A Context-Oriented Programming Approach to Dependency Hell

Yudai Tanabe, Aotani Tomoyuki, and Hidehiko Masuhara

School of Computing,

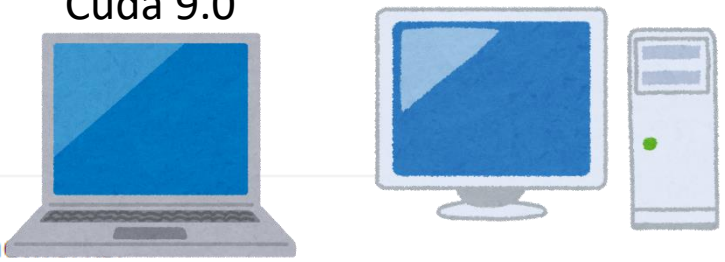
Department of Mathematical and Computing Science

Tokyo Institute of Technology

Downloading library
—Which library should we choose?—

Cuda 9.0

Cuda 8.0



cuDNN Download

NVIDIA cuDNN is a GPU-accelerated library of primitives for deep neural n

I Agree To the Terms of the cuDNN Software License Agreement

Cuda 9.2

Three cuDNN libraries for three environment

Which one…?

rerequisi … pported GPU … ding.

ide, In … nd Release Notes on the Deep Learning SDK Documentation web page.

Download cuDNN v7.1.4 (May 16, 2018), for CUDA 9.2

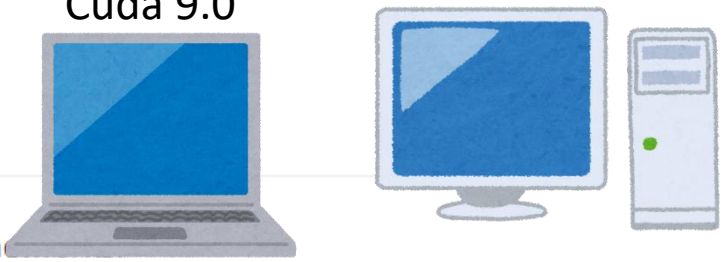
Download cuDNN v7.1.4 (May 16, 2018), for CUDA 9.0

Download cuDNN v7.1.4 (May 16, 2018), for CUDA 8.0

Downloading cuDNN
—One library in any environment—

Cuda 9.0

Cuda 8.0



cuDNN Download

NVIDIA cuDNN is a GPU-accelerated library of primitives for deep neural n

I Agree To the Terms of the cuDNN Software License Agreement

Cuda 9.2

rerequisi pported GPU
ding.
ide, In nd Release Notes on
the Deep Learning SDK Documentation web page.

One cuDNN library
for three environment

Download cuDNN v7.1.14 (May 16, 2018)

Multiple versions of libraries
—“Untyped version abstraction”—

- Make libcurl work with any version of openssl
- openssl is not backwards compatible
- libcurl uses macros to abstract differences between versions of openssl
  - Macros are untyped

```
#if OPENSSL_VERSION_NUMBER >= 0x0090581fL
#define HAVE_SSL_GET1_SESSION 1
#else
#undef HAVE_SSL_GET1_SESSION
#endif

#if OPENSSL_VERSION_NUMBER >= 0x00904100L
#define HAVE_USERDATA_IN_PWD_CALLBACK 1
#else
#undef HAVE_USERDATA_IN_PWD_CALLBACK
#endif

#if OPENSSL_VERSION_NUMBER >= 0x00907001L
```

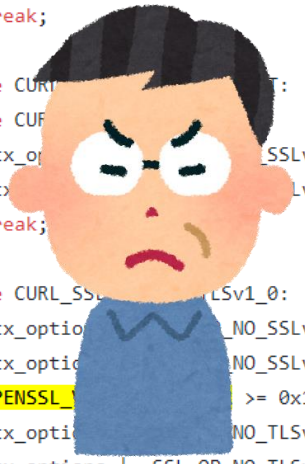
```
#if OPENSSL_VERSION_NUMBER >= 0x0090700
/* 0.9.6 didn't have X509_STORE_set_fla
#define HAVE_X509_STORE_SET_FLAGS 1
#else
#define X509_STORE_set_flags(x,y) Curl_
#endif

#if OPENSSL_VERSION_NUMBER >= 0x1000000
#define HAVE_ERR_REMOVE_THREAD_STATE 1
```

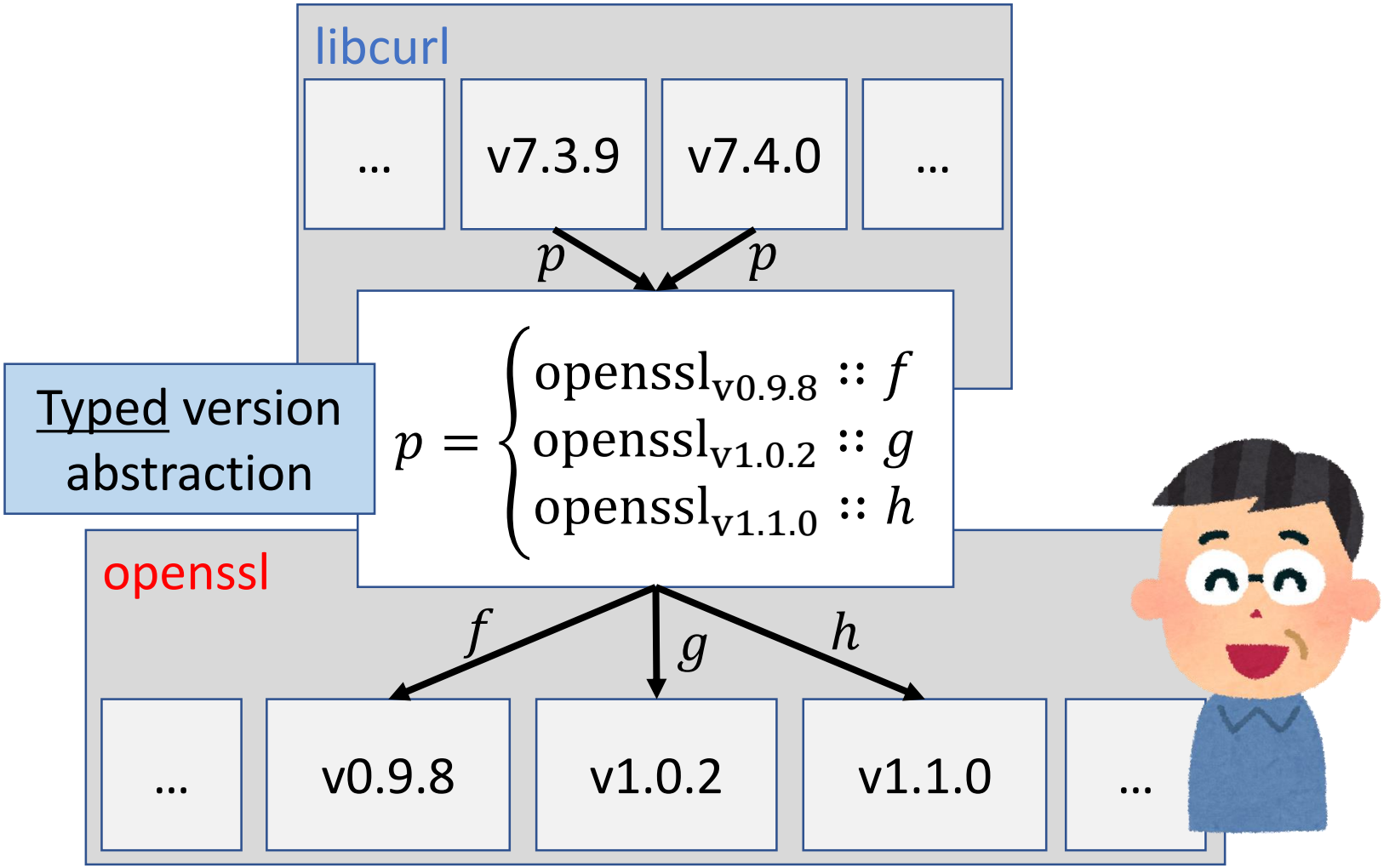
```
#if OPENSSL_VERSION_NUMBER >= 0x1000100F
    ctx_options |= SSL_OP_NO_TLSv1_1;
    ctx_options |= SSL_OP_NO_TLSv1_2;
#endif
    break;

 case CURL
 case CUR
    ctx_o                    SSLv2;
    ctx                      Lv3;
    break;

 case CURL_SS                LSv1_0:
    ctx_optio                NO_SSLv2;
    ctx_optio                NO_SSLv3;
#if OPENSSL_                 >= 0x1000100F
    ctx_opti                 NO_TLSv1_1;
    ctx_options |= SSL_OP_NO_TLSv1_2;
#endif
    break;
```

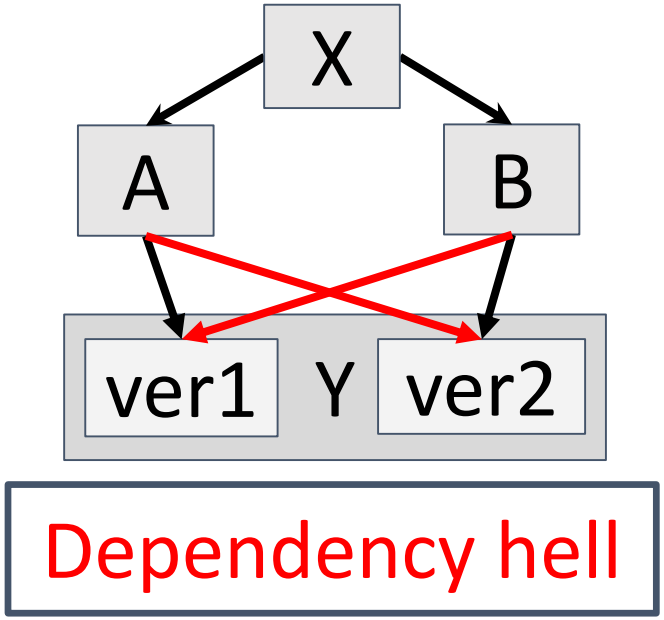


Multiple versions of libraries
—“Typed version abstraction”—



Dependency hell
—What is dependency hell?—

- **Dependency hell** : *when incompatible versions of a library are needed in one software artifact.*



depend on

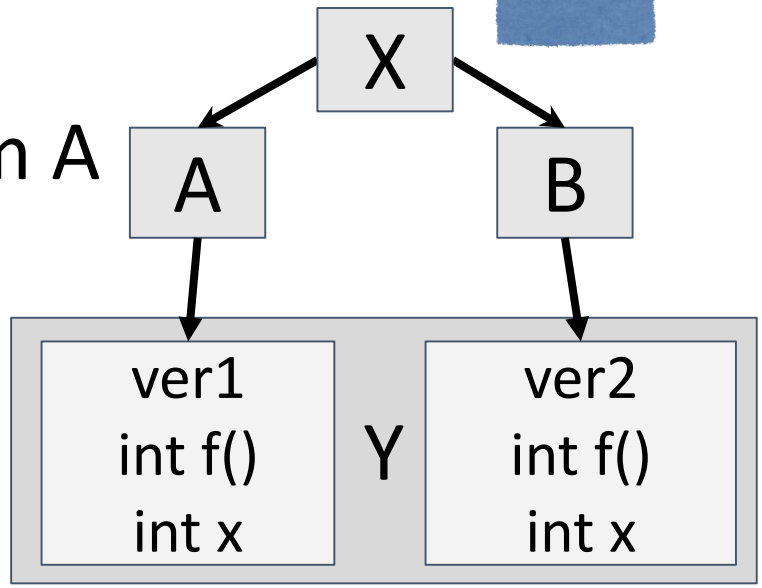
incompatible

Dependency hell
—How does dependency hell occur?—

For example:

Dependency hell
—One name, multiple definitions—

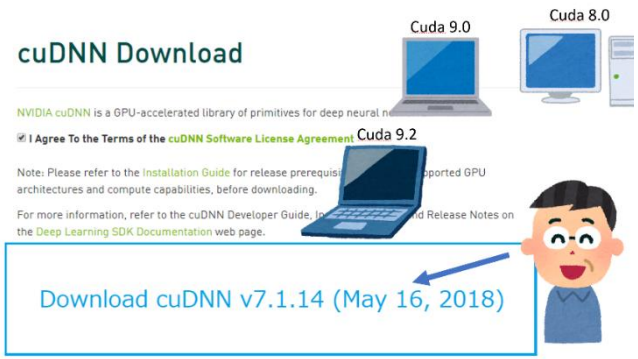
- Allows simultaneous use of multiple versions of a library
  - No collision even if there is a value / function with the same name
  - Allows selecting ver1 from A and ver2 from B

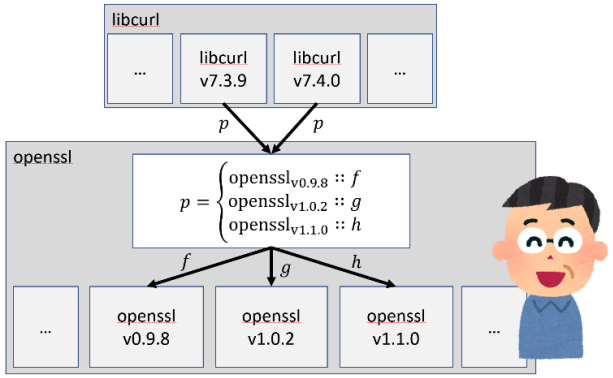


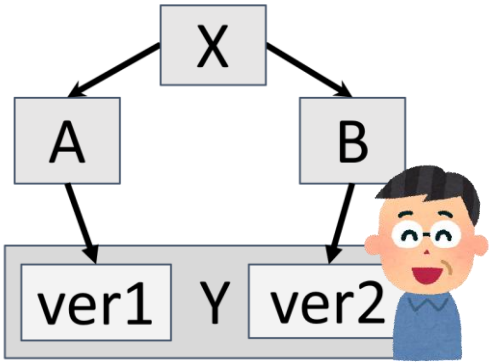
Our approach - COP

- We propose a solution to these problems using context-oriented programming.

Key idea: <u>Versions as contexts</u>

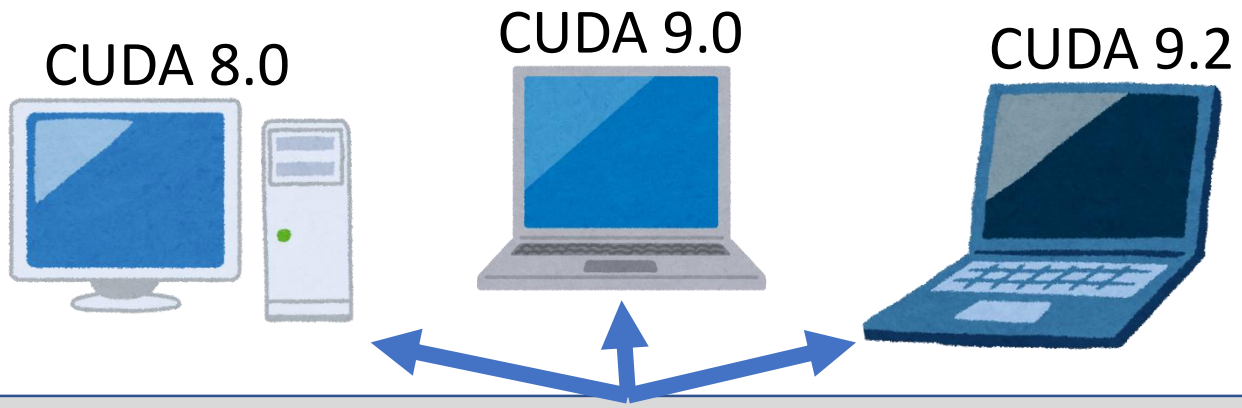




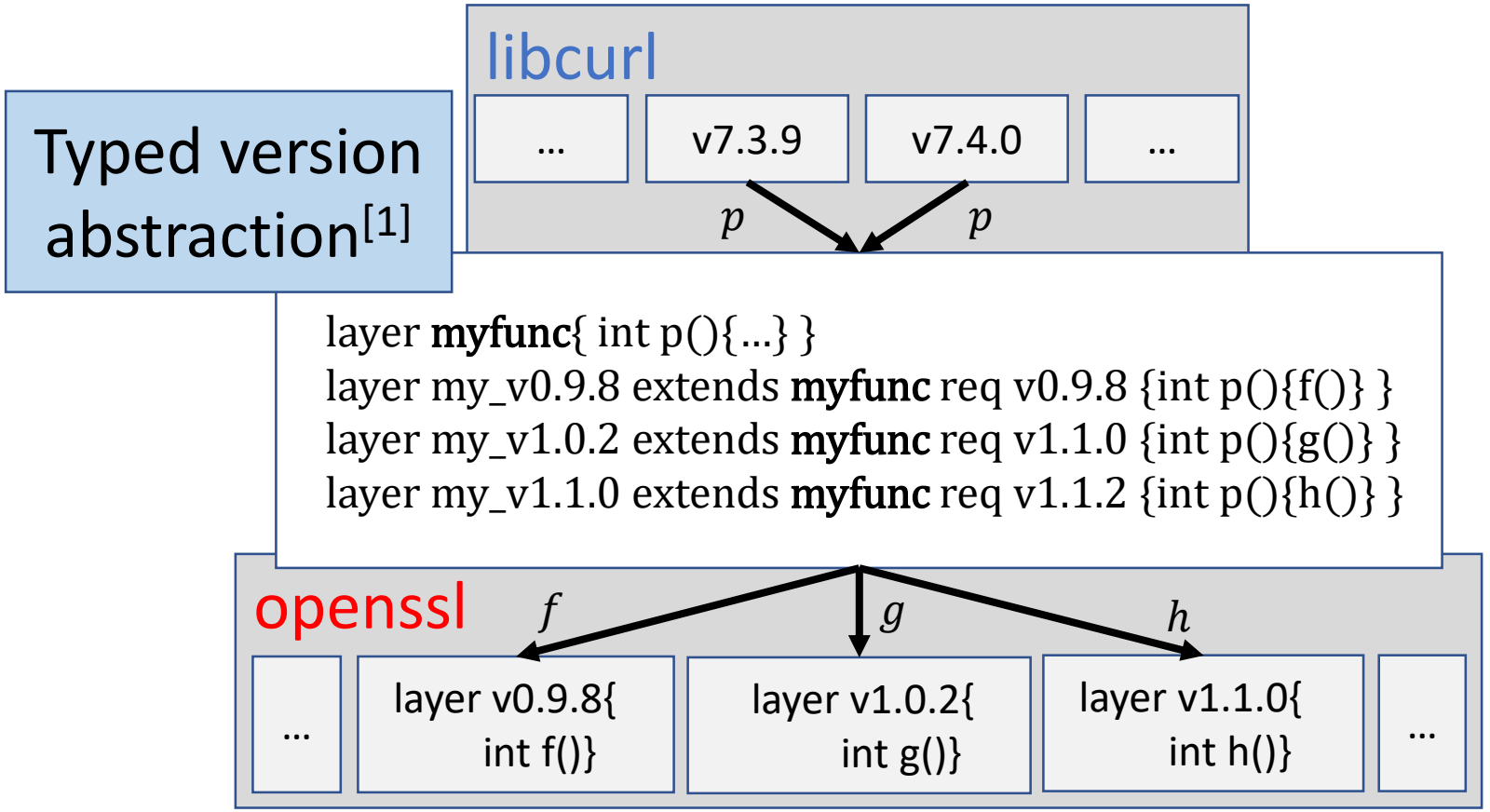


Application
—Downloading cuDNN—

- Can provide one version that will work properly in all environments



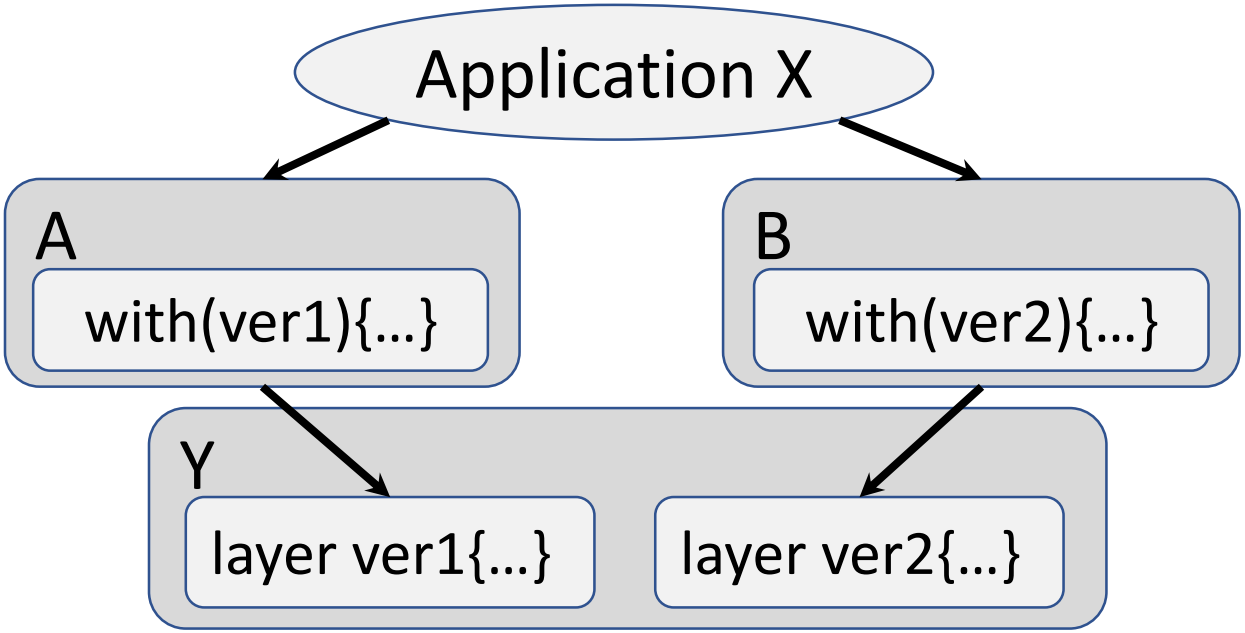
Application
—Depending on multiple versions of a library—



[1] Inoue H., Igarashi A: A Sound Type System for Layer Subtyping and Dynamically Activated First-Class Layers. In: APLAS 2015

Application
—Dependency hell—

- Each version of the library is represented *Layers*
- Allows selecting ver1 from A and ver2 from B using *with*



Basic idea

- Multiple versions of value and function definitions
- Put versions on *values* and propagate them
- Contexts are selected by extracting a specific version from the program

※ Assume the following for simplification

  - No I/O, no state.
  - The versions of library don't change from typechecking-time and at runtime.
  - Functions are values.

Issues

- Without classes and objects,
  - How do you represent version-dependent programs?
  - How do you interpret version-independent programs?

Issues

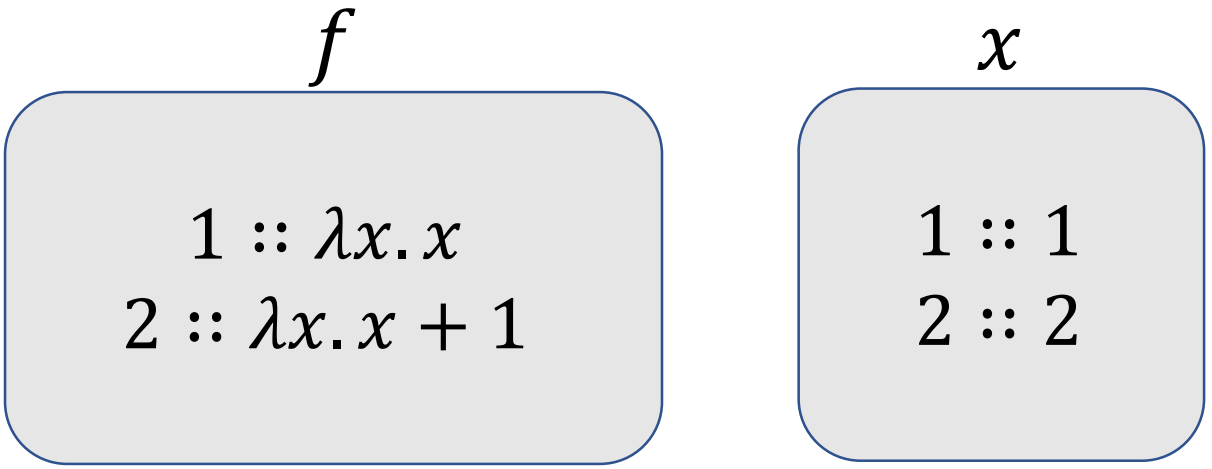
- Without classes and objects,
  - How do you represent version-dependent programs?
    - Ex.)

$$p = \begin{cases} \text{openssl}_{\text{v0.9.8}} :: f \\ \text{openssl}_{\text{v1.0.2}} :: g \\ \text{openssl}_{\text{v1.1.0}} :: h \end{cases} \qquad x = \begin{cases} \text{openssl}_{\text{v0.9.8}} :: v_1 \\ \text{openssl}_{\text{v1.0.2}} :: v_2 \end{cases}$$

  - How do you interpret version-independent programs?
    - Ex.)

$$p\ x = ???$$

Representation of version-dependent programs

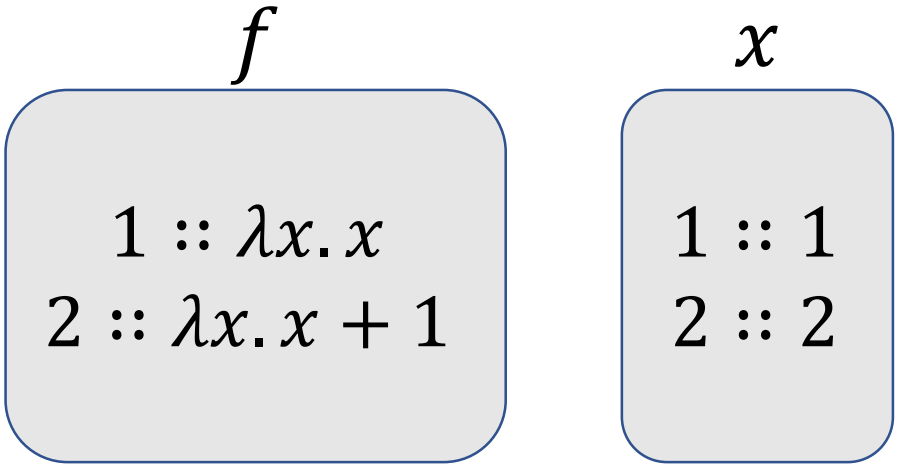


- Version abstraction are realized by packaging multiple versions of values into **versioned values** $\{\ldots\}$.

$$f = \{1 :: \lambda x.\, x, 2 :: \lambda x.\, x + 1\}$$
$$x = \{1 :: 1, 2 :: 2\}$$

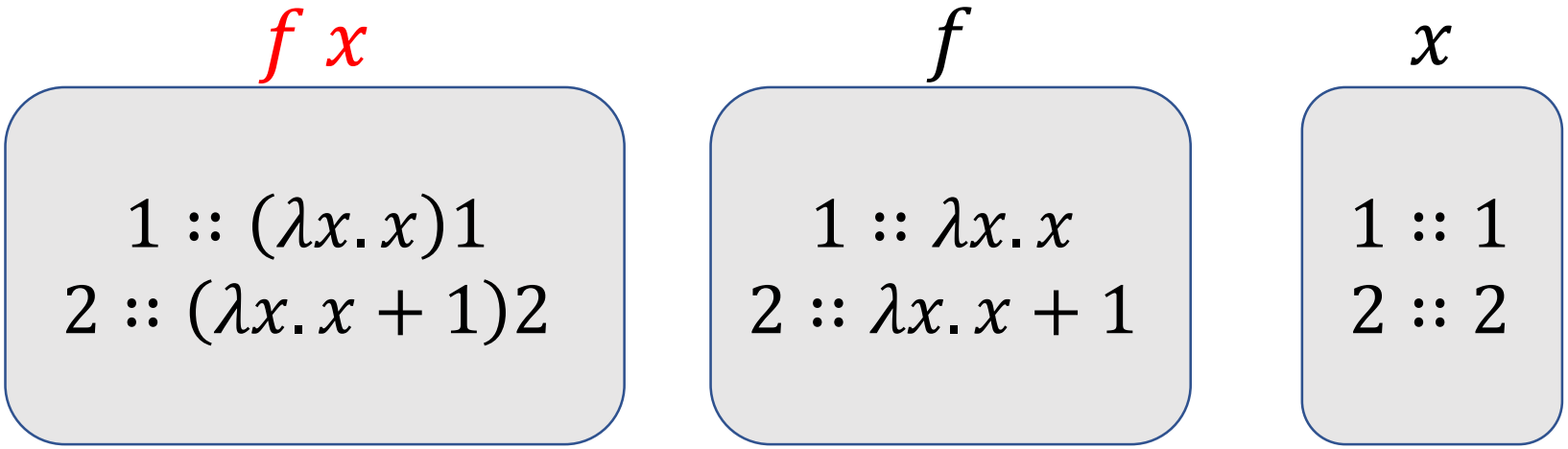
Interpretation of version-independent applications

$f\ x = ???$



- So, how should version-independent program: $f\ x$ be interpreted?
- We want to compute $f\ x$ in the current version, but how to get the version?

Interpretation of version-independent applications

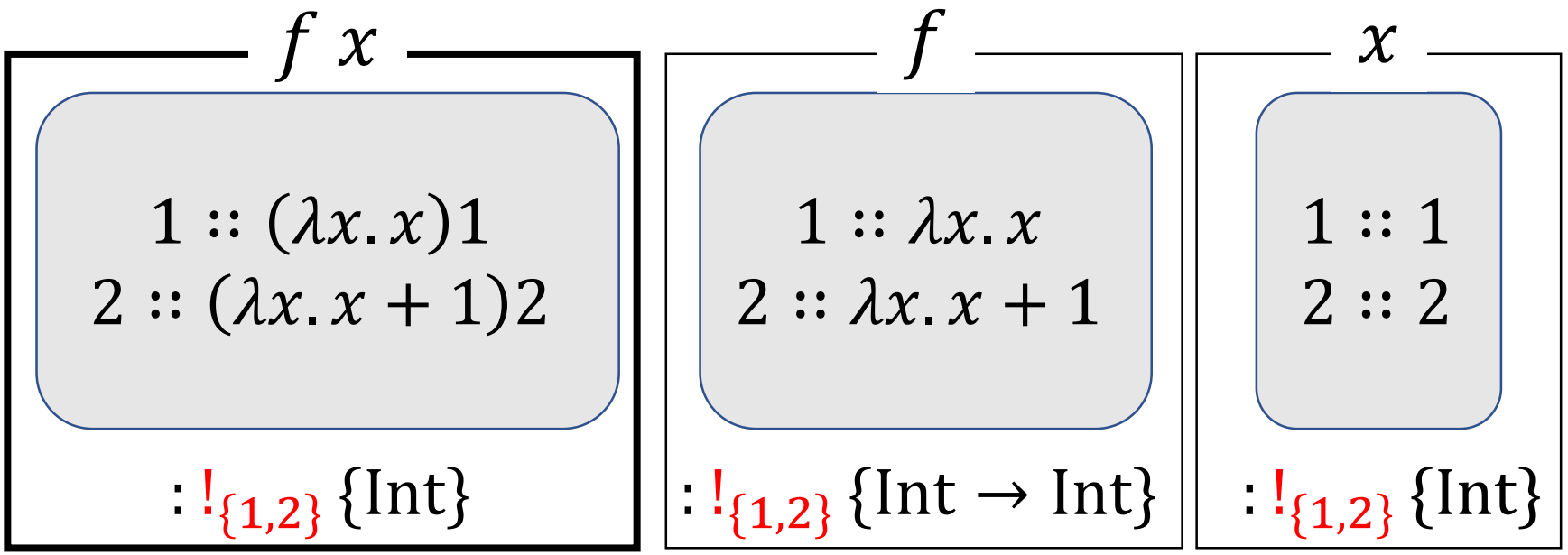


- We define $f\ x$ as calculating values <u>in all shared versions</u>.

$$f\ x = \{1 :: (\lambda x.x)1, 2 :: (\lambda x.x+1)2\ \}$$

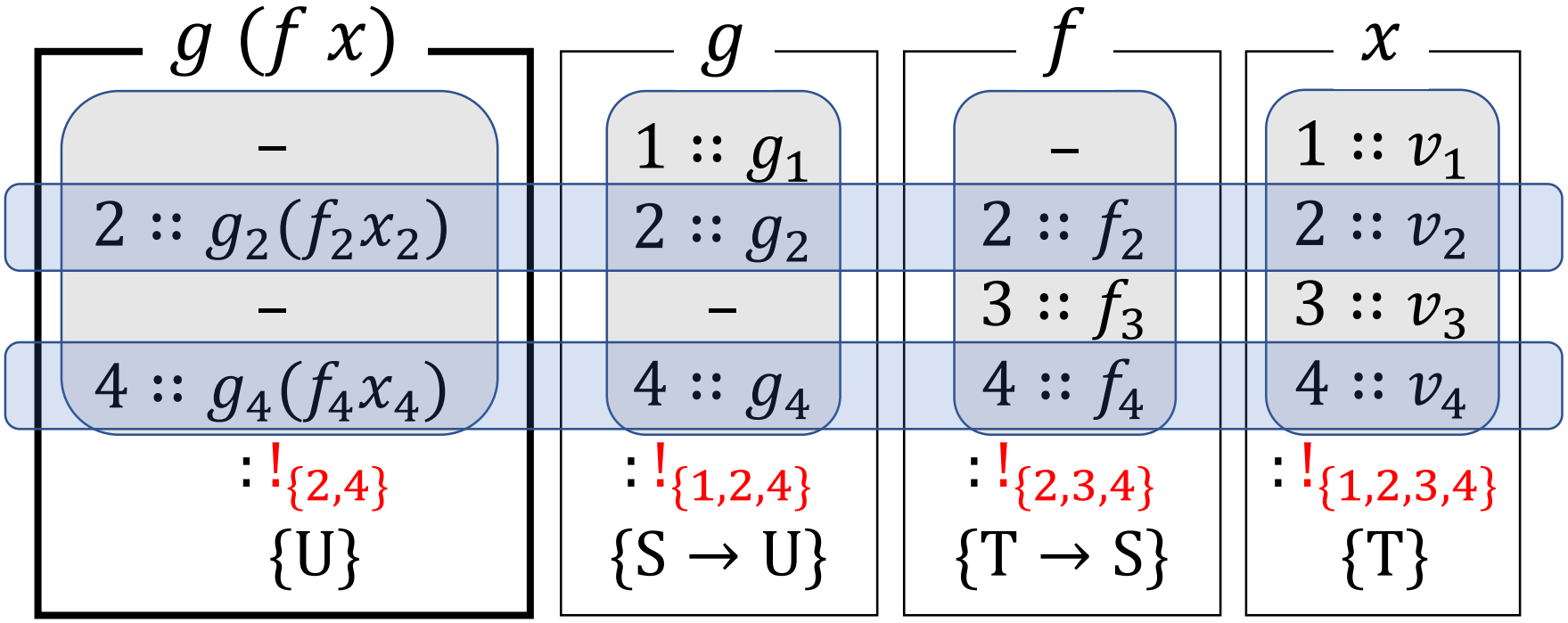
- $f\ x$ is defined in versions 1 and 2 where definitions exist in both f and x.

Types of versioned values



- The types of versioned values are defined using a ! annotation.

Flow of versions



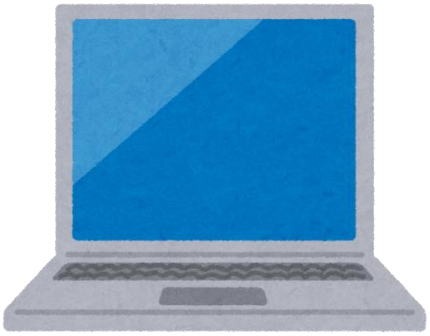
- Functional application is computed in all shared versions.

$$\{2,4\} = \{1,2,4\} \cap \{2,3,4\} \cap \{1,2,3,4\}$$

Evaluating versioned values

Source program

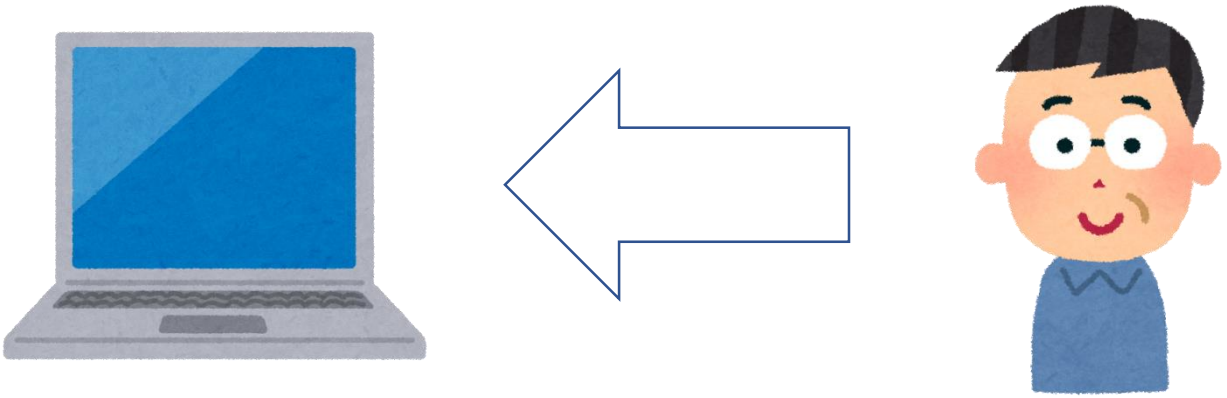
$g\ (f\ x)$



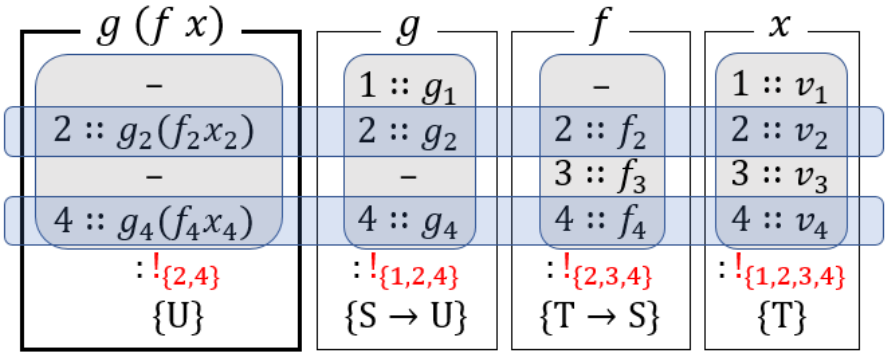
Evaluating versioned values

Source program
$g\ (f\ x)$

Compile the program



Evaluating versioned values



Source program
$g\ (f\ x)$

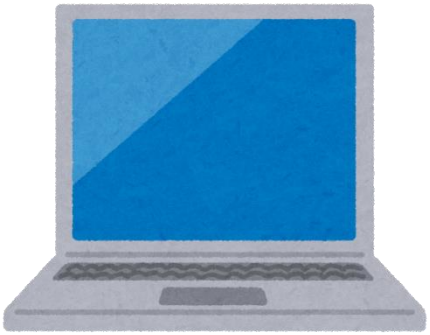
Compile the program

Type checking…

Evaluating versioned values

Compiled program
$g\ (f\ x)$: $!_{\{2,4\}}\ \{\mathrm{U}\}$

Compilation completed

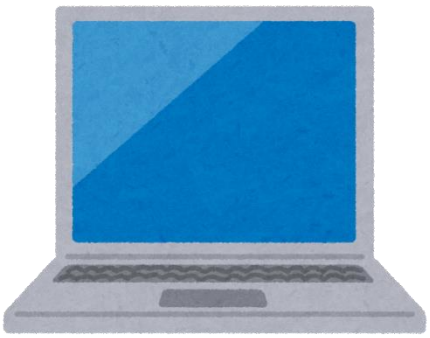


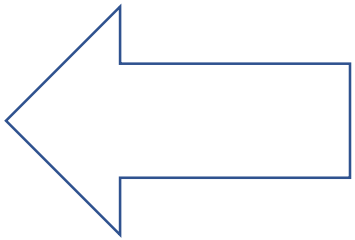
Evaluating versioned values

Compiled program

$g\ (f\ x)$: $!_{\{2,4\}}\ \{\mathrm{U}\}$

Run the program

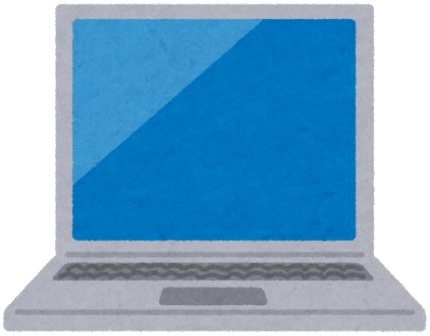


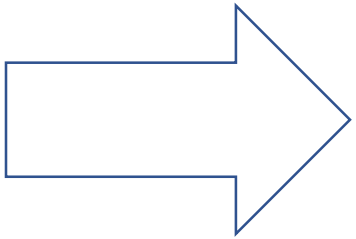


Evaluating versioned values

Compiled program
$g\ (f\ x)$: $!_{\{2,4\}}\ \{\mathrm{U}\}$

Which version do you want to run?



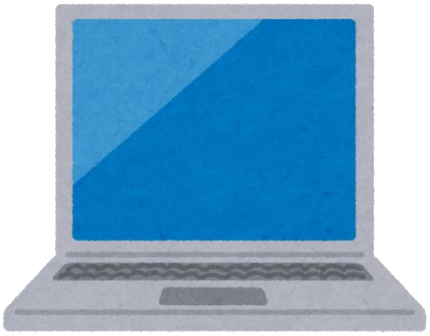


Evaluating versioned values

Compiled program
$g\ (f\ x)$: $!_{\{2,4\}}\ \{\mathrm{U}\}$

Which version do you want to run?

Version 2

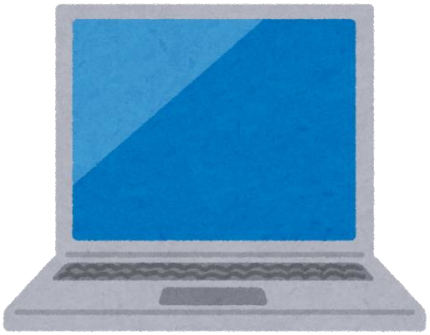


Evaluating versioned values

Compiled program
$g\ (f\ x)$: $!_{\{2,4\}}\ \{\mathrm{U}\}$

Evaluate $g\ (f\ x)$
at version 2

Version 2



Conclusion

- We propose the use of COP as a solution to dependency hell.
  - We allow libraries to have multiple versions of definitions and users to select one specific version.
  - We have developed a core calculus λ_{VL} and proven its soundness.

Difference from package managers

- For example, maven, sbt, OPAM, …
- Package managers aims to distinguish the package set with dependency
- Package managers don’t provide a mechanism to abstract versions

Relationship to COP

- Similarities:
  - Multiple-versioned programs
    ≃ using a COP layer as a version
  - Extracting specific versions
    ≃ COP layer activation