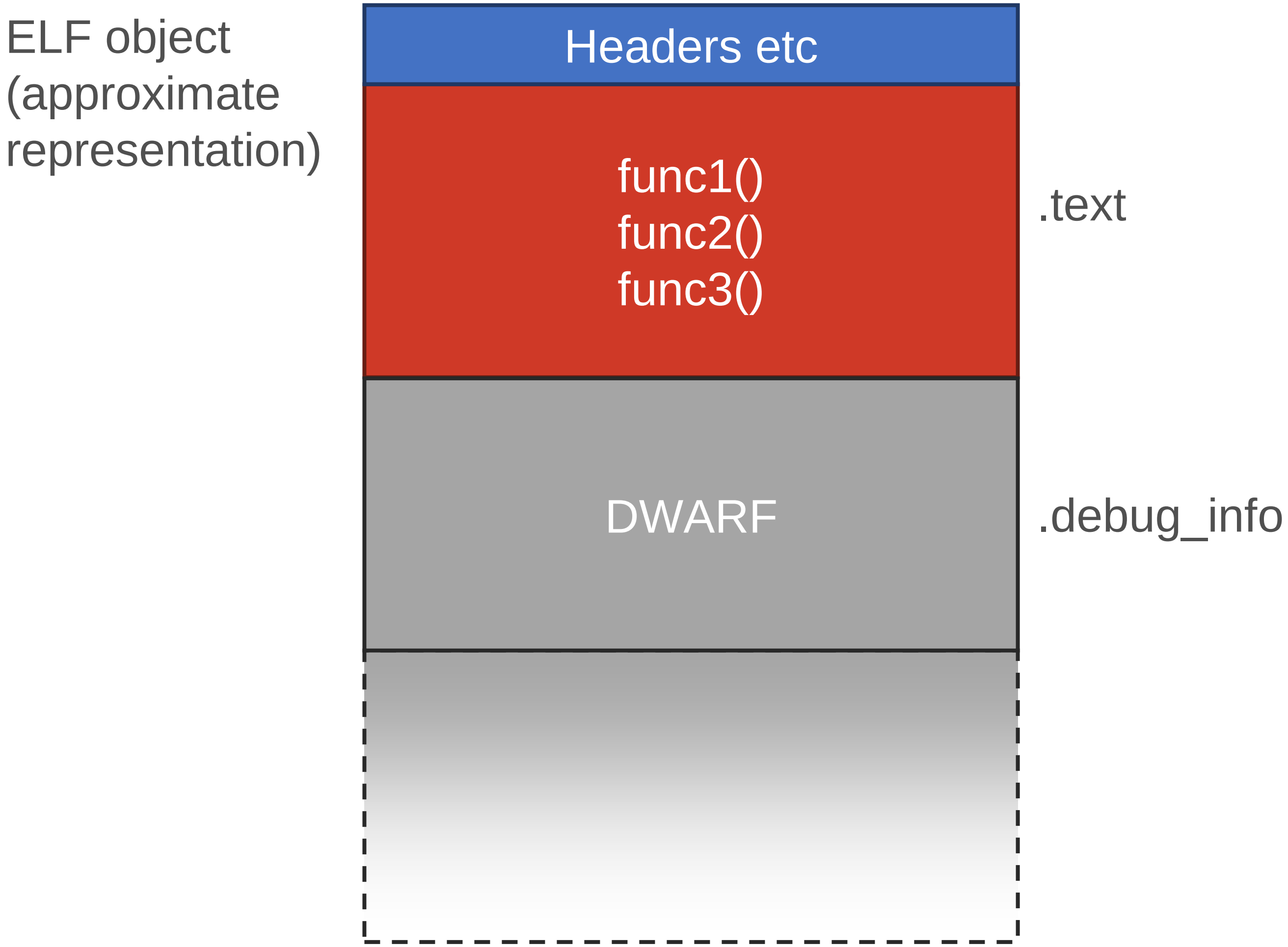
Dead Debug Data Elimination Using Fragmented DWARF

James Henderson

SN Systems (Sony Interactive Entertainment)

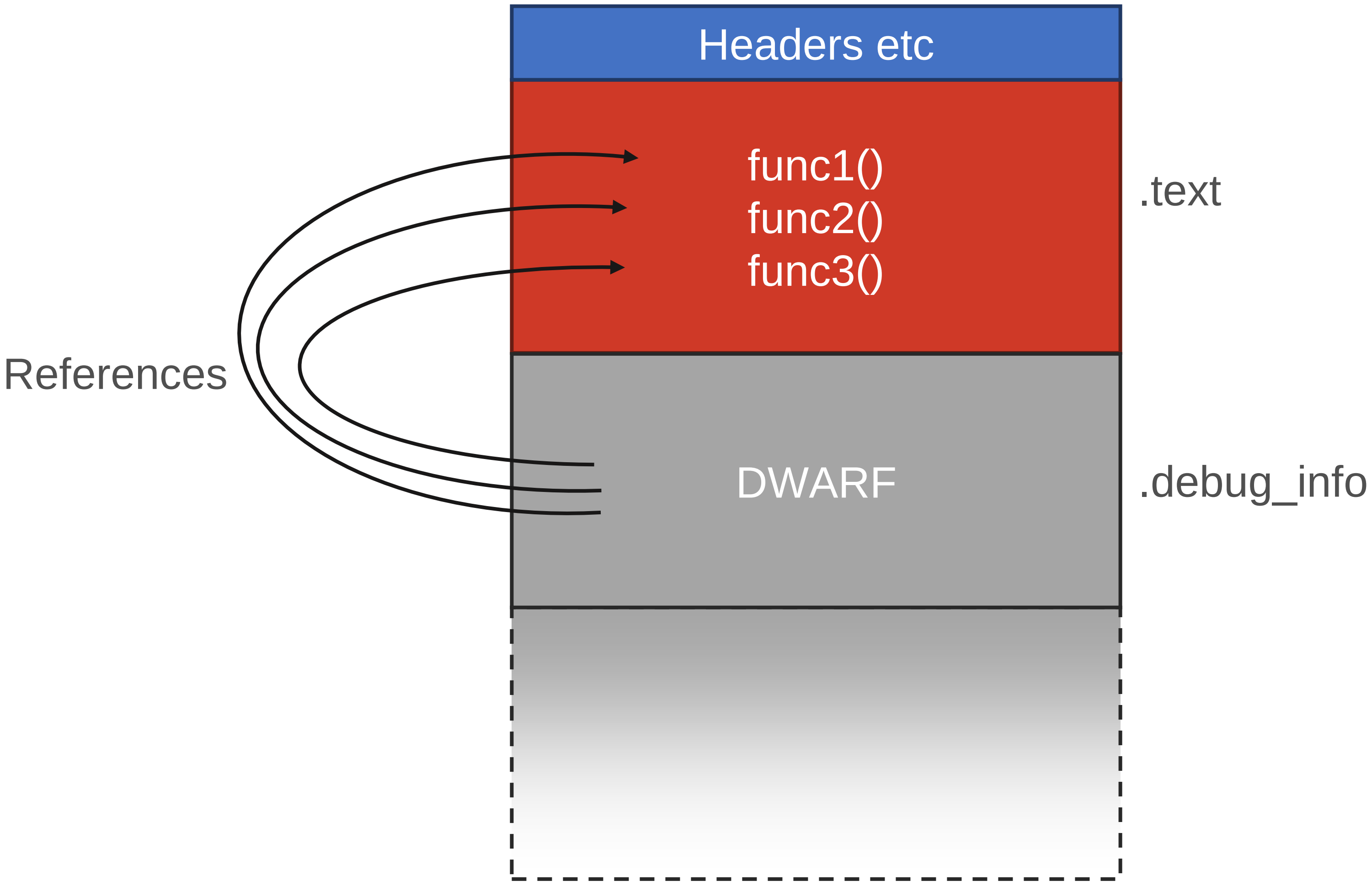
LLVM Developers' Meeting 2020

The Problem



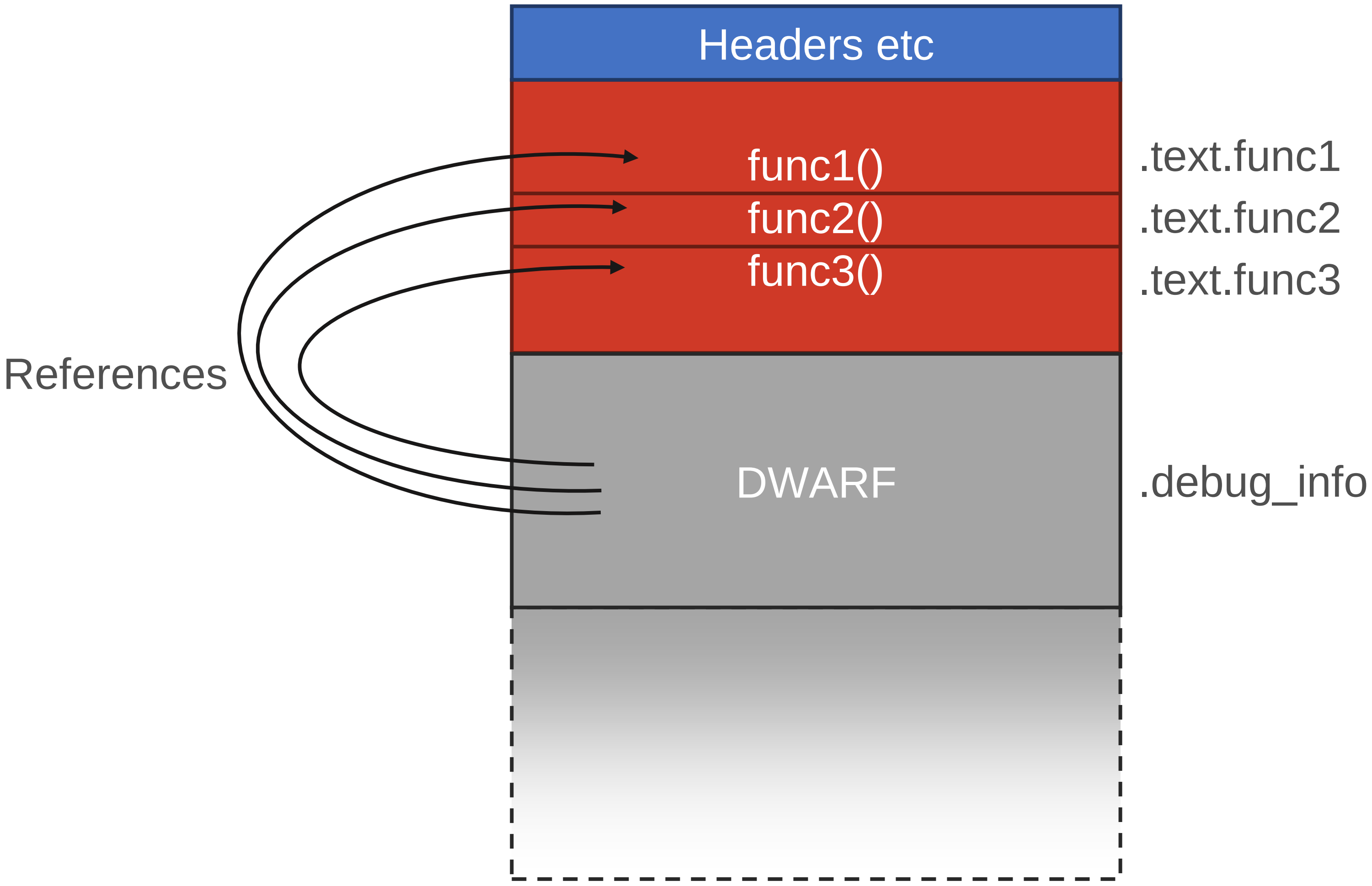
ELF object
(approximate
representation)
Headers etc
func1()
func2()
func3()
.text
DWARF
.debug_info

The Problem



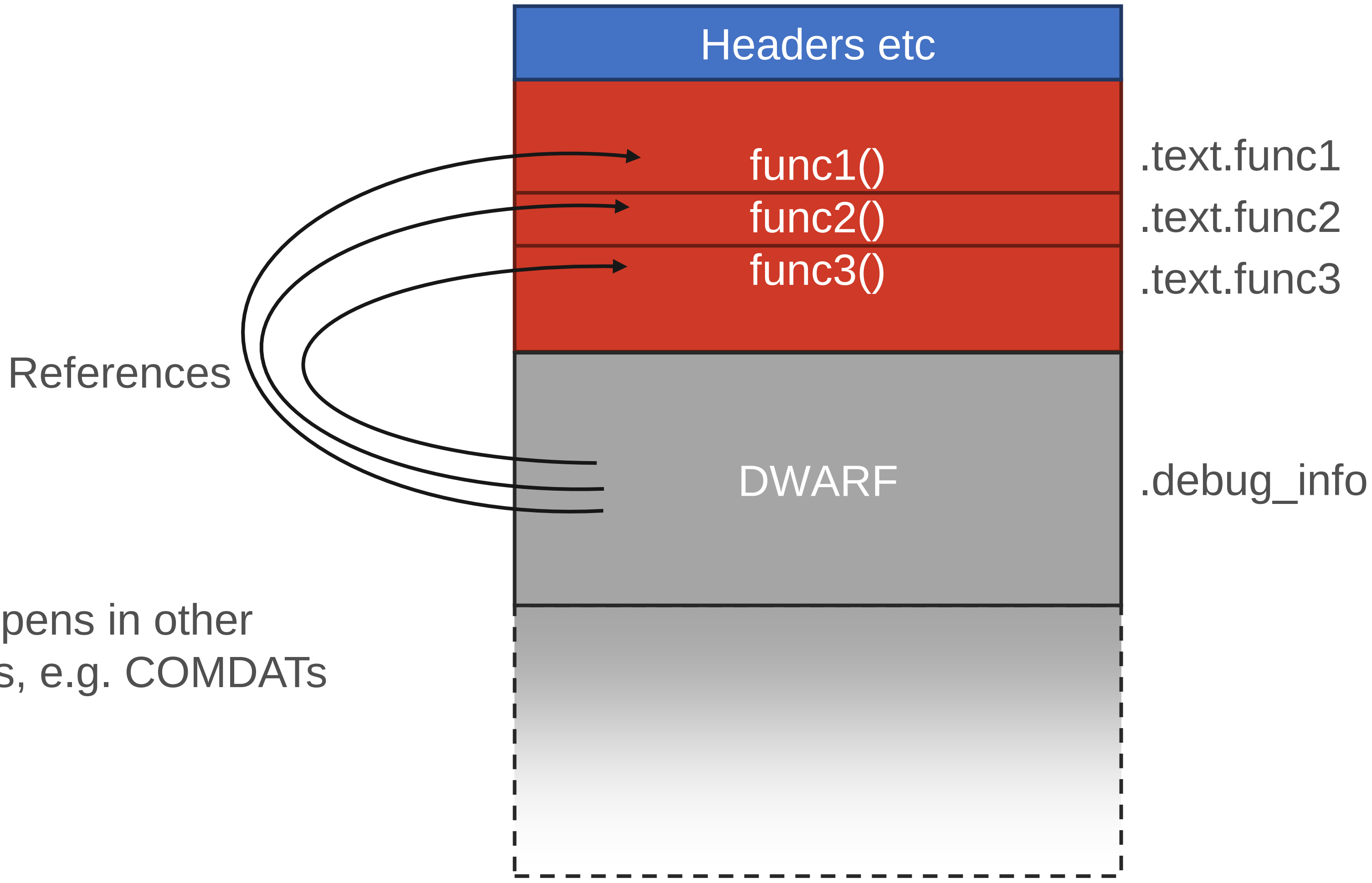
Headers etc
func1()
func2()
func3()
.text
References
DWARF
.debug_info

The Problem



clang -ffunction-sections ("section per function")

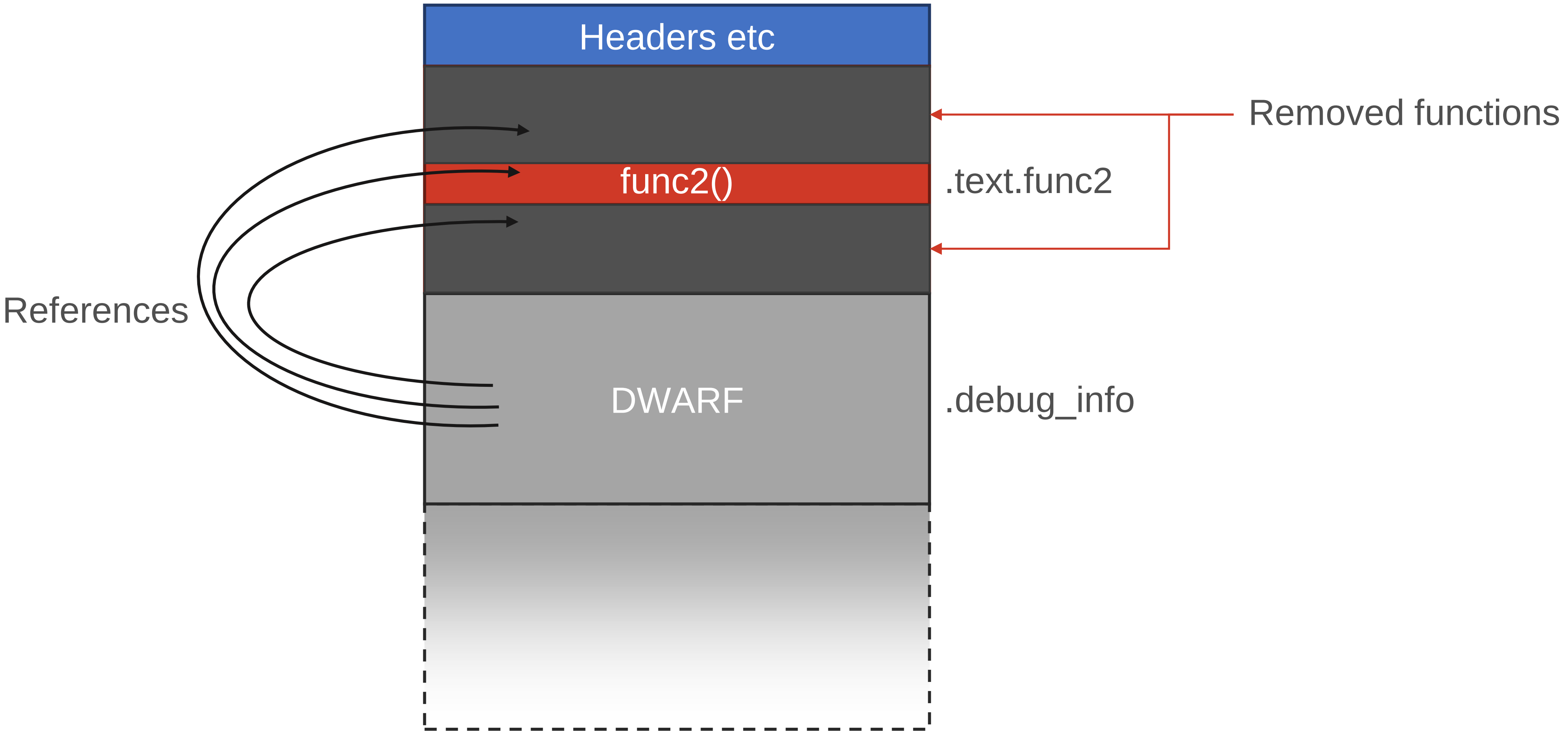
The Problem



Also happens in other situations, e.g. COMDATs

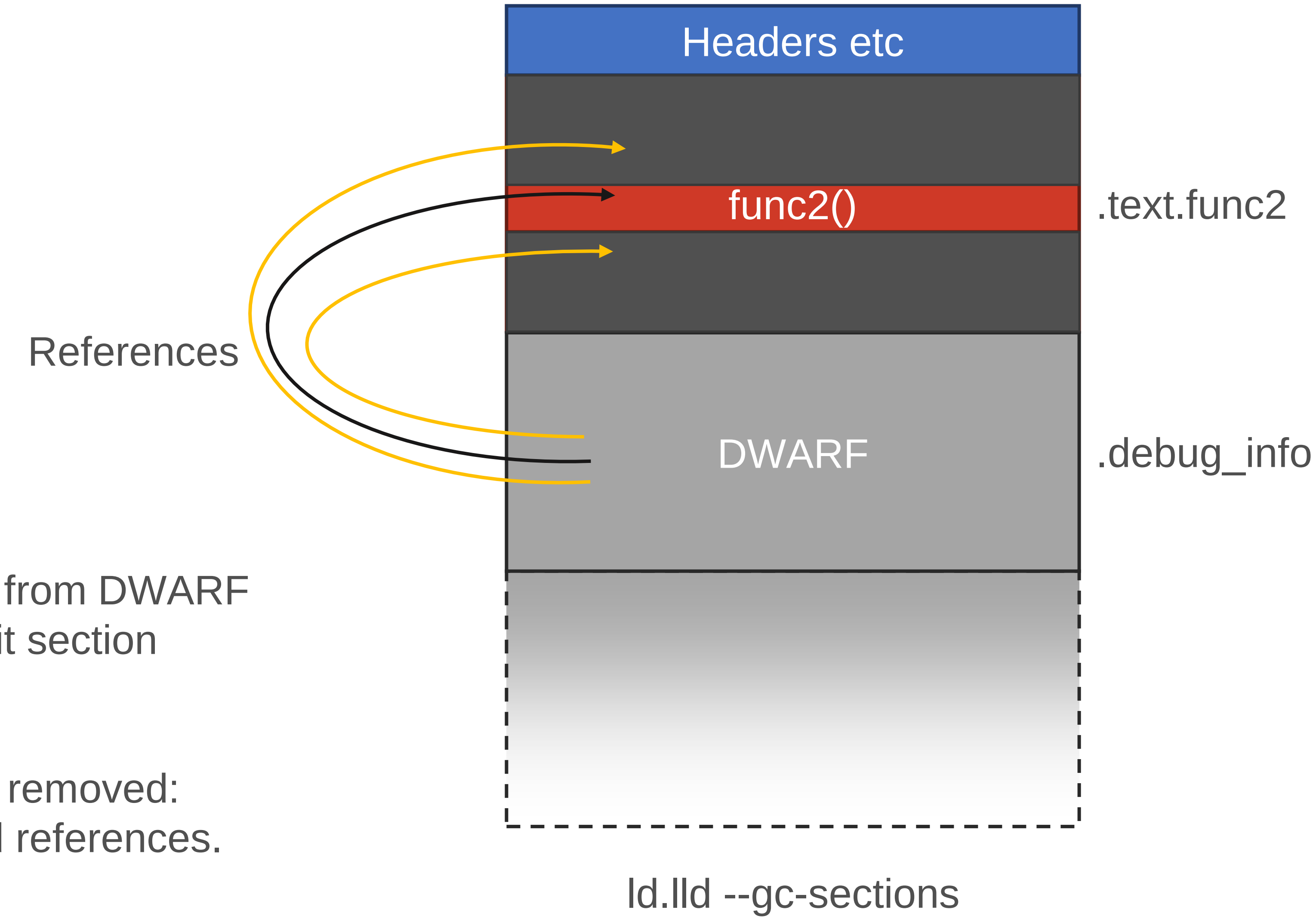
clang -ffunction-sections

The Problem



ld.lld --gc-sections

The Problem



References from DWARF do not inhibit section removal.

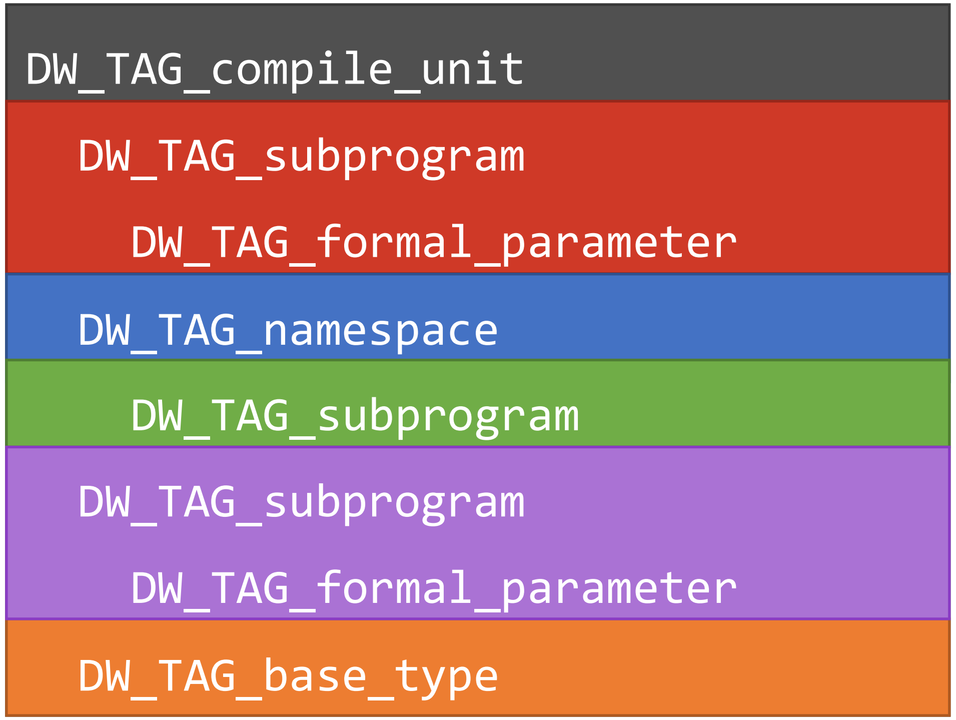
No DWARF removed: leaves dead references.

Fragmenting DWARF (example)

```
DW_TAG_compile_unit
  DW_TAG_subprogram
    DW_TAG_formal_parameter
  DW_TAG_namespace
    DW_TAG_subprogram
  DW_TAG_subprogram
    DW_TAG_formal_parameter
  DW_TAG_base_type
```

- Example (simplified) .debug_info tree.
- Contains 3 functions (“subprograms”).

Fragmenting DWARF (example)



- Fragment by splitting into “generic” bits and parts for specific functions (and variables).
- Result is 6 separate .debug_info sections.
- .debug_info sections for functions linked to corresponding .text sections.

Fragmenting DWARF (example)

```
DW_TAG_compile_unit

  DW_TAG_namespace

  DW_TAG_subprogram
    DW_TAG_formal_parameter
  DW_TAG_base_type
```

- If linker discards .text section, it also discards associated .debug_info section.

Fragmenting DWARF (example)

```
DW_TAG_compile_unit
  DW_TAG_namespace
  DW_TAG_subprogram
    DW_TAG_formal_parameter
  DW_TAG_base_type
```

- Linker concatenates like-named sections together.
- Result is smaller DWARF with no dead references.
- Same approach works for other debug sections.

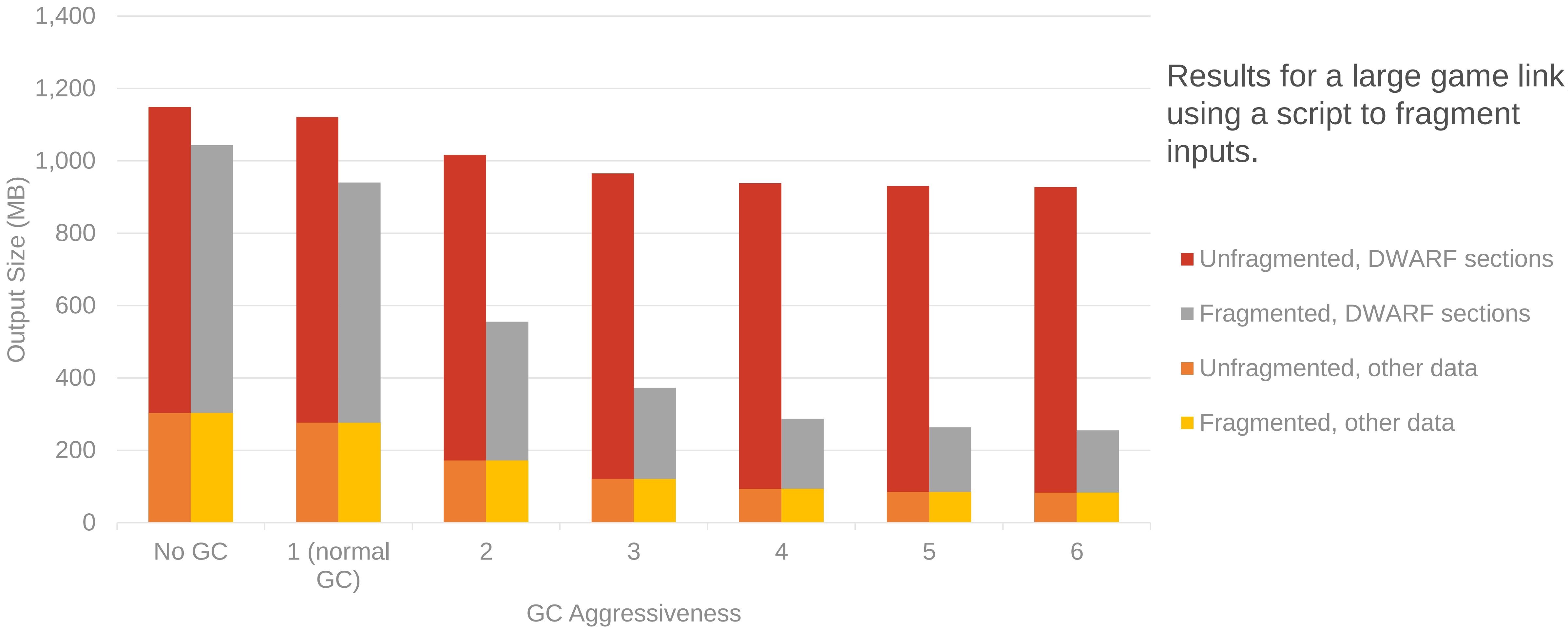
Fragmenting DWARF Limitations

- Works for .debug_info, .debug_line, .debug_aranges, .debug_ranges and .debug_loc.
  - Other sections don’t have direct references to variables/functions.
  - Doesn’t work for DWARF v5 .debug_rnglists/.debug_loclists, due to usage of entry indexes.

- Doesn’t get rid of all “useless” information.
  - E.g. empty namespace tags, unused .debug_abbrev entries.

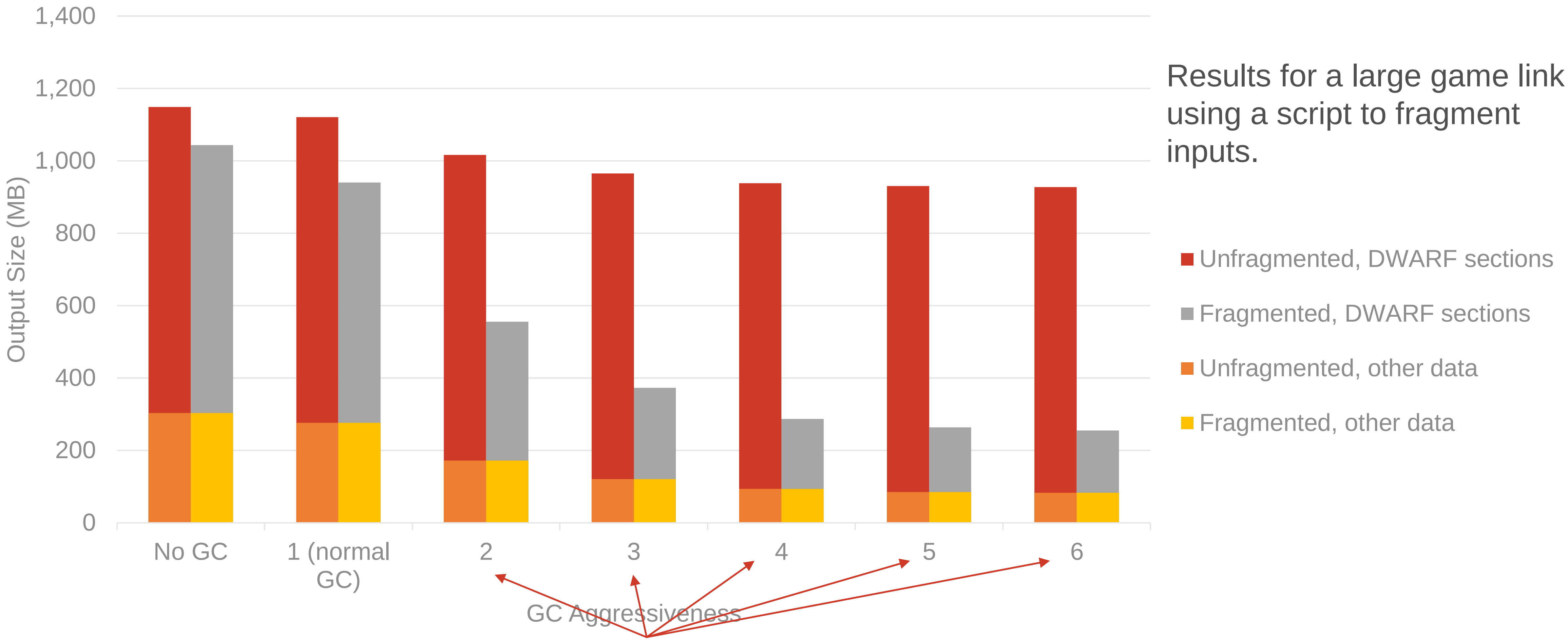
- Intermediate objects not valid DWARF…
  - … but consumers could be taught how to read them.

Performance (Output Size)



Results for a large game link, using a script to fragment inputs.

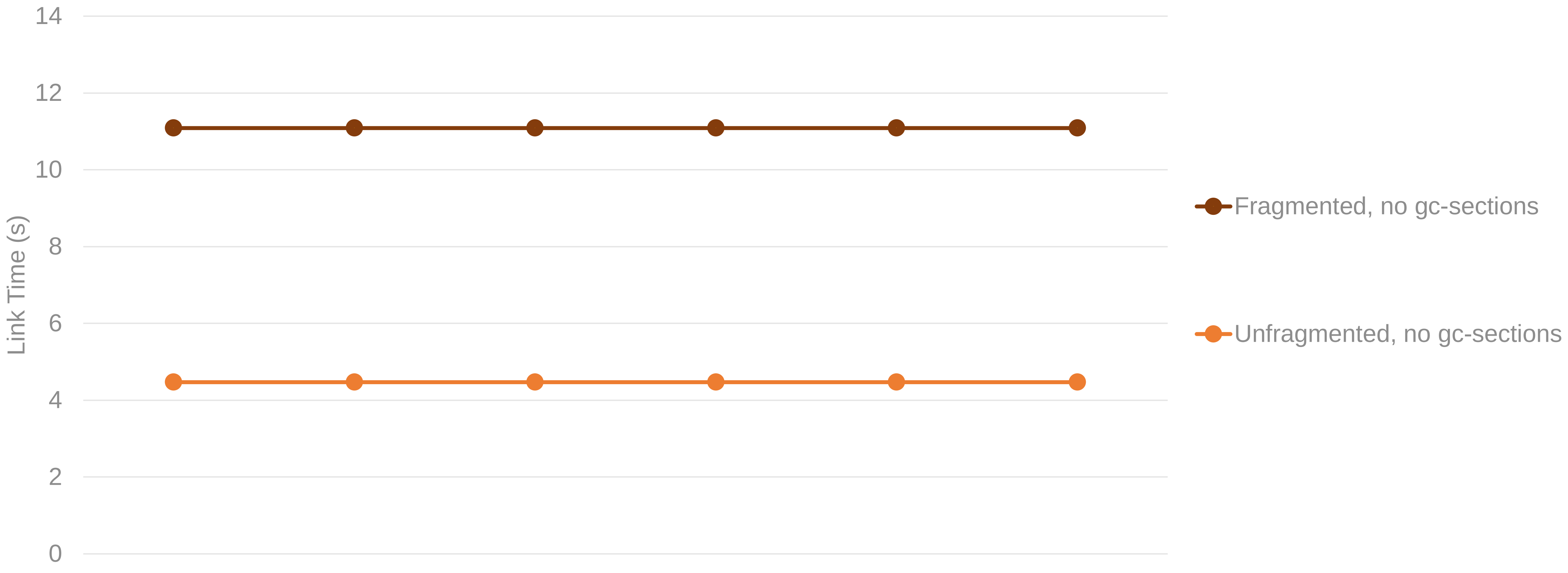
Performance (Output Size)



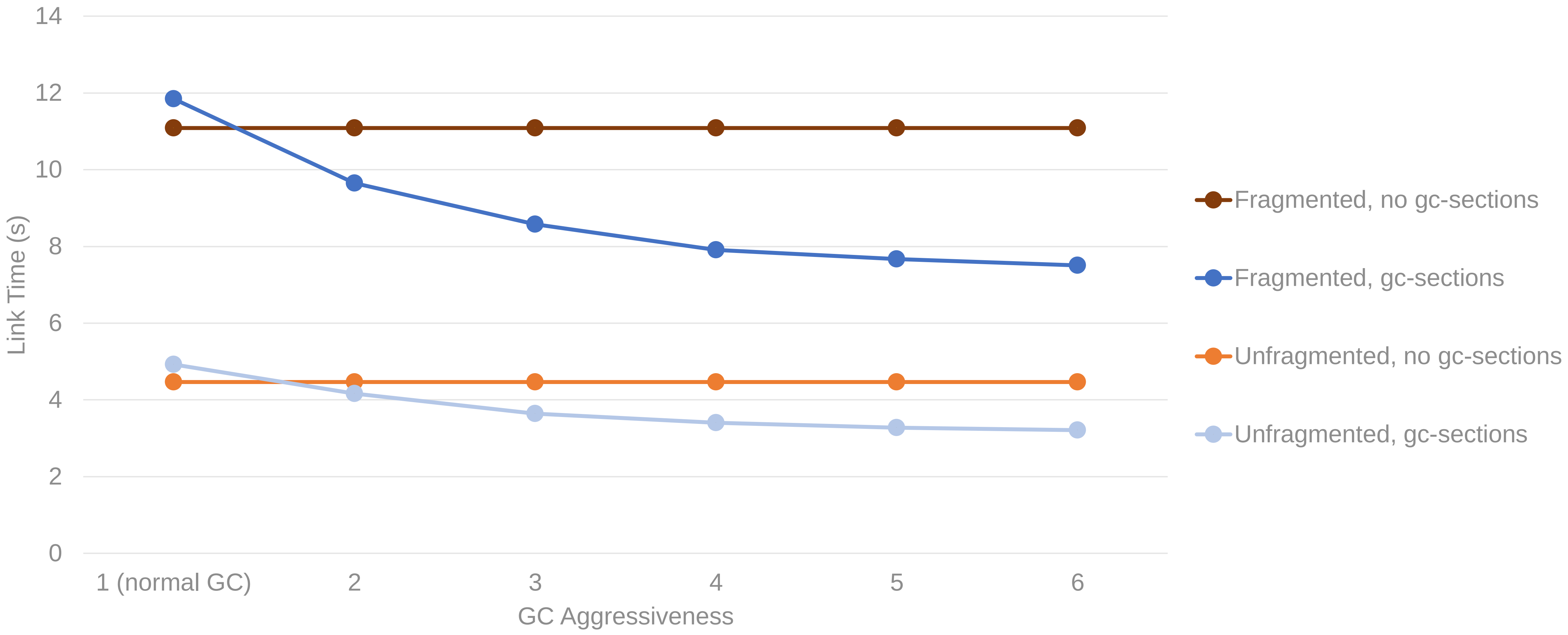
Artificially treat more sections as “dead” by ignoring relocations in liveness analysis

Results for a large game link, using a script to fragment inputs.

Performance (Link Time)



Performance (Link Time)



Caveats

- Results use a modified LLD to support:
  - Using SHF_LINK_ORDER without ordering anything.
    - Reordering would corrupt the debug data.
  - References sections in groups, from outside the group.
    - Illegal according to the ABI.
  - Could use Group Sections instead.

- Figures include LLD patch improving performance.
  - Avoids doing some unnecessary string comparisons for debug sections.
  - May not be 100% correct for all objects - needs further investigation.

Alternative Solutions

- Rewrite DWARF at link time.
    - What the Sony proprietary linker does for PlayStation® 4 .debug_line.
    - Theoretically what LTO could effectively do.
    - Investigated within LLD by Alexey Lapshin (https://reviews.llvm.org/D74169).
    - Slow, and not particularly within traditional linker's feature set.
        - 8 times slower in Alexey's initial prototype when linking clang.

- Post-link optimization
    - Wasted I/O.
    - Relies on being able to identify dead debug data without relocations.
    - See llvm-dwarfutil proposal (http://lists.llvm.org/pipermail/llvm-dev/2020-August/144579.html).

- Change DWARF structure in new standard
    - Doesn't solve issue for existing standards.

Conclusion

- Fragmenting the sections adds a lot of overhead.
  - Profiling LLD suggests it is largely due to the cost of creating more input sections internally.
  - String matching makes things slow.
  - Time savings from writing less outweighed by this overhead.

- Big size savings available, if willing to pay link time cost.
  - The more dead code, the better the trade-off.
  - Should improve debugger load times.

- Future work:
  - Investigate debugger load time improvements.
  - Use ELF Group sections instead of SHF_LINK_ORDER.
  - Investigate LLD performance improvements for many input sections.
  - Implement script in MC.

Appendix: Duration/Size Changes vs Unfragmented

- Figures for fragmented approach as a percentage of the unfragmented approach:

Relocations used for GC liveness analysis	Link Time	Size (total)	Size (debug data)
No GC	248%	91%	88%
100% (normal GC)	240%	84%	79%
80%	232%	55%	45%
60%	236%	39%	30%
40%	232%	31%	23%
20%	234%	28%	21%
0%	234%	28%	20%