Toolchain

What to Expect?

- W's of a Toolchain
- W's & How's of Cross Toolchain?
- Building a Cross Toolchain
- Testing a Cross Toolchain

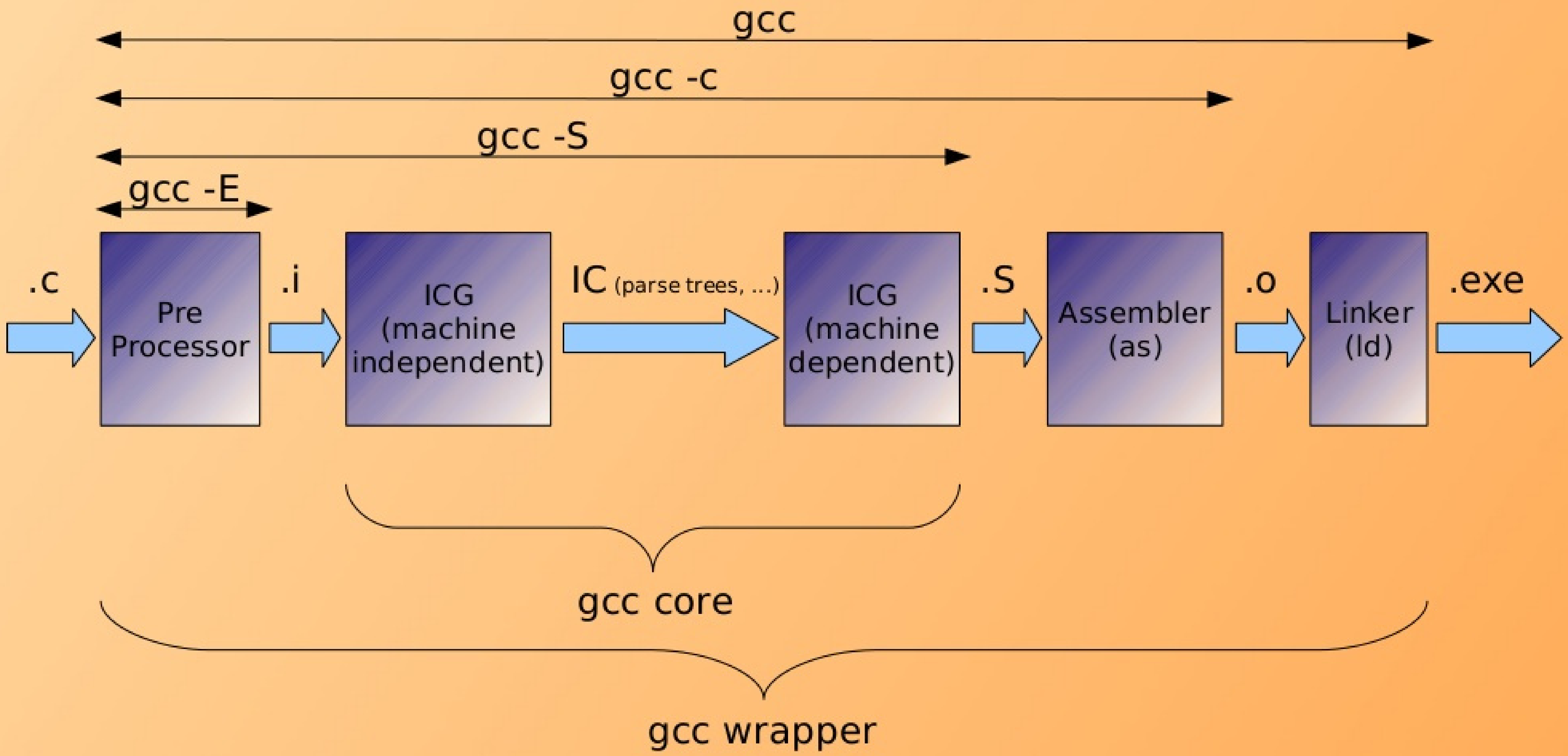
What is a Toolchain?

- Collection of Tools
- In Embedded context
  - Collection of C Compiler & its Friends
- Categorized under 3 umbrella
  - C Compiler (gcc)
  - Set of C Libraries (e.g. glibc, uClibc)
  - Binary Utilities (binutils)

Check on gcc

- How do you do the following?
  - Generate Object Code
  - Generate Assembly Code
  - Generate Pre-processed Code
  - Generate a Shared Library
  - Adding Header Path
  - Adding Library Path
  - Linking a Library
  - Excluding standard includes & libraries
  - Adding a “#define”

gcc Internals



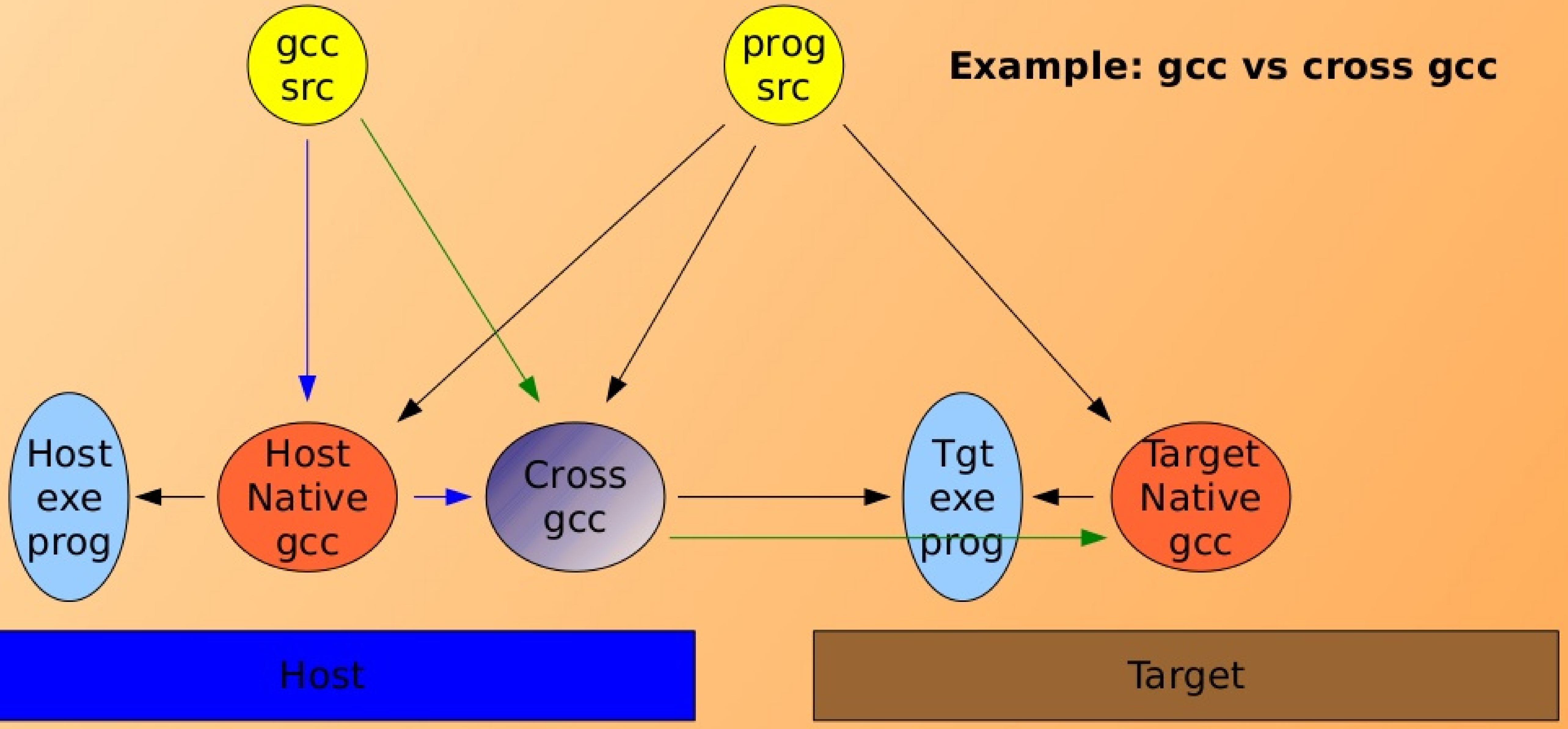
Set of C Libraries

- Generally useful Libraries
  - C, Math, Thread, Socket, ...
- Various Options
  - glibc
    - Complete featured but heavy on memory
    - Highly standard compatible
  - uClibc
    - Light-weight with mostly same features
    - But not that much standards compatible
  - Diet libc
    - Similar to uClibc, just that this has been done from scratch
    - Emphasis on minimizing size & optimizing performance

Binary Utilities

- as – GNU Assembler
- ld – GNU Linker
- gasp – GNU Assembler Pre-processor
- ar – Creates & Manipulates Archives
- nm – Lists the symbols in an Object file
- objcopy – Copies & Translates Object files
- objdump – Displays info about Content of the Object files
- ranlib – Generates an index to the content of Object files
- readelf – Displays info about an ELF format Object file
- size – Lists the sizes of sections within an Object file
- strings – Prints the strings of printable characters in Object files
- strip – Strips symbols from Object files
- c++filt – Converts low-level, mangled assembly labels resulting from overloaded C++ functions to their user-level names
- addr2line – Converts addresses into line numbers within original source files

What is Cross?



What is a Cross Toolchain?

Toolchain which has all “cross” tools

Why we need a Cross Toolchain?

- Embedded Systems are constrained
  - Toolchain demands heavy memory & performance
  - May not always have a console interface
  - Even if there, may be minimal
- Ease of Development
  - Complete accustomed Development Environment on the Host
  - Favourite Editors, GUIs, ...

How to get a Cross Toolchain?

- Get it pre-compiled from vendors
  - Popular: Code sourcery
  - Local: Requirement specific
- Build your own
  - Doing it manually is a complicated process
    - Inter Package version compatibility is the biggest challenge
  - But various automated tools are available today to simplify the process

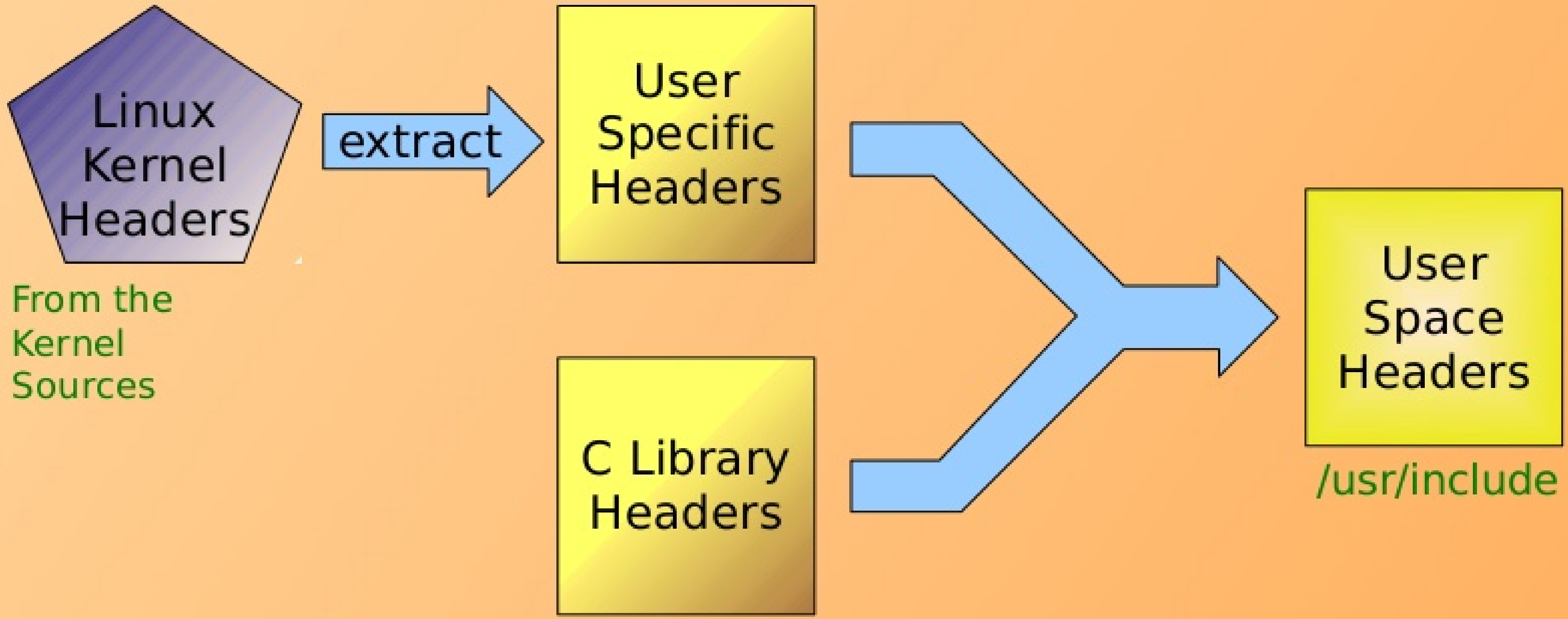
Automated Build Tools

- Crosstool
  - crosstool-ng.org
- Buildroot
  - buildroot.org
- Ptxdist
  - ptxdist.org

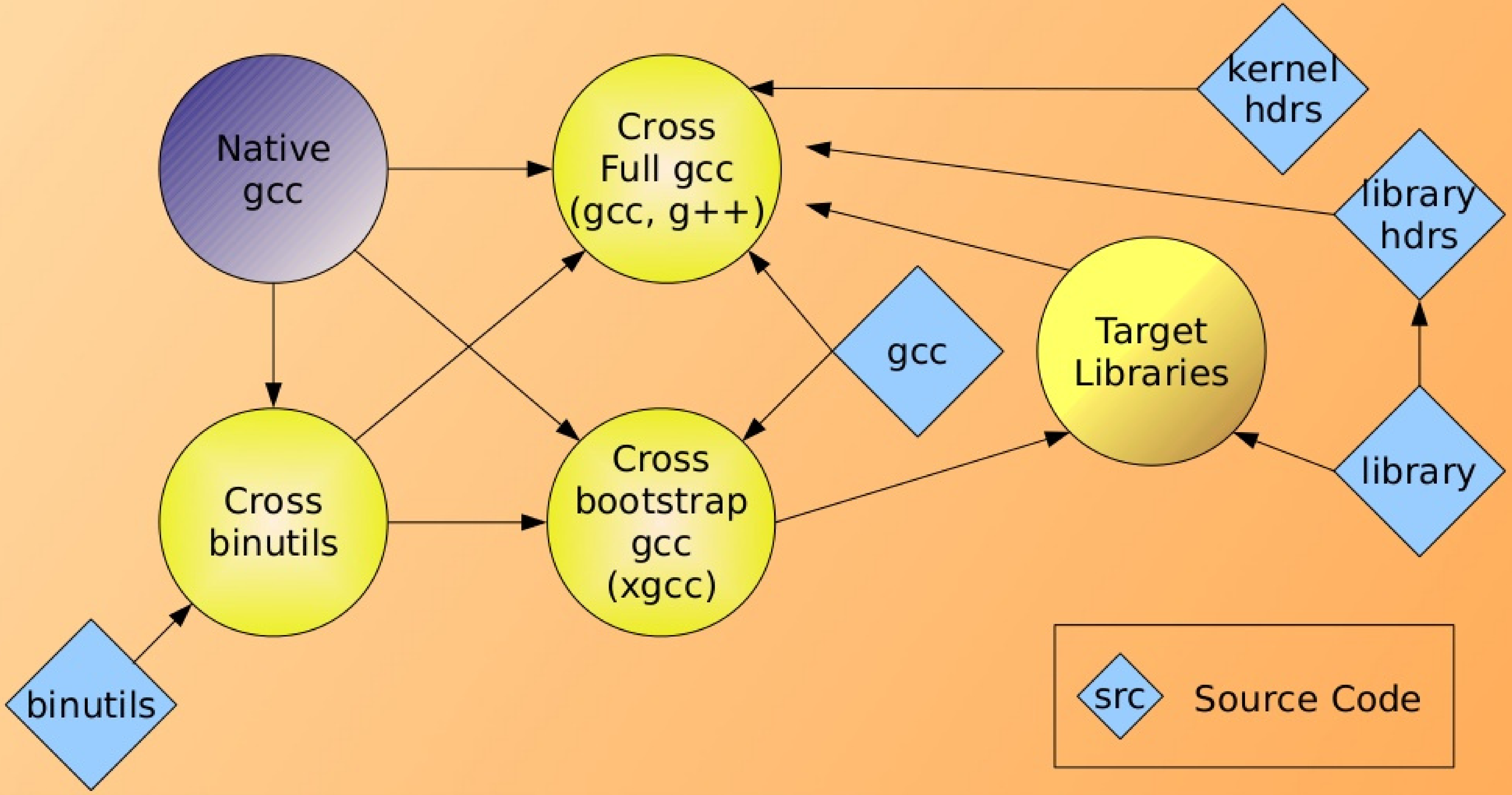
Cross Toolchain Building Overview

- Build of various GNU Packages involved
- Typical Build Steps for these Packages
  - Download & Unpack the source of the Package
  - Configure the Package for Cross-Platform Development
  - Build the Package (make)
  - Install the Package (make install)
- Configuring involves setting up build, host, target
  - ./configure build=... host=... target=...
- Using Triplet: cpu[-manufacturer][-kernel][-os/obj_file_fmt]
- Examples: i386-pc-linux-gnu, xscale-sun-solaris2.5/elf
- Tool Prefix: Same as the triplet

Cross Toolchain Headers



Cross Toolchain Component Dependency



Cross Toolchain Building Steps

- Set up Linux Kernel Headers
  - Ideally from the Kernel version being used
  - Commands
    - make ARCH=<arch> headers_check
    - make ARCH=<arch> INSTALL_HDR_PATH=install_dir/ headers_install
- Build Binary Utilities
- Build the bootstrap Compiler (The C only Compiler)
- Build the C Library
- Build the full Compiler

Building a Toolchain using Crosstool

- Install the Crosstool
  - cd cross-tool-ng
  - ./configure –prefix=/opt/board/
  - make
  - make install
  - cp ct-ng.comp /etc/bash_completion.d/
  - export PATH=$PATH:/opt/board/bin/

Building a Toolchain using Crosstool

- Build the Toolchain
  - mkdir ct-build src
  - cd ct-build/
  - mkdir .build
  - cp Templates/Toolchain/sources.tgz .build/

    (available from Downloads section of http://sysplay.in)
  - ct-ng menuconfig
  - ct-ng build

Testing a Cross Toolchain

- Compile a C program to various stages
  - Pre-process only
  - Get Assembly
  - Get Object
  - Get Executable
- Compile a C program with headers
- Compile a C program with linking libraries
- Create a C Program with floating point operations
- Execute & Test the generated target programs
- Toolchain “Self Contained” Test

What all have we learnt?

- W's of a Toolchain
  - Compiler
  - Binary Utilities
  - Set of C Libraries
- W's & How's of Cross Toolchain?
- Building a Cross Toolchain
  - Building Steps
  - Automated Build Tools
- Testing a Cross Toolchain

Any Queries?