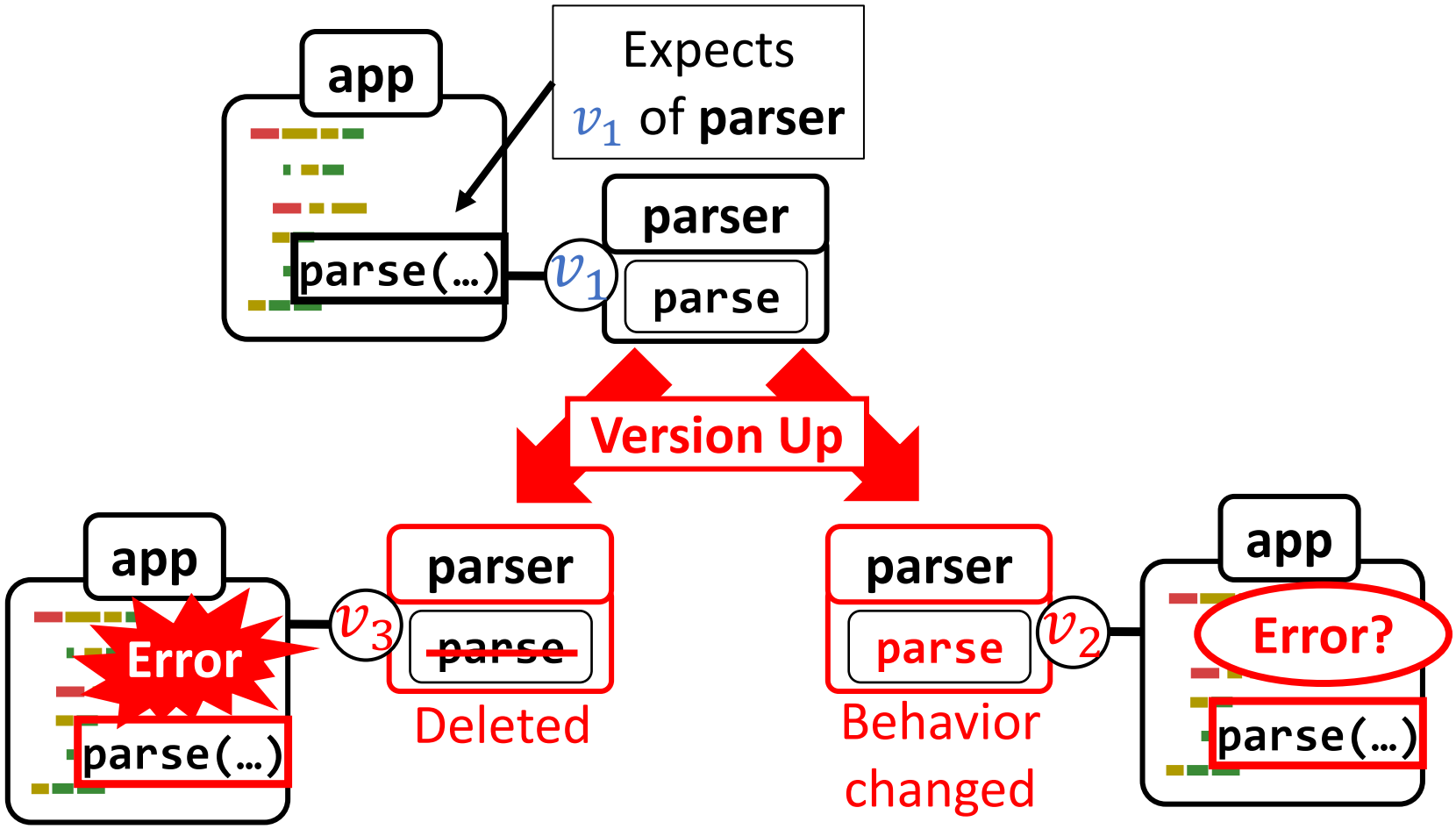
A Functional Programming Language with Versions

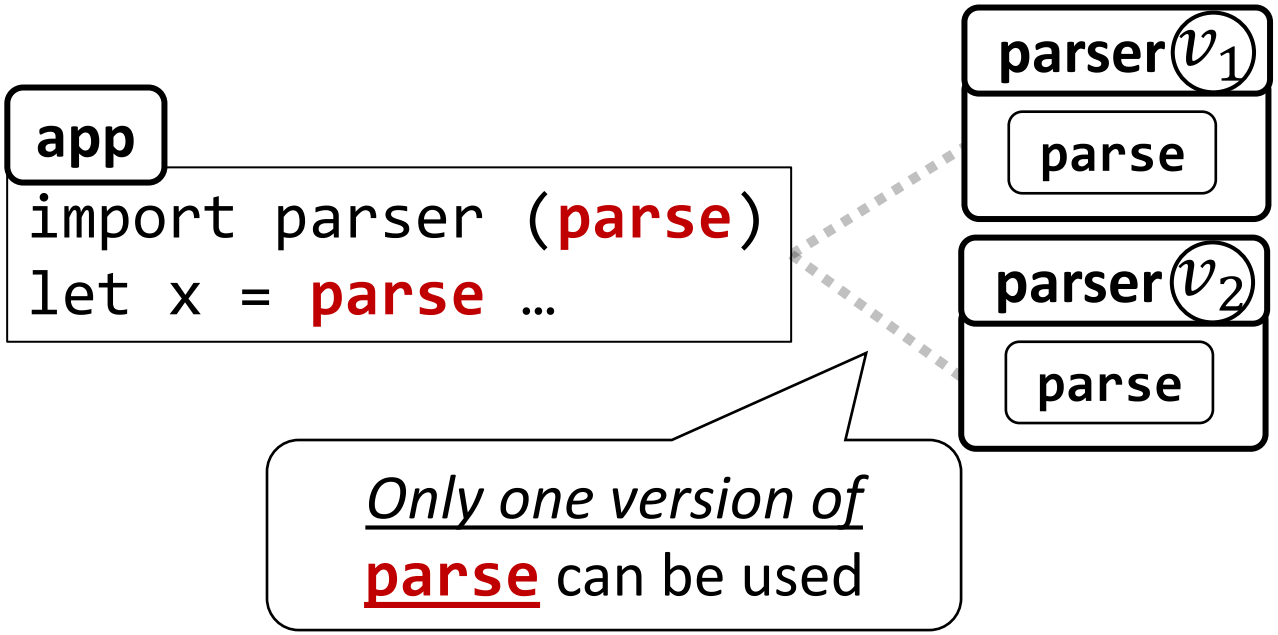
Yudai Tanabe[a)] Luthfan Anshar Lubis[a)]
Tomoyuki Aotani[b)] Hidehiko Masuhara[a)]

(a) Tokyo Institute of Technology (b) Mamezou

Version Update May Break Software

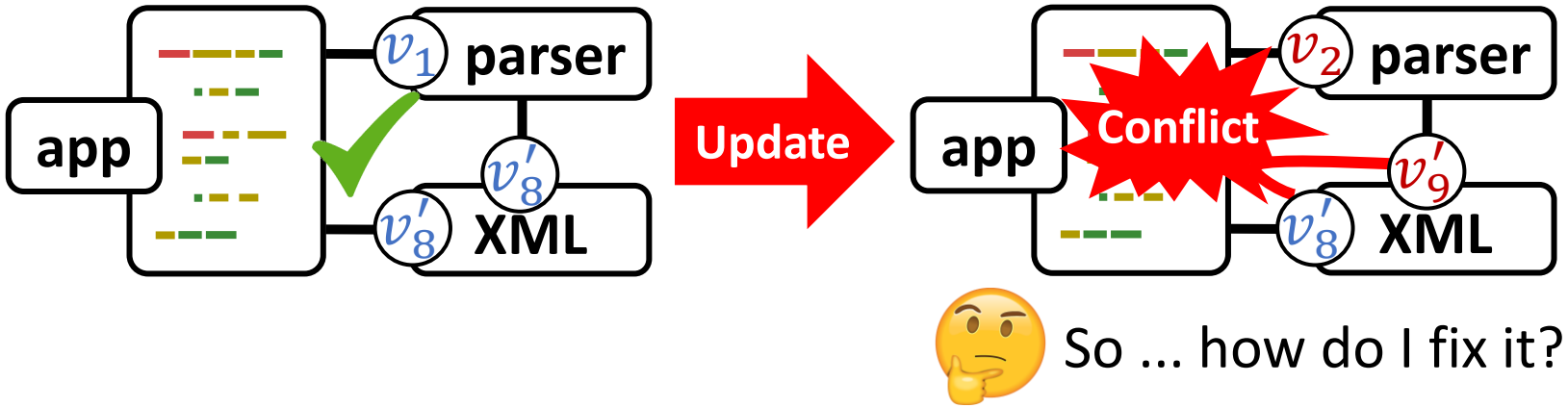


Implicit Assumption: One-version-at-a-time



Dependency Hell

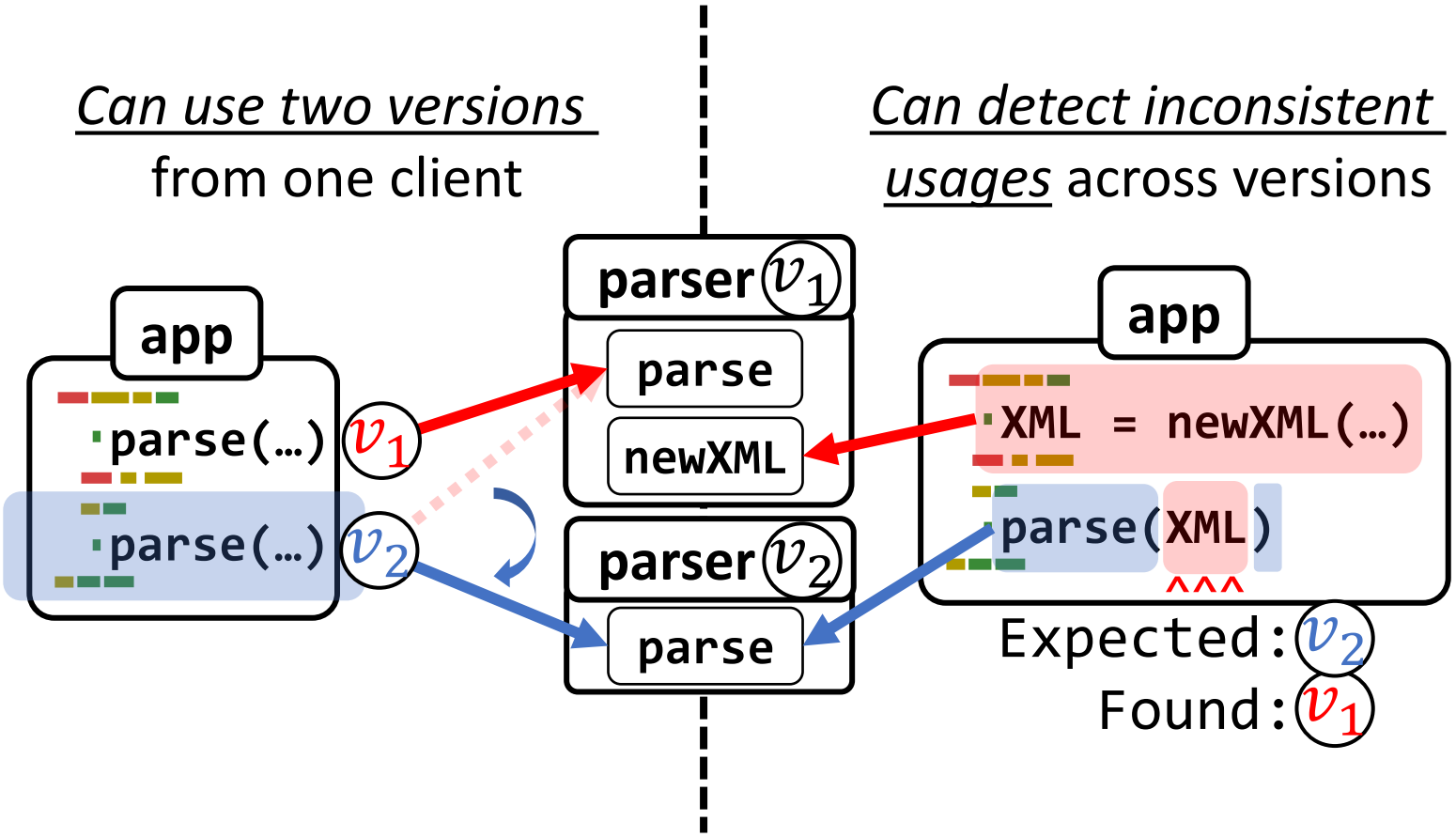
- Difficult to resolve indirect dependency conflicts



- Leads to version locking[Preston-Werner '13]

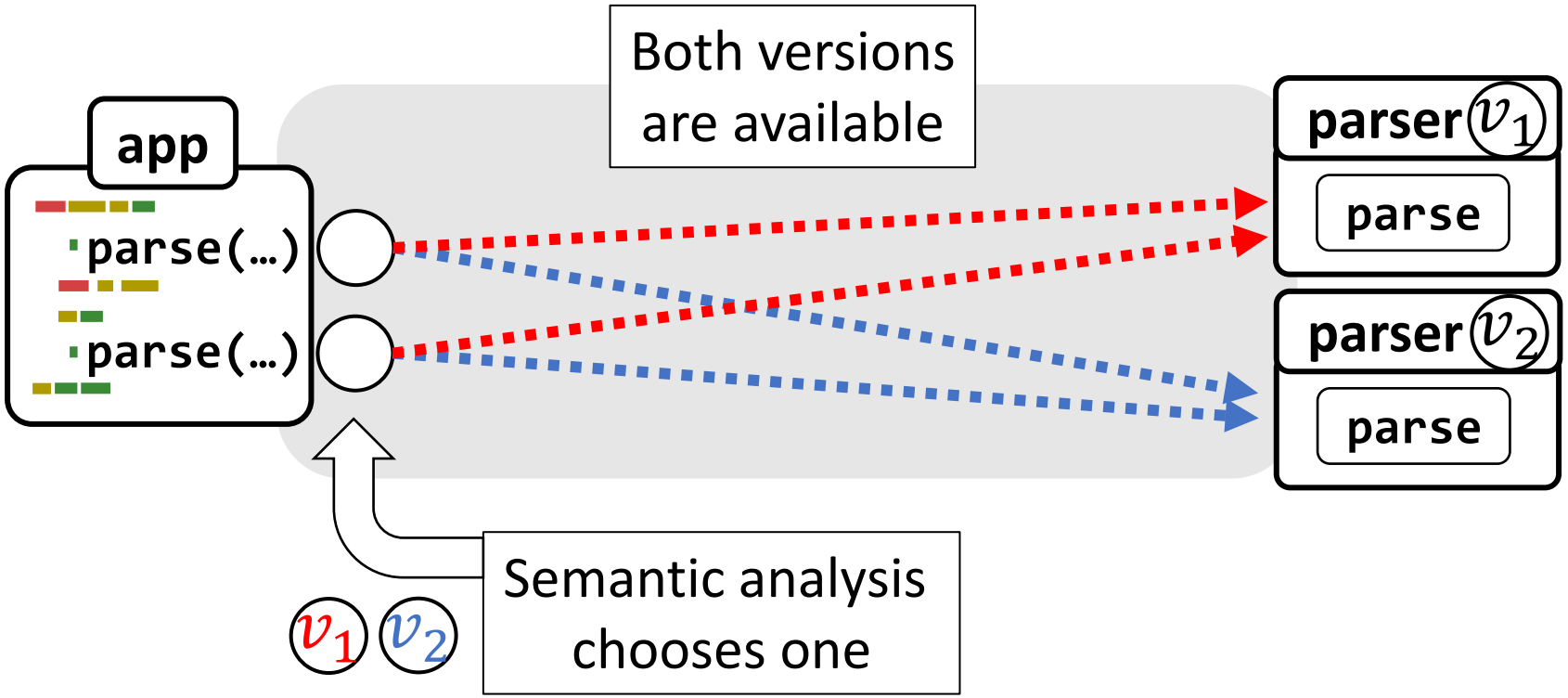
Developers are reluctant to update[Bavota '15]

Programming Language with Versions



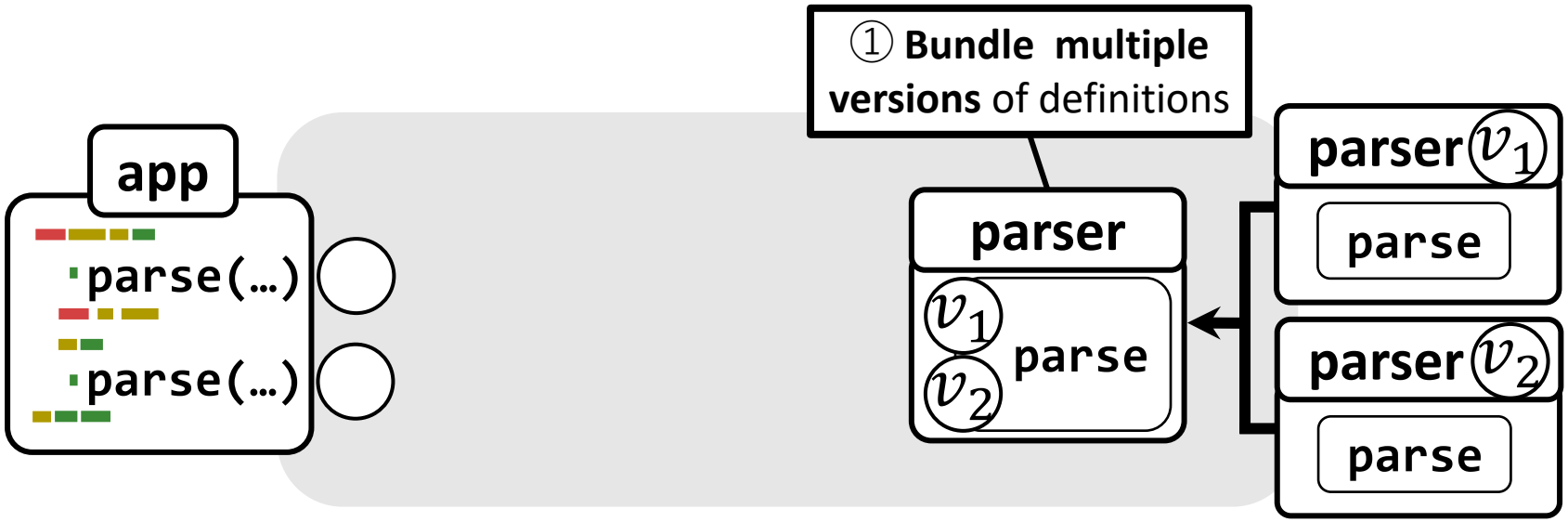
Language Overview

- **Idea**: *Multiple versions in one value*



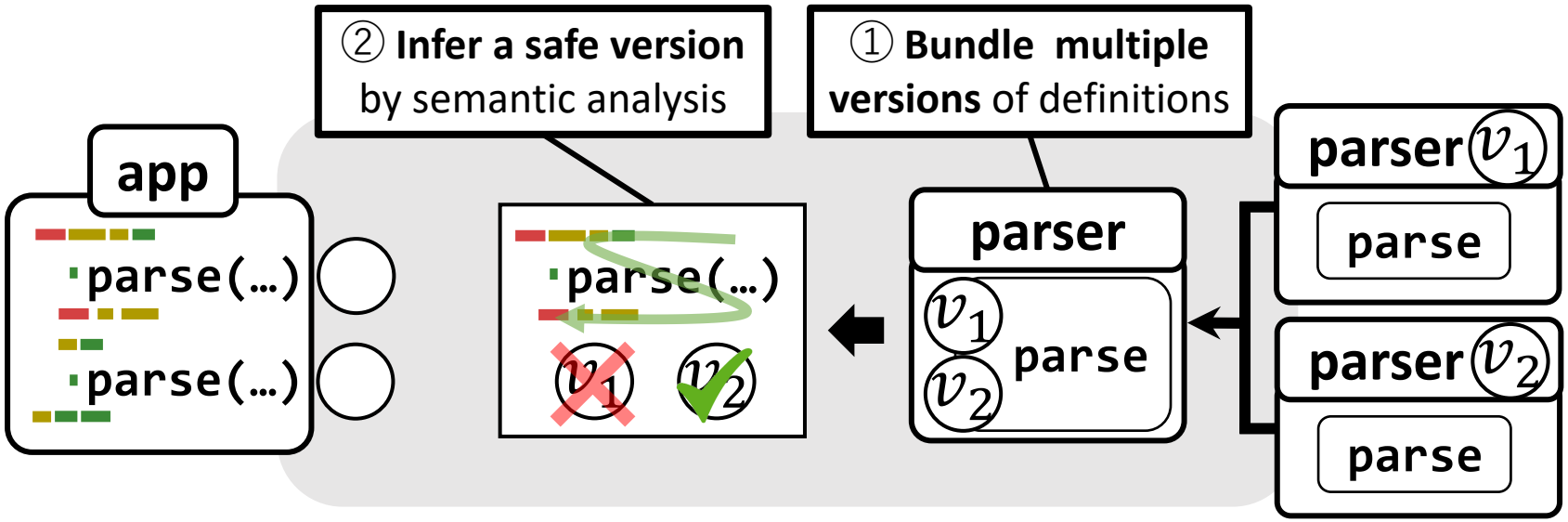
Language Overview

- **Idea**: *Multiple versions in one value*



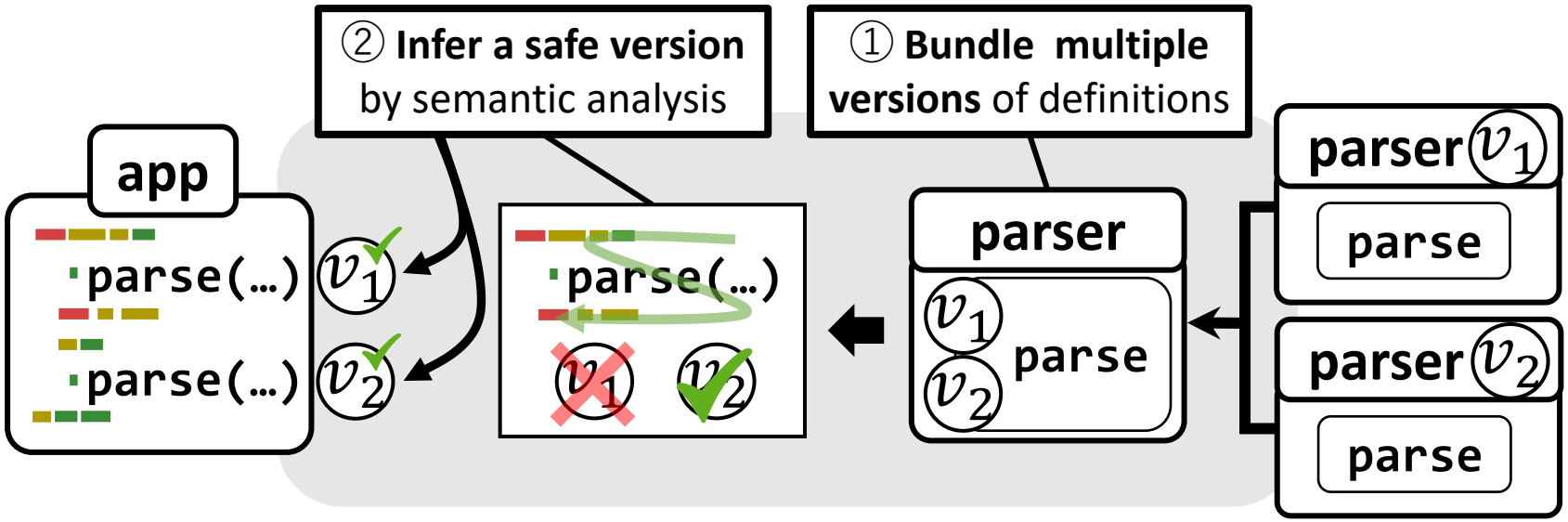
Language Overview

- **Idea**: *Multiple versions in one value*



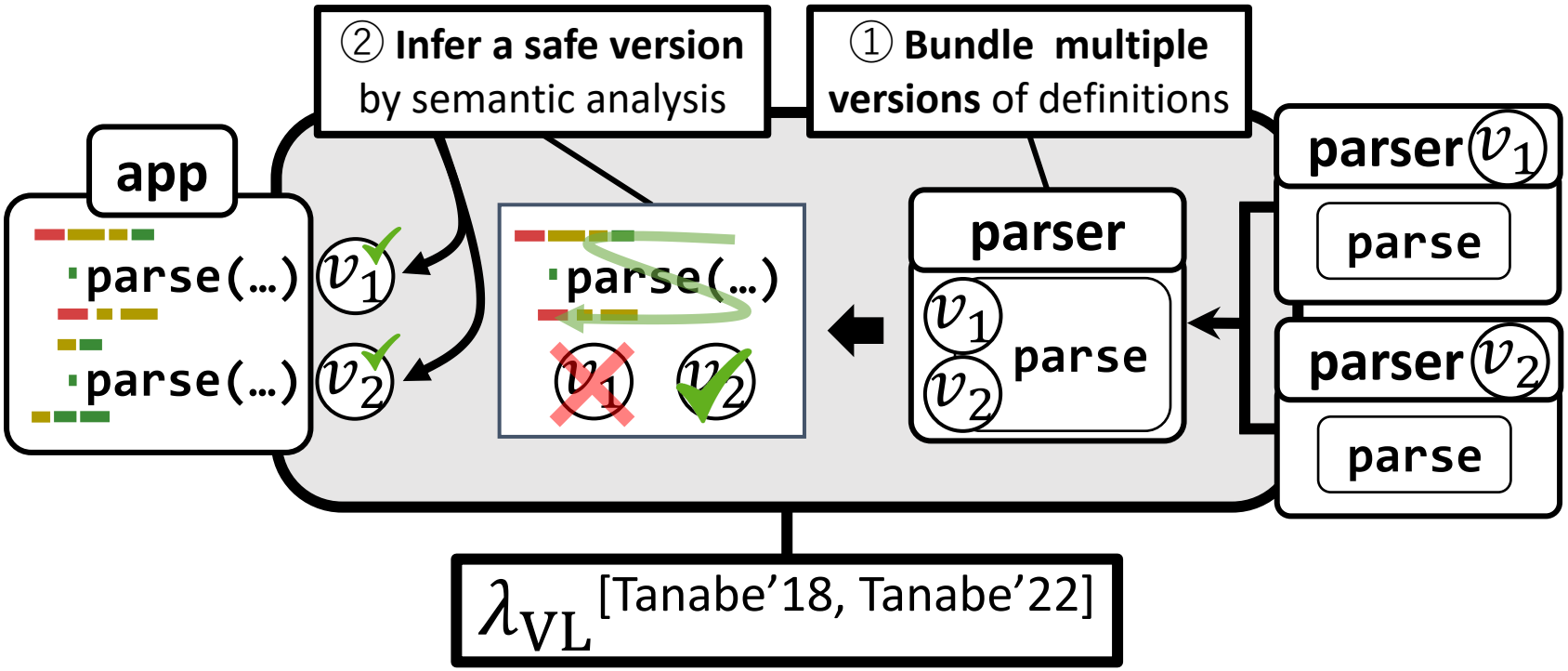
Language Overview

- **Idea**: *Multiple versions in one value*



Language Overview

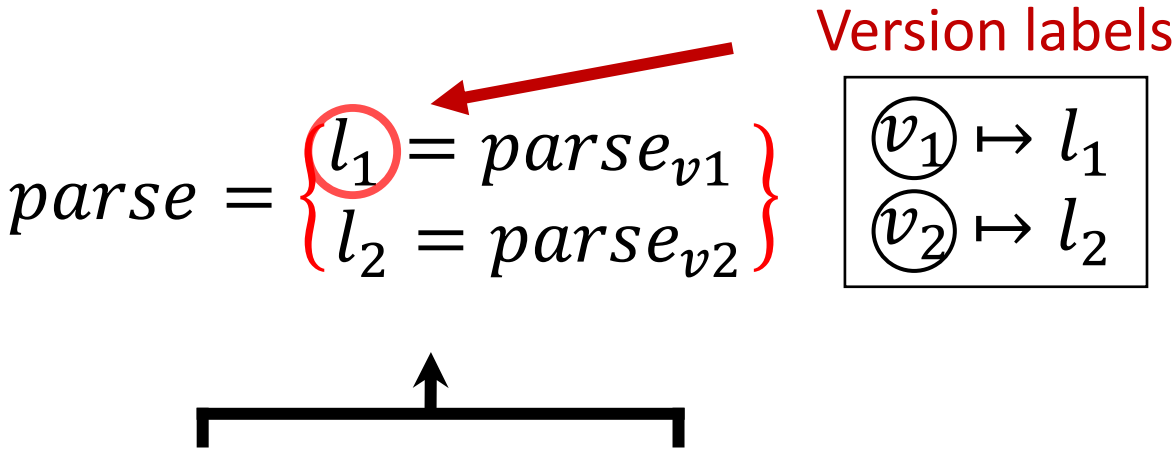
- **Idea**: *Multiple versions in one value*



λ_{VL}: Versioned Values

- **Versioned records**:

  Multiple versions in one value like ***a record***



$parse_{v1}$

```
= \xml ->
  let elem = pop …
  in …
```

$parse_{v2}$

```
= \xml ->
  let elem = pop …
  in …
```

λ_{VL} Program Construction

- By ***function application of versioned values***

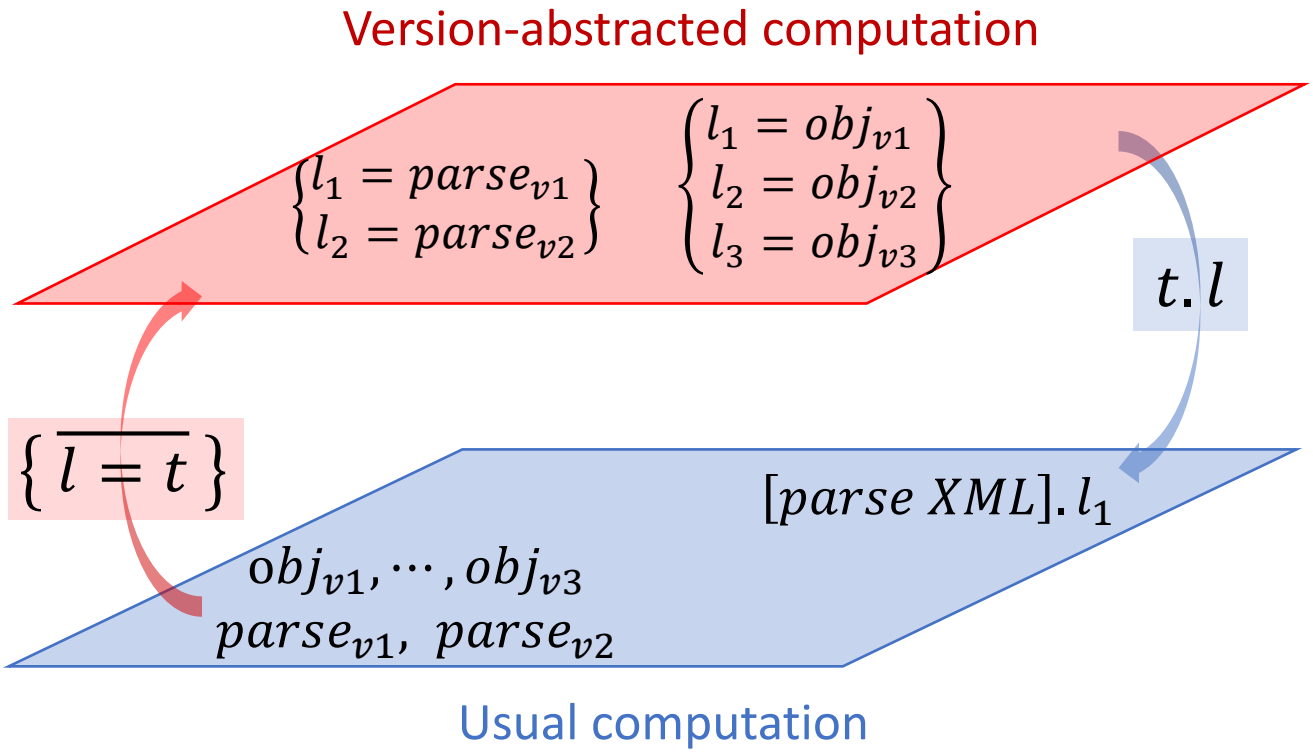
$$\boxed{\begin{array}{l}\text{let } [f] = parse \text{ in} \\ \text{let } [x] = XML \text{ in} \\ [f\ x]\end{array}}_{\lambda_{\mathrm{VL}}} \approx \left\{\begin{array}{l} l_1 = parse_{v1}\ obj_{v1} \\ l_2 = parse_{v2}\ \ obj_{v2} \\ \_ \end{array}\right\}$$

$$\underset{parse}{\left\{\begin{array}{l} l_1 = parse_{v1} \\ l_2 = parse_{v2} \\ \_ \end{array}\right\}} \quad \underset{XML}{\left\{\begin{array}{l} l_1 = obj_{v1} \\ l_2 = obj_{v2} \\ l_3 = obj_{v3} \end{array}\right\}}$$

Core Calculus

λ_{VL} Semantics

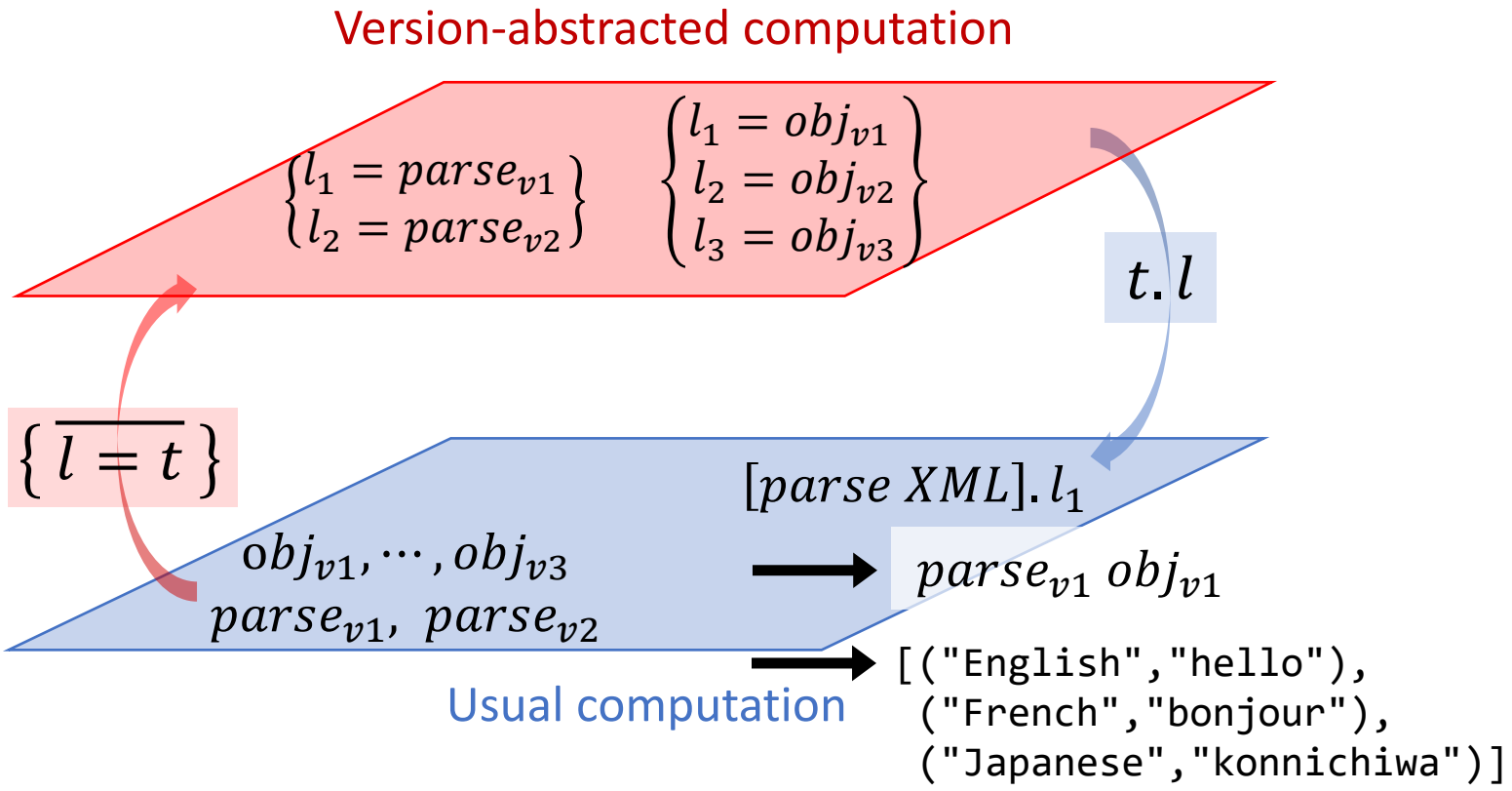
- **Extraction $t.\,l$:**
  Specify l to run the program in a specific version



Core Calculus

λ_{VL} Semantics

- **Extraction $t.\,l$:**
  Specify l to run the program in a specific version



Inconsistency to Detect in $\lambda_{\mathbf{VL}}$

- ***No extractable version***

$$\text{let } [f] = parse \text{ in}\\ \text{let } [x] = XML \text{ in}\\ [f\ x].l_1 \quad \approx \quad \begin{Bmatrix} l_1 = parse_{v1} \\ l_2 = parse_{v2} \end{Bmatrix} \begin{Bmatrix} l_1 = obj_1 \\ l_2 = obj_2 \\ l_3 = obj_3 \end{Bmatrix}$$

$$\text{let } [f] = parse \text{ in}\\ \text{let } [x] = XML \text{ in}\\ [f\ x].l_3 \quad \approx \quad \begin{Bmatrix} l_1 = parse_{v1} \\ l_2 = parse_{v2} \end{Bmatrix} \begin{Bmatrix} l_1 = obj_1 \\ l_2 = obj_2 \\ l_3 = obj_3 \end{Bmatrix}$$

λ_{VL} Type System: Versions as Resources

- Types are ***tagged with available version labels***

$$parse : \square_{\{\boldsymbol{l_1}, \boldsymbol{l_2}\}}(\mathrm{XML} \to A)$$

$$parse = \left\{ \begin{matrix} \boldsymbol{l_1} = parse_{v1} \\ \boldsymbol{l_2} = parse_{v2} \end{matrix} \right\}$$

$$XML : \square_{\{\boldsymbol{l_1}, \boldsymbol{l_2}, \boldsymbol{l_3}\}}\mathrm{XML}$$

$$XML = \left\{ \begin{matrix} \boldsymbol{l_1} = obj_1 \\ \boldsymbol{l_2} = obj_2 \\ \boldsymbol{l_3} = obj_3 \end{matrix} \right\}$$

Core Calculus

λ_{VL} Type System

- ***Collect consistent versions*** along with usual type checking

$$parse : \square_{\{l_1, l_2\}}(\mathrm{XML} \to A) \qquad XML : \square_{\{l_1, l_2, l_3\}}\mathrm{XML}$$

$$\begin{array}{l} \mathrm{let}\,[f] = parse\ \mathrm{in} \\ \mathrm{let}\,[x] = XML\ \mathrm{in} \\ [f\ x] \end{array} \quad : \square_{\underline{\{\boldsymbol{l_1}, \boldsymbol{l_2}\}}} A$$

$$\| $$

$$\{l_1, l_2, l_3\} \cap \{l_1, l_2\}$$

Core Calculus

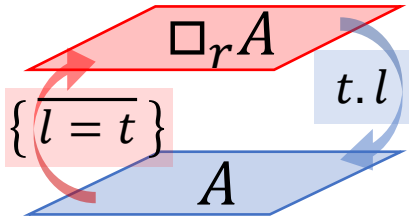
λ_{VL} Type System

- ***Inspect the consistency*** of specified version by an extraction

$$parse : \square_{\{l_1, l_2\}}(\mathrm{XML} \to A) \qquad XML : \square_{\{l_1, l_2, l_3\}}\mathrm{XML}$$

$$\begin{array}{l} \mathrm{let}\ [f] = parse\ \mathrm{in} \\ \mathrm{let}\ [x] = XML\ \mathrm{in} \\ [f\ x].\, l_1 \end{array} \quad : A$$

because $l_1 \in \{l_1, l_2, l_3\} \cap \{l_1, l_2\}$



Core Calculus

λ_{VL} Type System

- ***Inspect the consistency*** of specified version by an extraction

$$parse : \square_{\{l_1, l_2\}}(\mathrm{XML} \to A) \qquad XML : \square_{\{l_1, l_2, l_3\}}\mathrm{XML}$$

$$\begin{array}{l} \mathrm{let}\ [f] = parse\ \mathrm{in} \\ \mathrm{let}\ [x] = XML\ \mathrm{in} \\ [f\ x].\, l_3 \end{array} :$$

[ERROR]
Expected l_3,
but got l_1, l_2

because $l_3 \notin \{l_1, l_2, l_3\} \cap \{l_1, l_2\}$

Formalization with Coeffect Calculus

$$parse\colon [\mathrm{XML} \to \ldots]_{\{\boldsymbol{l_1}, l_2\}}$$

$$XML\colon [\mathrm{XML}]_{\{\boldsymbol{l_1}, l_2, l_3\}} \vdash [parse\ XML] : \square_{\{\boldsymbol{l_1}, l_2\}} A$$

All subterms hold the version label as ***a resource*** $\Leftrightarrow$ The program is available in the version

Coeffect calculus:
ℓRPCF[Brunel'14], GrMini[Orchard'19]

Coeffect Calculus

Coeffect calculus: ℓRPCF[Brunel'14], GrMini[Orchard'19]

$t ::= n \mid x \mid t_1 t_2 \mid \lambda x.t \mid$

$[t] \mid \mathrm{let}\,[x] = t_1 \,\mathrm{in}\, t_2 \mid$

Terms from coeffect calculus

$A ::= \mathrm{Int} \mid A \to A \mid \square_r A$

$\Gamma ::= \emptyset \mid \Gamma, x : A \mid \Gamma, x : [A]_r$

$r \in (\mathcal{R}, +, 0, \times, 1)$

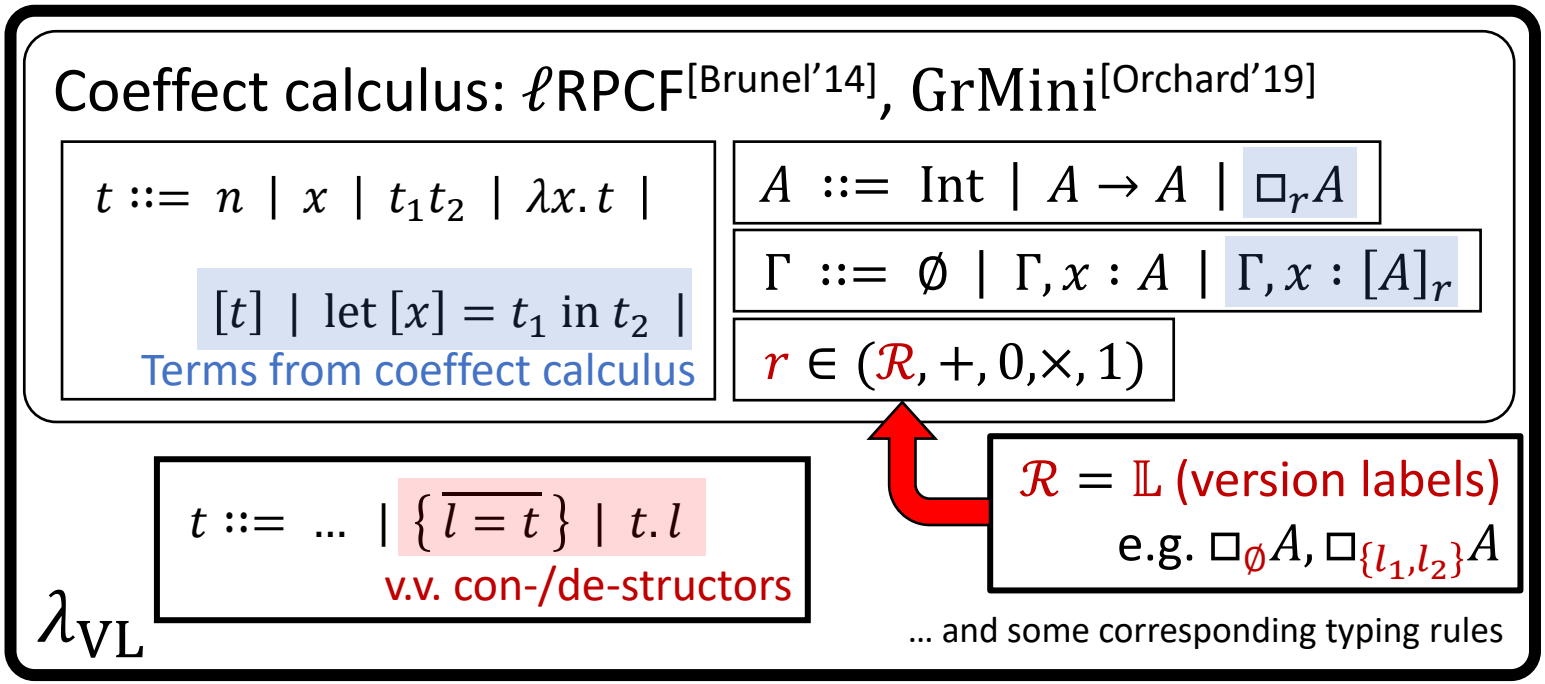
$\mathcal{R}$-parameterized type systems

$\mathcal{R} = \{\mathrm{Irrelevant}, \mathrm{Private}, \mathrm{Public}\}$
(security level[Orchard'19])

$\mathcal{R} = \mathbb{N}$ (exact usage[Petricek'14])
e.g. $\square_0 A, \square_2 A$

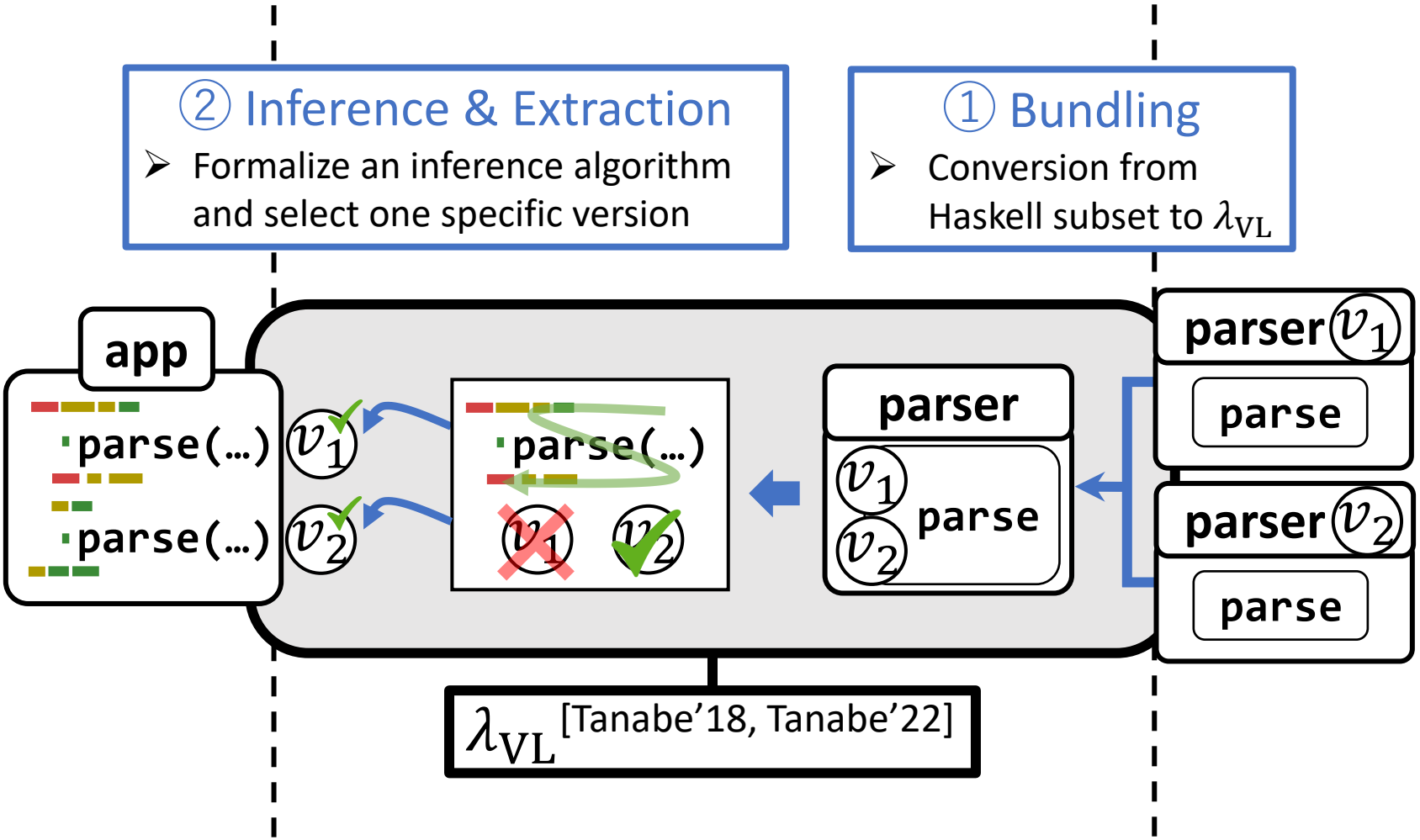
λ_{VL} Type System

- ℓRPCF-extension by ***versions as resources***



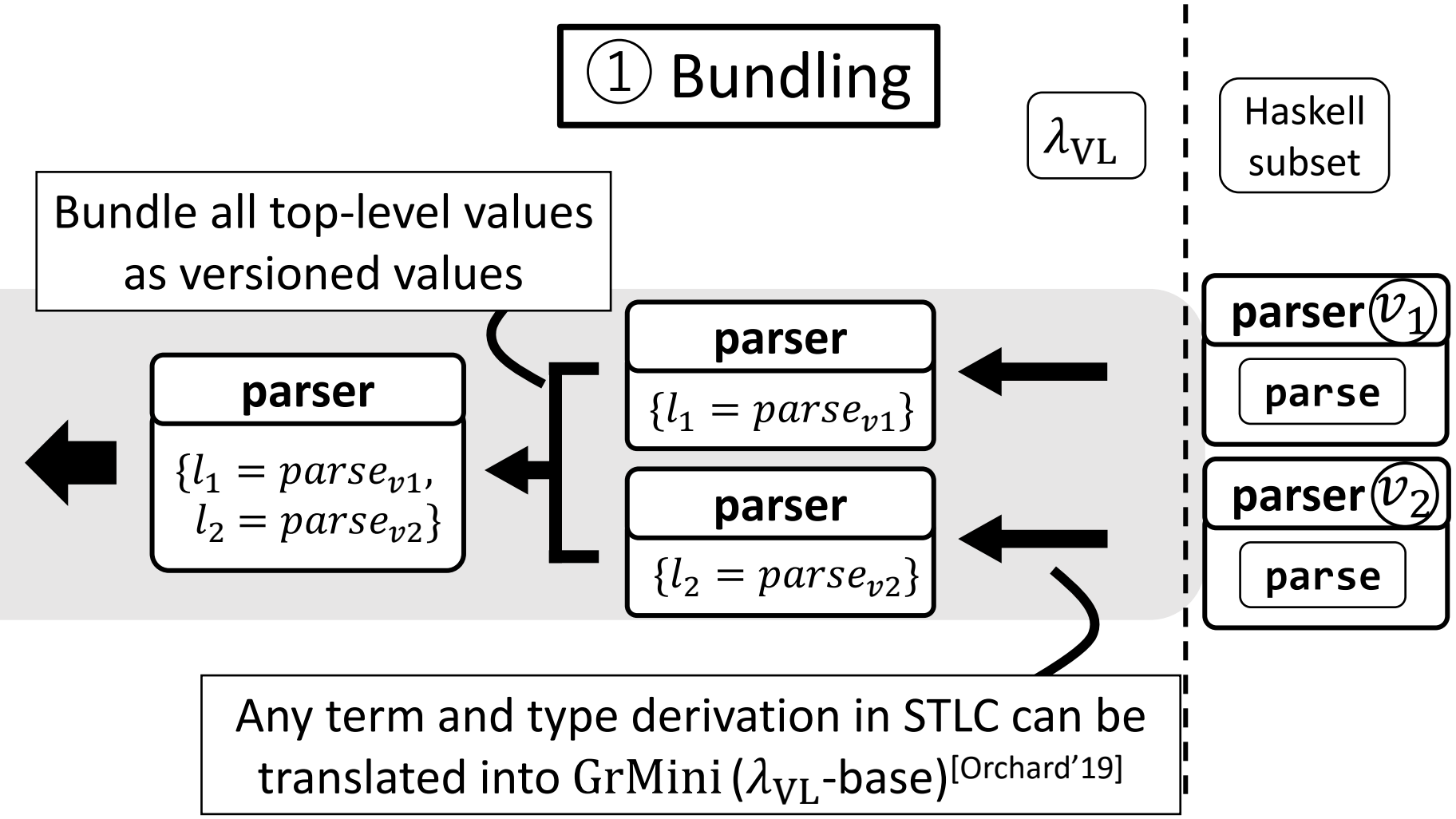
Future Work

Implementation



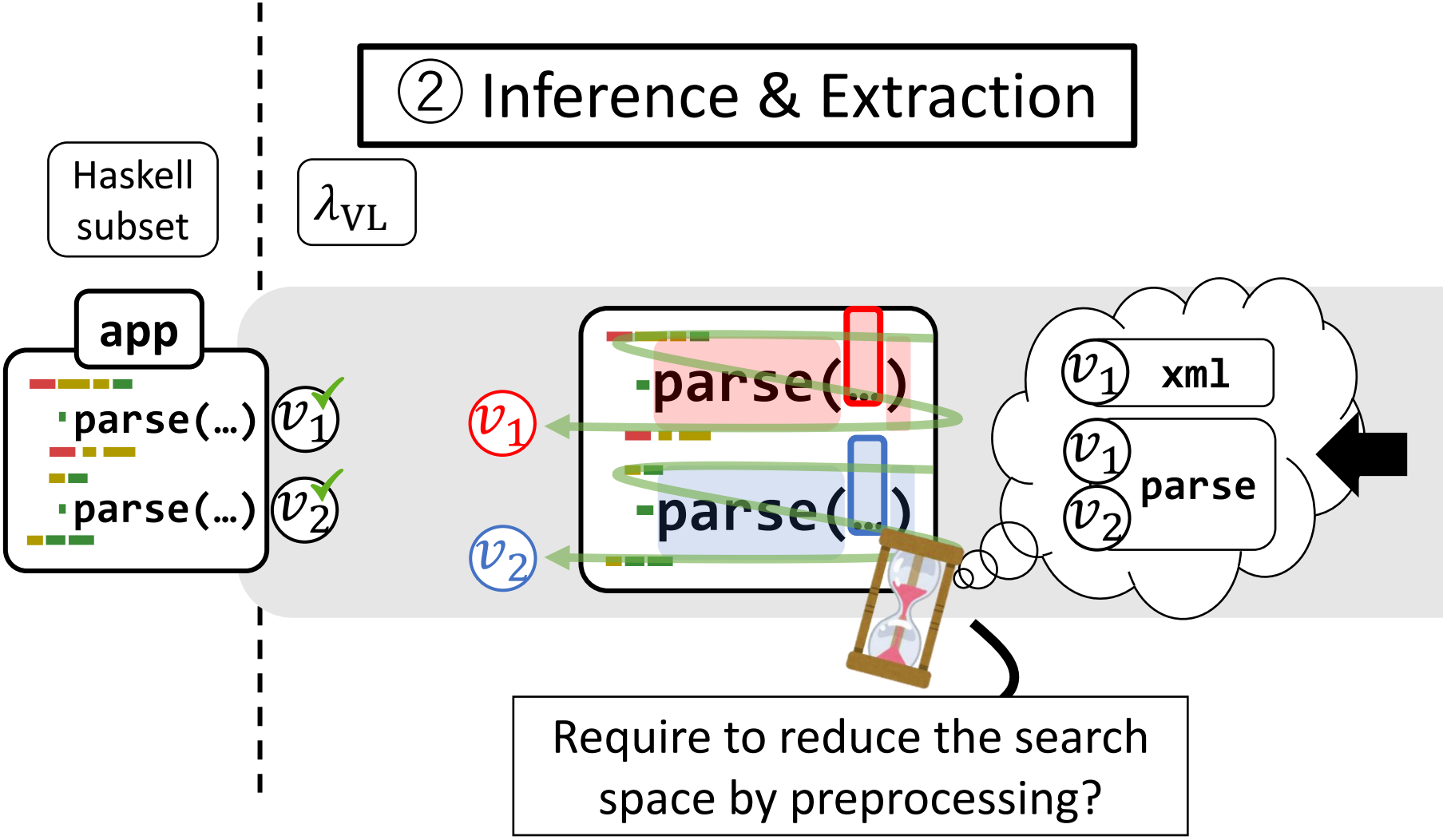
Future Work

Implementation



Future Work

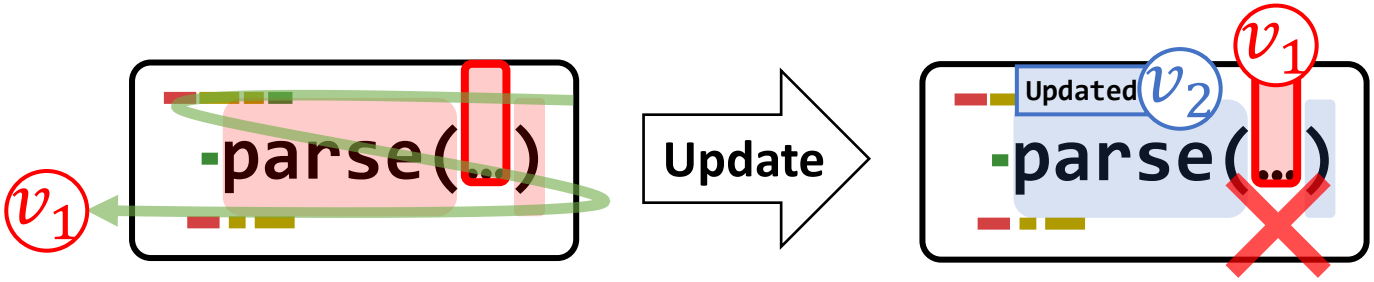
Implementation



Adapting Old Version into New Version

- Incorporate compatible updates

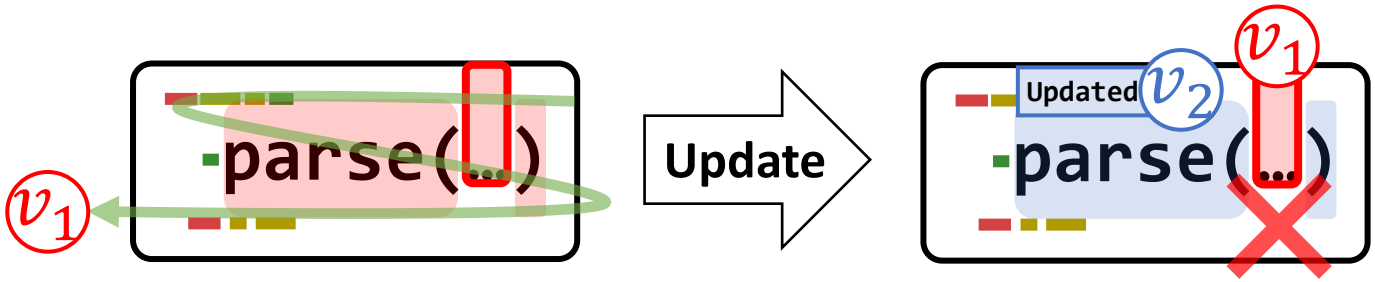
… but actually $parse_{v1}$ and $parse_{v2}$ are compatible



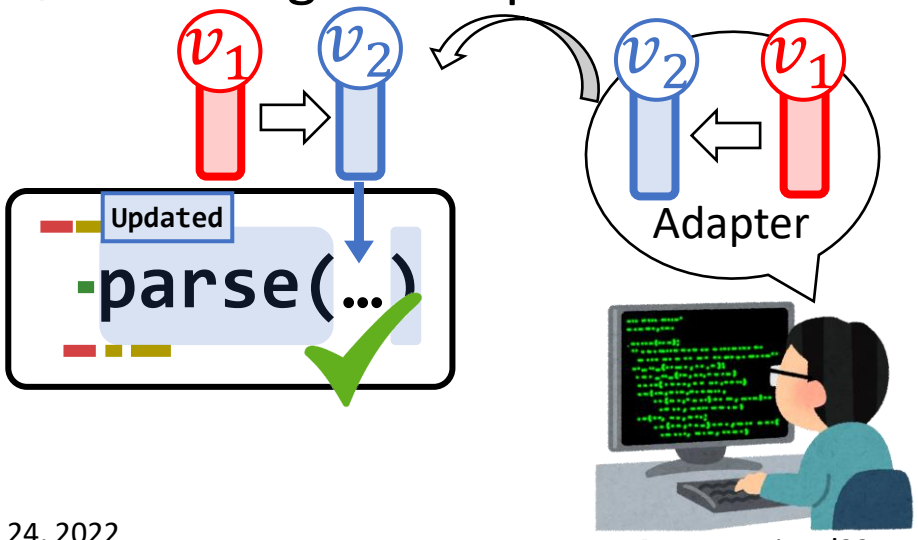
Adapting Old Version into New Version

- Incorporate compatible updates

… but actually $parse_{v1}$ and $parse_{v2}$ are compatible



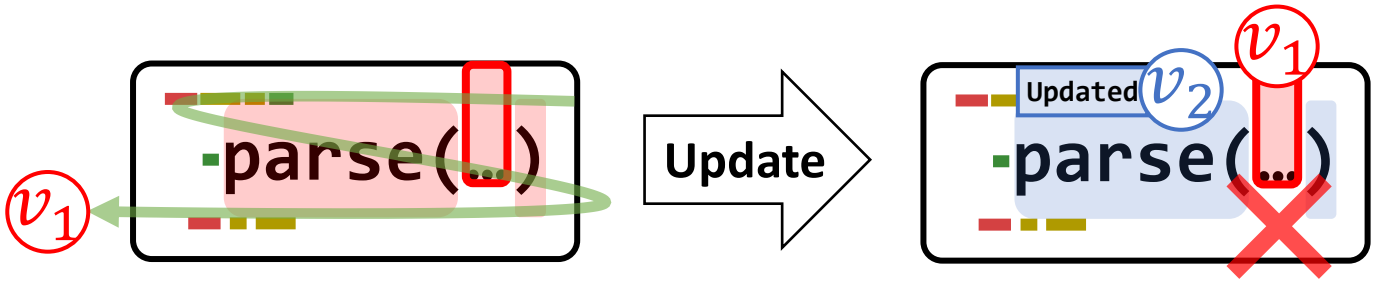
① Inserting an adapter



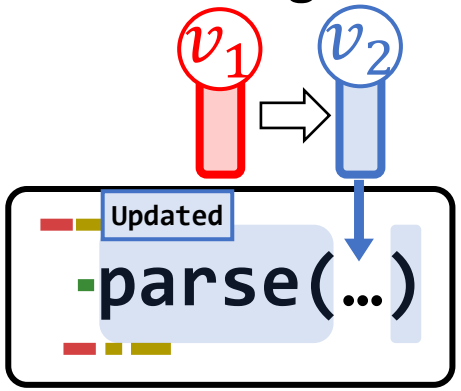
Adapting Old Version into New Version

- Incorporate compatible updates

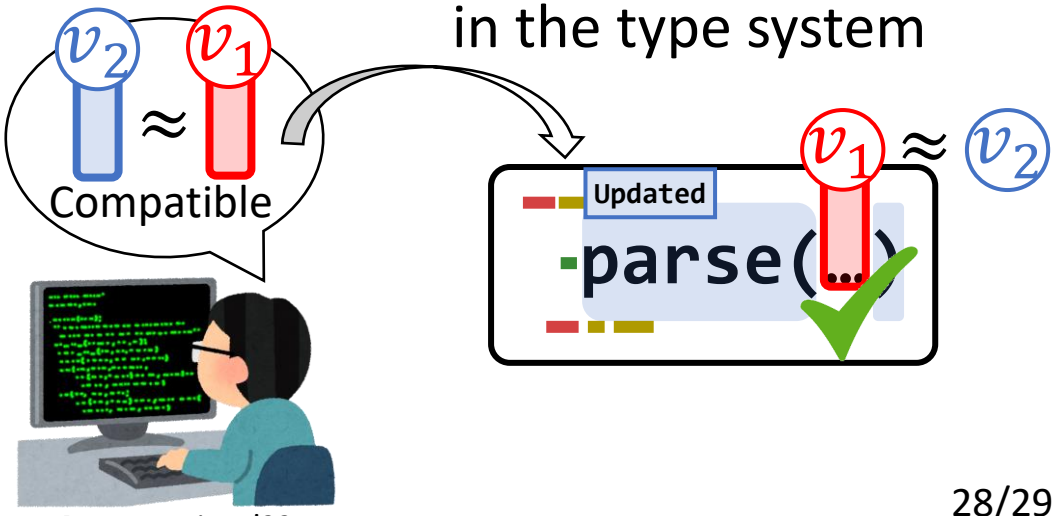
… but actually $parse_{v1}$ and $parse_{v2}$ are compatible



① Inserting an adapter



② Handling compatibility in the type system



<Programming>'22

Summary

Problem

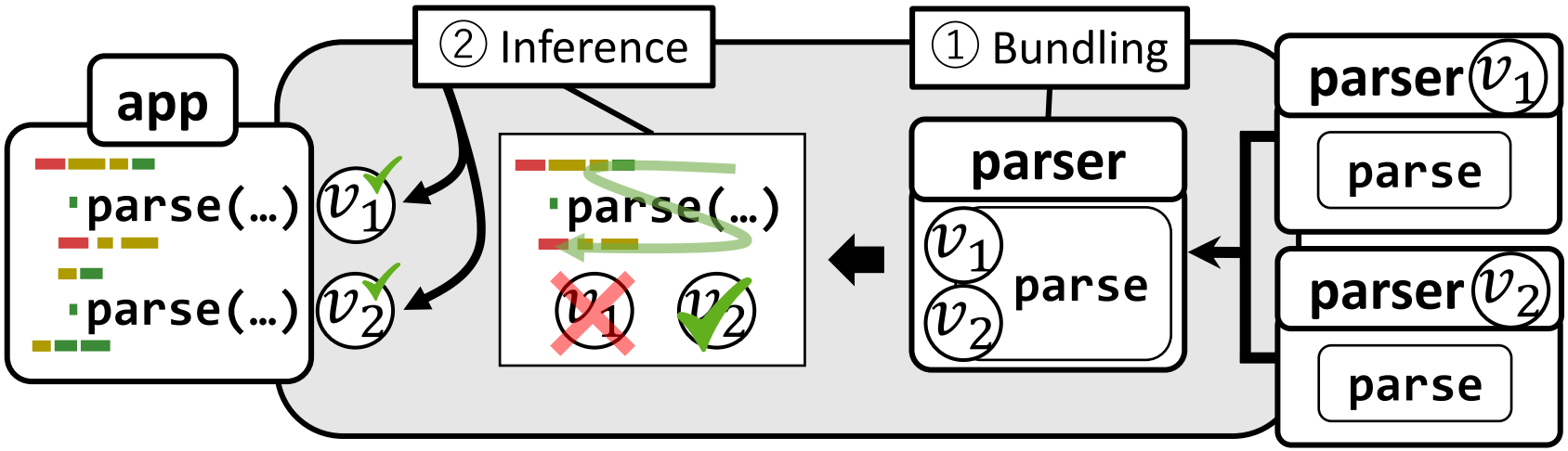
Implicit assumption:

One-version-at-a-time

Proposal

A language enables

Multiple versions in one value



Core: λ_{VL}

Coeffect calculus

+ *Versions as resources*

Future work

- ❑ Implementation
- ❑ Adaptation

For Java: BatakJava[TBD]