LLVM JIT: Upcoming Challenges and Opportunities

JIT-loading `clang`

JIT-loading `clang`

(no lazy compilation, just linking objects and archives)

JIT-loading `clang`

(no lazy compilation, just linking objects and archives)

```
% time llvm-jitlink <clang> --args -S -o - hello-world.c
47.32s user 30.21s system 534% cpu 14.517 total
```

JIT-loading `clang`

(no lazy compilation, just linking objects and archives)

```
% time llvm-jitlink <clang> --args -S -o - hello-world.c
47.32s user 30.21s system 534% cpu 14.517 total
```

- Was ~30s. 2x speedup from local-block-deps algorithm rewrite (~8 hours work)

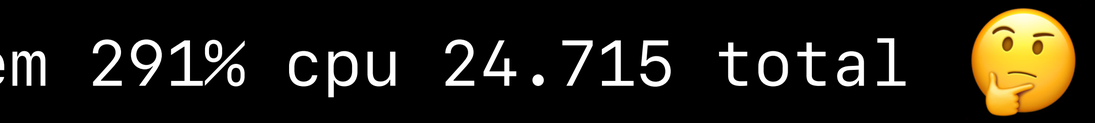
JIT-loading `clang`

(no lazy compilation, just linking objects and archives)

```
% time llvm-jitlink <clang> --args -S -o - hello-world.c
47.32s user 30.21s system 534% cpu 14.517 total
```

- Was ~30s. 2x speedup from local-block-deps algorithm rewrite (~8 hours work)
- Let's enable laziness:

```
45.23s user 26.82s system 291% cpu 24.715 total
```



JIT-loading `clang`

(no lazy compilation, just linking objects and archives)

```
% time llvm-jitlink <clang> --args -S -o - hello-world.c
47.32s user 30.21s system 534% cpu 14.517 total
```

- Was ~30s. 2x speedup from local-block-deps algorithm rewrite (~8 hours work)
- Let's enable laziness:

```
45.23s user 26.82s system 291% cpu 24.715 total
```

- Poor utilization, let's enable speculation:

```
117.19s user 35.87s system 572% cpu 26.732 total
```

JIT-loading `clang`

(no lazy compilation, just linking objects and archives)

```
% time llvm-jitlink <clang> --args -S -o - hello-world.c
47.32s user 30.21s system 534% cpu 14.517 total
```

- Was ~30s. 2x speedup from local-block-deps algorithm rewrite (~8 hours work)
- Let's enable laziness:

```
45.23s user 26.82s system 291% cpu 24.715 total
```

- Poor utilization, let's enable speculation:

```
117.19s user 35.87s system 572% cpu 26.732 total
```

- Why? Open question — attach a profiler and have at it!

ORC (On Request Compilation) Users

Xcode Previews PostgreSQL Julia

Mojo Jank Clasp `clang-repl` ...

ORC (On Request Compilation) Users

Xcode Previews PostgreSQL Julia

Mojo Jank Clasp `clang-repl` ...

- Some projects still on MCJIT:

LLDB Numba (llvmlite)

ORC (On Request Compilation) Users

Xcode Previews PostgreSQL Julia

Mojo Jank Clasp `clang-repl` ...

- Some projects still on MCJIT:

LLDB Numba (llvmlite)

Help projects migrate to ORC

Migration Blockers

Migration Blockers

- API
  - Stabilize LLJIT interface
  - Improve defaults, convenience APIs
  - Reorganize library
  - Expand C API

Migration Blockers

- API
  - Stabilize LLJIT interface
  - Improve defaults, convenience APIs
  - Reorganize library
  - Expand C API

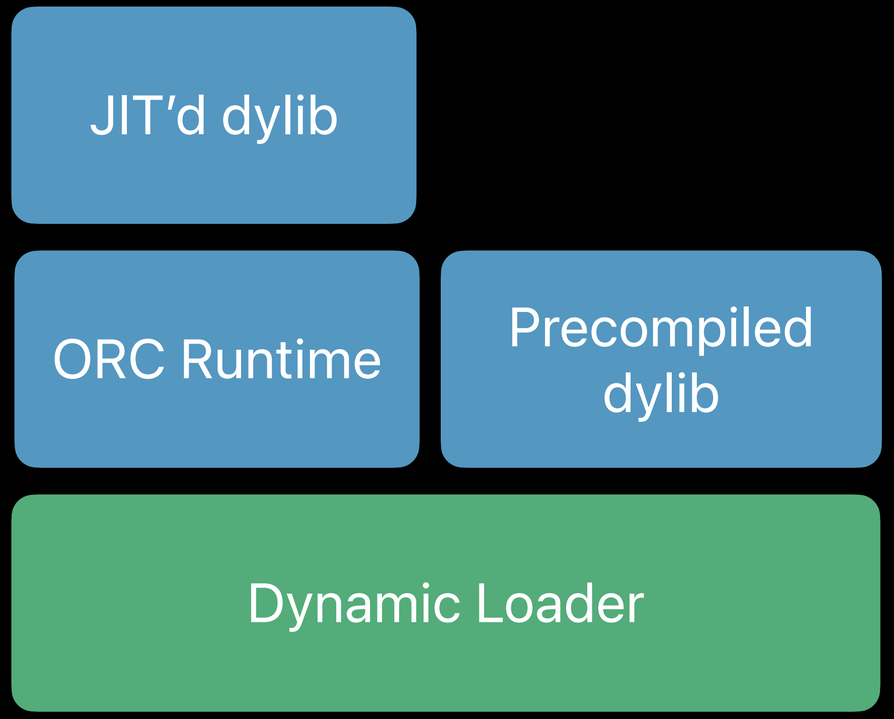
- Target Support
  - Improve Windows Support
    (Need a maintainer!)
    - MCJIT-style `dllimport` handling
    - Structured Exception Handling on by default
  - JITLink backends
    - e.g. Windows/aarch64

Integration and Tooling

- Debugger Support
  - Works, but off by default due to cost
  - Protocol redesign — pass debug info on disk, symbol table in memory
- Crash Reporting
  - General solution for symbolication

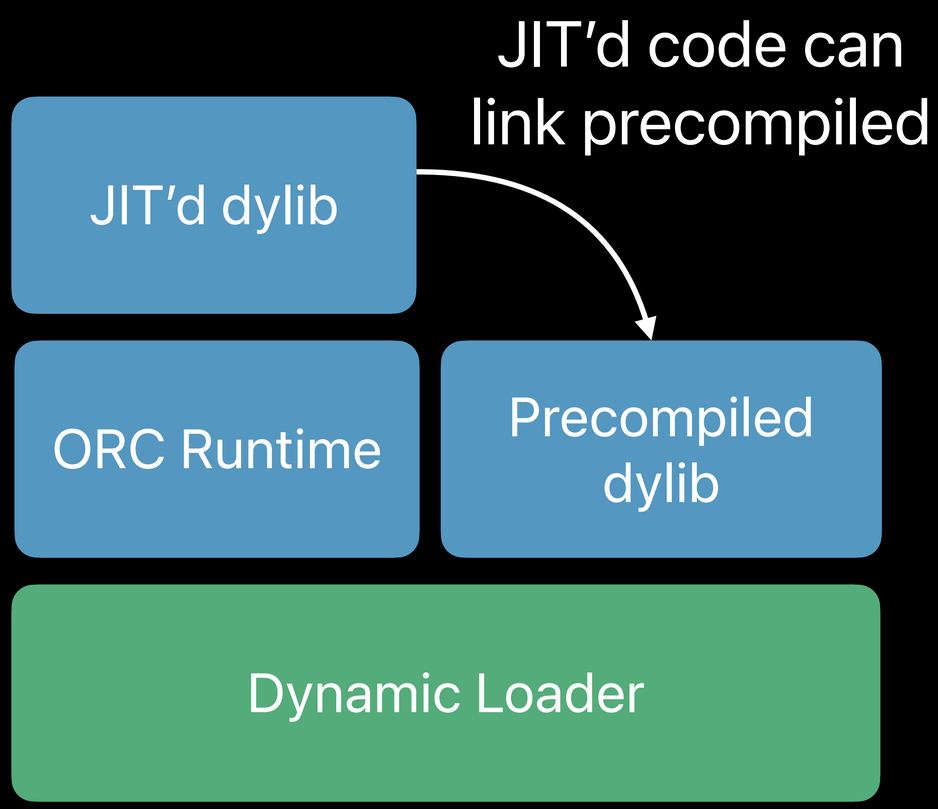
Integration and Tooling

Dynamic Loader Emulation



Integration and Tooling

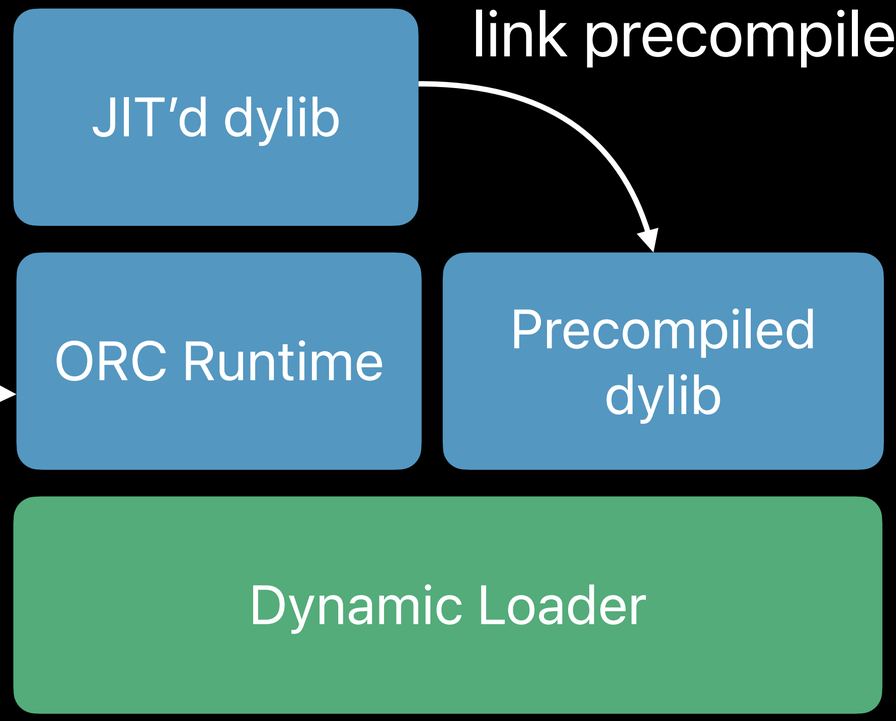
Dynamic Loader Emulation



Integration and Tooling

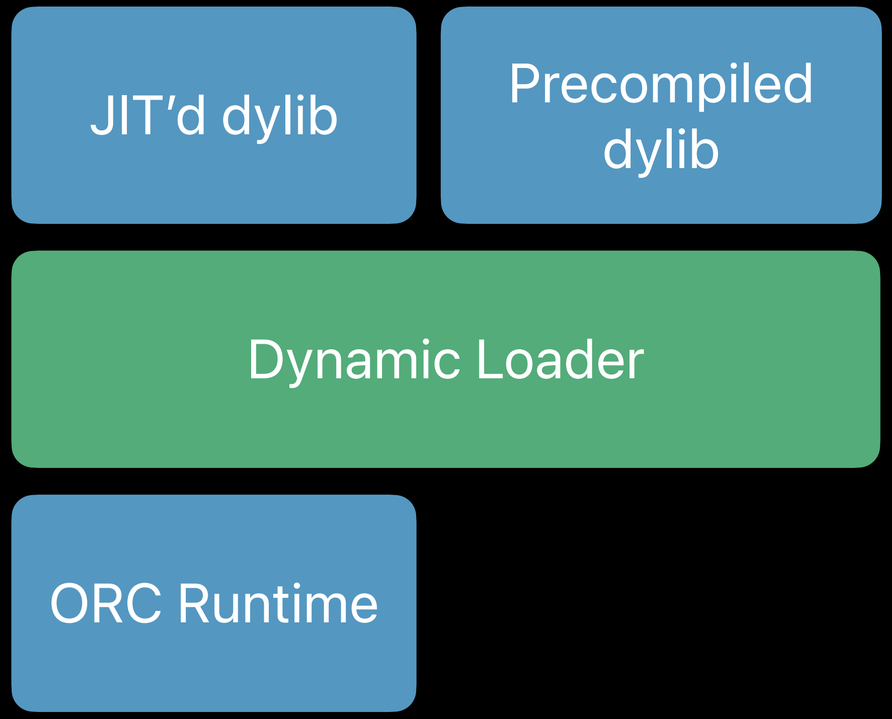
Dynamic Loader Emulation

ORC Runtime emulates Dynamic Loader APIs (`dlopen`, `dlsym`, etc.)



Integration and Tooling

Dynamic Loader *Integration*

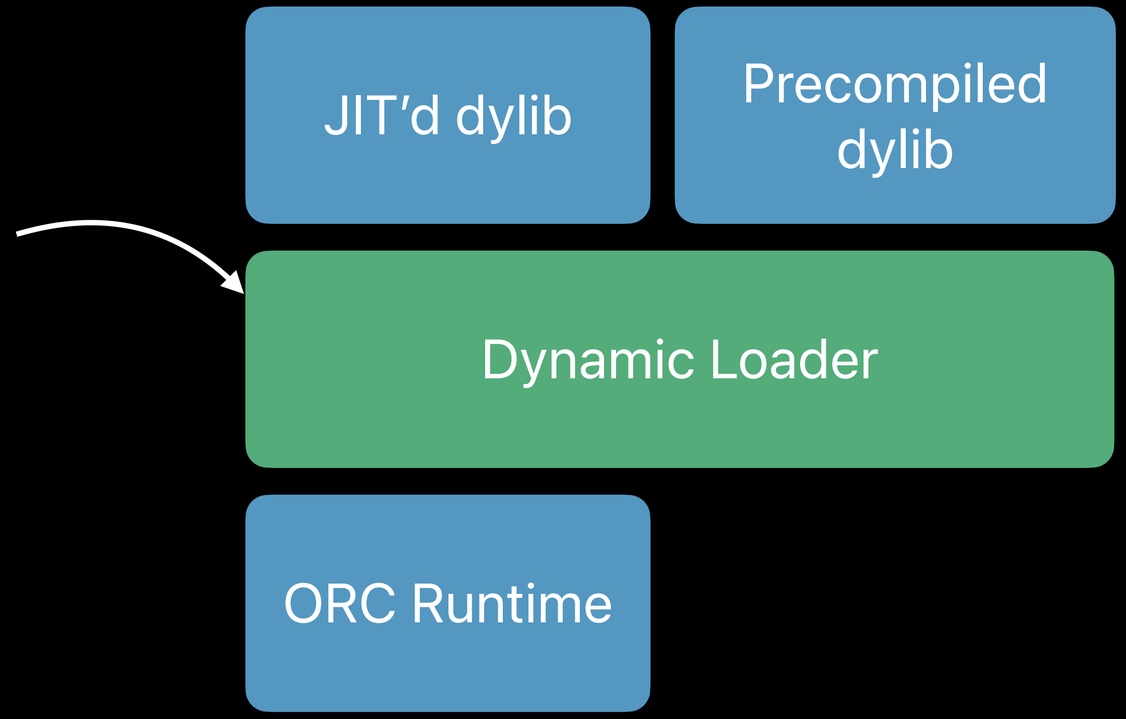


Integration and Tooling

Dynamic Loader *Integration*

JIT-aware Dynamic Loader

Calls to `dlopen`, `dlsym`, etc. forwarded to ORC runtime for JIT'd code

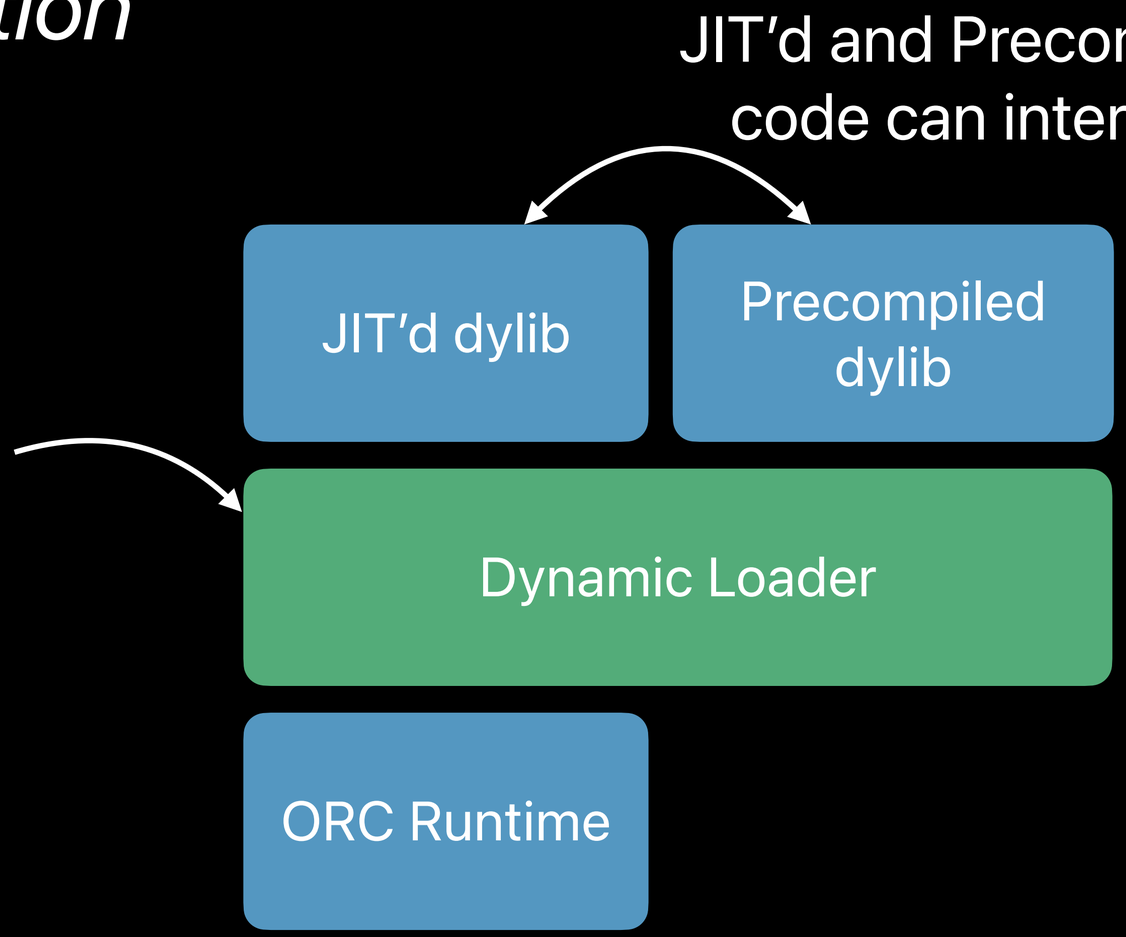


Integration and Tooling

Dynamic Loader *Integration*

JIT-aware Dynamic Loader

Calls to `dlopen`, `dlsym`, etc. forwarded to ORC runtime for JIT'd code

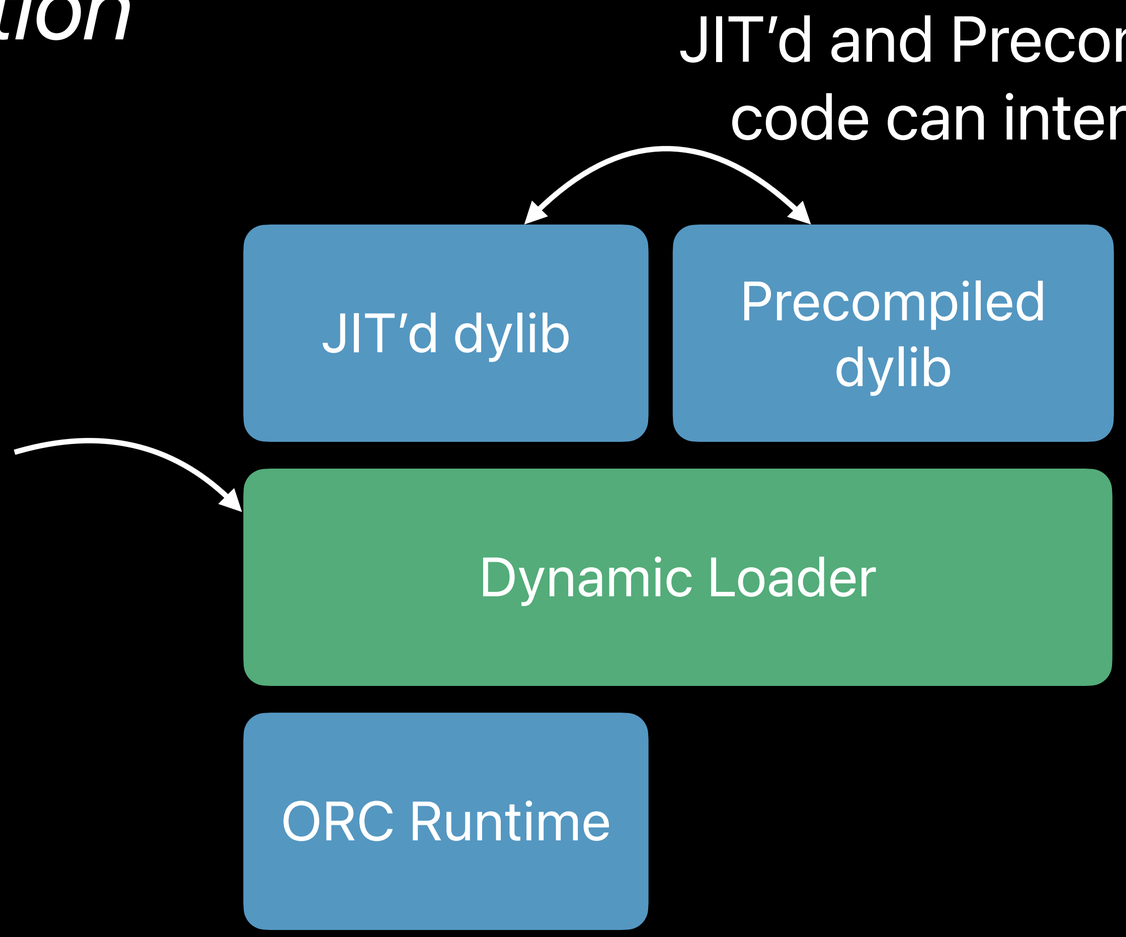


Integration and Tooling

Dynamic Loader *Integration*

JIT-aware Dynamic Loader

Calls to `dlopen`, `dlsym`, etc. forwarded to ORC runtime for JIT'd code



See "JIT-Loading Arbitrary Programs"

https://llvm.org/devmtg/2025-10/slides/technical_talks/hames.pdf

The Clang Interpreter

The Clang Interpreter

- Library for incremental compilation and execution using clang

```
clang-repl> printf("hello, world!\n");
hello, world!
```

The Clang Interpreter

- Library for incremental compilation and execution using clang

```
clang-repl> printf("hello, world!\n");
hello, world!
```

- Powers Xeus-cpp (Jupyter), cppyy (runtime C++/python bindings), Jank, ...

The Clang Interpreter

- Library for incremental compilation and execution using clang

```
clang-repl> printf("hello, world!\n");
hello, world!
```

- Powers Xeus-cpp (Jupyter), cppyy (runtime C++/python bindings), Jank, ...
  - Tutorial tomorrow: Hands-on Using Clang as a library

The Clang Interpreter

- Library for incremental compilation and execution using clang

```
clang-repl> printf("hello, world!\n");
hello, world!
```

- Powers Xeus-cpp (Jupyter), cppyy (runtime C++/python bindings), Jank, ...
  - Tutorial tomorrow: Hands-on Using Clang as a library
- Open design questions, performance opportunities

The Clang Interpreter

- Library for incremental compilation and execution using clang

```
clang-repl> printf("hello, world!\n");
hello, world!
```

- Powers Xeus-cpp (Jupyter), cppyy (runtime C++/python bindings), Jank, ...
  - Tutorial tomorrow: Hands-on Using Clang as a library
- Open design questions, performance opportunities
- `clang-repl` UX opportunities: crash recovery, default include paths, ...

More Open Opportunities

More Open Opportunities

- JIT code performance
  - Reoptimization, PGO-for-JIT, devirtualization, deoptimization for debugging, ...

More Open Opportunities

- JIT code performance
  - Reoptimization, PGO-for-JIT, devirtualization, deoptimization for debugging, …
- Memory management
  - Default allocator still stripes:
  - Should group by permissions:

More Open Opportunities

- JIT code performance
  - Reoptimization, PGO-for-JIT, devirtualization, deoptimization for debugging, …
- Memory management
  - Default allocator still stripes:
  - Should group by permissions:

- New ORC runtime: `llvm-project/orc-rt`
  - From scratch rewrite — memory, error handling, platforms, lifetime mgmt, …

More Open Opportunities

- JIT code performance
  - Reoptimization, PGO-for-JIT, devirtualization, deoptimization for debugging, …
- Memory management
  - Default allocator still stripes:
  - Should group by permissions:

- New ORC runtime: `llvm-project/orc-rt`
  - From scratch rewrite — memory, error handling, platforms, lifetime mgmt, …
- Many more…

Get Involved

- Check out the umbrella issue for this talk: https://llvm.org/PR189354
- Join #jit on the LLVM Discord
- Attend the monthly JIT Office Hours
  - Upcoming: Thursday 11:00am UTC, April 30th, 2026
- Come and chat at the conference!