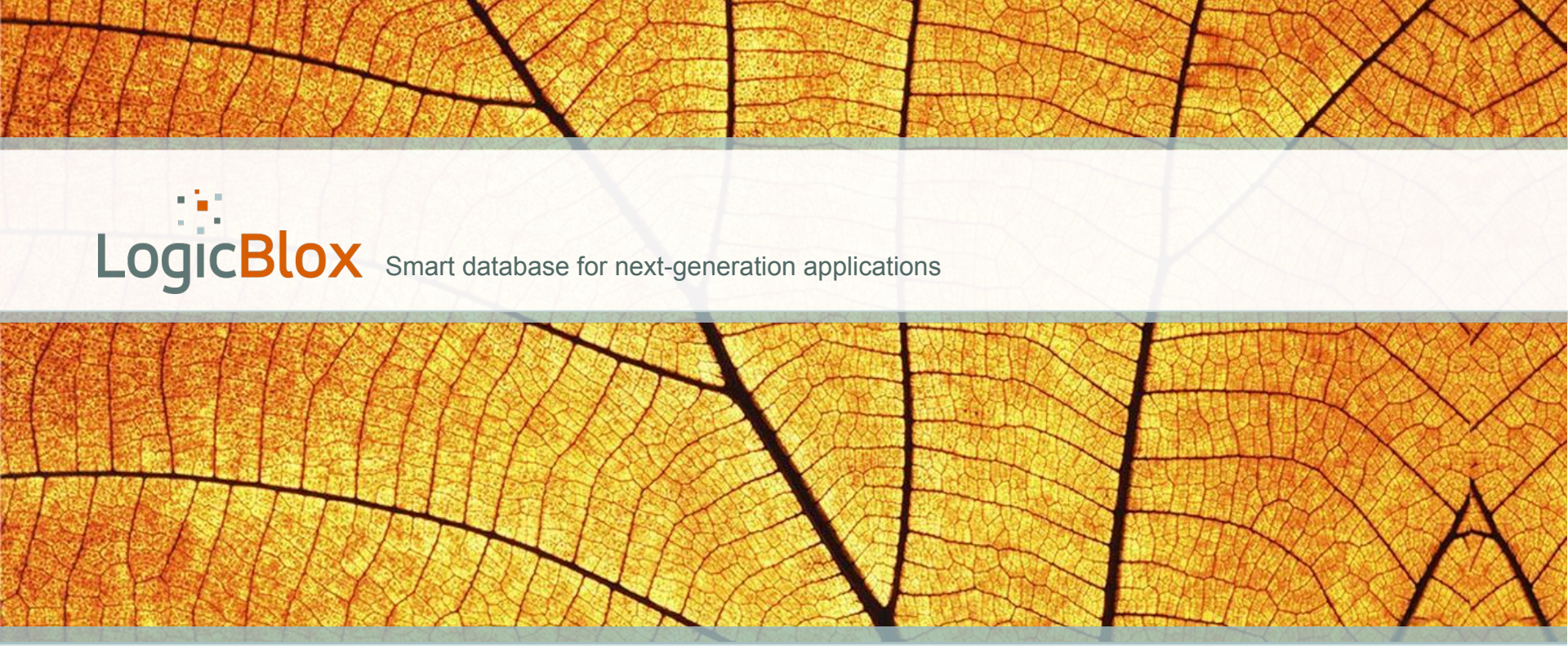


LogicBlox Smart database for next-generation applications

# LESSON 1 : WHAT, WHY, & HELLO WORLD!

## LOGICBLOX TRAINING

# WHAT'S IN A PLATFORM?

- databases
  - out-of-core data processing
  - ACID
  - transactional or analytical workload

- application servers
  - expressive (enough) language for business logic
  - language runtime
  - web container
  - security, distribution, queueing, persistence, etc.

# LOGICBLOX : A UNIFYING PLATFORM

- unifying database and application server
- unifying transactional and analytical

# WHY?

- unifying database and application server
  - productivity
    - mapping between data models tedious & valueless
    - transaction semantics across layers tricky
  - performance
    - transferring data between layers costly
    - optimization across layers manual and ineffective

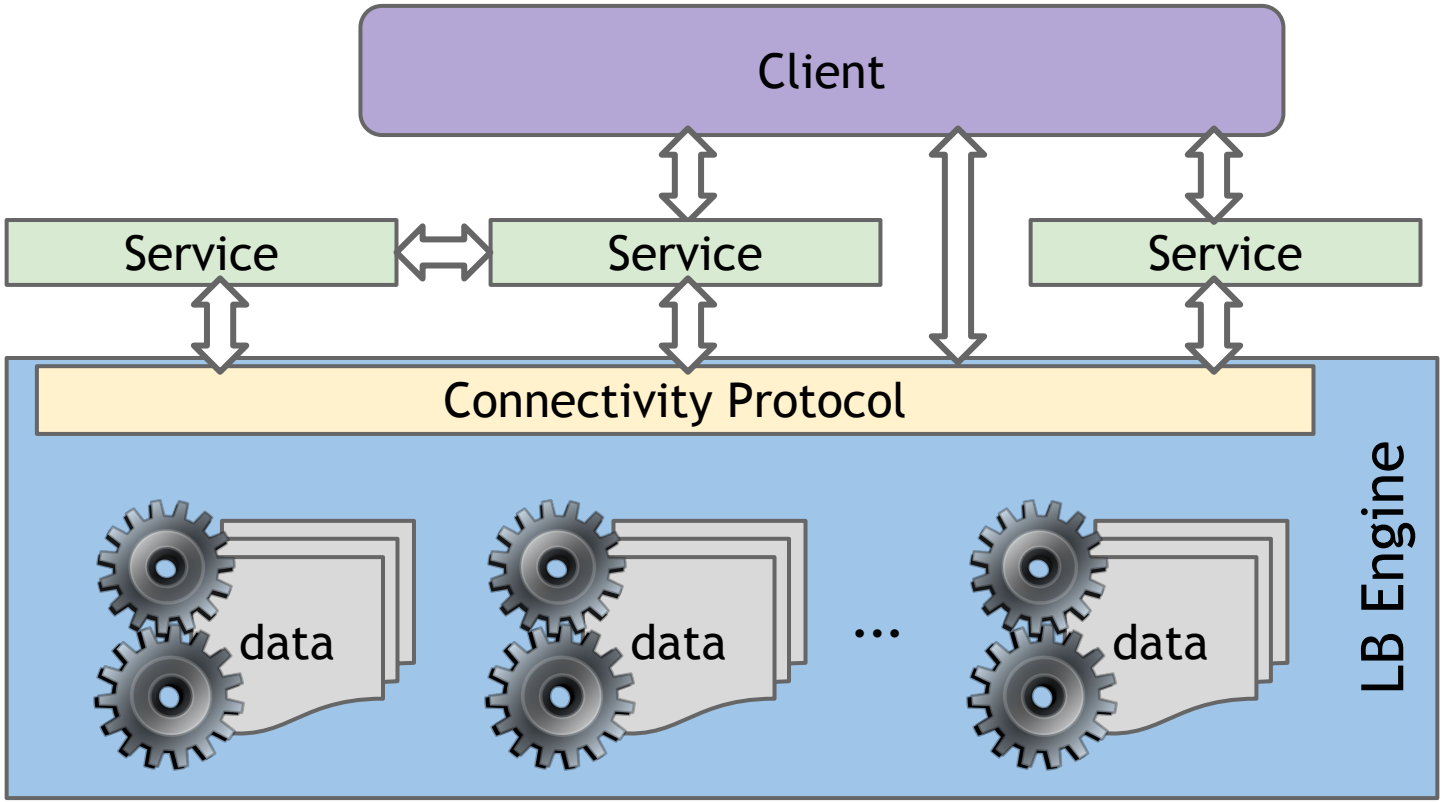
- unifying transactional and analytical
  - productivity
    - mapping between data models tedious
  - performance
    - real-time answers require real-time data

# HOW?

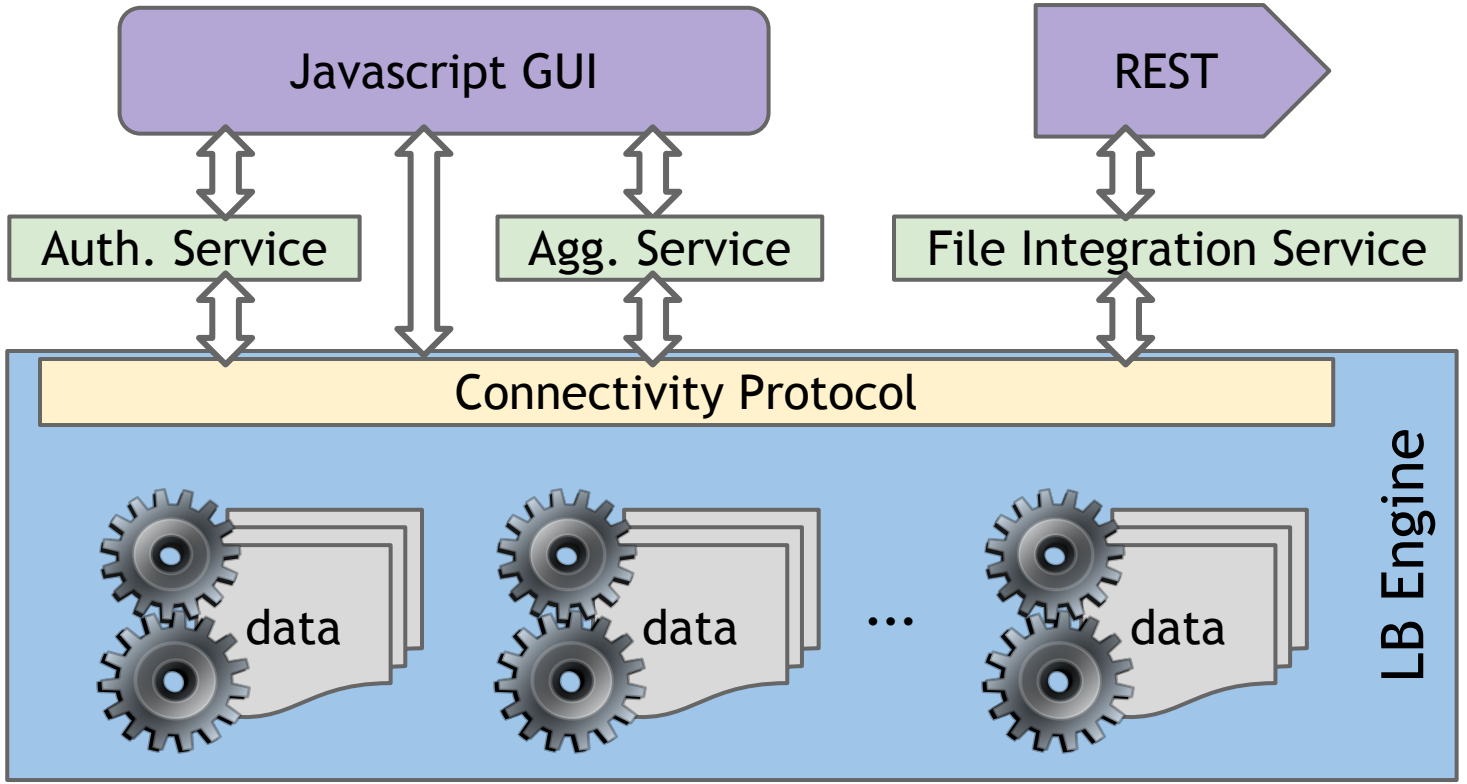
- the language
  - declarative programming model
    - performance, productivity
    - solving the hard problems of programming
  - expressive (enough) for applications and analysis
    - if/then/else, iteration, event, exceptions
    - natural for multi-dimensional analysis

- the evaluation engine
  - mixed transactional and analytical workloads
    - concurrency
    - fast recomputation of analytical results after updates

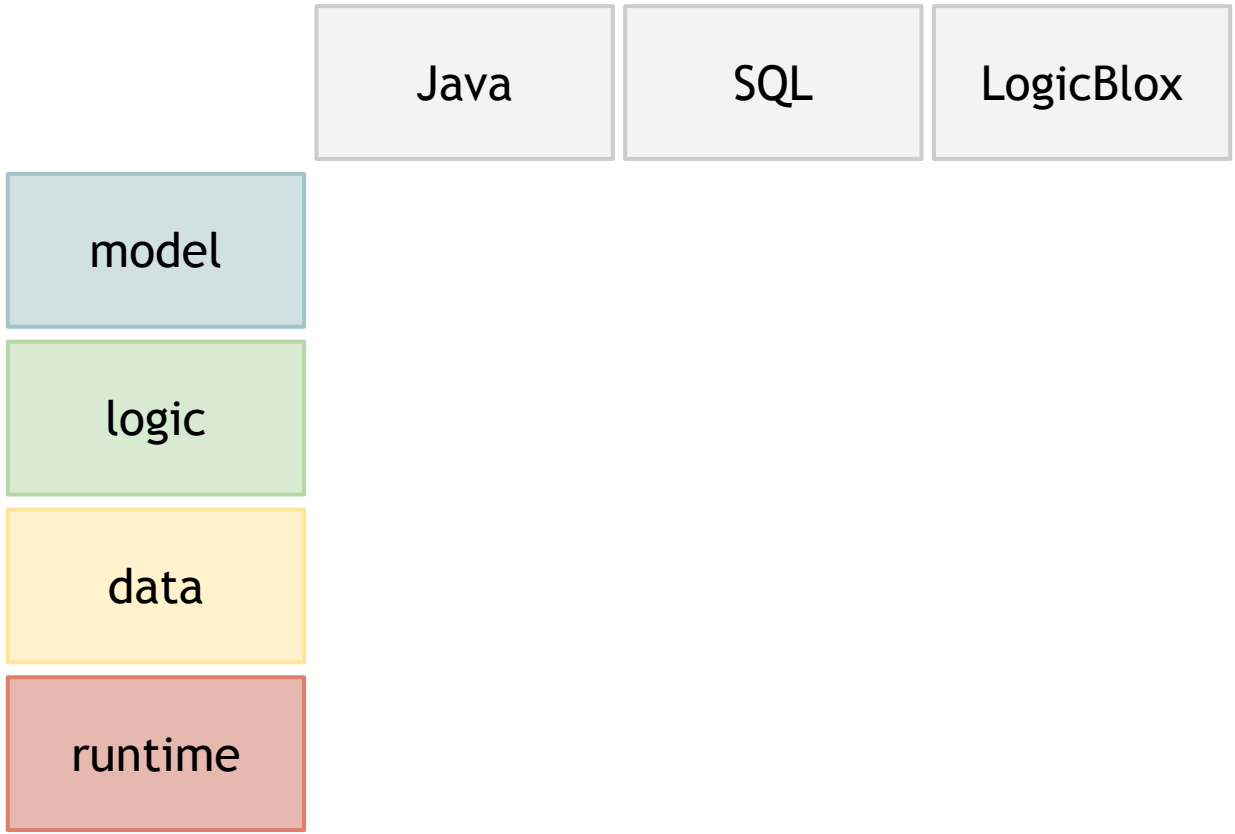
# TYPICAL APPLICATION ARCHITECTURE



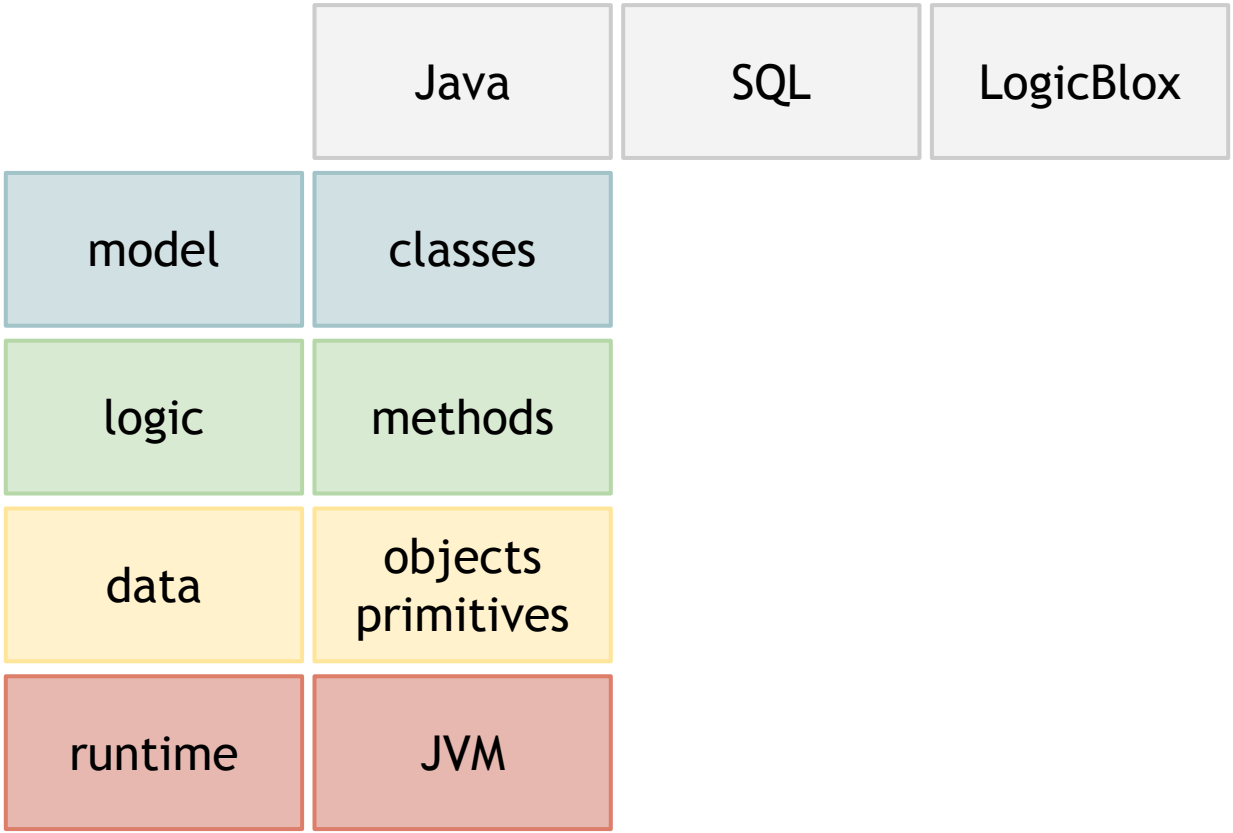
# EXAMPLE ARCHITECTURE



# LOGICBLOX : AN ANALOGY



# LOGICBLOX : AN ANALOGY



# LOGICBLOX : AN ANALOGY

| | Java | SQL | LogicBlox |
|---|---|---|---|
| model | classes | tables | |
| logic | methods | queries<br>views | |
| data | objects<br>primitives | relations | |
| runtime | JVM | database<br>runtime | |

# LOGICBLOX : AN ANALOGY

| | Java | SQL | LogicBlox |
|---|---|---|---|
| model | classes | tables | predicates |
| logic | methods | queries<br>views | rules<br>queries |
| data | objects<br>primitives | relations | sets of<br>tuples |
| runtime | JVM | database<br>runtime | LogicBlox<br>runtime |

# THIS LESSON

- model
  - predicate declarations and primitive types
- logic
  - rules: basic, conjunction, disjunction, negation
- data
  - specified using rules
- interacting with the runtime
  - lb command

# HELLO WORLD : MODEL

- model : **predicate declaration**

```
greetings(content) -> string(content).
```

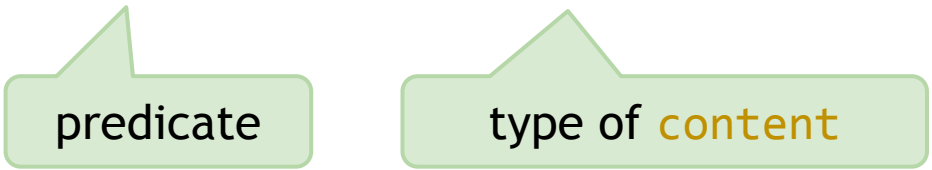
predicate

type of `content`

# HELLO WORLD : MODEL

- model : **predicate declaration**

```
greetings(content) -> string(content).
```




[LogicBlox 4 Reference Manual, Chapter 7: Predicates]

# HELLO WORLD : MODEL

- model : **predicate declaration**

```
greetings(content) -> string(content).
```

predicate

type of `content`

- SQL analogy

```
CREATE TABLE greetings (
  content VARCHAR(255)
)
```

# HELLO WORLD : MODEL

- model : **predicate declaration**

```
greetings(content) -> string(content).
```

predicate

type of content

- SQL analogy

```
CREATE TABLE greetings (
  content VARCHAR(255) NOT NULL
  UNIQUE(content)
)
```

# HELLO WORLD : DATA

- model : **predicate declaration**

```
greetings(content) -> string(content).
```

- data

```
greetings("Hello World").
```

# RUN HELLO WORLD

- create a **workspace**

```
% lb create helloworld
```

# RUN HELLO WORLD

- create a **workspace**

```
% lb create helloworld
```

- add logic

helloworld.logic

```
greetings(content) -> string(content).
greetings("Hello World").
```

```
% lb addblock helloworld -f helloworld.logic
```

# RUN HELLO WORLD

- create a **workspace**

```
% lb create helloworld
```

- add logic

helloworld.logic

```
greetings(content) -> string(content).
greetings("Hello World").
```

```
% lb addblock helloworld -f helloworld.logic
```

- print

```
% lb print helloworld greetings
"Hello World"
```

# PERSONALIZED HELLO WORLD : MODEL

- model

```
greetings(language, content) -> string(language), string(content).
```

# PERSONALIZED HELLO WORLD : MODEL

- model

```
greetings(language, content) -> string(language), string(content).

person_speaks(name, language) -> string(name), string(language).
```

# PERSONALIZED HELLO WORLD : MODEL

- model

```
greetings(language, content) -> string(language), string(content).

person_speaks(name, language) -> string(name), string(language).

person_hasGreeting(name, greeting) -> string(name), string(greeting).
```

# PERSONALIZED HELLO WORLD : LOGIC

- logic : a rule

```
person_hasGreeting(name, greeting)
  <-  person_speaks(name, language),
      greetings(language, greeting).
```

# PERSONALIZED HELLO WORLD : LOGIC

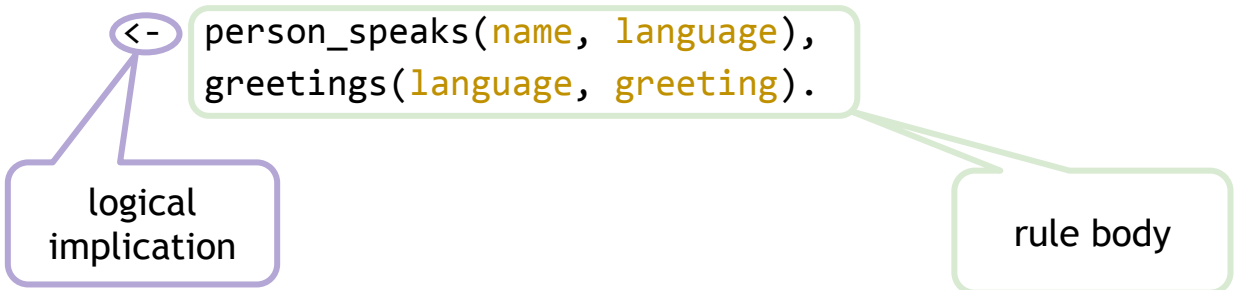
- logic : a rule

```
person_hasGreeting(name, greeting)
  <-  person_speaks(name, language),
      greetings(language, greeting).
```

# PERSONALIZED HELLO WORLD : LOGIC

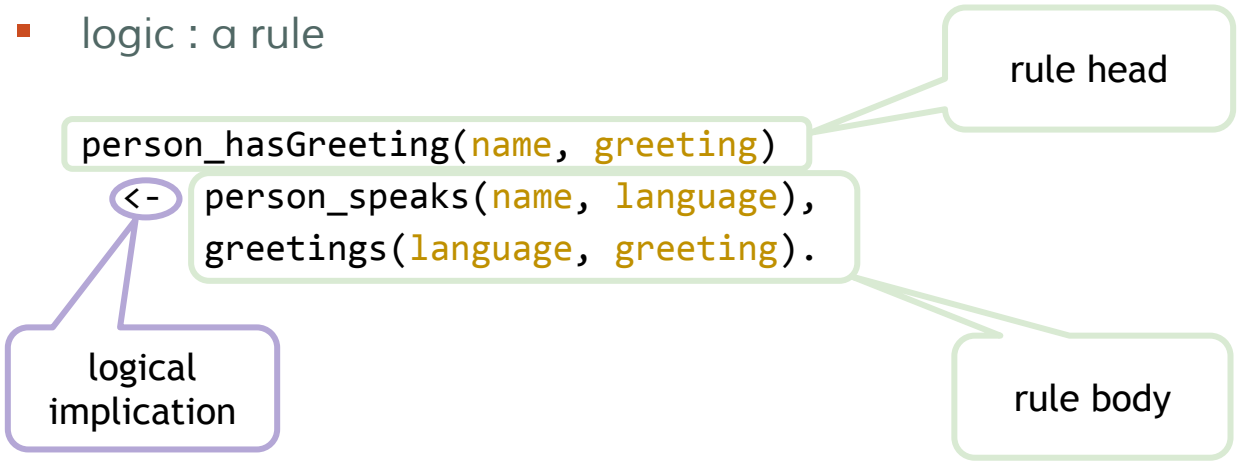
- logic : a rule

```
person_hasGreeting(name, greeting)
  <- person_speaks(name, language),
     greetings(language, greeting).
```



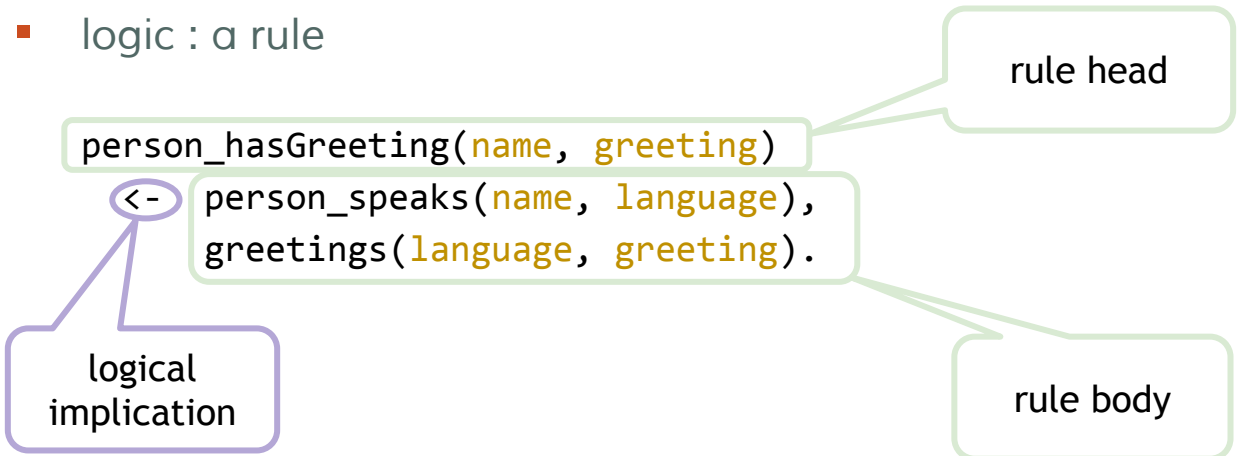
# PERSONALIZED HELLO WORLD : LOGIC

- logic : a rule



# PERSONALIZED HELLO WORLD : LOGIC

- logic : a rule




if rule body is true, then rule head is true

# PERSONALIZED HELLO WORLD : LOGIC

- logic : a rule

```
person_hasGreeting(name, greeting)
  <- person_speaks(name, language),
     greetings(language, greeting).
```

- SQL analogy

```
CREATE VIEW person_hasGreeting AS
SELECT
  person_speaks.name,
  greetings.content AS greeting
WHERE
  person_speaks.language = greetings.language
FROM person_speaks, greetings
```

# PERSONALIZED HELLO WORLD : LOGIC

- logic : a **view** rule

```
person_hasGreeting(name, greeting)
  <- person_speaks(name, language),
     greetings(language, greeting).
```

- SQL analogy

```
CREATE VIEW person_hasGreeting AS
SELECT
  person_speaks.name,
  greetings.content AS greeting
WHERE
  person_speaks.language = greetings.language
FROM person_speaks, greetings
```

# PERSONALIZED HELLO WORLD : LOGIC

- logic : a **view** rule

```
person_hasGreeting(name, greeting)
  <- person_speaks(name, language),
     greetings(language, greeting).
```

- SQL analogy

```
CREATE MATERIALIZED VIEW person_hasGreeting AS
SELECT DISTINCT
  person_speaks.name,
  greetings.content AS greeting
WHERE
  person_speaks.language = greetings.language
FROM person_speaks, greetings
```

# PERSONALIZED HELLO WORLD : DATA

- data

```
greetings("English", "Hello World").
greetings("German", "Hallo Welt").

person_speaks("Sarah", "English").
person_speaks("Lisa", "German").
```

# PERSONALIZED HELLO WORLD : DATA

- data

```
greetings("English", "Hello World").
greetings("German", "Hallo Welt").

person_speaks("Sarah", "English").
person_speaks("Lisa", "German").
```

rules where
rule body = true

# PERSONALIZED HELLO WORLD : DATA

- data

```
greetings("English", "Hello World")  <- 1 = 1.
greetings("German", "Hallo Welt")    <- 1 = 1.

person_speaks("Sarah", "English")    <- 1 = 1.
person_speaks("Lisa", "German")      <- 1 = 1.
```

rules where
rule body = true

# RUN PERSONALIZED HELLO WORLD

- install and print

```
% lb create --overwrite helloworld
```

# RUN PERSONALIZED HELLO WORLD

- install and print

```
% lb create --overwrite helloworld

% lb addblock helloworld -f helloworld.logic
```

# RUN PERSONALIZED HELLO WORLD

- install and print

```
% lb create --overwrite helloworld

% lb addblock helloworld -f helloworld.logic

% lb print helloworld person_hasGreeting
"Lisa"    "Hallo Welt"
"Sarah"   "Hello World"
```

# EVEN MORE PERSONALIZATION

- query

```
% lb query helloworld '
  _(greeting)
     <- person_hasGreeting("Sarah", greeting).'

"Hello World"
```

# EVEN MORE PERSONALIZATION

- query

```
% lb query helloworld '
  _(greeting)
     <- person_hasGreeting("Sarah", greeting).'

"Hello World"
```

# EVEN MORE PERSONALIZATION

- query

```
% lb query helloworld '
  _(greeting)
     <- person_hasGreeting("Sarah", greeting).'

"Hello World"

% lb query helloworld '
  _(greeting)
     <- person_hasGreeting("Lisa", greeting).'

"Hallo Welt"
```

# MORE ABOUT LOGIC

- adding more data

```
person_speaks("Guido van Rossum", "Dutch").
```

- personalized greeting for Guido?

```
% lb query helloworld '
  _(greeting)
     <- person_hasGreeting("Guido van Rossum",
                           greeting).'
```

# HANDLING GUIDO

- model
  - default greeting
- logic
  - default greeting is the English greeting
  - if a person speaks a language where no greeting is known, use default greeting

# HANDLING GUIDO : MODEL

- model : default greeting

```
default_greeting(greeting) -> string(greeting).
```

# HANDLING GUIDO : LOGIC

- logic
  - default greeting is the English greeting

```
default_greeting(greeting) <- greetings("English", greeting).
```

# HANDLING GUIDO : LOGIC

- logic
  - default greeting is the English greeting

```
default_greeting(greeting) <- greetings("English", greeting).
```

  - if a person speaks a language where no greeting is known, use default greeting

```
person_hasGreeting(name, greeting)
  <- person_speaks(name, language),
     !greetings(language, _),
     default_greeting(greeting).
```

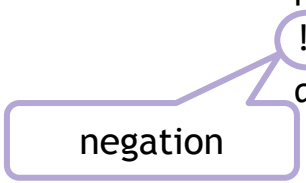
# HANDLING GUIDO : LOGIC

- logic
  - default greeting is the English greeting

```
default_greeting(greeting) <- greetings("English", greeting).
```

  - if a person speaks a language where no greeting is known, use default greeting

```
person_hasGreeting(name, greeting)
  <- person_speaks(name, language),
     !greetings(language, _),
     default_greeting(greeting).
```



# HANDLING GUIDO : DISJUNCTION

```
person_hasGreeting(name, greeting)
  <- person_speaks(name, language),
     !greetings(language, _),
     default_greeting(greeting).


person_hasGreeting(name, greeting)
  <- person_speaks(name, language),
     greetings(language, greeting).
```

# HANDLING GUIDO : DISJUNCTION

```
person_hasGreeting(name, greeting)
  <- person_speaks(name, language),
     ( !greetings(language, _),
       default_greeting(greeting)
     ; greetings(language, greeting) ).
```

disjunction

# LESSON 1 SUMMARY

- model
  - predicate declarations and primitive types
- logic
  - rules: basic, conjunction, disjunction, negation
- data
  - specified using rules
- interacting with the runtime
  - lb command : create, addblock, print, query

# LESSON 1 LAB EXERCISES