Linker Code Size Optimization for Native Mobile Applications

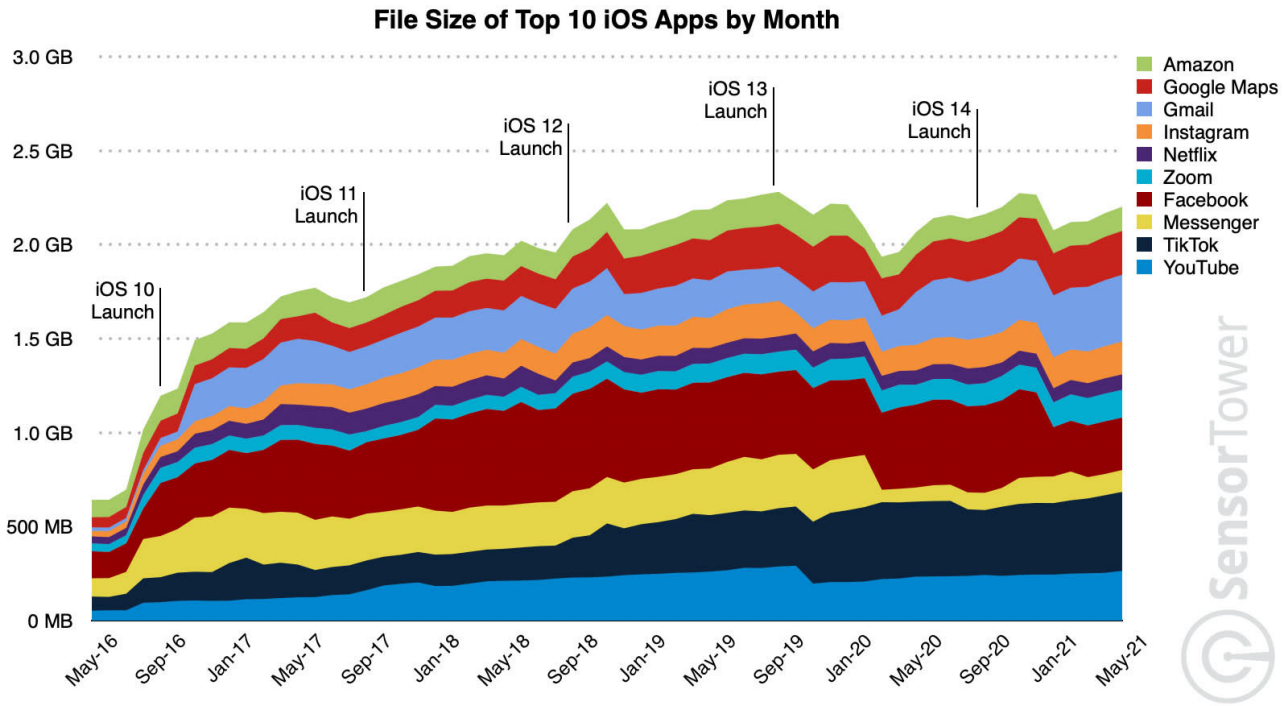
Gai Liu

in collaborations with Umar Farooq, Chengyan Zhao, Xia Liu and Nian Sun

ByteDance

Impact of Code Size on Mobile Applications

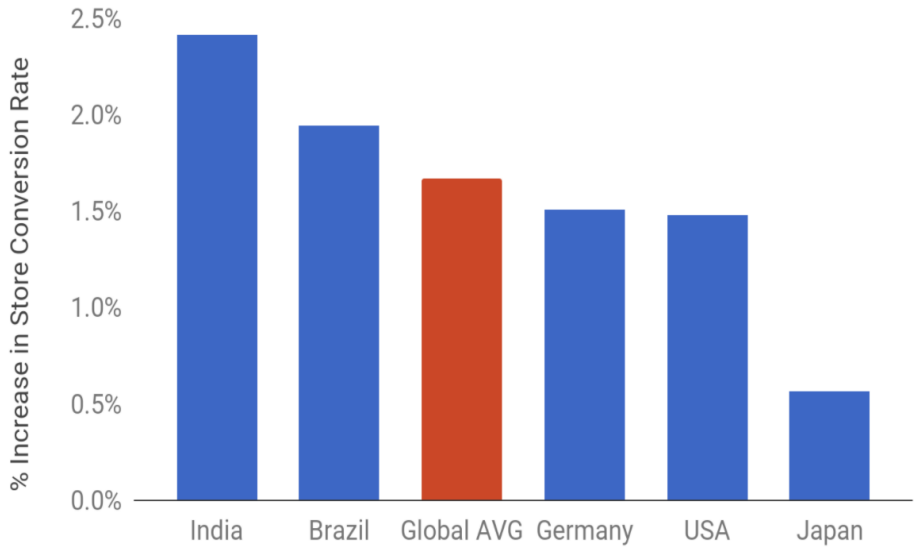
More features → larger apps



Credit: SensorTower

Larger apps decrease user engagement

Install conversion rate increase per 10MB decrease in APK size by market



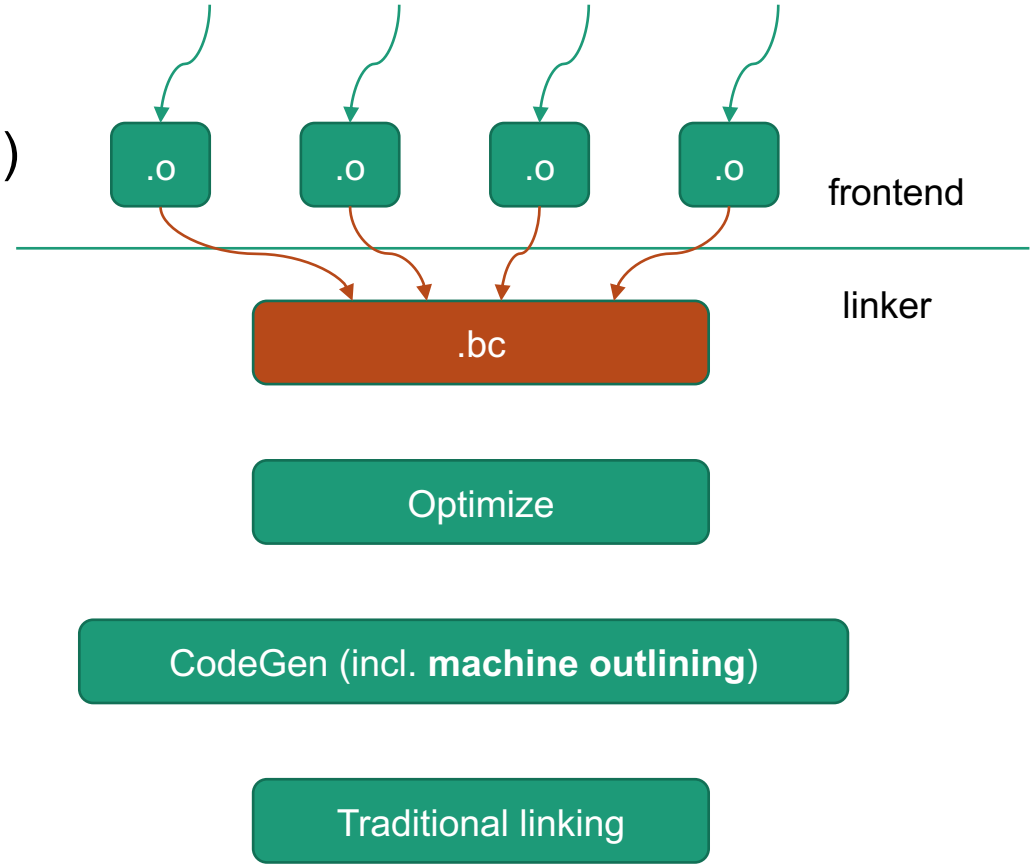
Credit: Sam Tolomei, Google, 2017

Today's topic: transformations to reduce code size for mobile apps

ByteDance

State of the Art Approaches

- Key is to enable **global** size optimization
  - Commonly through link-time optimization (LTO) + machine outlining
  - Monolithic LTO based approach [1]



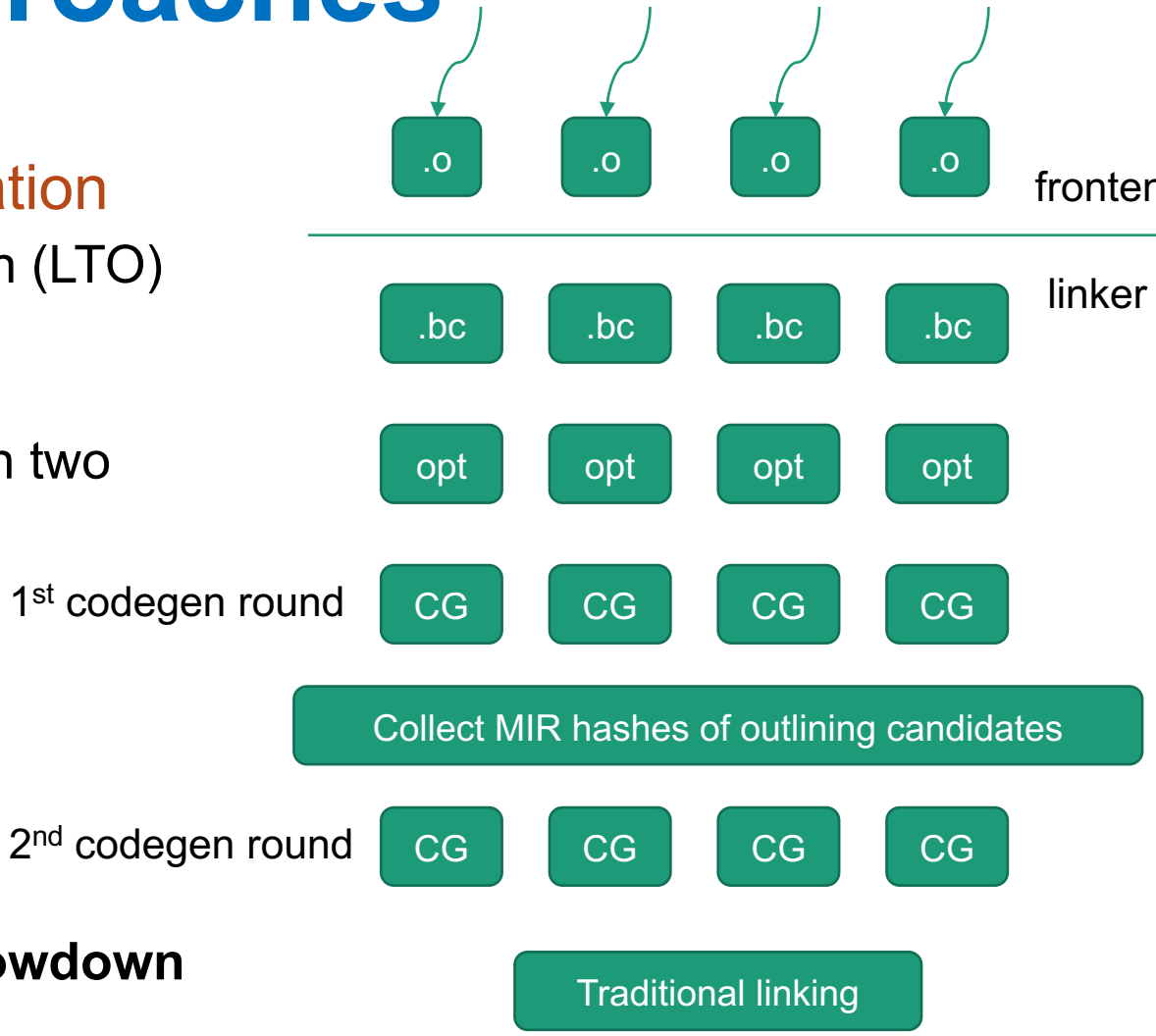
~3x slowdown vs. default pipeline!

[1] Chabbi, Lin, and Barik, CGO'21

ByteDance

State of the Art Approaches

- Key is to enable **global** size optimization
  - Commonly through link-time optimization (LTO) + machine outlining
  - Monolithic LTO based approach [1]
  - Size optimization based on thin LTO with two codegen rounds [2]



~40% slowdown

[1] Chabbi, Lin, and Barik, CGO'21
[2] Lee, Hoag and Tillmann, CC'22

ByteDance

Some Additional Challenges…

~50% code is from binary in our builds

	Compatible with existing build pipeline	Small build time penalty	Optimizes binary form libraries
Monolithic LTO			
Size optimizing thin LTO			

New approach?

ByteDance

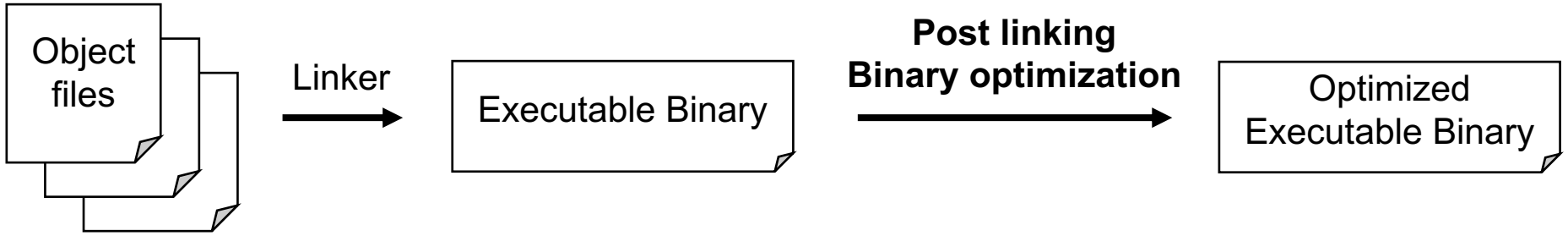
Proposal: Build an Optimizing Linker

- Existing linkers are good at symbol manipulation
  - Dead symbol stripping, function deduplication, etc.
  - But no sophisticated **code transformations inside functions**
- Optimizing code size in linker:

Compatible with existing build pipeline	Small build time penalty	Optimizes binary form libraries
✓ -fuse-ld=myld	✓ lightweight & specialized linker passes	✓

vs. Post Linking Optimization

Snippet of a post linking optimization pipeline:



Linker Opt.

- One implementation per linker
- Easy integration into build pipeline

Post Linking Opt.

- Independent of specific linker
- Additional step in the build pipeline

ByteDance

Rest of the Talk

- An infrastructure for linker code size optimization
  - Implemented on top of Apple's ld64 linker
    - Looking into lld porting
  - Targeting ARM64 architecture
- >10% binary size reduction in our iOS apps
- <10% end-to-end build time overhead

Key Components

Utilities & Analyses

- Instruction decoder
- Sequence hashing

Code transformations

- Sequence outlining
- Safe ICF

Non-code section updates

- Exception table
- Debug info
- etc.

Prepare for Outlining

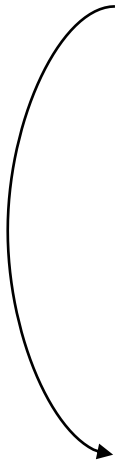
- Decode the instruction stream

```
f45c: sub sp, sp, #48
f460: stp x20, x19, [sp, #16]
f464: stp x29, x30, [sp, #32]
f468: add x29, sp, #32
f46c: adrp  x8, 44223
f470: ldr x8, [x8, #3288]
f474: stp x0, x8, [sp]
f478: adrp  x8, 55266
f47c: ldr x1, [x8, #376]
f480: mov x0, sp
f484: bl  _objc_msgSendSuper2
f488: mov x19, x0
f48c: cbz x0, 0xf4b4
f490: adrp  x8, 44163
f494: ldr x0, [x8, #256]
f498: adrp  x8, 55295
f49c: ldr x1, [x8, #2976]
f4a0: bl  _objc_msgSend
f4a4: ldr x8, [x19, #8]
f4a8: str x0, [x19, #8]
f4ac: mov x0, x8
f4b0: bl  _objc_release
f4b4: mov x0, x19
f4b8: ldp x29, x30, [sp, #32]
f4bc: ldp x20, x19, [sp, #16]
f4c0: add sp, sp, #48
f4c4: ret
```

Prepare for Outlining

- Mark “pivot” instructions
  - Instructions that are not suitable for outlining
  - Targets of branch instructions
  - Exception table range endpoints

```
f45c: sub sp, sp, #48
f460: stp x20, x19, [sp, #16]
f464: stp x29, x30, [sp, #32]
f468: add x29, sp, #32
f46c: adrp  x8, 44223
f470: ldr x8, [x8, #3288]
f474: stp x0, x8, [sp]
f478: adrp  x8, 55266
f47c: ldr x1, [x8, #376]
f480: mov x0, sp
f484: bl  _objc_msgSendSuper2
f488: mov x19, x0
f48c: cbz x0, 0xf4b4
f490: adrp  x8, 44163
f494: ldr x0, [x8, #256]
f498: adrp  x8, 55295
f49c: ldr x1, [x8, #2976]
f4a0: bl  _objc_msgSend
f4a4: ldr x8, [x19, #8]
f4a8: str x0, [x19, #8]
f4ac: mov x0, x8
f4b0: bl  _objc_release
f4b4: mov x0, x19
f4b8: ldp x29, x30, [sp, #32]
f4bc: ldp x20, x19, [sp, #16]
f4c0: add sp, sp, #48
f4c4: ret
```



pivot instructions

0xf4b4
…
….

ByteDance

Find Outlining Candidates

- Hash sequences across all functions
  - Skip pivot instructions

Example: len = 5

h1

```
f45c: sub sp, sp, #48
f460: stp x20, x19, [sp, #16]
f464: stp x29, x30, [sp, #32]
f468: add x29, sp, #32
f46c: adrp  x8, 44223
f470: ldr x8, [x8, #3288]
f474: stp x0, x8, [sp]
f478: adrp  x8, 55266
f47c: ldr x1, [x8, #376]
f480: mov x0, sp
f484: bl  _objc_msgSendSuper2
f488: mov x19, x0
f48c: cbz x0, 0xf4b4
f490: adrp  x8, 44163
f494: ldr x0, [x8, #256]
f498: adrp  x8, 55295
f49c: ldr x1, [x8, #2976]
f4a0: bl  _objc_msgSend
f4a4: ldr x8, [x19, #8]
f4a8: str x0, [x19, #8]
f4ac: mov x0, x8
f4b0: bl  _objc_release
f4b4: mov x0, x19
f4b8: ldp x29, x30, [sp, #32]
f4bc: ldp x20, x19, [sp, #16]
f4c0: add sp, sp, #48
f4c4: ret
```

hash	freq	len
h1	1	5

pivot instructions

0xf4b4
…
….

ByteDance

Find Outlining Candidates

- Hash sequences across all functions
  - Skip pivot instructions

Example: len = 5

```
f45c: sub sp, sp, #48
f460: stp x20, x19, [sp, #16]
f464: stp x29, x30, [sp, #32]
f468: add x29, sp, #32
f46c: adrp  x8, 44223
f470: ldr x8, [x8, #3288]
f474: stp x0, x8, [sp]
f478: adrp  x8, 55266
f47c: ldr x1, [x8, #376]
f480: mov x0, sp
f484: bl  _objc_msgSendSuper2
f488: mov x19, x0
f48c: cbz x0, 0xf4b4
f490: adrp  x8, 44163
f494: ldr x0, [x8, #256]
f498: adrp  x8, 55295
f49c: ldr x1, [x8, #2976]
f4a0: bl  _objc_msgSend
f4a4: ldr x8, [x19, #8]
f4a8: str x0, [x19, #8]
f4ac: mov x0, x8
f4b0: bl  _objc_release
f4b4: mov x0, x19
f4b8: ldp x29, x30, [sp, #32]
f4bc: ldp x20, x19, [sp, #16]
f4c0: add sp, sp, #48
f4c4: ret
```

h2

hash	freq	len
h1	1	5
h2	1	5

pivot instructions

0xf4b4
…
….

ByteDance

Find Outlining Candidates

- Hash sequences across all functions
  - Skip pivot instructions

Linear scan-based algorithm:

- Easy to implement & debug
- Linear time complexity (under a max sequence length)
- Exhaustive for a given *len*

Example: len = 5

h3

```
f45c: sub sp, sp, #48
f460: stp x20, x19, [sp, #16]
f464: stp x29, x30, [sp, #32]
f468: add x29, sp, #32
f46c: adrp  x8, 44223
f470: ldr x8, [x8, #3288]
f474: stp x0, x8, [sp]
...
f478: adrp  x8, 55266
f47c: ldr x1, [x8, #376]
f480: mov x0, sp
f484: bl  _objc_msgSendSuper2
f488: mov x19, x0
f48c: cbz x0, 0xf4b4
f490: adrp  x8, 44163
f494: ldr x0, [x8, #256]
f498: adrp  x8, 55295
f49c: ldr x1, [x8, #2976]
f4a0: bl  _objc_msgSend
f4a4: ldr x8, [x19, #8]
f4a8: str x0, [x19, #8]
f4ac: mov x0, x8
f4b0: bl  _objc_release
f4b4: mov x0, x19
f4b8: ldp x29, x30, [sp, #32]
f4bc: ldp x20, x19, [sp, #16]
f4c0: add sp, sp, #48
f4c4: ret
```

hash	freq	len
h1	1	5
h2	1	5
h3	1	5

pivot instructions

0xf4b4
…
….

ByteDance

Find Outlining Candidates

- Sort candidates by profitability

hash	freq	len
h1	4	4
h2	1	4
h3	8	4
…	…	…
h_n	3	8
…	…	…

Sort by profitability →

hash	freq	len
h15432	38	5
h80	12	8
h912	20	4
…	…	…
h2	1	4
…	…	…

Profitable

Not profitable

```
f45c: sub sp, sp, #48
f460: stp x20, x19, [sp, #16]
f464: stp x29, x30, [sp, #32]
f468: add x29, sp, #32
f46c: adrp  x8, 44223
f470: ldr x8, [x8, #3288]
f474: stp x0, x8, [sp]
f478: adrp  x8, 55266
f47c: ldr x1, [x8, #376]
f480: mov x0, sp
f484: bl  _objc_msgSendSuper2
f488: mov x19, x0
f48c: cbz x0, 0xf4b4
f490: adrp  x8, 44163
f494: ldr x0, [x8, #256]
f498: adrp  x8, 55295
f49c: ldr x1, [x8, #2976]
f4a0: bl  _objc_msgSend
f4a4: ldr x8, [x19, #8]
f4a8: str x0, [x19, #8]
f4ac: mov x0, x8
f4b0: bl  _objc_release
f4b4: mov x0, x19
f4b8: ldp x29, x30, [sp, #32]
f4bc: ldp x20, x19, [sp, #16]
f4c0: add sp, sp, #48
f4c4: ret
```


ByteDance

Outline Transformation

Relocations are handled automatically by downstream linker step

```
f45c: sub sp, sp, #48
f460: stp x20, x19, [sp, #16]
f464: stp x29, x30, [sp, #32]
f468: add x29, sp, #32
f46c: adrp  x8, 44223
f470: ldr x8, [x8, #3288]
f474: stp x0, x8, [sp]
f478: adrp  x8, 55266
f47c: ldr x1, [x8, #376]
f480: mov x0, sp
f484: bl  _objc_msgSendSuper2
f488: mov x19, x0
f48c: cbz x0, 0xf4b4
f490: adrp  x8, 44163
f494: ldr x0, [x8, #256]
f498: adrp  x8, 55295
f49c: ldr x1, [x8, #2976]
f4a0: bl  _objc_msgSend
f4a4: ldr x8, [x19, #8]
f4a8: str x0, [x19, #8]
f4ac: mov x0, x8
f4b0: bl  _objc_release
f4b4: mov x0, x19
f4b8: ldp x29, x30, [sp, #32]
f4bc: ldp x20, x19, [sp, #16]
f4c0: add sp, sp, #48
f4c4: ret
```

```
f0c0: stp x29, x30, [sp, #-16]
f0c4: bl  OUTLINED_PROLOG_71

f0c8: sub sp, sp, #48
f0cc: adrp  x8, 38502
f0d0: ldr x8, [x8, #736]
f0d4: bl  LD_OUTLINED_15371

f0d8: mov x19, x0
f0dc: cbz x0, 0xf0f0
f0e0: adrp  x8, 38439
f0e4: ldr x0, [x8, #2160]
f0e8: bl  LD_OUTLINED_65726

f0ec: bl  LD_OUTLINED_33189

f0f0: mov x0, x19
f0f4: add sp, sp, #48
f0f8: b OUTLINED_EPILOG_311
```

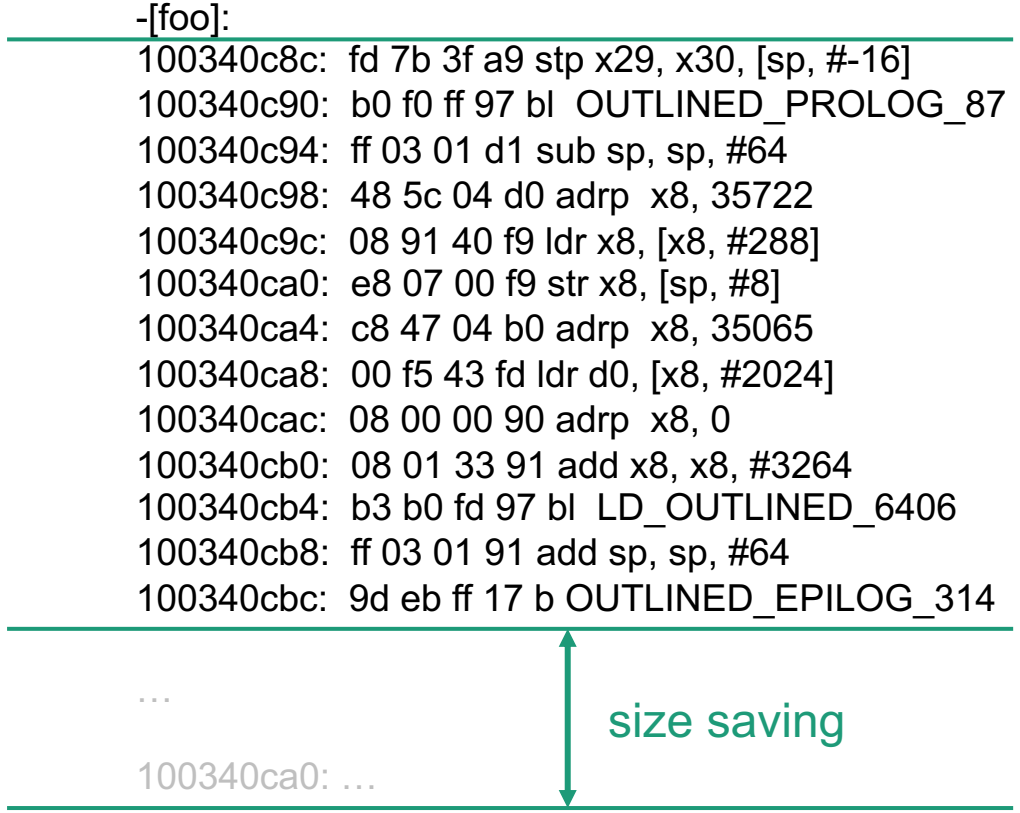
Update cbz's relative offset

```
OUTLINED_PROLOG_71:
 stp  x20, x19, [sp, #-32]
 sub  x29, sp, #16
 ret

LD_OUTLINED_15371:
 stp  x0, x8, [sp]
 adrp x8, 49429
 ldr  x1, [x8, #2752]
 mov  x0, sp
 b  _objc_msgSendSuper2

LD_OUTLINED_65726:
 adrp  x8, 49197
 ldr x1, [x8, #3000]
 b _objc_msgSend

LD_OUTLINED_33189:
 ldr x8, [x19, #8]
 str x0, [x19, #8]
 mov x0, x8
 b _objc_release

OUTLINED_EPILOG_3
11:
 ldp x29, x30, [sp, #-16]
 ldp x20, x19, [sp, #-32]
 ret
```

ByteDance

Update Non-Code Sections

- Fix content of sections that rely on instruction addresses
  - __debug_info
  - __debug_line
  - __gcc_except_table



DWARF debugging information entry

DW_TAG_subprogram
DW_AT_low_pc (0x0000000100340c8c)
DW_AT_high_pc (0x0000000100340ca0)
…

(0x0000000100340cbc)

ByteDance

Safe Identical Code Folding

- Outlining reduces redundancies at instruction sequence level
- ICF removes redundancies at function level
  - Mature pass in lld
- We enhanced the ICF pass in ld64
  - More aggressive size opt
  - Improved compile time
  - Made it safe under pointer comparisons
    - Use redirection instead of direct replacement
  - More details offline

Implementation & Experimental Setup

- ~3500 lines of C++ code as ld64 passes
- Benchmarks
  - Three commercial iOS apps
    1. News recommendation
    2. Short video hosting and sharing
    3. Enterprise collaboration client
  - mixture of Objective-C, Swift, C++, Rust
    - 1 to 2 million functions per app
- Build machine
  - 2.6 GHz, 6-core CPU, 64 GB RAM

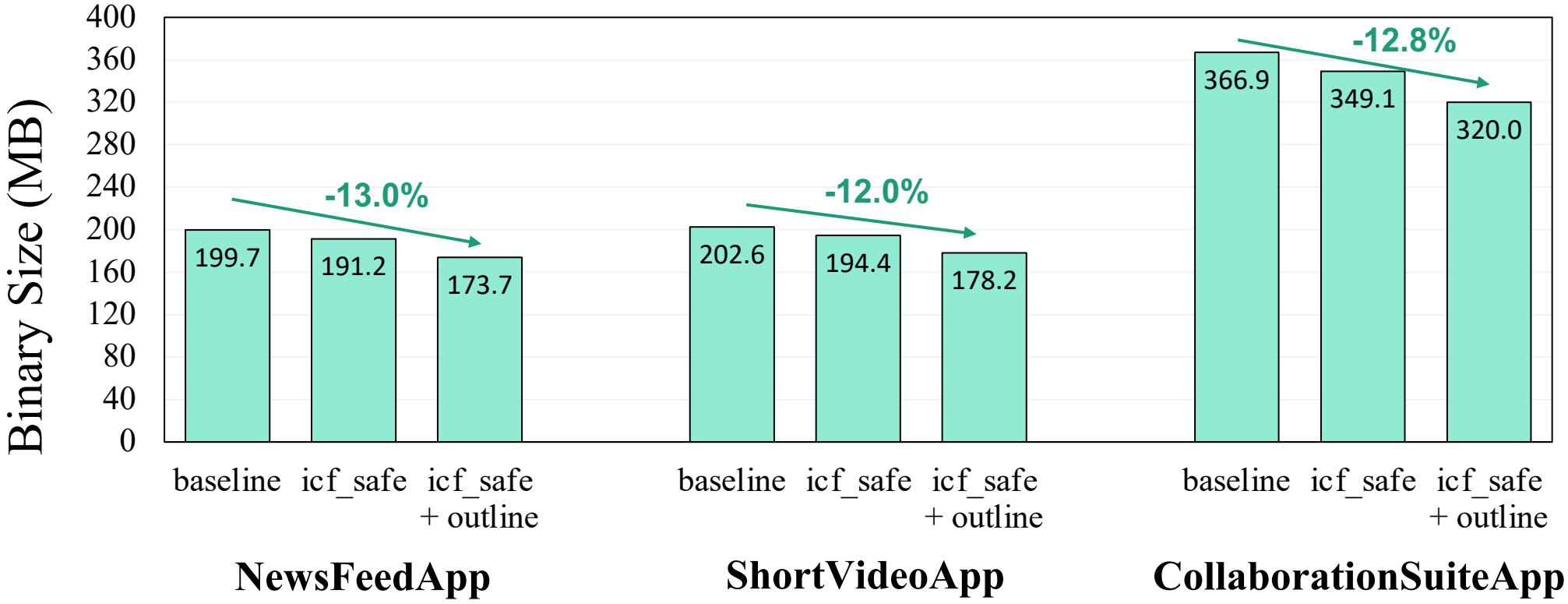
ld64 pass pipeline

Passes/Optimizations

- Objective-C optimizations
- Stub/GOT/TLV generation
- **Linker outlining**
- Order atoms
- **Safe identical code folding**
- Branch island/shim generation
- DTrace probe processing
- Compact unwind encoding

ByteDance

Size Reduction



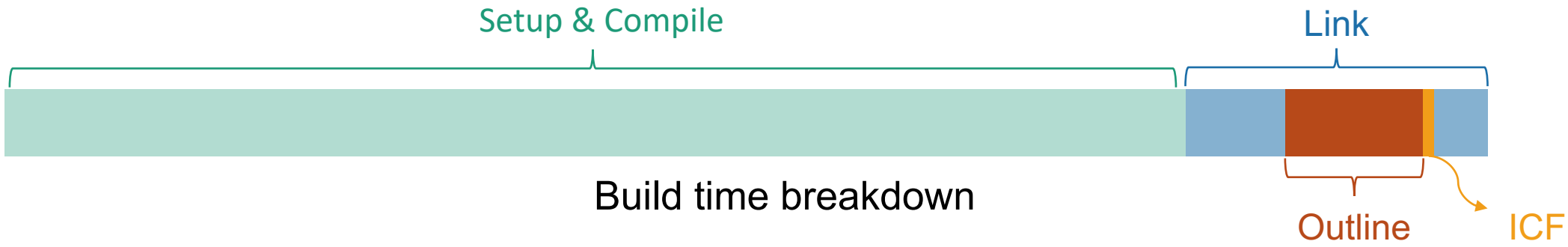
No noticeable performance degradations from production data

ByteDance

Build Time Comparisons

- Linker outline + ICF overhead
  - Doubles the link time
  - <10% overhead overall

	Wall Time
icf_safe pass	17s
outline pass	2m15s
Total link time	5m23s
Total build time	46m31s



Build time breakdown

- Comparison
  - Monolithic LTO increases link time by > 2 hours for our apps

ByteDance

Summary

- We presented a framework for native code size optimization within the linker
  - Minimal change to build pipeline
  - Small build time overhead
  - Optimizes binary libraries

- arXiv preprint
  - arxiv.org/abs/2210.07311v1

ByteDance