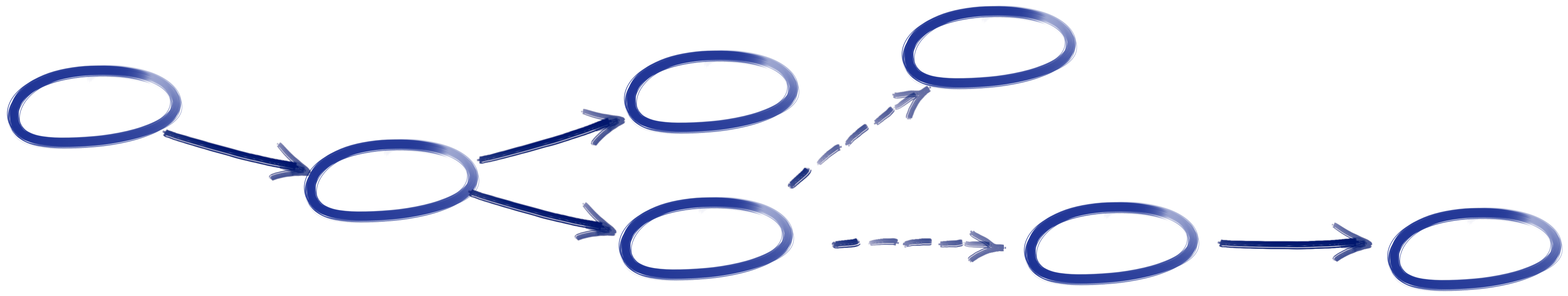


Pipes, Arrows, and the Universe

TECHNISCHE UNIVERSITÄT KAISERSLAUTERN
Albert Schimpf

DECLARATIVE TYPED SPECIFICATIONS FOR CONCURRENT AND COMPOSABLE WORKFLOWS

Tools & Toolboxes

Do one thing well

Compose tools together

What about the interface?

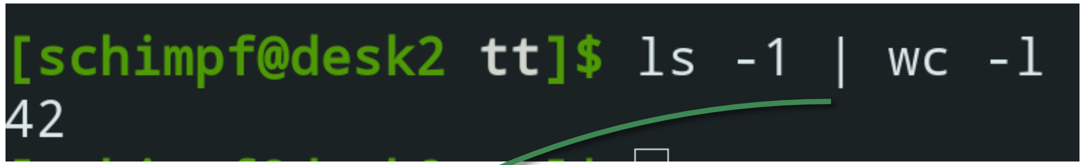
Universal Interface

"Expect the output of every program to become the input of another, as yet unknown, program"

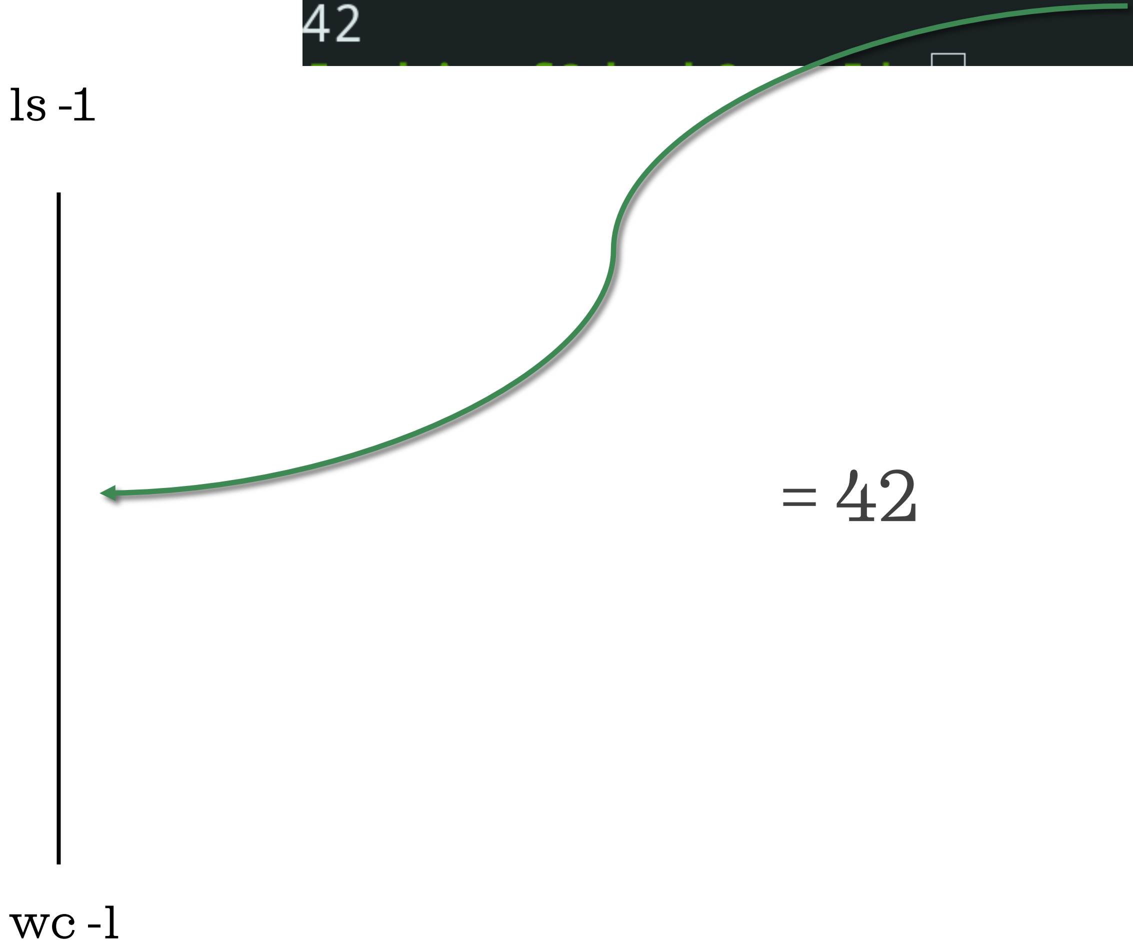
"Write programs that handle text streams, because that is a universal interface"

THE BELL SYSTEM TECHNICAL JOURNAL. M. D. MCILROY, ET. AL. "UNIX TIME-SHARING SYSTEM FORWARD". 1978

Pipe



```
[schimpf@desk2 tt]$ ls -1 | wc -l
42
```



= 42

Pipe

$$\begin{tikzcd}
p_1 \arrow[d, no head] & \text{String} \arrow[d] \\
p_2 & \text{String}
\end{tikzcd}$$

What's nice?

Separation of business logic and control flow

Composition always works

Good for small chains

Easy (implicit) wiring

☹

Shortcomings

Conveniently access output previous steps

Graph-structured control flow

Parsing embedded data structures

Universal Interface II

"Expect the output of every program to become the input of another, as yet unknown, program."

Write programs that handle maps[1], because that is a universal interface.

1: HTTPS://WWW.JSON.ORG/JSON-EN.HTML

Workflows

COMPOSITION

MAP INTERFACE

Workflows

COMPOSITION

MAP INTERFACE

Workflow Systems

Kernmantle ('20), Funflow, Porcupine, iTasks, Luigi, Argo, Oozie, NiFi, Azkaban, Apache Airflow, AWS Data Pipeline, …
(>280)

Map Interface

?

Github Actions

Workflows

COMPOSITION

MAP INTERFACE

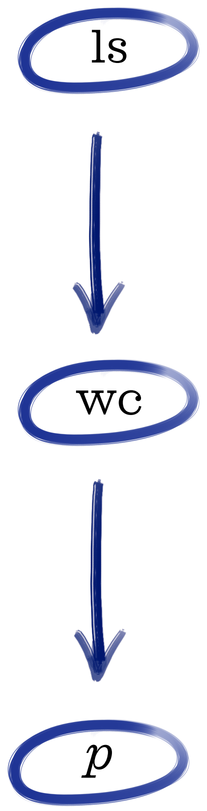
TYPES?

1. Chaining tasks

2. Graph of tasks (DAG)

Map Interface

Input Output

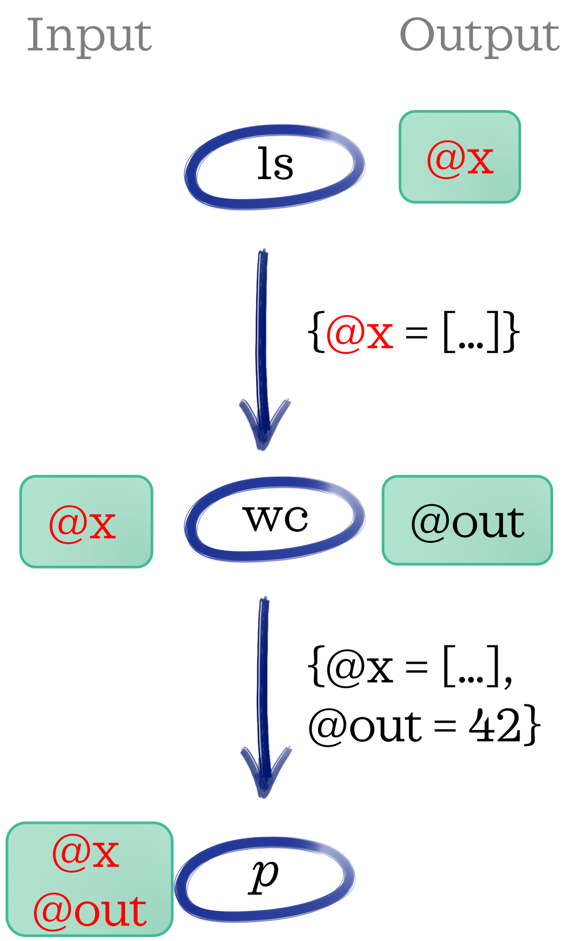


Configuration (input, output) of components

Data flow: Map instead of String

p can access any data in the map that accesses p

Transitive implicit dependencies on input possible



Configuration (input, output) of components

Data flow: Map instead of String

p can access any data in the map that accesses p

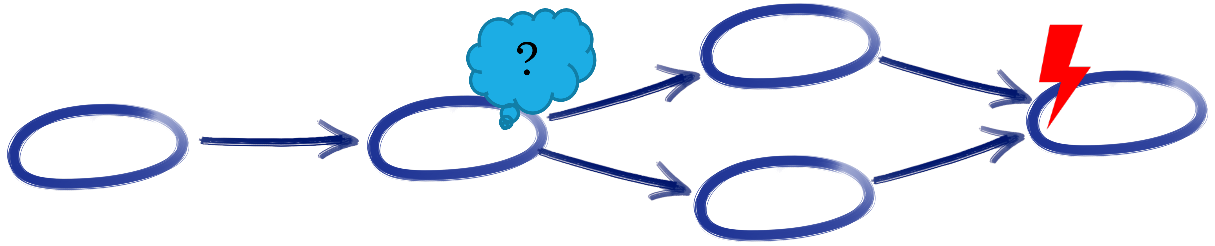
Transitive implicit dependencies on input possible

Graph of Tasks – Semantics

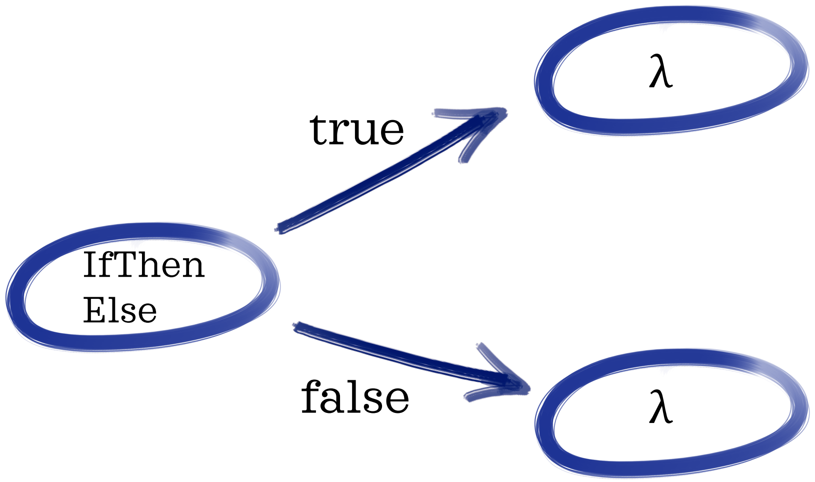
Multiple incoming arrows

Multiple outgoing arrows

Concurrency, isolation?

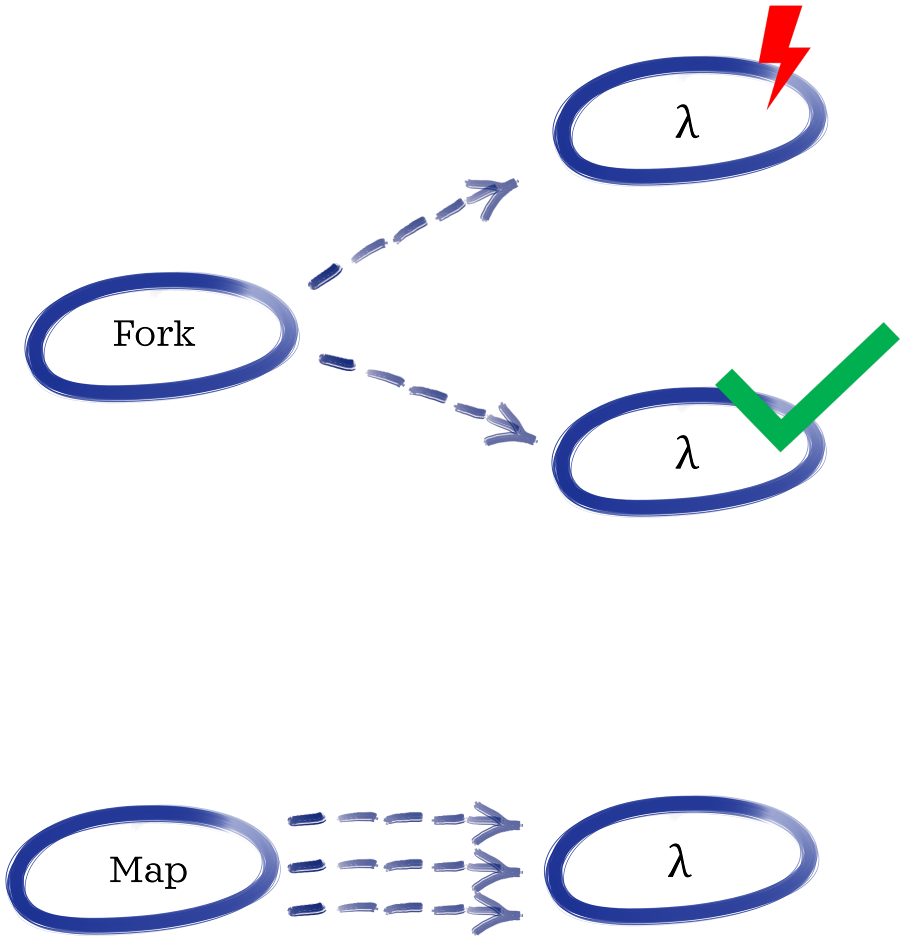


Multiple Outgoing Arrows



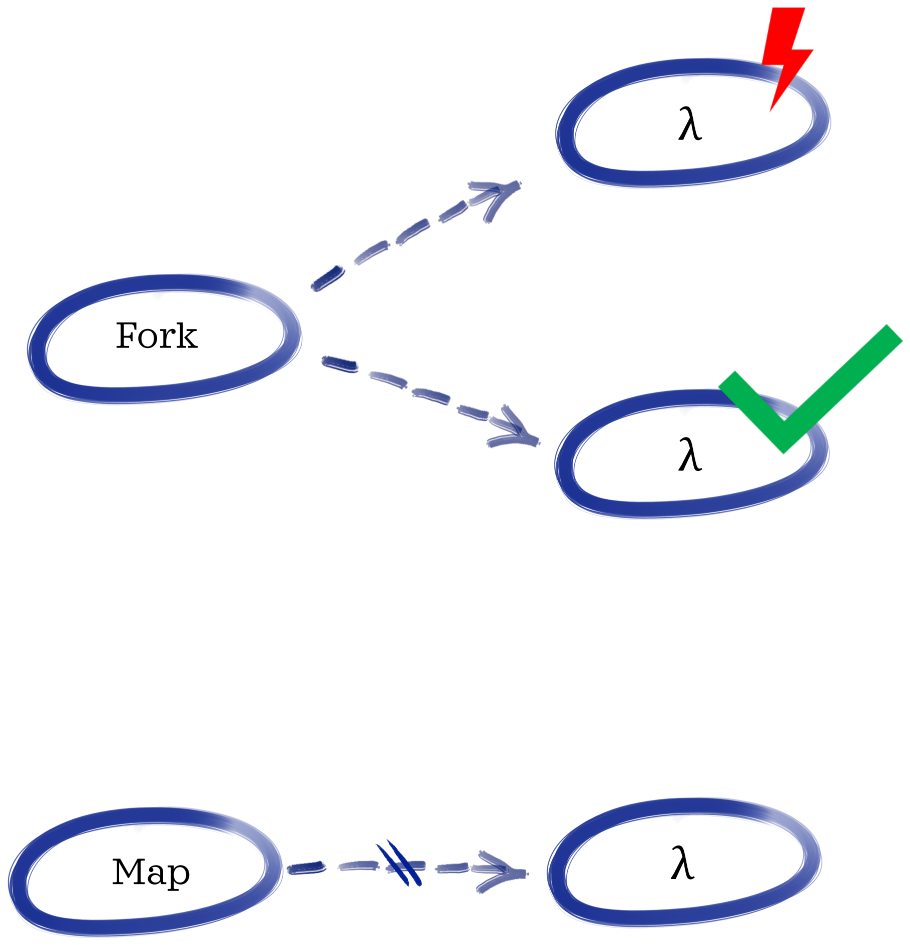
Isolation:

Dispatched Arrows

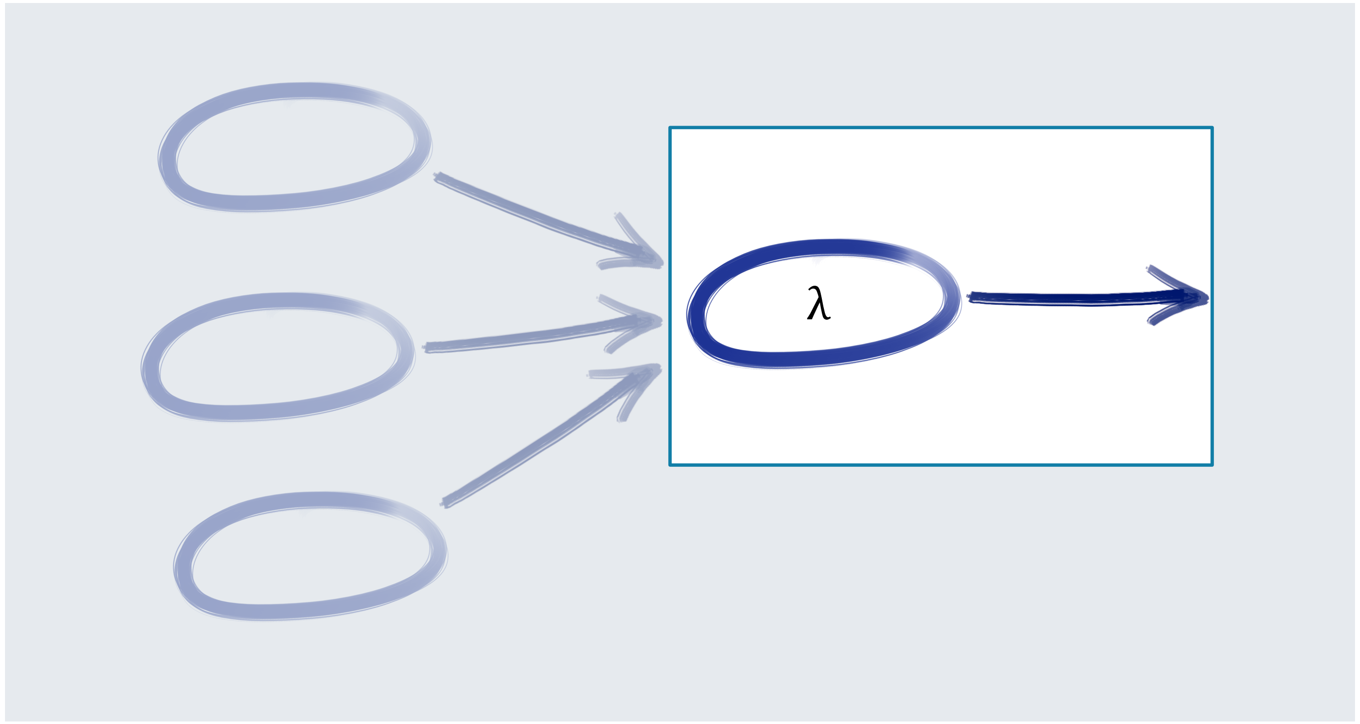


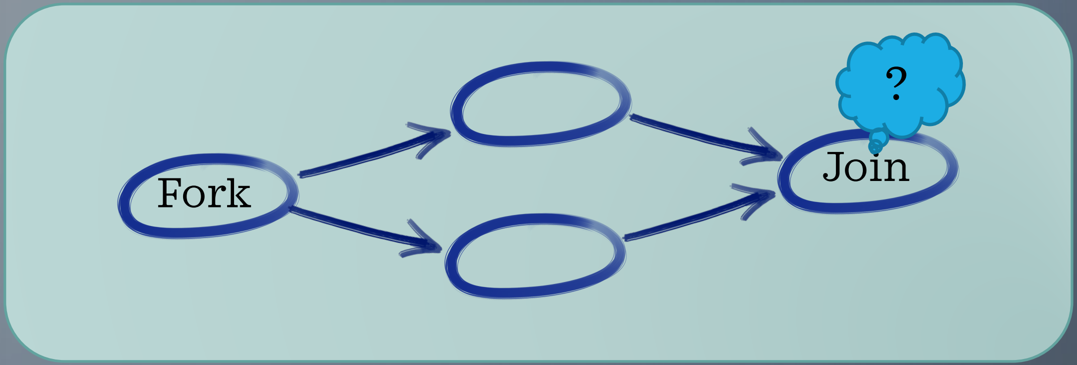
Isolation:

Dispatched Arrows



Multiple Incoming Arrows

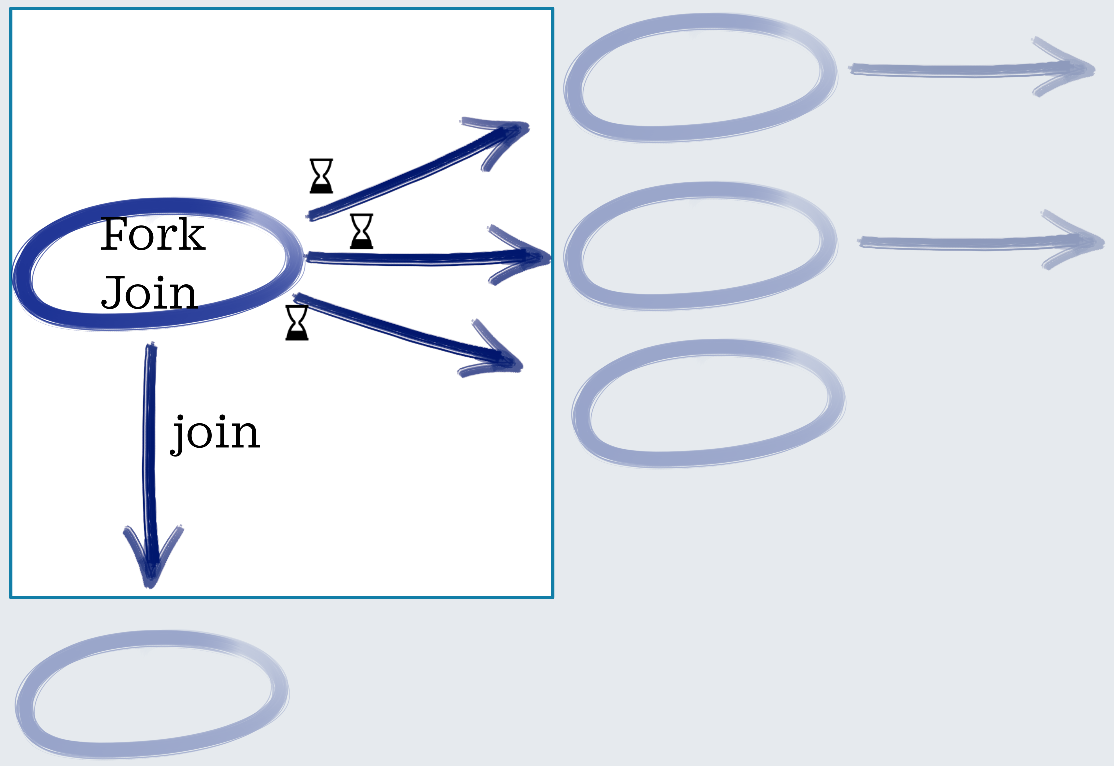




How to Join?

FUTURES!

Futures



Demo

SIMPLE EXAMPLES & PIPE INTERACTION

Types!

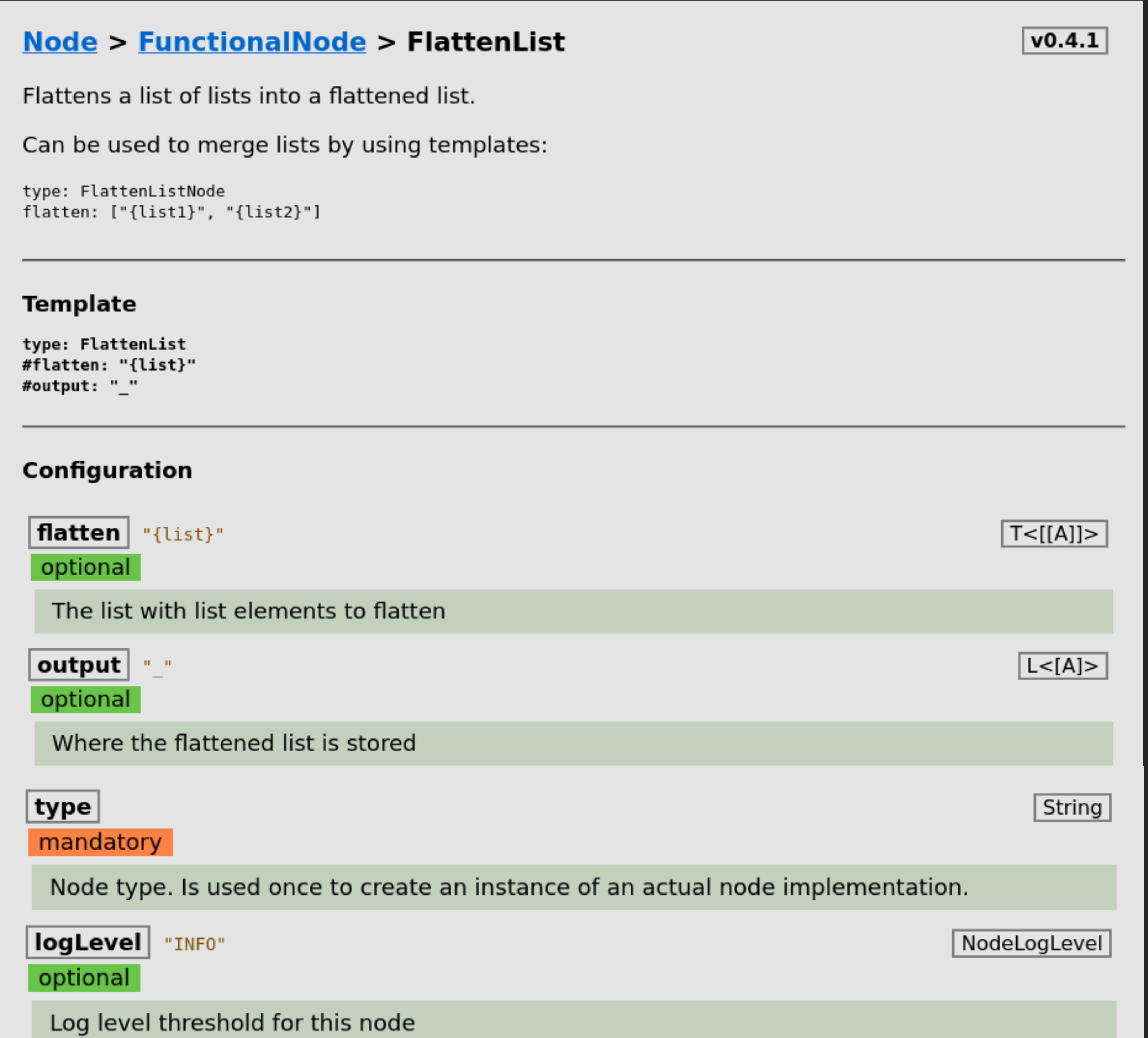
Atomic values

Lists

Maps

Custom records

Polymorphic types



[Node](#) > [FunctionalNode](#) > FlattenList v0.4.1

Flattens a list of lists into a flattened list.

Can be used to merge lists by using templates:

```
type: FlattenListNode
flatten: ["{list1}", "{list2}"]
```

Template

```
type: FlattenList
#flatten: "{list}"
#output: "_"
```

Configuration

flatten "{list}" T<[[A]]>
optional
The list with list elements to flatten

output "_" L<[A]>
optional
Where the flattened list is stored

type String
mandatory
Node type. Is used once to create an instance of an actual node implementation.

logLevel "INFO" NodeLogLevel
optional
Log level threshold for this node

Generated Documentation

Design & Demo

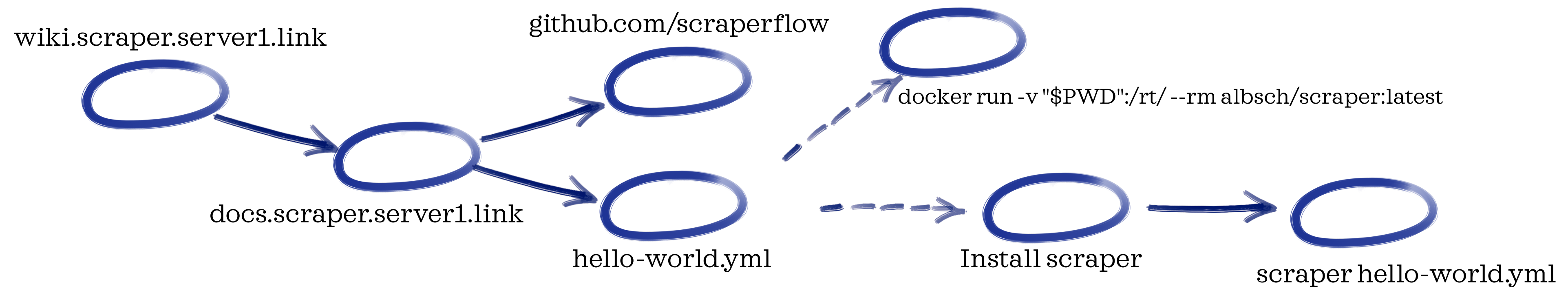
KEYWORD MONITORING

Summary

The universal string interface is not the only truth

Map interface allows for static type checking and more

Try it out yourself!


wiki.scraper.server1.link
github.com/scraperflow
docker run -v "$PWD":/rt/ --rm albsch/scraper:latest
docs.scraper.server1.link
hello-world.yml
Install scraper
scraper hello-world.yml

TECHNISCHE UNIVERSITÄT KAISERSLAUTERN
Albert Schimpf

Pipes, Arrows, and the Universe

DECLARATIVE TYPED SPECIFICATIONS FOR CONCURRENT AND COMPOSABLE WORKFLOWS