
DESIGN
FAST
BLOOP
CONCURRENT
SERVER
BUILD
CHALLENGES

VEGAN

ORGANIC

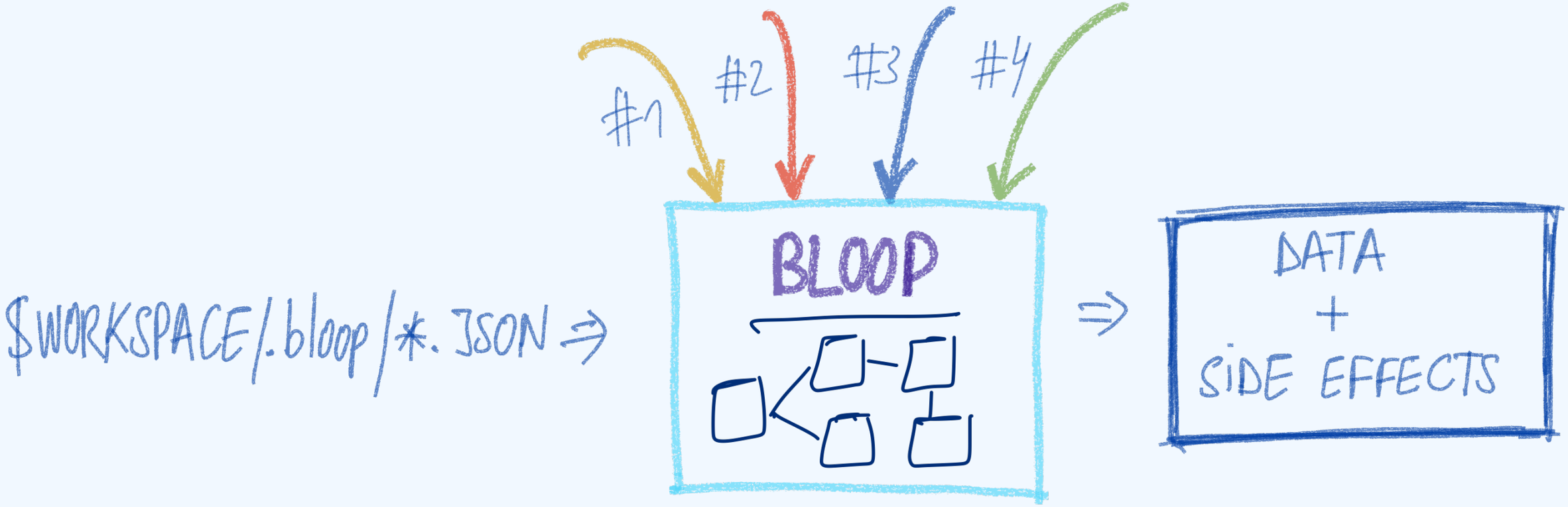
I RUN IN THE BACKGROUND OF YOUR MACHINE SERVICING REQUESTS AS FAST AS POSSIBLE. I WANT TO MAKE YOU **HAPPY**.

ENERGY FRIENDLY

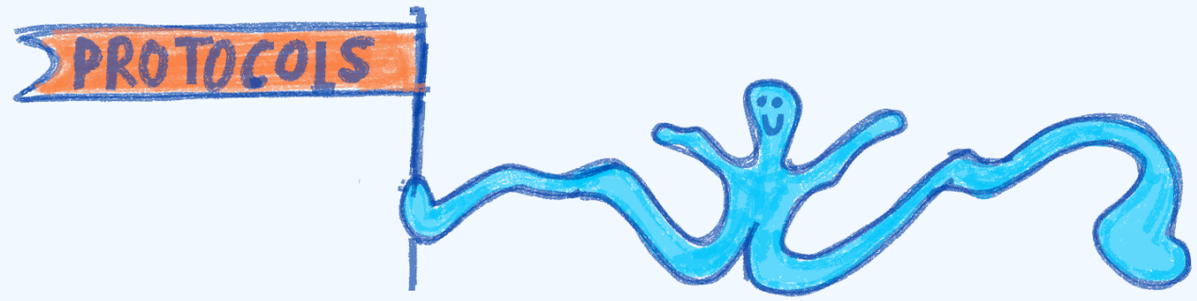
SUCH FAST

REAL FAST

bloop {compile, test, run, debug}



Highlights



Tooling as a service, focus on protocols < BSP, Nailgun

Integration-friendly < high quality docs; Tooling → launcher < binary, library

Integrates with editors, build tools, third-party tools.

A nice trade-off between centralization & decentralization,
in an environment with many constraints & particular culture.

Credits

Scala Center (Heather Miller, Sébastien Doeraene)

Martin Duhem., co-author of Bloop

Olafur Páll Geirsson, creator of Metals

Tomasz Godzik, Marek Żarnowski @ Virtuslab

Developers solving similar problems (Ensime, IntelliJ, ...)

External contributors & happy users

How does bloop understand your build?

build tools

sbt gradle
maven mill
Fury seed
pants

writes

*.json

FILE SYSTEM

reads

BLOOP

INCREMENTAL STATE

DAG

daemon

bidirectional

metals
intellij
...?

Supports:

- Fast import project
- Updates on build changes

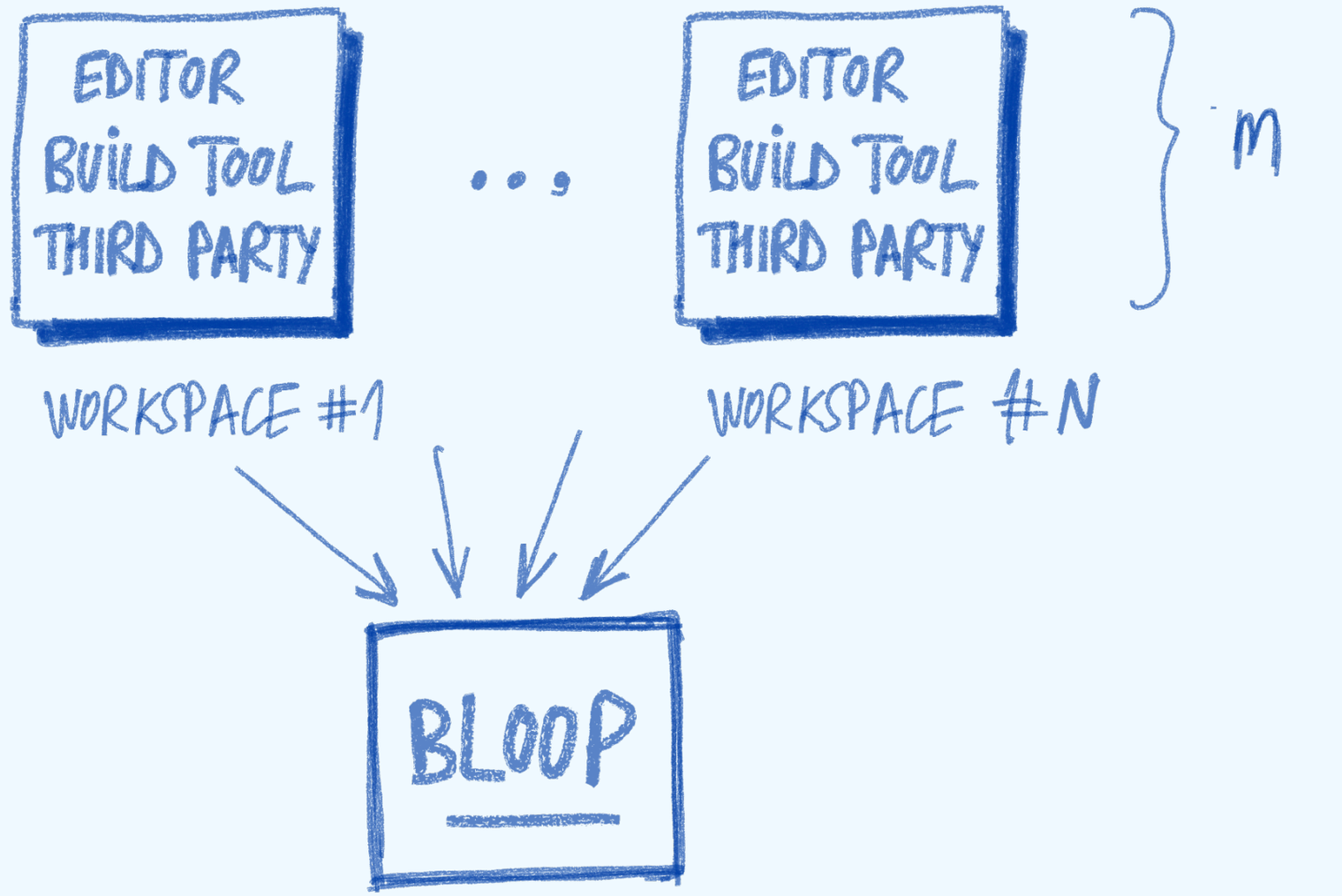
1 USER

- HIGH EXPECTATIONS #1
- WANTS TO BE BLOCKED BY ITS THINKING #2
- LAZY, DOESN'T READ DOCS #3

independent clients

N CLIENTS = k · m

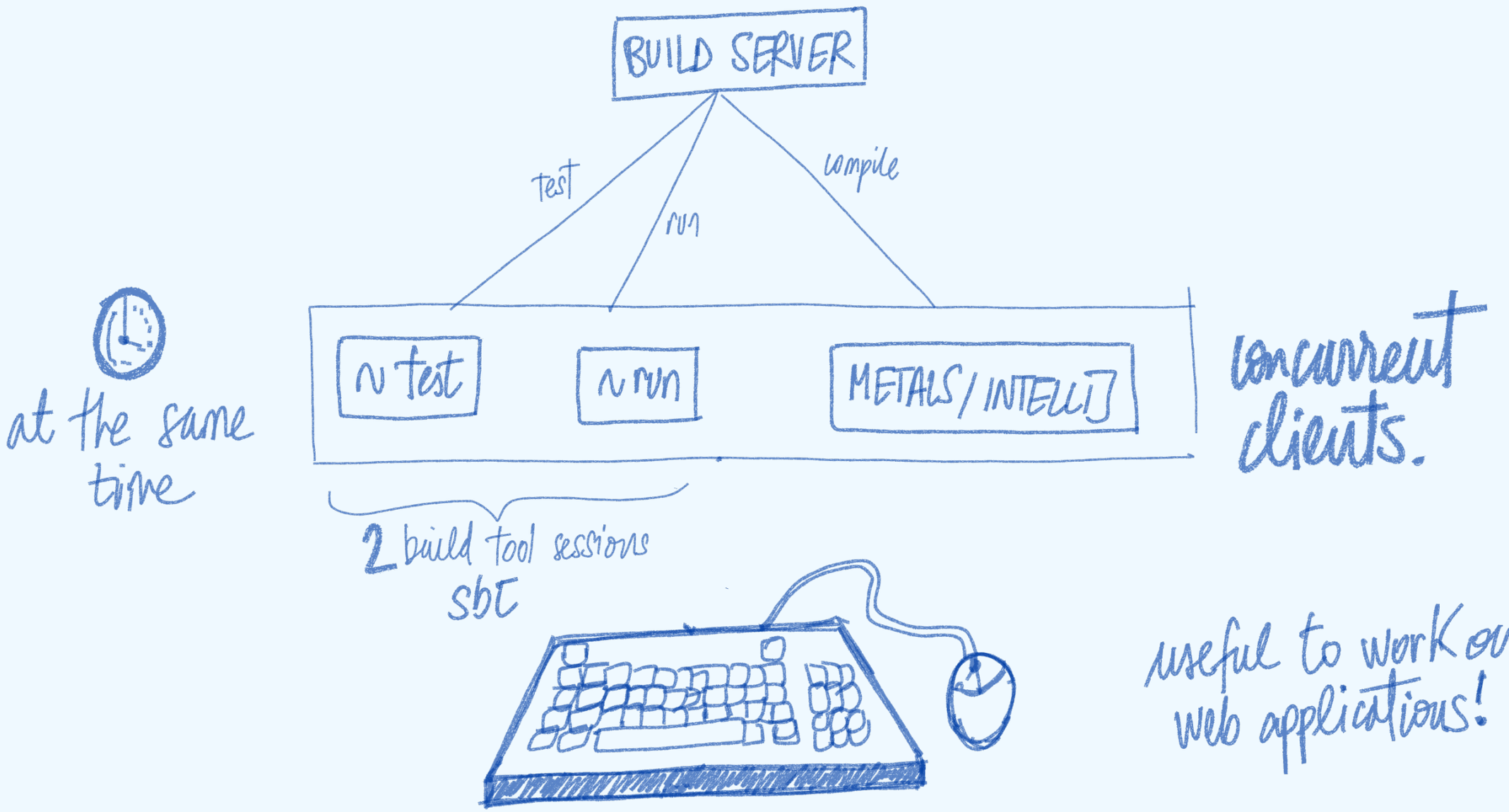
k WORKSPACES
m TOOL INSTANCES



1 BLOOP

Case study: concurrent clients.

currently not possible!



at the same time

concurrent clients.

useful to work on web applications!

Semantics of a build server?

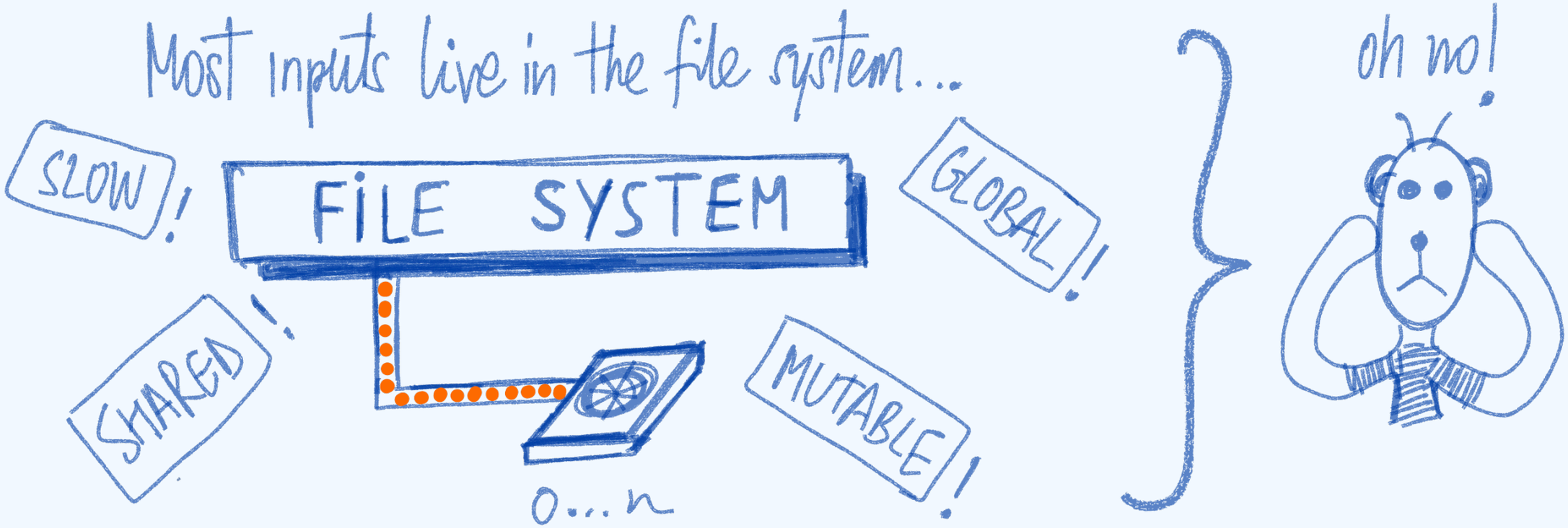
What if...

- Two clients compile at the same time?
- Clients cancel compilation?
- A client compiles when another is running Tests?

Compile Inputs
(low-level)

- sources (*.scala, *.java)
- resources (generated or non-generated)
- classpath (provided by build tool)
- scala version and options
- project metadata (more generally)
- internal state.

Most inputs live in the file system...



2 properties to rule them all

Compile deduplication

same inputs = same outputs

requires emulating a compilation

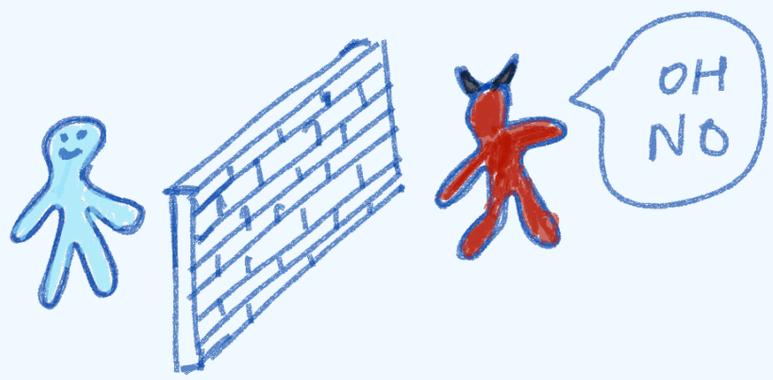
motivated by efficiency

Compiler isolation

different inputs ⇒ different outputs

requires inputs to be { immutable if shared / mutable if unique }

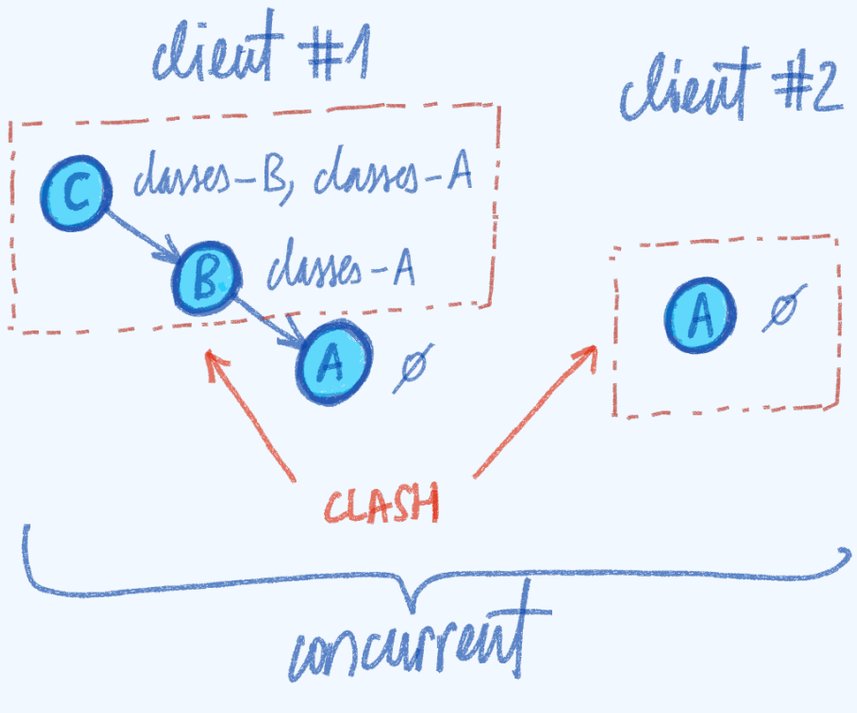
motivated by correctness



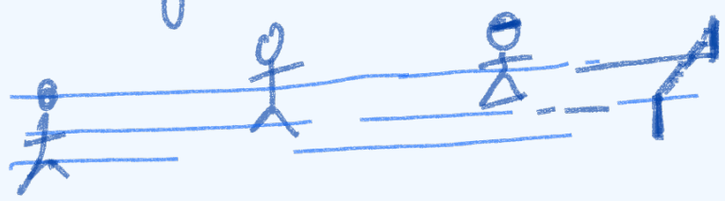
Compiler isolation

Challenges

- Compiling overlapping DAG subgraphs
  Classpath contains references to current dirs.



- Accessing latest compiler state in server



RACE CONDITIONS!

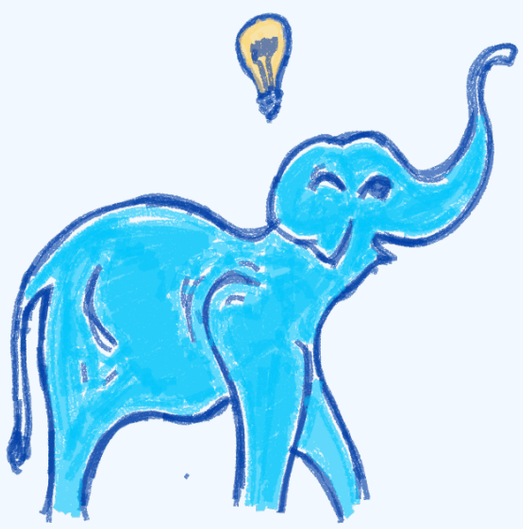
- Persisting latest state of a project in-disk.

Sounds familiar?

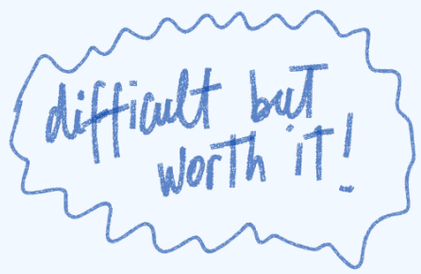
We need { **A**tomicity, **C**onsistency, **I**solation, **D**urability }

Solutions for compiler isolation

Blocking < waste of resources, doesn't scale
nobody likes unnecessary blocking ☹



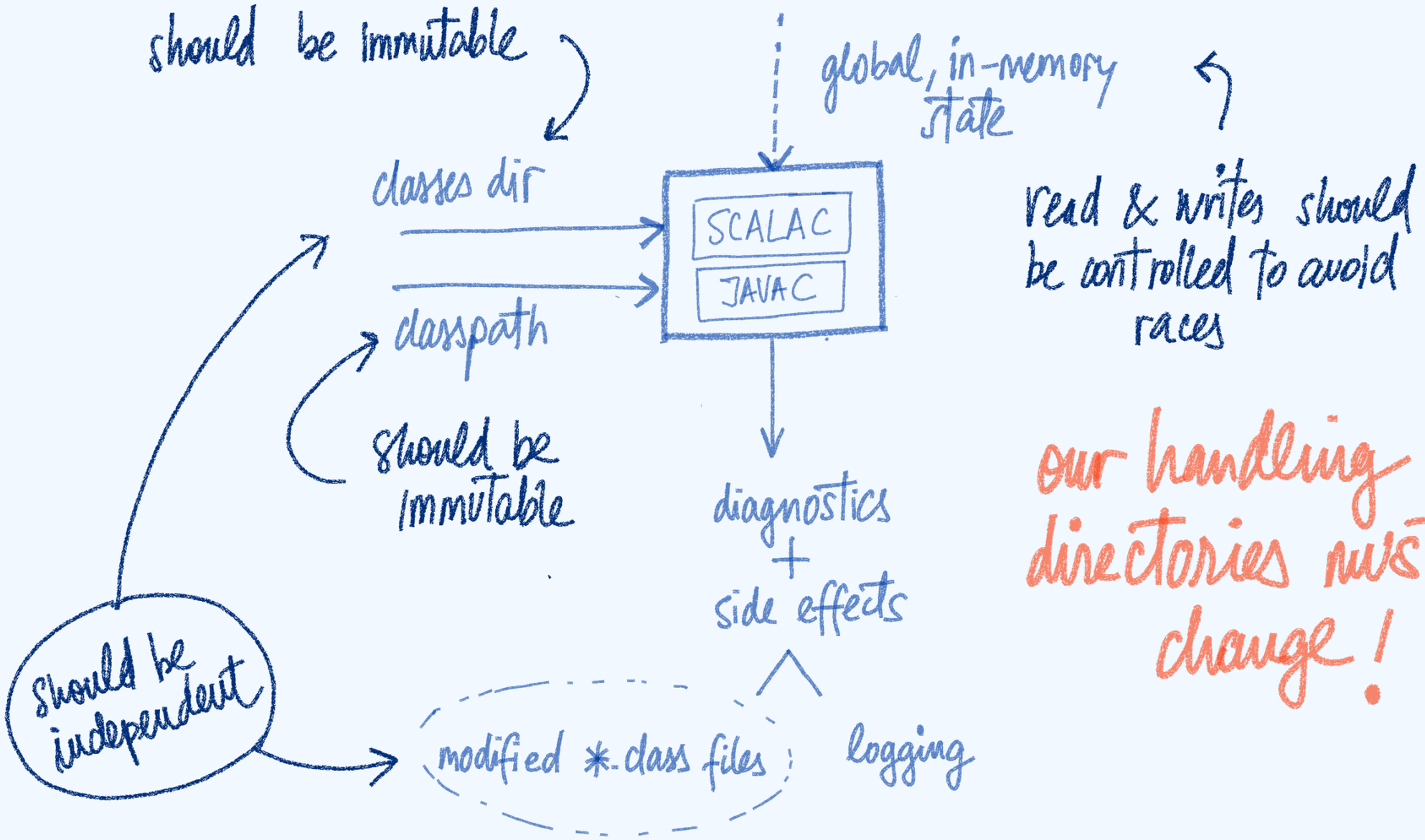
Non-blocking



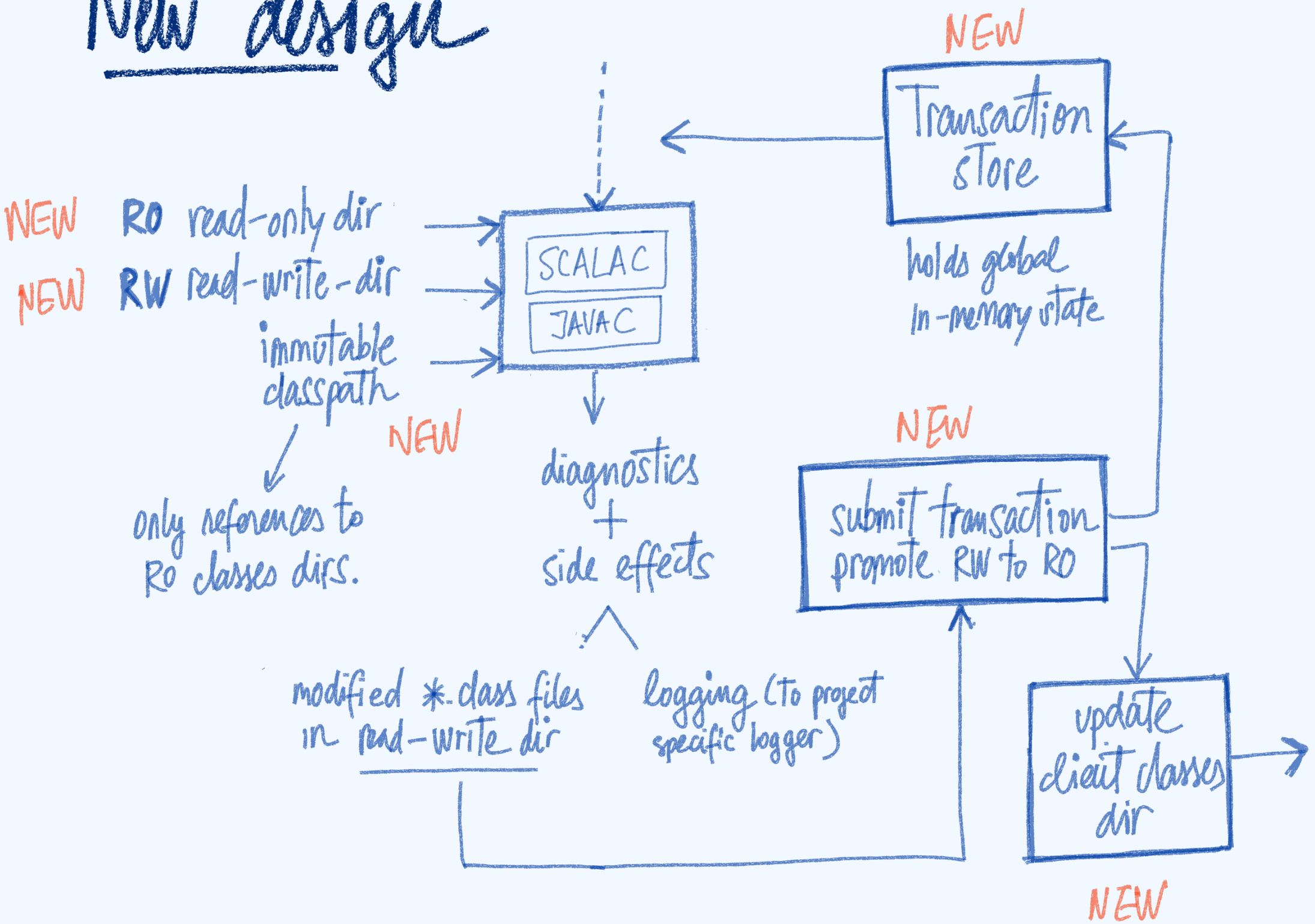
- devise clever way to avoid sharing
- use transaction-based compilation engine to provide ACID semantics
- implies independent outputs
  ⇒ make approach fast

Avoid sharing state < files → compilation outputs
in-memory mutable data

Avoid sharing state < files → compilation outputs
in-memory mutable data



New design


NEW
Transaction store
holds global in-memory state
NEW RO read-only dir
NEW RW read-write-dir
immutable classpath
only references to RO classes dirs.
SCALAC
JAVAC
NEW
diagnostics + side effects
NEW
submit transaction promote RW to RO
update client classes dir
NEW
modified *.class files in read-write dir
logging (To project specific logger)

Compiler isolation

Every client has its own unique set of directories.
They get populated by the compilation engine (in parallel).

Build tool generating build files keeps the original dirs.

Compilation engine works with read-only directories
New read-write directory promoted to read-only when result is cached.

Compile deduplication

Avoids extra compilations when inputs are the same.

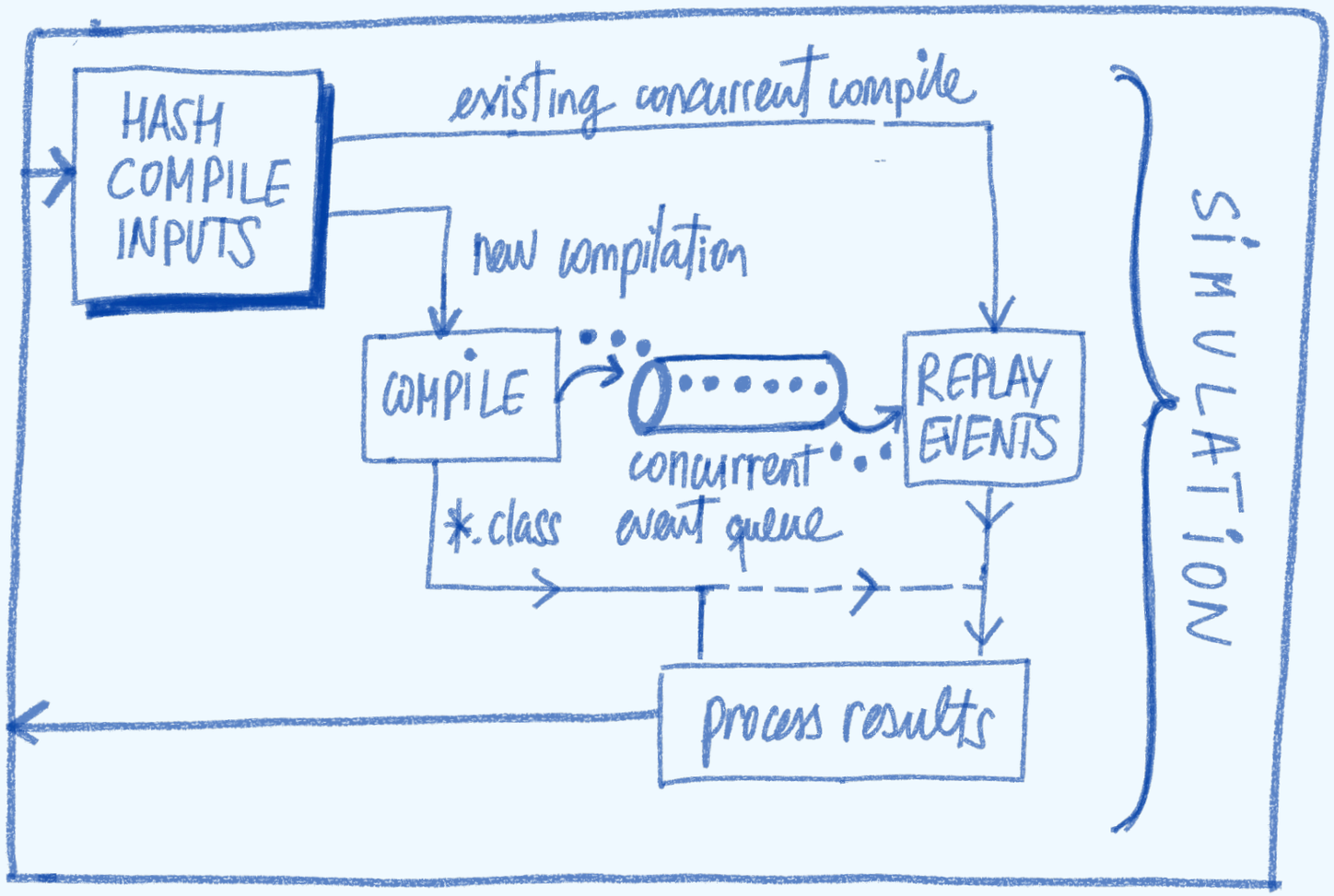
- Crucial for real-world use cases: several clients sending many compile requests for the same build at the same time

Key idea = simulate compilation

Compile deduplication

All clients should perceive the same behavior

Simulation depends on client data



Offloading compilation from sbt

sbt shell

> test

> ~ run

(sbt) compile
- make sure bloop configs are up-to-date
- ask bloop to compile via BSP

Seamless integration means...

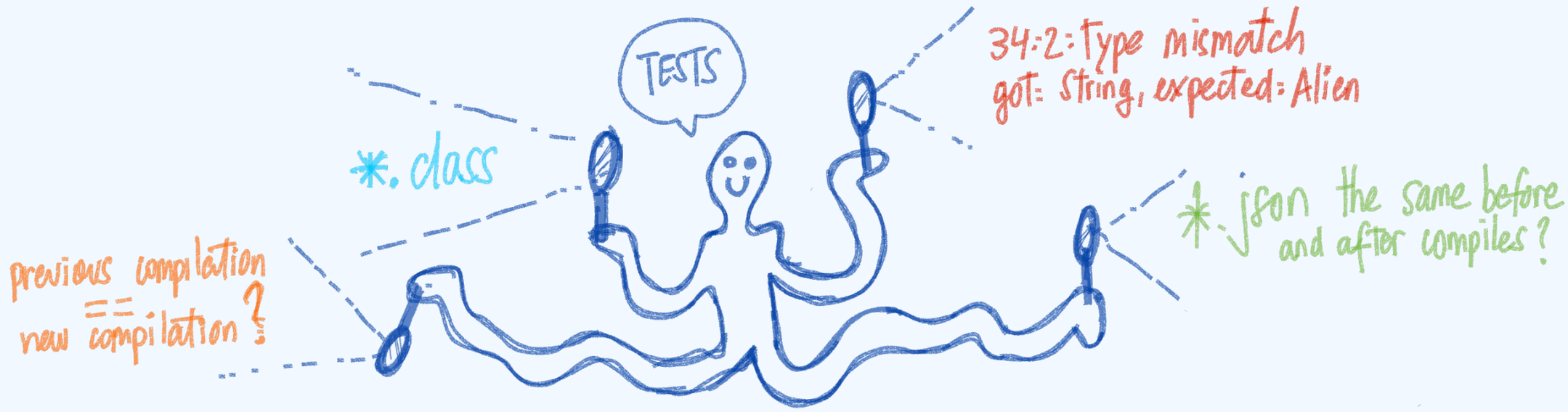
- People don't need to change their workflows
- Compilations are shared with other clients.

Testing the build server

Treats build server as a blackbox and tests all build side effects.

Focus on compiler invariants before & after build actions.
- Compiler diagnostics, compilation products (+ metadata), etc.

Randomized tests of concurrent clients simulating real-world scenarios
~ **11** KLOC of tests for (protocol, platform, client type).

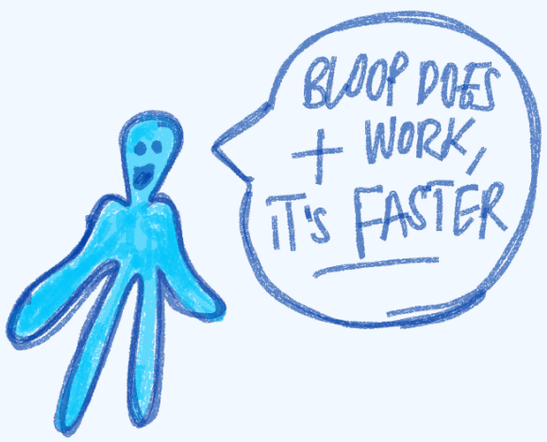


Performance evaluation

Basic numbers.
(for macOS)

~ 200 - 500 ms incremental compiles
in medium-sized codebases

~ 500 ms full no-op compiles
in big codebases + 200 modules



Concrete numbers

AKKA

sbt	2 minute, 25 seconds
bloop	1 minute, 2 seconds

GUARDIAN / FRONTEND

sbt	44 seconds
bloop	29 seconds

CIRCE

sbt	1 minute, 5 seconds
bloop	50 seconds

HTTP4S

sbt	48 seconds
bloop	43 seconds

ATLAS

sbt	9 seconds 10 ms
bloop	9 seconds 950 ms

Oracle Java 8
Linux, Intel i7 @ 3.5GHz

:)!

GC-friendly

Architecture

Order-of-magnitude less usage of resources

Mitigates overhead of Scala versions & platforms

Smarter scheduling

Fast task engine

Implementation

Parallel IO

Fancy use of compiler

Reliable caching

Where does Bloop speed come from?

(even if it does more work?!)

Caching

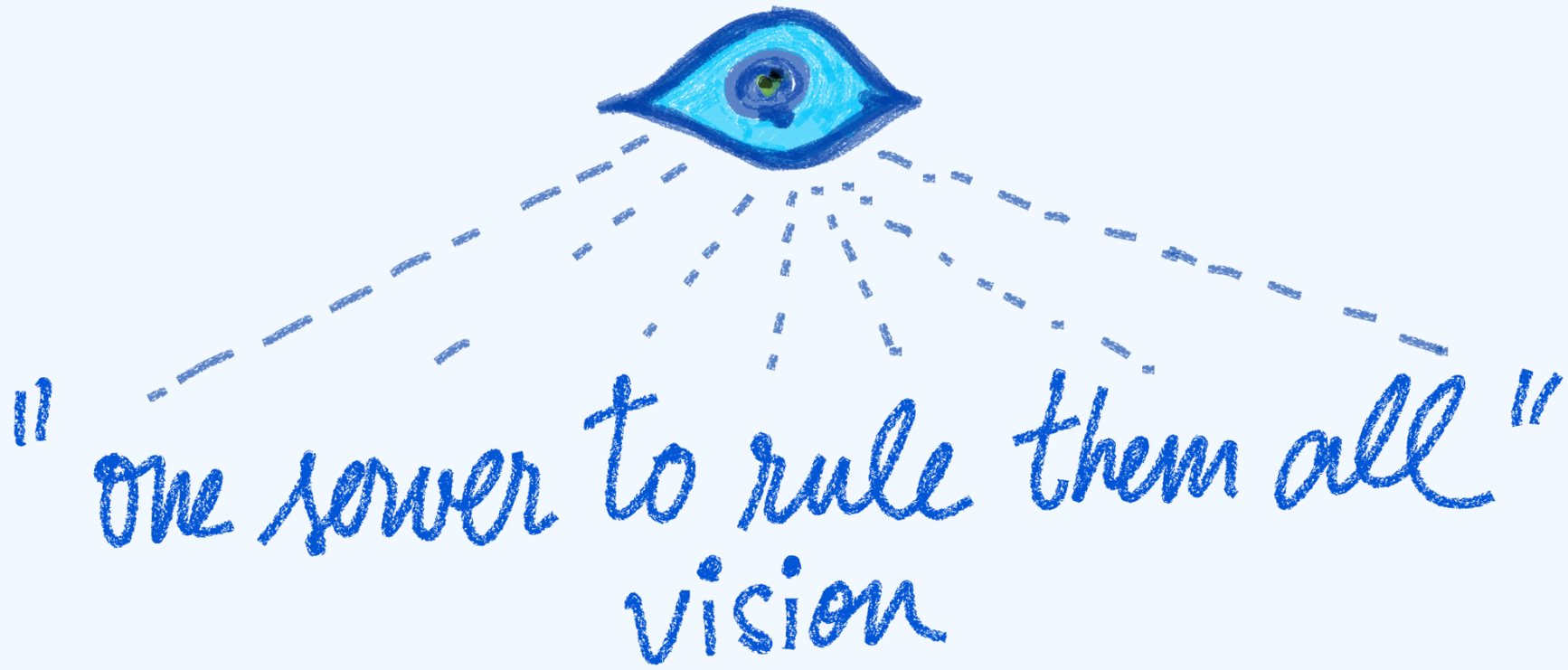
Caches compile inputs heavily < Sources
                                Classpath

But reliably: no dependency on file watching

Caches compiler plugins classloaders automatically

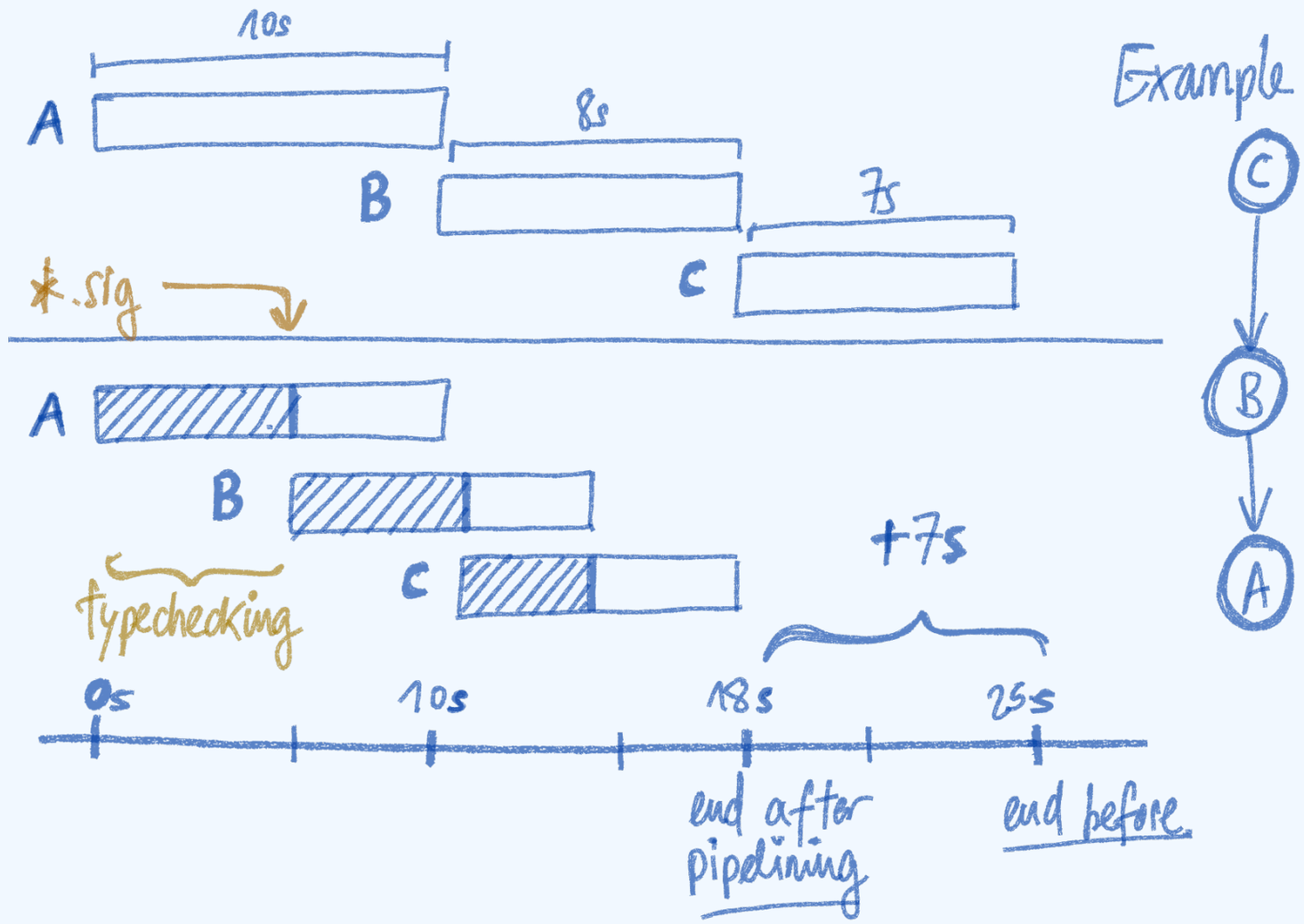
upcoming: will also cache macro classloaders

Bloop's good defaults policy


"one server to rule them all"
vision
users
win
win
tool
maintainers
compiler
engineers

Build pipelining

A compilation technique to speed up the compilation time of a build graph by starting compilation of dependencies sooner.



Example build.

Build pipelining

Upcoming blog post about it...

Example of a build-related technique that we can push directly to our users. *

Summary

Bloop is pushing the state-of-the-art of build servers to make Scala developers more productive.

Result? A tool that works & integrates with the existing ecosystem out-of-the-box.

Give it a try! scalacenter.github.io/bloop

Running, testing & Debugging from the editor

Build pipelining enabled by default

Future work: 1.4.0

GraalVM client binary & shaded launcher (no more installation)

Offloading compilation from sbt

Thank you!

THE END