# Axioms of Versioning

Marc de Graauw

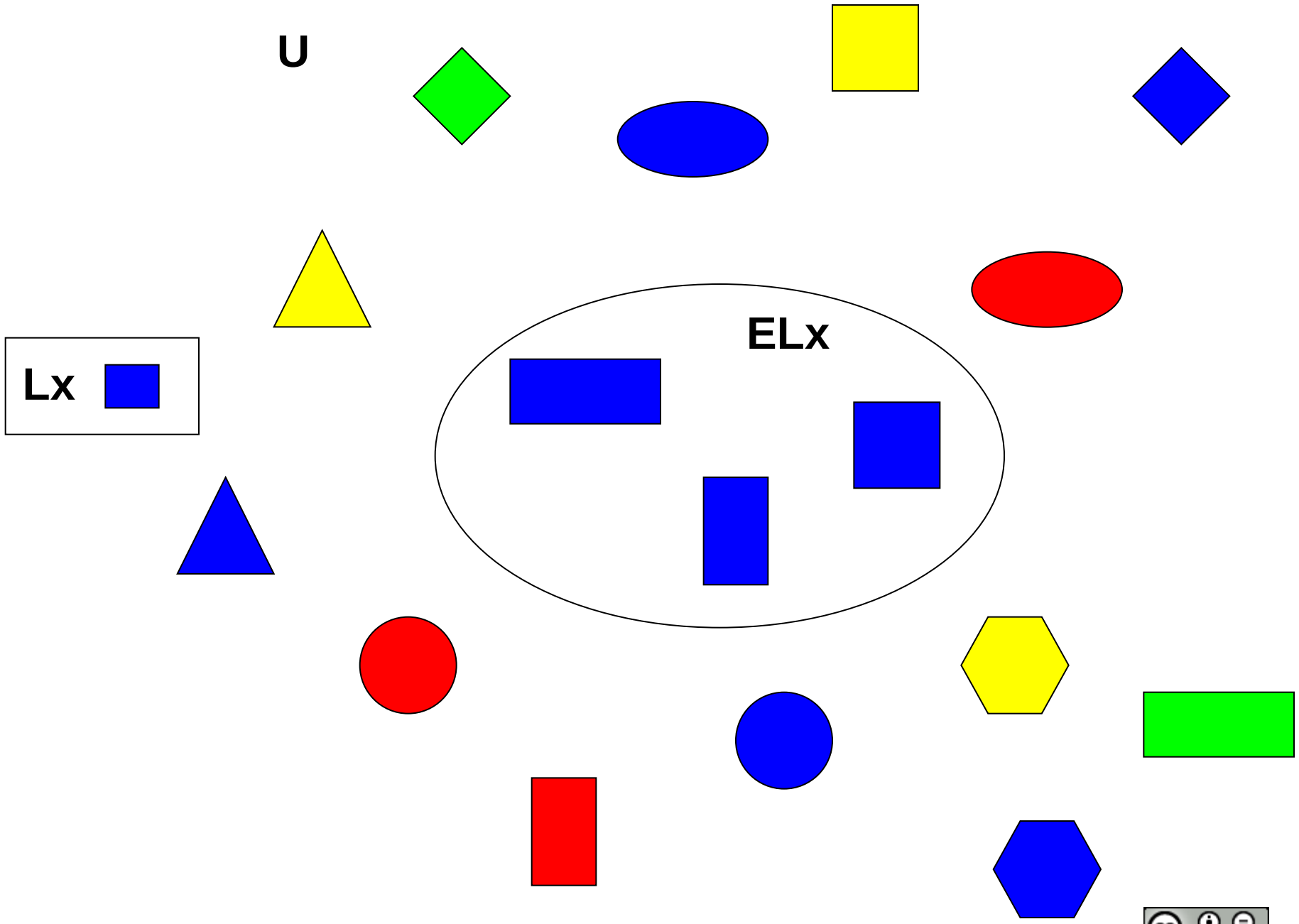
# Intro

- What is *compatibility* anyway?
- A relation between two: languages? applications? documents or messages? schema's?
- What is back- and forward compatibility?
- What does it mean when we say a version of a language is back- or forward compatible with another?
- Focus on messaging

# Semantical Equivalence Sets

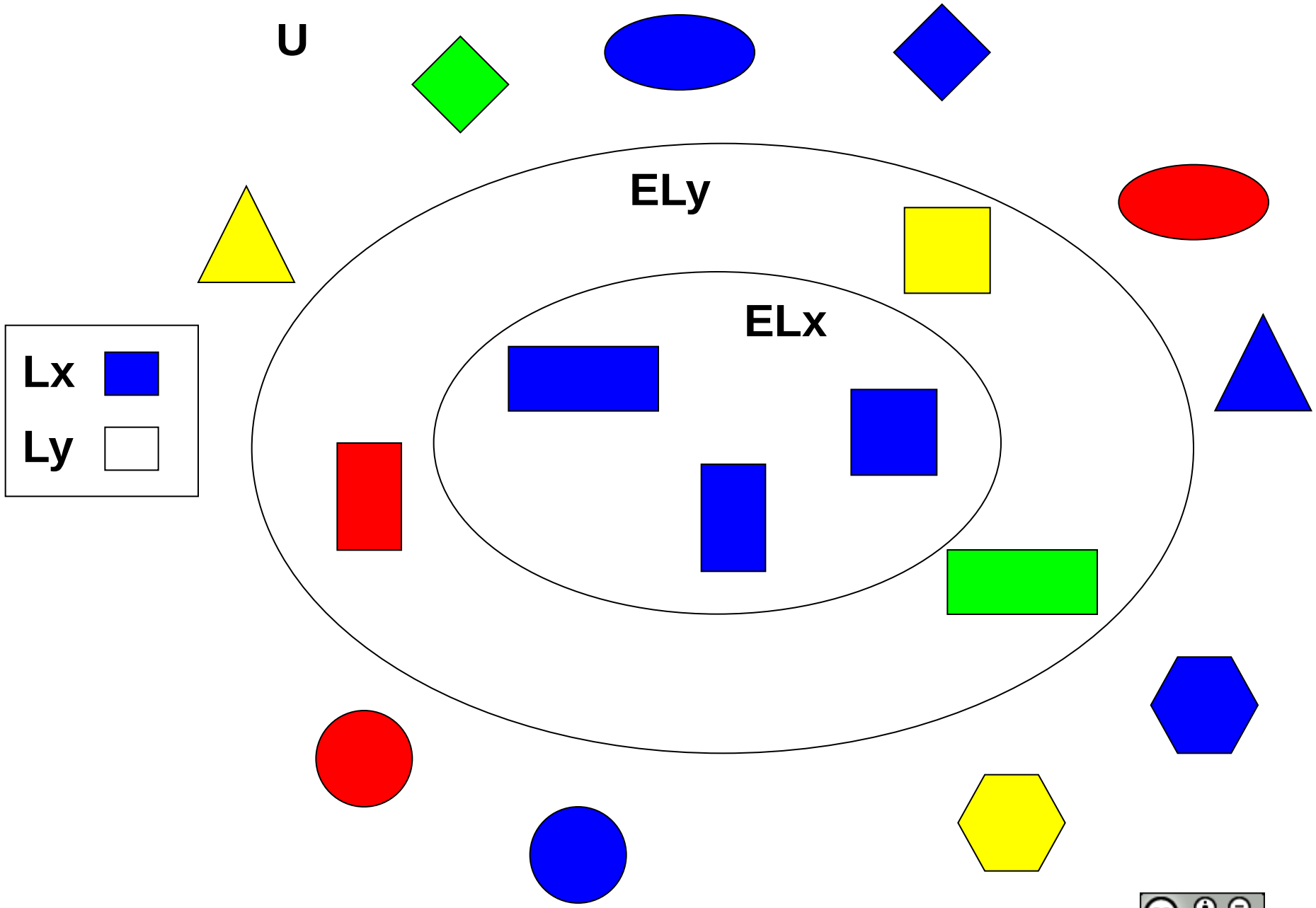
# Refreshening Set Theory

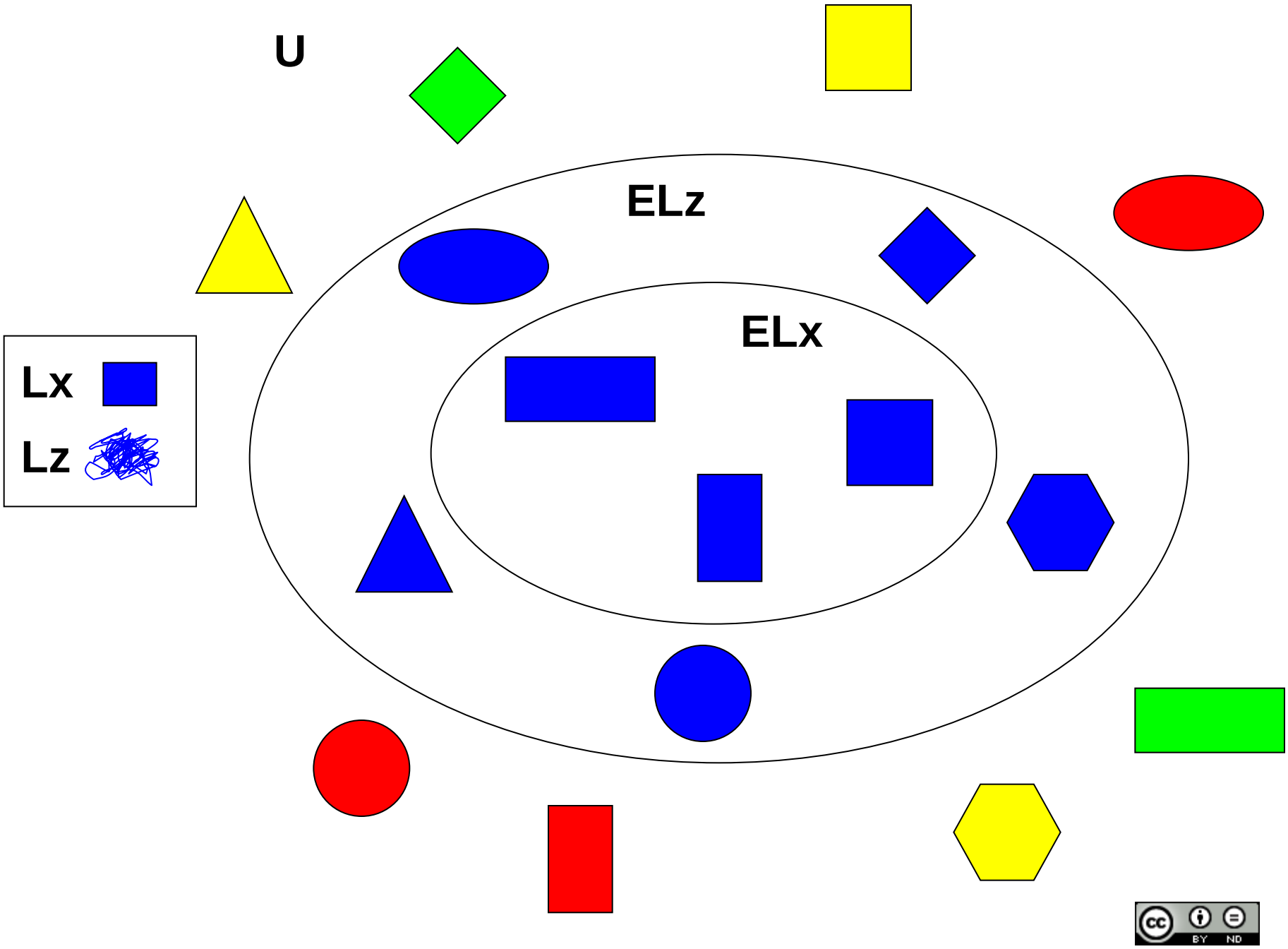
- { cat, dog, lion }
- { 1, 2, 3 }
- { an integer between zero and four }
- { } = ø
- { cat, cat } = { cat }
- { cat, dog } = { dog, cat }
- { cat, dog } subset of { cat, dog, mouse }
- { cat, dog } intersection { cat, lion } = { cat }
- { cat, dog } union { cat, lion } = { cat, dog, lion }

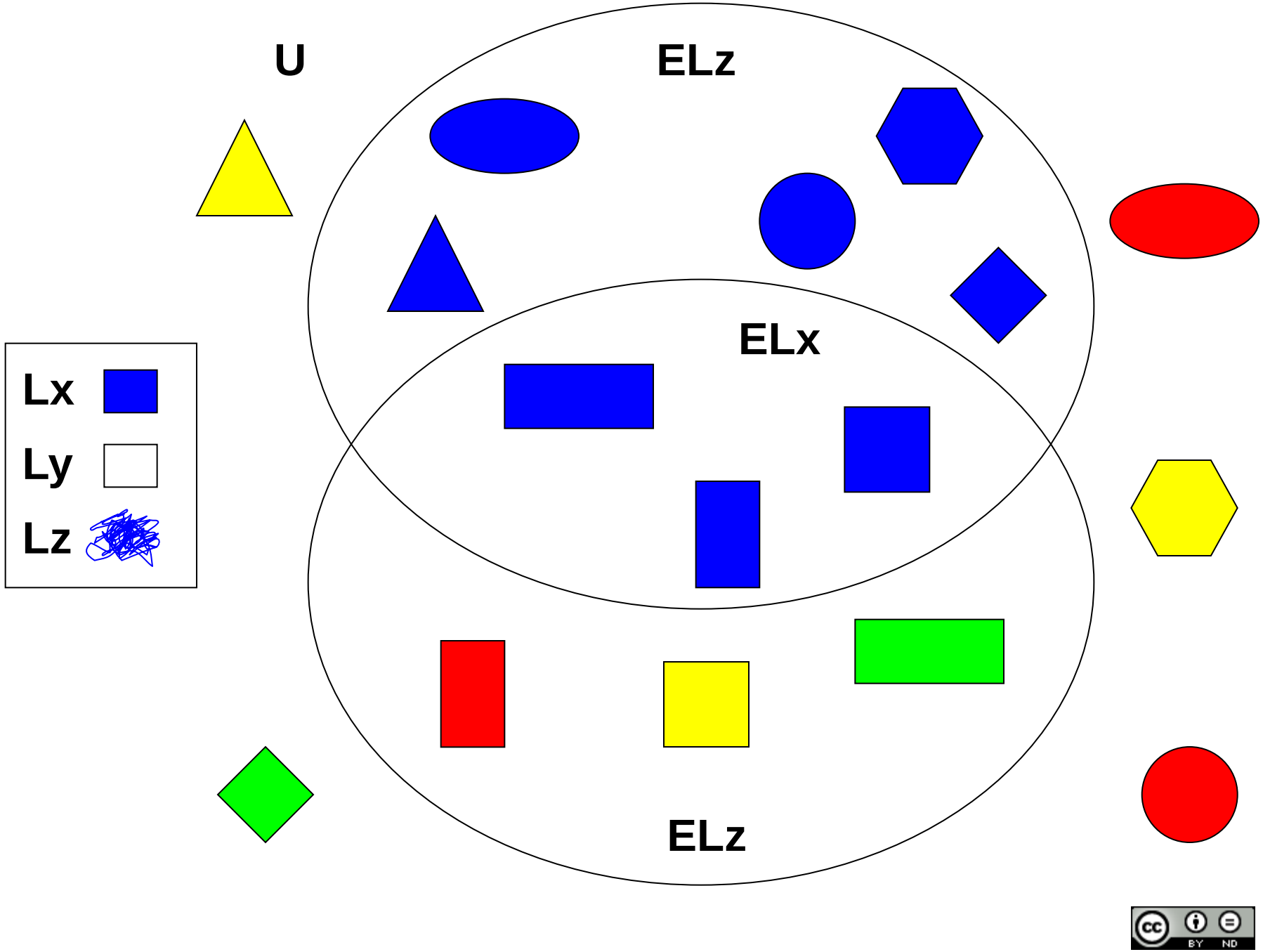

U
ELx
Lx

# Extension of a language

- There's a language specification, Lx
- Lx documents are blue rectangles
- The *extension* ELX of Lx are all (possible) blue rectangles
- All non-blues and all non-rectangles are not in ELx
- The set of all blues is a *superset* of ELx
- The set of all rectangles are a is a *superset* of ELx
- Ly and Lz are extensional superlanguages of Lx


U
ELy
ELx
Lx
Ly


U
ELz
ELx
Lx
Lz

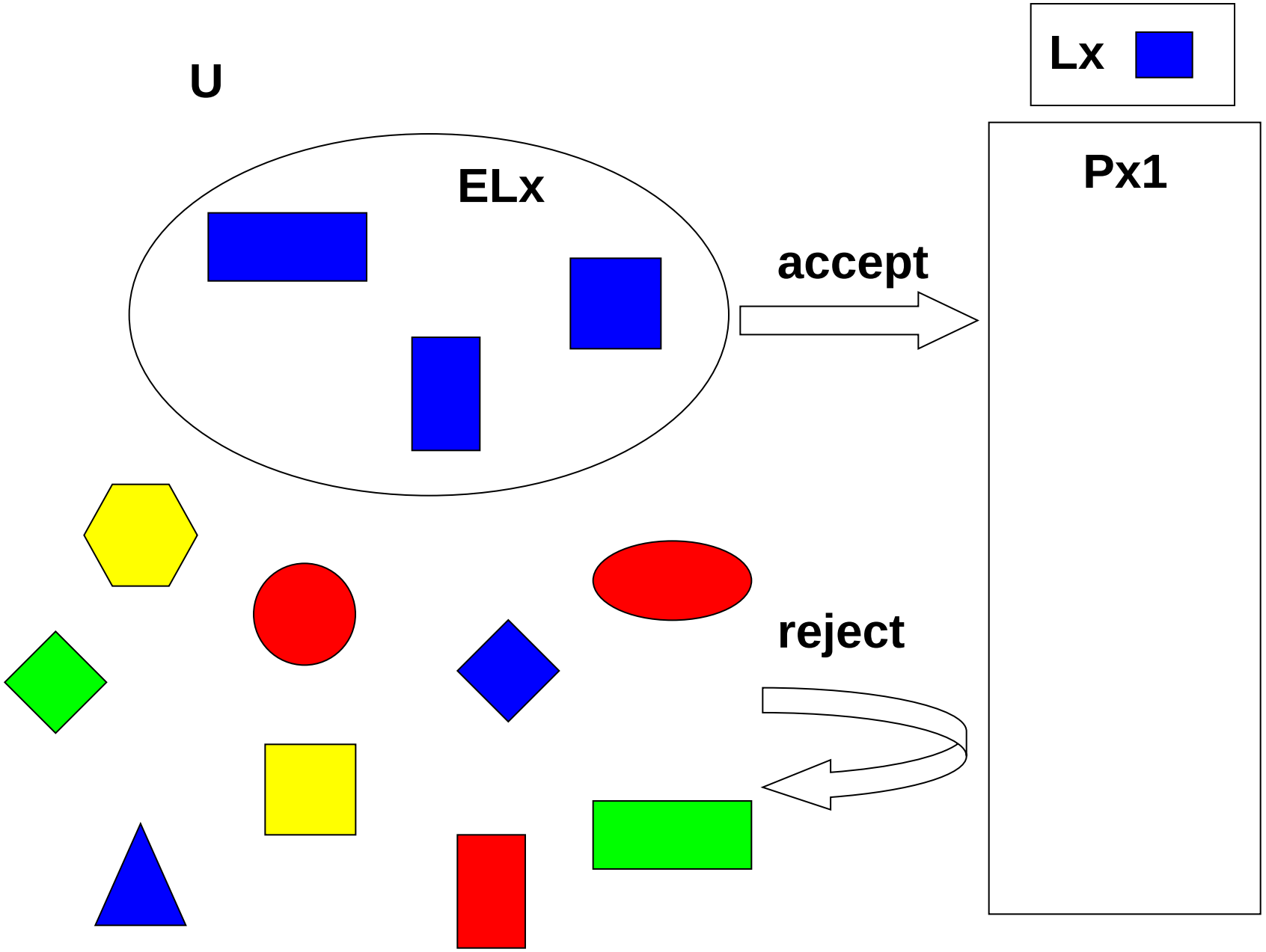


# Extensional equivalence

- L1: about traffic lights
  - <code>1</code> = Red
  - <code>2</code> = Green
- L2: about gender
  - <code>1</code> = Male
  - <code>2</code> = Female

# Extensional equivalence

- Extensional equivalence only applies to the set of documents
- If two languages have the same documents, they are extensionally equivalent, even if they are about completely different things
- L1 may be about traffic lights and L2 about gender, but if for both the set of documents is { <code>1</code>, <code>2</code> } then they are extensionally equivalent, even if their meaning is completely different



# Accept and Reject

- Applications can read and write (consume/produce, send/receive) documents
- Assume now: docs read == docs written
- Processor Px accepts all blue rectangles, and rejects all other documents
- Through accepting and rejecting Px establishes the set of documents which are syntactically conformant with Lx

# Semantics of a language

- Some languages may define little semantics
- Most often: in natural language description
- Plus: schema's, UML, other
- Use formal semantics for defining compatibility?
  - logic, OWL
  - too hard for capturing semantics of a complex language as HL7v3
  - defeats the problem

# Semantics and Behavior

- Ground notion of compatibility in application behavior?
- Language spec should enable an engineer to implement conforming application
- Application exhibits behavior
- Word processor reads doc
  - behavior: display text, properly formatted
- Medical application receiver prescription message
  - behavior: display appropriate medicine and dosage
- Languages may not define application behavior

# Message endpoint

# Behavior of message endpoint

- Message endpoint: black box
- Exhibits some behavior
- But: may be no behavior at all
  - change of address message: no immediate visible behavior
- Non-deterministic: pharmacist may ignore or change medical prescription

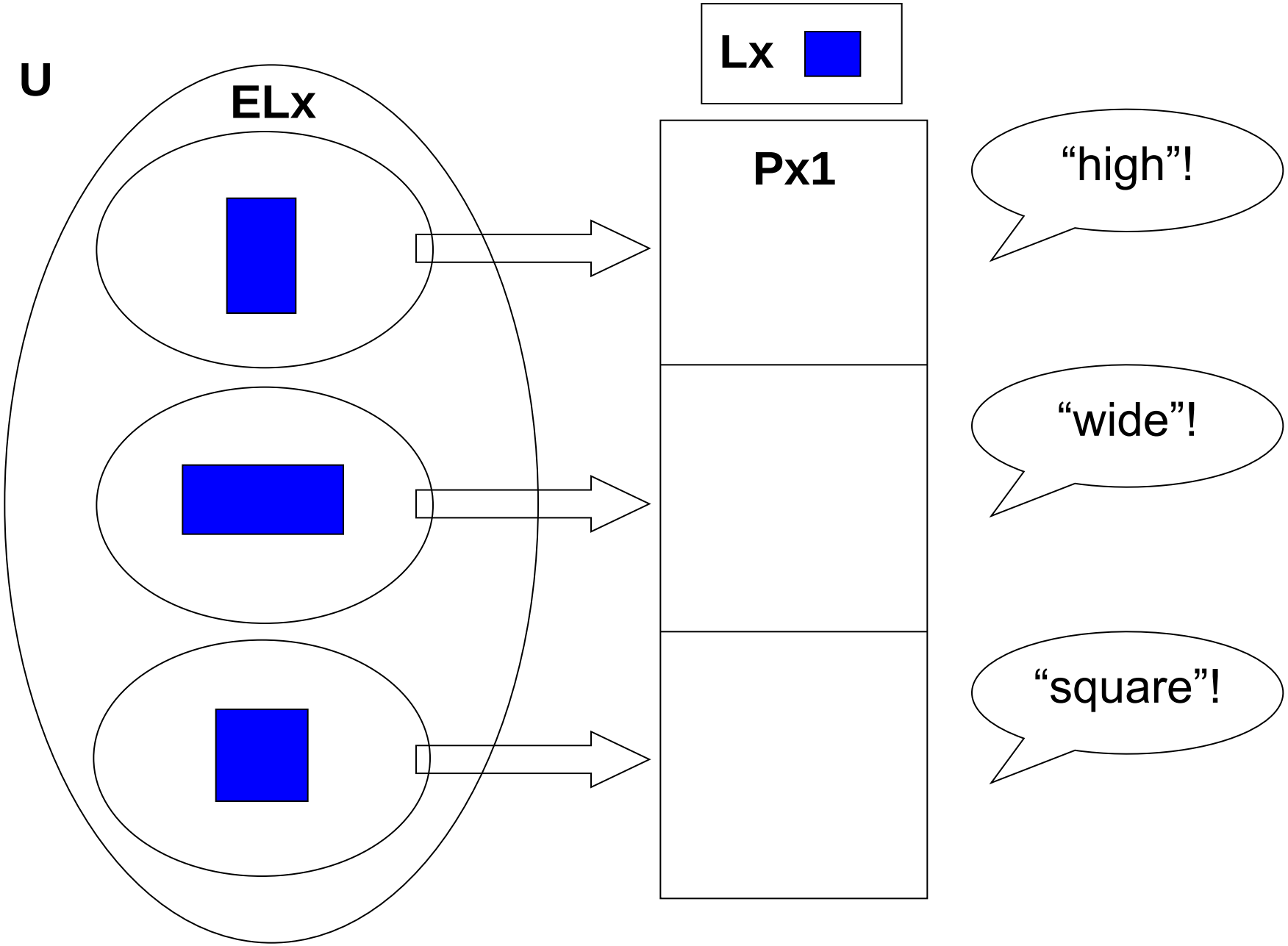
```
<code code=”27”
  codeSystem=”2.16.840.1.113883.2
  .4.4.5” />
```

"Dissolve in water"

# Message endpoint

# Behavior of message endpoint

- Endpoint is black box
- Take humans “out of the box”
- Humans judge behavior
- Assume deterministic behavior
  - start-state plus message determine end-state
- Behavior of endpoint is testable condition
- Real life behavior does not matter much
- Base semantics in behavior: perfect fit



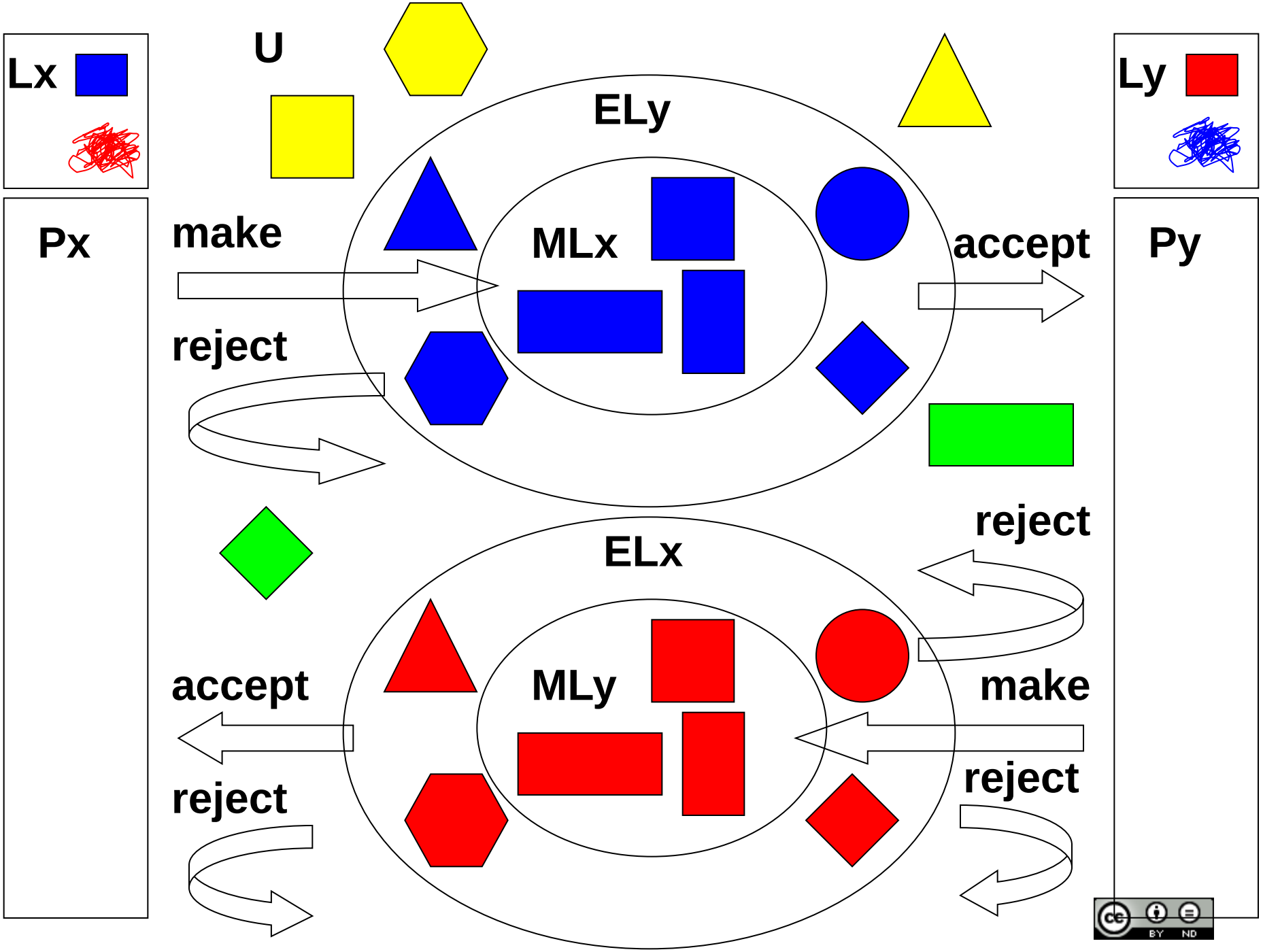
# Processors and Behavior

- For every document received, Px exhibits behavior
- Does document determine behavior?
- Reliable messaging
  - order is executed
  - duplicate is ignored
- Document determines behavior
  - assuming the same start-state
  - enough to make compatibility testable

Compatibility is a relation between two applications

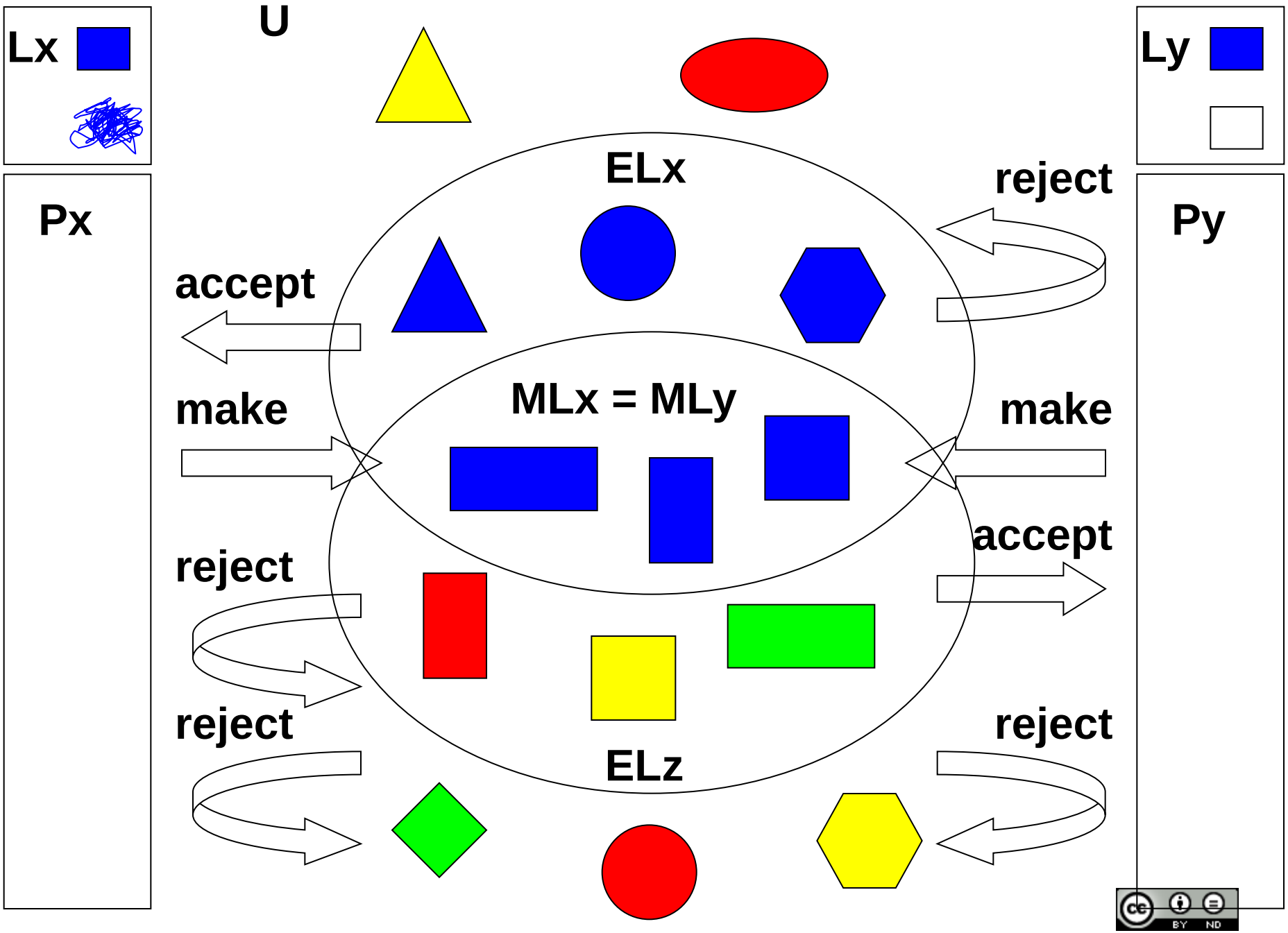
# Some Idealizations

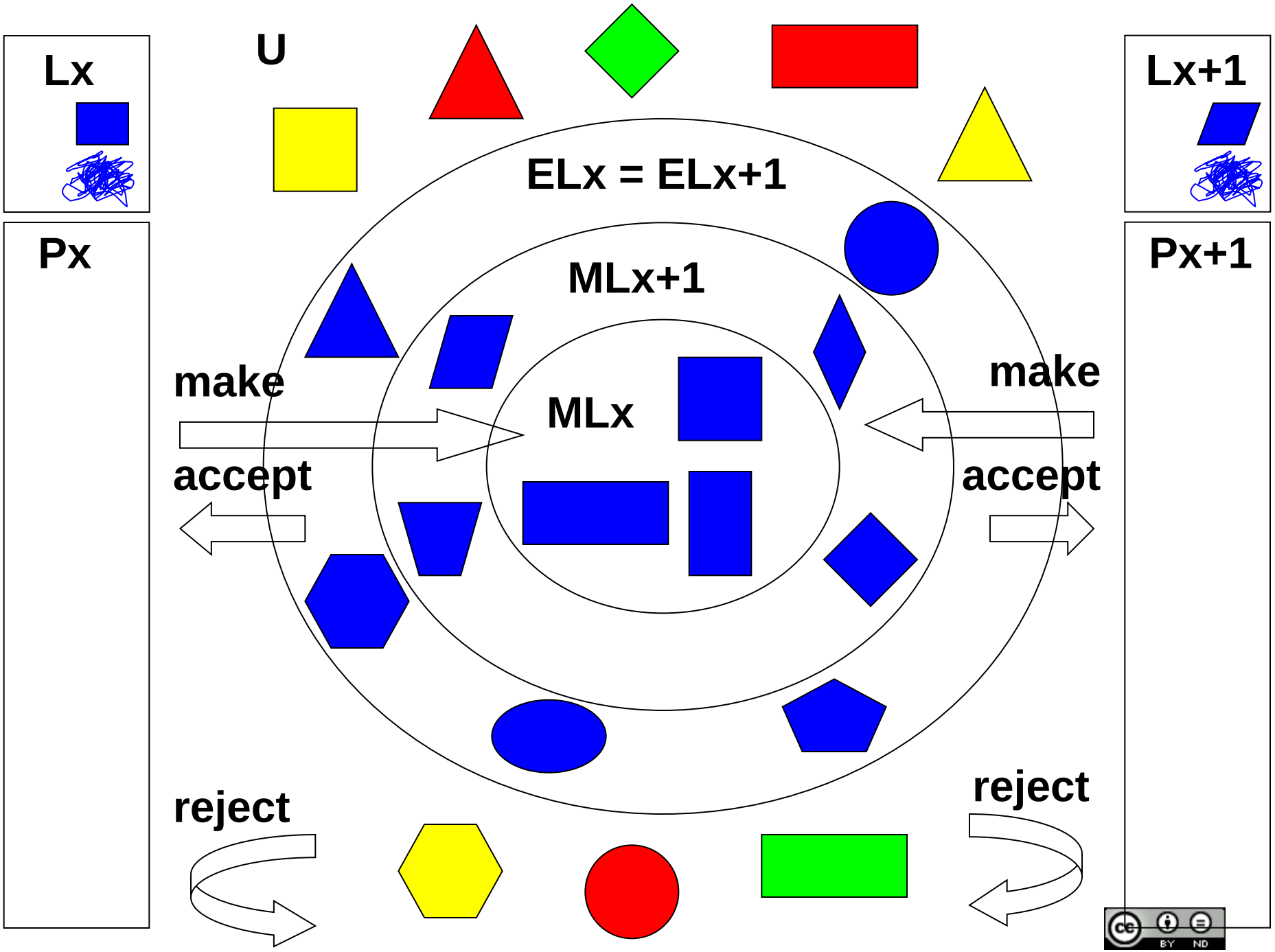
- Every language specification is flawless
- There is a perfect processor for every language
- That processor is an exemplar of the language
- Not: docs read == docs written


Lx
U
ELy
Ly
Px
make
MLx
accept
Py
reject
reject
ELx
accept
MLy
make
reject
reject

# Syntactical Compatibility

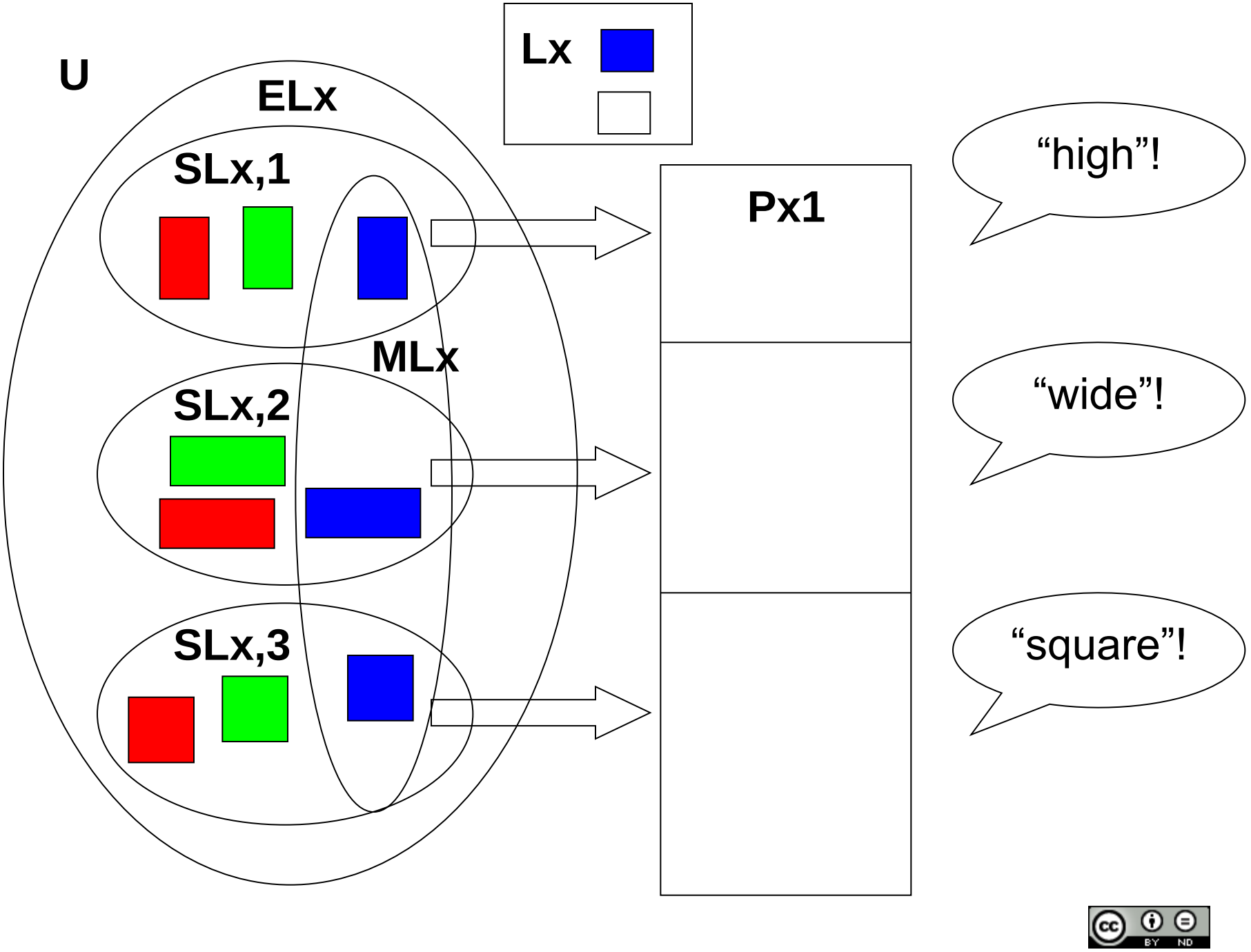
- Two language processors are syntactically compatible if they accept each other's documents
- Common case: both implement the same language
- Also: one processor makes orders, second processor accepts those but only makes order confirmations

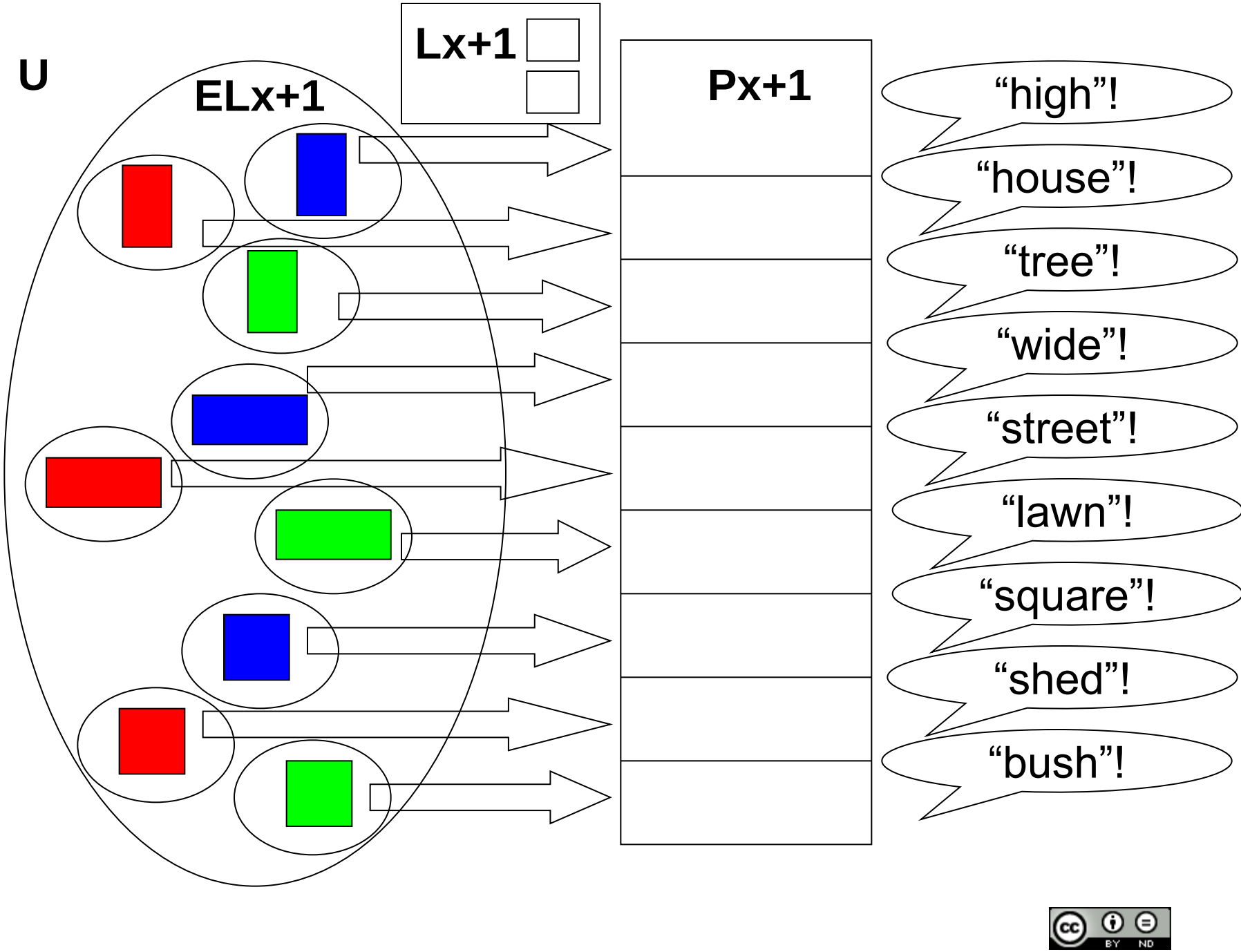

Lx
U
Ly
Px
ELx
reject
Py
accept
make
MLx = MLy
make
accept
reject
reject
reject
ELz


Lx
U
Lx+1
ELx = ELx+1
Px
MLx+1
Px+1
make
MLx
make
accept
accept
reject
reject

# Compatibility and Versioning

- Backward syntactical compatibility: Px+1 must accept all documents of Px
- Forward syntactical compatibility: Px must accept all documents of Px+1
- Key: make a processor accept more than it produces (or understands)
- Accept/reject establishes syntactical compatibility
- Accept/reject is itself behavioral (thus semantical)


U
ELx
SLx,1
MLx
SLx,2
SLx,3
Lx
Px1
“high”!
“wide”!
“square”!


U
ELx+1
Lx+1
Px+1
“high”!
“house”!
“tree”!
“wide”!
“street”!
“lawn”!
“square”!
“shed”!
“bush”!

# Semantical Equivalence Set

- For each document d which Px accepts, there is a set of documents, with which Px does the same as with d: the semantical equivalence set SLx,d of document d
- Px+1 slices SLx,d up into smaller sets, for which Px+1 adds behavior

# Semantical Equivalence Set

- For every document produced by Px, Px+1 will behave as Px expects
  - More general: Px+1 should not violate the semantics defined in Lx
- It is safe for Px to send messages to Px+1
- For documents produced by Px+1, Px+1 will ‘know’ how Px will behave
  - If Px’s behavior is not acceptable, Px+1 should not send the message
  - More general: Px+1 must ensure Px will not accept the message and exhibit the behaviour defined in Lx
- It is safe for Px to send messages to Px+1

Implementing
syntactical forward compatiblity
enables
semantic enhancement

# Conclusion

The basics of compatible versioning are:

1. make sure Px accepts more documents than it produces (or fully understands), and,
2. partition the semantical equivalence sets of Px into smaller, more refined semantical equivalence sets for Px+1 with additional (new) behavior.

# Definitions

1. Two languages are *extensionally equivalent* if they accept the same set of documents.
2. A language is an *extensional sublanguage* of a second language if all documents accepted by the first language are also accepted by the second.
3. If two processors behave the same for every document which belongs to a language Lx, the processors are *behaviourally equivalent* for Lx.
4. Two languages are *syntactically compatible* if they accept the documents produced by each other.
5. A language change is *syntactically backward compatible* if a new receiver accepts all documents produced by an older sender.
6. A language change is *syntactically forward compatible* if an old receiver accepts all documents produced by a new sender.
7. If two languages take the same documents as input, and their processors behave the same for every document, the languages are *semantically equivalent*.
8. The *semantical equivalence set* of a document d is the set of documents which make a processor behave the same as d.
9. A language is a *semantical superlanguage* of a second language if for all documents produced by the second language, processors of the first language behave the same as processors of the second language.
10. A later version of a language is *semantically backward compatible* with an earlier version if the later version is a semantical superlanguage of the earlier one (an old sender may expect a newer, but semantically backward compatible, receiver to behave as the sender intended).
11. An earlier language is *semantically forward compatible* with a later language iff the later language is a semantical sublanguage of the earlier one (this is only possible if a language loses semantics).
12. A later language *semantically extends* and earlier language if the later language introduces new behaviour for some documents accepted, but not produced by the earlier one.