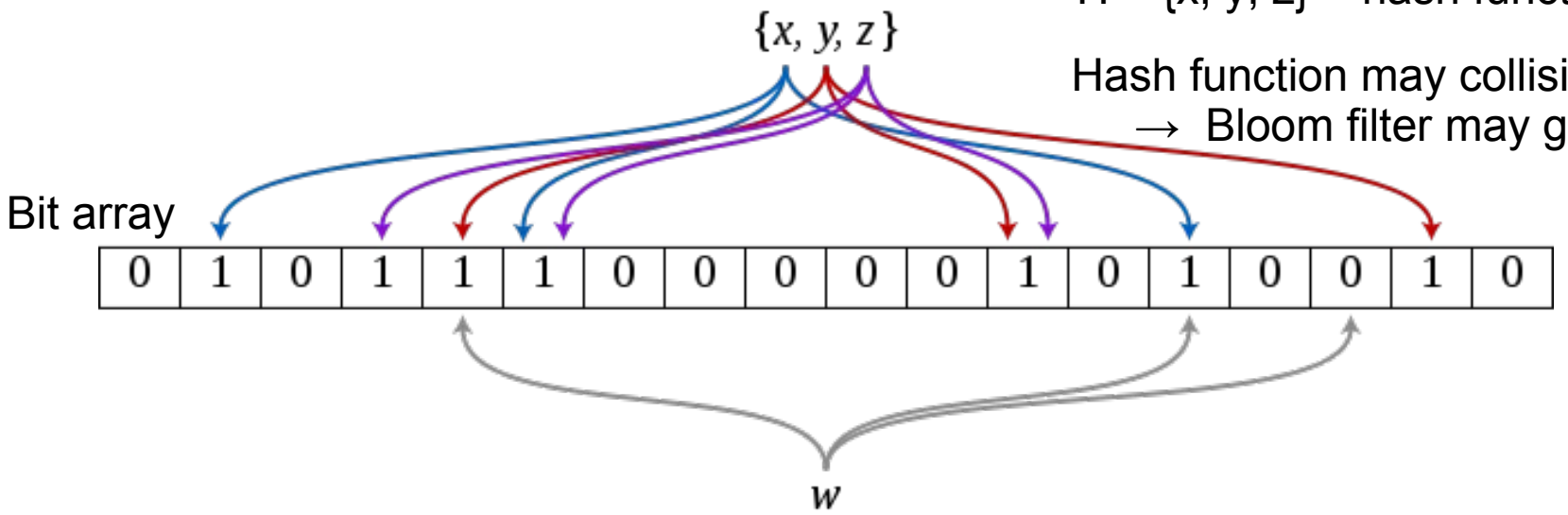
GNU-style hash

Experiment by Jakub Jelinek
http://cygwin.com/ml/libc-alpha/2006-06/msg00098.html

	Symbols in ELF	lookup #	fail#	gnu hash	filtered by bloom
gnu.gp	3758	23702	19950	23310	18234(78%)
gnu.gp.re	3758	20544	16792	19604	14752(75%)
gnu.lp	61750	460996	399252	450074	345032(76%)
gnu.lp.re	61750	481626	419882	448492	342378(76%)

H = {x, y, z} = hash functions

Hash function may collision
→ Bloom filter may got false positives



NOTE: Android 4.0 removes the support of prelinker, but gnu style hash is still useful.

Action Items

TODO

- Use eglibc-like Option Group
  - based on POSIX.1-2001 specifications
- Comply with relevant standards
- Resolve external toolchain issues
  - Allow arm-none-eabi- and arm-none-linux-gnueabi-
  - Don't depend on prebuilt toolchain anymore
- Collaboration: crosstool-ng, buildroot, yocto, ...
- Validation: improve unit-test, bench, ABI test
- More SoC enhancements
- Extensions
  - BioMP: Migrating OpenMP into Bionic

    http://code.google.com/p/biomp/

Reference

- olibc hosted at GitHub: **`http://olibc.github.com/`**
- Optimizing Android Performance with GCC Compiler, Geunsik Lim
- Embedded-Appropriate Crash Handling in Linux, Tim Bird
- Smaller and Faster Android, Shih-wei Liao, Google Inc.

http://0xlab.org