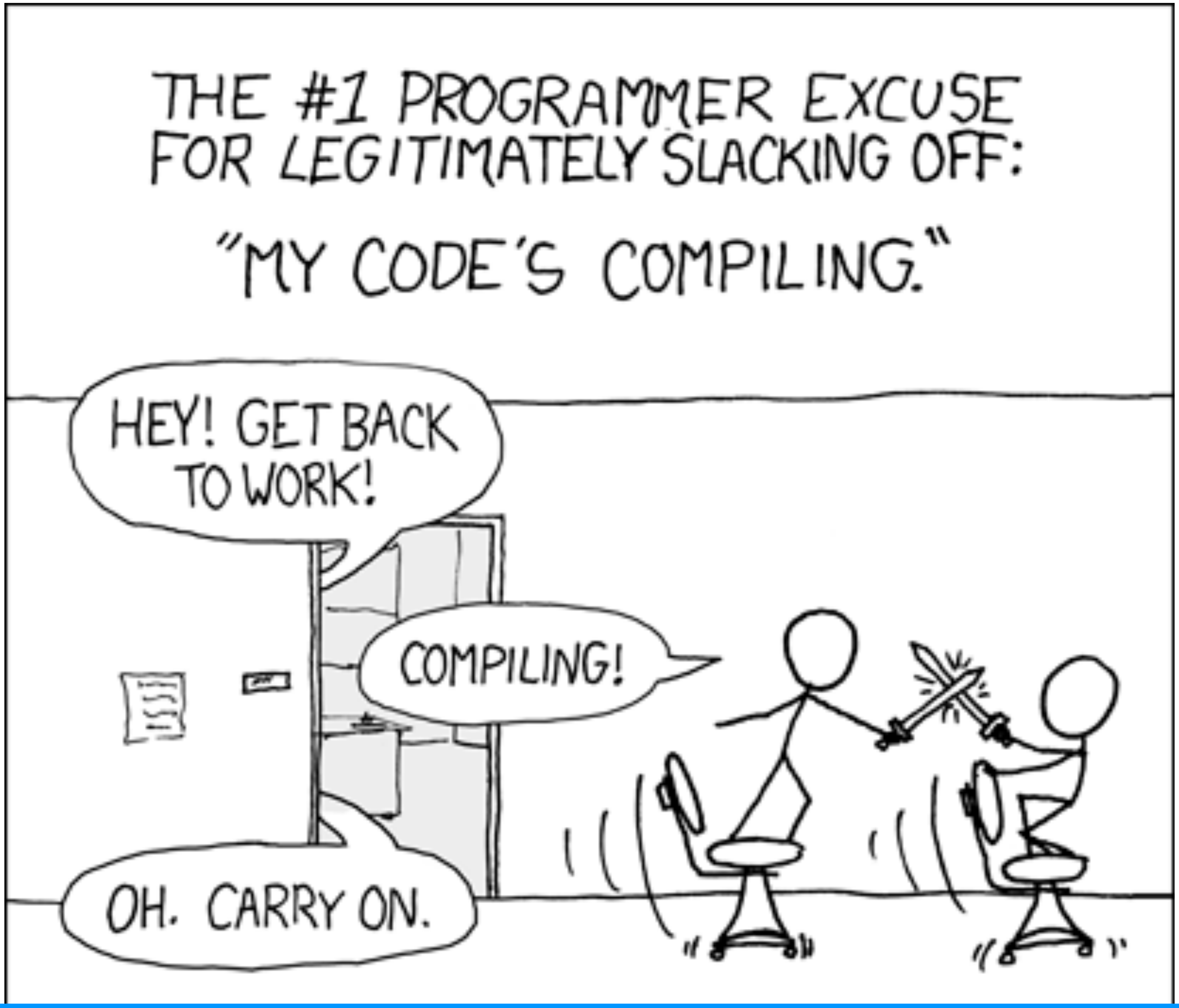
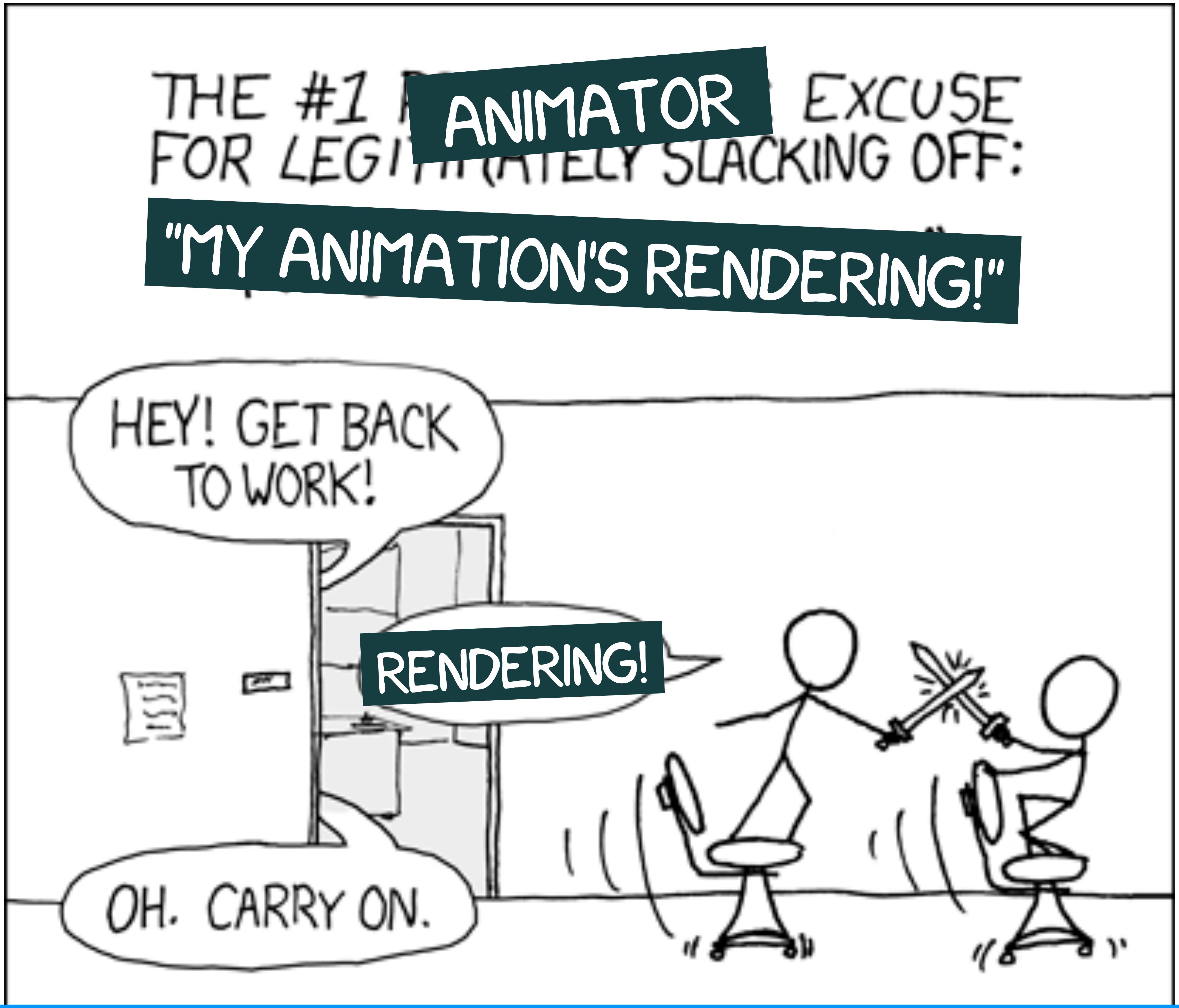
Supercomputing as a Service:
Massively-Parallel Jobs on FaaS Platforms

Sadjad Fouladi
Stanford University



Compiling **`clang`** takes **>2 hours.**

Compressing a **15-minute 4K video** takes **~7.5 hours.**



Rendering **each frame** of **Monsters University** took **29 hours.**

The Problem

Many of these pipelines take *hours and hours* to finish.

The Question

Can we achieve interactive speeds in these applications?

The Answer

Massive Parallelism*

* *well, probably.*

How to get thousands of threads?

- The largest companies are able to operate massive datacenters that can support such levels of parallelism.
- But, end users and developers are unable to scale their resource footprint to thousands of parallel threads on demand in an efficient and scalable manner.

Classic Approach: VMs

- Infrastructure-as-a-Service
  - Thousands of threads
  - Arbitrary Linux executables
  - 👎 Minute-scale startup time (OS has to boot up, ...)
  - 👎 High minimum cost

Cloud function services have (as yet) unrealized power

- AWS Lambda, Google Cloud Functions, IBM Cloud Functions, Azure Functions, etc.

- Intended for event handlers and Web microservices, *but...*

- Features:

✔ Thousands of threads

✔ Arbitrary Linux executables

✔ Sub-second startup

✔ Sub-second billing

3,600 threads for one second → 10¢

Supercomputing as a Service

Encoding

Compressing this video will take a long time. How do you want to execute this job?

Locally (~5 hours)

Remotely (~5 secs, 50¢)

Cancel

Two projects that we did based on this promise:

- **ExCamera:** Low-Latency Video Processing
- **gg:** `make -j1000` (and other jobs) on FaaS infrastructure

ExCamera: Low-Latency Video Processing Using Thousands of Tiny Threads

---

Sadjad Fouladi, Riad S. Wahby, Brennan Shacklett, Karthikeyan Balasubramaniam, William Zeng, Rahul Bhalerao, Anirudh Sivaraman, George Porter, and Keith Winstein. *"Encoding, Fast and Slow: Low-Latency Video Processing Using Thousands of Tiny Threads."* In 14th USENIX Symposium on Networked Systems Design and Implementation (NSDI'17).

What we currently have

Google Docs

- People can make changes to a word-processing document
- The changes are instantly visible for the others

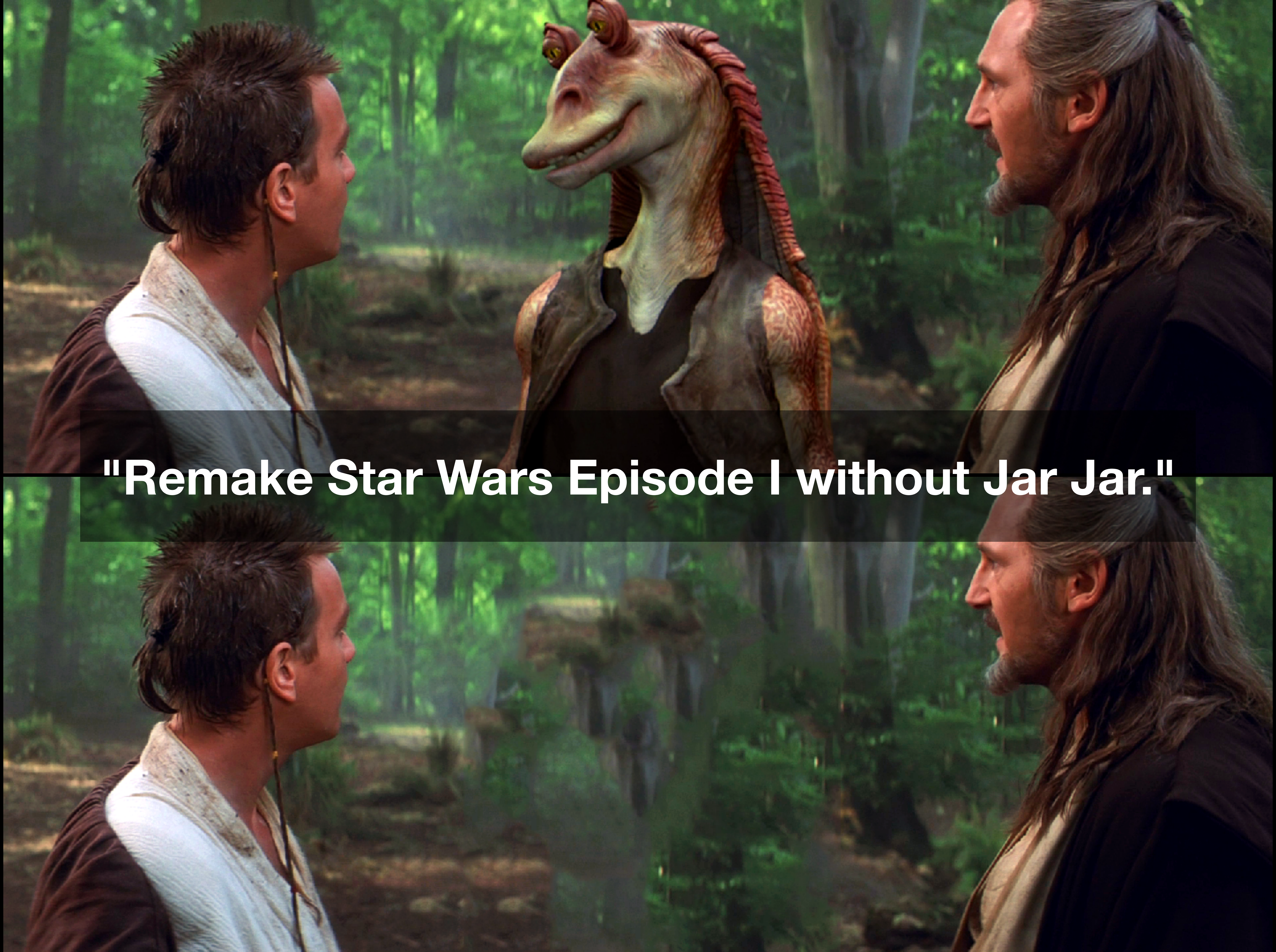
What we would like to have

Google Docs *for Video?*

- People can interactively edit and transform a video
- The changes are instantly visible for the others

"Apply this awesome filter to my video."

"Look everywhere for this face in this movie."

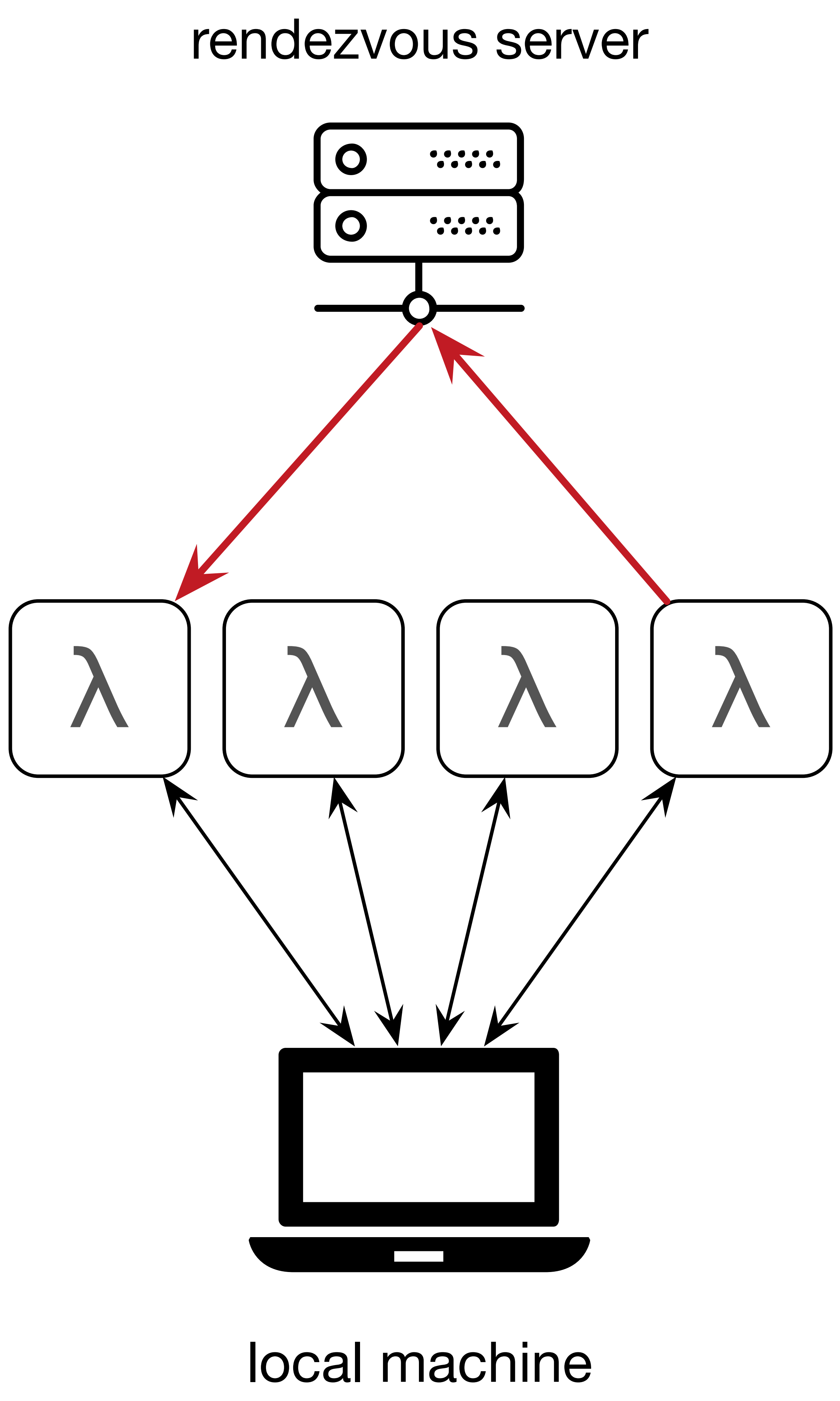

"Remake Star Wars Episode I without Jar Jar."

Challenges in low-latency video processing

- Low-latency video processing would need **thousands of threads**, **running in parallel**, with **instant startup.**
- However, **the finer-grained the parallelism, the worse the compression efficiency.**

First challenge: thousands of threads

- We built ***mu***, a library for designing and deploying general-purpose parallel computations on a commercial "cloud function" service.
- The system starts up thousands of threads in seconds and manages inter-thread communication.
- *mu* is open-source software: *https://github.com/excamera/mu*



Second challenge: parallelism hurts compression efficiency

- Existing video codecs only expose a simple interface that's not suitable for massive parallelism.
- We built a video codec in **explicit state-passing style**, intended for **massive fine-grained parallelism**.
- Implemented in 11,500 lines of C++11 for Google's VP8 format.

```
decode(state, frame) → (state′, image)
encode(state, image) → interframe
rebase(state, image, interframe) → interframe′
```

14.8-minute 4K Video @20dB	
vpxenc Single-Threaded	**453 mins**
vpxenc Multi-Threaded	**149 mins**
YouTube (H.264)	**37 mins**
ExCamera	**2.6 mins**

ExCamera

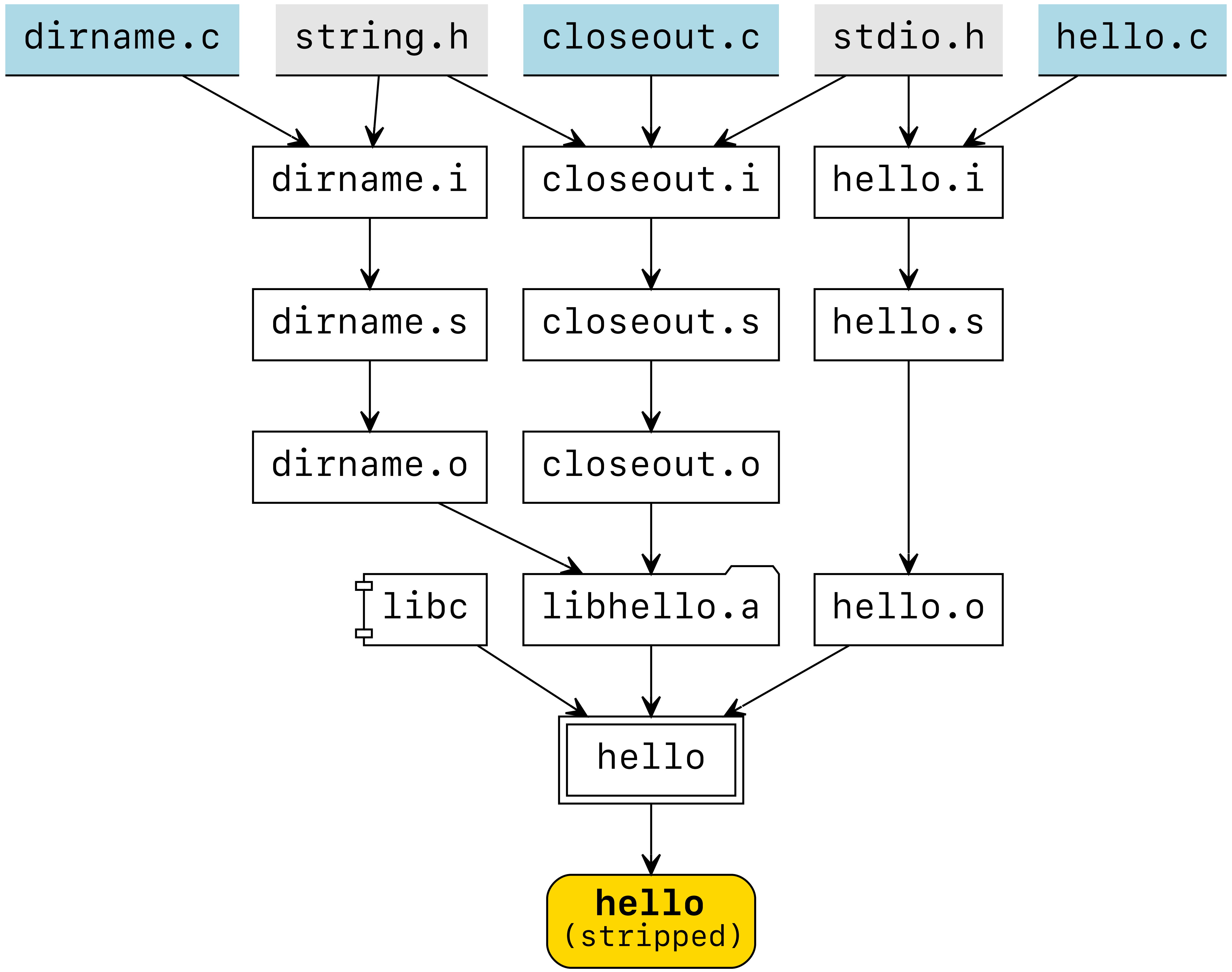
- Two major contributions:
  - Framework to run **5,000-way parallel jobs** with IPC on a commercial “cloud function” service.
  - Purely functional video codec for **massive fine-grained parallelism**.
- 56× faster than existing encoder, for <$6.

`gg: make -j1000` (and other jobs) on function-as-a-service infrastructure

Sadjad Fouladi, Dan Iter, Shuvo Chatterjee, Christos Kozyrakis, Matei Zaharia, Keith Winstein

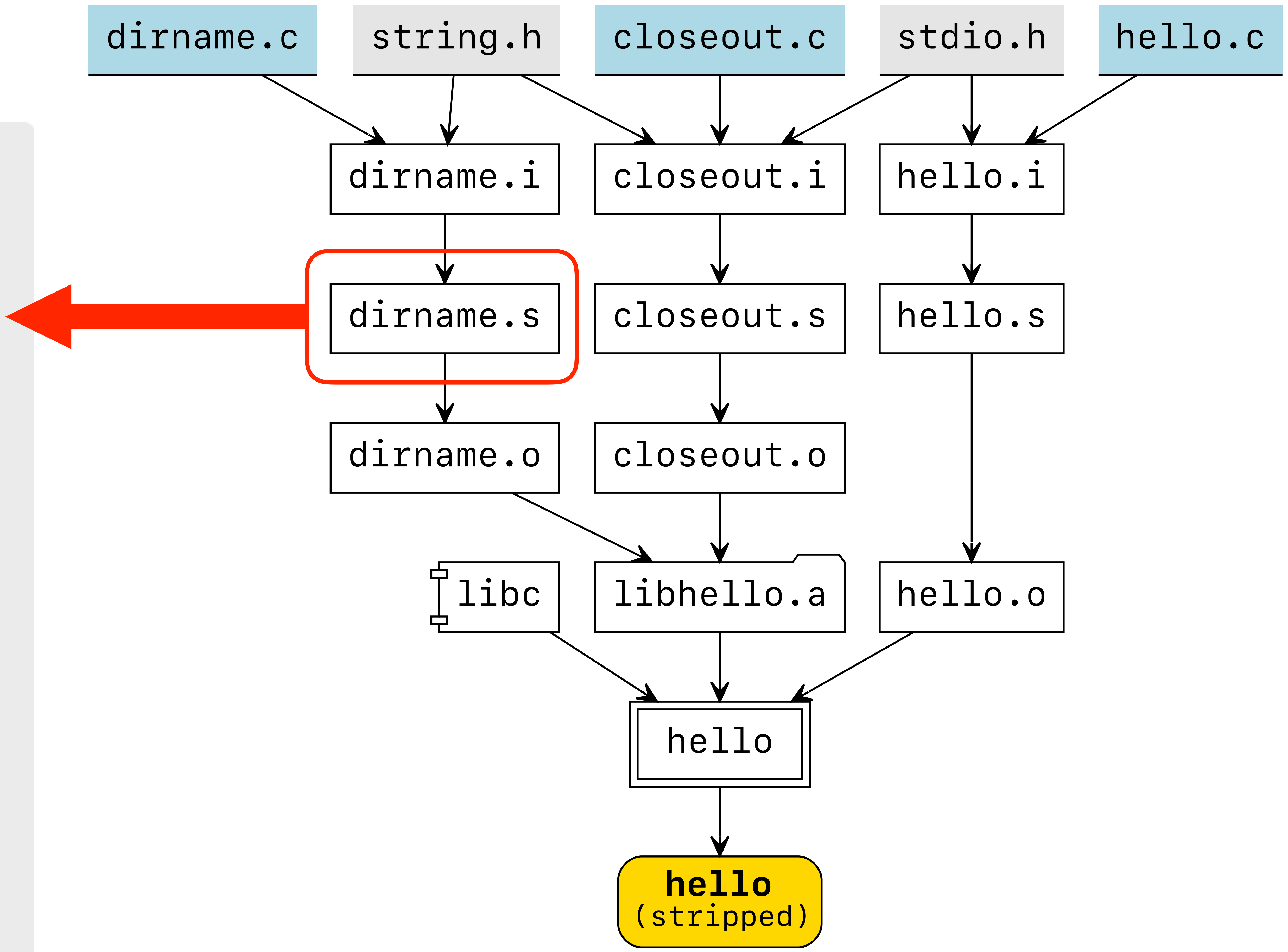
What is gg?

- gg is a system for executing interdependent software workflows across thousands of short-lived “lambdas”.



"*Thunk*" abstraction

```
{ "function": { "exe": "g++",
                "args": ["-S", "dirname.i",
                         "-o",...],
                "hash": "A5BNh" },
  "infiles": [
    { "name": "dirname.i",
      "order": 1,
      "hash": "SoYcD"
    },
    {
      "name": "g++",
      "order": 0,
      "hash": "A5BNh"
    }
  ],
  "outfile": "dirname.s"
}
```



"*Thunk*" abstraction

```
{ "function": { "exe": "g++",
                "args": ["-S", "dirname.i",
                         "-o",...],
                "hash": "AsBNh" },
  "infiles": [
    { "name": "dirname.i",
      "order": 1,
      "hash": "SoYcD"
    },
    {
      "name": "g++",
      "order": 0,
      "hash": "ts0sB"
    }
  ],
  "outfile": "dirname.s"
}
```

- Thunk is an abstraction for representing a morsel of computation in terms of **a function** and **its complete functional footprint**.

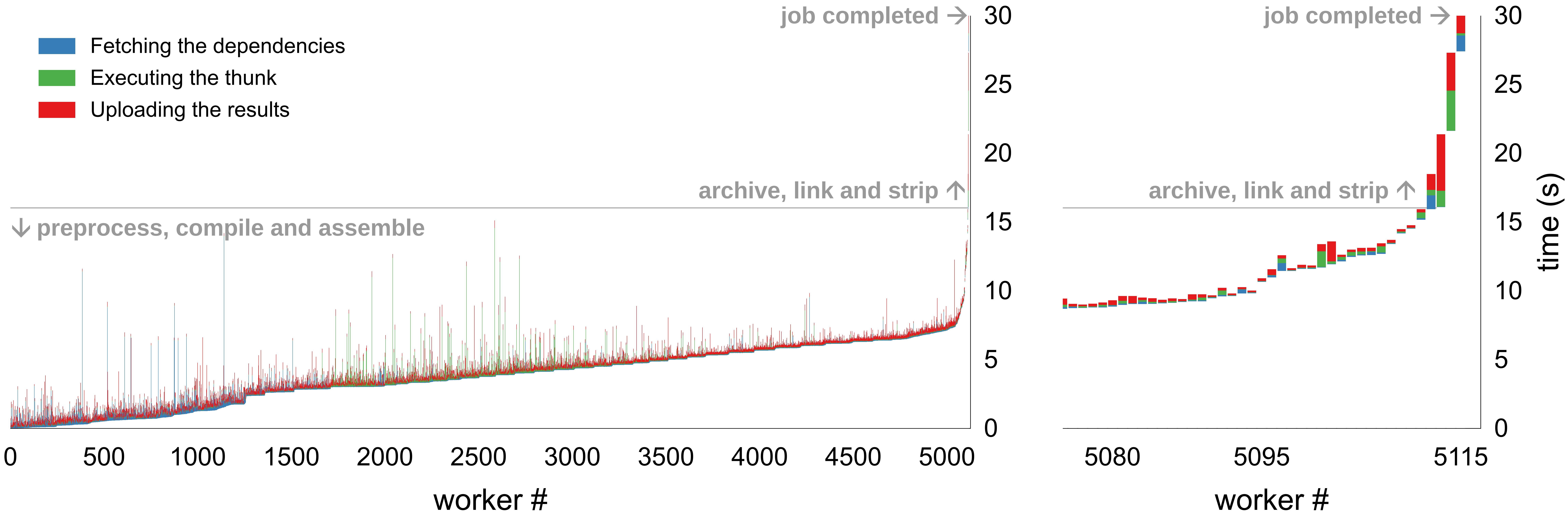
- Thunks can be **forced** *anywhere*, on the local machine, or on a remote VM, or *inside a lambda function.*

Execution

- Generating the dependency graph in terms of *thunks*:
  `gg-infer make`

- Forcing the thunk, recursively:
  `gg-force --jobs 1000 bin/clang`

Compiling FFmpeg using gg



Evaluation

	single-core	gg (λ)
`ffmpeg`	9m 45s	35s
`inkscape`	33m 35s	1m 15s
`llvm`	1h 16m 18s	1m 11s

gg is open-source software

https://github.com/StanfordSNR/gg

Takeaways

- The future is granular, interactive and massively parallel.
- Many applications can benefit from this "*Laptop Extension*" model.
- Better platforms are needed to be built to support "bursty" massively-parallel jobs.

https://github.com/StanfordSNR/gg