# cymbl: To -jInfinity & Beyond

**William S. Moses**

Kevin Kwok

wmoses@mit.edu

June 3, 2021

# Compilation Bottlenecks

- As software proliferates in all parts of life, the amount of code in the world has grown exponentially
    - As of the 2015, Google alone had more than 9 million source code files (>2 billion LOC)[1]
- Compiling code is a bottleneck for development, testing, and publication of software
- Most compilation tasks are highly parallel (many individual files) but practically limited by the number of cores on your machine
- Most builds unnecessarily repeat existing work
    - Everyone building the same existing package
    - Development is incremental — typically few files are modified in a given patch

[1] Rachel Potvin and Josh Levenberg. 2016. Why Google stores billions of lines of code in a single repository. Commun. ACM 59, 7 (July 2016), 78–87. DOI:https://doi.org/10.1145/2854146

# Ideal Remote Compilation

- Drop in replacement without rewriting the codebase (e.g. “it just works”)
- Infinite parallelism by offloading compilation to remote machines
- Cache equivalent compilation tasks rather than recomputing

# Existing Remote Compilation Tools

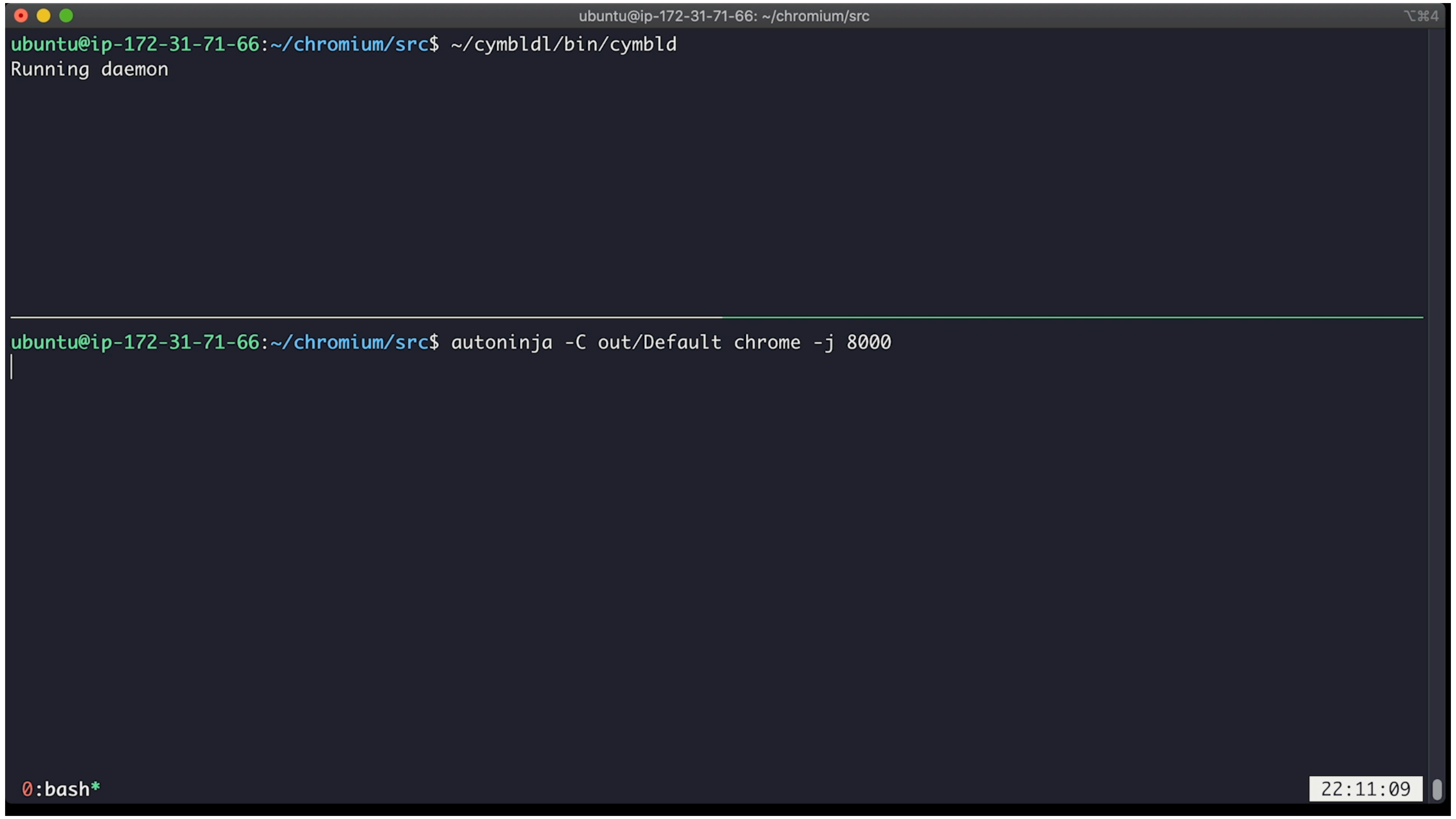
| | Compatibility | Parallelism | Caching |
|---|---|---|---|
| Bazel | ✗ Must use build system | ✗ Requires user cluster* | ? Per-codebase caching |
| DistCC | ? Models compile command | ✗ Requires user cluster | ✗ Limited or no caching |
| Goma | ? Models compile command | ✗ Requires user cluster | ? Per-codebase caching |
| gg | ✗ Models all build commands | ✓ On-demand compute | ? Per-invocation caching |

## cymbl

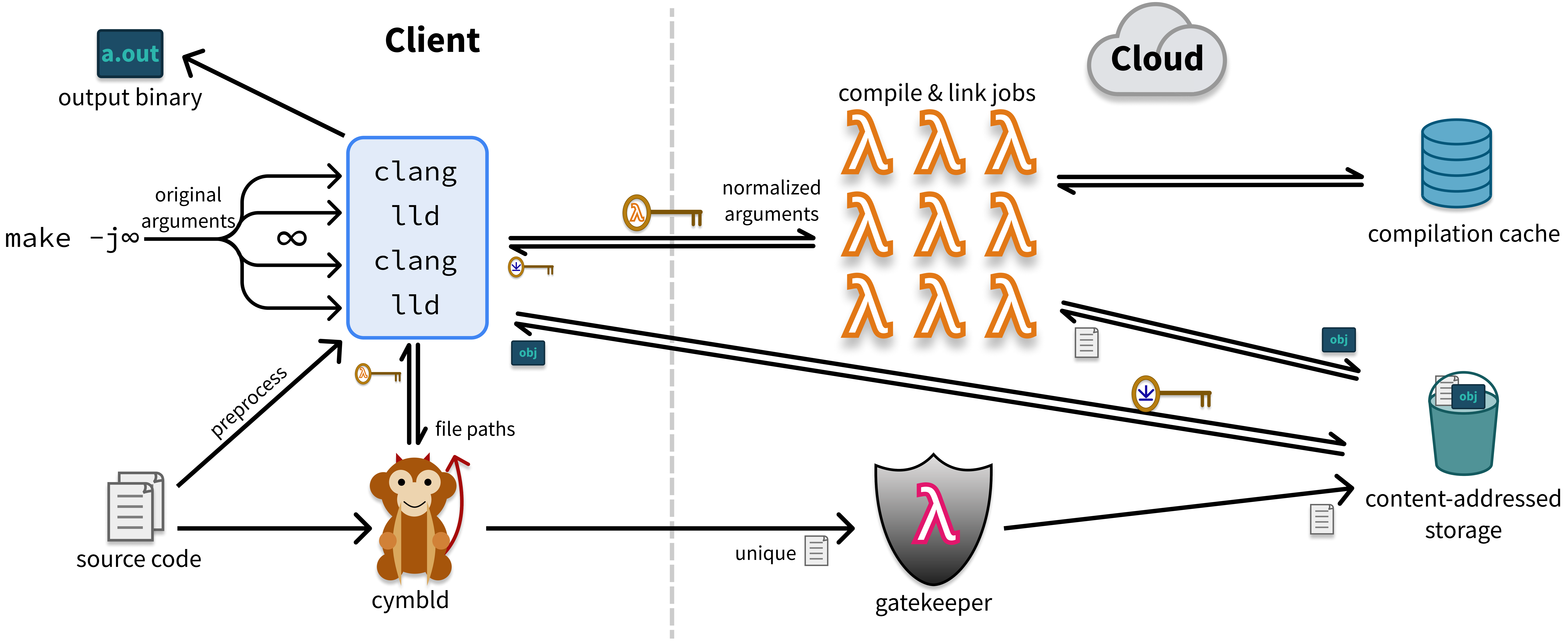
- Idea: Integrate remote execution into the compiler itself
  - Usable in any existing build system & “model” will always be perfect
  - Much more effective cache as the compiler has all the relevant information to normalize builds
  - Merging remote execution and the compiler results in much more efficient execution, reducing both latency and total build time
- Leverages cloud functions to provide infinite parallelism without requiring the user to maintain infrastructure and without gg’s requirement to model all commands
- Reduces 21-hour Chrome build down to a few minutes

# Drop-in Replacement

- After downloading Cymbl, change the default compiler to use Cymbl instead of default
- When building, set desired parallelism and let it run!

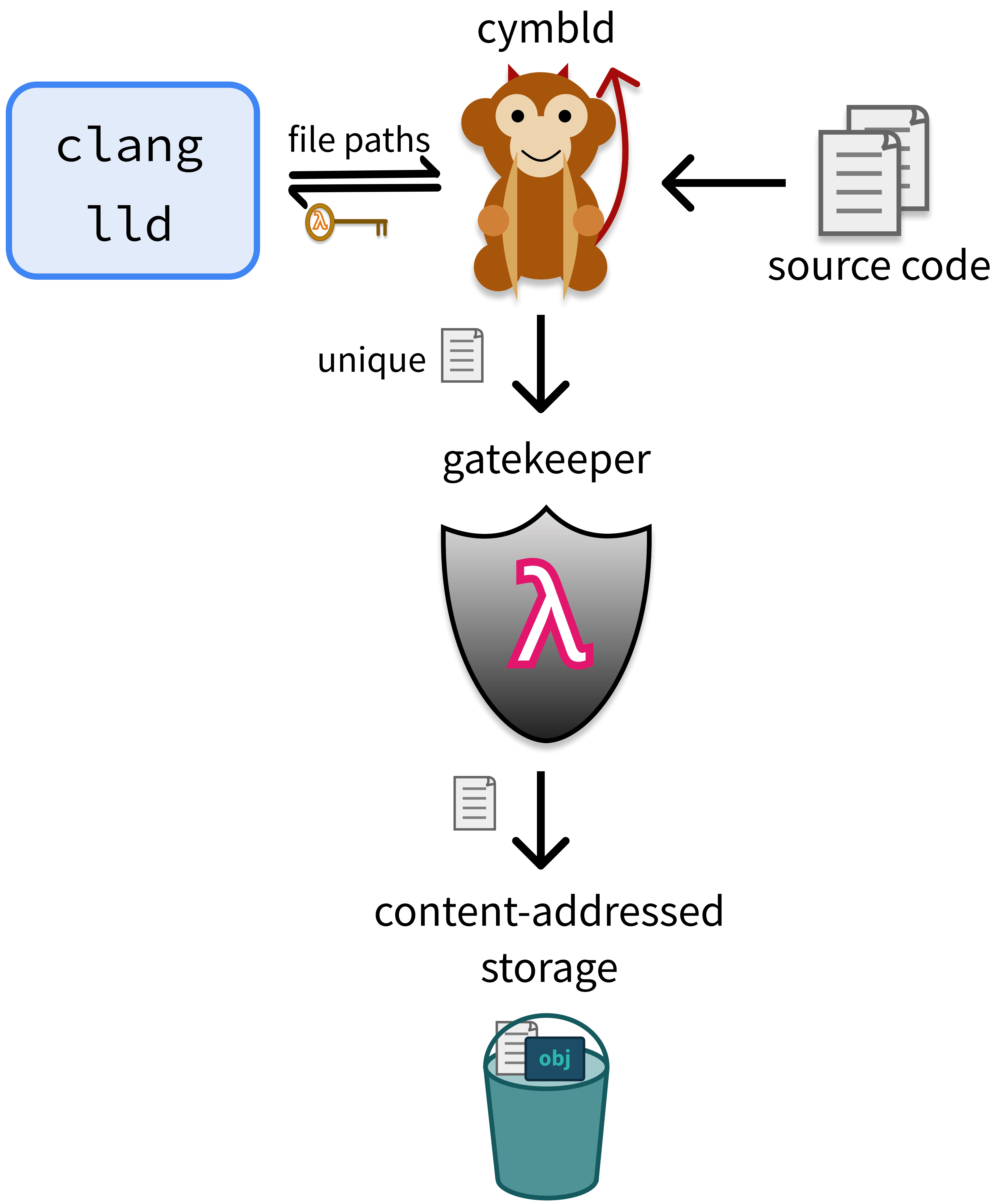


# Cymbl Design



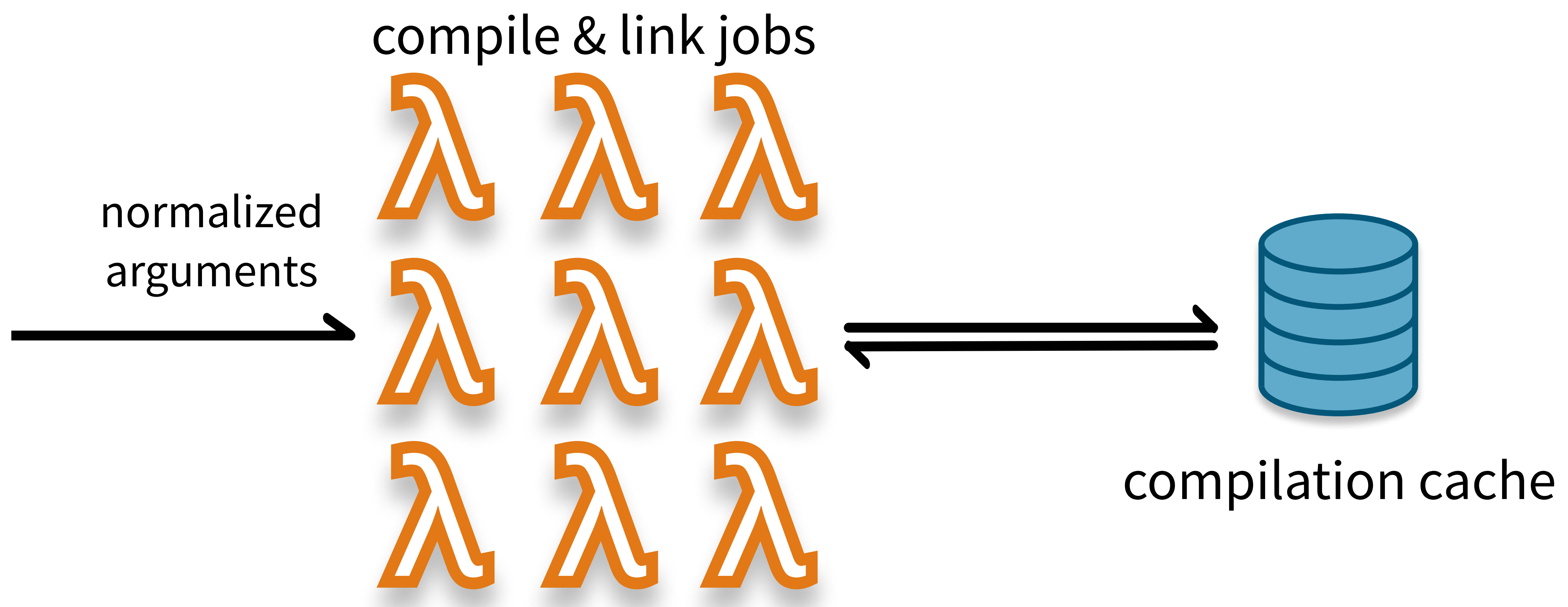
# Cymbl Daemon (cymbld)

- Many compilation tasks share the same dependencies, so to avoid duplicate uploads, file uploading is handled by a shared daemon process (cymbld)
- clang and lld processes send dependency file paths to cymbld through IPC.
- cymbld hashes, dedups, and batches before querying the server for cache misses
- cymbld uploads files and notifies clang/lld when dependencies have been uploaded and provides credentials for invoking lambdas



# Caching

- Ensure Deterministic Builds
    - Rewrite all “time of build” macros to be a fixed constant for determinism
    - All files used are explicitly passed by hash
- Normalize tasks for better cache hits
- When executing a task, first check it exists inside the cache and if so immediately return the result



# Task Normalization by Preprocessing Source

```
clang -x objective-c -target arm64-apple-ios10.0 -DDEBUG=1
-DOBJC_OLD_DISPATCH_PROTOTYPES=0 -DBUILD_ID=fadb4ca184dcb4680 -isysroot /
Applications/Xcode.app/Contents/Developer/Platforms/iPhoneOS.platform/
Developer/SDKs/iPhoneOS14.2.sdk -iquote /Users/wmoses/Library/Developer/Xcode/
DerivedData/UIViewPropertyAnimatorObjCSample-gmyxiqyiqqtmgfbeqgqiuwfodewt/
Build/Intermediates.noindex/UIViewPropertyAnimatorObjCSample.build/Debug-
iphoneos/UIViewPropertyAnimatorObjCSample.build/
UIViewPropertyAnimatorObjCSample-generated-files.hmap -I/Users/wmoses/Library/
Developer/Xcode/DerivedData/UIViewPropertyAnimatorObjCSample-
gmyxiqyiqqtmgfbeqgqiuwfodewt/Build/Intermediates.noindex/
UIViewPropertyAnimatorObjCSample.build/Debug-iphoneos/
UIViewPropertyAnimatorObjCSample.build/UIViewPropertyAnimatorObjCSample-own-
target-headers.hmap -I/Users/wmoses/Library/Developer/Xcode/DerivedData/
UIViewPropertyAnimatorObjCSample-gmyxiqyiqqtmgfbeqgqiuwfodewt/Build/
Intermediates.noindex/UIViewPropertyAnimatorObjCSample.build/Debug-iphoneos/
UIViewPropertyAnimatorObjCSample.build/UIViewPropertyAnimatorObjCSample-all-
target-headers.hmap -iquote /Users/wmoses/Library/Developer/Xcode/DerivedData/
UIViewPropertyAnimatorObjCSample-gmyxiqyiqqtmgfbeqgqiuwfodewt/Build/
Intermediates.noindex/UIViewPropertyAnimatorObjCSample.build/Debug-iphoneos/
UIViewPropertyAnimatorObjCSample.build/UIViewPropertyAnimatorObjCSample-
project-headers.hmap -I/Users/wmoses/Library/Developer/Xcode/DerivedData/
UIViewPropertyAnimatorObjCSample-gmyxiqyiqqtmgfbeqgqiuwfodewt/Build/Products/
Debug-iphoneos/include -I/Users/wmoses/Library/Developer/Xcode/DerivedData/
UIViewPropertyAnimatorObjCSample-gmyxiqyiqqtmgfbeqgqiuwfodewt/Build/
Intermediates.noindex/UIViewPropertyAnimatorObjCSample.build/Debug-iphoneos/
UIViewPropertyAnimatorObjCSample.build/DerivedSources-normal/arm64 -I/Users/
wmoses/Library/Developer/Xcode/DerivedData/UIViewPropertyAnimatorObjCSample-
gmyxiqyiqqtmgfbeqgqiuwfodewt/Build/Intermediates.noindex/
UIViewPropertyAnimatorObjCSample.build/Debug-iphoneos/
UIViewPropertyAnimatorObjCSample.build/DerivedSources/arm64 -I/Users/wmoses/
Library/Developer/Xcode/DerivedData/UIViewPropertyAnimatorObjCSample-
gmyxiqyiqqtmgfbeqgqiuwfodewt/Build/Intermediates.noindex/
UIViewPropertyAnimatorObjCSample.build/Debug-iphoneos/
UIViewPropertyAnimatorObjCSample.build/DerivedSources -F/Users/wmoses/Library/
Developer/Xcode/DerivedData/UIViewPropertyAnimatorObjCSample-
gmyxiqyiqqtmgfbeqgqiuwfodewt/Build/Products/Debug-iphoneos /Users/wmoses/apple/
iOS-10-Sampler/UIViewPropertyAnimator/UIViewPropertyAnimatorObjCSample/
UIViewPropertyAnimatorObjCSample/PropertyAnimatorViewController.m -o /Users/
wmoses/Library/Developer/Xcode/DerivedData/UIViewPropertyAnimatorObjCSample-
gmyxiqyiqqtmgfbeqgqiuwfodewt/Build/Intermediates.noindex/
UIViewPropertyAnimatorObjCSample.build/Debug-iphoneos/
UIViewPropertyAnimatorObjCSample.build/Objects-normal/arm64/
PropertyAnimatorViewController.o
```

- Identify required arguments & inputs (**purple**)
- Remove unused defines (**blue**)
- Normalize include paths (**green**)
- Provide map of exactly what files are used with their corresponding hash in content-addressable storage (**red**)

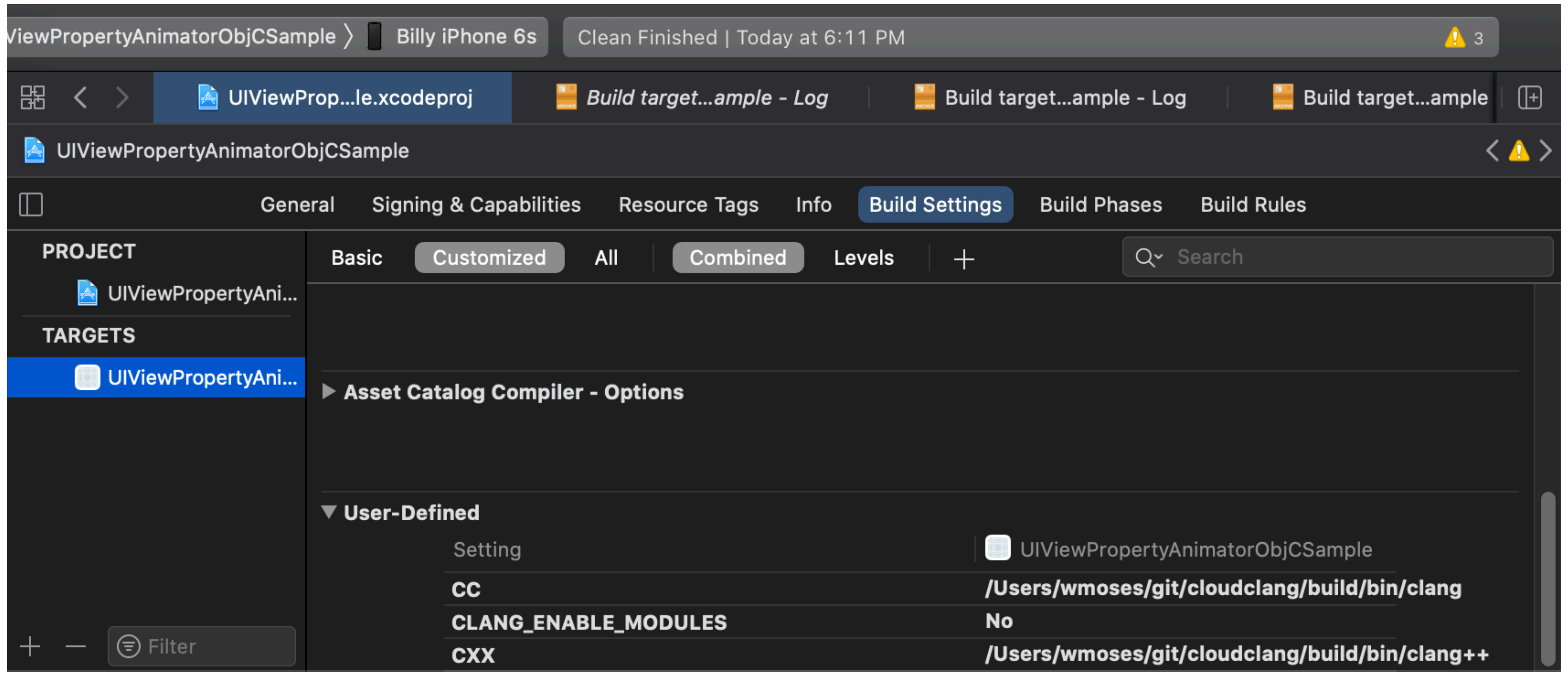
```
args: ["-cc1", "-triple", "arm64-apple-ios10.0.0",
       "-o", "o0", "-x", "objective-c",
       "PropertyAnimatorViewController.m",
       "-internal-isystem", "/fakeroot-s"],
inputs: {
  "/fakeroot-s/UIKit.framework/Headers/UIKit.h":
    "wFrlpQYtbT2X04lsYCr+rKR3FfJUGhvy9Xw8sIYcGG4=",
  "PropertyAnimatorViewController.h":
    "fke8yluU1f/H55VrnLK3xOzubvr/3h24VjBSW8aZc+Q=",
  "PropertyAnimatorViewController.m":
    "uqncMKT16aeuzIjFrlwkYh4vH0Wtp1nB+Nz8Vc82nuc="
}
```

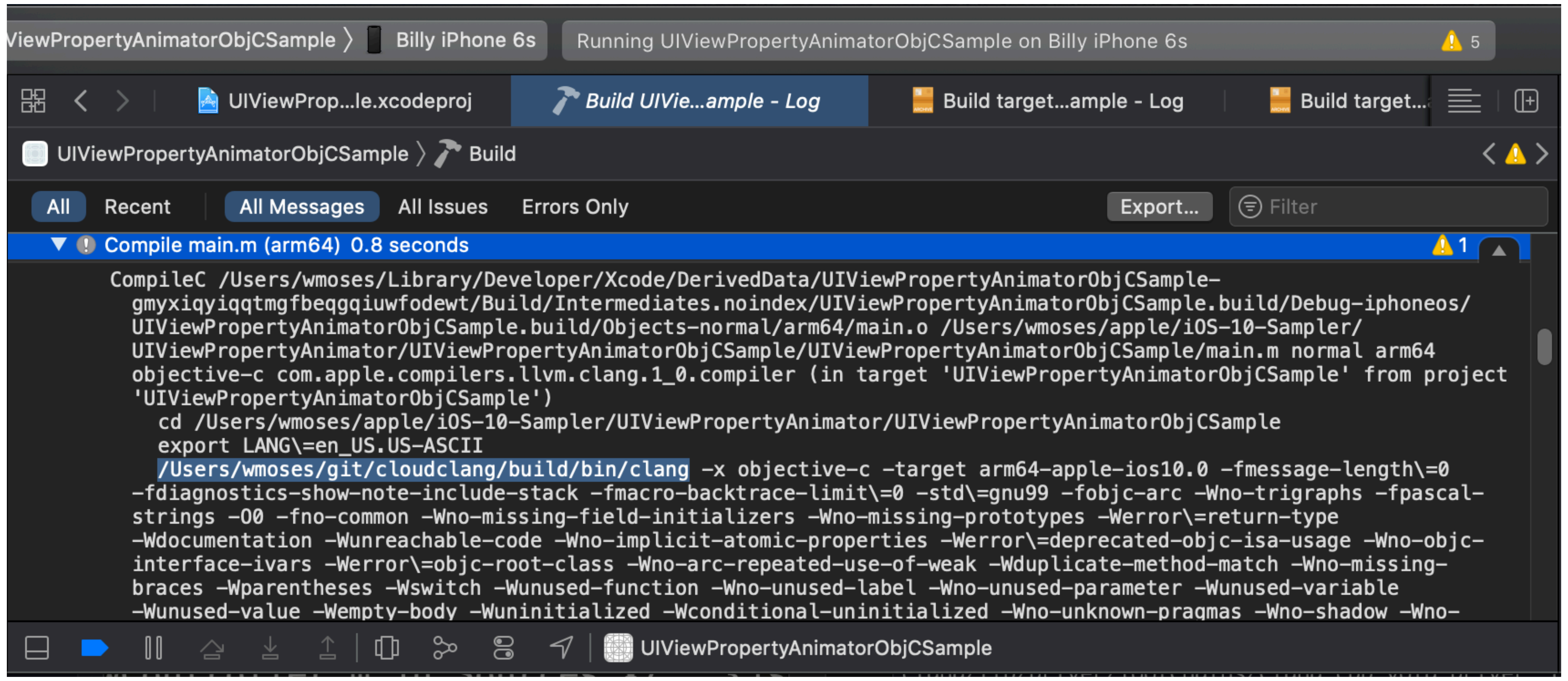
# Cross-Platform & Cross-Architecture

- When client binaries are run it identifies the desired target platform and architecture which are later passed to the lambda compilation task

- Every Raspberry Pi is secretly a thousand-core compiling supercomputer!

    - Compile for ARM iOS/macOS on x86 Linux cluster (or other)



# Performance Optimizations

Three primary components of Cymbl compilation time:

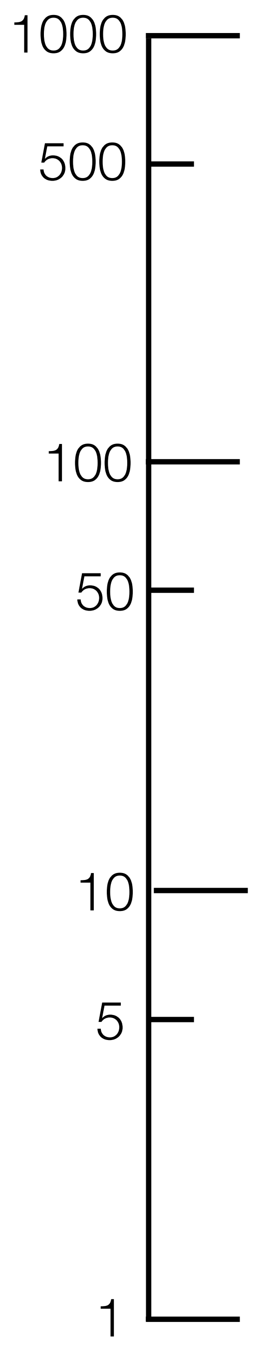
1. File Transfer (upload inputs / download results)
2. Communication Latency
3. Remote Task Execution (clang/lld jobs)
    - Time = Money and Shared among everyone
    - Full link time optimization (libc, libc++, DNS resolver, boringssl, curl, libclang, …)
    - Statically link everything
        - Bonus: Binaries are very portable (no dependencies)

# File Transfer

- Biggest (initial) bottleneck for clients is transferring inputs/results
- Daemon serves as a single point to optimize transfers (rather than per process)
    - Staged existence caching (with invalidation)
        - Local concurrent map (fastest); Batched remote check (mid speed); (potential) re-upload (slowest)
    - Limit the number of concurrent uploads/connections (per network performance)
    - Assuming cluster network is much faster than one's ISP, batch upload many files together for later split by remote upload processing lambda
- Storage with weaker properties (non-atomic) is vastly faster than that with stronger properties
    - Design invalidation-safe idempotent upload process
    - Retry compilation task if file has not been propagated to storage where needed

# Latency

- Reducing the latency of file existence checks and already-cached tasks is key to performance of large workloads




~300ms : “Standard” lambda function

~50ms : FullLTO + statically linked everything
Latency reduced both by time optimization and reduction in file size

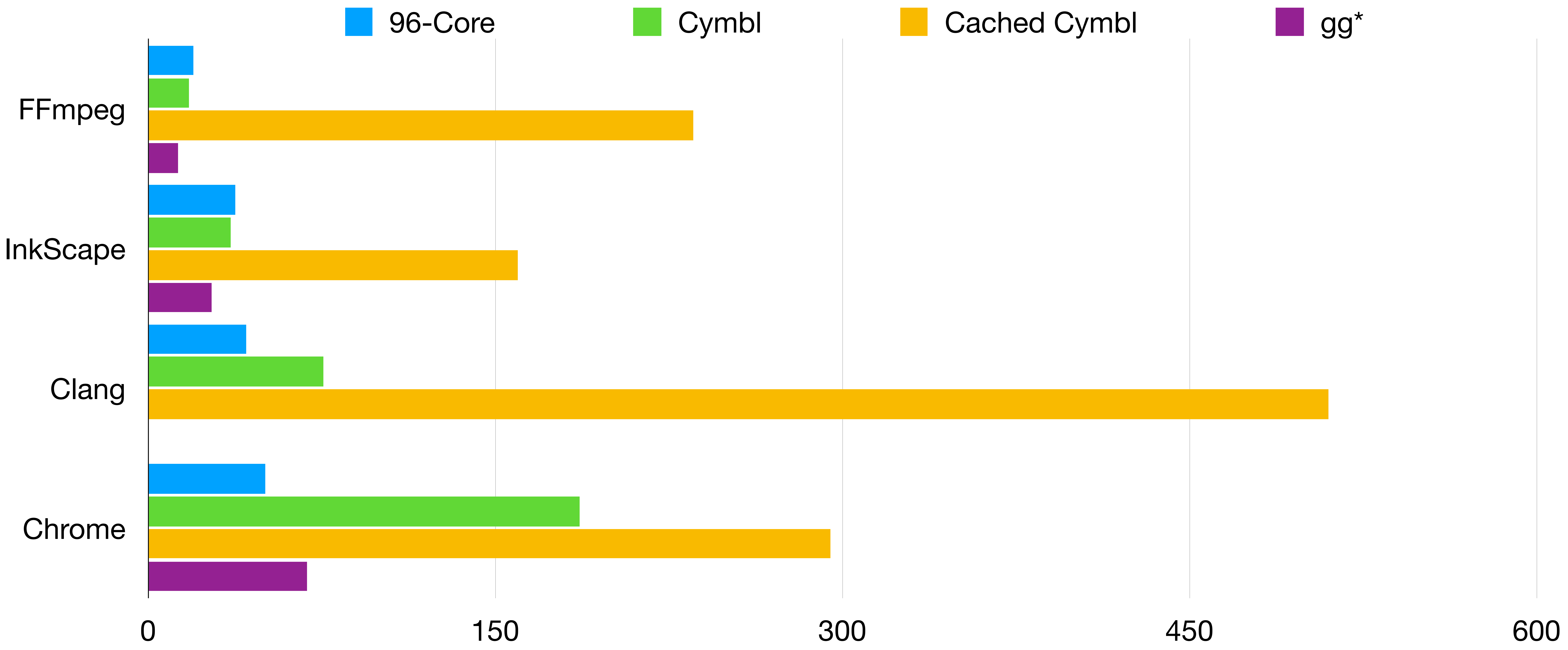
~5ms : Handwritten AWS API, persistent connection, fine tuning flags

# Evaluation

| | 1-Core | 96-Core | Cymbl | Cached Cymbl | gg* |
|---|---|---|---|---|---|
| FFmpeg | 9.43 | 0.48 | 0.53 | 0.04 | 0.73* |
| InkScape | 39.96 | 1.06 | 1.12 | 0.25 | 1.45* |
| Clang | 183.55 | 4.32 | 2.42 | 0.36 | |
| Chrome | 1302.65 | 25.71 | 6.99 | 4.42 | 18.92* |

*gg results taken from paper, due to inability to reproduce results

# Relative Speed-up (vs Single Core)




*gg results taken from paper, due to inability to reproduce results

# Costs & Other Analysis

- Costs computed for initial ram budget (3GB)
- 50k file compilation task
- 96-core cost $4.08/hour (need the hour)
    - ~2x more expensive uncached [3.5x speed]
    - ~22x cheaper when cached [and 300x speed]
- 47 hours of compute for uncached; 1 hour of compute for cached

| | Chrome Uncached | Chrome Cached |
|---|---|---|
| clang | $8.478 | $0.184 |
| lld | $0.047 | $0.002 |
| exists | $0.014 | $0.000 |
| upload | $0.026 | $0.000 |
| Total | $8.565 | $0.186 |

# Optimized Costs

- As >99.996% **tasks use <1.5GB (can half the cost)**
- 50k file compilation task
- 96-core cost $4.08/hour (need the hour)
    - ~On par when uncached [3.5x speed]
    - ~43x cheaper when cached [and 300x speed]
- 47 hours of compute for uncached; 1 hour of compute for cached

| | Chrome Uncached | Chrome Cached |
|---|---|---|
| clang | $4.240 | $0.092 |
| lld | $0.047 | $0.002 |
| exists | $0.014 | $0.000 |
| upload | $0.026 | $0.000 |
| Total | $4.326 | $0.094 |

# Security

- All accesses to any cloud data are mediated by a Gatekeeper
- Gatekeeper only grants downloads of results of tasks submitted by that user
  - Cannot download another’s source
  - Cannot download another’s artifacts without a compilation job that would result in that artifact anyways
- Remaining attack vector: brute force timing attack of existence queries for source code / compilation jobs to attempt to identify another user’s source:
  - Intractable space size (all programs) and only can work once (since all brute forced jobs will be subsequently cached)

# Potential Additional Security Extensions

**Increasingly Paranoid Threat Model**

***No Artifact Timing Attacks***

- Solution: Per user / company cache, or disable compilation cache
- Cost: Reduction or loss of caching speedups

***No Input Timing Attacks***

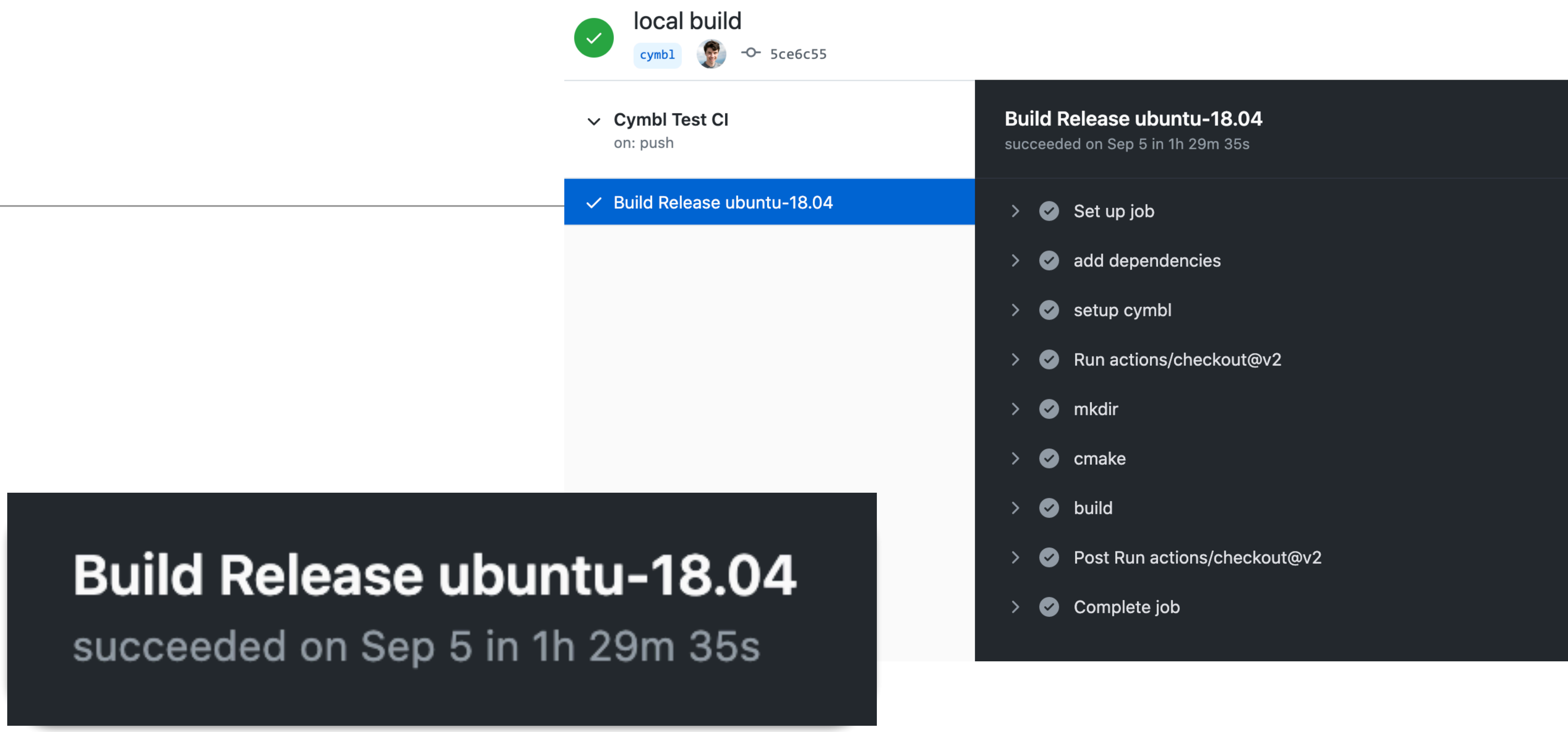
- Solution: Per user / company content-addressable storage
- Cost: Reduction of file-upload speedups AND costs to the left

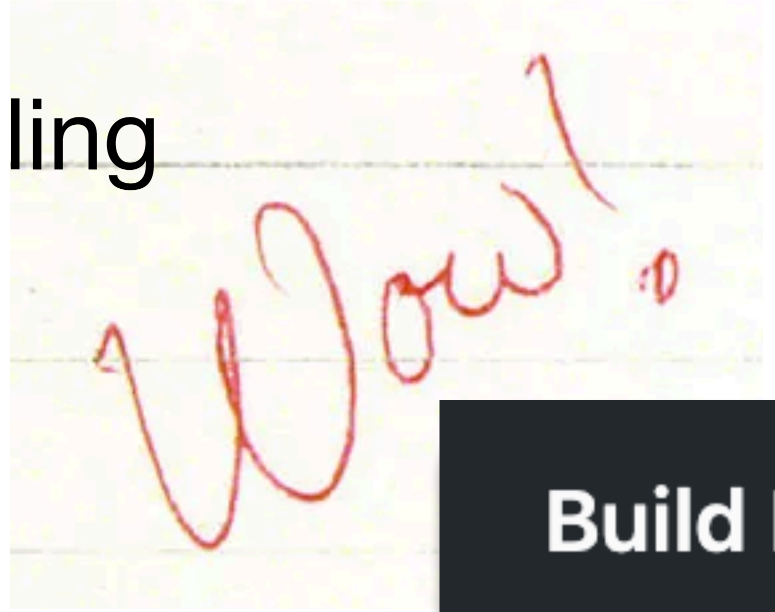
***Distrust service provider***

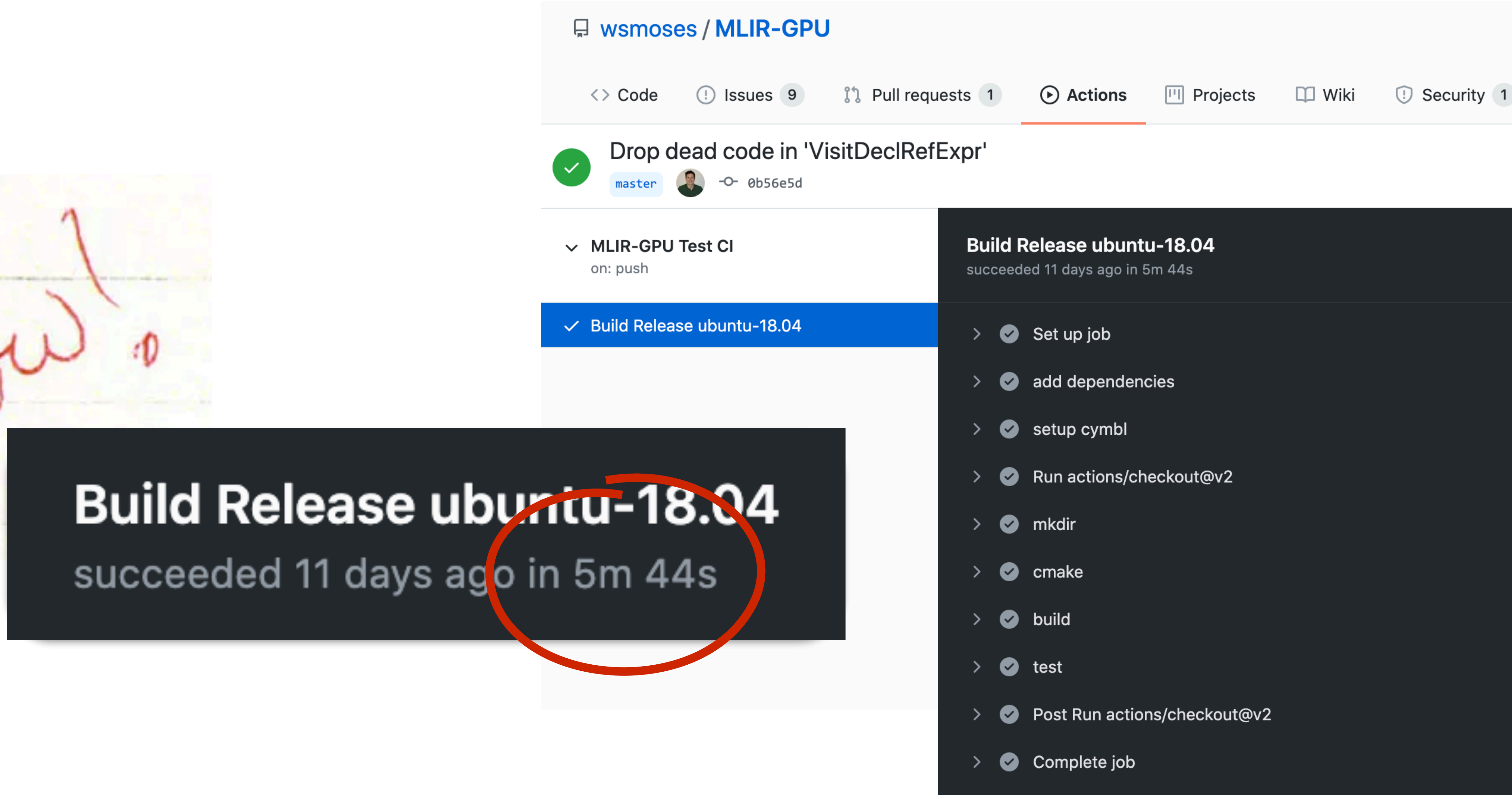
- Solution: User / company hosted task executors
- Cost: Maximum parallelism is limited to the size of the cluster, cost of maintaining a cluster, AND costs to the left

# Status & Limitations

- Built on top of LLVM version 11
  - Tool can (and has been) rebased across LLVM versions
  - LLD only supports ELF not MACH targets (cymbl mach target works but LLVM proper doesn't handle frameworks)
- Does not yet support caching with modules (falling back to caching with headers)
- Use as compile-tool and CI for MIT projects
- Accepting beta users for SAAS



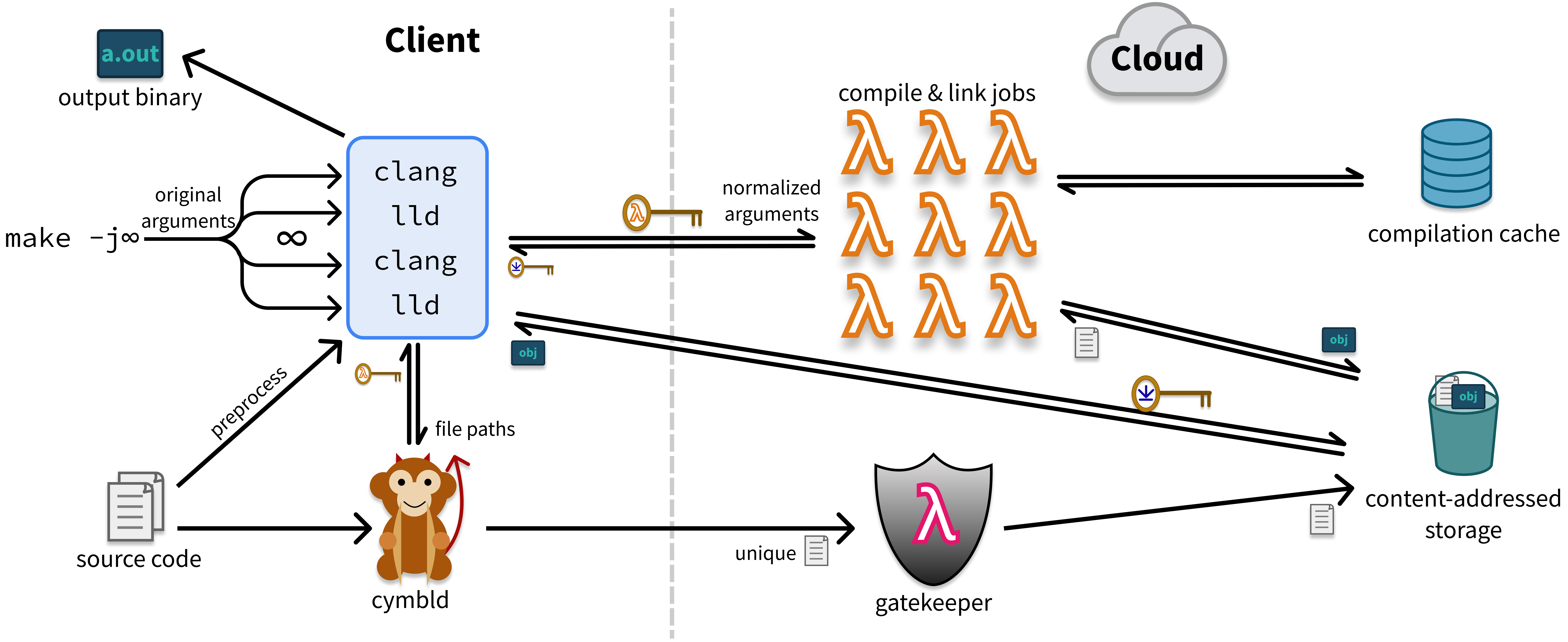
# Future Work

- Global Scale Compilation
  - Super-optimization
  - Profile-guided optimization database
- Language Extension (Swift, Rust, Go)
- Fine-Granularity Caching

# Conclusions

- Raspberry Pi + Cymbl Cloud = Compiling Supercomputer!
- Compiler-level integration enables significantly better caching and compatibility
- State-of-the-art performance without the cost of a cluster
- Sign up for our beta! https://cymbl.dev/
- William S. Moses was supported in part by a DOE Computational Sciences Graduate Fellowship DE-SC0019323.

# Questions?



Client
Cloud
a.out
output binary
make -j∞
original
arguments
∞
clang
lld
clang
lld
preprocess
file paths
source code
cymbld
normalized
arguments
obj
compile & link jobs
compilation cache
obj
content-addressed
storage
unique
gatekeeper

# Backup Slides

# Usage

- Same compiler binaries can be used for either local or remote builds
    - Environmental variable enables or disables (CYMBL=On by default)

# Existing Techniques

- Compatibility
    - Build-System Based (Bazel)
        - Requires rewriting all code to use the given build system, which handles remote task execution
    - Substitution-Based (Goma, DistCC, IceCC, gg)
        - Create fake “cc” compiler scripts to intercept tasks and execute remotely
        - gg builds a static graph of all computations ahead of time (potentially faster) at the cost of requiring all commands in the build process to be perfectly modeled
        - Requires maintaining an accurate model of all potential flags / behaviors for all tools, quickly becoming out of date and unlikely to align with a given system
- Excluding gg, all tools require a user-maintained cluster, limiting parallelism and increasing cost
- Caches at best recognize files in the same codebase being compiled in the same way

# Potential Additional Security Extensions

- Per user / company cache, or disable entirely (request no cache)
  - Pro: Eliminate any compilation-job cache timing attacks
  - Con: Reduction or loss of caching speedups
- Per user / company content-addressable storage
  - Pro: Eliminate any input file cache timing attacks
  - Con: Above and reduction of file-upload speedups
- User / company-hosted job executors
  - Pro: No need to trust service provider (e.g. AWS)
  - Con: Above and maximum parallelism is limited to size of cluster which must be always on