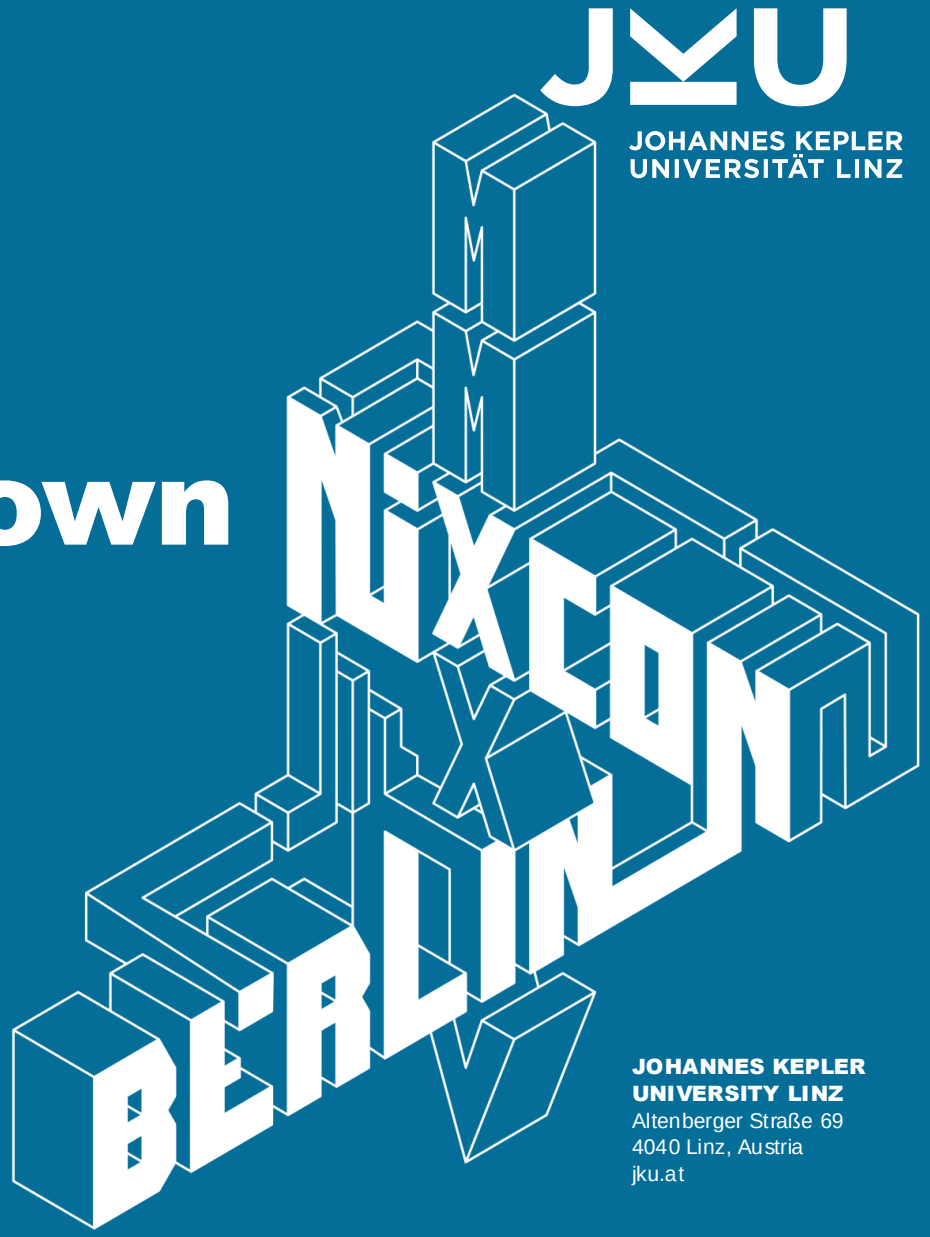
NixCon '24

hashes all the way down

Martin Schwaighofer
Johannes Kepler University Linz
2024-10-25, Berlin (Germany)



JOHANNES KEPLER
UNIVERSITÄT LINZ

JOHANNES KEPLER UNIVERSITY LINZ
Altenberger Straße 69
4040 Linz, Austria
jku.at

Cloud Build Systems

Build Systems à la Carte

ANDREY MOKHOV, Newcastle University, United Kingdom
NEIL MITCHELL, Digital Asset, United Kingdom
SIMON PEYTON JONES, Microsoft Research, United Kingdom

Build systems are awesome, terrifying – and unloved. They are used by every developer around the world, but are rarely the object of study. In this paper we offer a systematic, and executable, framework for developing and comparing build systems, viewing them as related points in landscape rather than as isolated phenomena. By teasing apart existing build systems, we can recombine their components, allowing us to prototype new build systems with desired properties.

CCS Concepts: • **Software and its engineering**; • **Mathematics of computing**;

Additional Key Words and Phrases: build systems, functional programming, algorithms

ACM Reference Format:
Andrey Mokhov, Neil Mitchell, and Simon Peyton Jones. 2018. Build Systems à la Carte. *Proc. ACM Program. Lang.* 2, ICFP, Article 79 (September 2018), 29 pages. https://doi.org/10.1145/3236774

JKU JOHA
UNIVE

Extending Cloud Build Systems to Eliminate Transitive Trust

Extending Cloud Build Systems to Eliminate Transitive Trust

Martin Schwaighofer
martin.schwaighofer@ins.jku.at
Johannes Kepler University Linz
Linz, Austria

Michael Roland
michael.roland@ins.jku.at
Johannes Kepler University Linz
Linz, Austria

René Mayrhofer
rm@ins.jku.at
Johannes Kepler University Linz
Linz, Austria

Abstract

Trusting the output of a build process requires trusting the build process itself, and the build process of all inputs to that process, and so on. Cloud build systems, like Nix or Bazel, allow their users to precisely specify the build steps making up the intended software supply chain, build the desired outputs as specified, and on this basis delegate build steps to other builders or fill shared caches with their outputs. Delegating build steps or consuming artifacts from shared caches, however, requires trusting the executing builders, which makes cloud build systems better suited for centrally managed deployments than for use across distributed ecosystems. We propose two key extensions to make cloud build systems better suited for use in distributed ecosystems. Our approach attaches metadata to the existing cryptographically secured data structures and protocols, which already link build inputs and outputs for the purpose of caching. Firstly, we include builder provenance data, recording which builder executed the build, its software stack, and a remote attestation, making this information verifiable. Secondly, we include a record of the outcome of how the builder resolved each dependency. Together, these two measures eliminate transitive

1 Introduction

1.1 Motivation

Securely building, deploying, and maintaining software packages, which practically always, but not always visibly, have deep hierarchies of dependencies, is not a solved issue. In these deep hierarchies we must not only consider direct dependencies, but also

- transitive dependencies, including build tools like compilers,
- as well as the hosts building these dependencies,
- and so forth, recursively.

This is necessary to guard against backdoored dependencies, including toolchains [19]. Sadly, in practice we often have to optimistically trust build hosts with dependency specification and resolution, and trust the software stack of the build hosts themselves. As a consequence, our ability to react to security issues deep within these hierarchies is severely lacking. See Figure 1 which illustrates the concept of transitive trust.

As part of our efforts to prevent a compromise in the supply chain we have to consider not only

JKU JOHA
UNIVI

Cloud Build Systems: Output Lookup by Hash I

Definition I

Cloud build systems construct a **dependency tree** in which each node is identified by a content or input hash.

Terminal inputs, which are leaves in the dependency tree, for example source files or binary blobs, are referred to by **content hash** (a hash of their contents).

The inner nodes of the dependency tree are build steps, which are always identified by **input hash** (a hash of their input set).

...

Cloud Build Systems: Output Lookup by Hash II

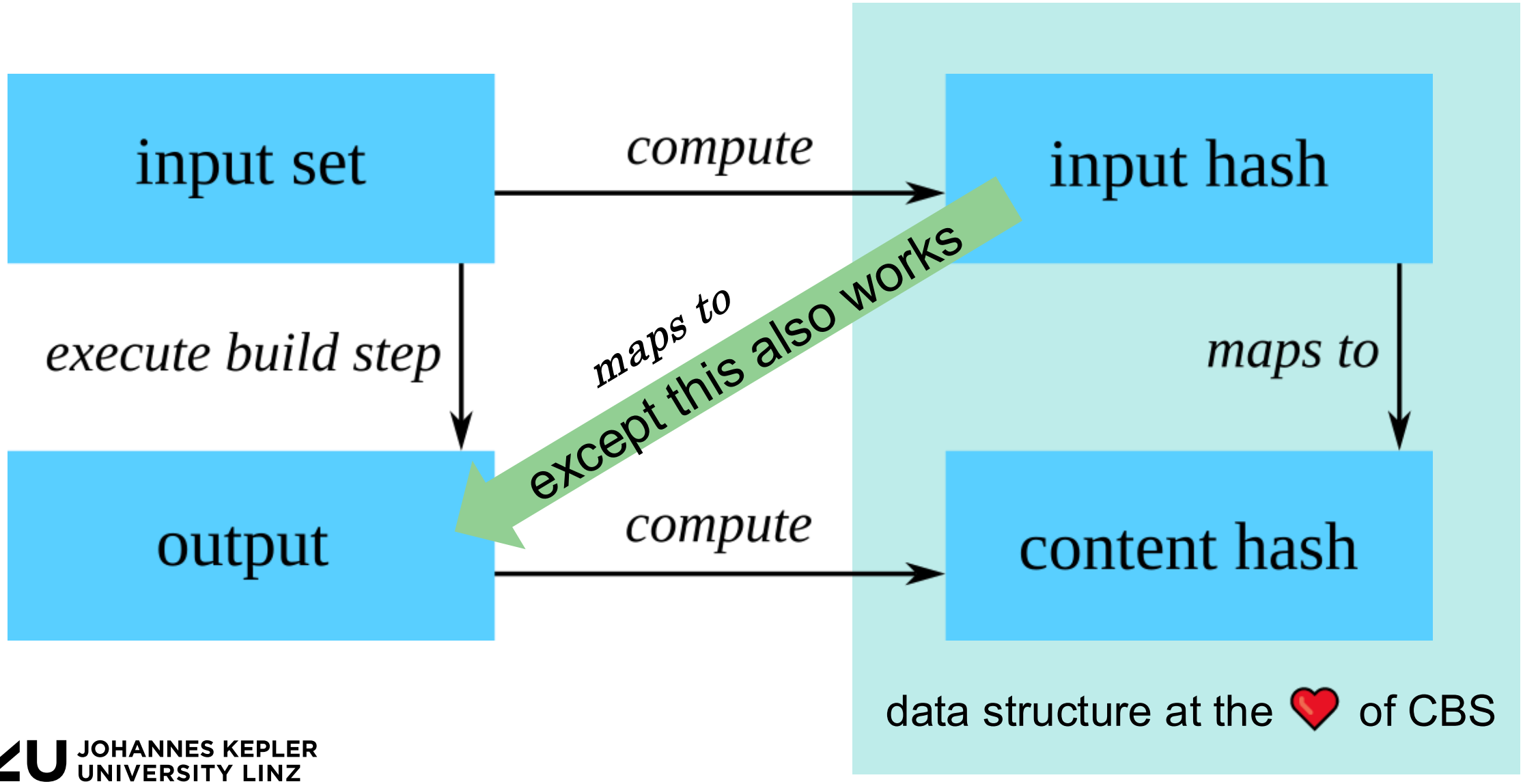
Definition II

The input set of a build step consists of the build instructions that get executed during the specific step, including any terminal inputs, and either

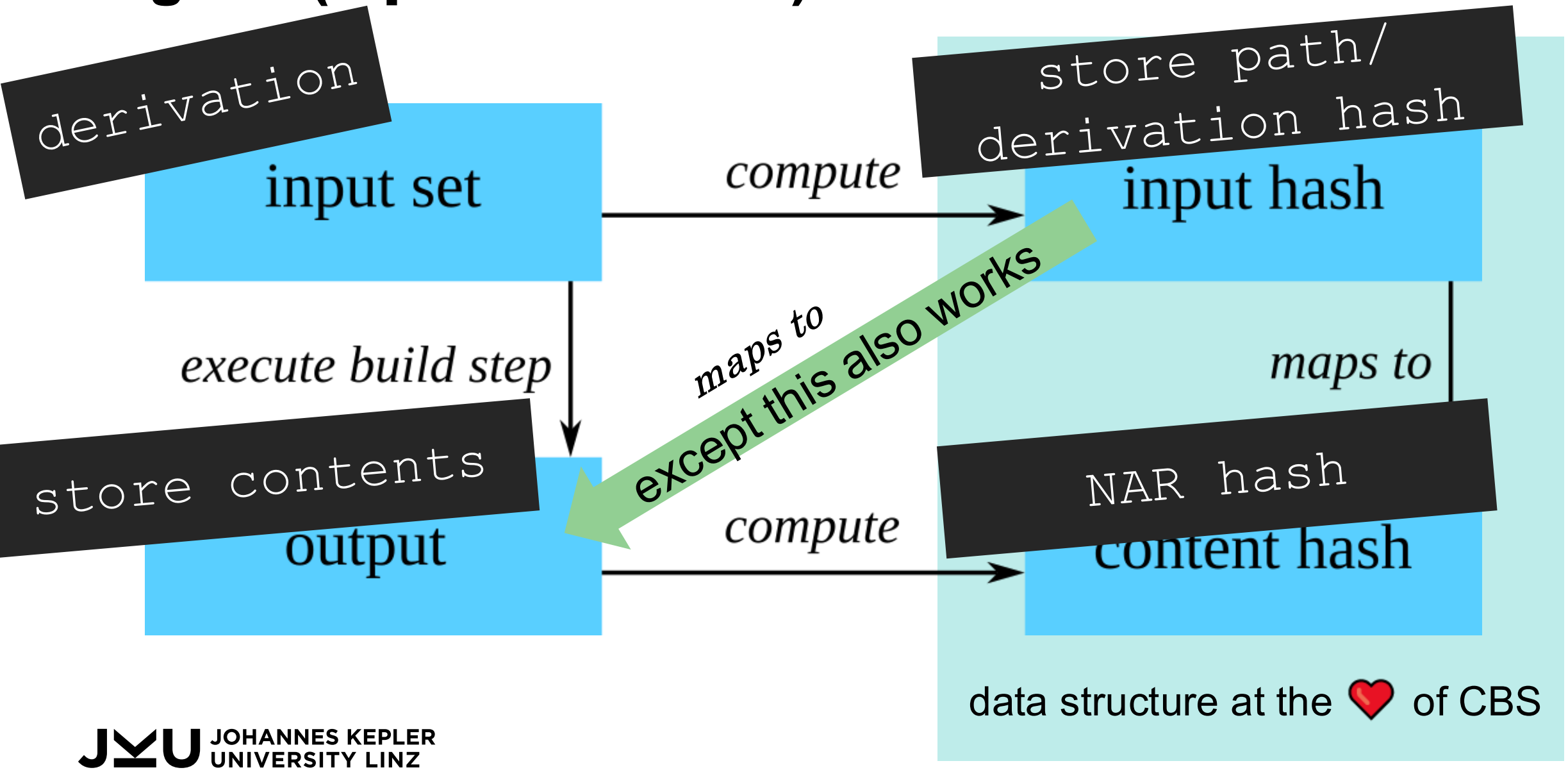
a) the content hashes of the outputs obtained by building all direct dependencies, or
b) the input hashes of all dependencies, including transitive dependencies via recursion.

The two tracing approaches are known as using **constructive traces** (a), and **deep constructive traces up to terminal inputs** (b).

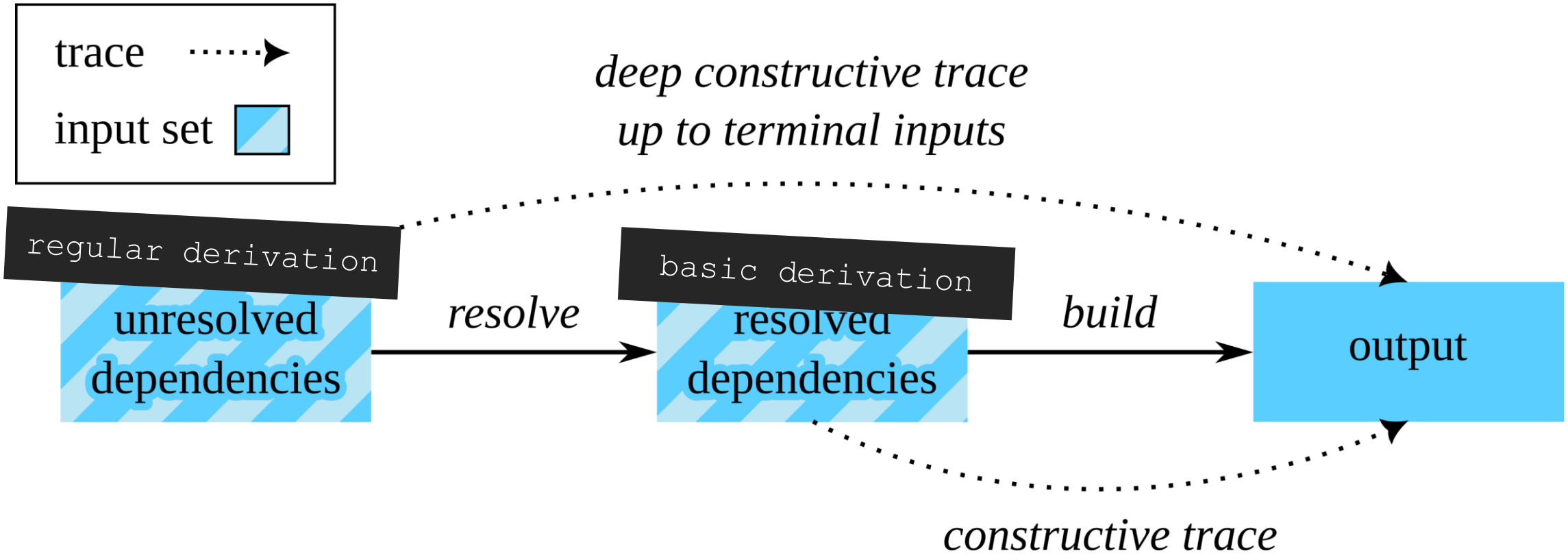
Cloud Build System Terms Illustrated



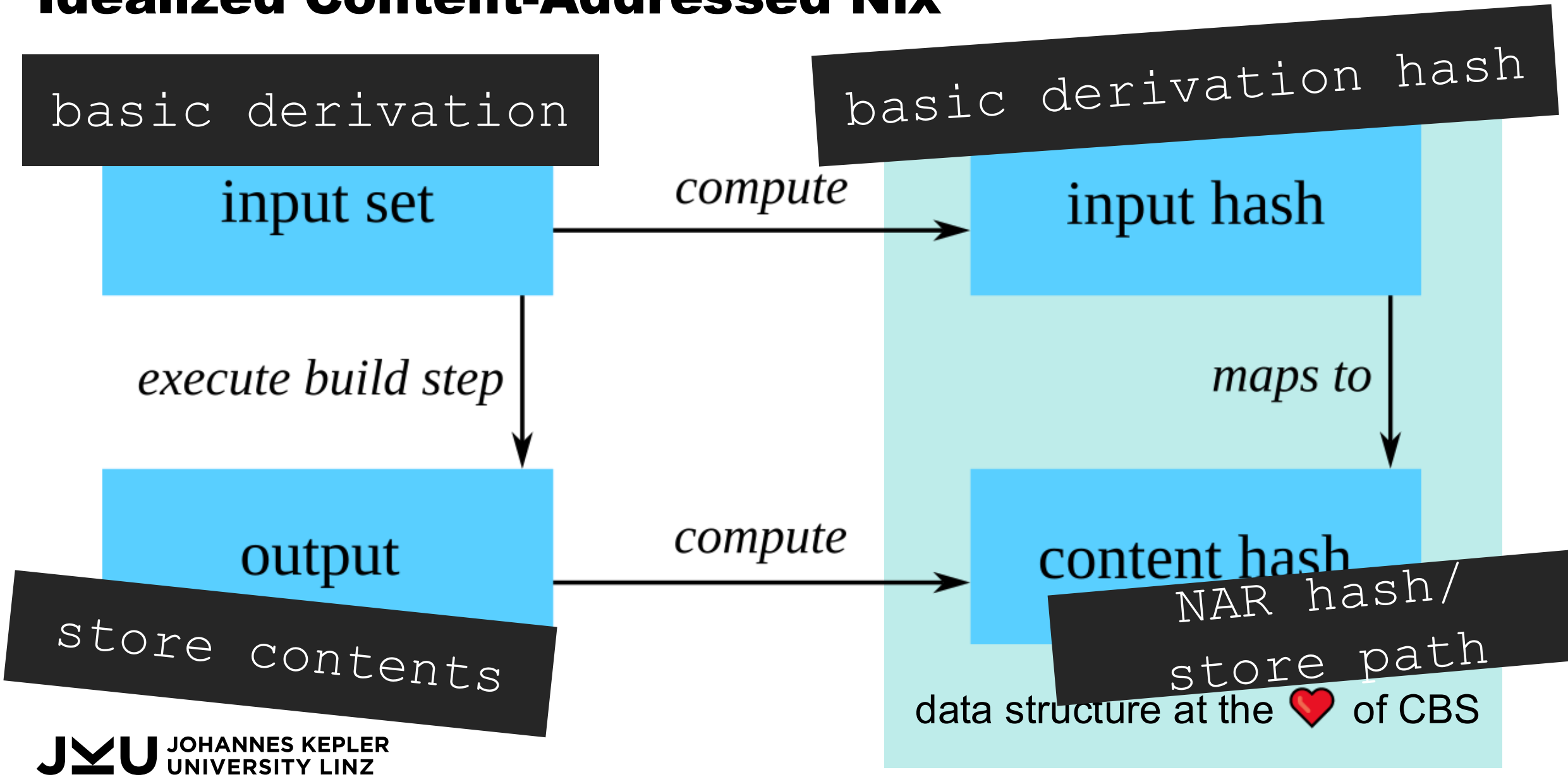
Regular (Input-Addressed) Nix



Dependency Resolution



Idealized Content-Addressed Nix



Example Derivation

simple.c

```
void main() {
    puts("Hello NixCon 2024!");
}
```

simple.nix

```
derivation {
  ...
  src = ./simple.c
  ...
}
```

Example Derivation

simple_builder.sh

```
export PATH="$coreutils/bin:$gcc/bin"
mkdir -p $out/bin
gcc -o $out/bin/simple $src
```

simple.nix

```
derivation {
  ...
  builder = "${pkgs.bash}/bin/bash";
  args = [ ./simple_builder.sh ];
  ...
}
```

Example Derivation

simple.nix

```
let
  pkgs = import <nixpkgs> { };
in
derivation {
  name = "simple";
  builder = "${pkgs.bash}/bin/bash";
  args = [ ./simple_builder.sh ];
  inherit (pkgs) gcc coreutils;
  src = ./simple.c;
  system = builtins.currentSystem;
}
```

input hash ∞ addressing

- Nix has a global namespace of store paths
- This has pros and cons

naming is hard

- input addressed derivation → derivation with recursive hashing of input derivations
- content addressed derivation → derivation with content hashing of input derivations

trust is moved out of the store not disappeared

- That's it. No magic here.

Acknowledgements

This work has been carried out within the scope of Digidow, the Christian Doppler Laboratory for Private Digital Authentication in the Physical World and has partially been supported by the LIT Secure and Correct Systems Lab. We gratefully acknowledge financial support by the Austrian Federal Ministry of Labour and Economy, the National Foundation for Research, Technology and Development, the Christian Doppler Research Association, 3 Banken IT GmbH, ekey biometric systems GmbH, Kepler Universitätsklinikum GmbH, NXP Semiconductors Austria GmbH & Co KG, Österreichische Staatsdruckerei GmbH, and the State of Upper Austria.

References

The simple derivation example is based on:

https://nixos.org/guides/nix-pills/07-working-derivation

Build systems a la carte:

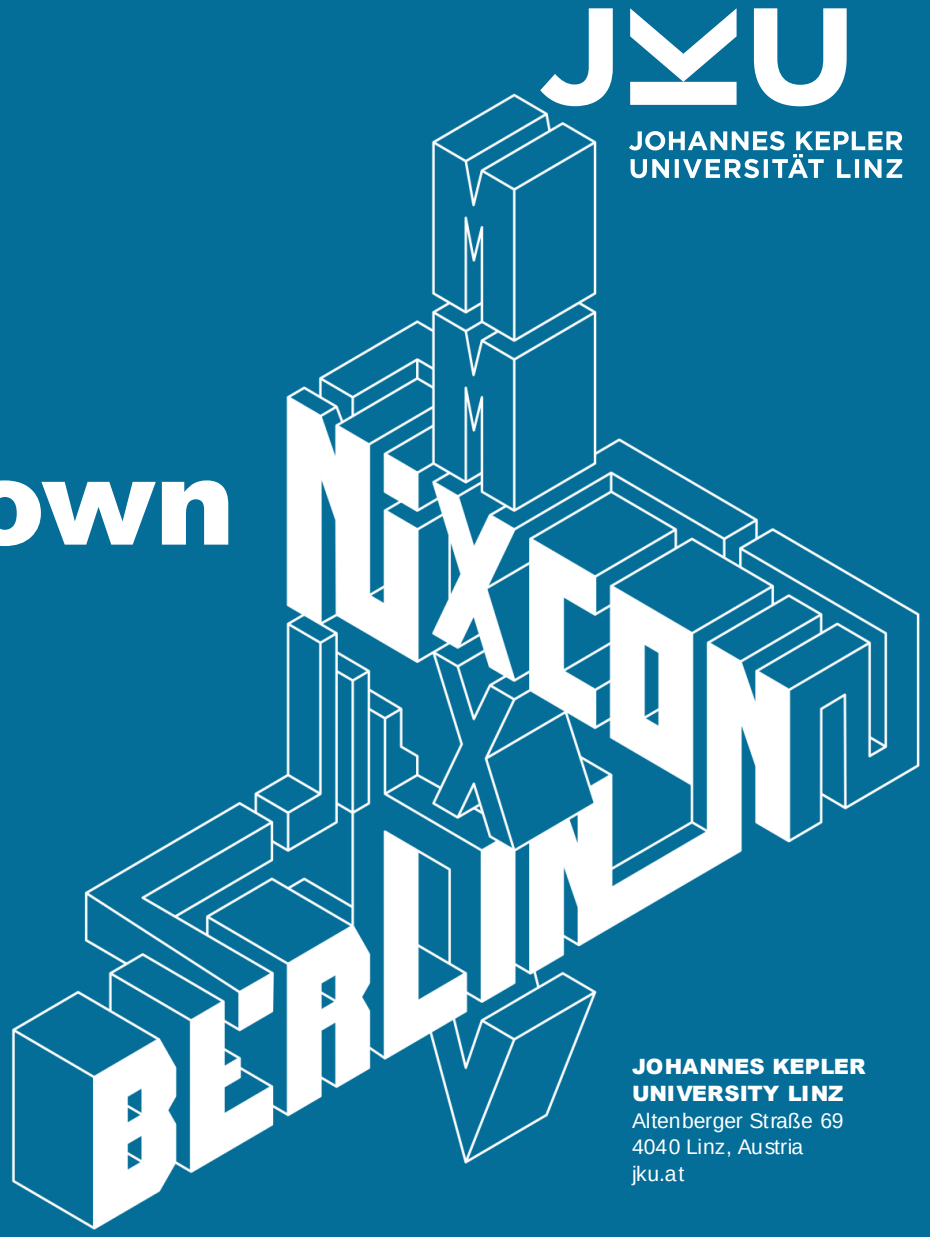
https://doi.org/10.1145/3236774

Extending Cloud Build Systems To Eliminate Transitive Trust (Chapter 3.1 and Note 4):
https://www.digidow.eu/publications/2024-schwaighofer-scored/Schwaighofer_2024_SCORED24_CloudBuildSystemsTrust.pdf

NixCon '24

hashes all the way down

Martin Schwaighofer
Johannes Kepler University Linz
2024-10-25, Berlin (Germany)



JOHANNES KEPLER
UNIVERSITÄT LINZ

JOHANNES KEPLER
UNIVERSITY LINZ
Altenberger Straße 69
4040 Linz, Austria
jku.at