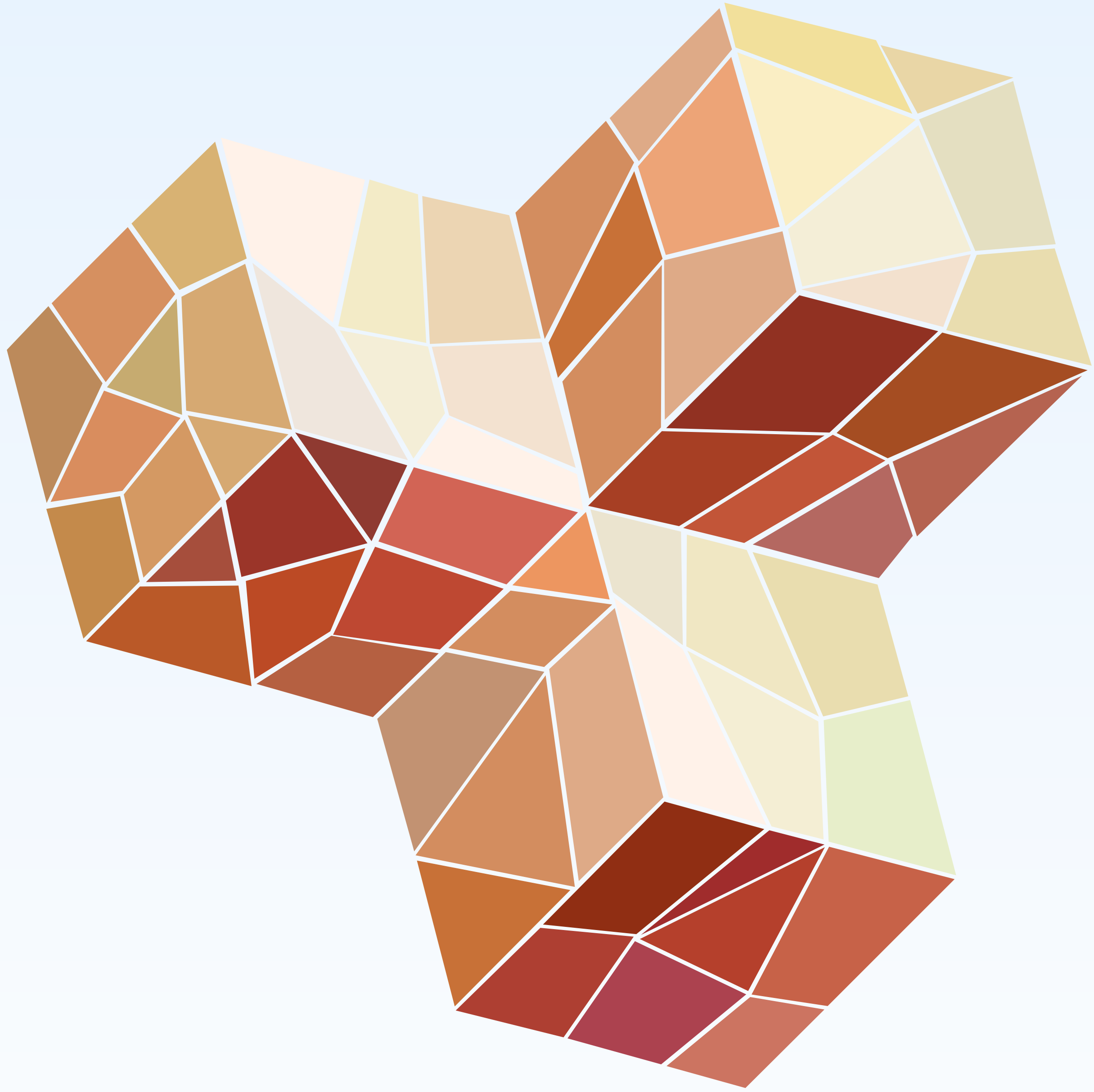


Modus

A language for building container images

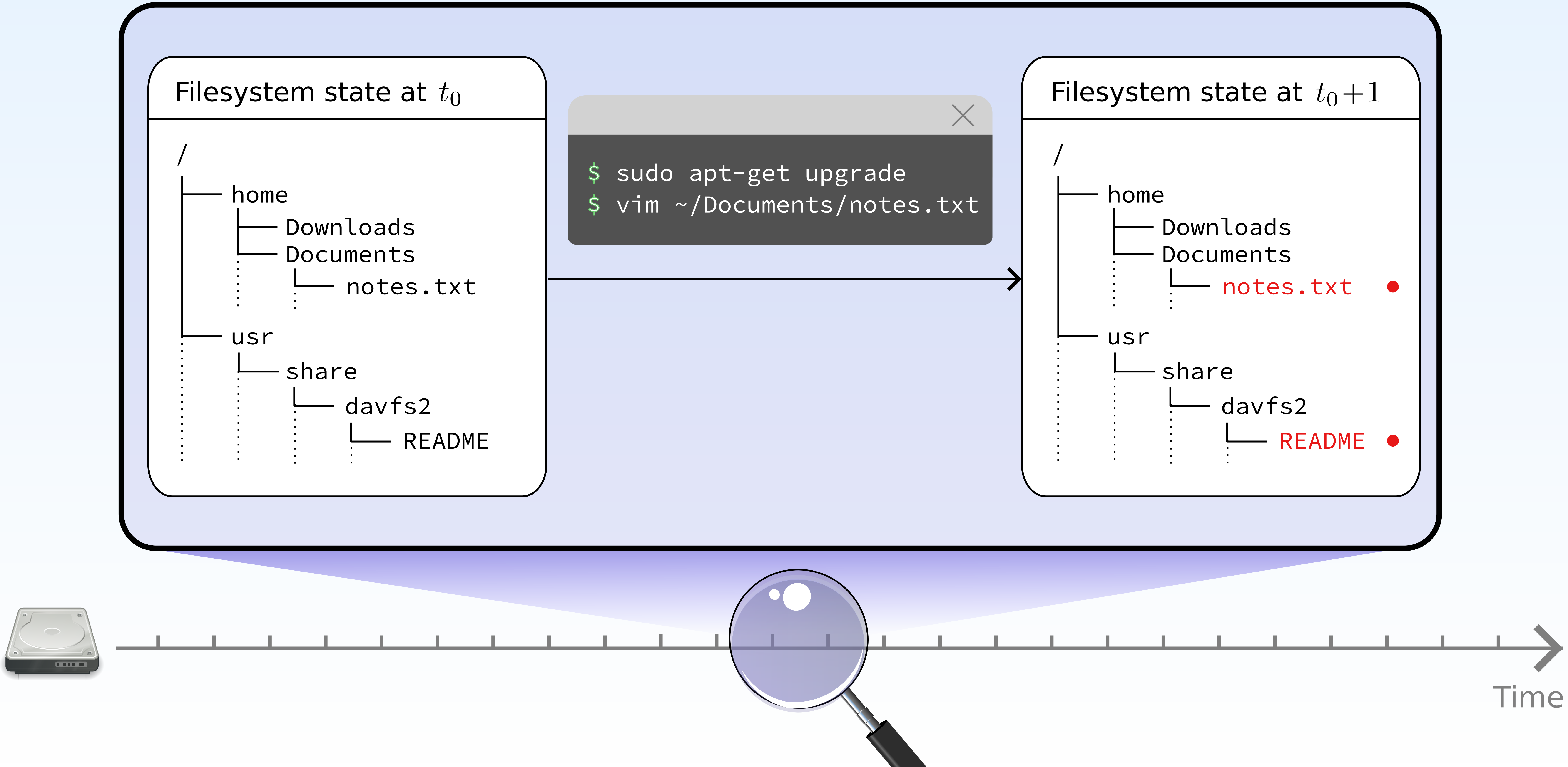
Chris Tomy

Tingmao Wang

Earl T. Barr

Sergey Mechtaev

System Evolution As a Sequence of Mutations



Filesystem state at t_0

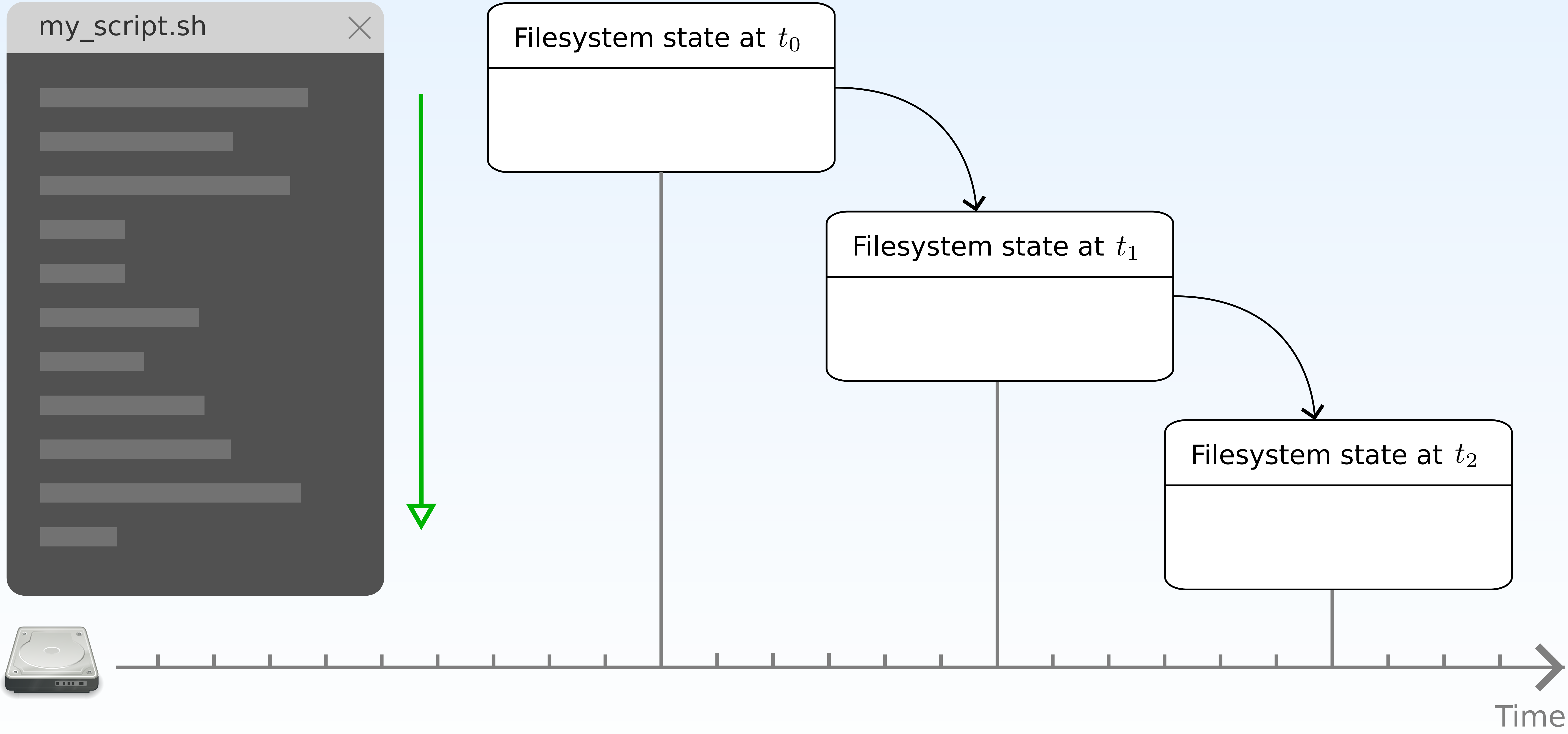
```
/
├── home
│   ├── Downloads
│   └── Documents
│   :   └── notes.txt
│   :   :
└── usr
:   └── share
:   :   └── davfs2
:   :   :   └── README
:   :   :   :
```

```
$ sudo apt-get upgrade
$ vim ~/Documents/notes.txt
```

Filesystem state at $t_0 + 1$

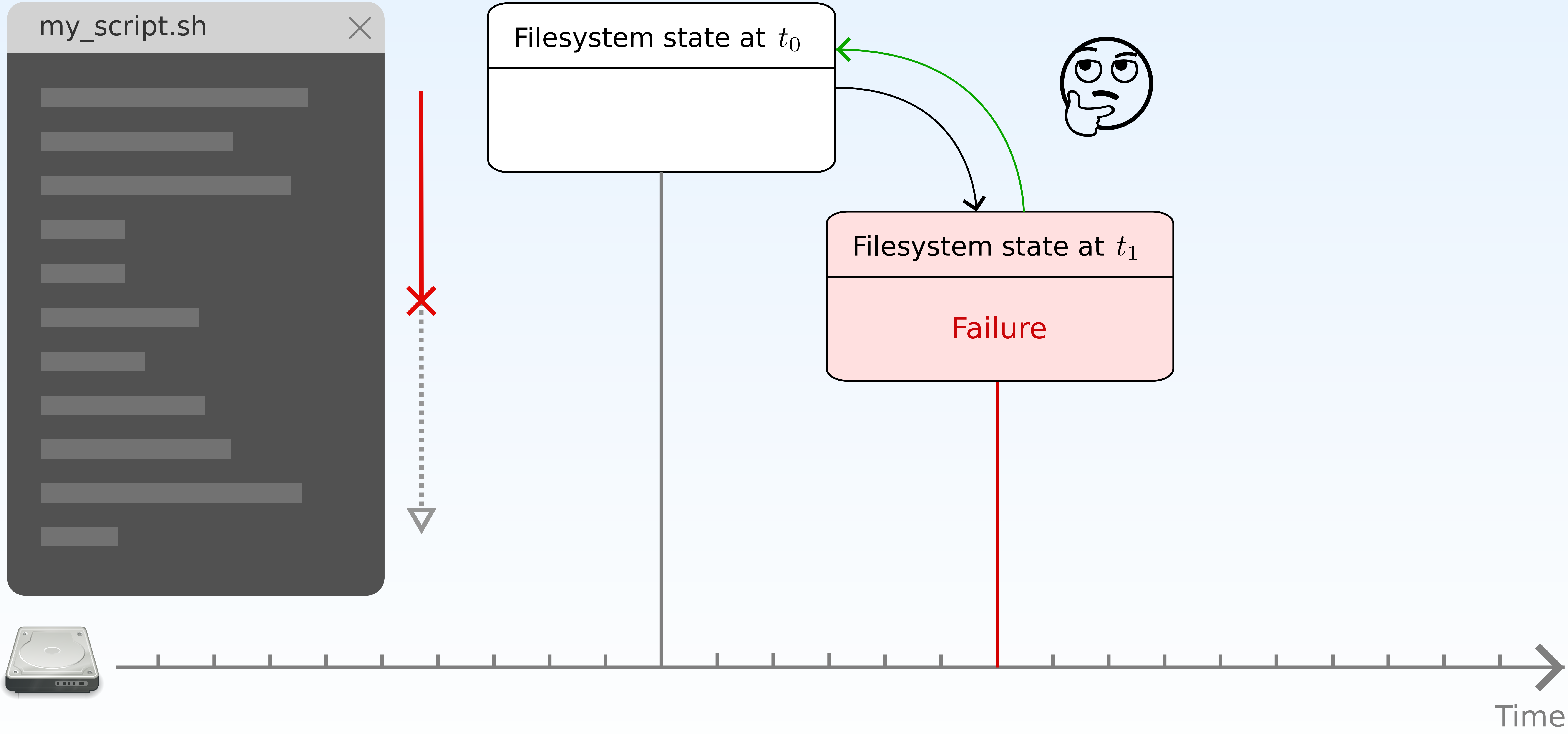
```
/
├── home
│   ├── Downloads
│   └── Documents
│   :   └── notes.txt   ●
│   :   :
└── usr
:   └── share
:   :   └── davfs2
:   :   :   └── README   ●
:   :   :   :
```

Execution As a Sequence of Mutations

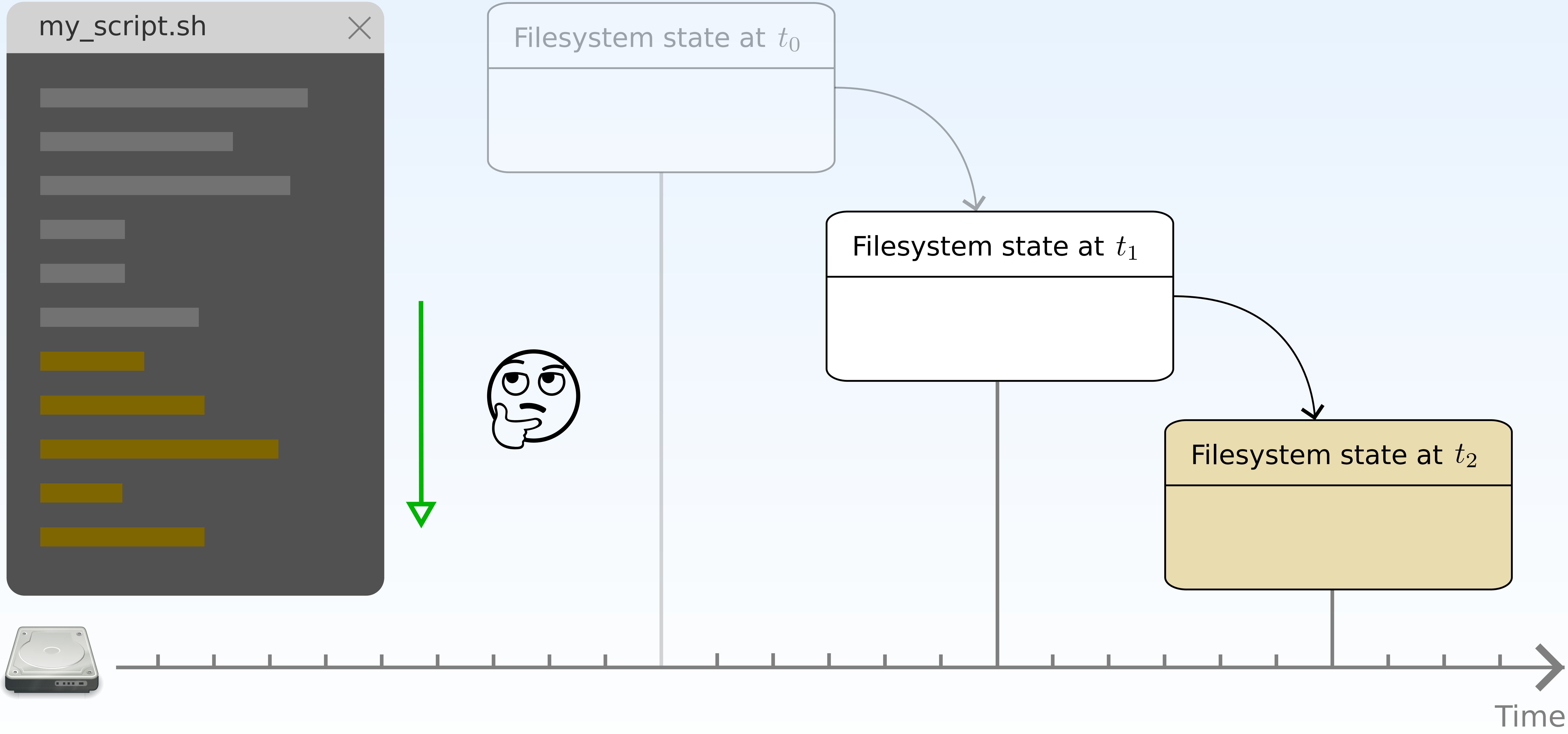


my_script.sh
Filesystem state at t_0
Filesystem state at t_1
Filesystem state at t_2
Time

Hard to Restart Failed Execution



Hard to Reuse Intermediate Results



my_script.sh

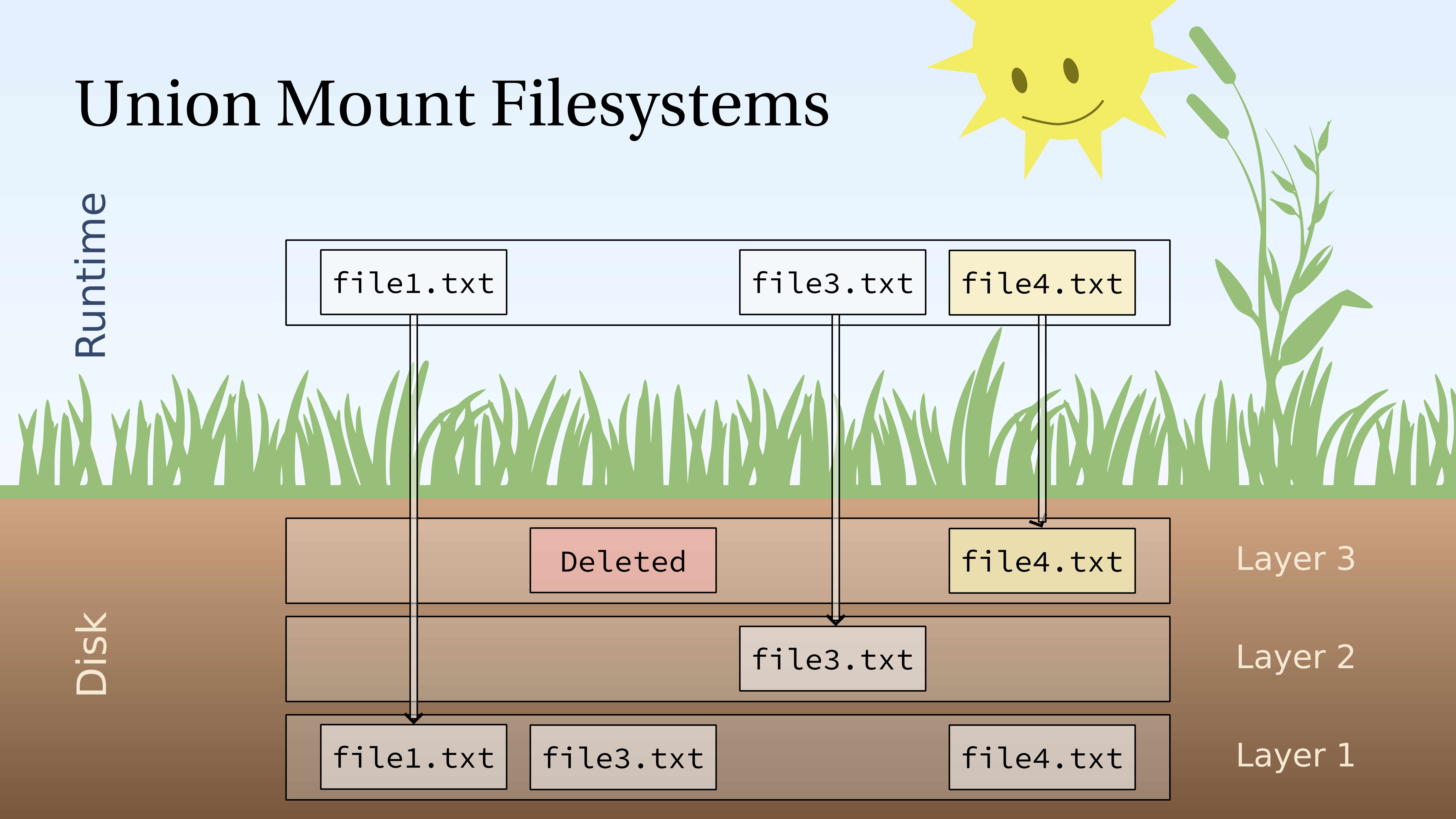
Filesystem state at t_0

Filesystem state at t_1

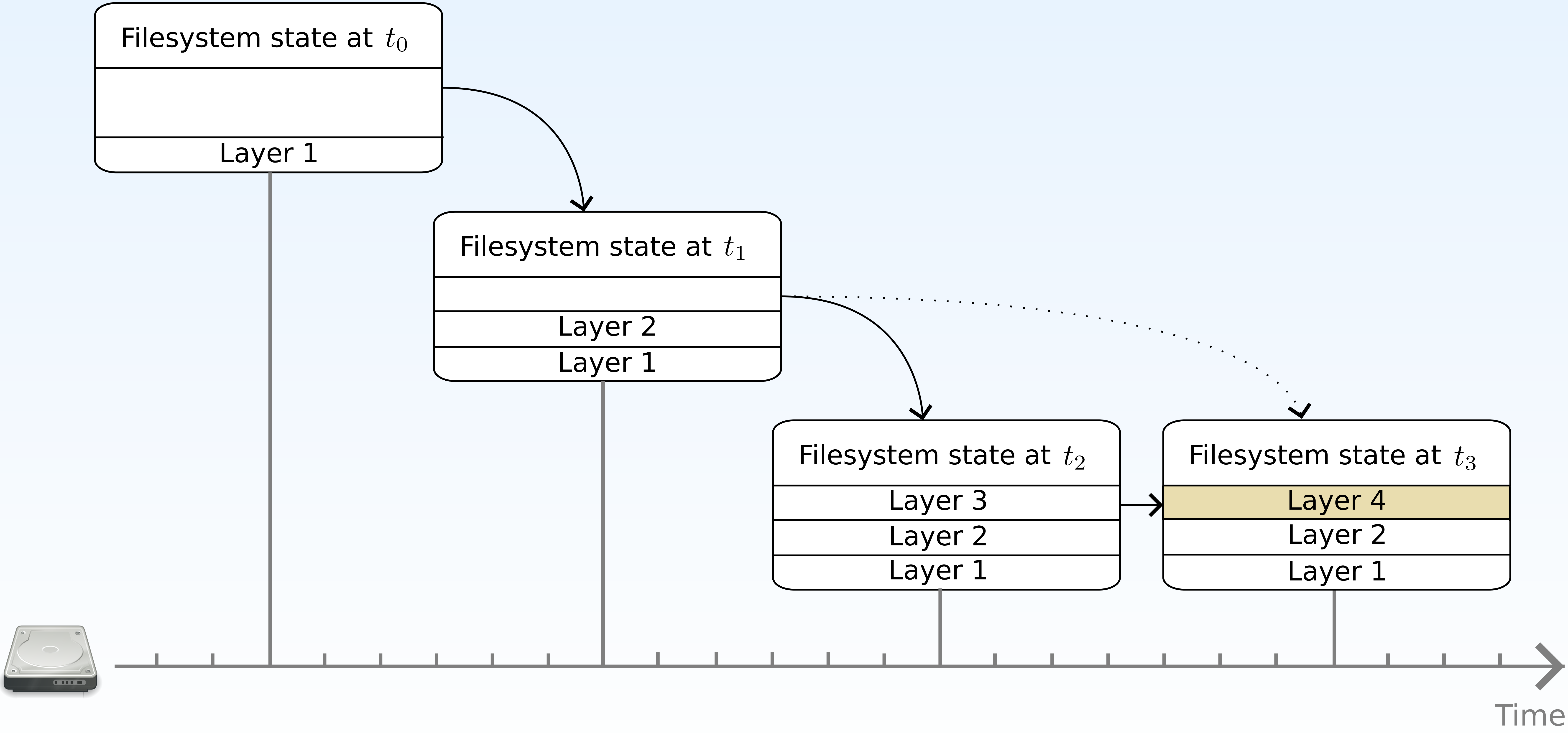
Filesystem state at t_2

Time

Union Mount Filesystems



Layers for Reusing Results of Execution



Filesystem state at t_0

Layer 1

Filesystem state at t_1

Layer 2

Layer 1

Filesystem state at t_2

Layer 3

Layer 2

Layer 1

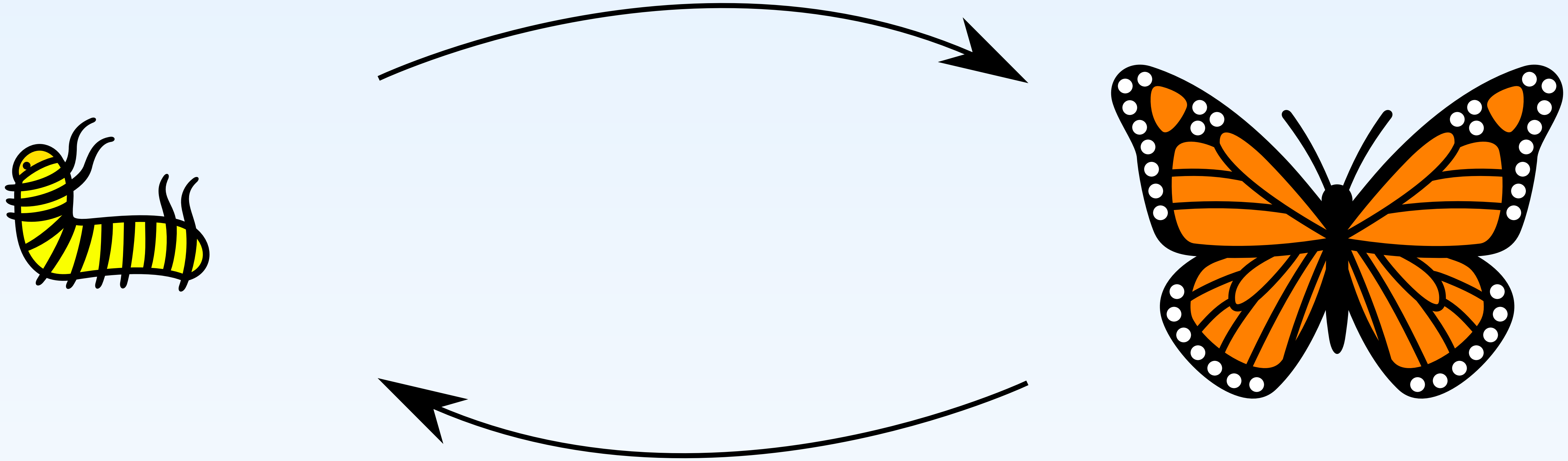
Filesystem state at t_3

Layer 4

Layer 2

Layer 1

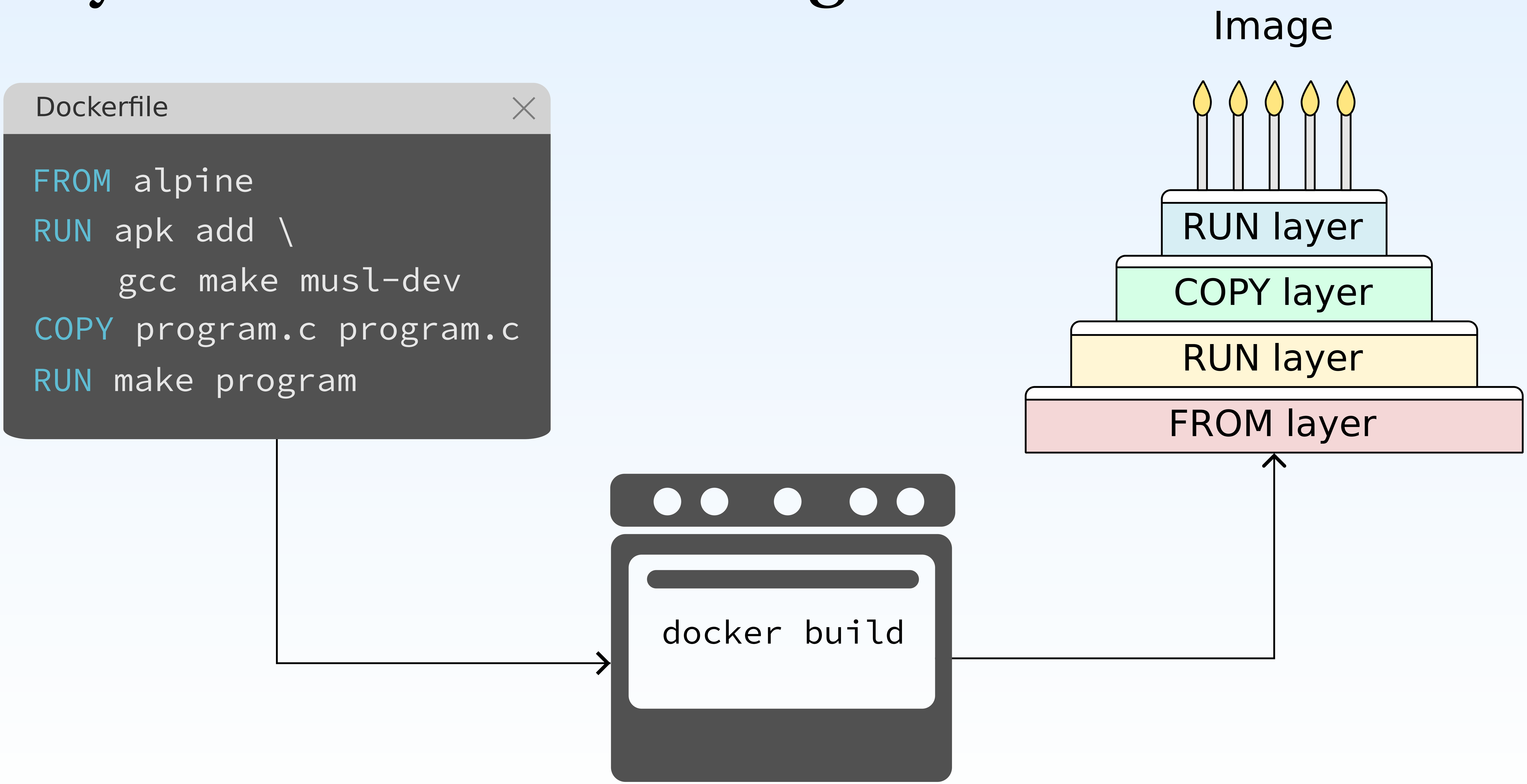
Images & Containers



Image, according to OCI standard, is a snapshot of a union mount filesystem with certain configuration files.

Container is an isolated user-space instance that looks like real computers from the point of view of programs running in it.

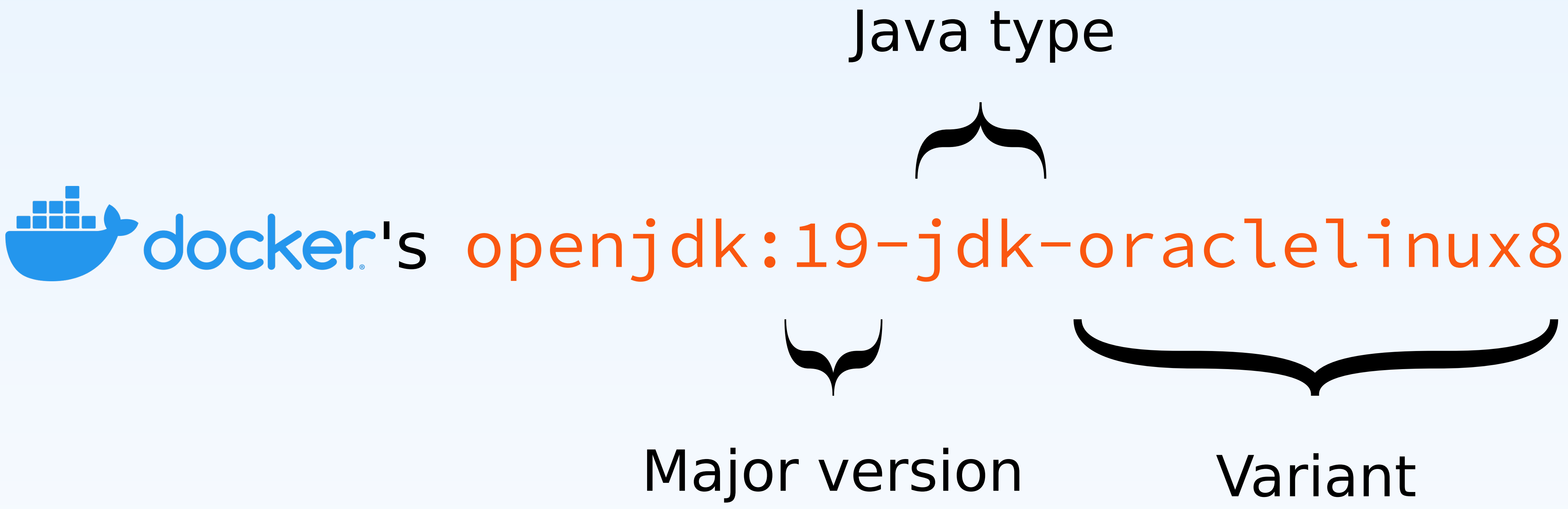
Layered Container Images



Dockerfile

```
FROM alpine
RUN apk add \
    gcc make musl-dev
COPY program.c program.c
RUN make program
```

Image Parameters



Java type

docker's `openjdk:19-jdk-oraclelinux8`

Major version

Variant

Ideal Containerisation Language

An ideal language should Dockerfiles

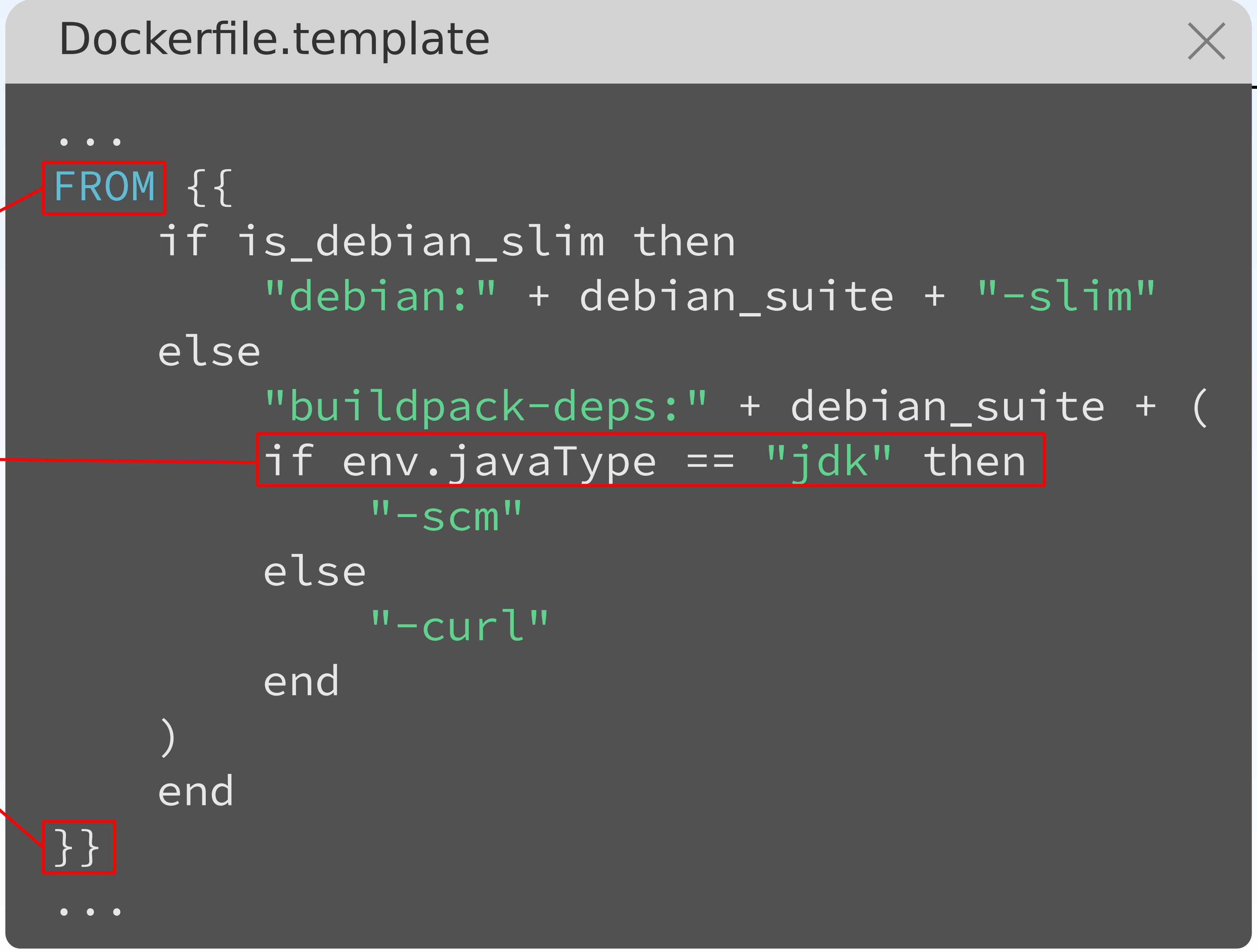
1. Be able to express interactions among build parameters. ✗
2. Be able to specify complex build workflows. ✗
3. Automatically parallelise builds.
4. Cache builds:
   a). Automatically restart failed executions;
   b). Automatically reuse shared parts of previous executions.
5. Help to reduce image size. ✗
6. Simplify maintenance:
   a). Provide zero-cost modularity and code reuse; ✗
   b). Be declarative;
   c). Be non-Turing-complete.

Build Parameters in Dockerfiles

docker's OpenJDK Images

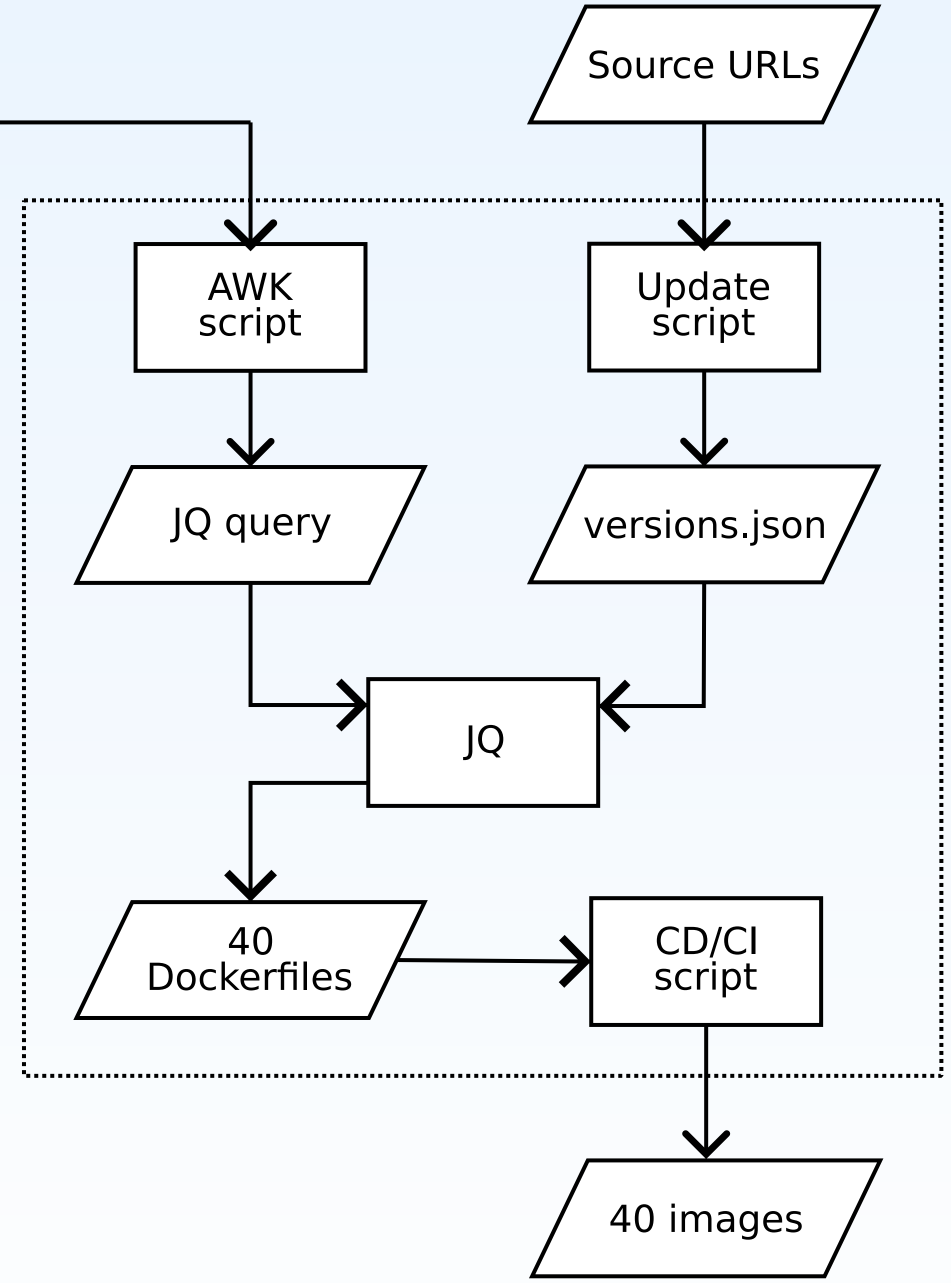
Three languages:

1. Dockerfile
2. JQ query
3. Syntax parsed by AWK

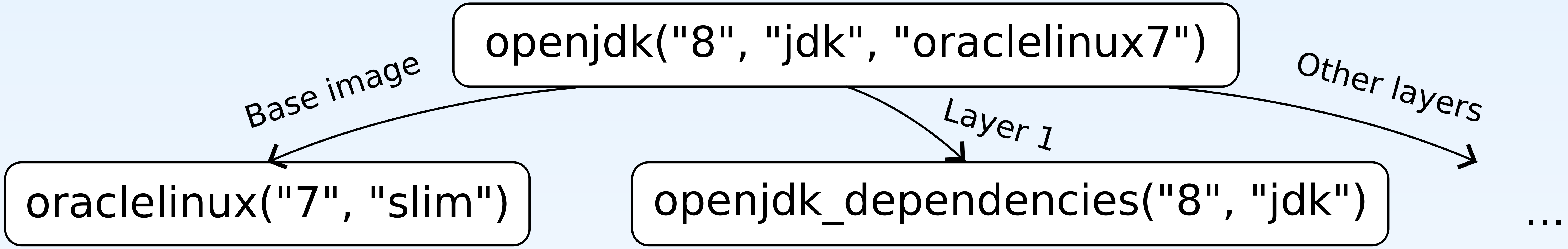


Dockerfile.template

```
...
FROM {{
    if is_debian_slim then
        "debian:" + debian_suite + "-slim"
    else
        "buildpack-deps:" + debian_suite + (
        if env.javaType == "jdk" then
            "-scm"
        else
            "-curl"
        end
    )
    end
}}
...
```



Modus: Containers as Deductive Database



openjdk("8", "jdk", "oraclelinux7")

Base image

Layer 1

Other layers

oraclelinux("7", "slim")

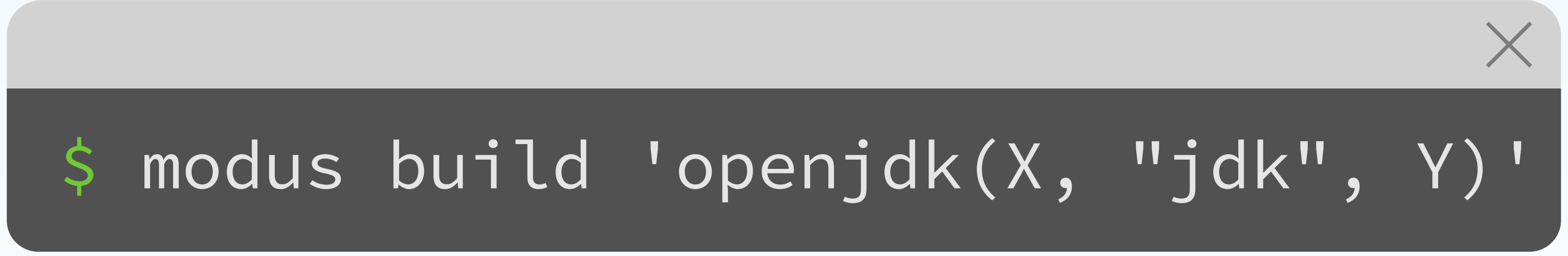
openjdk_dependencies("8", "jdk")

...

Images and layers are ground facts.

$$\begin{aligned}
&openjdk(major\_version, java\_type, variant) \leftarrow \\
&\quad openjdk\_base(variant), \\
&\quad openjdk\_dependencies(major\_version, java\_type), \\
&\quad \ldots
\end{aligned}$$

Build instructions are rules.



```
$ modus build 'openjdk(X, "jdk", Y)'
```

Images are built via queries.

Modus Syntax

Rules:

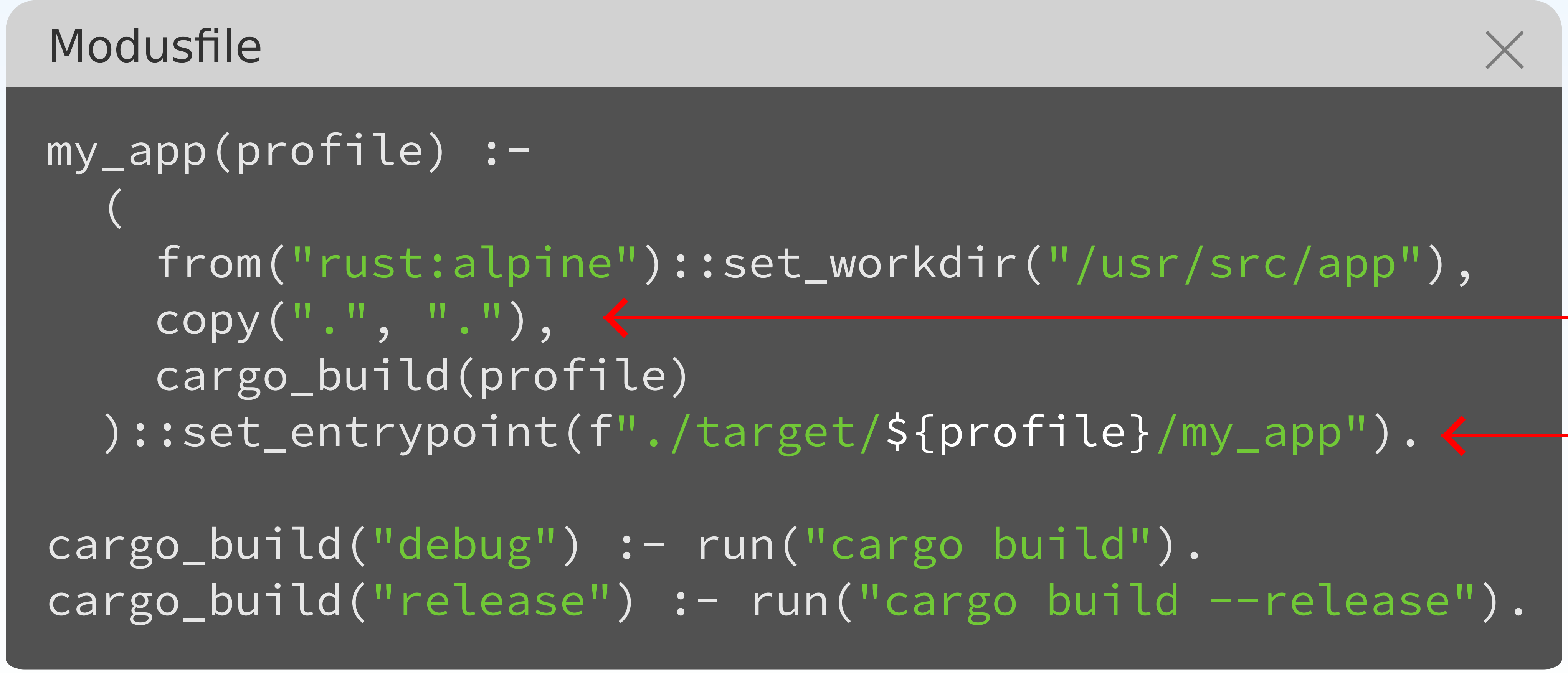
```
<IMAGE ATOM> :- <IMAGE EXPR>.
<LAYER ATOM> :- <LAYER EXPR>.
<LOGIC ATOM> :- <LOGIC EXPR>.
```

Image expression:

```
<IMAGE EXPR>, <LAYER EXPR>, ..., <LAYER EXPR>
<IMAGE ATOM>
<EXPR>::<IMAGE OPERATOR>
```

Layer expression:

```
<LAYER EXPR>, ..., <LAYER EXPR>
<LAYER ATOM>
<EXPR>::<LAYER OPERATOR>
```



Modusfile

```
my_app(profile) :-
  (
    from("rust:alpine")::set_workdir("/usr/src/app"),
    copy(".", "."),
    cargo_build(profile)
  )::set_entrypoint(f"./target/${profile}/my_app").

cargo_build("debug") :- run("cargo build").
cargo_build("release") :- run("cargo build --release").
```

Layer atom.

Image operator applied to image expression.

Modus Semantics: Proof Tree as Build DAG

```
app(base, "dev", target) :-
  dev_image(base),
  copy(".", "/app/"),
  make(target).

dev_image("alpine") :-
  from("alpine"),
  run("apk add gcc make").
dev_image("bullseye") :- from("gcc:bullseye").

app(base, "prod", "release") :-
  prod_image(base),
  app(base, "dev", "release")::copy("/app", "/app").

prod_image("alpine") :- from("alpine").
prod_image("bullseye") :- from("debian:bullseye-slim").

make("debug") :- run("cd /app/ && make debug").
make("release") :- run("cd /app/ && make").
```

```
└ app("alpine", "prod", "release")
  ├ prod_image("alpine")
  │ └ from("alpine")
  ├ (
  │ └ app("alpine", "dev", "release")
  │   ├ dev_image("alpine")
  │   │ ├ from("alpine")
  │   │ └ run("apk add gcc make")
  │   ├ copy(".", "/app/")
  │   └ make("release")
  │     └ run("cd /app/ && make")
  └ )::copy("/app", "/app")
```

Modus Is a Dialect of Datalog

Modus uses Datalog because it is:

1. expressive;
2. decidable, so it can always generate a minimal proof;
3. declarative, the success of solving does not depend on the ordering of clauses;

```
my_image(x) :-
  from("ubuntu"),
  layer1(x),
  layer2(x).
```

```
layer1(x) :-
  run("apt-get install vim"),
  x != "1".
```

```
layer2(x) :-
  run("apt-get install emacs"),
  x = "2".
```

In Prolog, the value of x
is not instantiated
at this location.

Non-Grounded Variables

Standard Datalog forbids non-grounded variables:

```
app(cflags) :-
  from("gcc:latest"),
  copy(".", "."),
  run(f"gcc ${cflags} test.c -o test").
```

A "fix" for standard Datalog:

```
supported_flags("-g").
supported_flags("").

app(cflags) :-
  from("gcc:latest"),
  copy(".", "."),
  run(f"gcc ${cflags} test.c -o test").
```

Modus solution: defer the evaluation of predicates with non-grounded variables until all of their arguments are bound to constants.

Parameter Dependencies

Fragment of OpenJDK template:

```
FROM {{
  if is_debian_slim then
    "debian:" + debian_suite + "-slim"
  else
    "buildpack-deps:" + debian_suite + (
    if env.javaType == "jdk" then
      "-scm"
    else
      "-curl"
    end
  )
  end
}}
```

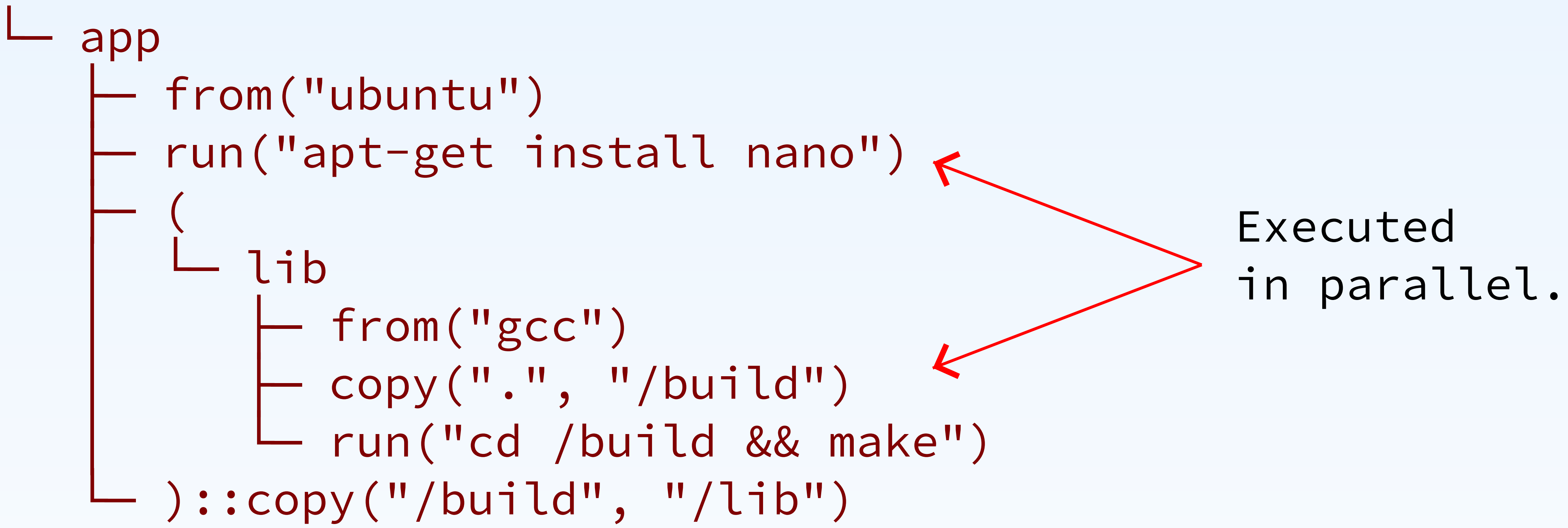
Corresponding fragment of Modusfile:

```
debian_image(VARIANT, JAVA_TYPE) :-
  (
    is_debian_slim(VARIANT, DEBIAN_SUITE),
    from(f"debian:${DEBIAN_SUITE}-slim")
  ;
    is_debian(VARIANT),
    debian_suffix_type(SUFFIX, JAVA_TYPE),
    from(f"buildpack-deps:${VARIANT}${SUFFIX}")
  ).
debian_suffix_type("-scm", "jdk").
debian_suffix_type("-curl", "jre").
```

Parallel Builds

```
app :-
  from("ubuntu"),
  run("apt-get install nano"),
  lib::copy("/build", "/lib").

lib :-
  from("gcc"),
  copy(".", "/build"),
  run("cd /build && make").
```



```
└─ app
   ├─ from("ubuntu")
   ├─ run("apt-get install nano")
   ├─ (
   │  └─ lib
   │     ├─ from("gcc")
   │     ├─ copy(".", "/build")
   │     └─ run("cd /build && make")
   └─ )::copy("/build", "/lib")
```

Executed
in parallel.

Caching

Cache is invalidated when any part of the script is changed:

```
FROM gcc:bullseye AS bullseye_dev_release
COPY program.c program.c
ARG TARGET
RUN if [ "$TARGET" = "debug" ] ; then \
      CFLAGS=-g make program ; \
    else \
      make program ; \
    fi
```

Cache is invalidated when a part relevant to build paramenters is changed:

```
app(target) :-
  from("gcc:bullseye"),
  copy("program.c", "program.c"),
  make(target).

make("debug") :-
  run("make program")::in_env("CFLAGS", "-g").
make("release") :- run("make program").
```

Optimising Image Size: Merging Layers

```
app(build_mode) :-
    from("gcc:latest"),
    (
        copy("src", "src"),
        make(build_mode),
        run("rm -rf src")
    )::merge.
make("release") :- run("cd src; make install").
make("debug") :- run("cd src; make -e install")::in_env("CFLAGS", "-g").
```

The operator ::merge is applied to a fragment of code to ensure that it will produce a single layer.

Optimising Image Size: Auxiliary Containers

```
copy_convert(file, dest) :-
    (
        from("debian:bullseye-slim"),
        run("apt-get update && apt-get install dos2unix"),
        copy(file, f"/tmp/${file}"),
        run(f"dos2unix /tmp/${file}")
    )::copy(f"/tmp/${file}", dest).
app :-
    from("debian:bullseye-slim"),
    copy_convert("my_local_script.sh", ".").
```

The operator ::copy is applied to copy a file converted to UNIX format from a temporary image.

Modularity & Code Reuse: Abstraction

```
install(lib, version) :-
    run(f"wget https://example.com/libs/${lib}-v${version}.tar.gz && \
          tar xf ${lib}-v${version}.tar.gz && \
          mv ${lib}-v${version}/ /build"),
    run("cd /build && make install"),
    run(f"rm ${lib}-v${version}.tar.gz && \
          rm -rf /build").
app :-
    from("gcc:latest"),
    install("liba", "1.3.5"),
    install("libb", "4.1").
```

The operator ::copy is applied to copy a file converted to UNIX format from a temporary image.

Modularity & Code Reuse: Standard Library

```
base(distr_version, python_version) :-
    semver_geq(distr_version, "16.04"),
    from(f"ubuntu:${distr_version}"),
    run(f"apt-get update && apt-get install -y python${python_version} \
          && rm -rf /var/lib/apt/lists/*").
```

Using the built-in predicate semver_geq to compare versions of Ubuntu.

```
app :-
    from("debian:bullseye-slim"),
    (
        run("apt-get update"),
        run("apt-get upgrade"),
        run("apt-get install build-essential")
    )::in_env("DEBIAN_FRONTEND", "noninteractive").
```

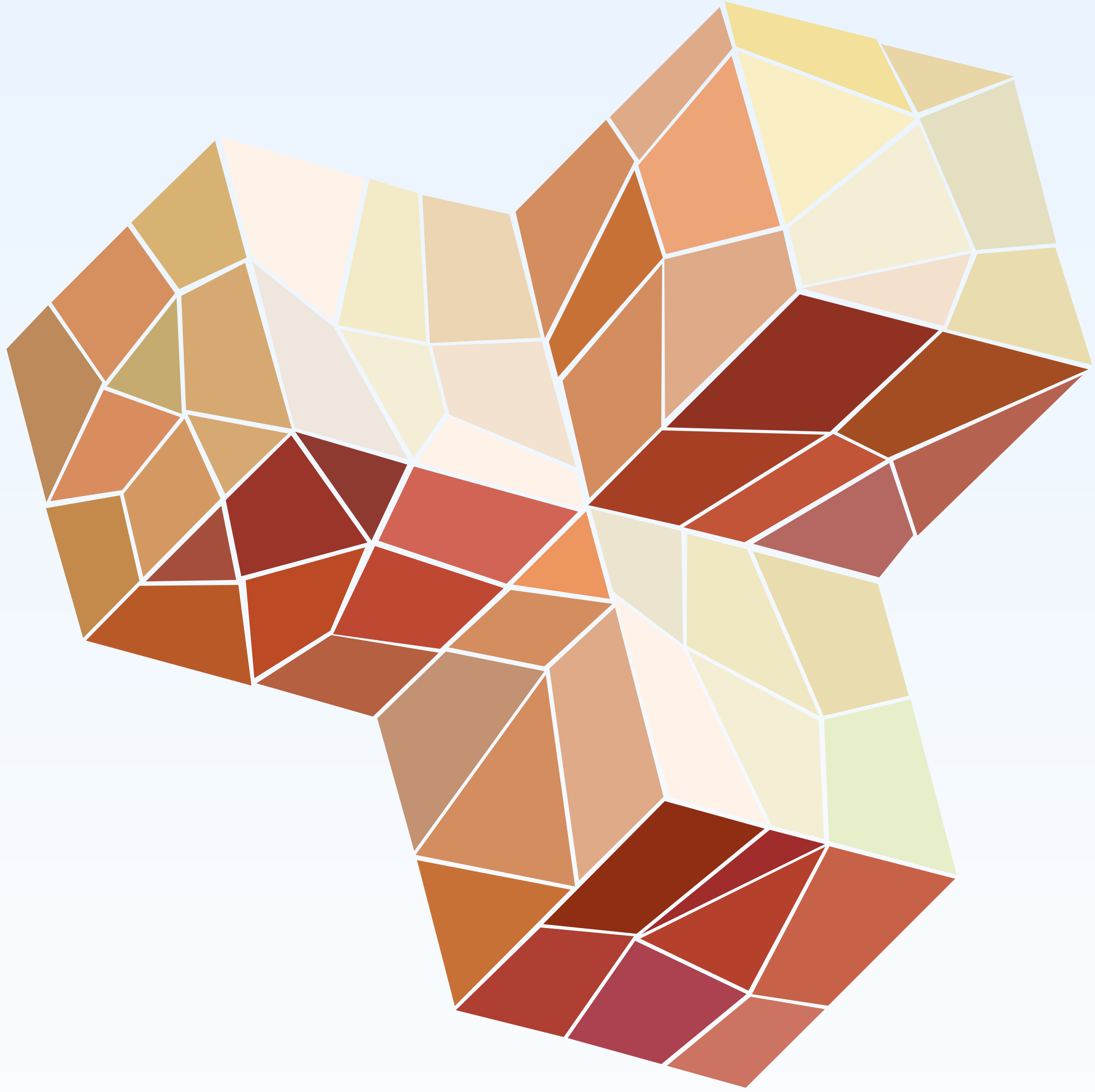
Using the built-in operator ::in_env to execute commands in a custom environment.

Evaluation on DockerHub Images

Project	Templating method	Outputs	Modusfile size
Ubuntu	bash	6	-6%
Redis	sed + awk	9	-13%
Nginx	sed	8	-21%
NodeJS	sed + awk	32	-19%
MySQL	awk + jq	4	-53%
Traefik	envsubst	4	-16%
OpenJDK	awk + jq	40	-52%

Modus significantly reduced the code size without sacrifycing speed and image efficiency.

Summary



Modus is a language for building Docker/OCI container images. It uses logic programming to express interactions among build parameters, specify complex build workflows, automatically parallelise and cache builds, help to reduce image size, and simplify maintenance.

Website: https://modus-continens.com/

Playground: https://play.modus-continens.com/

Documentation: https://docs.modus-continens.com/