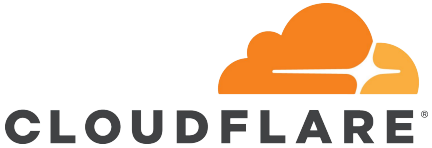


The definitive guide to make software fail *on ARM64*

Ignat Korchagin

@secumod

$ whoami

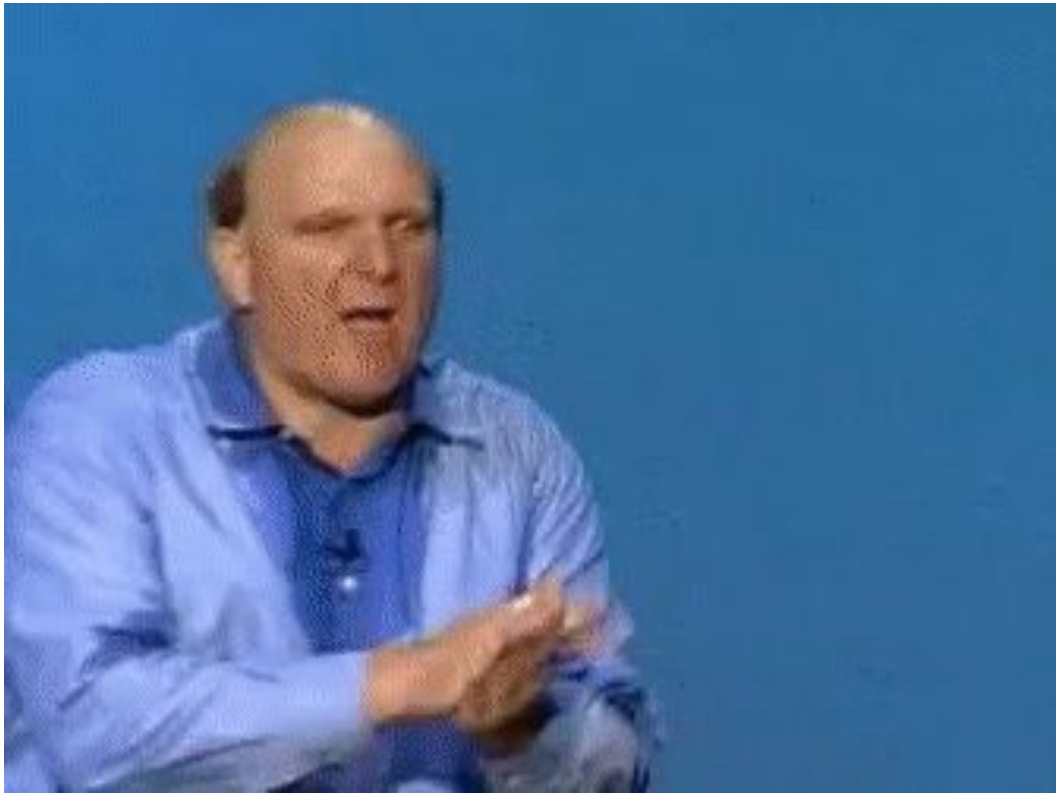
- Performance and security at Cloudflare
- Passionate about security and crypto
- Enjoy low level programming

Initial ARM64 integration

Initial integration in the DC

CLOUDFLARE

Consider your developers



CLOUDFLARE

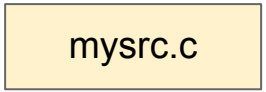
Building packages for ARM64

production arch != developer arch

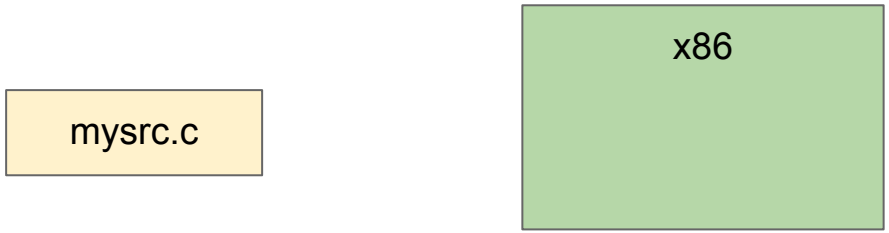
Building packages for ARM64

CLOUDFLARE

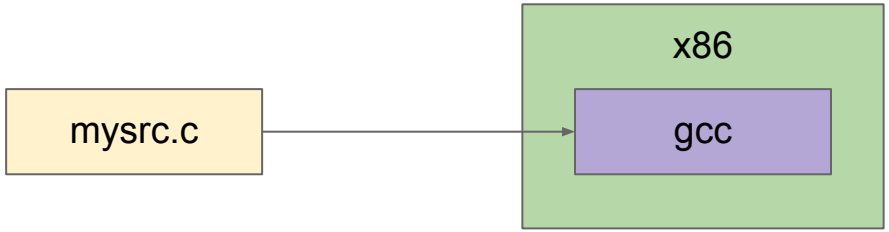
Compiler



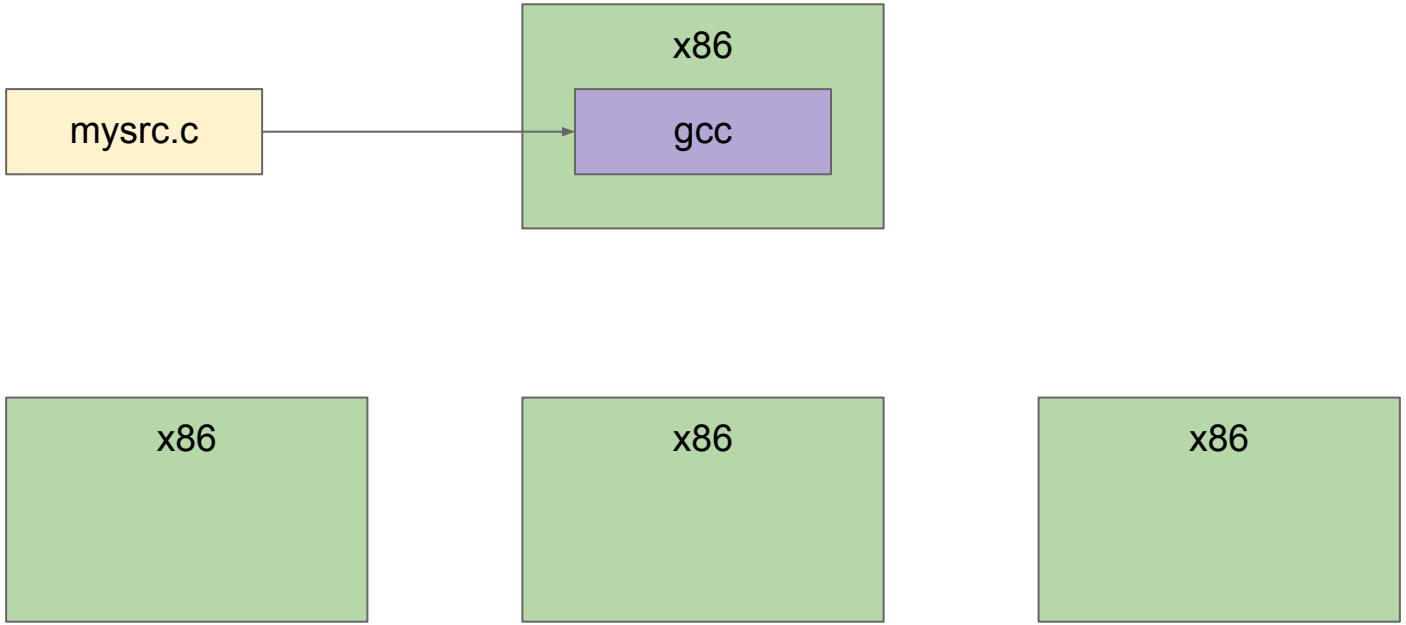
Compiler



Compiler

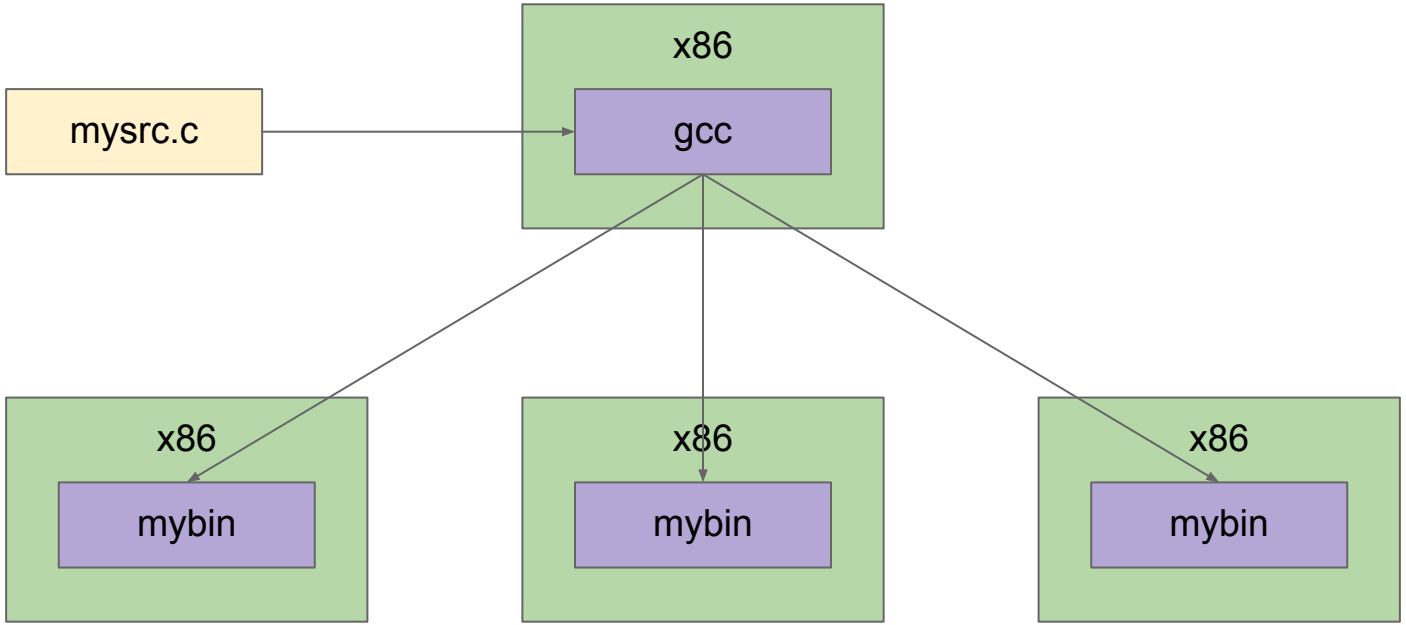


Compiler



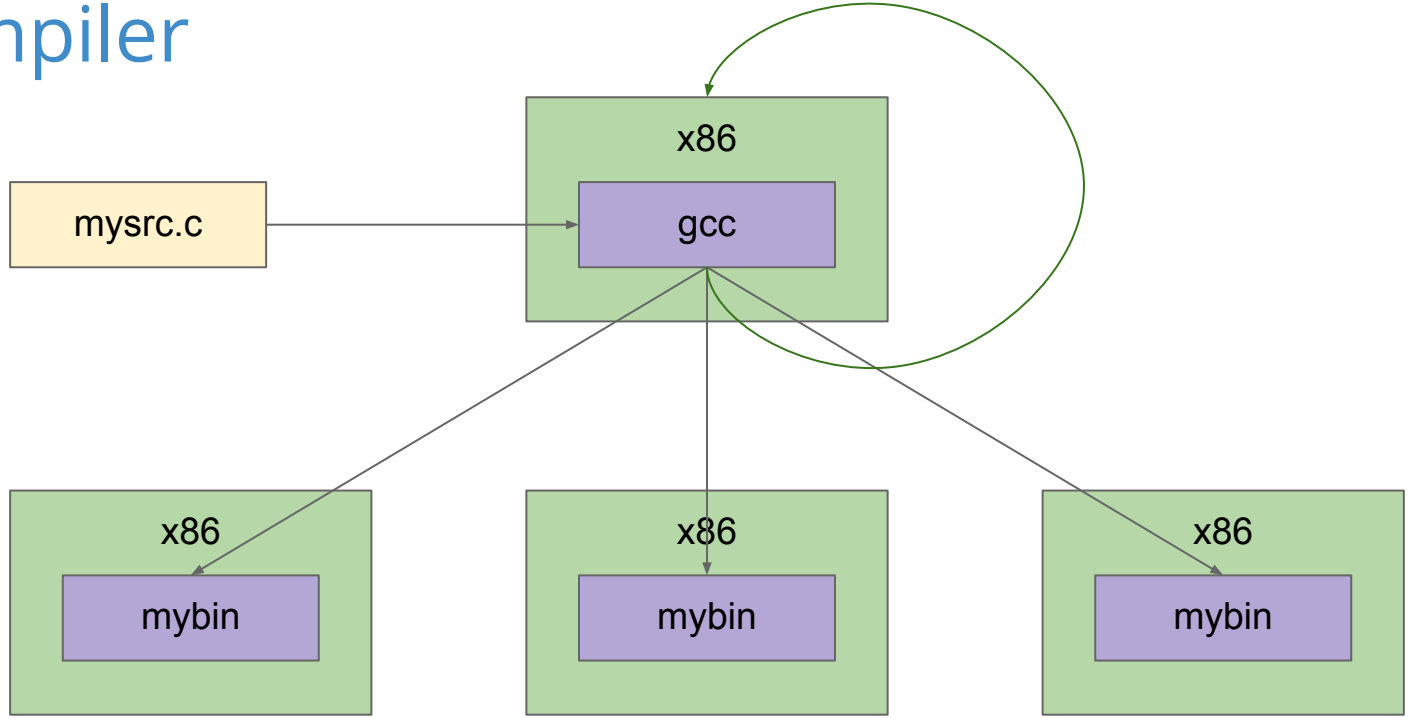
CLOUDFLARE

Compiler



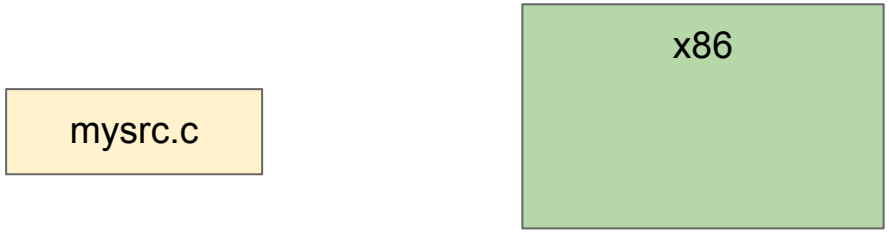
CLOUDFLARE

Compiler

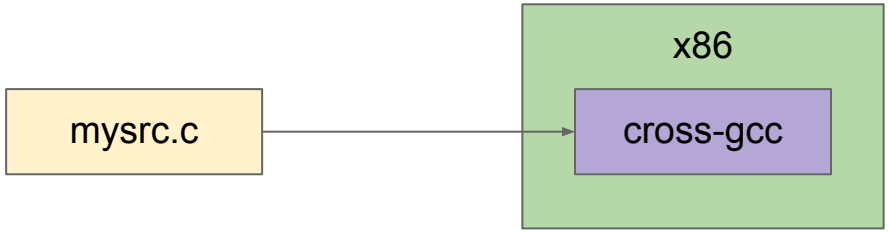


CLOUDFLARE

Cross-compiler

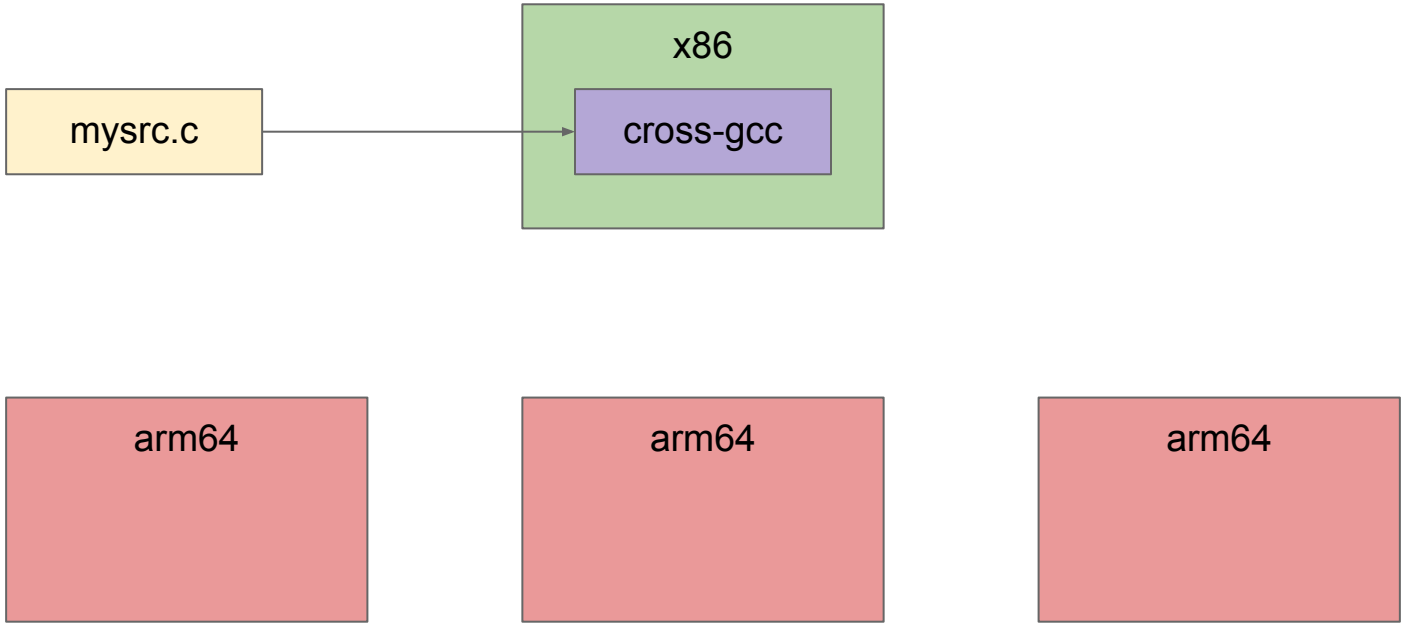


Cross-compiler



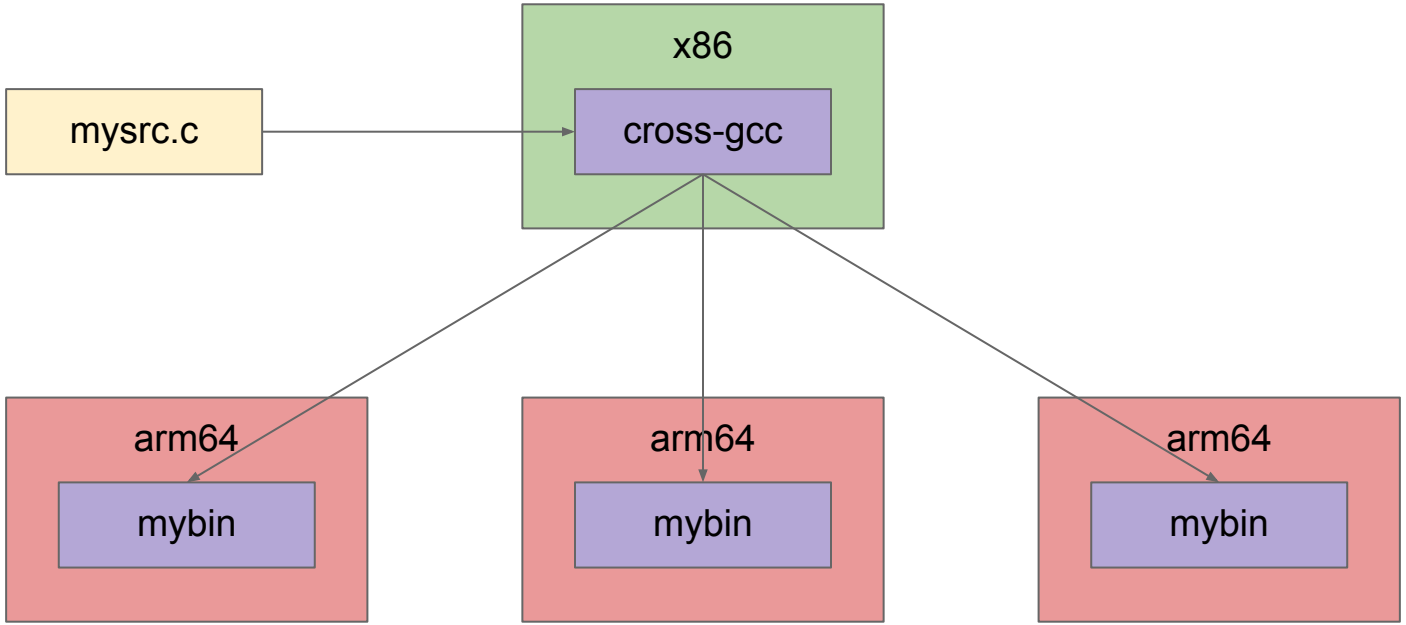
CLOUDFLARE

Cross-compiler



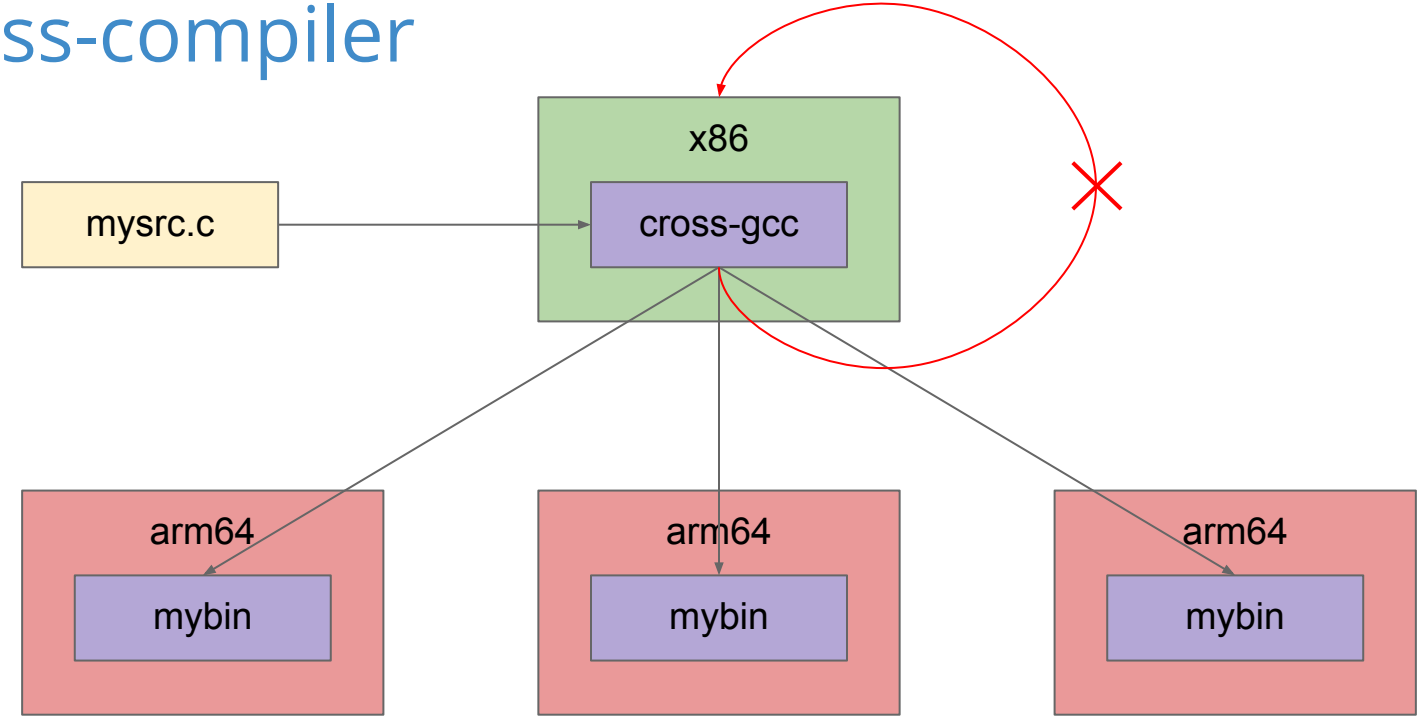
CLOUDFLARE

Cross-compiler



CLOUDFLARE

Cross-compiler



CLOUDFLARE

Cross-compiler terminology

- host - architecture, where the compiler runs

Cross-compiler terminology

- host - architecture, where the compiler runs
- target - architecture, for which the compiler generates machine code

Cross-compiler terminology

- host - architecture, where the compiler runs
- target - architecture, for which the compiler generates machine code
- when host == target, it is “native” compilation
  - subset of a more general cross-compilation

cross-compiling example

```
ignat@dev:~$ gcc -static -o mybin mysrc.c
```

cross-compiling example

```
ignat@dev:~$ gcc -static -o mybin mysrc.c
ignat@dev:~$ readelf -h mybin | grep -i machine
  Machine:                           Advanced Micro Devices X86-64
```

CLOUDFLARE

cross-compiling example

```
ignat@dev:~$ gcc -static -o mybin mysrc.c
ignat@dev:~$ readelf -h mybin | grep -i machine
  Machine:                           Advanced Micro Devices X86-64
ignat@dev:~$ ./mybin
Hello, world!
```

CLOUDFLARE

cross-compiling example

```
ignat@dev:~$ gcc -static -o mybin mysrc.c
ignat@dev:~$ readelf -h mybin | grep -i machine
  Machine:                           Advanced Micro Devices X86-64
ignat@dev:~$ ./mybin
Hello, world!
ignat@dev:~$ aarch64-linux-gnu-gcc -static -o mybin mysrc.c
```

CLOUDFLARE

cross-compiling example

```
ignat@dev:~$ gcc -static -o mybin mysrc.c
ignat@dev:~$ readelf -h mybin | grep -i machine
  Machine:                           Advanced Micro Devices X86-64
ignat@dev:~$ ./mybin
Hello, world!
ignat@dev:~$ aarch64-linux-gnu-gcc -static -o mybin mysrc.c
ignat@dev:~$ readelf -h mybin | grep -i machine
  Machine:                           AArch64
```

CLOUDFLARE

cross-compiling example

```
ignat@dev:~$ gcc -static -o mybin mysrc.c
ignat@dev:~$ readelf -h mybin | grep -i machine
  Machine:                           Advanced Micro Devices X86-64
ignat@dev:~$ ./mybin
Hello, world!
ignat@dev:~$ aarch64-linux-gnu-gcc -static -o mybin mysrc.c
ignat@dev:~$ readelf -h mybin | grep -i machine
  Machine:                           AArch64
ignat@dev:~$ ./mybin
bash: ./mybin: cannot execute binary file: Exec format error
```

Compile time problems

Common misconception

CLOUDFLARE

CC: hardcoded architecture-specific flags

Symptom:

- broken both native and cross builds

CC: hardcoded architecture-specific flags

Symptom:

- broken both native and cross builds
- `gcc: error: unrecognized command line option '-msse2'`

CLOUDFLARE

CC: hardcoded architecture-specific flags

Symptom:

- broken both native and cross builds
- `gcc: error: unrecognized command line option '-msse2'`

Cause:

- hardcoded architecture-specific flags in the build system

CLOUDFLARE

CC: hardcoded architecture-specific flags

Symptom:

- broken both native and cross builds
- `gcc: error: unrecognized command line option '-msse2'`

Cause:

- hardcoded architecture-specific flags in the build system
- `CFLAGS := … -msse2 …` or `CFLAGS += -msee2 …`

CLOUDFLARE

CC: hardcoded architecture-specific flags

Developers:

- put architecture-specific flags in a separate variable, one for each architecture

CC: hardcoded architecture-specific flags

Developers:

- put architecture-specific flags in a separate variable, one for each architecture

```
# Makefile
TARGET_ARCH := … # somehow identify the target architecture
CFLAGS_x86_64 := -msse2 …
CFLAGS_aarch64 := -mabi=lp64 …
TARGET_CFLAGS += CFLAGS_$(TARGET_ARCH)
```

CC: no separation between host and target flags

Symptom:

- broken cross build

CC: no separation between host and target flags

Symptom:

- broken cross build
- usually happens, when the compiler output needs additional post-processing (ex. format conversion)

CC: no separation between host and target flags

Symptom:

- broken cross build
- usually happens, when the compiler output needs additional post-processing (ex. format conversion)
- post-processing tool source is part of the project

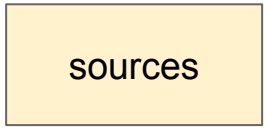
CC: no separation between host and target flags

Symptom:

- broken cross build
- usually happens, when the compiler output needs additional post-processing (ex. format conversion)
- post-processing tool source is part of the project
- `gcc: error: unrecognized command line option '-msse2'`

CLOUDFLARE

CC: no separation between host and target flags

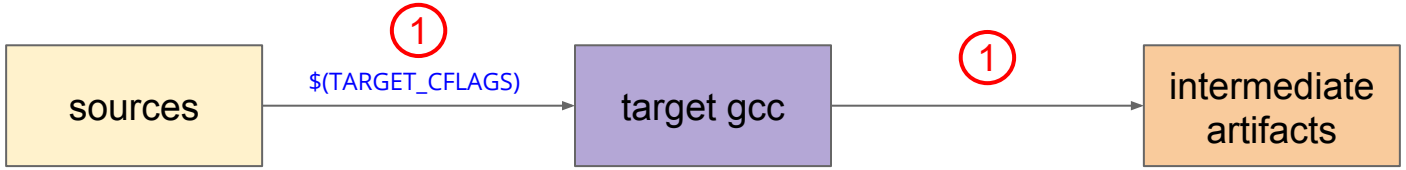


CLOUDFLARE

CC: no separation between host and target flags

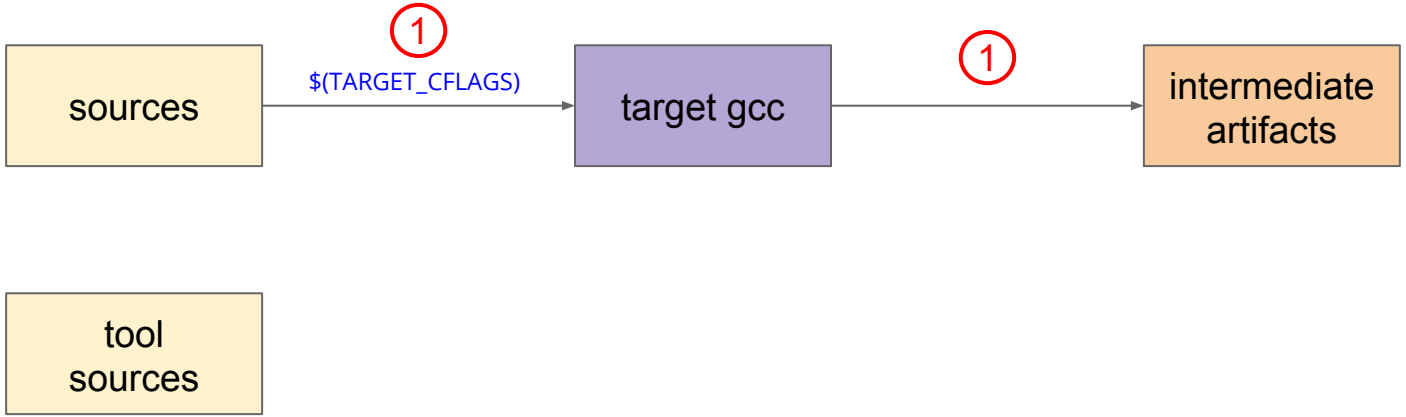
CLOUDFLARE

CC: no separation between host and target flags



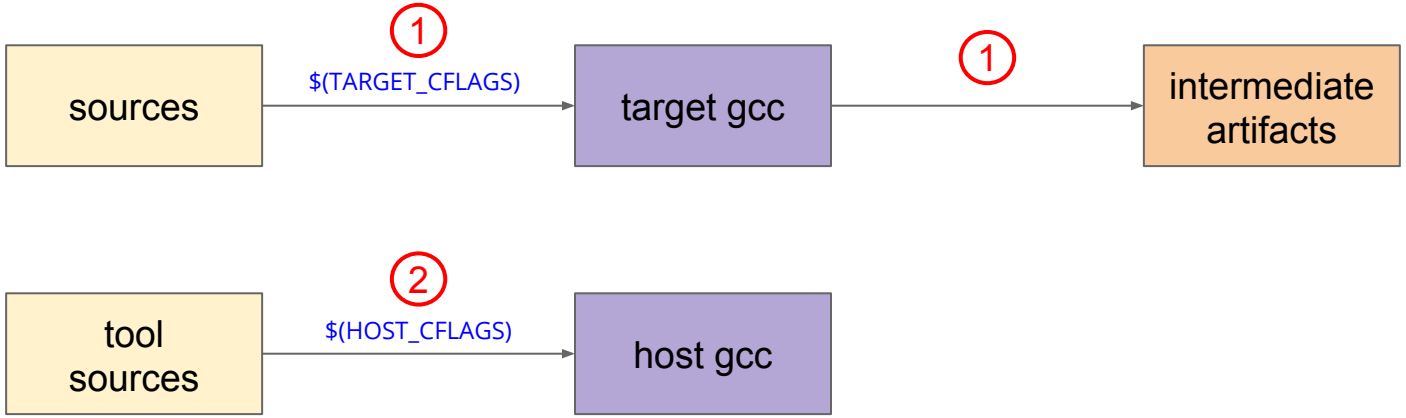
CLOUDFLARE

CC: no separation between host and target flags



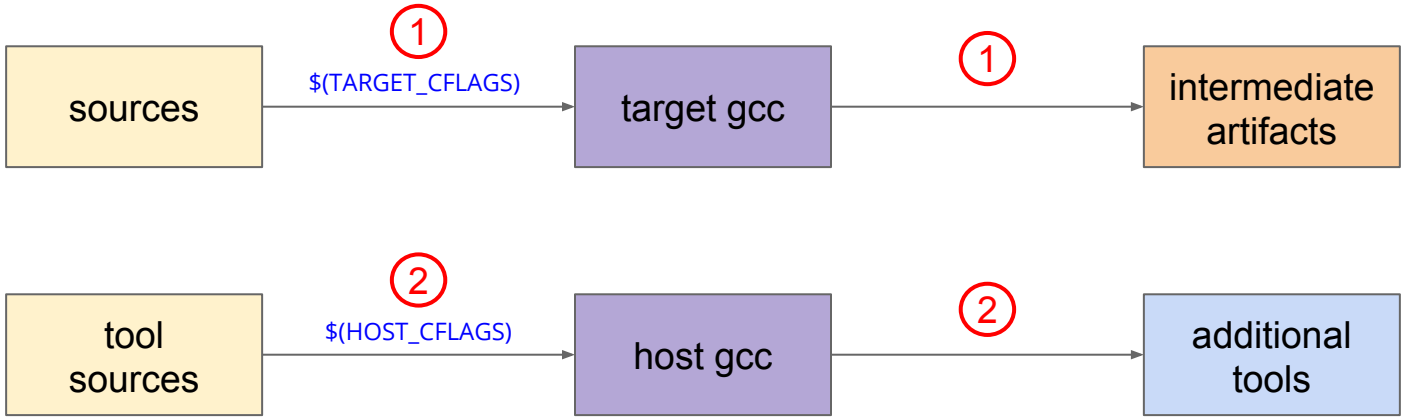
CLOUDFLARE

CC: no separation between host and target flags



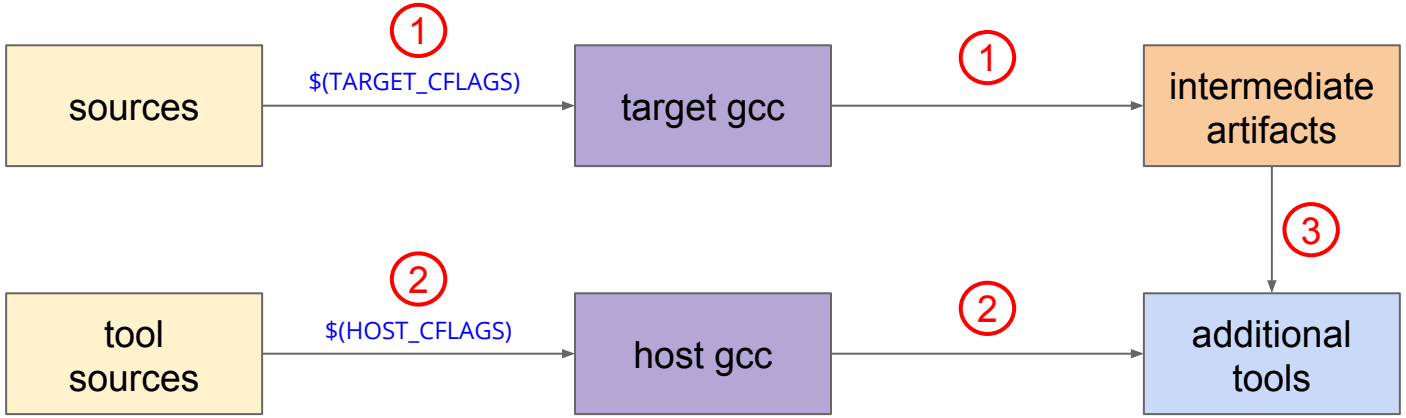
CLOUDFLARE

CC: no separation between host and target flags



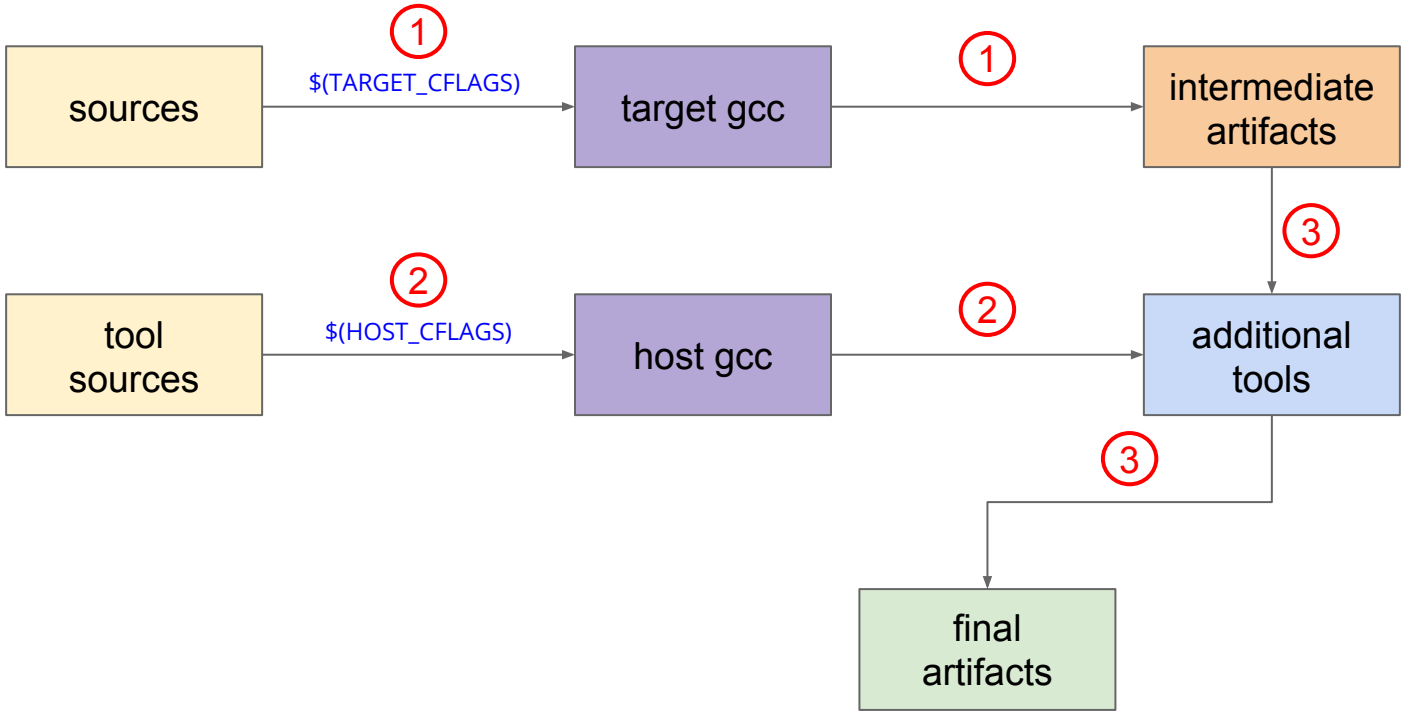
CLOUDFLARE

CC: no separation between host and target flags



CLOUDFLARE

CC: no separation between host and target flags



CLOUDFLARE

CC: no separation between host and target flags

Cause:

- `$(CFLAGS)` use instead of `$(TARGET_CFLAGS)` and `$(HOST_CFLAGS)`

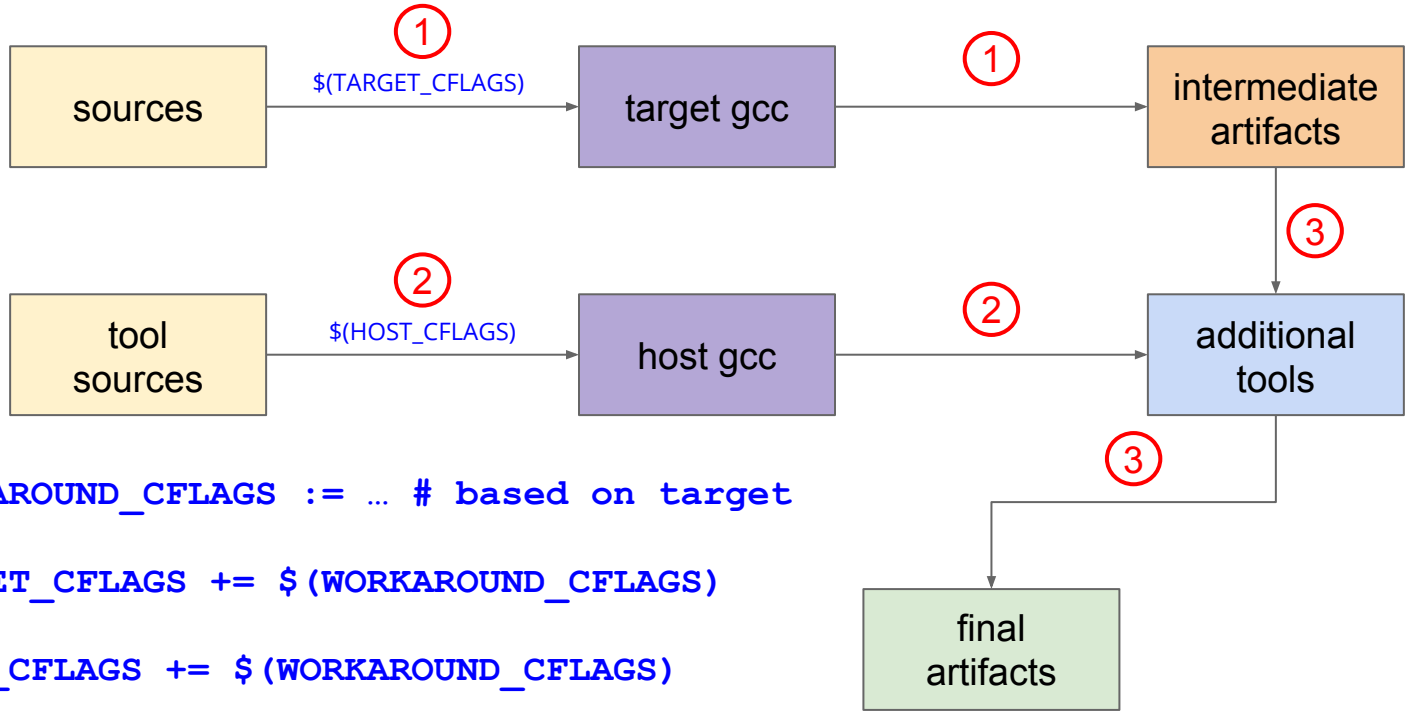
CC: no separation between host and target flags

Cause:

- `$(CFLAGS)` use instead of `$(TARGET_CFLAGS)` and `$(HOST_CFLAGS)`
- use of some `$(ADDITIONAL_CFLAGS)` which are based either only on the target or the host
  - see the usage of `$(WORKAROUND_CFLAGS)` in the iPXE build system: https://github.com/ipxe/ipxe

CLOUDFLARE

CC: no separation between host and target flags




```
WORKAROUND_CFLAGS := … # based on target
...
TARGET_CFLAGS += $(WORKAROUND_CFLAGS)
...
HOST_CFLAGS += $(WORKAROUND_CFLAGS)
```

CLOUDFLARE

CC: no separation between host and target flags

Developers:

- put architecture-specific flags in a separate variable, one for each architecture

CC: no separation between host and target flags

Developers:

- put architecture-specific flags in a separate variable, one for each architecture
- always prefix any compiler/linker options with `TARGET_` or `HOST_`
  - ~~`$(WORKAROUND_CFLAGS)`~~, `$(TARGET_WORKAROUND_CFLAGS)` and `$(HOST_WORKAROUND_CFLAGS)`
  - use `$(COMMON_CFLAGS)` if needed

CLOUDFLARE

CC: no separation between host and target flags

DevOps:

- provide the tools/support to test cross-compilation in the CI
  - x86 to arm64 is generally a good start

CC: no separation between host and target flags

DevOps:

- provide the tools/support to test cross-compilation in the CI
  - x86 to arm64 is generally a good start
- lint project build systems for non-prefixed variable definitions

CLOUDFLARE

Slower build times - it’s a feature!

CLOUDFLARE

CC: reuse of host binaries in target artifacts

Symptom:

- broken artifacts

CC: reuse of host binaries in target artifacts

Symptom:

- broken artifacts
- usually happens, when the compiler output needs additional post-processing (ex. format conversion)

CC: reuse of host binaries in target artifacts

Symptom:

- broken artifacts
- usually happens, when the compiler output needs additional post-processing (ex. format conversion)
- post-processing tool source is part of the project

CC: reuse of host binaries in target artifacts

Symptom:

- broken artifacts
- usually happens, when the compiler output needs additional post-processing (ex. format conversion)
- post-processing tool source is part of the project
- post-processing tool is also released as an artifact

CC: reuse of host binaries in target artifacts

Symptom:

- broken artifacts
- usually happens, when the compiler output needs additional post-processing (ex. format conversion)
- post-processing tool source is part of the project
- post-processing tool is also released as an artifact
- `./fixdep: cannot execute binary file: Exec format error`

CLOUDFLARE

CC: reuse of host binaries in target artifacts

Cause:

- incorrect usage of `$(HOST_CC)` vs `$(TARGET_CC)`

CLOUDFLARE

CC: reuse of host binaries in target artifacts

Cause:

- incorrect usage of `$(HOST_CC)` vs `$(TARGET_CC)`
- incorrect build dependency declaration
  - “make” may consider the dependency, built with `$(HOST_CC)` already satisfied, when doing the target build and not rebuild it with `$(TARGET_CC)`

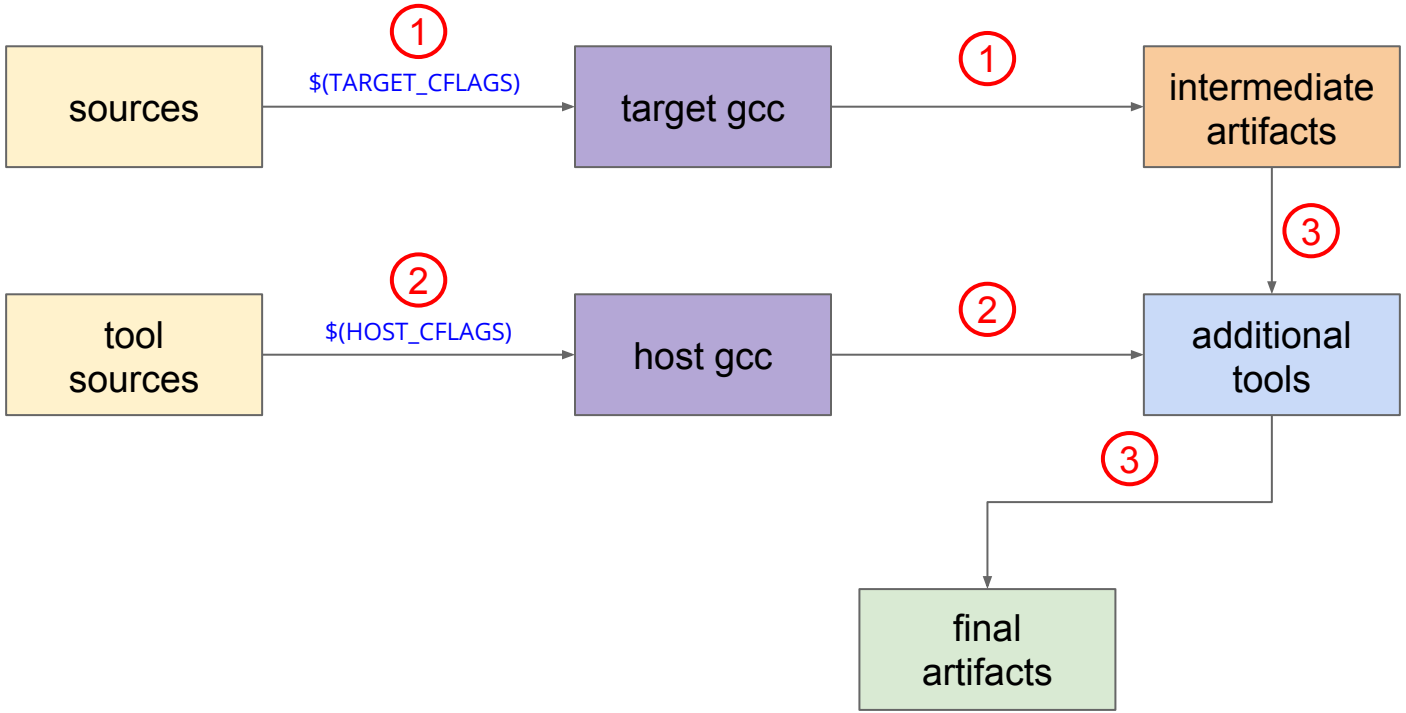
CLOUDFLARE

CC: reuse of host binaries in target artifacts

Cause:

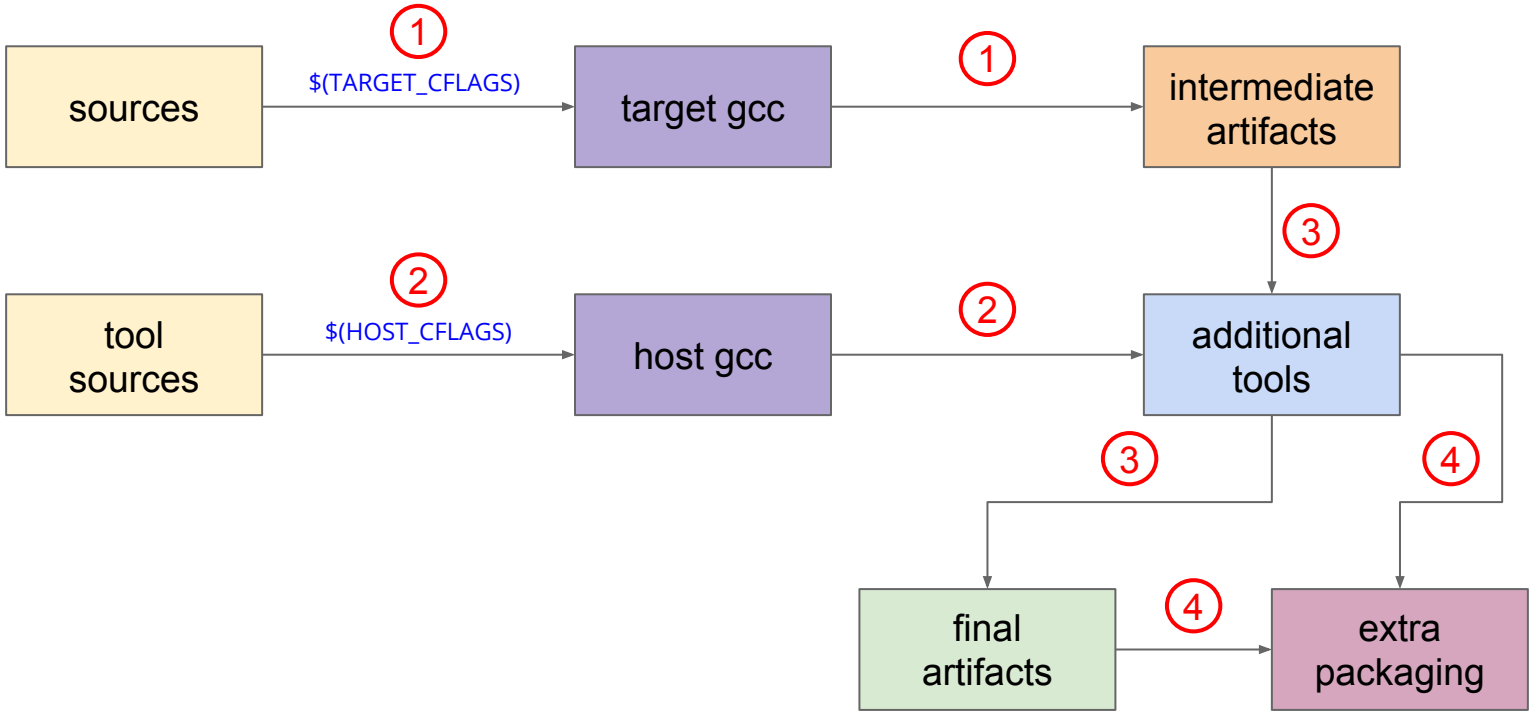
- incorrect usage of `$(HOST_CC)` vs `$(TARGET_CC)`
- incorrect build dependency declaration
  - “make” may consider the dependency, built with `$(HOST_CC)` already satisfied, when doing the target build and not rebuild it with `$(TARGET_CC)`
- example: vanilla Linux kernel Debian packaging
  - broken “linux-headers” .deb package when cross-compiling

CC: reuse of host binaries in target artifacts



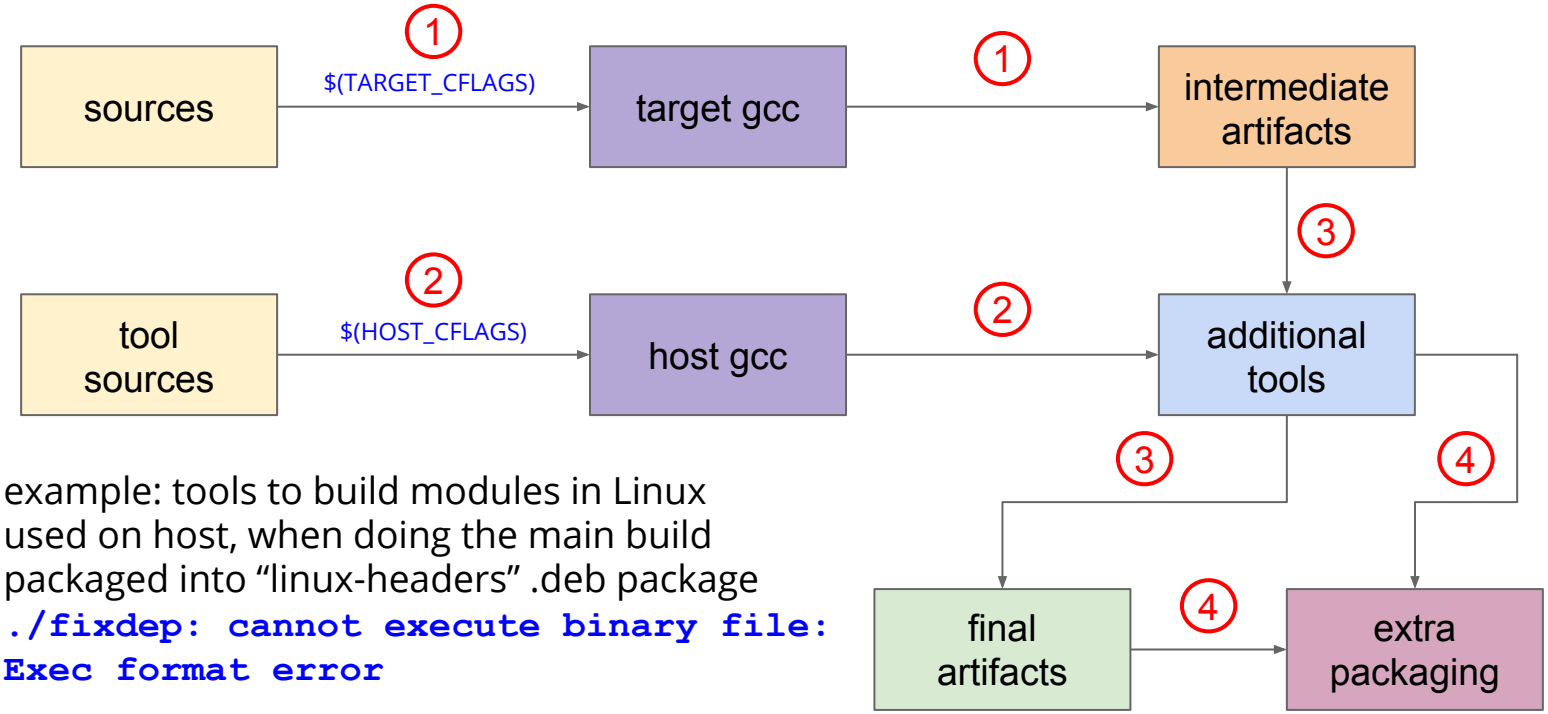
CLOUDFLARE

CC: reuse of host binaries in target artifacts



CLOUDFLARE

CC: reuse of host binaries in target artifacts



- example: tools to build modules in Linux
- used on host, when doing the main build
- packaged into “linux-headers” .deb package
- `./fixdep: cannot execute binary file: Exec format error`

CLOUDFLARE

CC: reuse of host binaries in target artifacts

Developers:

- ensure all target artifacts are processed with `$(TARGET_CC)`

CLOUDFLARE

CC: reuse of host binaries in target artifacts

Developers:

- ensure all target artifacts are processed with `$(TARGET_CC)`
- put host and target output in different directories
  - clearly shows which artifacts are not compiled either for host or target architecture
  - ensures “make” does not consider target dependency satisfied, if only the host version was built, because of different filesystem paths

CLOUDFLARE

CC: reuse of host binaries in target artifacts

DevOps:

- provide the tools/support to test cross-compilation in the CI
  - x86 to arm64 is generally a good start

CC: reuse of host binaries in target artifacts

DevOps:

- provide the tools/support to test cross-compilation in the CI
  - x86 to arm64 is generally a good start
- inspect the final artifacts for anomalies
  - for example, there should be no x86 executables in the arm64 .deb package

CLOUDFLARE

Runtime problems

Out of memory with plenty of memory

Symptom:

- the process complains about not being able to allocate memory

Out of memory with plenty of memory

Symptom:

- the process complains about not being able to allocate memory
- there is plenty of free memory in the system

Out of memory with plenty of memory

Symptom:

- the process complains about not being able to allocate memory
- there is plenty of free memory in the system
- the process is using mmap syscall for file I/O
  - most database workloads

Out of memory with plenty of memory

Symptom:

- the process complains about not being able to allocate memory
- there is plenty of free memory in the system
- the process is using mmap syscall for file I/O
  - most database workloads
- `ENOMEM: Cannot allocate memory`

CLOUDFLARE

32-bit vs 64-bit

- 32-bit allows to address only up to 4GB

32-bit vs 64-bit

- 32-bit allows to address only up to 4GB

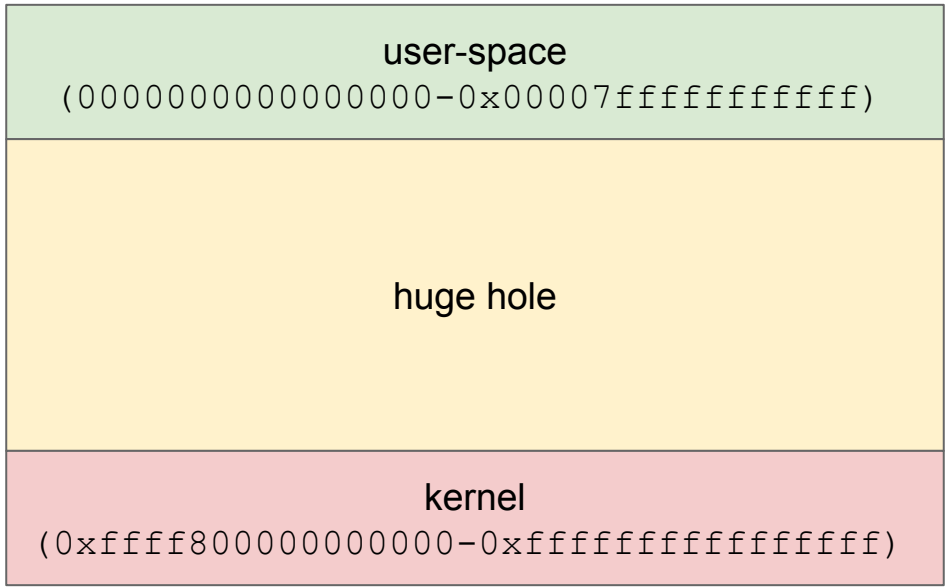
- 64-bit allows to address up to 17179869184GB
  - or “more than enough...”

CLOUDFLARE

The cake is a lie

CLOUDFLARE

Linux process virtual memory map (x86)



https://www.kernel.org/doc/Documentation/x86/x86_64/mm.txt

CLOUDFLARE

Linux process virtual memory map

- actually you can have only 47-bit addresses in user-space on x86_64
  - so it is only 131072GB compared to promised 17179869184GB

Linux process virtual memory map

- actually you can have only 47-bit addresses in user-space on x86_64
  - so it is only 131072GB compared to promised 17179869184GB
- on arm64 you get only 39-bit addresses if you take Linux defaults
  - only 512GB addressable space

https://www.kernel.org/doc/Documentation/arm64/memory.txt

Linux process virtual memory map

Developers:

- try to avoid using unbounded memory mappings

Linux process virtual memory map

Developers:

- try to avoid using unbounded memory mappings
- try to identify the upper bound of the user-space addressable space and compare to the mapped file size

Linux process virtual memory map

Developers:

- try to avoid using unbounded memory mappings
- try to identify the upper bound of the user-space addressable space and compare to the mapped file size

DevOps:

- make sure to review your second architecture kernel memory layout config
  - you might need to recompile the kernel

Linux process virtual memory map (cont.)

- recompiled the arm64 kernel with 48-bit user-space addresses (256TB space)

Linux process virtual memory map (cont.)

- recompiled the arm64 kernel with 48-bit user-space addresses (256TB space)
- some workloads started to crash randomly

Linux process virtual memory map (cont.)

- recompiled the arm64 kernel with 48-bit user-space addresses (256TB space)
- some workloads started to crash randomly
- traced down to Lua code

LuaJIT lightuserdata

- simple “efficient” C-interface

LuaJIT lightuserdata

- simple “efficient” C-interface
- operates directly on C-pointers

LuaJIT lightuserdata

- simple “efficient” C-interface
- operates directly on C-pointers
- uses (supposedly unused) upper bits of the address to store some metadata
  - `0x00007fffffffffff`

https://github.com/LuaJIT/LuaJIT/blob/f5d424afe8b9395f0df05aba905e0e1f6a2262b8/src/lj_obj.h#L173-L193

CLOUDFLARE

LuaJIT lightuserdata assumptions

```
173  /* Internal object tags.
174  **
175  ** Internal tags overlap the MSW of a number object (must be a double).
176  ** Interpreted as a double these are special NaNs. The FPU only generates
177  ** one type of NaN (0xfff8_0000_0000_0000). So MSWs > 0xfff80000 are available
178  ** for use as internal tags. Small negative numbers are used to shorten the
179  ** encoding of type comparisons (reg/mem against sign-ext. 8 bit immediate).
180  **
181  **                  ---MSW---.---LSW---
182  ** primitive types |  itype  |         |
183  ** lightuserdata   |  itype  |  void * |  (32 bit platforms)
184  ** lightuserdata   |ffff|    void *    |  (64 bit platforms, 47 bit pointers)
185  ** GC objects      |  itype  |  GCRef  |
186  ** int (LJ_DUALNUM)|  itype  |   int   |
187  ** number           -------double------
188  **
189  ** ORDER LJ_T
190  ** Primitive types nil/false/true must be first, lightuserdata next.
191  ** GC objects are at the end, table/userdata must be lowest.
192  ** Also check lj_ir.h for similar ordering constraints.
193  */
```

CLOUDFLARE

Linux process virtual memory

Developers:

- state assumptions in code, not comments
  - check assumptions early and error out with a meaningful error message

Linux process virtual memory

Developers:

- state assumptions in code, not comments
  - check assumptions early and error out with a meaningful error message
- don’t over optimise
  - provide a fallback (less optimal) generic implementation

Linux process virtual memory

Developers:

- state assumptions in code, not comments
  - check assumptions early and error out with a meaningful error message
- don’t over optimise
  - provide a fallback (less optimal) generic implementation

DevOps:

- ditto

CLOUDFLARE

Pagesize

CLOUDFLARE

Pagesize

- a minimum discrete block of volatile memory

Pagesize

- a minimum discrete block of volatile memory
- many database-like workloads try to keep track of allocated pages
  - faster memory access
  - avoid memory fragmentation
  - efficient memory reuse

Pagesize

Symptom:

- the process uses much more memory on secondary architecture

Pagesize

Symptom:

- the process uses much more memory on secondary architecture
- otherwise, working as intended
  - although it depends how aggressive the code is with memory management

Pagesize

Cause:

- the process has hardcoded page size in code

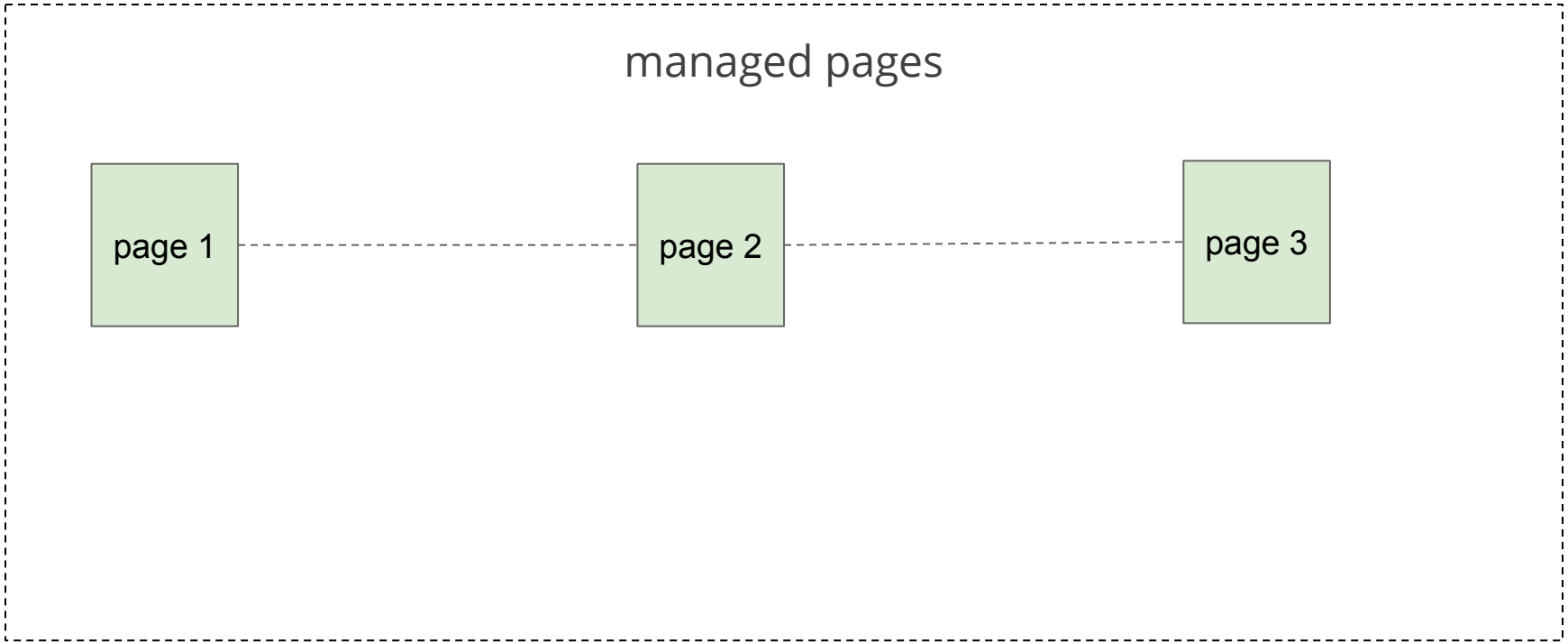
Pagesize

Cause:

- the process has hardcoded page size in code
- the target architecture has a different page size
  - arm64 may have 4k, 16k or 64k pages

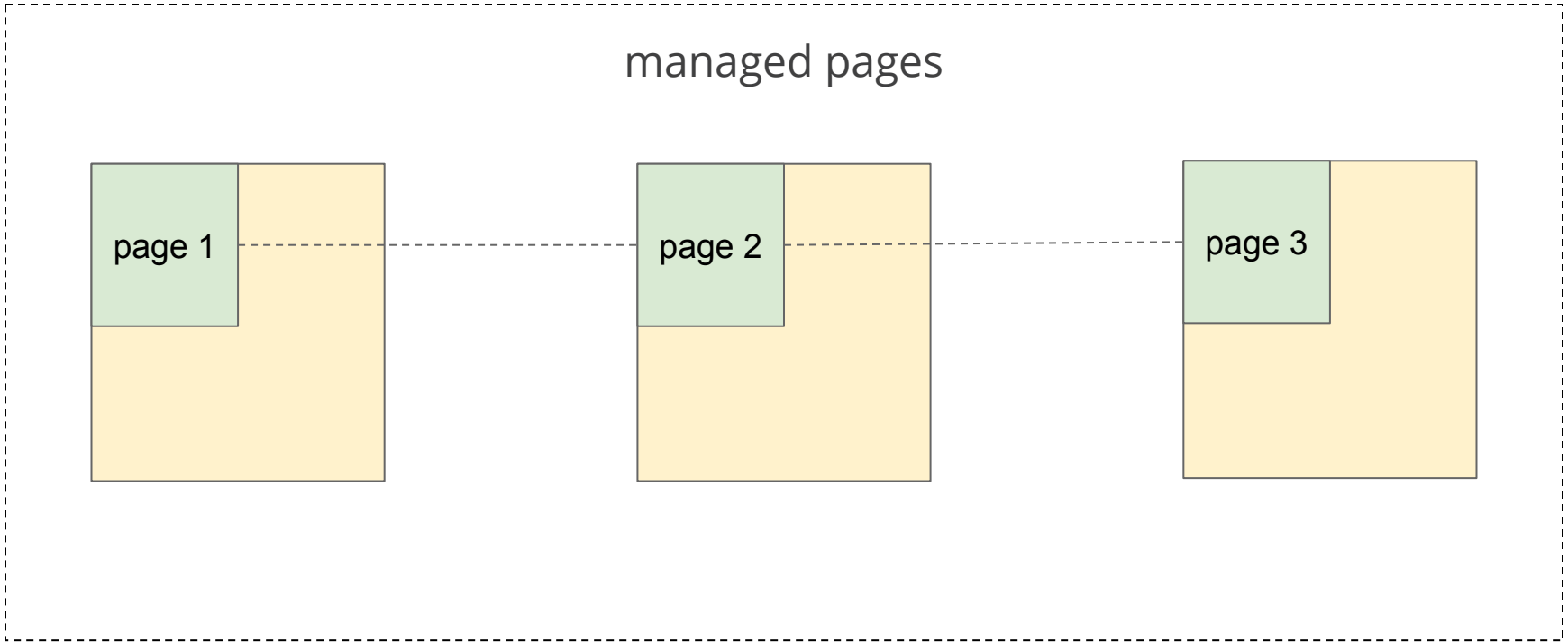
https://www.kernel.org/doc/Documentation/arm64/memory.txt

Pagesize 4k



CLOUDFLARE

Pagesize 16k



CLOUDFLARE

Pagesize

Developers:

- ~~`#define PAGE_SIZE 4096`~~
  - ~14k+ exact matches on GitHub

CLOUDFLARE

Pagesize

Developers:

- ~~`#define PAGE_SIZE 4096`~~
  - ~14k+ exact matches on GitHub
- `long page_size = sysconf(_SC_PAGESIZE);`

CLOUDFLARE

Pagesize

Developers:

- ~~`#define PAGE_SIZE 4096`~~
  - ~14k+ exact matches on GitHub
- `long page_size = sysconf(_SC_PAGESIZE);`

DevOps:

- monitor process memory usage on different architectures

CLOUDFLARE

Filesystem block size

- like pagesize, but for files

Filesystem block size

- like pagesize, but for files
- minimum amount any piece of data can occupy on disk, so determines physical file size
  - even 1 byte file will occupy at least “block” bytes

Filesystem block size

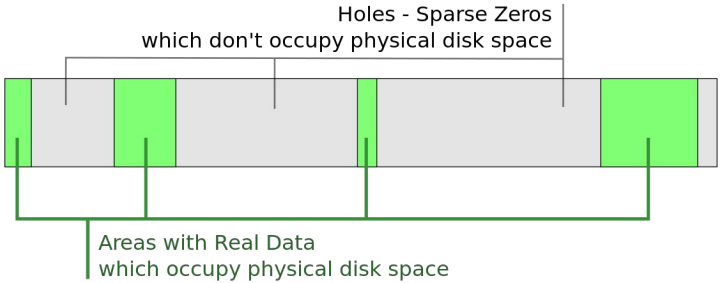
- like pagesize, but for files
- minimum amount any piece of data can occupy on disk, so determines physical file size
  - even 1 byte file will occupy at least “block” bytes
- multiple of the underlying block device block size
  - typical values are 512 bytes or 4k

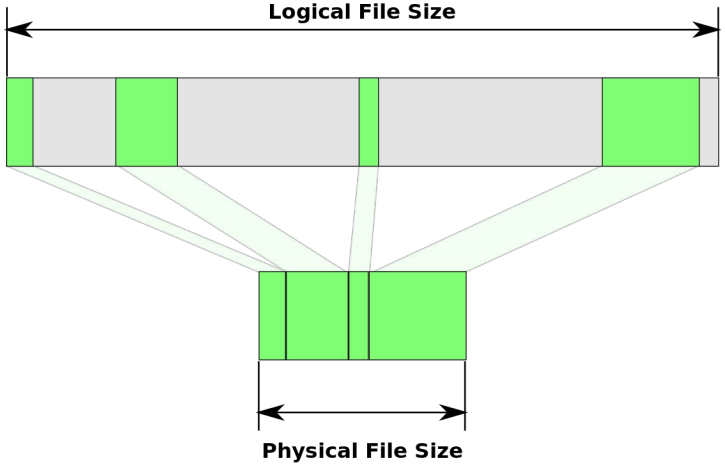
Filesystem block size

- like pagesize, but for files
- minimum amount any piece of data can occupy on disk, so determines physical file size
  - even 1 byte file will occupy at least “block” bytes
- multiple of the underlying block device block size
  - typical values are 512 bytes or 4k
- mostly useful for sparse files

https://en.wikipedia.org/wiki/Sparse_file

Sparse files





CLOUDFLARE

Filesystem block size

Symptom:

- the sparse file test fails on arm64
  - https://github.com/capnproto/capnproto

Filesystem block size

Symptom:

- the sparse file test fails on arm64
  - https://github.com/capnproto/capnproto
- the test fails only, when the test suite is run from tmpfs

Filesystem block size

Cause:

- the process has hardcoded block size in code

Filesystem block size

Cause:

- the process has hardcoded block size in code
- on memory-backed filesystems block size == page size
  - arm64 may have 4k, 16k or 64k pages

https://www.kernel.org/doc/Documentation/arm64/memory.txt

Filesystem block size

Developers:

- ~~`#define BLOCK_SIZE 4096`~~

CLOUDFLARE

Filesystem block size

Developers:

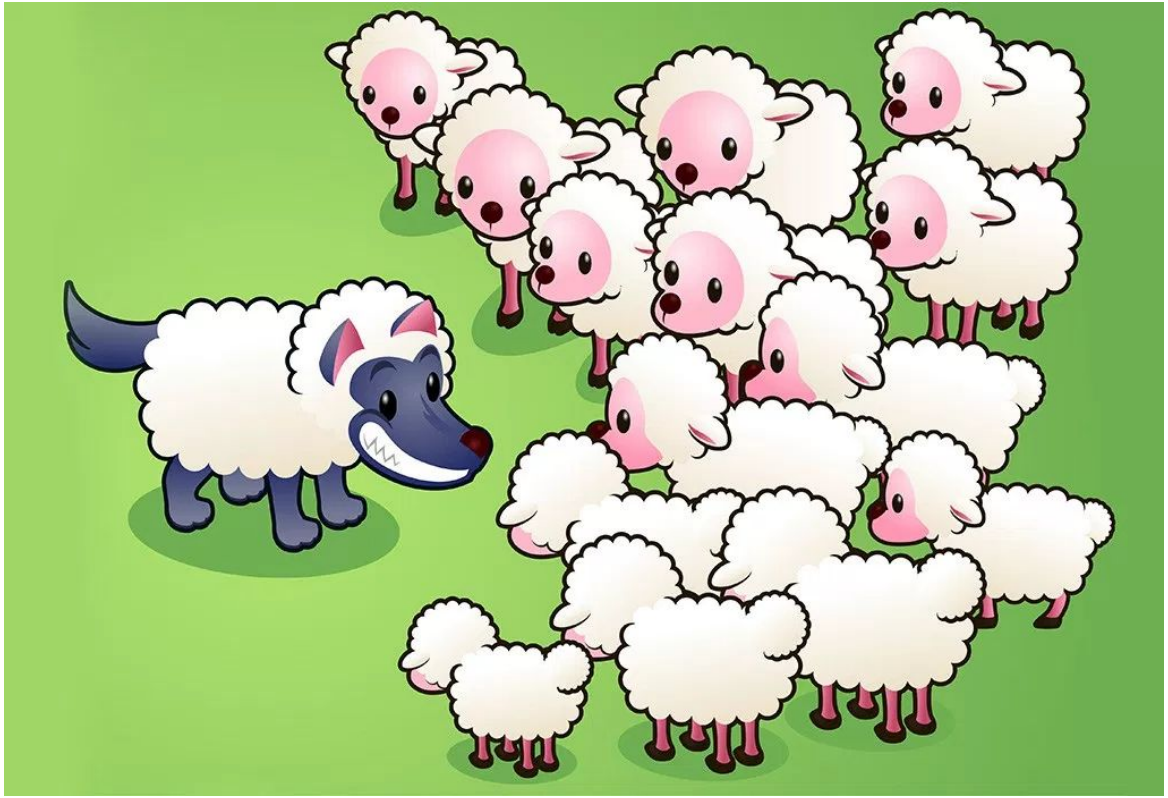
- ~~`#define BLOCK_SIZE 4096`~~
- `stat("/the/file", &stats); blksize_t block_size = stats.st_blksize;`

CLOUDFLARE

Conclusions

- even “portable” code with no assembly can fail in many ways on a different architecture
- for developers:
  - don’t over optimise, provide fallback implementations
  - don’t rely on assumptions and test them in code if you have to
  - provide meaningful error messages
- for devops:
  - ensure the CI environment can test diverse architectures and configurations
  - provide tools/linters to enforce best-practices in code and build scripts

ARM64 in production



CLOUDFLARE

Thank you!