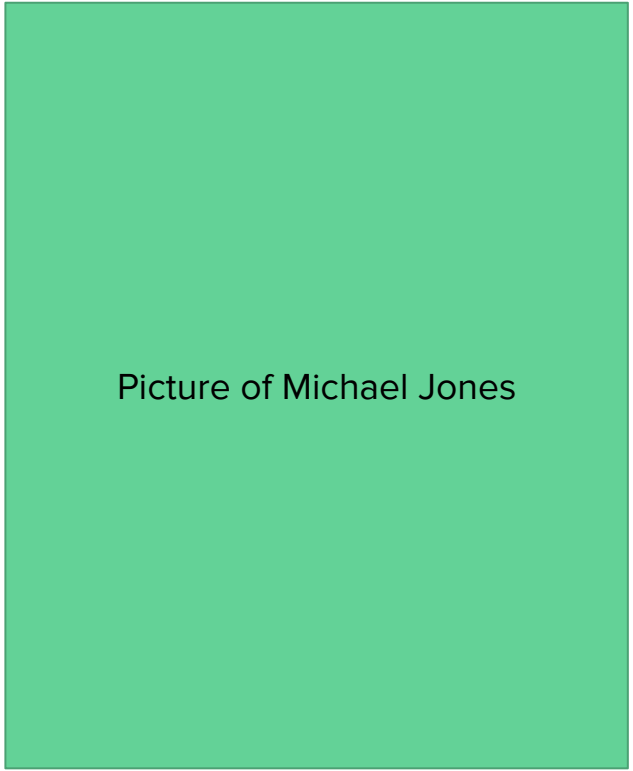
Climbing the Ladder of Complete

LLVM-libc Past and Future

Who Am I?

Michael Jones

- Me Google 5 years
- Me libc guy
- Me have biiiig plan



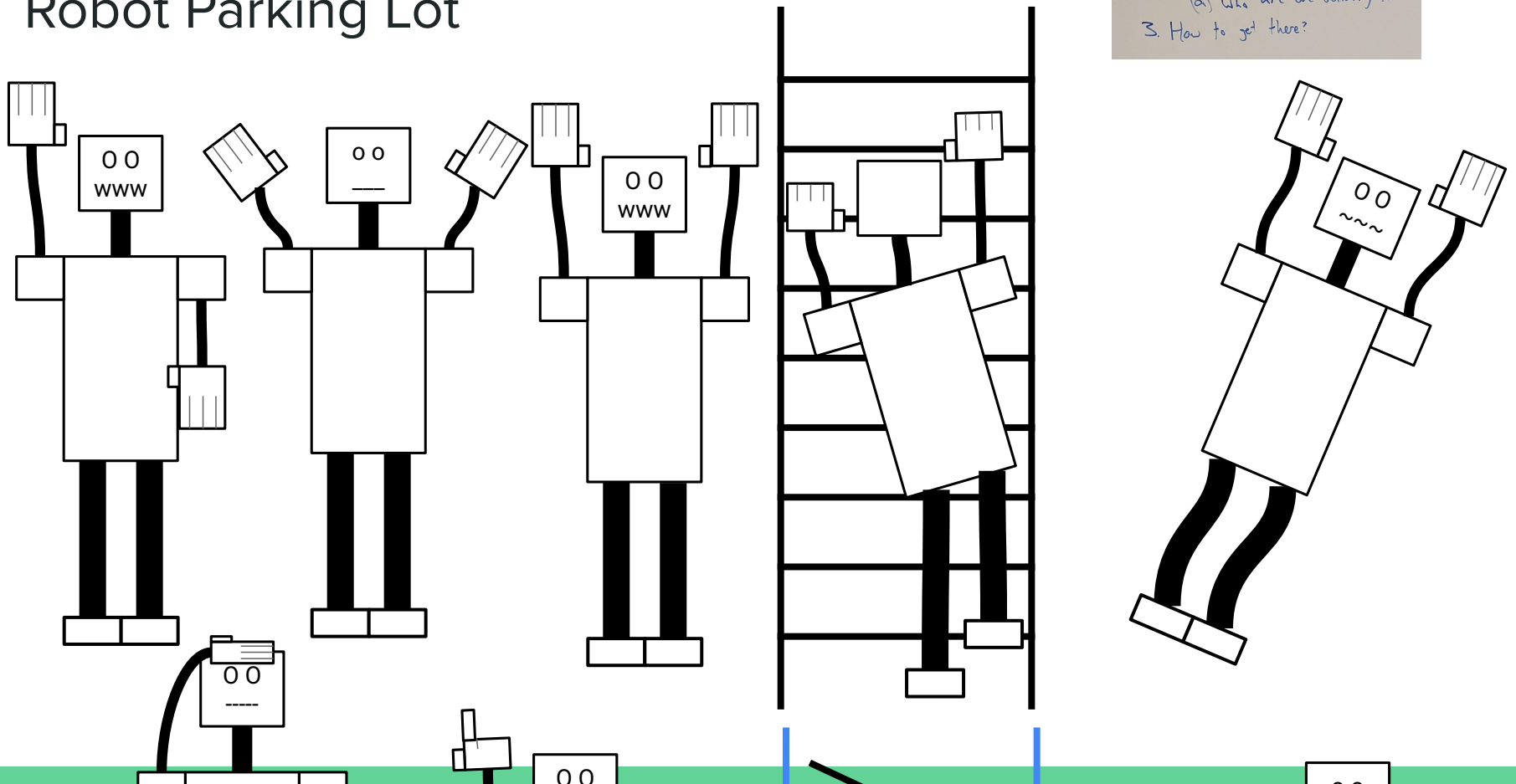
Who Am I?

Michael Jones

- At Google for 5 years
- Started right out of college
  - Learned a lot from the LLVM community
- Lead Maintainer of LLVM-libc

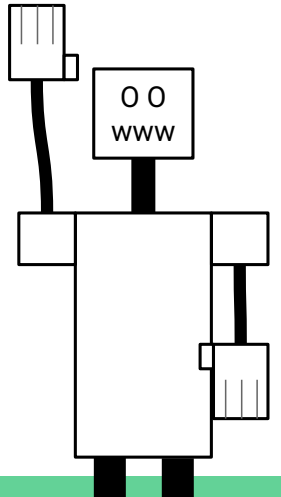
Robot Parking Lot

1. What: strategy going forward
2. Why (1)?
   (a) Who are we building for?
3. How to get there?



Presentation Outline

1. 2019 Proposal for LLVM-libc
2. Who are our users?
3. Updated LLVM-libc design
4. What do we do now?



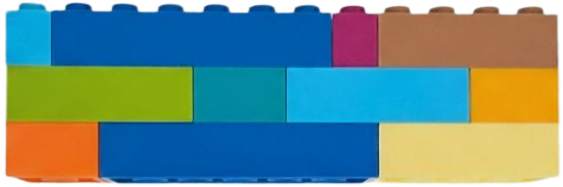
2019 Proposal for LLVM-libc:

Guiding Principles:

1. Libc "as a library"
2. Support Static linking
3. Standards as guidelines
4. Careful with vendor extensions
5. Exemplar of LLVM tooling

Not Focuses:

1. Dynamic Loading
2. More architectures

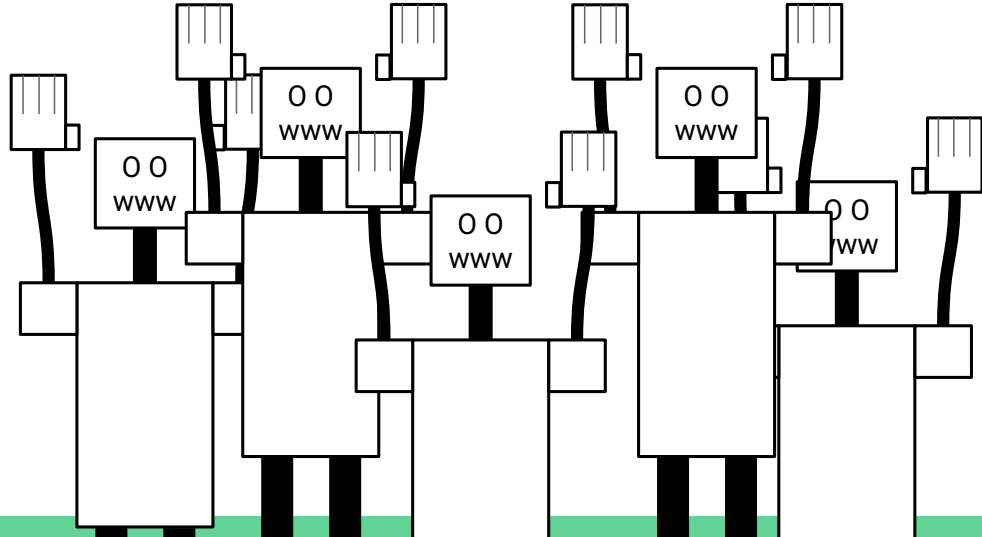


“The project should mesh with the "as a library" philosophy of the LLVM project: even though "the C Standard Library" is nominally "a library," most implementations are, in practice, quite monolithic.”

- Siva Chandra, 2019

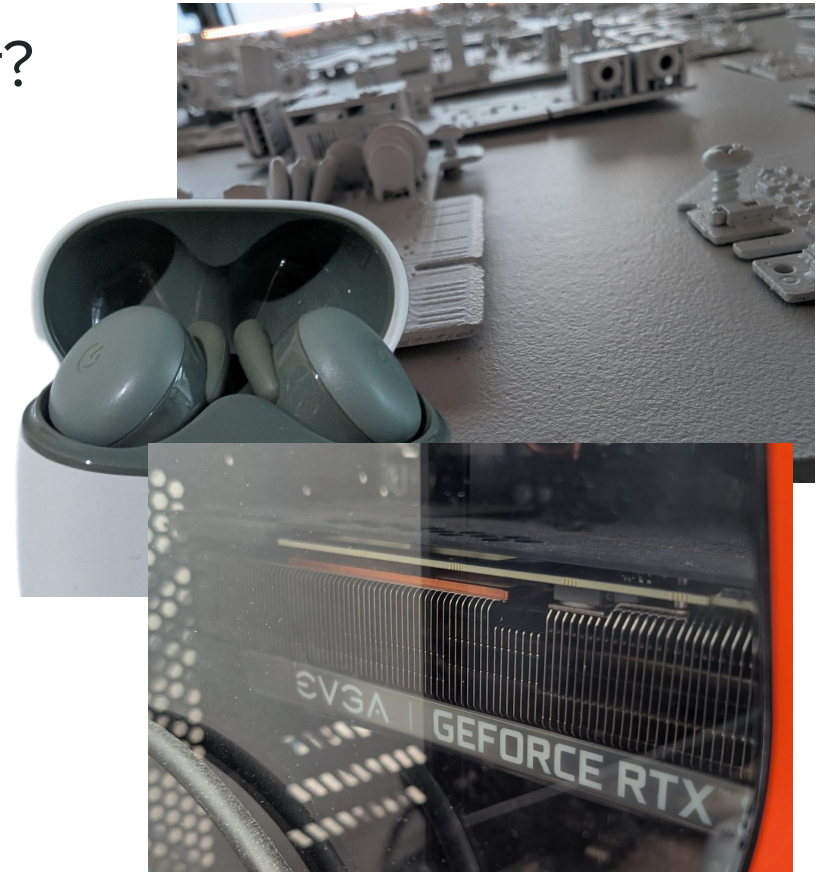
Where are we now?

- >1000 functions implemented
- 6 Processor Architectures Supported
- ~50 people with >10 commits
- ~220 people with at least 1 commit
- 11 maintainers



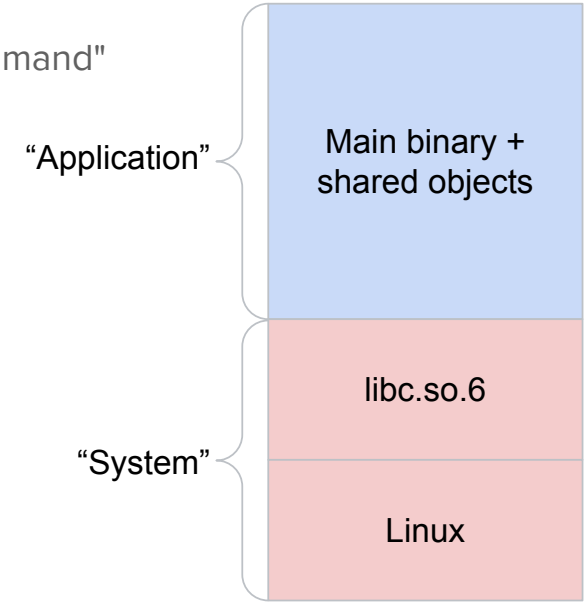
What do people use LLVM-libc for?

- Known Environments
  - Containerized servers (e.g. Google)
  - Embedded devices
  - GPUs
- Libc sources as a library
  - Hand-in-hand
    - libc++
    - OpenMP
    - clang (soon)
  - External
    - Bionic (Android's libc)
    - Fuchsia
    - Emscripten (Available experimentally)

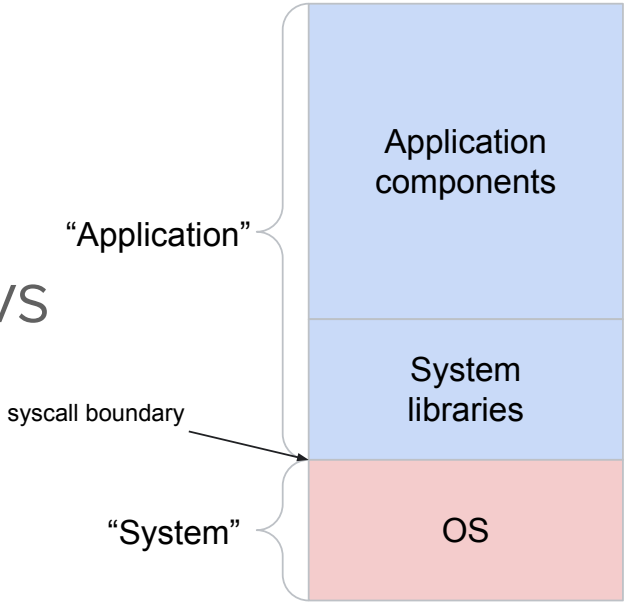


What do they need?

- Libc in the application
  - Static Linking
  - Header Library
  - "Runtimes on demand"

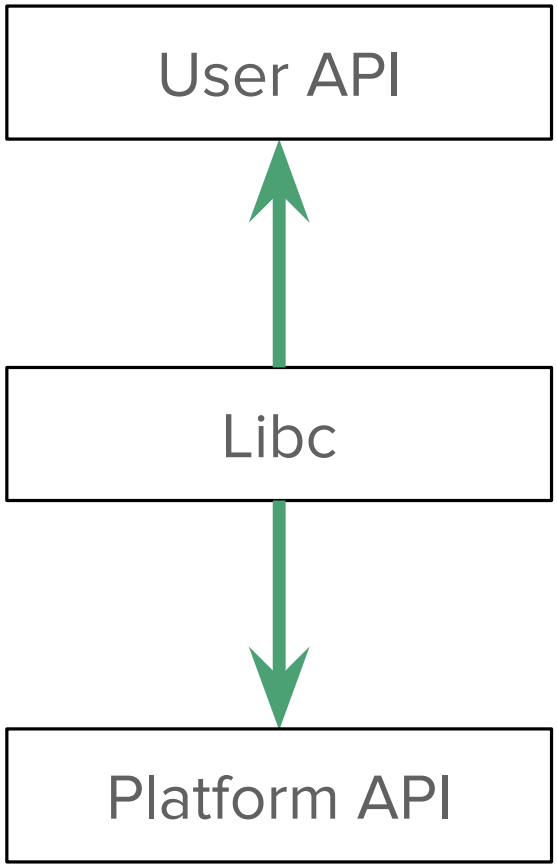


VS



What do they need?

- Libc in the application
  - Static Linking
  - Header Library
  - "Runtimes on demand"
- Portability
  - Libc that's easy to port
  - Same API across different hardware



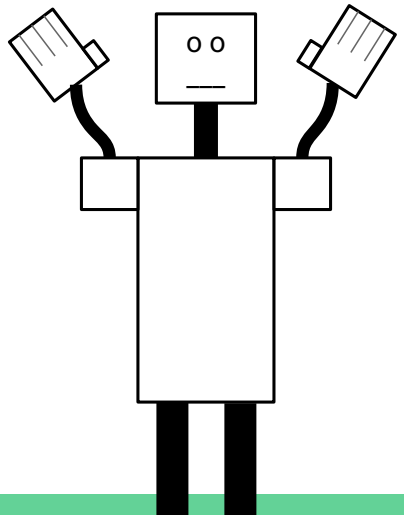
What do they need?

- Libc in the application
  - Static Linking
  - Header Library
  - "Runtimes on demand"
- Portability
  - Libc that's easy to port
  - Same API across different hardware
- Quality
  - Performance
  - code size
- Customizability
  - No one size fits all

What is lower priority?

- Dynamic Loading
- ABI stability
- Legacy Misfeatures

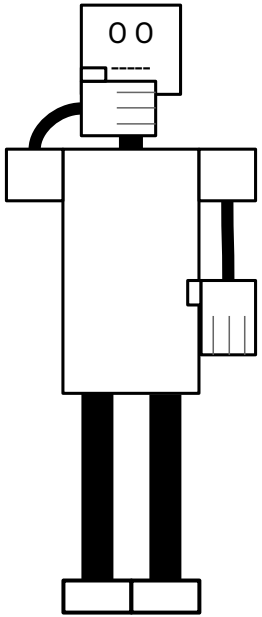
Locales as implicit arguments



Updated LLVM-libc design

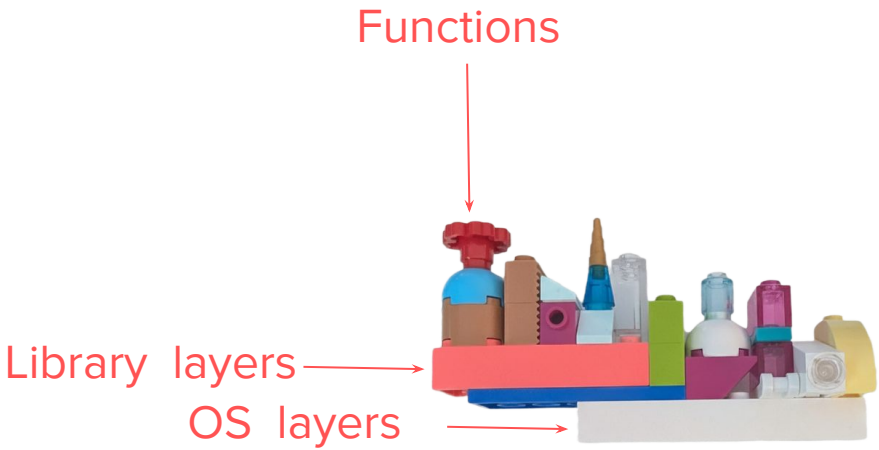
Guiding Principles:

- Modularity
- Multiplatform
- Community Oriented



Modularity

- Functions are independent
  - Vertical Modularity
- APIs to the OS level are generic
  - Horizontal Modularity
- Modules are reusable
  - Using libc as a library
- Modules are replaceable
  - E.g. downstream vendors optimizing



Multiplatform

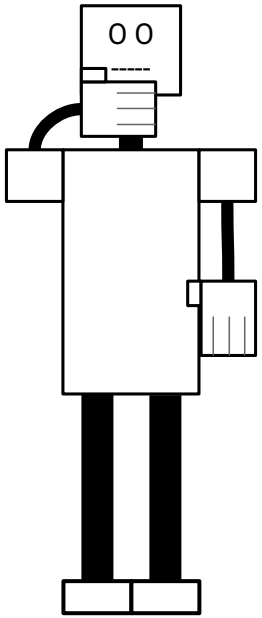
- Code is in C++, not assembly
- Platform specific code is clearly separated
- Most code is Platform independent
- LLVM-libc has one frontend, many backends

```
FILE* fopen(path, mode) {
    fd = open(path);
    return fdopen(fd, mode);
}
```

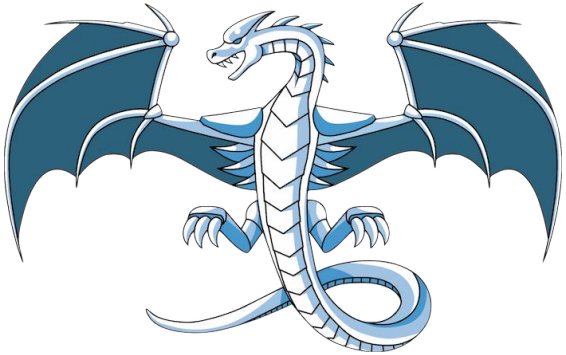
Community Oriented

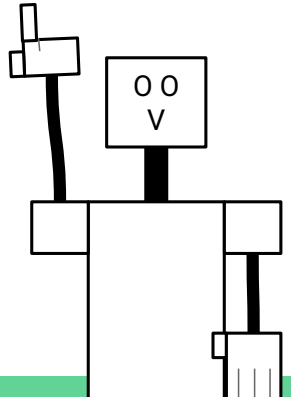
- A library doesn't appear, it's made by a community.
- A community that is welcoming to newcomers is one where people will join.
- A community that is friendly, respectful, and can handle disagreement is one where people will stay.
- A project with new opportunities is one where people will grow.

What's the next step?



LLVM-libc Production Ready For Clang 2026





Why Clang?

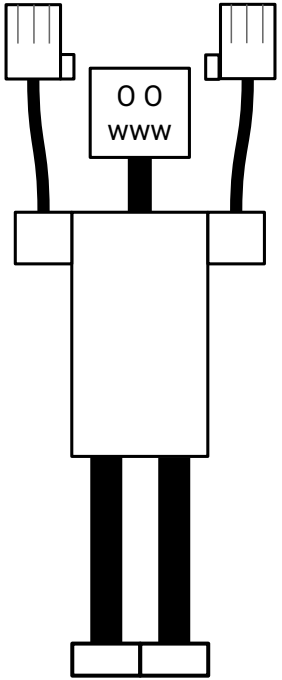
Good stress test of LLVM-libc

Real, production program

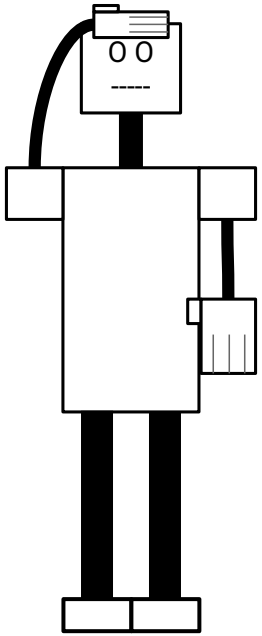
Widely available

Lets LLVM be self hosting

It's exciting!



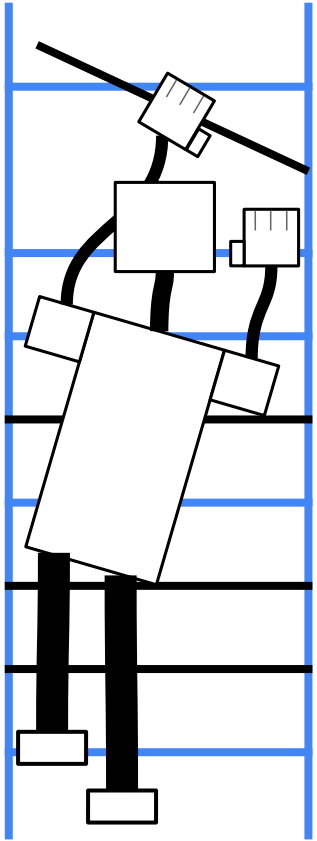
How do we get there?



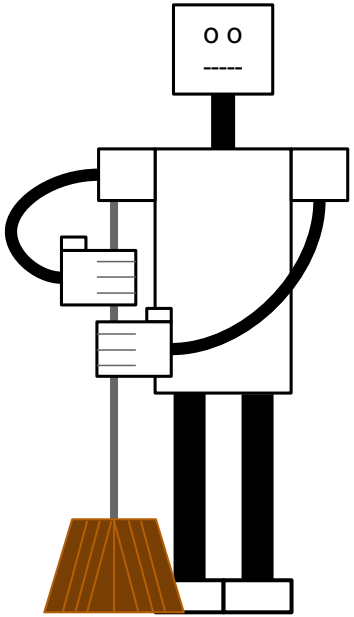
Test each module

Overlay where possible

Set up LLVM-libc/clang buildbot



Refactor and Modularize



Clean up technical debt

End of 2025

- [x] Present Strategy
- [ ] Design Cleanups

Early 2026

- ❑ Implement Cleanups
- ❑ Add functions for Clang
- ❑ Set up Clang/LLVM-libc bot

End of 2026

- ❑ Polish For Clang
- ❑ Complete sets
- ❑ Set new goal

We can do this!

(Questions?)