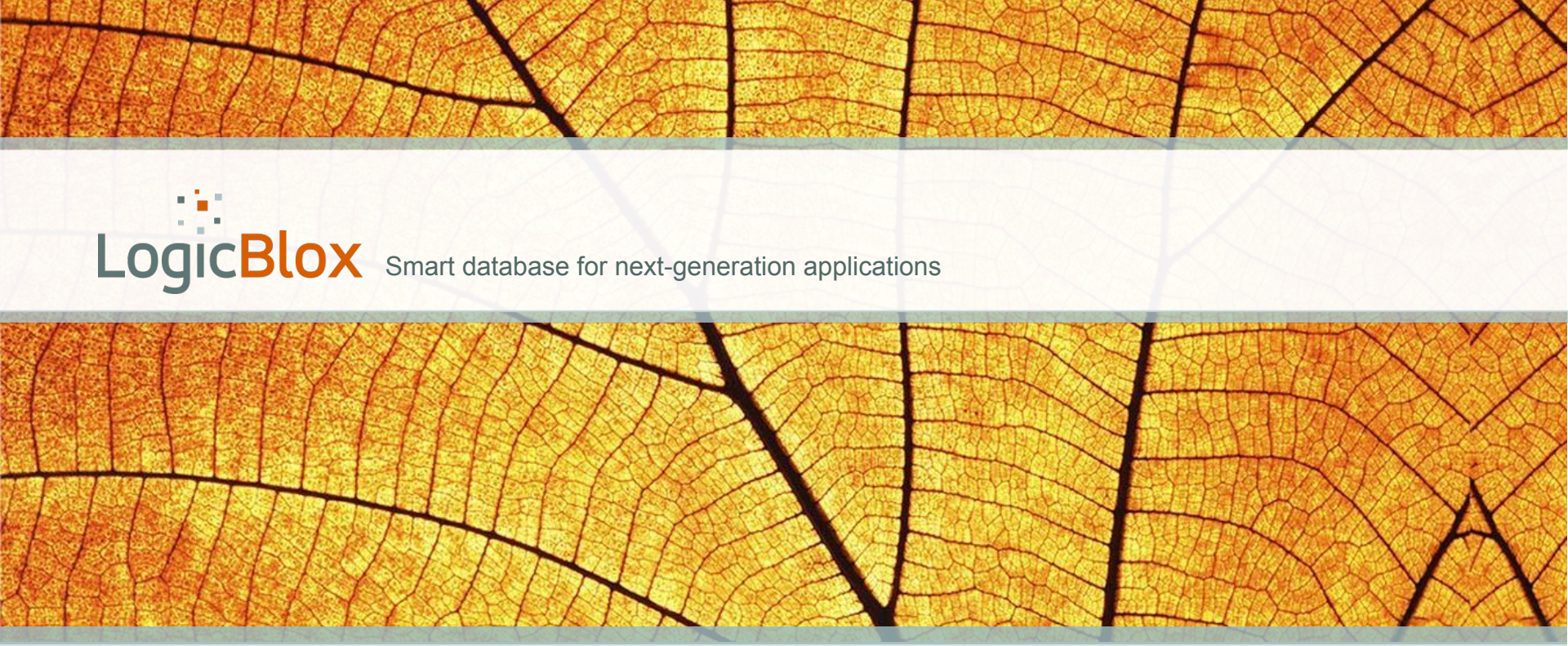


LogicBlox Smart database for next-generation applications

LESSON 2 : STATE AND EVENTS

LOGICBLOX TRAINING

A CLOSER LOOK AT HELLO WORLD

helloworld.logic

```
/****** model ************************/
greetings(language, content) -> ... .
person_speaks(name, language) -> ... .
person_hasGreeting(name, greeting) -> ... .


/****** logic ************************/
person_hasGreeting(name, greeting)
  <- person_speaks(name, language),
     greetings(language, greeting).


/****** data ************************/
greetings("German", "Hallo Welt").
person_speaks("Lisa", "German").
```

A CLOSER LOOK AT HELLO WORLD

helloworld.logic

```
/****** model ************************/
greetings(language, content) -> ... .
person_speaks(name, language) -> ... .
person_hasGreeting(name, greeting) -> ... .
```

Predicate Declarations

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

```
/****** data ************************/
greetings("German", "Hallo Welt").
person_speaks("Lisa", "German").
```

A CLOSER LOOK AT HELLO WORLD

helloworld.logic

```
/****** model ************************/
greetings(language, content) -> ... .
person_speaks(name, language) -> ... .
person_hasGreeting(name, greeting) -> ... .
```

Predicate Declarations

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

View Rules

```
/****** data ************************/
greetings("German", "Hallo Welt").
person_speaks("Lisa", "German").
```

A CLOSER LOOK AT HELLO WORLD

helloworld.logic

```
/****** model ************************/
greetings(language, content) -> ... .
person_speaks(name, language) -> ... .
person_hasGreeting(name, greeting) -> ... .
```

Predicate Declarations

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

View Rules

```
/****** data ************************/
greetings("German", "Hallo Welt").
person_speaks("Lisa", "German").
```

View Rules

RUNNING THE PROGRAM

```
% lb create --overwrite helloworld

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

SEMANTICS OF ADDBLOCK

```
% lb create --overwrite helloworld

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

- open transaction
- create predicates
- add the rules
- rule maintenance
- commit

SEMANTICS OF ADDBLOCK

```
% lb create --overwrite helloworld

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

- open transaction
- create predicates
- add the rules
- rule maintenance
- commit

live in the workspace until the end of time: ***"database lifetime"***

SEMANTICS OF ADDBLOCK

```
% lb create --overwrite helloworld

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

- open transaction
- create predicates
- add the rules
- rule maintenance
- commit

live in the workspace until the end of time: ***"database lifetime"***

all database lifetime rules

RULE MAINTENANCE

- runtime ensures that the contents of predicates are consistent with the rules

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


/****** data ************************/
greetings("German", "Hallo Welt").
person_speaks("Lisa", "German").
```

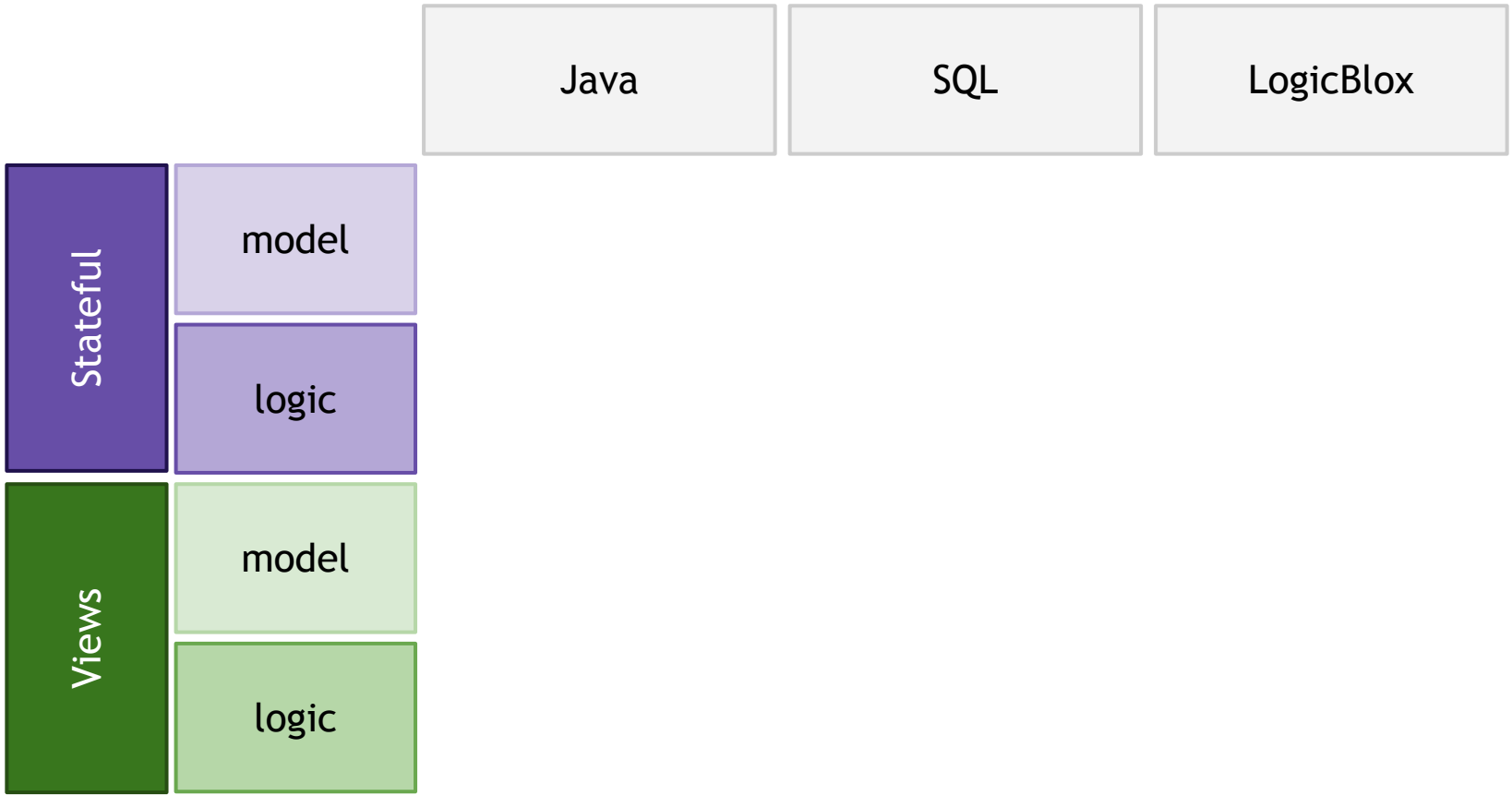
MAKING A CHANGE

- runtime ensures that the contents of predicates are consistent with the rules

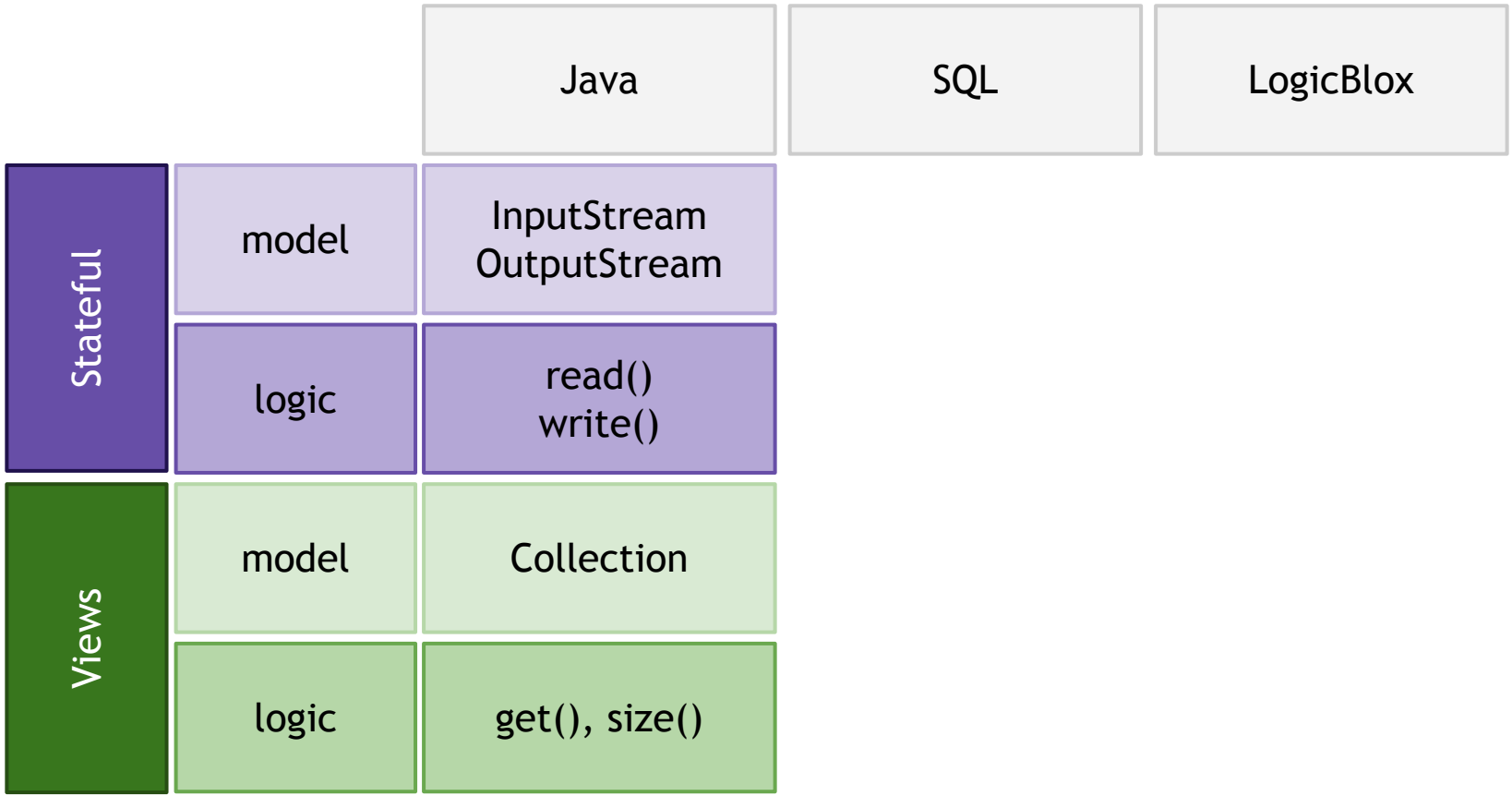
```
/****** logic ************************/
person_hasGreeting(name, greeting)
  <- person_speaks(name, language),
     greetings(language, greeting).


/****** data *************************/
greetings("German", "Hallo Welt").
person_speaks("Lisa", "German").
```

How to make "Lisa" speak "Hungarian", instead?

STATE IN PROGRAMMING SYSTEMS



STATE IN PROGRAMMING SYSTEMS



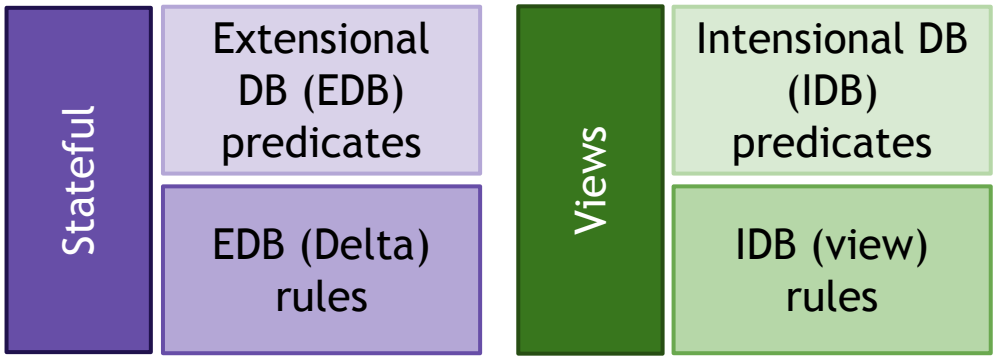
STATE IN PROGRAMMING SYSTEMS

		Java	SQL	LogicBlox
Stateful	model	InputStream OutputStream	tables	
	logic	read() write()	INSERT INTO	
Views	model	Collection	views, queries	Intensional DB (IDB) predicates
	logic	get(), size()	CREATE VIEW AS SELECT	IDB (view) rules

STATE IN PROGRAMMING SYSTEMS

		Java	SQL	LogicBlox
Stateful	model	InputStream OutputStream	tables	Extensional DB (EDB) predicates
	logic	read() write()	INSERT INTO	EDB (Delta) rules
Views	model	Collection	views, queries	Intensional DB (IDB) predicates
	logic	get(), size()	CREATE VIEW AS SELECT	IDB (view) rules

STATE IN LOGICBLOX



- strict separation between stateful & views
  - predicates & rules: either IDB or EDB, not both
  - IDB rules derive into IDB predicates
  - EDB rules derive into EDB predicates

EDB PREDICATE

```
/****** model ************************/
person_speaks(name, language)
  -> string(name), string(language).



/****** data *************************/
person_speaks("Lisa", "German").
```

EDB PREDICATE

```
/****** model ************************/
person_speaks(name, language)
  -> string(name), string(language).

lang:derivationType[`person_speaks] = "Extensional".



/****** data ************************/
person_speaks("Lisa", "German").
```

EDB PREDICATE AND IDB RULE

helloworld.logic

```
/****** model ************************/
person_speaks(name, language)
  -> string(name), string(language).

lang:derivationType[`person_speaks] = "Extensional".



/****** data ************************/
person_speaks("Lisa", "German").
```

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

EDB PREDICATE AND IDB RULE

helloworld.logic

```
/****** model ******************/
person_speaks(name, language)
  -> string(name), string(language).

lang:derivationType[`person_speaks] = "Extensional".



/****** data ******************/
person_speaks("Lisa", "German").
```

Predicate 'person_speaks' is an EDB predicate, with derivation type 'extensional', but this rule uses it as an IDB. (**code: EDB_RULE**)

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

EDB PREDICATE AND DELTA RULE

```
/****** model ************************/
person_speaks(name, language)
  -> string(name), string(language).

lang:derivationType[`person_speaks] = "Extensional".



/****** data ************************/
+person_speaks("Lisa", "German").
```

EDB PREDICATE AND DELTA RULE

```
/****** model ************************/
person_speaks(name, language)
  -> string(name), string(language).

lang:derivationType[`person_speaks] = "Extensional".



/****** data ************************/
+person_speaks("Lisa", "German").
```

assertion operator

EDB PREDICATE AND RULE

helloworld.logic

```
/****** model ************************/
person_speaks(name, language)
  -> string(name), string(language).

lang:derivationType[`person_speaks] = "Extensional".
```

data.logic

```
/****** data ************************/
+person_speaks("Lisa", "German").
```

ADDBLOCK VS. EXEC

```
% lb create --overwrite helloworld

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

data.logic

```
/****** data ************************/
+person_speaks("Lisa", "German").
```

ADDBLOCK VS. EXEC

```
% lb create --overwrite helloworld

% lb addblock helloworld -f helloworld.logic

% lb exec helloworld -f data.logic
```

data.logic

```
/****** data ************************/
+person_speaks("Lisa", "German").
```

ADDBLOCK VS. EXEC

```
% lb create --overwrite helloworld

% lb addblock helloworld -f helloworld.logic

% lb exec helloworld -f data.logic
```

"database lifetime"

data.logic

```
/****** data ************************/
+person_speaks("Lisa", "German").
```

ADDBLOCK VS. EXEC

```
% lb create --overwrite helloworld

% lb addblock helloworld -f helloworld.logic

% lb exec helloworld -f data.logic
```

"database lifetime"

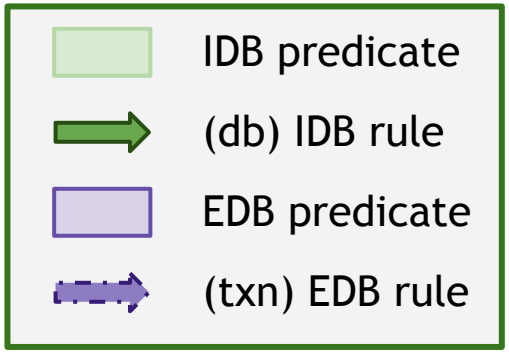
"transaction lifetime"

data.logic

```
/****** data ************************/
+person_speaks("Lisa", "German").
```

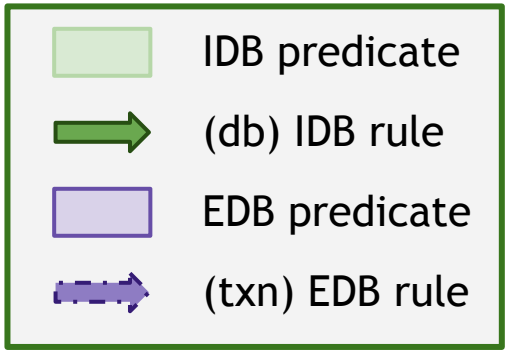
ADDBLOCK VS. EXEC

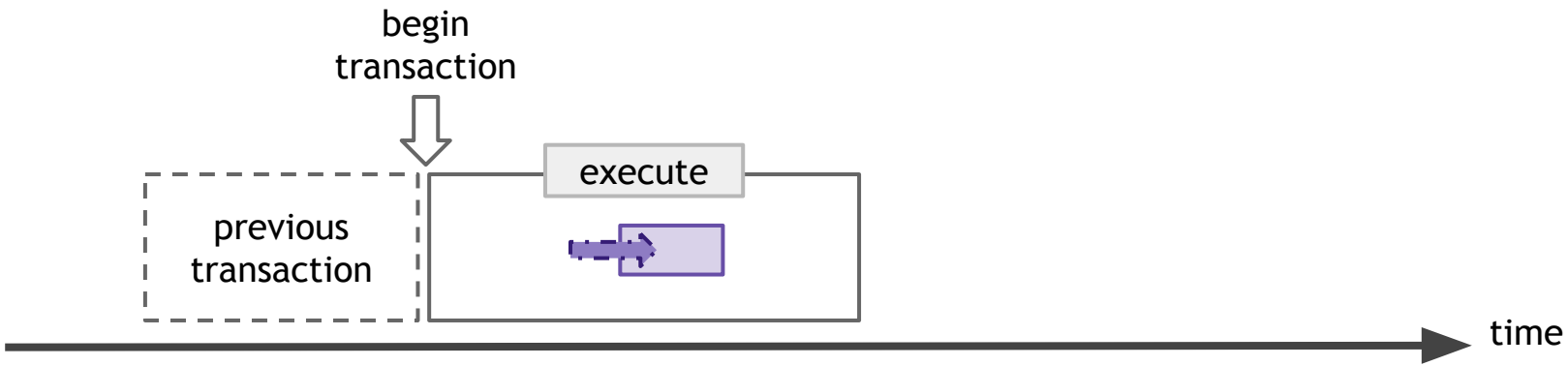
```
% lb create --overwrite helloworld

% lb addblock helloworld -f helloworld.logic

% lb exec helloworld -f data.logic
```



ADDBLOCK VS. EXEC

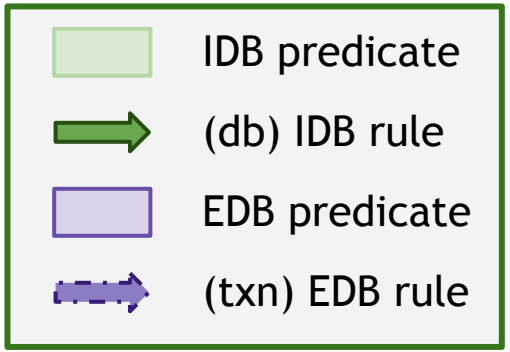
```
% lb create --overwrite helloworld

% lb addblock helloworld -f helloworld.logic

% lb exec helloworld -f data.logic
```

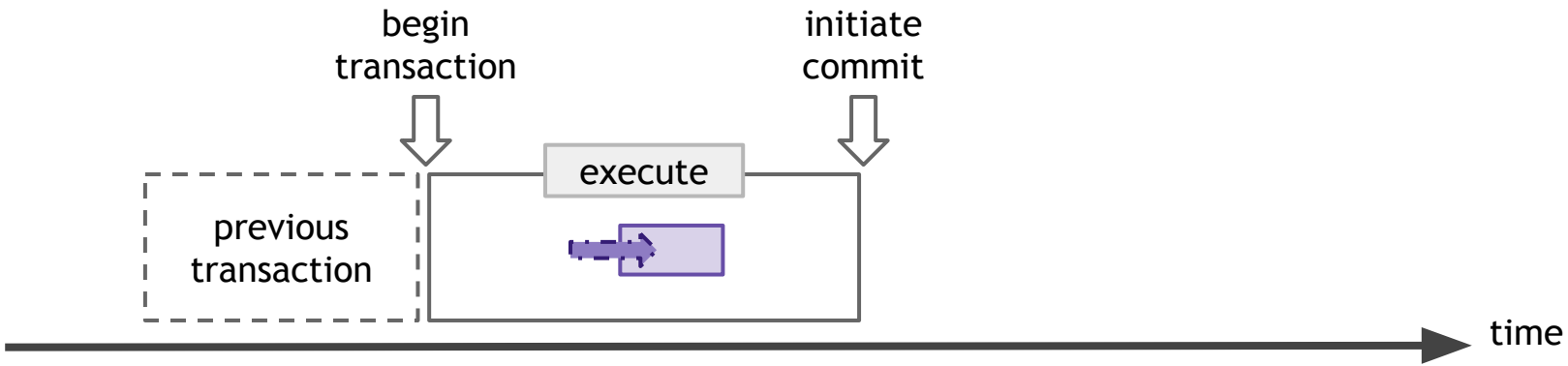




ADDBLOCK VS. EXEC

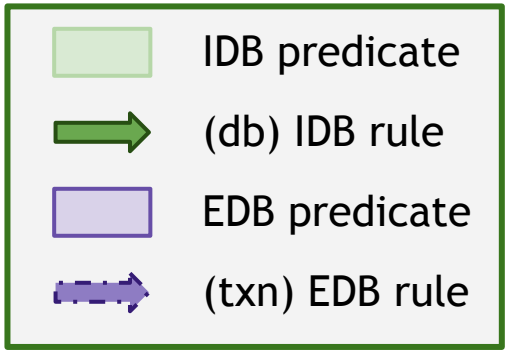
```
% lb create --overwrite helloworld

% lb addblock helloworld -f helloworld.logic

% lb exec helloworld -f data.logic
```

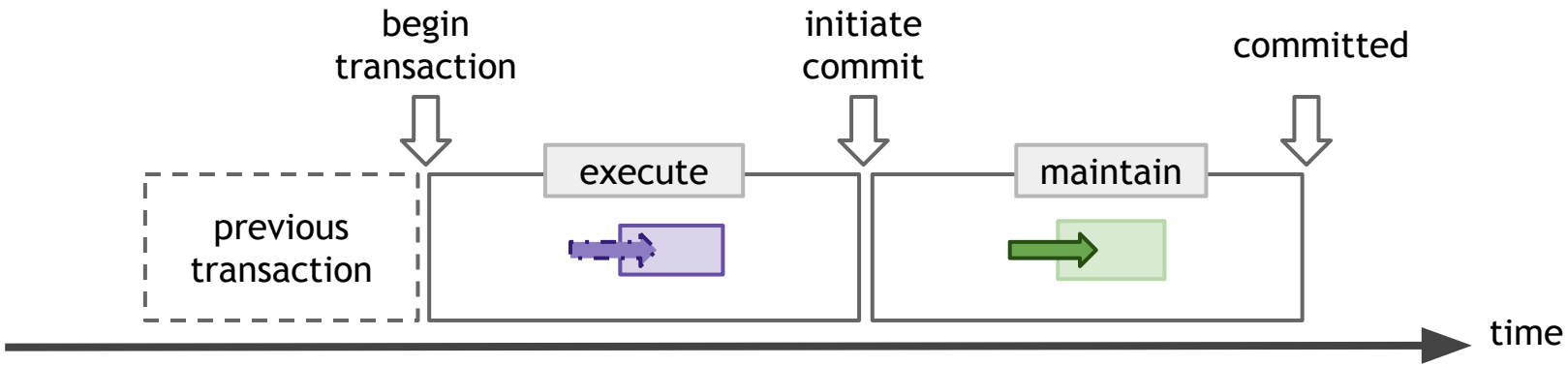




ADDBLOCK VS. EXEC

```
% lb create --overwrite helloworld

% lb addblock helloworld -f helloworld.logic

% lb exec helloworld -f data.logic
```



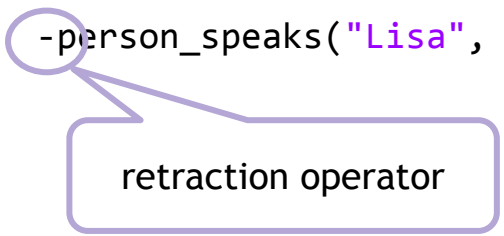


MODIFY PERSON_SPEAKS

```
% lb create --overwrite helloworld

% lb addblock helloworld -f helloworld.logic

% lb exec helloworld -f data.logic

% lb exec helloworld '
    -person_speaks("Lisa", "German").'
```

MODIFY PERSON_SPEAKS

```
% lb create --overwrite helloworld

% lb addblock helloworld -f helloworld.logic

% lb exec helloworld -f data.logic

% lb exec helloworld '
   -person_speaks("Lisa", "German").'
```



UPDATE = RETRACT AND ASSERT

```
% lb create --overwrite helloworld

% lb addblock helloworld -f helloworld.logic

% lb exec helloworld -f data.logic

% lb exec helloworld '
    -person_speaks("Lisa", "German").
    +person_speaks("Lisa", "Hungarian").'
```

UPDATE = RETRACT AND ASSERT

```
% lb create --overwrite helloworld

% lb addblock helloworld -f helloworld.logic

% lb exec helloworld -f data.logic

% lb exec helloworld '
    -person_speaks("Lisa", "German").
    +person_speaks("Lisa", "Hungarian").'

% lb print helloworld person_speaks
"Lisa"      "Hungarian"
```

DETOUR : LB PREDINFO

```
% lb predinfo helloworld person_speaks

info {
  name: "person_speaks"
  qualified_name: "person_speaks"
  arity: 2
  key_argument: "string"
  key_argument: "string"

  ...

  is_ordered: false
  has_default_value: false
  is_pulse_predicate: false
  is_calculated: false
  derivation_type: EXTENSIONAL
  is_constructor: false
}
```

DETOUR : LB PREDINFO

```
% lb predinfo helloworld person_hasGreeting

info {
  name: "person_hasGreeting"
  qualified_name: "person_hasGreeting"
  arity: 2
  key_argument: "string"
  key_argument: "string"

  ...

  is_ordered: false
  has_default_value: false
  is_pulse_predicate: false
  is_calculated: false
  derivation_type: DERIVED_AND_STORED
  is_constructor: false
}
```

RETRACTING NON-EXISTENT TUPLE

```
% lb print helloworld person_speaks
"Lisa"      "Hungarian"

% lb exec helloworld '
    -person_speaks("Lisa", "German").'
```

RETRACTING NON-EXISTENT TUPLE

```
% lb print helloworld person_speaks
"Lisa"      "Hungarian"

% lb exec helloworld '
    -person_speaks("Lisa", "German").'
```

```
Error: Functional dependency violation: person_speaks at position Lisa,
German occurred while validating deltas, with base value: , assertion
value: , retraction value:
```

RETRACTING SAFELY

```
% lb print helloworld person_speaks
"Lisa"      "Hungarian"

% lb exec helloworld '
    -person_speaks("Lisa", "German")
       <- person_speaks@prev("Lisa", "German").'
```

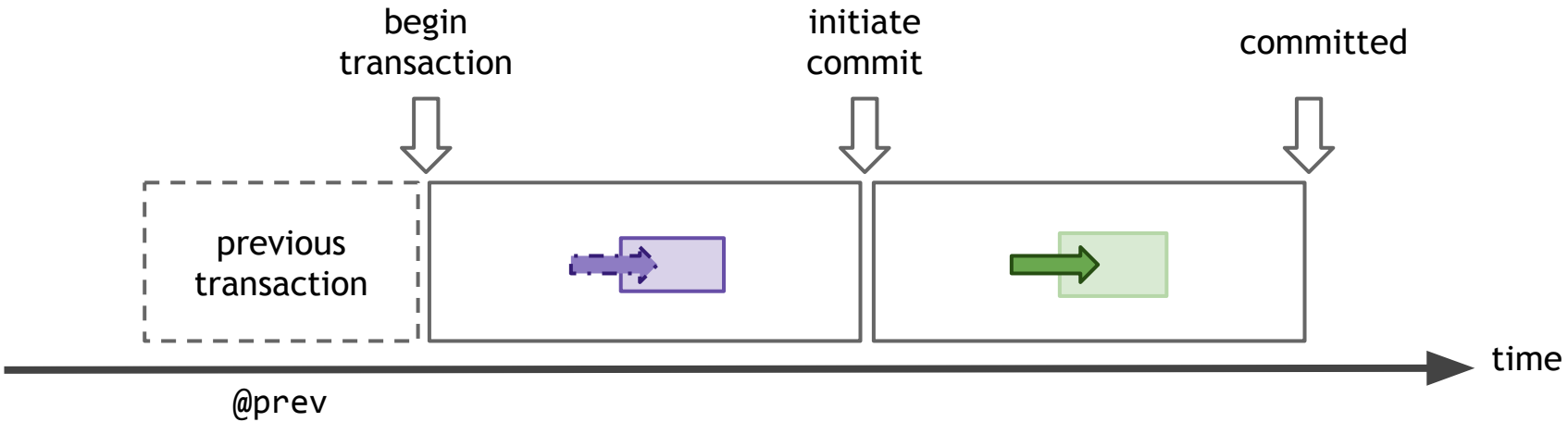
RETRACTING SAFELY

```
% lb print helloworld person_speaks
"Lisa"      "Hungarian"

% lb exec helloworld '
    -person_speaks("Lisa", "German")
       <- person_speaks@prev("Lisa", "German").'
```

stage annotation

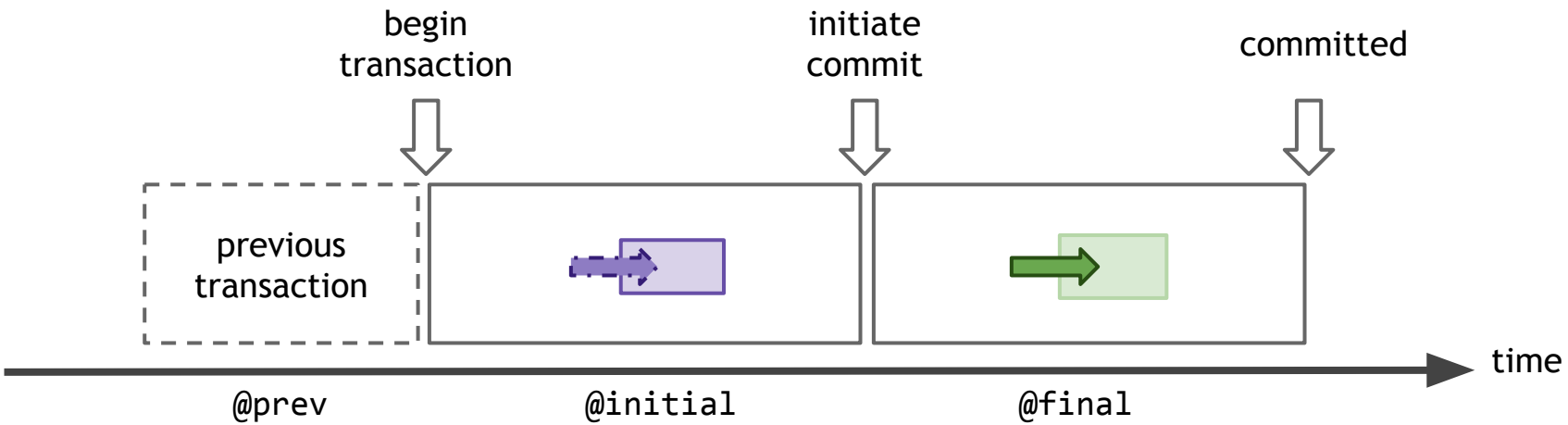
STAGE ANNOTATIONS

```
% lb print helloworld person_speaks
"Lisa"      "Hungarian"

% lb exec helloworld '
    -person_speaks("Lisa", "German")
       <- person_speaks@prev("Lisa", "German").'
```



STAGE ANNOTATIONS

```
% lb print helloworld person_speaks
"Lisa"      "Hungarian"

% lb exec helloworld '
    -person_speaks("Lisa", "German")
       <- person_speaks@prev("Lisa", "German").'
```



REACTING TO EVENTS

helloworld.logic

```
...

language(lang) -> string(lang).
lang:derivationType[`language] = "Extensional".
```

REACTING TO EVENTS

helloworld.logic

```
...

language(lang) -> string(lang).
lang:derivationType[`language] = "Extensional".
```

```
% lb exec helloworld '
    +person_speaks("Lisa", "German").
    +language(lang)
      <- +person_speaks(_, lang).'
```

POP QUIZ 1

helloworld.logic

```
...

language(lang) -> string(lang).
lang:derivationType[`language] = "Extensional".
```

```
% lb exec helloworld '
    +person_speaks("Lisa", "German").
    +language(lang)
      <- +person_speaks(_, lang).'

% lb print helloworld language
??
```

POP QUIZ 1 ANSWER

helloworld.logic

```
...

language(lang) -> string(lang).
lang:derivationType[`language] = "Extensional".
```

```
% lb exec helloworld '
    +person_speaks("Lisa", "German").
    +language(lang)
      <- +person_speaks(_, lang).'

% lb print helloworld language
"German"
```

REACTING TO EVENTS

helloworld.logic

```
...

language(lang) -> string(lang).
lang:derivationType[`language] = "Extensional".
```

```
% lb exec helloworld '
    +person_speaks("Lisa", "German").
  +language(lang)
    <- +person_speaks(_, lang).'
```

How to make this rule apply to every transaction?

DATABASE LIFETIME DELTA RULES

events.logic

```
language(lang) -> string(lang).
lang:derivationType[`language] = "Extensional".

+language(lang)
  <- +person_speaks(_, lang).
```

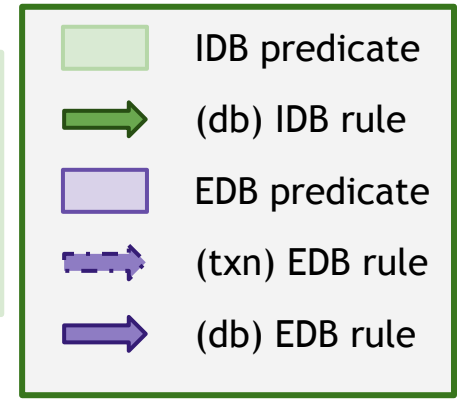
```
% lb addblock helloworld -f events.logic
```

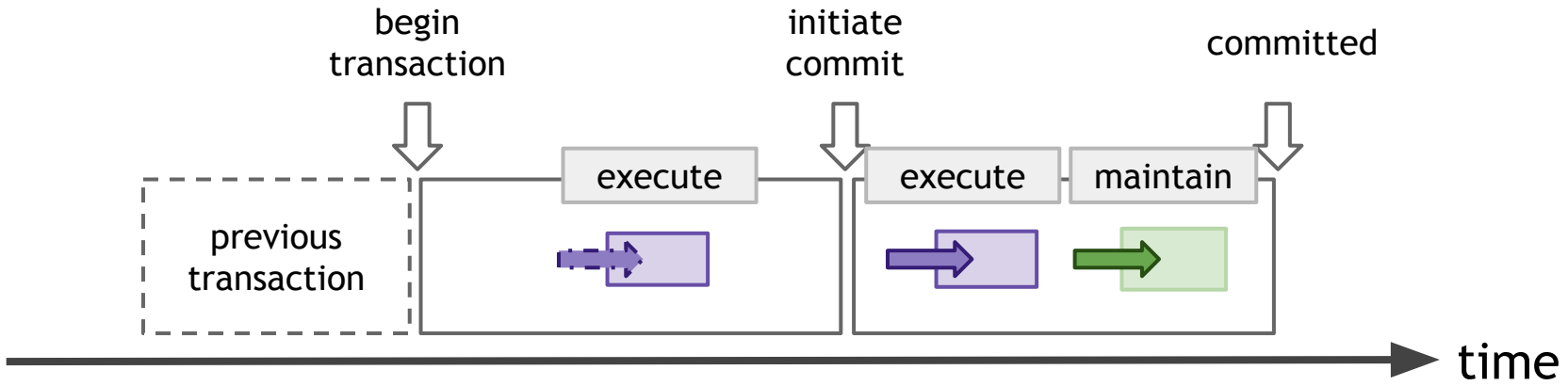
DATABASE LIFETIME DELTA RULES

events.logic

```
language(lang) -> string(lang).
lang:derivationType[`language] = "Extensional".

+language(lang)
  <- +person_speaks(_, lang).
```

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





POP QUIZ 2

events.logic

```
language(lang) -> string(lang).
lang:derivationType[`language] = "Extensional".

+language(lang)
  <- +person_speaks(_, lang).
```

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

% lb exec helloworld '
    +person_speaks("Sarah", "Chinese").'

% lb print helloworld language

??
```

POP QUIZ 2 ANSWER

events.logic

```
language(lang) -> string(lang).
lang:derivationType[`language] = "Extensional".

+language(lang)
  <- +person_speaks(_, lang).
```

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

% lb exec helloworld '
    +person_speaks("Sarah", "Chinese").'

% lb print helloworld language
"Chinese"
```

POP QUIZ 3

events.logic

```
language(lang) -> string(lang).
lang:derivationType[`language] = "Extensional".

+language(lang)
  <- +person_speaks(_, lang).
```

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

% lb exec helloworld '
    +person_speaks("Sarah", "Chinese").'

% lb exec helloworld '
    -person_speaks("Sarah", "Chinese").'

% lb print helloworld language

??
```

POP QUIZ 3 ANSWER

events.logic

```
language(lang) -> string(lang).
lang:derivationType[`language] = "Extensional".

+language(lang)
  <- +person_speaks(_, lang).
```

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

% lb exec helloworld '
    +person_speaks("Sarah", "Chinese").'

% lb exec helloworld '
    -person_speaks("Sarah", "Chinese").'

% lb print helloworld language
"Chinese"
```

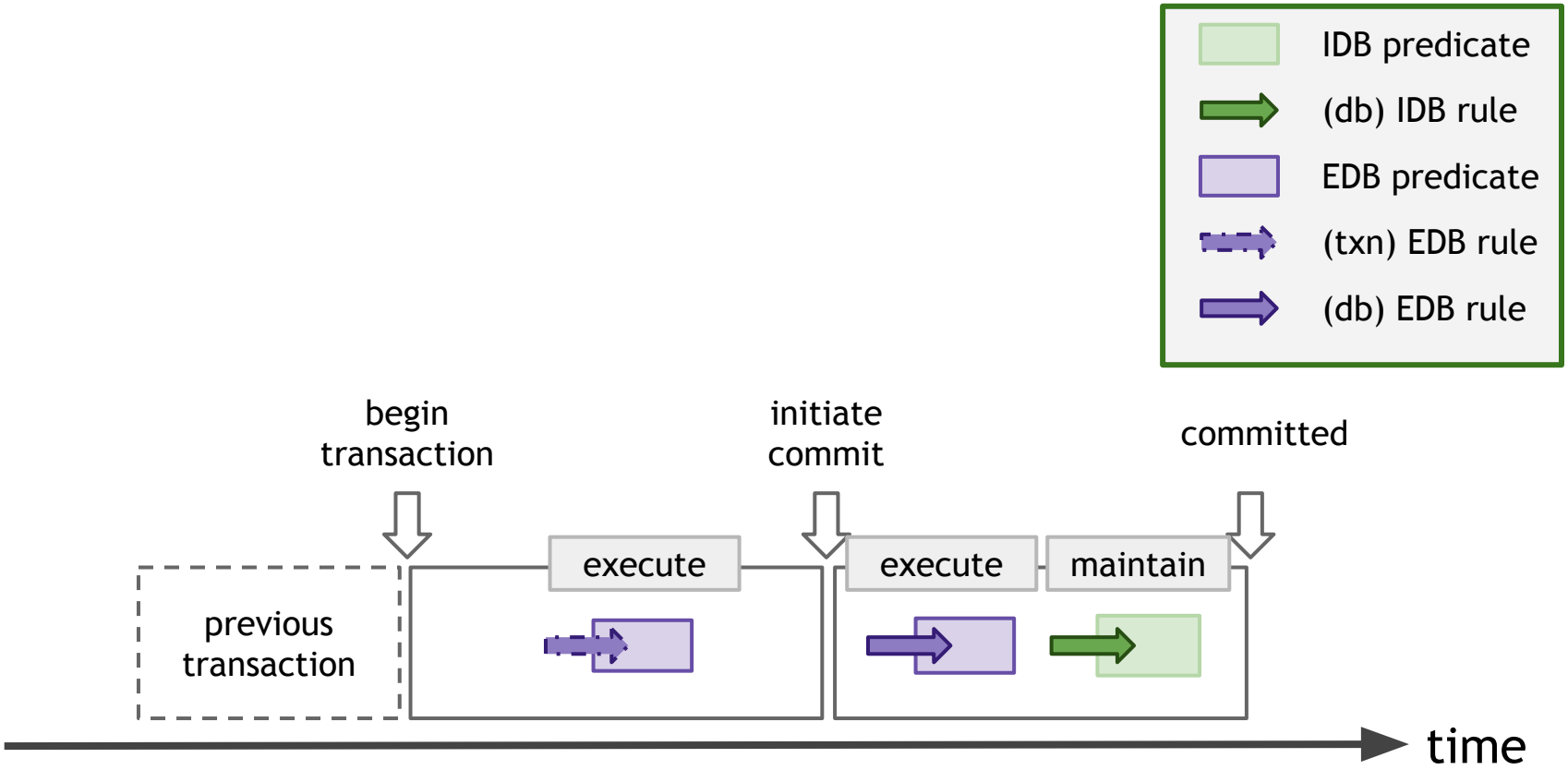
IDB VS. EDB

```
language(lang)
  <- person_speaks(_,lang).
```

```
+language(lang)
  <- +person_speaks(_,lang).
```

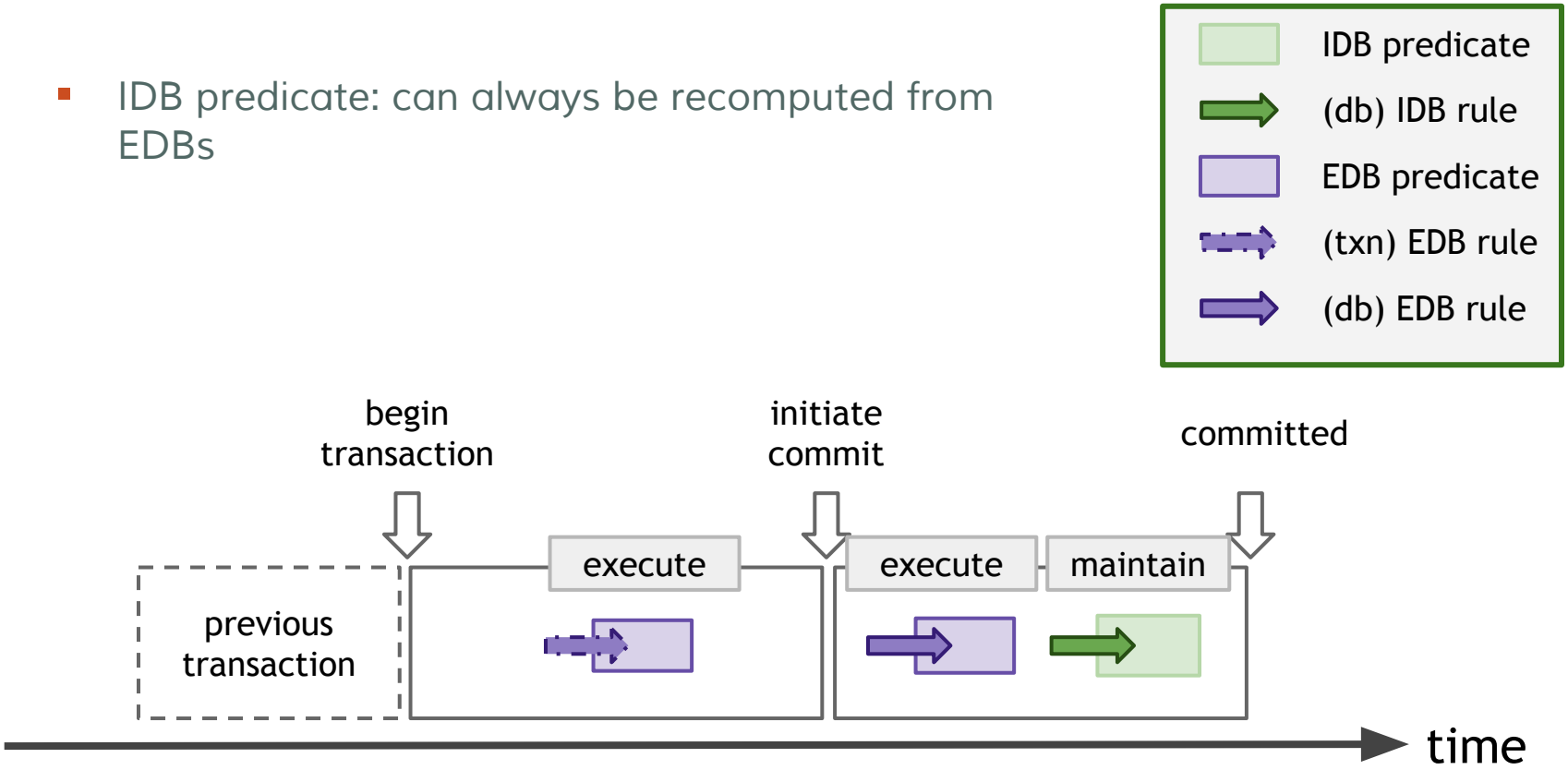
- IDB
  - rules: functions from *EDB state -> IDB state*
  - predicate: can always be recomputed from EDBs
- EDB
  - rules: functions from *(state, event) -> state'*
  - predicate: can be recomputed by replaying event rules in order

IDB VS. EDB : IN PICTURES



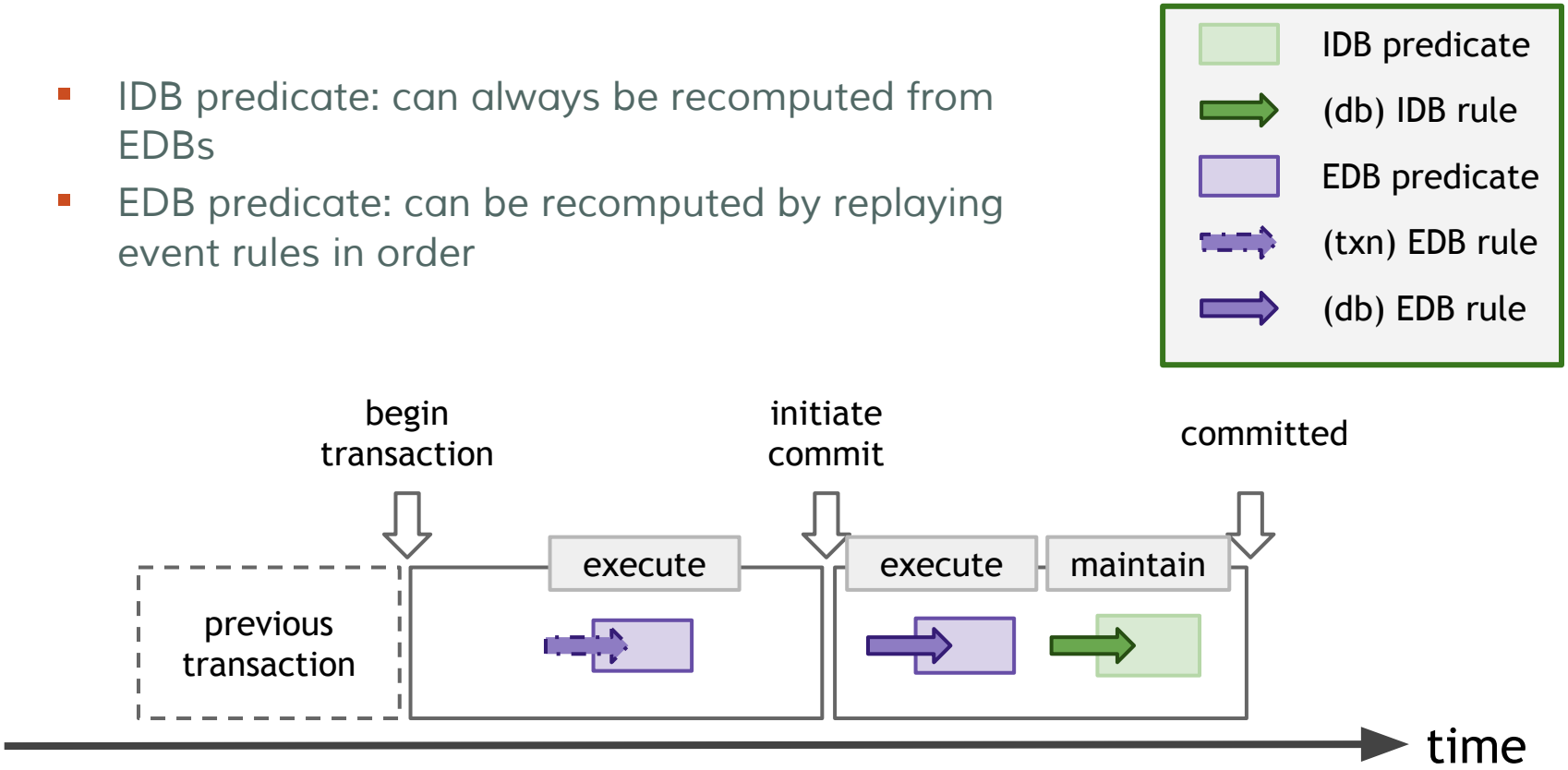
IDB VS. EDB : IN PICTURES

- IDB predicate: can always be recomputed from EDBs



IDB VS. EDB : IN PICTURES

- IDB predicate: can always be recomputed from EDBs
- EDB predicate: can be recomputed by replaying event rules in order



UPDATE IDB

helloworld.logic

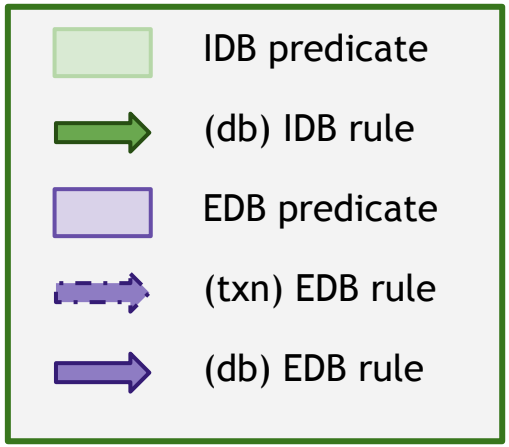
```
...
person_hasGreeting(name, greeting)
  <- person_speaks(name, language),
     greetings(language, greeting).
```

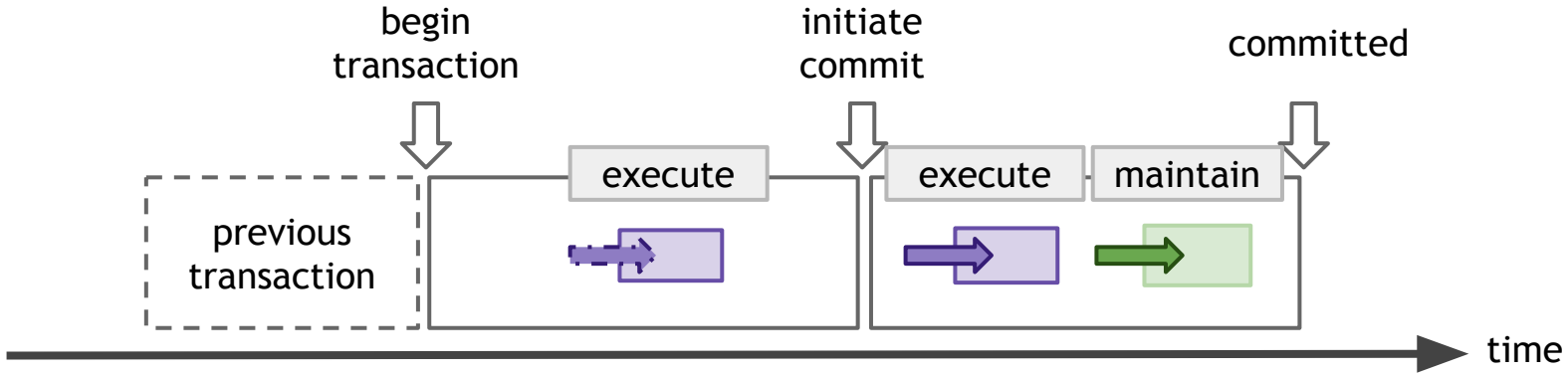
"database lifetime"

```
% lb exec helloworld '
    +person_hasGreeting("Martin", "Hallo Wereld").'

% lb exec helloworld '
    _(greeting)
      <- person_hasGreeting("Martin", greeting).' --print
```

UPDATE IDB : IN PICTURES

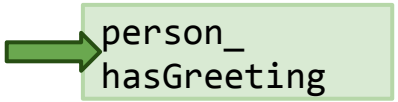
```
person_hasGreeting(name, greeting)
  <- person_speaks(name, language),
     greetings(language, greeting).

% lb exec helloworld '
    +person_hasGreeting("Martin", "Hallo Wereld").'
```

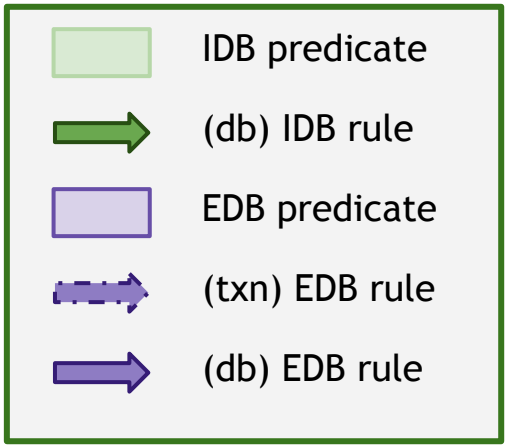




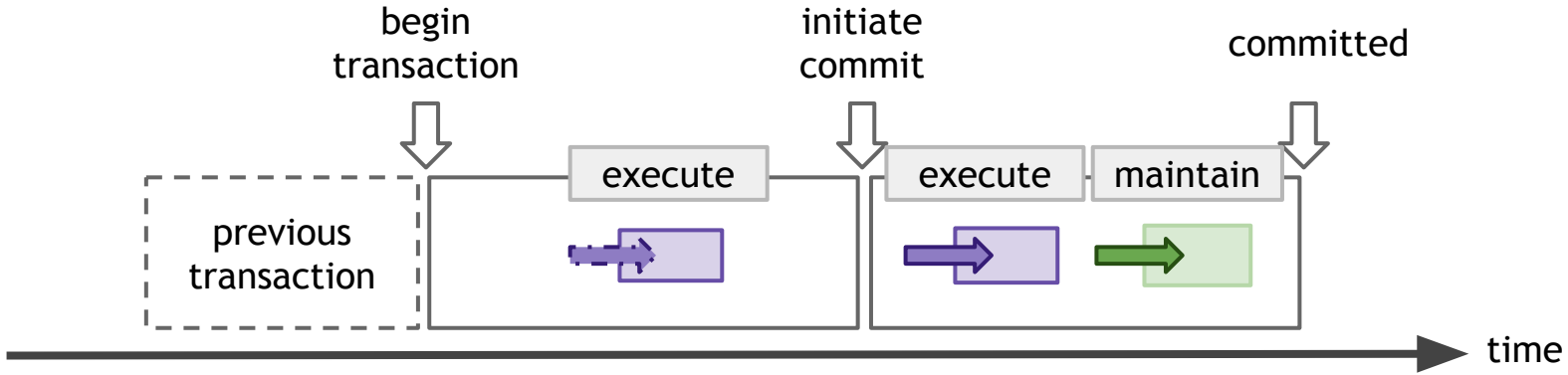
UPDATE IDB : IN PICTURES

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






```
% lb exec helloworld '
    +person_hasGreeting("Martin", "Hallo Wereld").'
```



UPDATE IDB : IN PICTURES

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

person_ hasGreeting

```
% lb exec helloworld '
    +person_hasGreeting("Martin", "Hallo Wereld").'
```

person_ hasGreeting

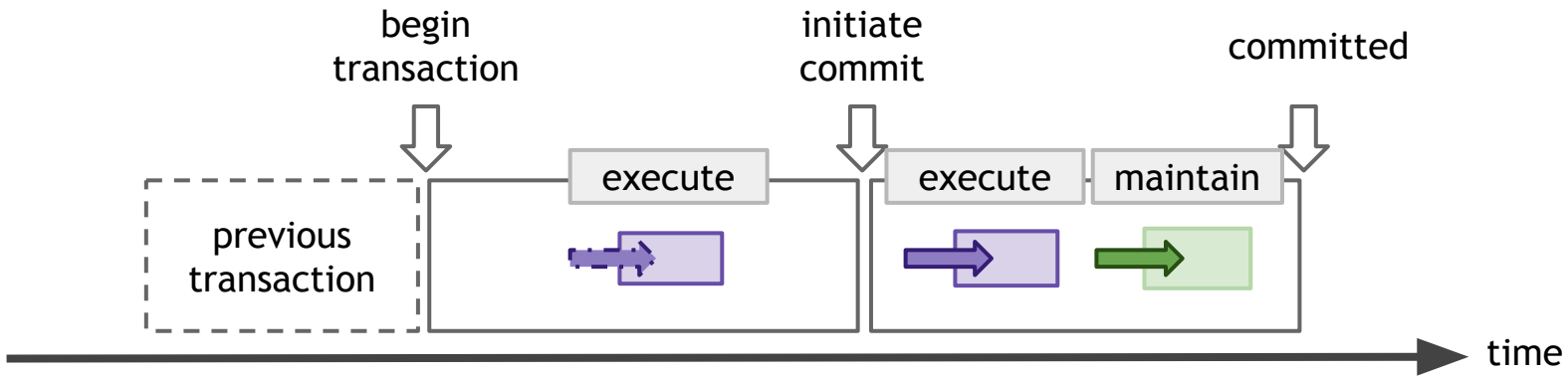
IDB predicate

(db) IDB rule

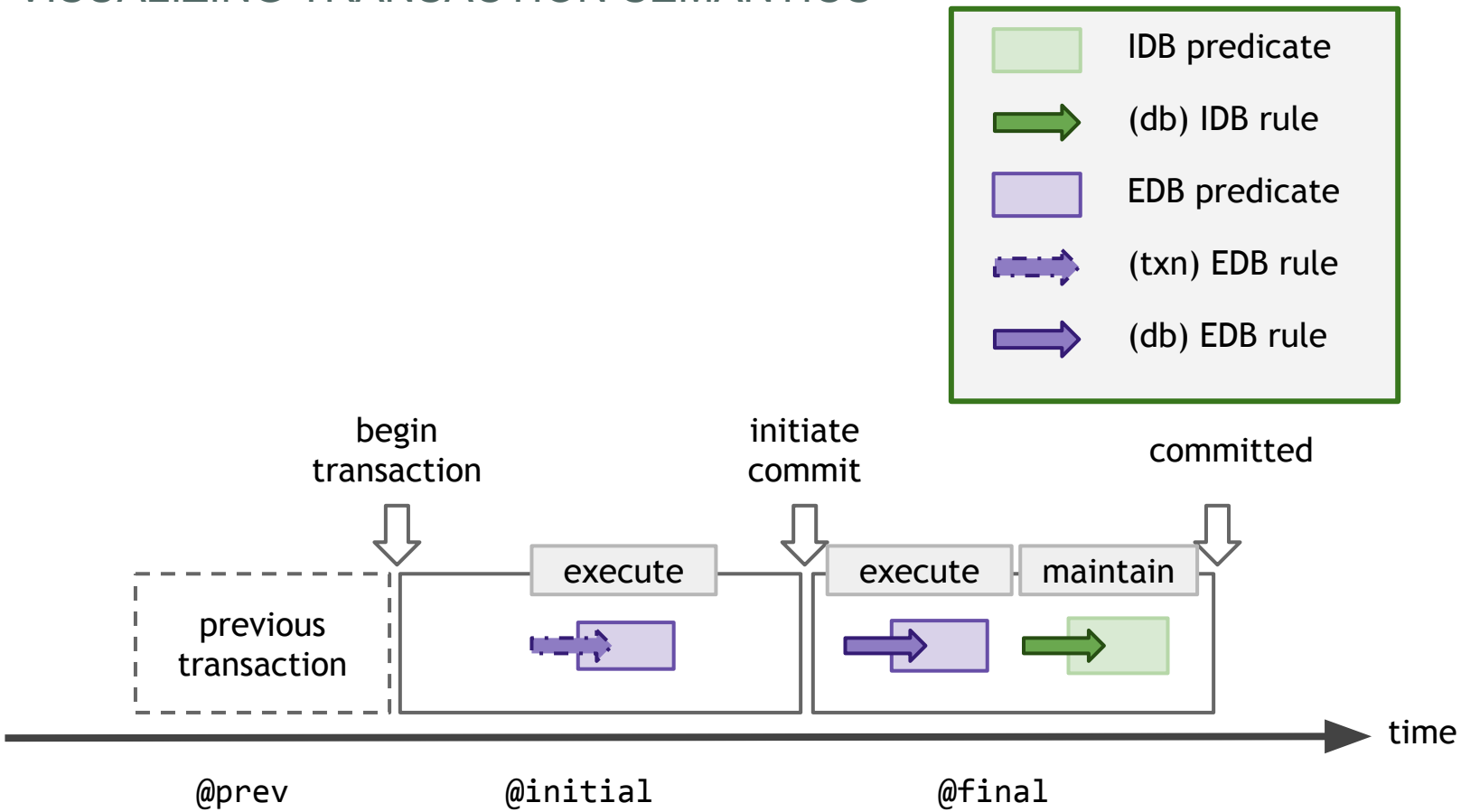
EDB predicate

(txn) EDB rule

(db) EDB rule



VISUALIZING TRANSACTION SEMANTICS



SUMMARY OF CONCEPTS

- EDB
  - predicate content set by external interaction
  - rule: function of time and state
    - delta operators: +, -
    - stage annotations: @prev, @initial, @final
- IDB
  - predicate is a view over state only
  - rule: function of state only
- Transaction semantics
- Lifetime of predicates and rules

LESSON 2 LAB EXERCISES