# GenFPL:
## DSL-embeddable functional programming languages

Meinte Boersma (DSL Consultancy)

LangDev CON 2024

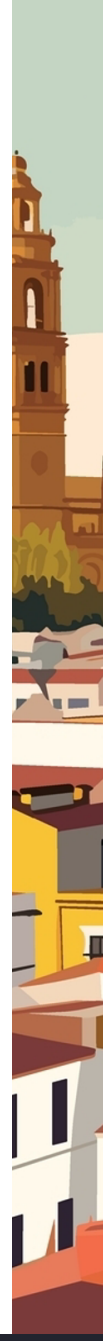
Seville 17-19 October, 2024

https://langdevcon.org

# Accessibility

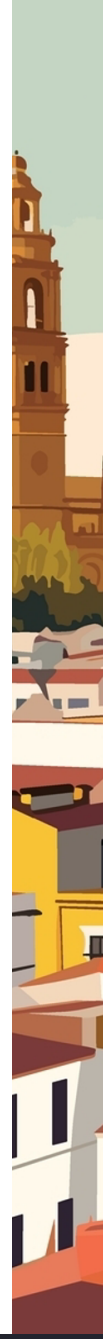
This presentation and its code available at:

https://github.com/dslmeinte/GenFPL-langdev2024

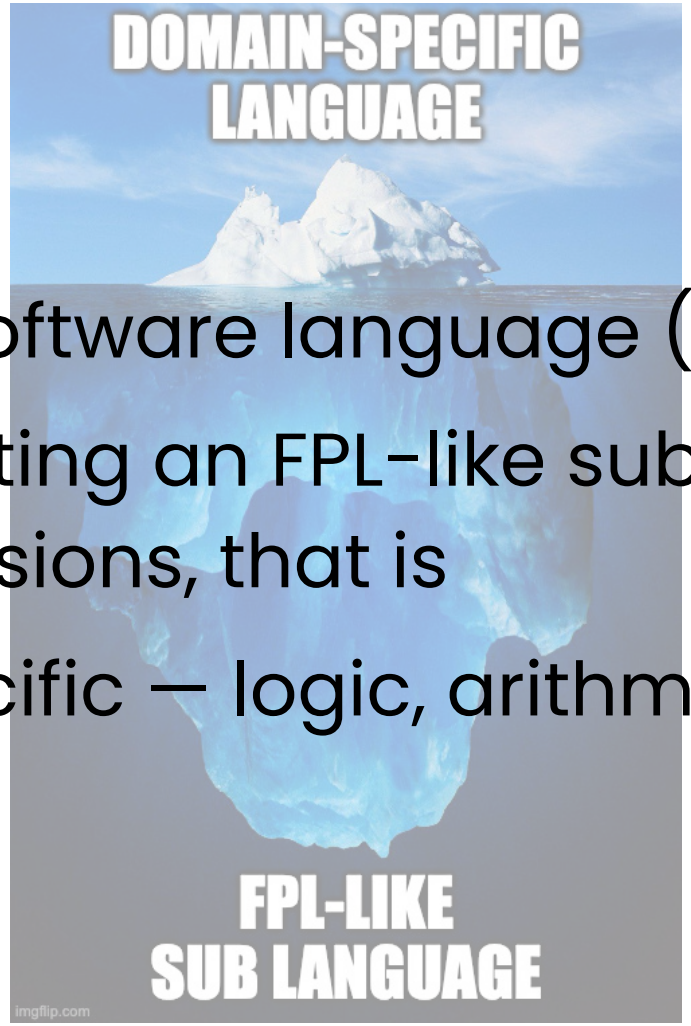
# Caveats

1. GenFPL = “generate FPL”, not “gener{al|ic} FPL”
2. GenFPL is in its ~~infancy~~ fetal stage



# Quick quiz (AKA “market fit research”)

Who among us

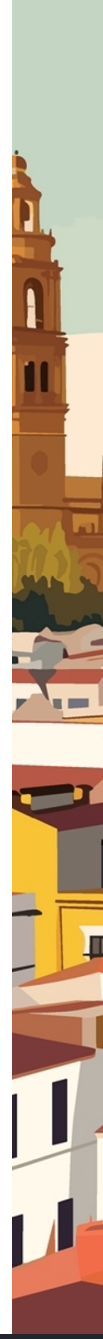



1. Have developed a software language (DSL, etc.), and
2. Ended up implementing an FPL-like sub language for (declarative) expressions, that is
3. Quite domain-**a**specific — logic, arithmetic, etc.

# What is GenFPL?

- JavaScript (Node.js/NPM) tooling…
- …to quickly implement FPL-like sub languages
- Located at: https://github.com/dslmeinte/GenFPL (license=Apache 2.0)

- *Powered by* LANGUAGE INTERFACES ON THE WEB LIONWEB



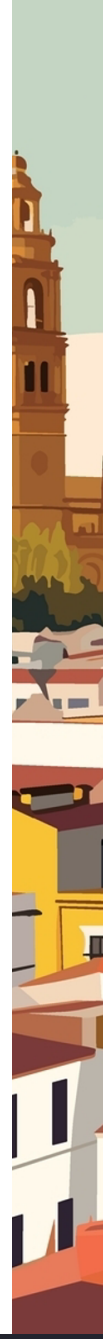
# Why create GenFPL?

- Because there is a need for rapid, industrialized implementation of embeddable sub FPLs — (see quiz).
- But… *KernelF* ?! Not everything happens in MPS.
- Showcase and augment LionWeb.
- To challenge some PL-"traditions".
- To scratch my FPL-itch without needing to have to deal with limitations/idiosyncrasies of an existing FPL.
- For fun!
  - …this talk…

*Powered by*

# Contents (not in order)

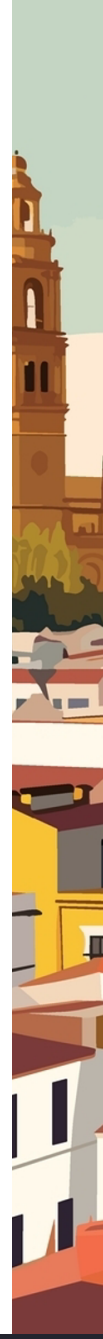
1. Demo GenFPL
    a. Installation and making a configuration
    b. Implementing and testing an interpreter
    c. Accessing records
2. Some(anti-)patterns for sub FPLs
    a. Typical *areas* and their meta-hierarchy
    b. To `stdlib`, or not to `stdlib`?



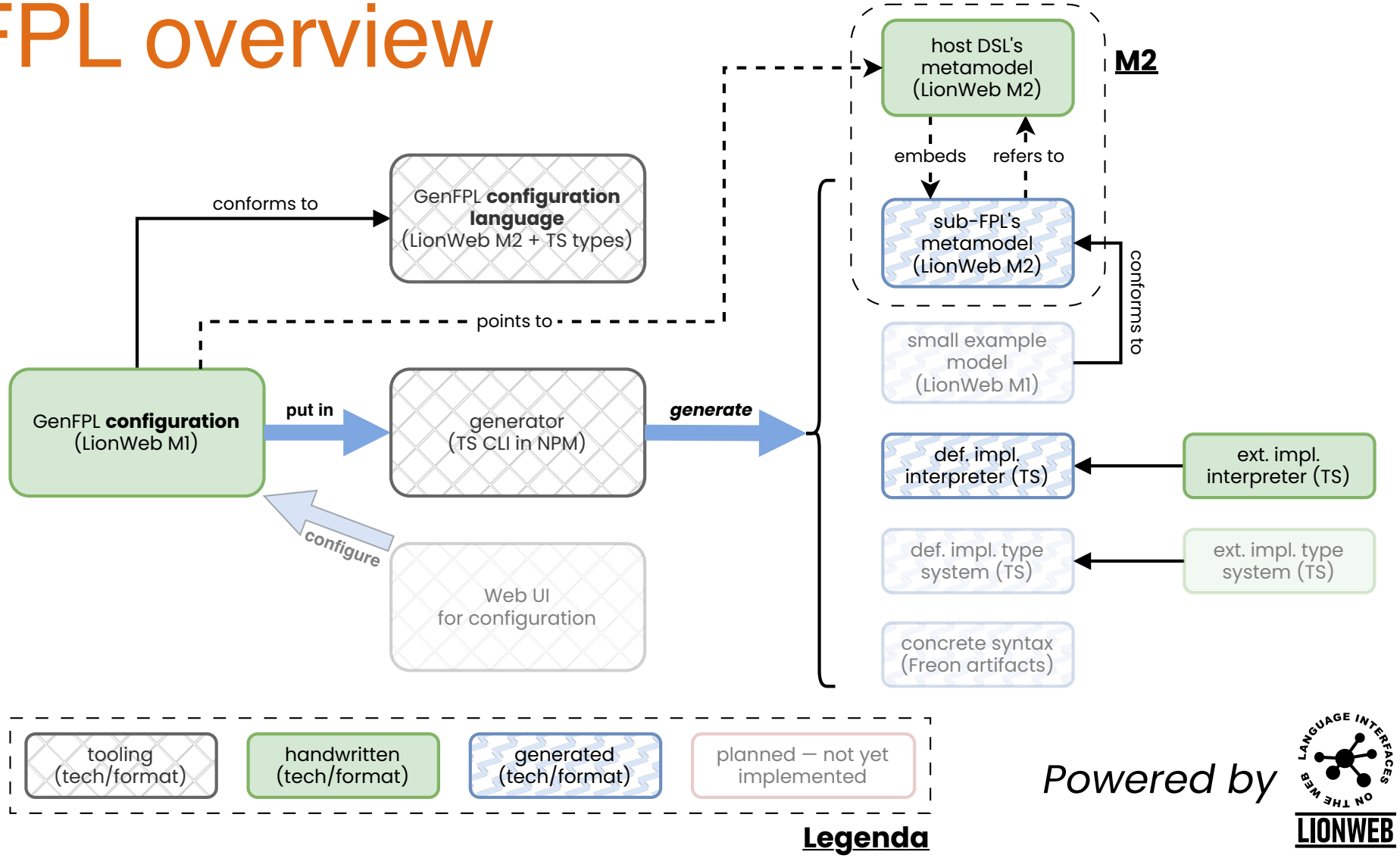
# What is an FPL anyway?

- **Funclarative**[1] expressions language
- Governed by a substitution model,
  so admits to algebraic reasoning
  - Makes it simpler to reason about programs
- Quite simple to correctly implement semantics
  and type system

1) term coined by: Markus Völter



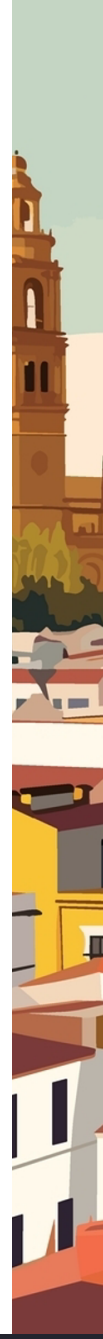
# GenFPL overview



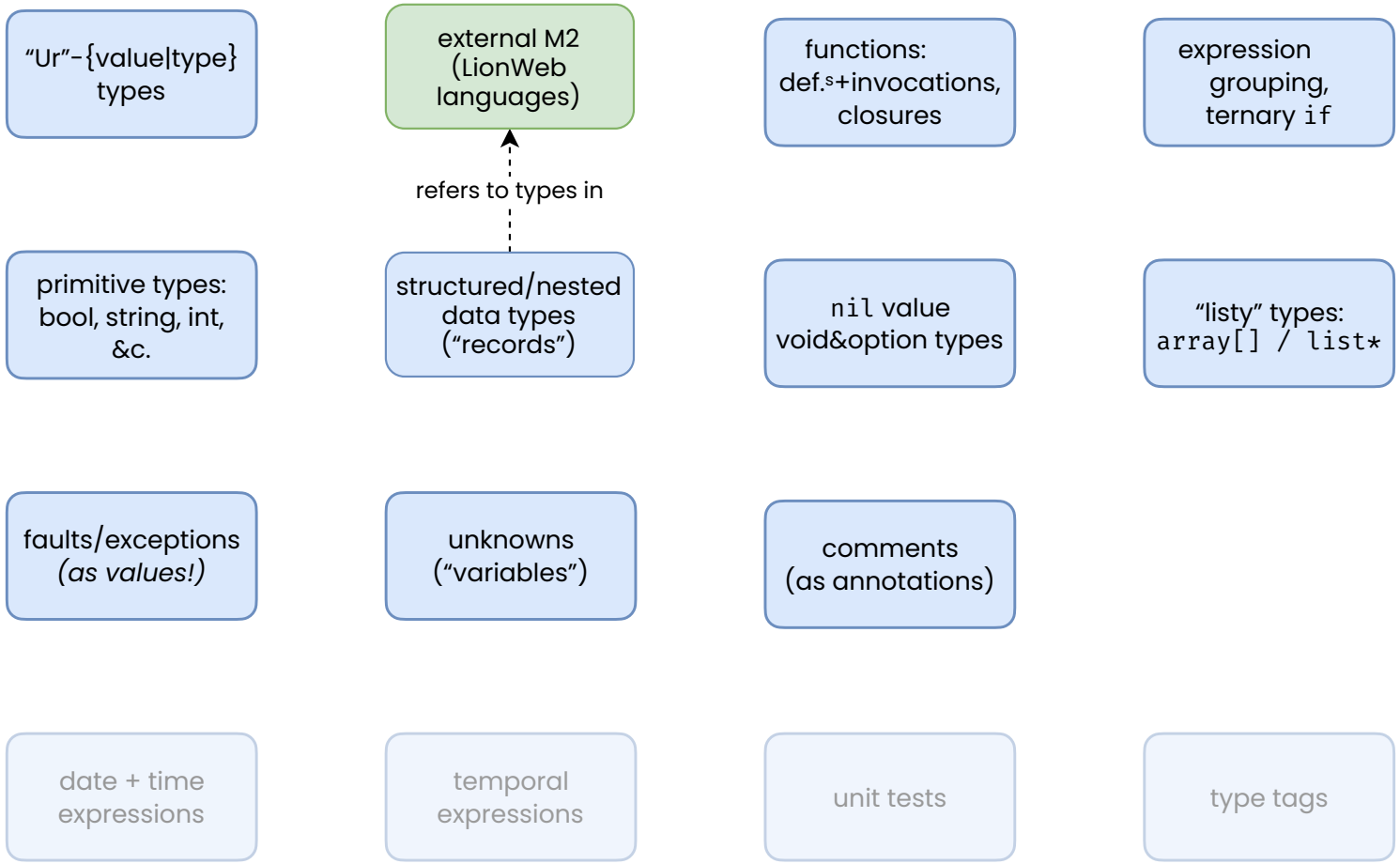

*Powered by* LIONWEB

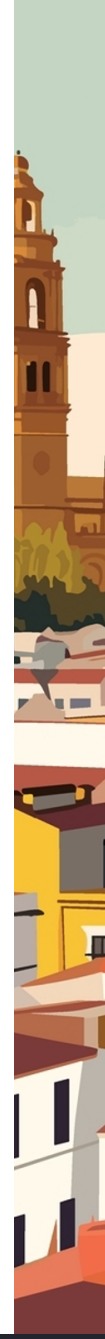
# Design

- Generate parts of sub FPL from a configuration:
  - Metamodel (M2)
  - Extensible default implementation of interpreter
  - (Future work: type system, Freon integration, etc.)
- Granularity: *areas* ~ modules

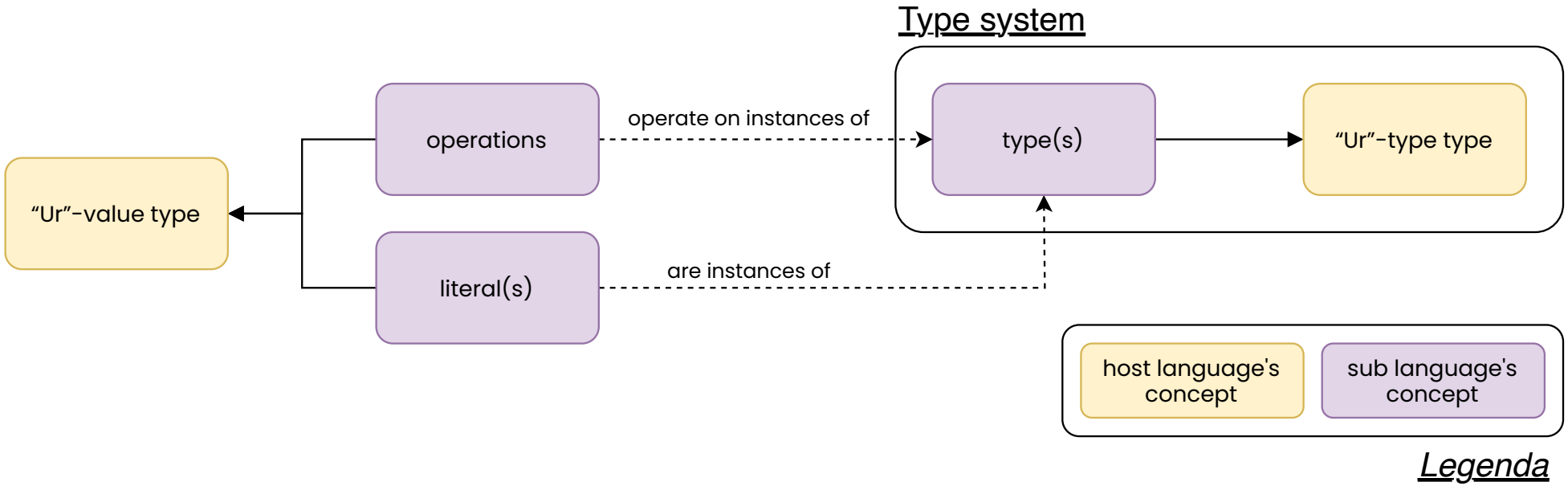


# Areas of sub-FPLs



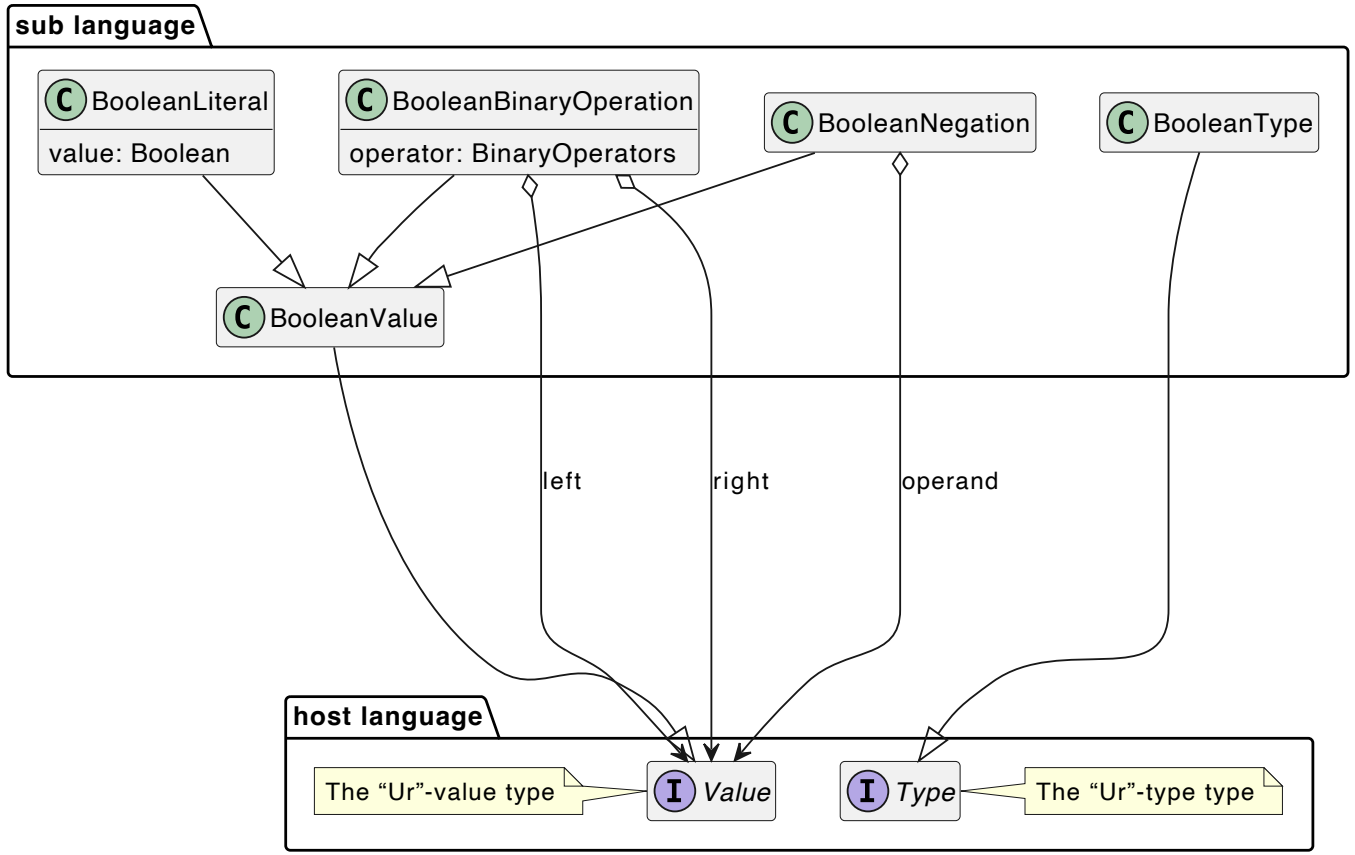
# Meta-hierarchy of an area



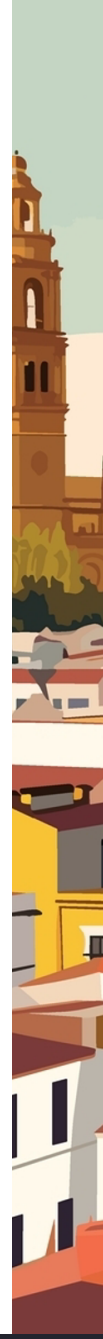

"Ur"-{type|value} types are specified in the GenFPL configuration

# Meta-hierarchy of an area *(cont.d)*

Example: **boolean** area

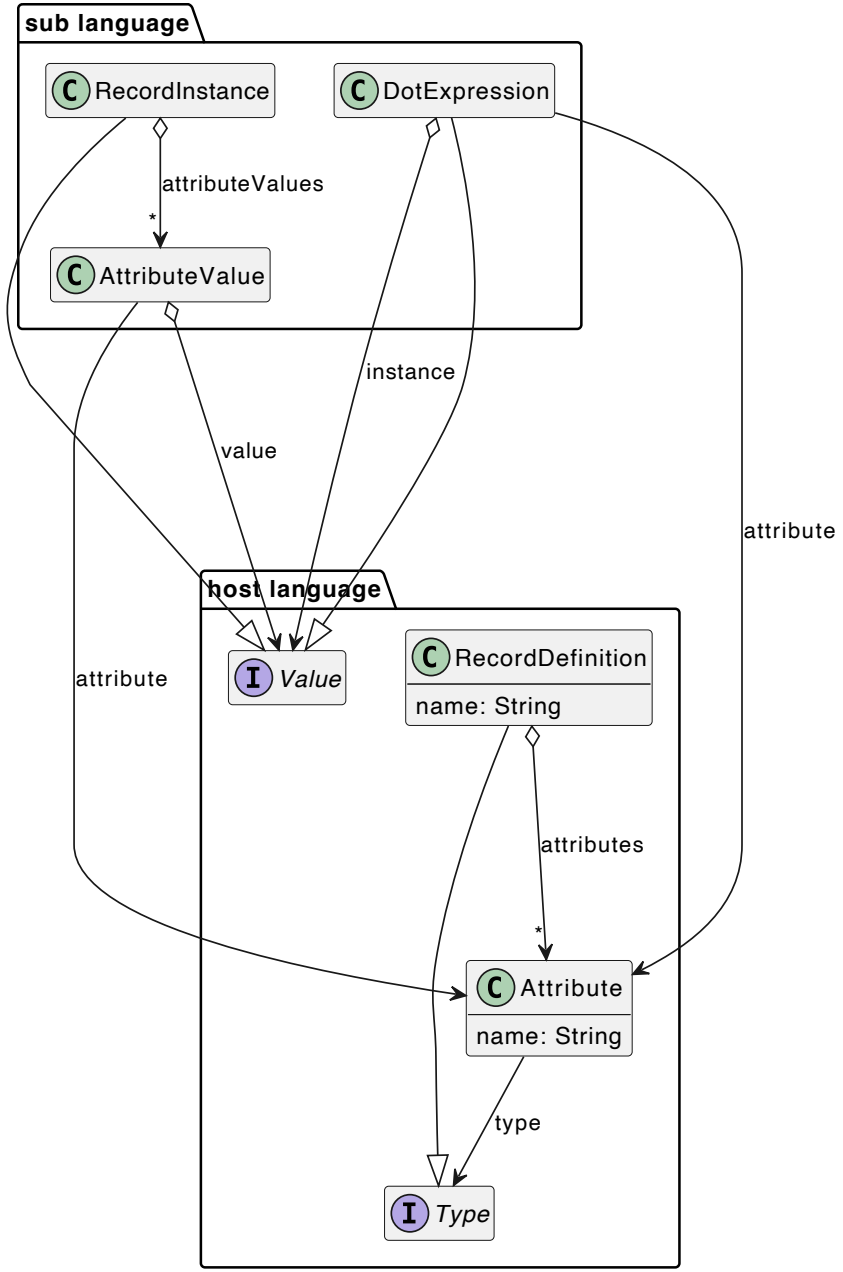


# Demo (1/2)

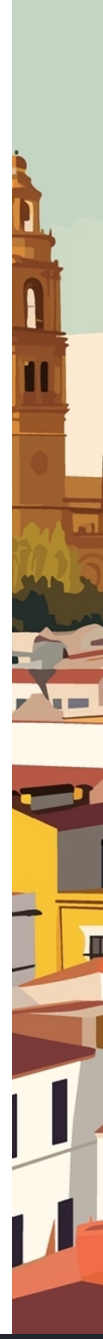


# Accessing records

- Observation: host language often has concepts for (nested) data structures — e.g. “records”.
- Want to be able to access attribute values on instances of those.
- Solution: configuration points to concepts in the host language, and generate appropriate concepts.

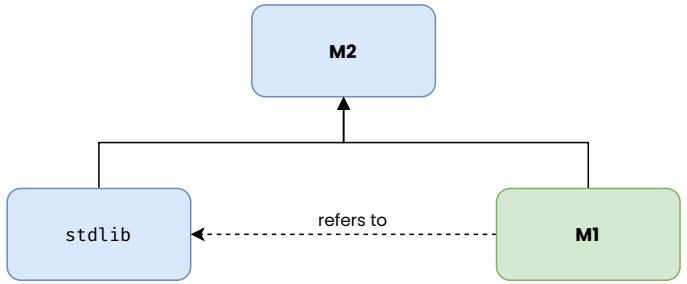


# Demo (2/2)



# To `stdlib`, or not to `stdlib`?

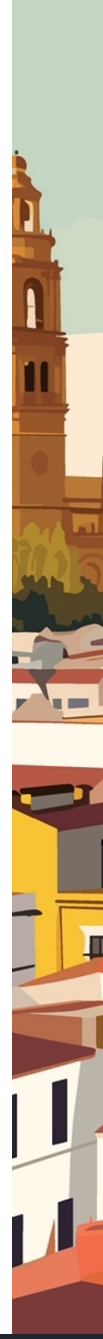
- A `stdlib` adds features to a language without enlarging the M2. Idea:




- Cost: need generic concepts to be able to define the `stdlib` *including type system* → an “inner metamodel”

# To `stdlib`, or not to `stdlib`? *(cont.d)*

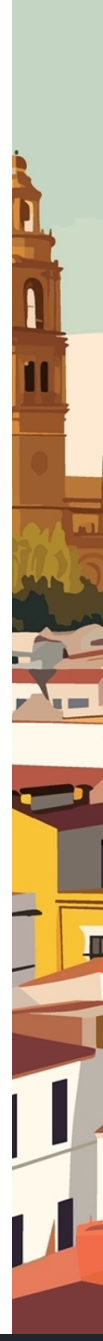
- Pros:
  - Fewer concepts to deal with (eventually)
    - More malleable
  - Better abstractions and generalizations
- Cons:
  - No syntactic difference: “everything’s an *<X>*” ⇒ worse discoverability
  - More complex type system



# To `stdlib`, or not to `stdlib`? *(cont.[d])*

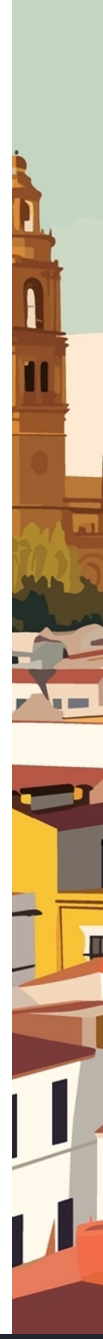
In the context of GenFPL:

- Generation is cheap
- ⇒ Pros of `stdlib` disappear, while cons would still be “hit”
- ⇒ Design choice: no `stdlib`



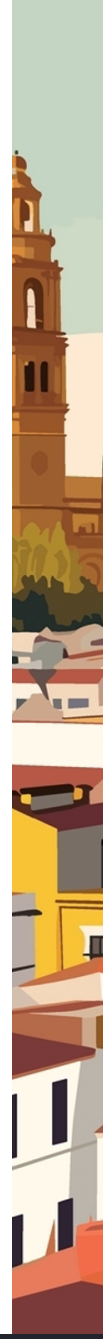
# Conclusions

- Interesting to do this gener{atively|ically}
- Generating a language means keeps complexity of it down
- Good input for LionWeb
- Plenty of work to do

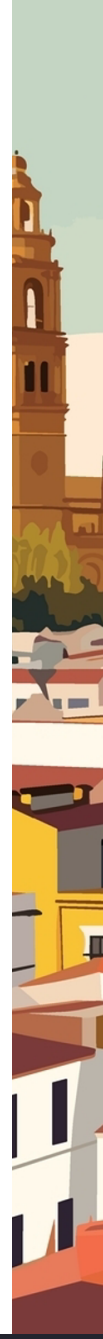


# Future work — plans / ideas

- Integrate with Freon for a concrete syntax
- More areas
- A CLI tool
- Type system
- Nice UI for configuration
- Generate a generator



# Questions?



# Thank you!

And generate your sub-FPL *today!*

LangDev CON 2024

**Seville 17-19 October, 2024**

https://langdevcon.org