

# *OrcJIT at scale with the llvm-autojit plugin*

↓ slides



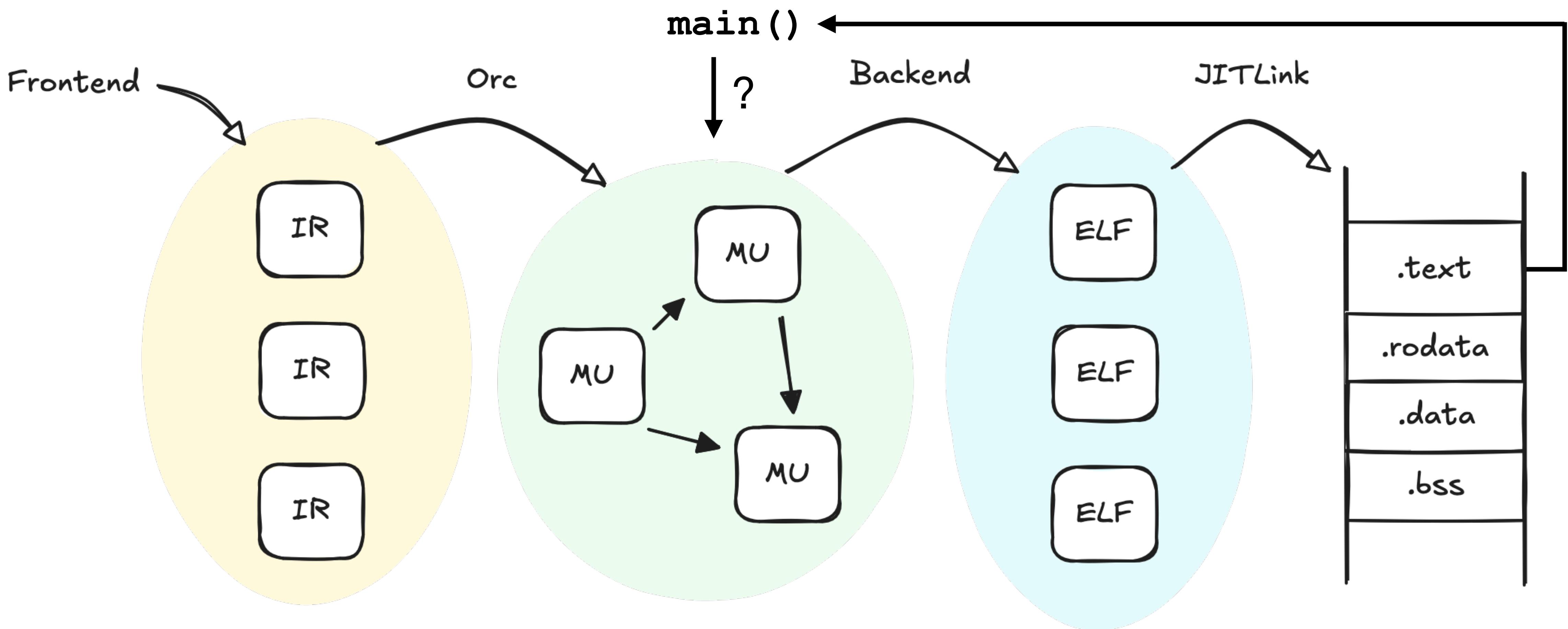
↓ repo



Stefan Gränitz / FOSDEM / 31 January 2026

# OrcJIT

In-memory on-request compilation of LLVM IR code



# OrcJIT

## Hello world example with the LLVM interpreter

```
> cat hello.c
#include <stdio.h>

int main(void) {
    printf("hello!\n");
    return 0;
}

> clang -c -emit-llvm hello.c -o hello.bc
> lli hello.bc
hello!
```

# OrcJIT

## Small C example: bzip2

```
> cd specCPU2006/original/401.bzip2/bzip2-1.0.3
> ls *.c
blocksort.c  bzip2.c  bzip2recover.c  bzlib.c  compress.c  crctable.c
decompress.c  dlltest.c  huffman.c  mk251.c  randtable.c  spewG.c  unzcrash.c

> cat Makefile | wc -l
205

> make CFLAGS="-emit-llvm"
...
```

# OrcJIT

## Building with make

```
> make CFLAGS="-emit-llvm"
clang-21 -emit-llvm -c blocksort.c
clang-21 -emit-llvm -c huffman.c
clang-21 -emit-llvm -c crctable.c
clang-21 -emit-llvm -c randtable.c
clang-21 -emit-llvm -c compress.c
clang-21 -emit-llvm -c decompress.c
clang-21 -emit-llvm -c bzlib.c
rm -f libbz2.a
llvm-ar-21 cq libbz2.a blocksort.o huffman.o crctable.o randtable.o compress.o decompress.o
bzlib.o
llvm-ar-21: error: blocksort.o: No such file or directory
make: *** [Makefile:35: libbz2.a] Error 1
> ls *.bc
blocksort.bc  bzlib.bc  compress.bc  crctable.bc  decompress.bc  huffman.bc  randtable.bc
```

# OrcJIT

Cover the rest of the way on foot

```
> clang-21 -emit-llvm -c bzip2.c
```

```
> ls *.bc
```

```
blocksort.bc  bzip2.bc   bzlib.bc   compress.bc  crctable.bc  decompress.bc  huffman.bc  
randtable.bc
```

```
> lli --extra-module=blocksort.bc \  
    --extra-module=bzlib.bc \  
    --extra-module=compress.bc \  
    --extra-module=crctable.bc \  
    --extra-module=decompress.bc \  
    --extra-module=huffman.bc \  
    --extra-module=randtable.bc bzip2.bc --compress data.txt
```

```
> ls data.*
```

```
data.txt  data.txt.bz2
```



# OrcJIT

Cover the rest of the way on foot

```
> clang-21 -emit-llvm -c bzip2.c
```

```
> ls *.bc
blocksort.bc  bzip2.bc    bzlib.bc    compress.bc   crctable.bc  decompress.bc  huffman.bc
randtable.bc
```

```
> lli --extra-module=blocksort.bc \
      --extra-module=bzlib.bc \
      --extra-module=compress.bc \
      --extra-module=crctable.bc \
      --extra-module=decompress.bc \
      --extra-module=huffman.bc \
      --extra-module=randtable.bc bzip2.bc --compress data.txt
```

We load all code on startup!

```
> ls data.*
data.txt  data.txt.bz2
```

# OrcJIT

Loads all code on startup

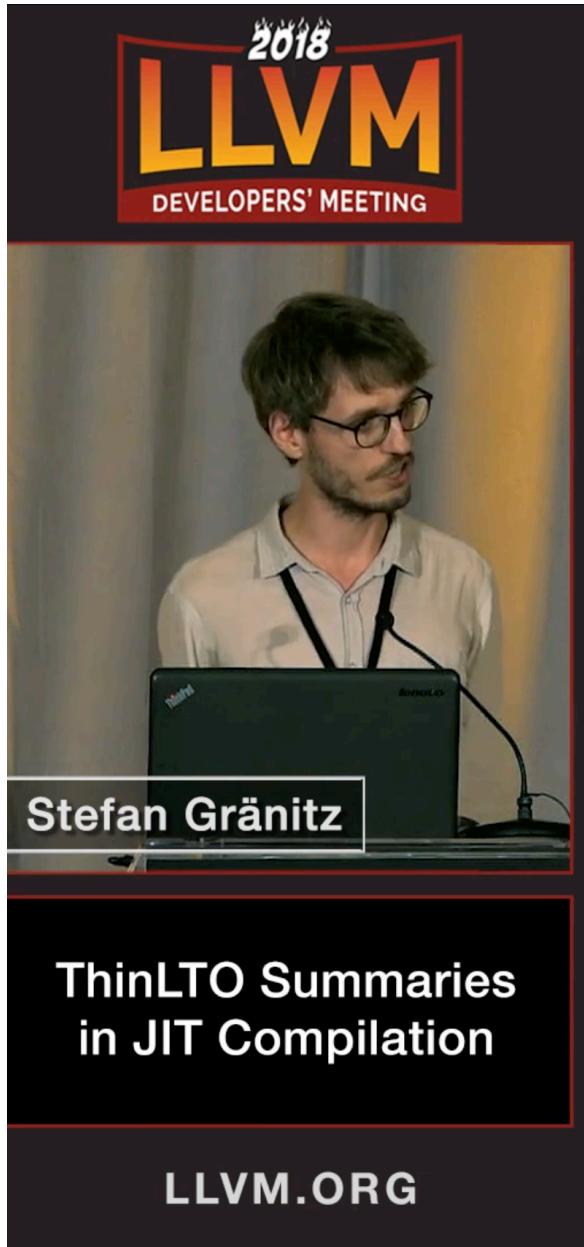
Before entering `main()`

we have to...

- Parse all symbol names
- Allocate global variables
- Initialize global variables

# OrcJIT

Loads all code on startup



Build Clang Stage1 in-memory?  
Compile less:  
Stage1 code coverage building Stage2 (Clang-7)

Functions	Translation Units
39204 / 91591	785 / 1393

... and benefit from pipelining!

LLVM.ORG

Before entering `main()`  
we have to...

- Parse all symbol names
- Allocate global variables
- Initialize global variables

}

ThinLTO Summaries



<https://www.youtube.com/watch?v=ZCnHxRhQmvs>

# OrcJIT

Loads all code on startup

Before entering `main()`  
we have to...

- Parse all symbol names
- Allocate global variables
- Initialize global variables



# OrcJIT

Loads all code on startup

Before entering `main()`  
we have to...

- Parse all symbol names
- Allocate global variables
- Initialize global variables

Status quo:  
**We load all code  
on startup!**

# OrcJIT

Cover the rest of the way on foot

We cannot just `-emit-llvm`  
for a real-world project

```
> clang-21 -emit-llvm -c bzip2.c
```

```
> ls *.bc
blocksort.bc  bzip2.bc    bzlib.bc    compress.bc   crctable.bc  decompress.bc  huffman.bc
randtable.bc
```

```
> lli --extra-module=blocksort.bc \
      --extra-module=bzlib.bc \
      --extra-module=compress.bc \
      --extra-module=crctable.bc \
      --extra-module=decompress.bc \
      --extra-module=huffman.bc \
      --extra-module=randtable.bc bzip2.bc --compress data.txt
```

```
> ls data.*
data.txt  data.txt.bz2
```

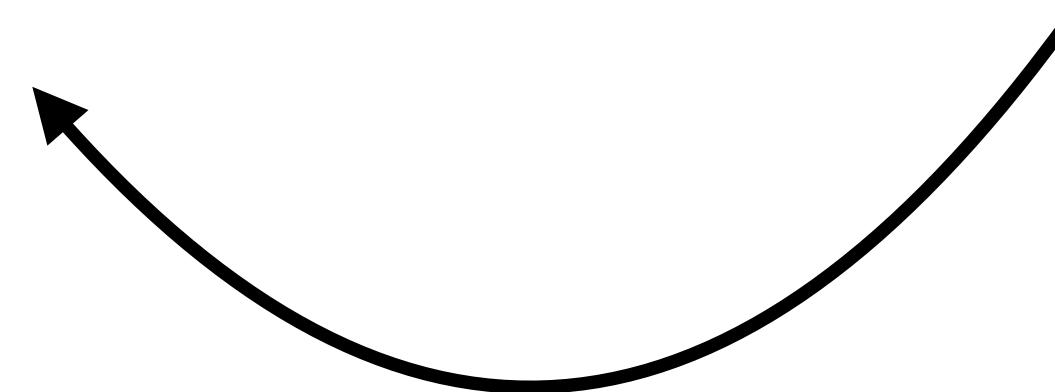
# OrcJIT

Who is using it out there?

- Julia: <https://docs.julialang.org/en/v1/devdocs/jit/>
- Common Lisp: <https://clasp-developers.github.io/news.html>
- C / C++ / Rust / Swift / Zig:



Missing build system integration



# **llvm-autojit plugin**

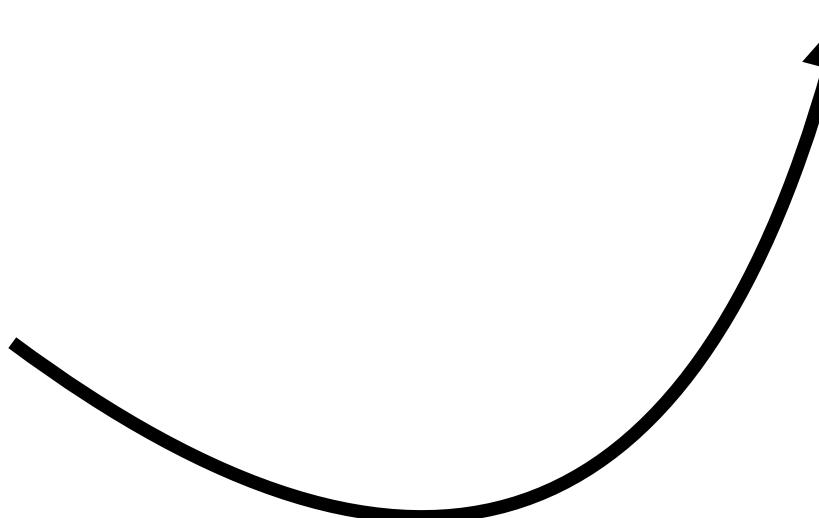
Generic OrcJIT integration across LLVM compilers

# llvm-autojit plugin

Generic OrcJIT integration across LLVM compilers



Repo: <https://github.com/weliveindetail/llvm-autojit>



# llvm-autojit plugin

Generic OrcJIT integration across LLVM compilers



## Idea:

- ▶ Hook into compiler pipeline early
- ▶ Cut out code and store it somewhere (e.g. on disk)
- ▶ Inject JIT loader
- ▶ Compile and link "shallow" binary as usual

# llvm-autojit plugin

Building bzip2 with make



```
> make CFLAGS="-fpass-plugin=/usr/lib/llvm-21/plugins/autojit.so"
clang-21 -fpass-plugin=/usr/lib/llvm-21/plugins/autojit.so -c blocksort.c
clang-21 -fpass-plugin=/usr/lib/llvm-21/plugins/autojit.so -c huffman.c
clang-21 -fpass-plugin=/usr/lib/llvm-21/plugins/autojit.so -c crctable.c
clang-21 -fpass-plugin=/usr/lib/llvm-21/plugins/autojit.so -c randtable.c
clang-21 -fpass-plugin=/usr/lib/llvm-21/plugins/autojit.so -c compress.c
clang-21 -fpass-plugin=/usr/lib/llvm-21/plugins/autojit.so -c decompress.c
clang-21 -fpass-plugin=/usr/lib/llvm-21/plugins/autojit.so -c bzlib.c
rm -f libbz2.a
llvm-ar-21 cq libbz2.a blocksort.o huffman.o crctable.o randtable.o compress.o decompress.o bzlib.o
llvm-ranlib-21 libbz2.a
clang-21 -fpass-plugin=/usr/lib/llvm-21/plugins/autojit.so -c bzip2.c
clang-21 -O0 -g -fpass-plugin=/home/ez/Develop/llvm-project-main/build-external-llvm21/install/lib/
autojit.so -Wl,-rpath=/home/ez/Develop/llvm-project-main/build-external-llvm21/install/lib -L/home/ez/
Develop/llvm-project-main/build-external-llvm21/install/lib -lautojit-runtime -rdynamic -o bzip2 bzip2.o
-L. -Wl,--whole-archive -lbz2 -Wl,--no-whole-archive
```

# llvm-autojit plugin

Works with any build system

- ▶ make
- ▶ CMake
- ▶ Cargo: requires Rust Nightly for plugin support

Find samples here:

<https://github.com/weliveindetail/llvm-autojit/actions/runs/19178131679>



# llvm-autojit plugin

Generic OrcJIT integration across LLVM compilers



## Idea:

- ▶ Hook into compiler pipeline early
- ▶ Cut out code and store it somewhere (e.g. on disk)
- ▶ Inject JIT loader
- ▶ Compile and link "shallow" binary as usual

# Ilvm-autojit transform

## Input

```
; Function Attrs: noinline nounwind optnone uwtable
define dso_local void @BZ2_compressBlock(ptr noundef %0, i8 noundef zeroext %1) #0 !dbg !180 {
    %3 = alloca ptr, align 8
    %4 = alloca i8, align 1
    store ptr %0, ptr %3, align 8
    #dbg_declare(ptr %3, !183, !DIExpression(), !184)
    store i8 %1, ptr %4, align 1
    #dbg_declare(ptr %4, !185, !DIExpression(), !186)
    %5 = load ptr, ptr %3, align 8, !dbg !187
    %6 = getelementptr inbounds nuw %struct.EState, ptr %5, i32 0, i32 17, !dbg !189
    %7 = load i32, ptr %6, align 4, !dbg !189
    %8 = icmp sgt i32 %7, 0, !dbg !190
    br i1 %8, label %9, label %63, !dbg !190
    ...
138:                                ; preds = %136, %112
    ret void, !dbg !311
}
```

# llvm-autojit transform

## Output

```
; Function Attrs: noinline nounwind optnone uwtable
define dso_local void @BZ2_compressBlock(ptr noundef %0, i8 noundef zeroext %1) #0 {
entry:
%existing_ptr = load ptr, ptr @_llvm_autojit_ptr_BZ2_compressBlock, align 8
%2 = icmp eq ptr %existing_ptr, null
br i1 %2, label %materialize, label %call
materialize:
store ptr inttoptr (i64 -4592883590450763574 to ptr), ptr @_llvm_autojit_ptr_BZ2_compressBlock, align 8
call void @_llvm_autojit_materialize(ptr @_llvm_autojit_ptr_BZ2_compressBlock)
%materialized_ptr = load ptr, ptr @_llvm_autojit_ptr_BZ2_compressBlock, align 8
br label %call

call:                                     ; preds = %materialize, %entry
%impl_ptr = phi ptr [ %existing_ptr, %entry ], [ %materialized_ptr, %materialize ]
tail call void %impl_ptr(ptr %0, i8 %1)
ret void
}
```

13853860483258788042

# llvm-autojit transform

## Output

```
@__llvm_autojit_ptr_BZ2_compressBlock = internal global ptr null
@__llvm_autojit_lazy_file = private unnamed_addr constant [49 x i8] c"/tmp/autojit_4f0e030a358521399cbcd216e6c35c26.bc\00"

@llvm.global_ctors = appending global [1 x { i32, ptr, ptr }] [{ i32, ptr, ptr } {
    i32 100, ptr @_GLOBAL__sub_I_compress.c_llvm_autojit_init, ptr null
}]

; Function Attrs: nounwind
define internal void @_GLOBAL__sub_I_compress.c_llvm_autojit_init() #1 section ".text.startup" {
entry:
    call void @_llvm_autojit_register(ptr @_llvm_autojit_lazy_file)
    ret void
}

declare void @_llvm_autojit_materialize(ptr)
declare void @_llvm_autojit_register(ptr)
```

# llvm-autojit plugin

Generic OrcJIT integration across LLVM compilers



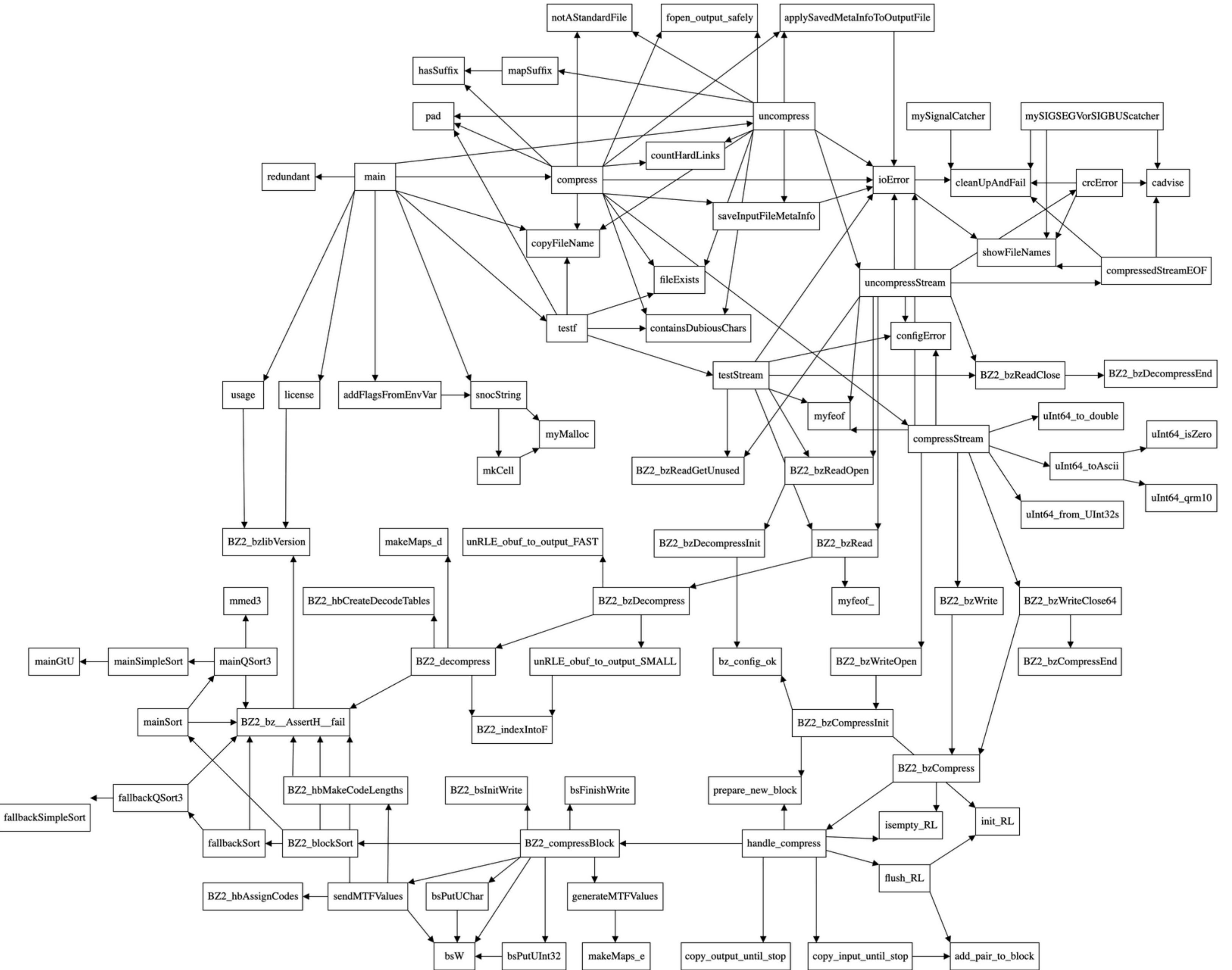
## Idea:

- ▶ Hook into compiler pipeline early
- ▶ Cut out code and store it somewhere (e.g. on disk)
- ▶ Inject JIT loader
- ▶ Compile and link "shallow" binary as usual

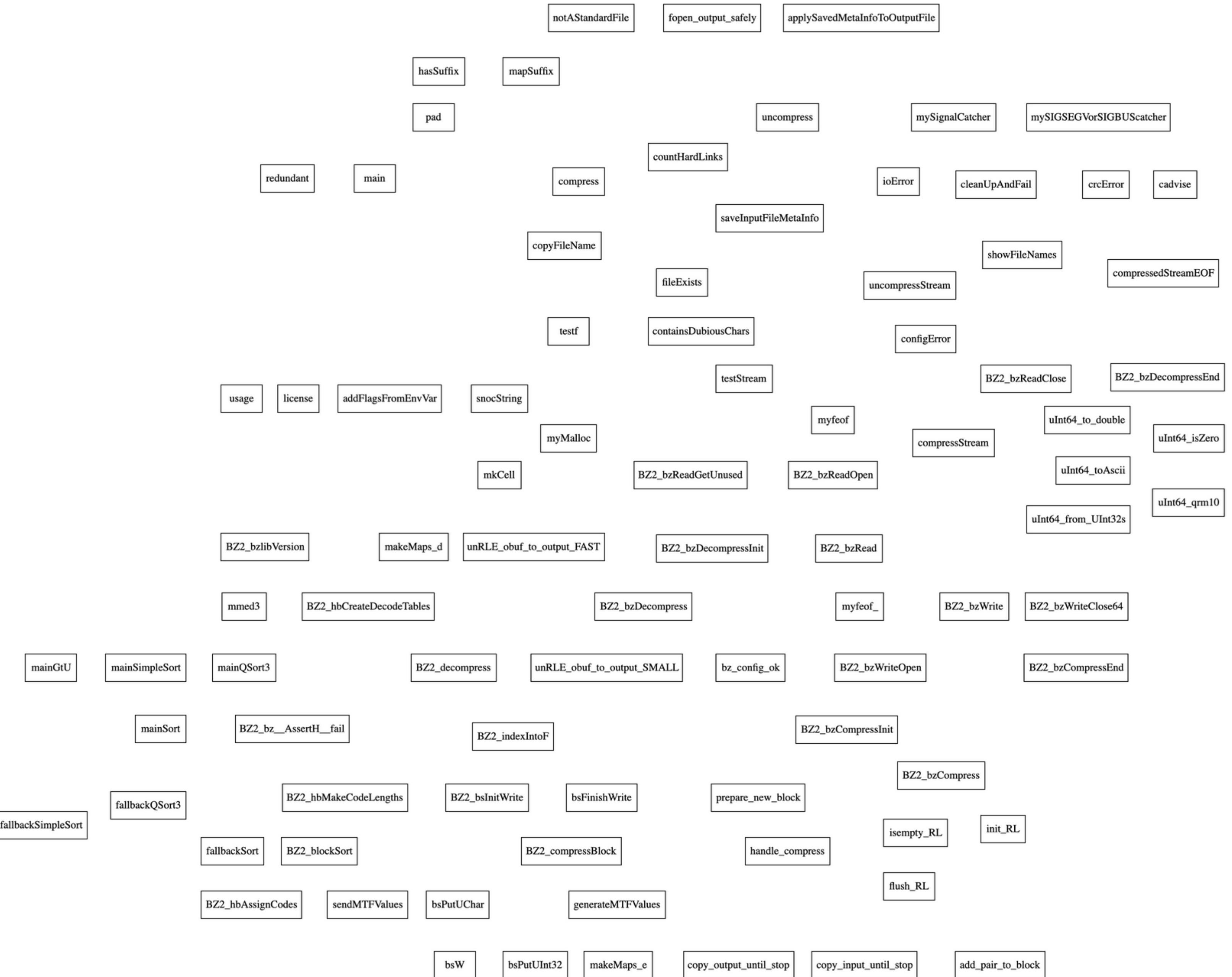
# bzip2

## regular

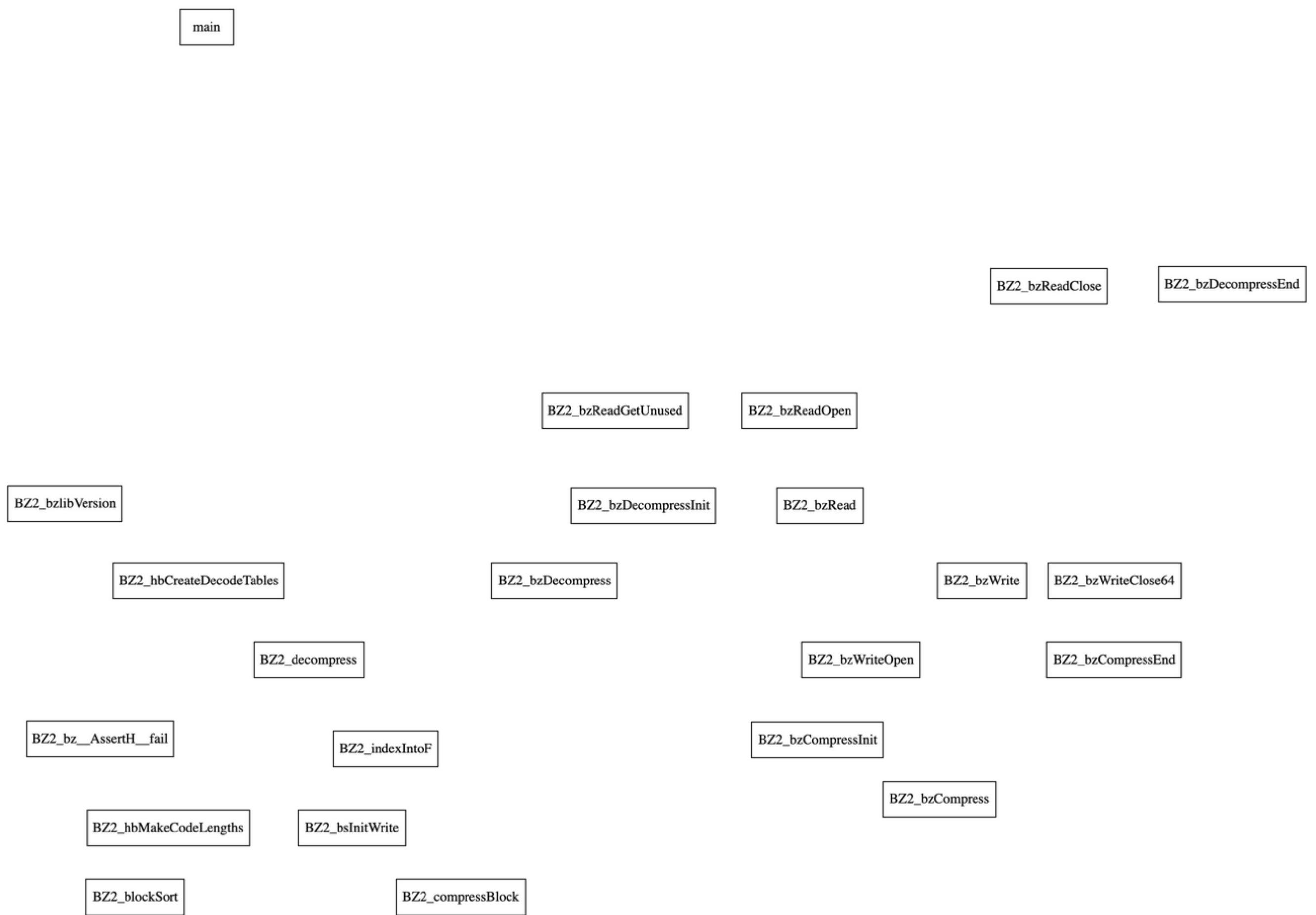
## callgraph



bzip2  
autojit  
callgraph  
(almost)



# bzip2 autojit callgraph



# bzip2 binary size

```
1-$ ls -lh build_regular/bzip2
2-rwxrwxr-x 1 ez ez 190K Nov 20 10:59 build_regular/bzip2
3
4-$ llvm-objdump -h build_regular/bzip2 > logs/bench-regular-sections.log
5-build_regular/bzip2: file format elf64-x86-64
6
7 Sections:
```

Idx	Name	Size	VMA	Type
0		00000000	0000000000000000	
1	.interp	0000001c	000000000000318	DATA
2	.note.gnu.property	00000020	000000000000338	
3	.note.gnu.build-id	00000024	000000000000358	
4	.note.ABI-tag	00000020	00000000000037c	
5	.gnu.hash	00000024	0000000000003a0	
6	.dynsym	00000468	0000000000003c8	
7	.dynstr	000001b1	000000000000830	
8	.gnu.version	0000005e	0000000000009e2	
9	.gnu.version_r	00000050	000000000000a40	
10	.rela.dyn	00000348	000000000000a90	
11	.rela.plt	00000390	000000000000dd8	
12	.init	0000001b	0000000000002000	TEXT
13	.plt	00000270	0000000000002020	TEXT
14	.plt.got	00000008	0000000000002290	TEXT
15	.text	00014993	00000000000022a0	TEXT
16	.fini	0000000d	00000000000016c34	TEXT
17	.rodata	000024fa	00000000000017000	DATA
18	.eh_frame_hdr	0000037c	000000000000194fc	DATA
19	.eh_frame	00000db0	00000000000019878	DATA
20	.init_array	00000008	000000000001bdd0	
21	.fini_array	00000008	000000000001bdd8	
22	.dynamic	000001e0	000000000001bde0	
23	.got	00000040	000000000001bf0	DATA
24	.got.plt	00000148	000000000001c000	DATA
25	.data	00000d10	000000000001c150	DATA
26	.bss	00001130	000000000001ce60	BSS
27	.comment	00000080	0000000000000000	
28	.debug_info	00004f55	0000000000000000	DEBUG
29	.debug_abbrev	000009e8	0000000000000000	DEBUG
30	.debug_line	00007e10	0000000000000000	DEBUG
31	.debug_str	000014d7	0000000000000000	DEBUG

```
1+$ ls -lh build_autojit_shlib/bzip2
2-rwxrwxr-x 1 ez ez 43K Nov 20 10:50 build_autojit_shlib/bzip2
3
4+$ llvm-objdump -h build_autojit_shlib/bzip2 > logs/bench-autojit-shlib-sections.log
5-build_autojit_shlib/bzip2: file format elf64-x86-64
6
7 Sections:
```

Idx	Name	Size	VMA	Type
0		00000000	0000000000000000	
1	.interp	0000001c	000000000000318	DATA
2	.note.gnu.property	00000020	000000000000338	
3	.note.gnu.build-id	00000024	000000000000358	
4	.note.ABI-tag	00000020	00000000000037c	
5	.gnu.hash	0000027c	0000000000003a0	
6	.dynsym	00000768	000000000000620	
7	.dynstr	0000053d	000000000000d88	
8	.gnu.version	0000009e	00000000000012c6	
9	.gnu.version_r	00000030	0000000000001368	
10	.rela.dyn	00000390	0000000000001398	
11	.rela.plt	00000030	0000000000001728	
12	.init	0000001b	0000000000002000	TEXT
13	.plt	00000030	0000000000002020	TEXT
14	.plt.got	00000008	0000000000002050	TEXT
15	.text	00000fac	0000000000002060	TEXT
16	.fini	0000000d	000000000000300c	TEXT
17	.rodata	00000201	0000000000004000	DATA
18	.eh_frame_hdr	0000016c	0000000000004204	DATA
19	.eh_frame	00000564	0000000000004370	DATA
20	.init_array	00000038	0000000000005d98	
21	.fini_array	00000008	0000000000005dd0	
22	.dynamic	00000200	0000000000005dd8	
23	.got	00000028	0000000000005fd8	DATA
24	.got.plt	00000028	0000000000006000	DATA
25	.data	00000d10	0000000000006030	DATA
26	.bss	00001248	0000000000006d40	BSS
27	.comment	00000080	0000000000000000	
28	.debug_info	000012d7	0000000000000000	DEBUG
29	.debug_abbrev	00000424	0000000000000000	DEBUG
30	.debug_line	000003f8	0000000000000000	DEBUG
31	.debug_str	0000065a	0000000000000000	DEBUG

# bzip2 binary size

```
1- $ ls -lh build_regular/bzip2
2- -rwxrwxr-x 1 ez ez 190K Nov 20 10:59 build_regular/bzip2
3
4- $ llvm-objdump -h build_regular/bzip2 > logs/bench-regular-sections.log
5- build_regular/bzip2: file format elf64-x86-64
6
7 Sections:
```

Idx	Name	Size	VMA	Type
0		00000000	0000000000000000	
1	.interp	0000001c	000000000000318	DATA
2	.note.gnu.property	00000020	000000000000338	
3	.note.gnu.build-id	00000024	000000000000358	
4	.note.ABI-tag	00000020	00000000000037c	
5	.gnu.hash	00000024	0000000000003a0	
6	.dynsym	00000468	0000000000003c8	
7	.dynstr	000001b1	000000000000830	
8	.gnu.version	0000005e	0000000000009e2	
9	.gnu.version_r	00000050	000000000000a40	
10	.rela.dyn	00000348	000000000000a90	
11	.rela.plt	00000390	000000000000dd8	
12	.init	0000001b	0000000000002000	TEXT
13	.plt	00000270	0000000000002020	TEXT
14	.plt.got	00000008	0000000000002290	TEXT
15	.text	00014993	00000000000022a0	TEXT
16	.fini	0000000d	00000000000016c34	TEXT
17	.rodata	000024fa	00000000000017000	DATA
18	.eh_frame_hdr	0000037c	000000000000194fc	DATA
19	.eh_frame	00000db0	00000000000019878	DATA
20	.init_array	00000008	000000000001bdd0	
21	.fini_array	00000008	000000000001bdd8	
22	.dynamic	000001e0	000000000001bde0	
23	.got	00000040	000000000001bf0	DATA
24	.got.plt	00000148	000000000001c000	DATA
25	.data	00000d10	000000000001c150	DATA
26	.bss	00001130	000000000001ce60	BSS
27	.comment	00000080	0000000000000000	
28	.debug_info	00004f55	0000000000000000	DEBUG
29	.debug_abbrev	000009e8	0000000000000000	DEBUG
30	.debug_line	00007e10	0000000000000000	DEBUG
31	.debug_str	000014d7	0000000000000000	DEBUG

```
1+ $ ls -lh build_autojit_shlib/bzip2
2+ -rwxrwxr-x 1 ez ez 43K Nov 20 10:50 build_autojit_shlib/bzip2
3
4+ $ llvm-objdump -h build_autojit_shlib/bzip2 > logs/bench-autojit-shlib-sections.log
5+ build_autojit_shlib/bzip2: file format elf64-x86-64
6
7 Sections:
```

Idx	Name	Size	VMA	Type
0		00000000	0000000000000000	
1	.interp	0000001c	000000000000318	DATA
2	.note.gnu.property	00000020	000000000000338	
3	.note.gnu.build-id	00000024	000000000000358	
4	.note.ABI-tag	00000020	00000000000037c	
5	.gnu.hash	0000027c	0000000000003a0	
6	.dynsym	00000768	000000000000620	
7	.dynstr	0000053d	000000000000d88	
8	.gnu.version	0000009e	00000000000012c6	
9	.gnu.version_r	00000030	0000000000001368	
10	.rela.dyn	00000390	0000000000001398	
11	.rela.plt	00000030	0000000000001728	
12	.init	0000001b	0000000000002000	TEXT
13	.plt	00000030	0000000000002020	TEXT
14	.plt.got	00000008	0000000000002050	TEXT
15	.text	00000fac	0000000000002060	TEXT
16	.fini	0000000d	000000000000300c	TEXT
17	.rodata	00000201	0000000000004000	DATA
18	.eh_frame_hdr	0000016c	0000000000004204	DATA
19	.eh_frame	00000564	0000000000004370	DATA
20	.init_array	00000038	0000000000005d98	
21	.fini_array	00000008	0000000000005dd0	
22	.dynamic	00000200	0000000000005dd8	
23	.got	00000028	0000000000005fd8	DATA
24	.got.plt	00000028	0000000000006000	DATA
25	.data	00000d10	0000000000006030	DATA
26	.bss	00001248	0000000000006d40	BSS
27	.comment	00000080	0000000000000000	
28	.debug_info	000012d7	0000000000000000	DEBUG
29	.debug_abbrev	00000424	0000000000000000	DEBUG
30	.debug_line	000003f8	0000000000000000	DEBUG
31	.debug_str	0000065a	0000000000000000	DEBUG

# bzip2 data section

8	Idx	Name	Size	VMA	Type
9	0		00000000	0000000000000000	
10	1	.interp	0000001c	000000000000318	DATA
11	2	.note.gnu.property	00000020	000000000000338	
12	3	.note.gnu.build-id	00000024	000000000000358	
13	4	.note.ABI-tag	00000020	00000000000037c	
14-	5	.gnu.hash	00000024	0000000000003a0	
15-	6	.dynsym	00000468	0000000000003c8	
16-	7	.dynstr	000001b1	000000000000830	
17-	8	.gnu.version	0000005e	0000000000009e2	
18-	9	.gnu.version_r	00000050	000000000000a40	
19-	10	.rela.dyn	00000348	000000000000a90	
20-	11	.rela.plt	00000390	000000000000dd8	
21	12	.init	0000001b	0000000000002000	TEXT
22	13	.plt	00000270	0000000000002020	TEXT
23	14	.plt.got	00000008	0000000000002290	TEXT
24	15	.text	00014993	00000000000022a0	TEXT
25	16	.fini	0000000d	00000000000016c34	TEXT
26	17	.rodata	000024fa	00000000000017000	DATA
27	18	.eh_frame_hdr	0000037c	000000000000194fc	DATA
28	19	.eh_frame	00000db0	00000000000019878	DATA
29	20	.init_array	00000008	0000000000001bdd0	
30	21	.fini_array	00000008	0000000000001bdd8	
31	22	.dynamic	000001e0	0000000000001bde0	
32	23	.got	00000040	0000000000001bf0	DATA
33	24	.got.plt	00000148	0000000000001c000	DATA
34	25	.data	00000d10	0000000000001c150	DATA
35	26	.bss	00001130	0000000000001ce60	BSS
36	27	.comment	00000080	0000000000000000	
37	28	.debug_info	00004f55	0000000000000000	DEBUG
38	29	.debug_abbrev	000009e8	0000000000000000	DEBUG
39	30	.debug_line	00007e10	0000000000000000	DEBUG
40	31	.debug_str	000014d7	0000000000000000	DEBUG
41	32	.debug_addr	00000ca0	0000000000000000	DEBUG
42	33	.debug_line_str	00000181	0000000000000000	DEBUG
43	34	.debug_rnglists	0000001b	0000000000000000	DEBUG
44	35	.debug_str_offsets	00001028	0000000000000000	DEBUG
45	36	.symtab	00001470	0000000000000000	
46	37	.strtab	00000c2c	0000000000000000	
47	38	.shstrtab	0000018a	0000000000000000	

8	Idx	Name	Size	VMA	Type
9	0		00000000	0000000000000000	
10	1	.interp	0000001c	000000000000318	DATA
11	2	.note.gnu.property	00000020	000000000000338	
12	3	.note.gnu.build-id	00000024	000000000000358	
13	4	.note.ABI-tag	00000020	00000000000037c	
14+	5	.gnu.hash	0000027c	0000000000003a0	
15+	6	.dynsym	00000768	000000000000620	
16+	7	.dynstr	0000053d	000000000000d88	
17+	8	.gnu.version	0000009e	00000000000012c6	
18+	9	.gnu.version_r	00000030	0000000000001368	
19+	10	.rela.dyn	00000390	0000000000001398	
20+	11	.rela.plt	00000030	0000000000001728	
21	12	.init	0000001b	0000000000002000	TEXT
22	13	.plt	0000030	0000000000002020	TEXT
23	14	.plt.got	00000008	0000000000002050	TEXT
24	15	.text	00000fac	0000000000002060	TEXT
25	16	.fini	0000000d	000000000000300c	TEXT
26	17	.rodata	00000201	0000000000004000	DATA
27	18	.eh_frame_hdr	0000016c	0000000000004204	DATA
28	19	.eh_frame	00000564	0000000000004370	DATA
29	20	.init_array	00000038	0000000000005d98	
30	21	.fini_array	00000008	0000000000005dd0	
31	22	.dynamic	00000200	0000000000005dd8	
32	23	.got	00000028	0000000000005fd8	DATA
33	24	.got.plt	00000028	0000000000006000	DATA
34	25	.data	00000d10	0000000000006030	DATA
35	26	.bss	0001248	0000000000006d40	BSS
36	27	.comment	00000080	0000000000000000	
37+	28	.debug_info	000012d7	0000000000000000	DEBUG
38+	29	.debug_abbrev	00000424	0000000000000000	DEBUG
39+	30	.debug_line	000003f8	0000000000000000	DEBUG
40+	31	.debug_str	0000065a	0000000000000000	DEBUG
41+	32	.debug_addr	00000198	0000000000000000	DEBUG
42+	33	.debug_line_str	0000017f	0000000000000000	DEBUG
43+	34	.debug_str_offsets	00000484	0000000000000000	DEBUG
44+	35	.symtab	00000de0	0000000000000000	
45+	36	.strtab	000009fa	0000000000000000	
46+	37	.shstrtab	0000017a	0000000000000000	

# OrcJIT

Loads all code on startup

Before entering `main()`  
we have to...

- Parse all symbol names
- Allocate global variables
- Initialize global variables

~~Status quo:  
We load ~~all~~ code  
on startup!~~

# Ilvm-autojit benchmarks

<https://github.com/weliveindetail/Ilvm-autojit#benchmark>

```
> ./bzip2.sh --runs=10
Benchmark 401.bzip2 (shlib) in 10 runs
Compile-time regular:
  In 10 runs mean (min/max) time in seconds was: 0.944 (0.708 / 1.235)
Compile-time AutoJIT:
  In 10 runs mean (min/max) time in seconds was: 0.744 (0.537 / 1.018)
```

Binary sizes:

Regular: 190 kb  
AutoJIT: 45 kb

Run-time regular:

```
In 3 runs mean (min/max) time in seconds was: 2.611 (2.485 / 2.760)
323c74b2d7815a0b22979f45a93323e0  bzip2/outputs/data1_regular.txt.bz2
c3c5e912092b78a8a53faa34e9d7494e  bzip2/outputs/data2_regular.txt.bz2
```

Run-time AutoJIT:

```
In 3 runs mean (min/max) time in seconds was: 2.745 (2.604 / 2.889)
323c74b2d7815a0b22979f45a93323e0  bzip2/outputs/data1_autojit.txt.bz2
c3c5e912092b78a8a53faa34e9d7494e  bzip2/outputs/data2_autojit.txt.bz2
```

# Ilvm-autojit runtimes in poc release

<https://github.com/weliveindetail/Ilvm-autojit/releases/tag/v0.1>

```
> tar -C /usr/local -xzvf llvm-autojit-21_v0.1_x86-64.tar.gz
./bin/autojitd
./lib/libautojit_static-x86_64.a } static out-of-process runtime
./lib/libautojit-runtime.so.21.1 } dynamic in-process runtime
./lib/libautojit-runtime.so
./lib/autojit.so - pass plugin
```

# Ilvm-autojit plugin

A novel approach for OrcJIT at scale

- Generic plugin for LLVM-based compilers and all build systems
- Cut out IR code at compile-time while keeping ABI intact
  - ✓ Reduce binary size and compile-time
  - ✓ Incremental builds avoid relink, as long as we don't touch ABI
  - ✓ Mix autojit objects with regular ones
- Global variables remain in static executable
  - ✓ Achieve true lazy loading
  - ✓ Support link-once symbols efficiently



# llvm-autojit plugin

Generic OrcJIT integration across LLVM compilers



<https://github.com/weliveindetail/llvm-autojit>