Embedded Linker

Why another linker??

Parth Arora, Rishabh Bali and Shankar Kalpathi Easwaran

Qualcomm India Private Limited

partaror@qti.qualcomm.com

Agenda

Linker Overview

How is embedded programming different?

What is eld?

Why eld?

Key Features: Extended diagnostics and detailed map-file

Key Features: Plugin Framework

Key Features: Modular target design

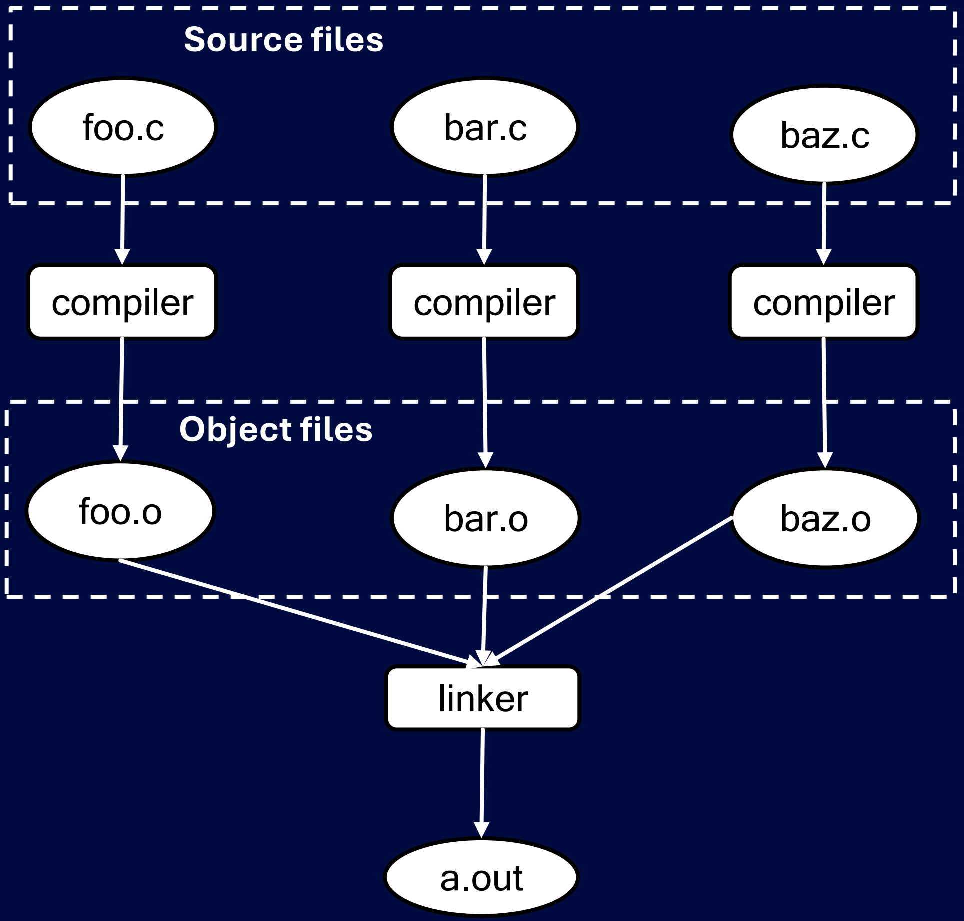
Linker comparison

Future goals

students to undertake on-site internships during their 7th semester for academic credit. He has expressed interest in extending his internship to cover this period.

Linker overview

- Linker plays a key role in a toolchain
  - “Transforms all the ingredients into a meal that is ready to eat.”
- Linker operation:Read Input files
  - Resolve symbols
  - Match input sections to output sections using linker script
  - Layout image
  - Resolve relocations
  - Output image
- Widely available linkers for building applications:
  - GNU Linker, gold linker (gold), LLVM linker (lld), mclinker, Mold linker (mold), Wild linker, ELF toolchain linker



What is embedded programming?

- Programming for a specific hardware.
- More than 90% of computing devices are embedded systems.
- Embedded systems are essential when the goals include:
  - low cost
  - low power
  - long-life
  - Timing constraints
  - high reliability

How is embedded programming different?

- Limited compute and memory resources on the target device.

- Heterogenous memory
  - fast and slow memory regions
  - memory layout constraints

- Need for detailed diagnostics and layout information for analysis when the image layout does not meet expectation.
  - Comparing map-files between a good and a bad build quickly helps to diagnose an issue.

- Need of custom support in toolchain to meet specialized requirements.
  - Overlays
  - Security features such as embedding cryptographic signatures.

```
MEMORY {
  flash : ORIGIN=0x0, LENGTH = 0x1000
  ram : ORIGIN=0x1000, LENGTH = 0x2000
}

SECTIONS  {
  .text : {
    *(SORT(.text.sorted.*))
    *(.text)
  } >flash AT>flash
  .data : { *(.data) } >ram AT>ram
  …
}
```

Linker script example

What is eld?

- eld is a GNU-compatible ELF linker. Designed to be used as a drop-in replacement to GNU.
- Derived from mclinker.
- Utilizes LLVM for reading and writing object files instead of reinventing the wheel.
- Supports ARM, AArch64, Hexagon, and RISCV.
  - x86-64 support is in progress.
- Some of the supported features:
  - Static linking
  - Partial linking
  - dynamic linking
    - does not support symbol versioning
  - LTO
  - GNU linker scripts

Why eld?

- User friendly features for embedded development to diagnose problems.
- Linker plugins provide precise control over image layout and enable custom functionalities for complex and specialized use-cases.
- Detailed map-file.
- Detailed documentation and FAQ with examples to help users understand linker behavior.
- The reproduce functionality creates standalone reproducers, crucial for replicating build issues
- Modular target design that makes it easy to modify / add an architecture.

Key features: Extended diagnostics and detailed map-file

- A linker map-file describes the image layout, along with symbols and sections from individual input file.
- Linker map-file helps to understand the output image, and is a key component for debugging any complex issue. Map-files are crucial for embedded projects.
- eld generates highly detailed map-file that contains linker stats, archive member information, input files information, detailed output layout, plugins information, symbol resolution details and more.
- eld map-files enables debugging of linker script expressions, symbol resolution, plugins and more.
- Map file can be customized using `–MapDetail` option which supports:
  - `show-initial-layout, show-symbol-resolution, ...`

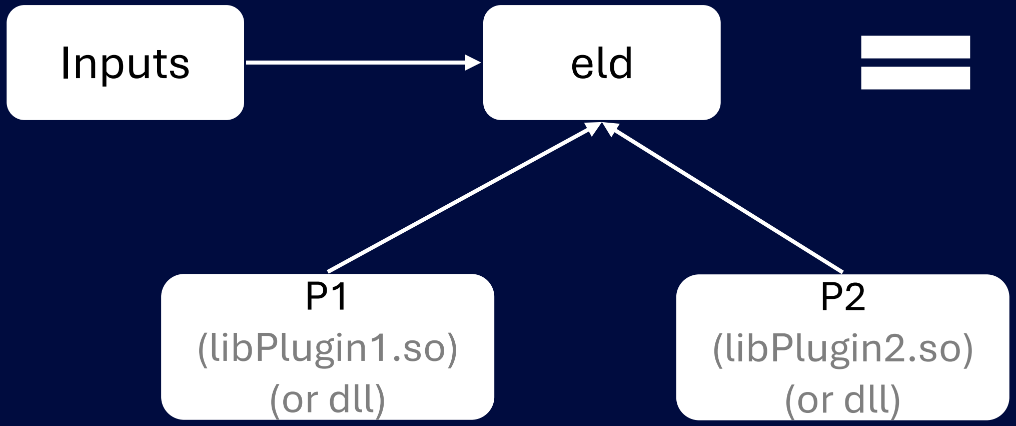
Key features: Extended diagnostics and detailed map-file

- Extensive trace options. `--trace` supports:
  - `linker-script, garbage-collection, plugin, live-edges, ...`

- Extensive warning options:
  - `-Wall, -W[no-]linker-script, -W[no-]attribute-mix, -W[no-]archive-file, -W[no-]bad-dot-assignments, ...`

Key features: Plugin framework

- eld plugin framework allows users to customize the link behavior without any modifications to the linker.
- The plugin framework provides finer control over the image layout than what is possible using traditional linker scripts.
- Link customization possibilities are endless with linker plugins!



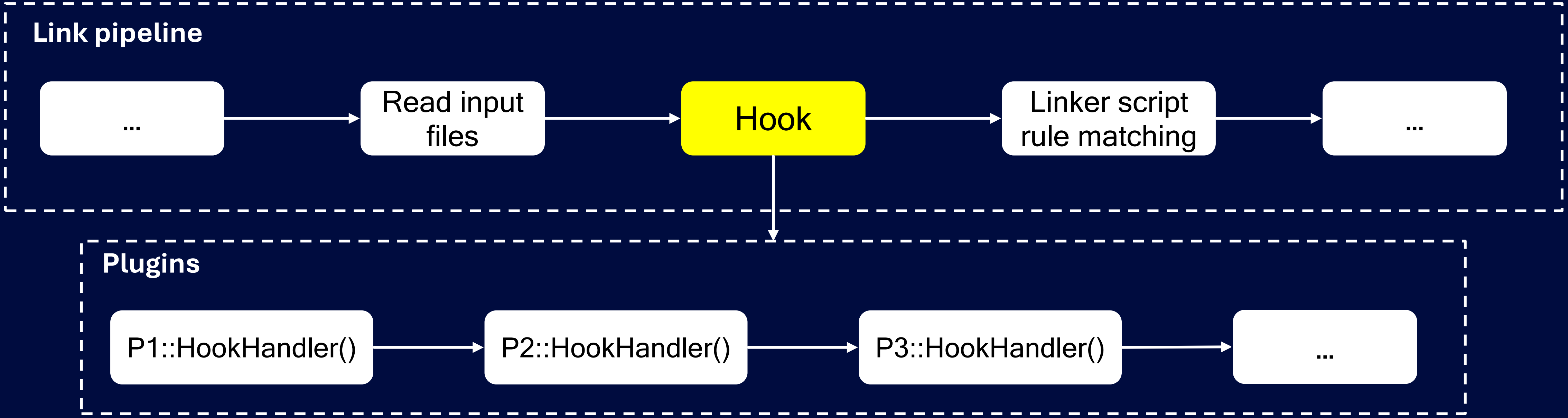
Plugins customized link behavior

P1 and P2 are plugins

Key features: Plugin framework

- Plugins can inspect and modify any of the key linker features and data structures such as: symbols, sections, relocations, input files, rule-matching, diagnostics, LTO, garbage-collection, and more.

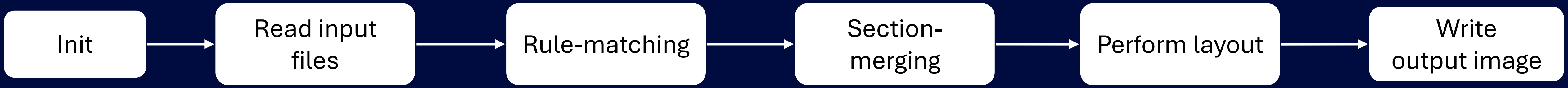
- The plugin framework provides hooks at key points of the link pipeline and plugins implement hook handlers to customize the link behavior.



Key features: Plugin framework: Plugin Hooks

- Hooks names indicate where the hook is placed within the link pipeline. There are two kinds of hooks:
  - **Visit<Component>**: These hooks are called just after creating the component. For example: VisitSection and VisitSymbol.
  - **ActBefore<LinkState>**: These hooks are called just before the linker enters a particular link state. For example: ActBeforeRuleMatching and ActBeforeSectionMerging

  - Simplified link pipeline:



Key features: Plugin Framework: use-case examples

- Plugin framework enables complex image layouts such as section placement based on PGO information and section budgeting.

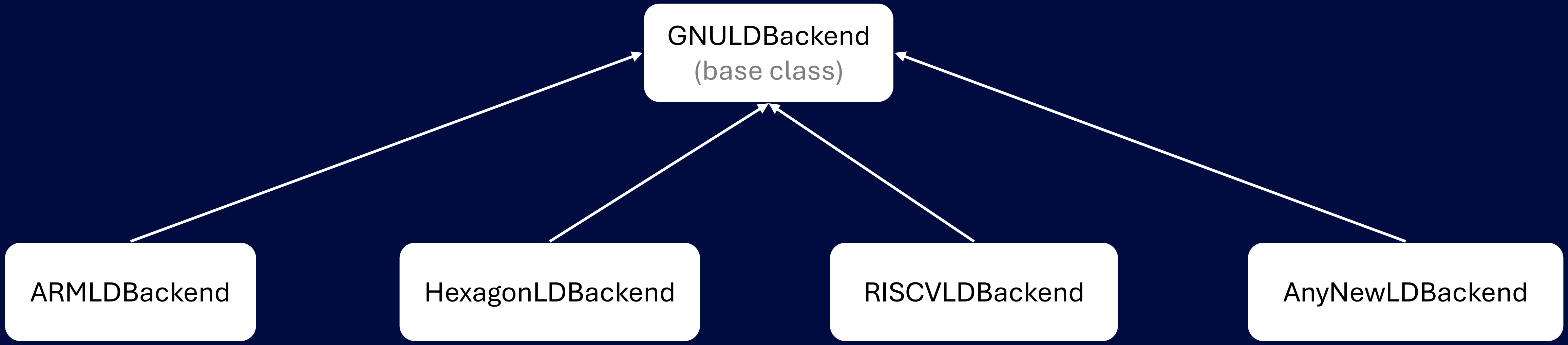
- Plugin framework design can be used to create a plugin for supporting LTO with linker scripts.

- It can be used to speed-up link time by utilizing a caching layer for link steps, without any modification to the core linker functionality.

- It can be used to achieve a communication mechanism between compiler and the linker to enable specialized use-cases.

Key features: Modular Target Design

- Modifying a backend does not require changes to the core linker logic.
- New architectures can be added by simply implementing new target files and registering the target.



Target backend class can override any functionality specified in the base class GNULDBackend

Comparing features among linkers

Comparing features among linkers

Feature	eld	GNU linker	lld	mold
GNU linker scripts	+	+	+	-
GNU command-line options	+	+	+	+
GNU linking semantics	+	+	-	-
ARM, AArch64, RISCV	+	+	+	+
Static linking	+	+	+	+
Dynamic linking	+	+	+	+
Symbol versioning	-	+	+	+
Linker map-file	+++	++	+	+
Multithreaded design	+	-	+	+

Comparing features among linkers

Feature	eld	GNU linker	lld	mold
Easily parseable map-file format	+	-	-	-
X86-64 target	-	+	+	+
Garbage-collection	+	+	+	+
Linker relaxations	+	+	+	+
Tracing information	++	+	+	+
Linker plugins	++	+	-	-
Compiler-to-linker communication	+	-	-	-
Reproduce functionality	+	-	+	+

Future goals

- Feature parity with LLVM lld and GNU ld.
- Further performance improvements.
- Reduce memory utilization.
- Command line tools to debug issues and query map file.

Thank you

Nothing in these materials is an offer to sell any of the components or devices referenced herein.

Qualcomm and Snapdragon are trademarks or registered trademarks of Qualcomm Incorporated.
Other products and brand names may be trademarks or registered trademarks of their respective owners.

References in this presentation to “Qualcomm” may mean Qualcomm Incorporated, Qualcomm Technologies, Inc., and/or other subsidiaries or business units within the Qualcomm corporate structure, as applicable. Qualcomm Incorporated includes our licensing business, QTL, and the vast majority of our patent portfolio. Qualcomm Technologies, Inc., a subsidiary of Qualcomm Incorporated, operates, along with its subsidiaries, substantially all of our engineering, research and development functions, and substantially all of our products and services businesses, including our QCT semiconductor business.

Snapdragon and Qualcomm branded products are products of Qualcomm Technologies, Inc. and/or its subsidiaries.
Qualcomm patented technologies are licensed by Qualcomm Incorporated.

Follow us on: in 𝕏

For more information, visit us at qualcomm.com & qualcomm.com/blog