

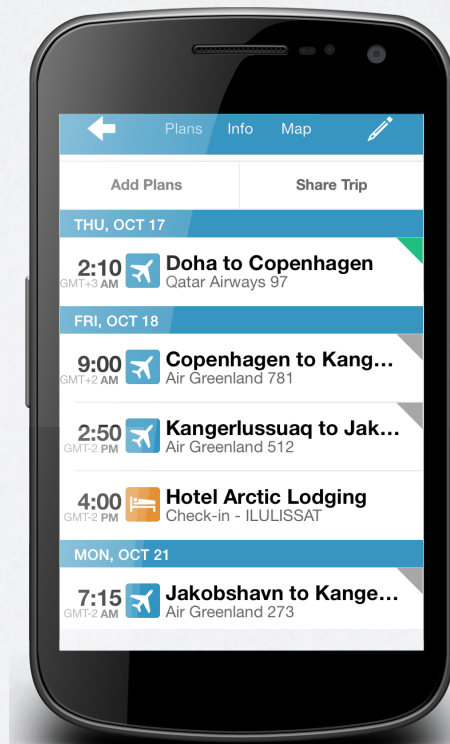
Controlling Data Flow with a Policy-Based Programming Language for the Web

Thierry Sans

Iliano Cervesato

Soha Hussein

Privacy from the user's perspective



Privacy from the user's perspective



Privacy from the user's perspective



Privacy from the user's perspective



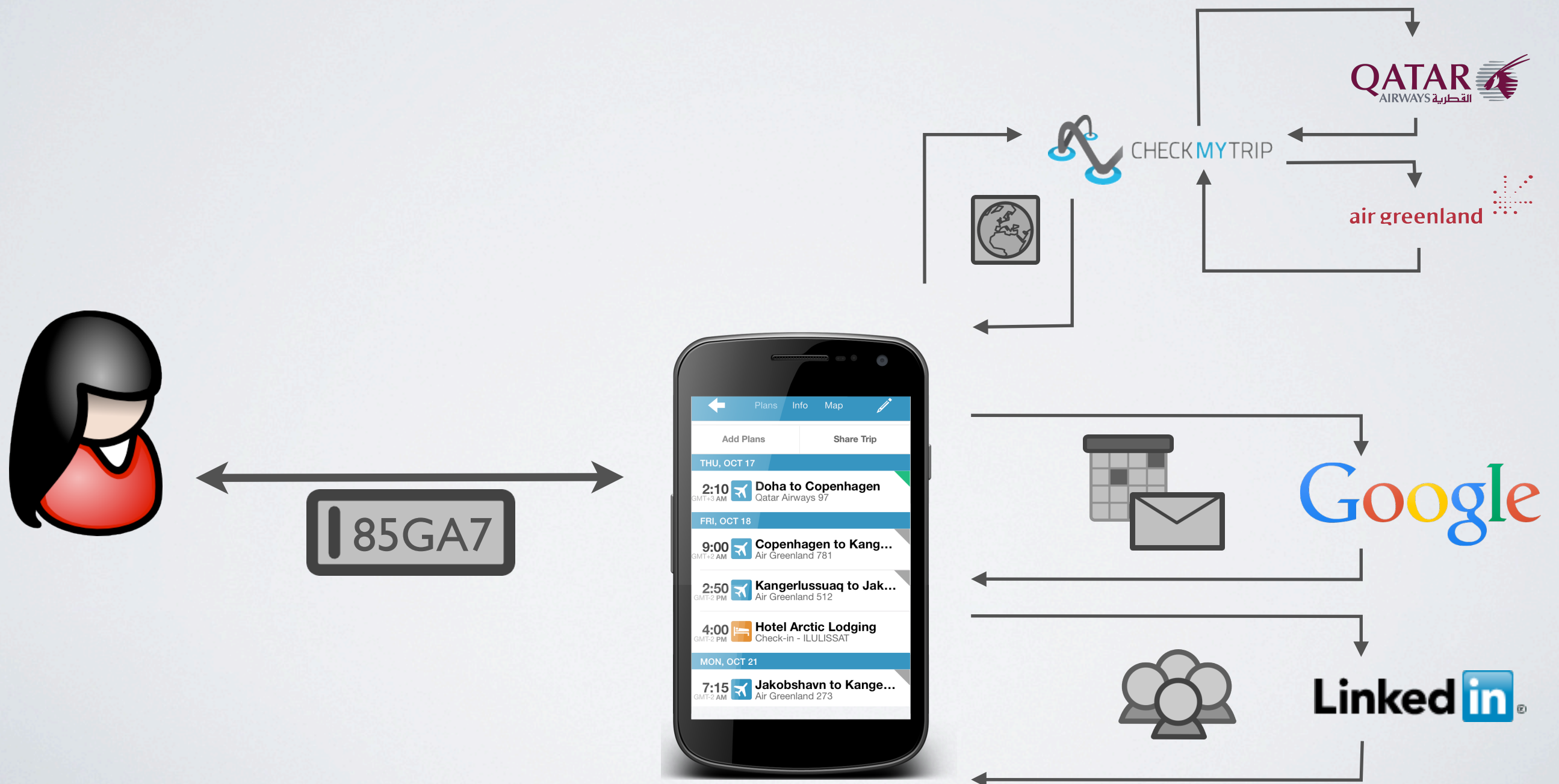
Privacy from the user's perspective



Privacy from the user's perspective



Privacy from the user's perspective

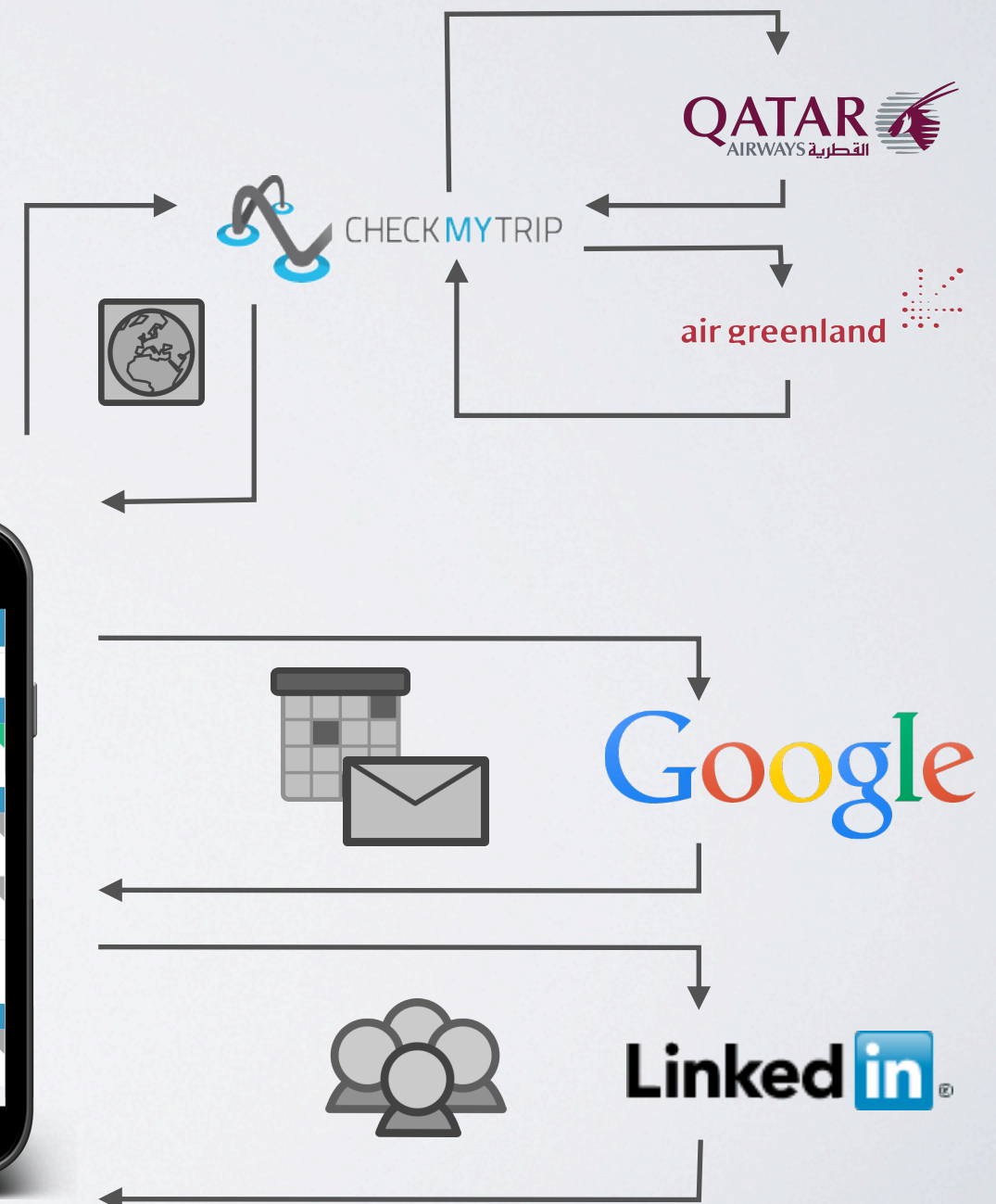
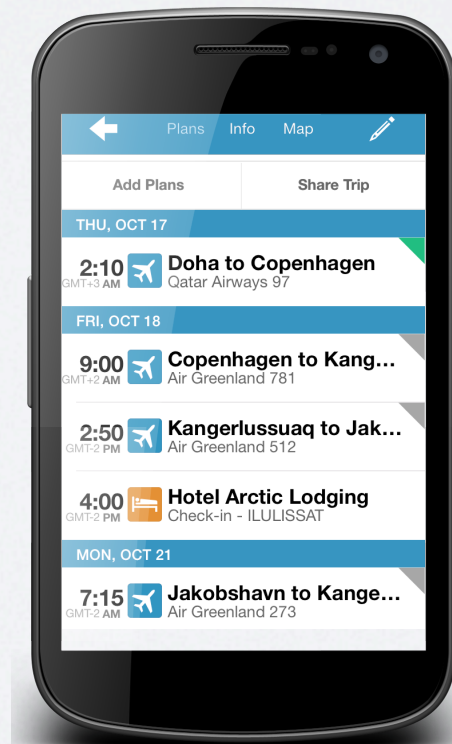


Privacy from the user's perspective

Should I use this app?
How does this app use my data?

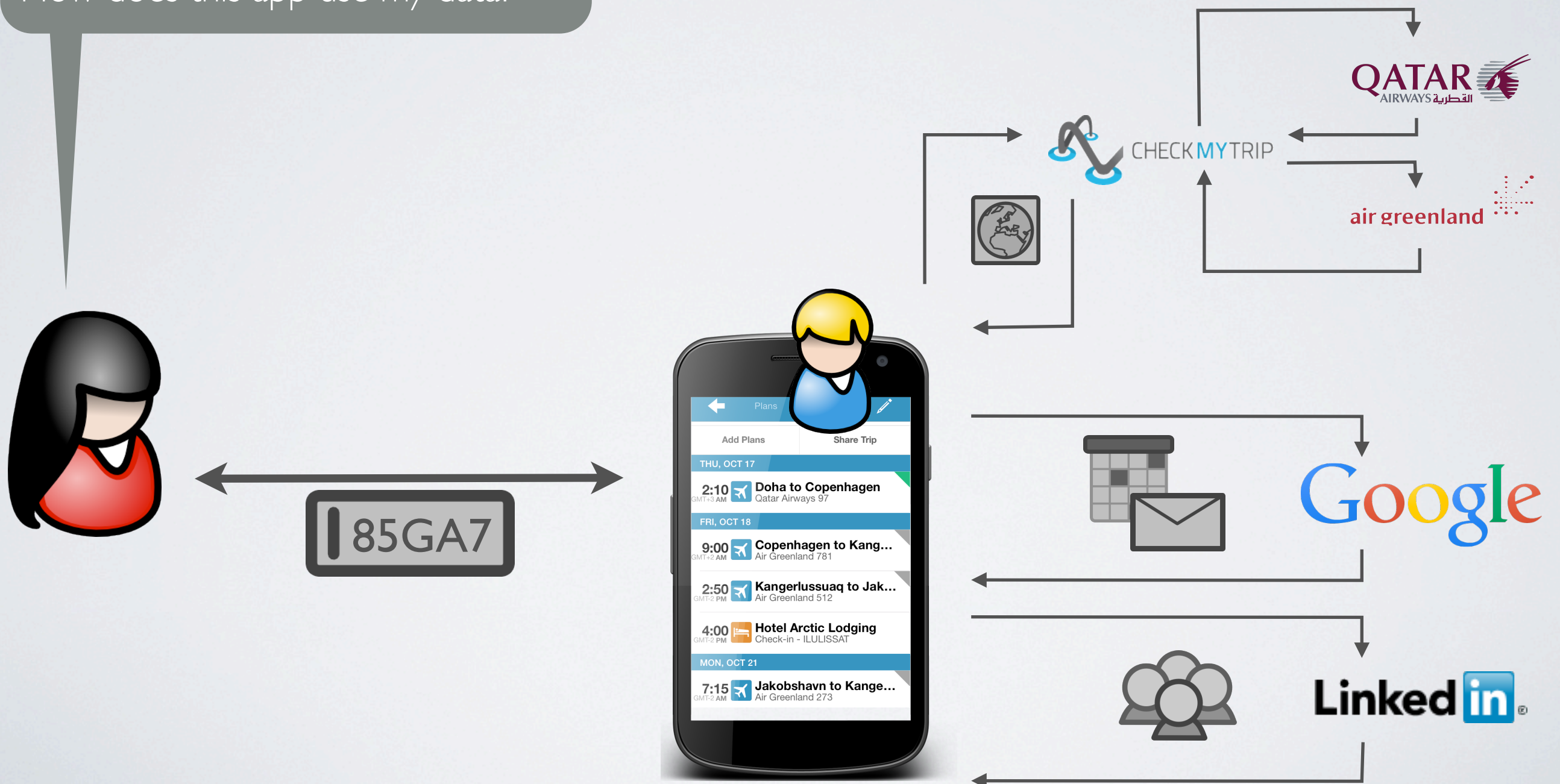


85GA7



Privacy from the user's perspective

Should I use this app?
How does this app use my data?



Privacy from the user's perspective

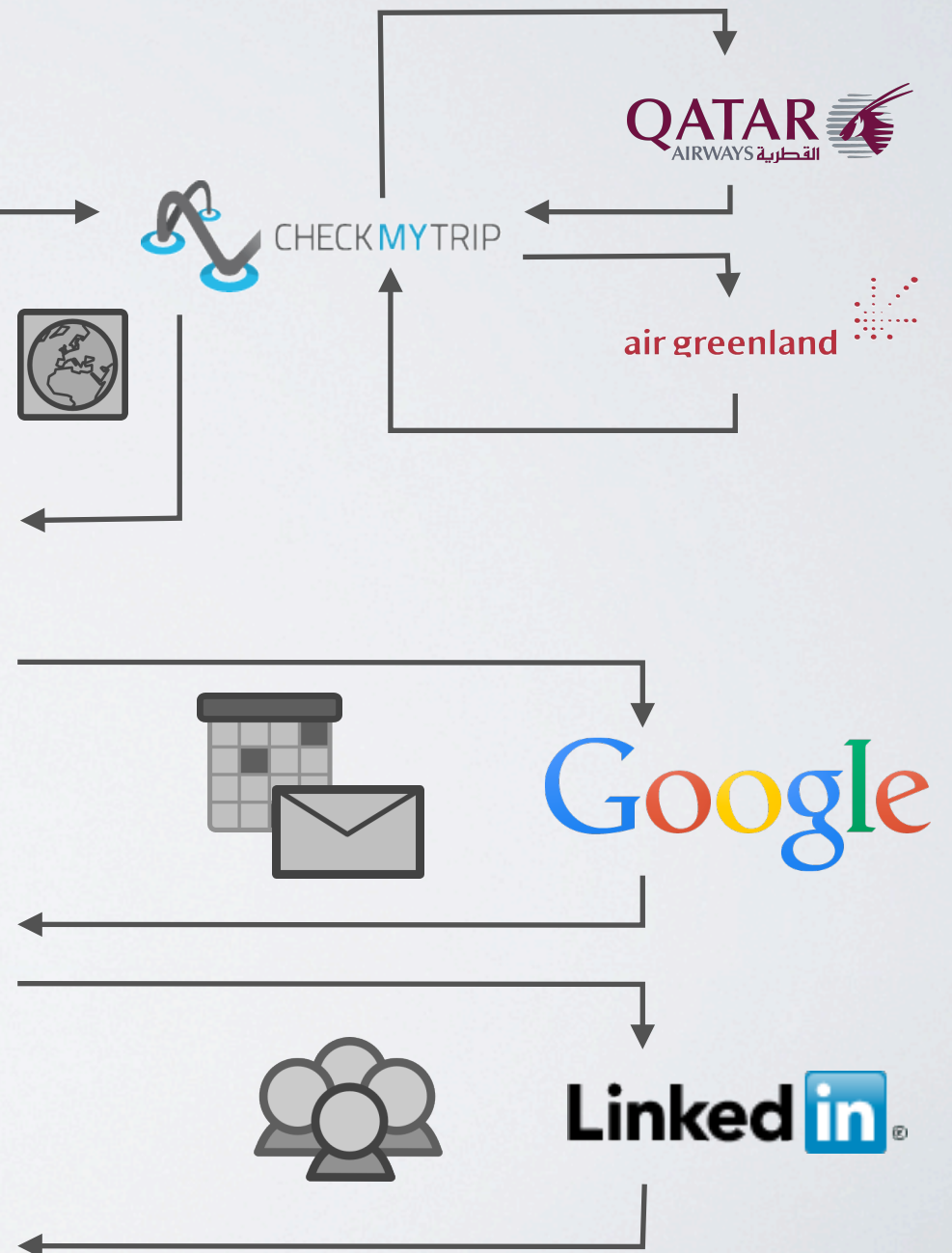
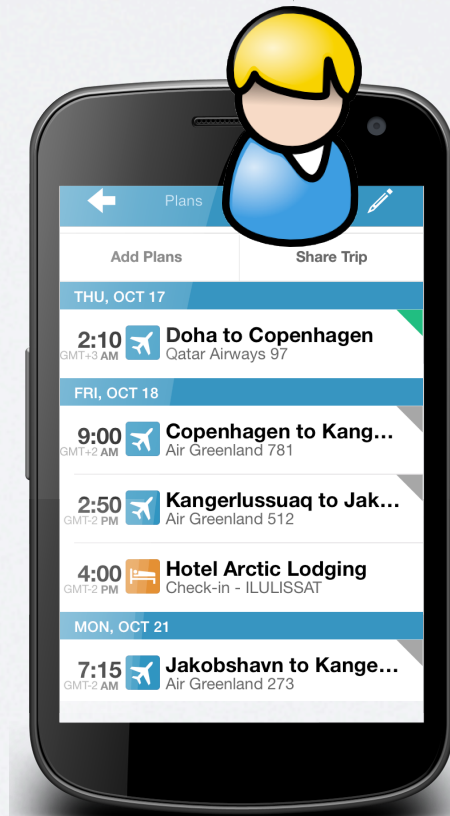
Should I use this app?
How does this app use my data?



App's Privacy Policy



85GA7



Privacy from the user's perspective

Should I use this app?
How does this app use my data?

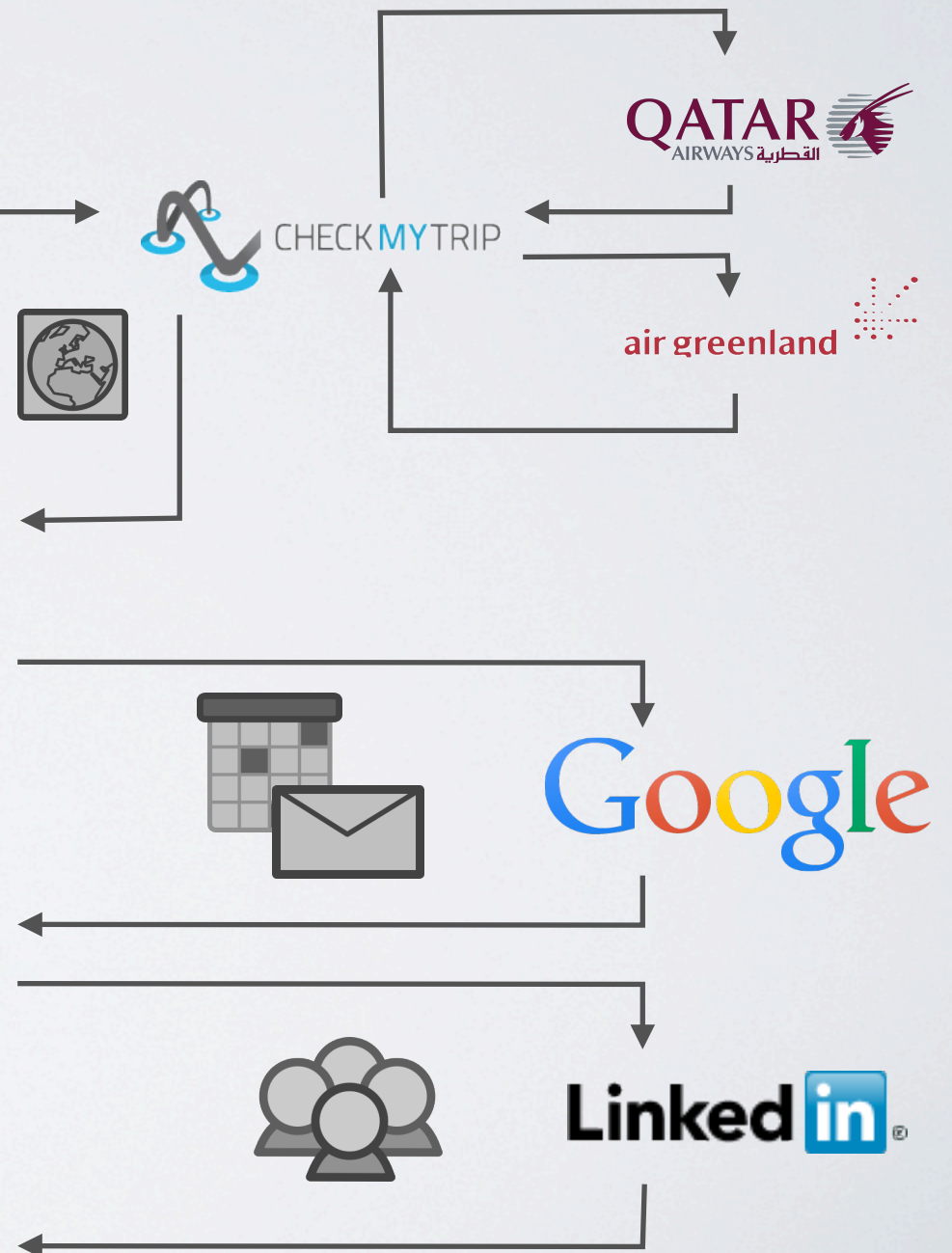
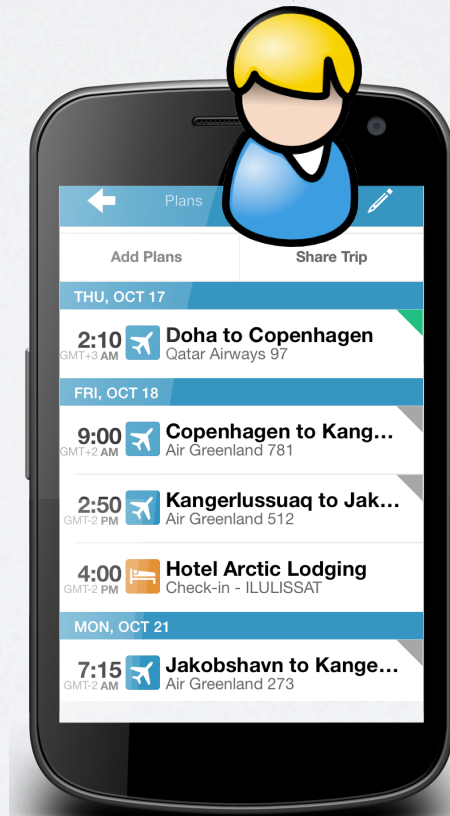
User's Privacy
Expectations



App's Privacy Policy



85GA7



Privacy from the user's perspective

Should I use this app?
How does this app use my data?

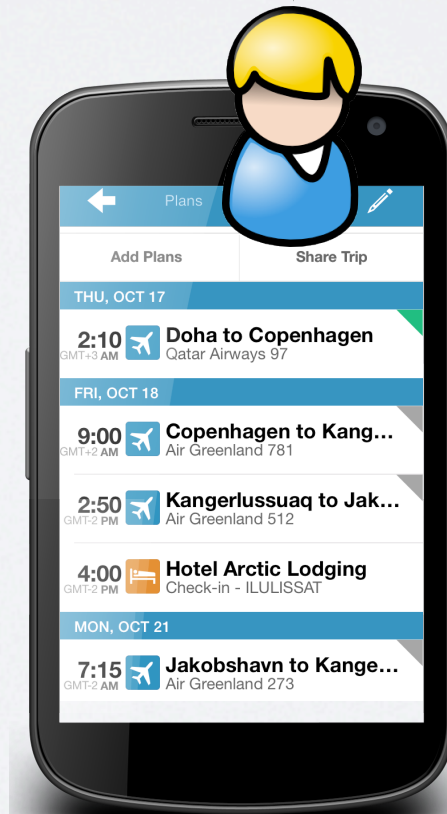
User's Privacy
Expectations



App's Privacy Policy



85GA7



Privacy Policy

(last revised April 16, 2012)

The following Privacy Policy summarizes the various ways that we use the information you provide to us or we gather from you while you use the tripit.com online service (the "Service"), as provided by Concur Technologies, Inc. ("Concur" or "us" or "we"), through our website at www.tripit.com (the "Site"). It is our goal to provide you with information that is tailored to your individual needs and, at the same time, protect your privacy.



1. What Information We Collect from You

When you register with Concur, we ask for your email address and password and once registered allow you to provide us with your name, home location and up to three other email addresses in order to access your account as well as other information including your travel information, history and preferences. Concur then uses this information to build personalized travel itineraries.

Google ID

We allow you to use your Google ID in order to set up a Triplt Service account. If you wish to use your Google account you are taken to the Google website to login to your account, this information is then shared back with us for the sole purpose of setting up your account.

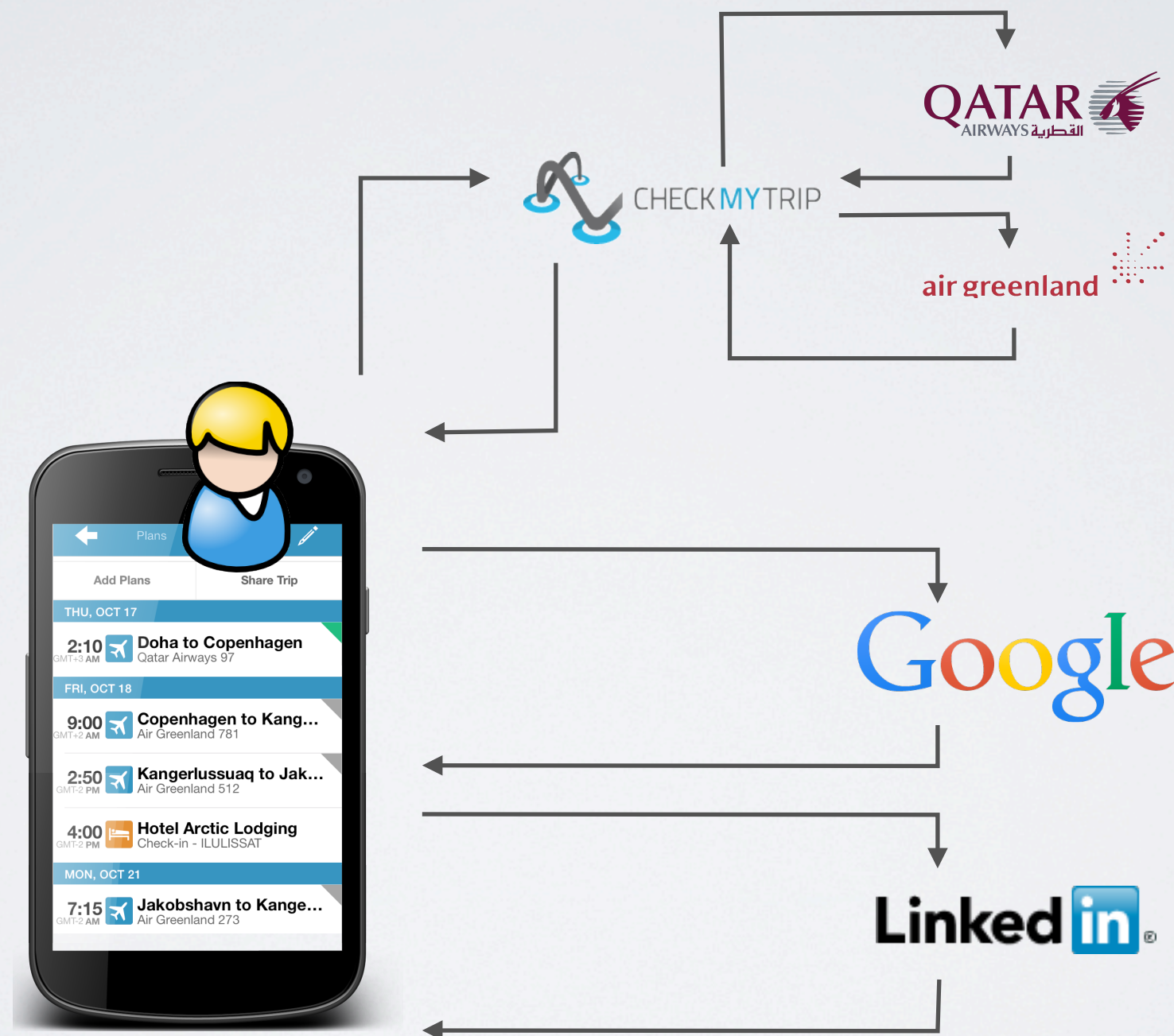
My Travel Application on LinkedIn

We allow you to share your travel information on LinkedIn by installing the "My Travel" application and then authorizing LinkedIn to access your Triplt Service account. Once the application is installed, you can invite your LinkedIn connections to use the same application.

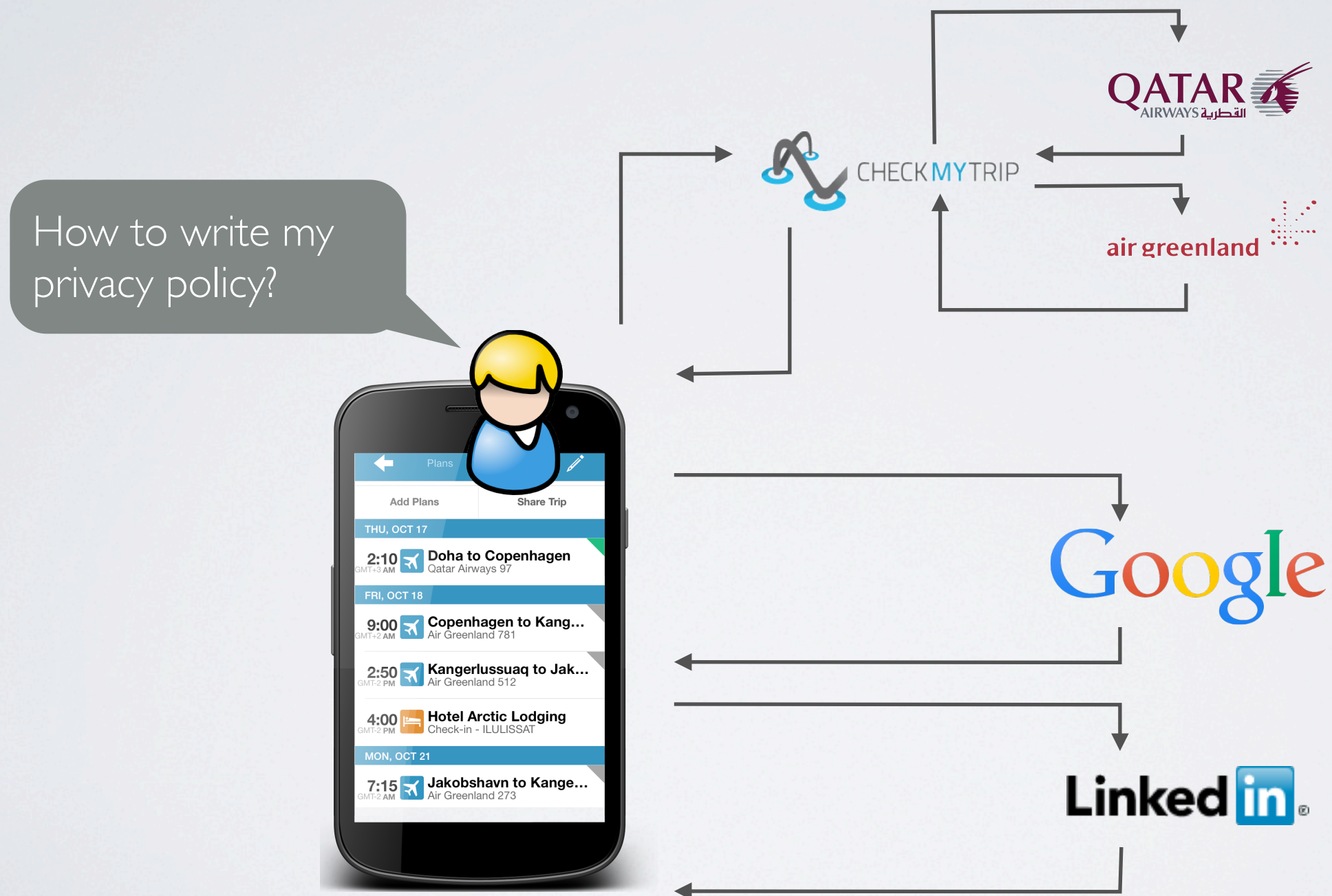
4. When We May Share Your Information

The information we gather is used to create and maintain your travel itineraries and your travel profile and, if you have elected to receive them, send you marketing messages on behalf of Triplt and our travel partners. We also use the information that we gather to target content on our Site to more closely match your interests and, if you have selected to receive email communication from us, target email marketing messages that are more appropriate to your needs.

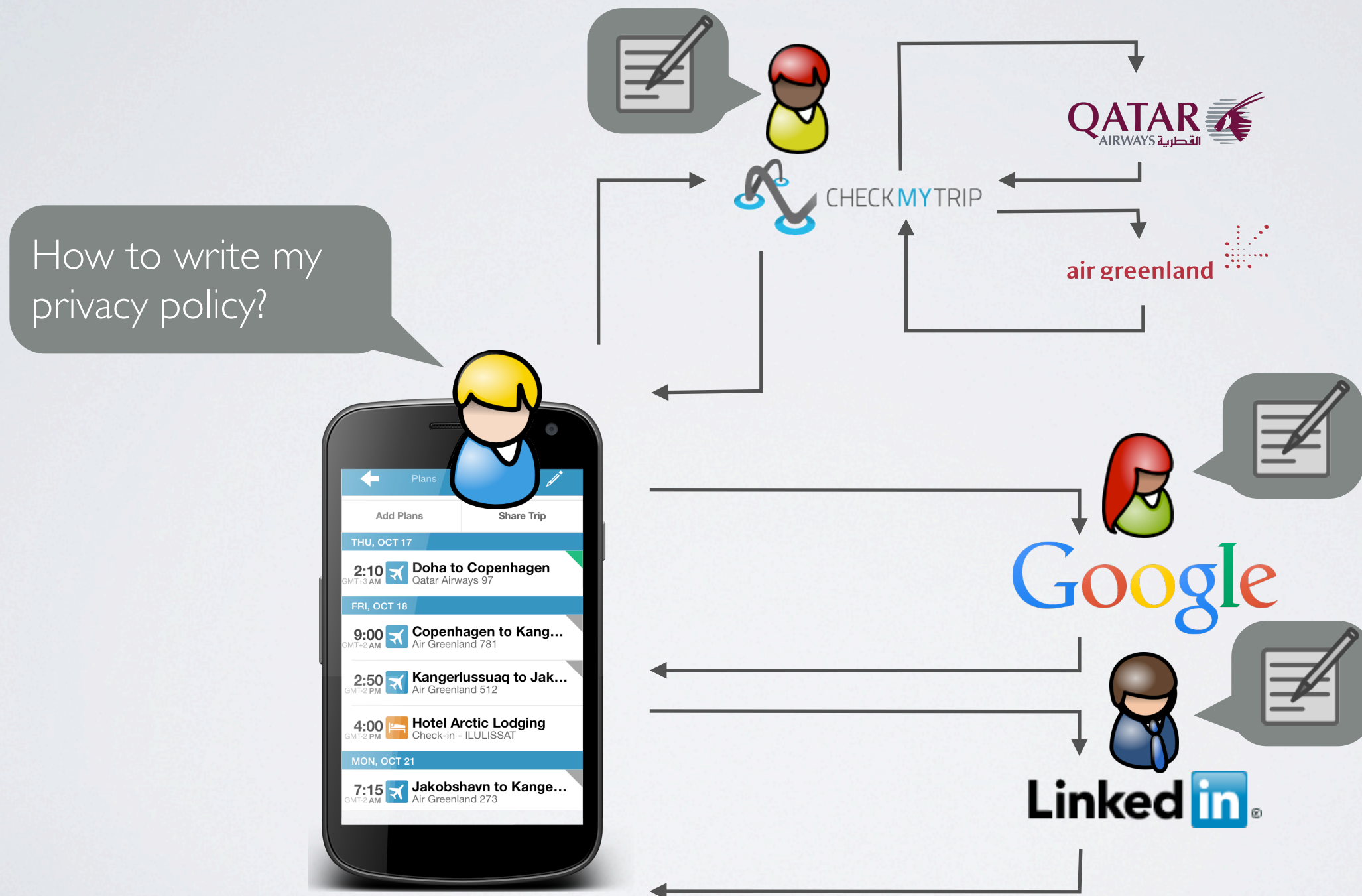
Privacy from the web developer's perspective



Privacy from the web developer's perspective



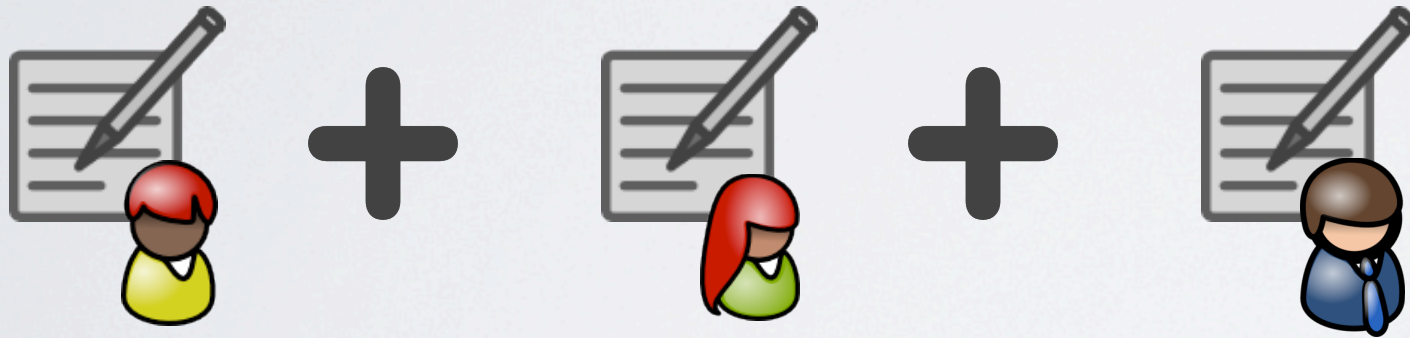
Privacy from the web developer's perspective



Can we generate a privacy policy automatically?

Can we generate a privacy policy automatically?

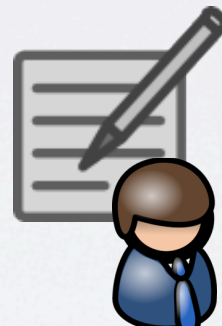
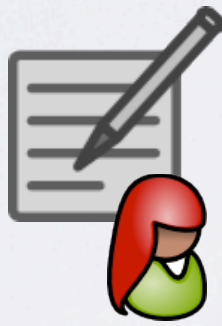
Based on the privacy policies of
third party service providers ...



Can we generate a privacy policy automatically?

Based on the privacy policies of third party service providers ...

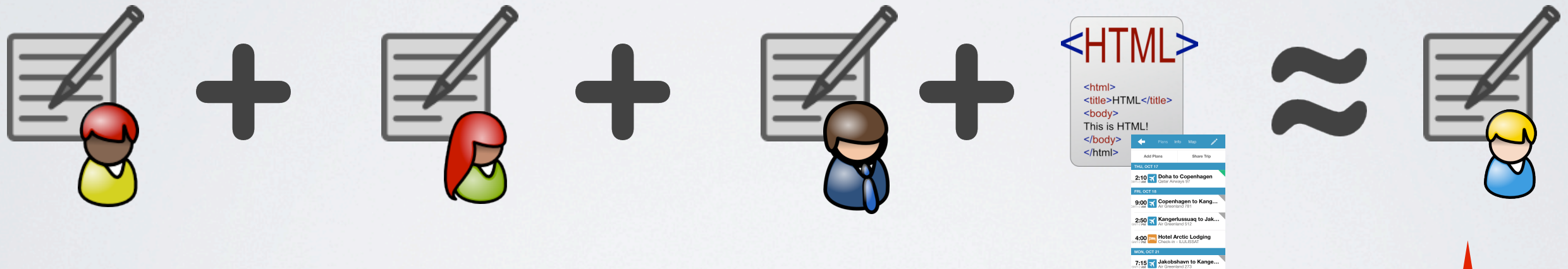
... and based on how the program combines these services ...



Can we generate a privacy policy automatically?

Based on the privacy policies of third party service providers ...

... and based on how the program combines these services ...

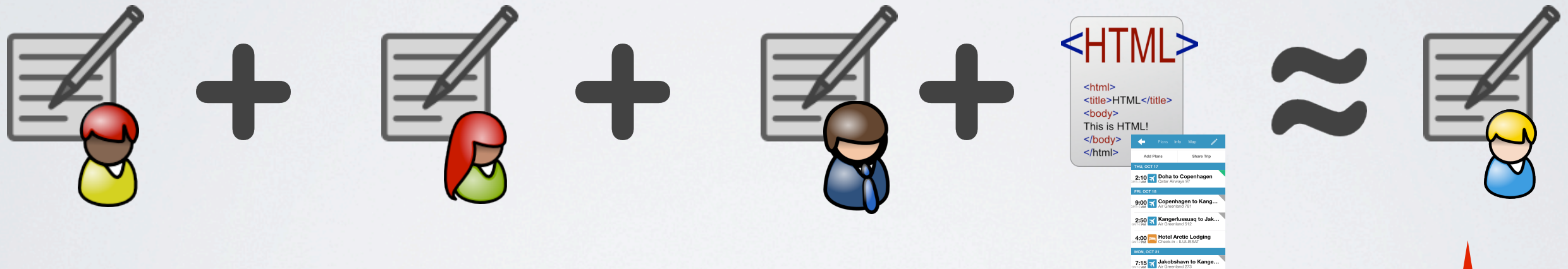


... can we infer how user's data will be used? *

Can we generate a privacy policy automatically?

Based on the privacy policies of third party service providers ...

... and based on how the program combines these services ...



... can we infer how user's data will be used? *

* no adversary model, third parties are trustworthy

Two solutions

Two solutions

Solution 1 : Analyze the programs and infer the data flow

➡ Static analysis, abstract interpretation

Two solutions

Solution 1 : Analyze the programs and infer the data flow

➔ Static analysis, abstract interpretation

Solution 2 : Design a new programming language

➔ **Qwel** (Qatar Web Programming Language) with data flow

Qwel : lambda calculus

+ primitives for remote procedure call

+ ...

Fragment of the language syntax

Types $\tau ::= \text{unit} \mid \tau \times \tau' \mid \tau \rightarrow \tau' \mid \tau \Rightarrow \tau'$

Expressions $e ::= x \mid \lambda x:\tau.e \mid (e_1 e_2) \mid \langle e_1, e_2 \rangle \mid \text{fst } e \mid \text{snd } e \mid ()$
 $\mid \text{w/u} \mid \text{publish } x:\tau.e \mid \text{call } e_1 \text{ with } e_2$

Qwel : lambda calculus

+ primitives for remote procedure call
+ ...

Fragment of the language syntax

Types $\tau ::= \text{unit} \mid \tau \times \tau' \mid \tau \rightarrow \tau' \mid \tau \Rightarrow \tau'$

Expressions $e ::= x \mid \lambda x:\tau.e \mid (e_1 e_2) \mid \langle e_1, e_2 \rangle \mid \text{fst } e \mid \text{snd } e \mid ()$
 $\mid w/u \mid \text{publish } x:\tau.e \mid \text{call } e_1 \text{ with } e_2$

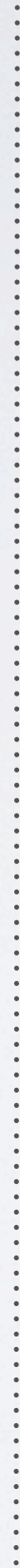
Typing

$\Sigma \mid \Gamma \vdash_w e : \tau$ “e has type τ at w w.r.t. Σ and Γ ”

$\Gamma ::= \cdot \mid \Gamma, x : \tau$

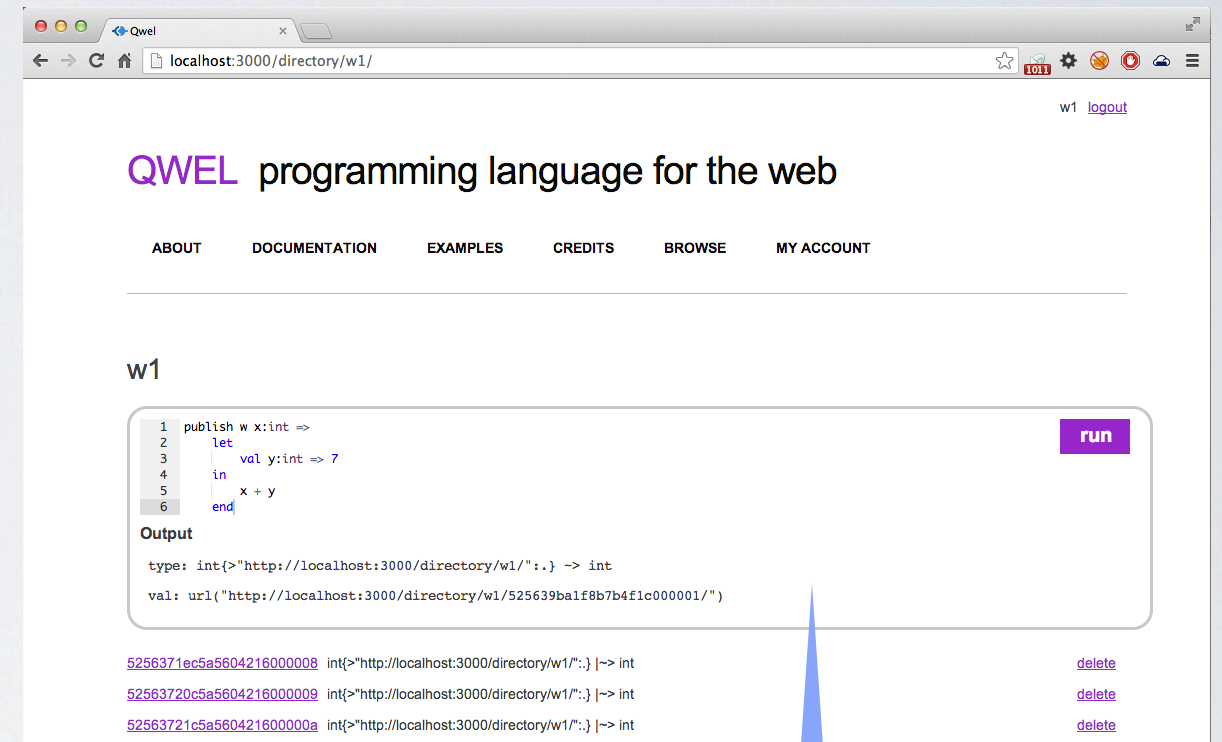
$\Sigma ::= \cdot \mid \Sigma, w/u : \tau \Rightarrow \tau$

Programming with Qwel



Programming with Qwel

@w₁



The screenshot shows a web browser window with the URL `localhost:3000/directory/w1/`. The page title is "QWEL programming language for the web". There is a navigation bar with links: ABOUT, DOCUMENTATION, EXAMPLES, CREDITS, BROWSE, and MY ACCOUNT. Below the navigation bar, there is a section titled "w1". Inside this section, there is a code editor with the following code:

```
1 publish w x:int =>
2   let
3     val y:int => 7
4   in
5     x + y
6   end
```

There is a "run" button to the right of the code editor. Below the code editor, there is an "Output" section showing the following text:

```
type: int{>"http://localhost:3000/directory/w1/" :> int
val: url("http://localhost:3000/directory/w1/525639balf8b7b4f1c000001/")
```

Below the output, there are three lines of text, each with a unique identifier and a "delete" link:

```
5256371ec5a5604216000008 int{>"http://localhost:3000/directory/w1/" :> int delete
52563720c5a5604216000009 int{>"http://localhost:3000/directory/w1/" :> int delete
52563721c5a560421600000a int{>"http://localhost:3000/directory/w1/" :> int delete
```

publish x : τ . e

Programming with Qwel

@w₁

Qwel

localhost:3000/directory/w1/

w1

logout

QWEL

programming language for the web

ABOUT

DOCUMENTATION

EXAMPLES

CREDITS

BROWSE

MY ACCOUNT

w1

1

publish w x:int =>

2

let

3

val y:int => 7

4

in

5

x + y

6

end

run

Output

type: int{>"http://localhost:3000/directory/w1/" :> } -> int

val: url("http://localhost:3000/directory/w1/525639balf8b7b4f1c000001/")

5256371ec5a5604216000008

int{>"http://localhost:3000/directory/w1/" :> } -> int

delete

52563720c5a5604216000009

int{>"http://localhost:3000/directory/w1/" :> } -> int

delete

52563721c5a560421600000a

int{>"http://localhost:3000/directory/w1/" :> } -> int

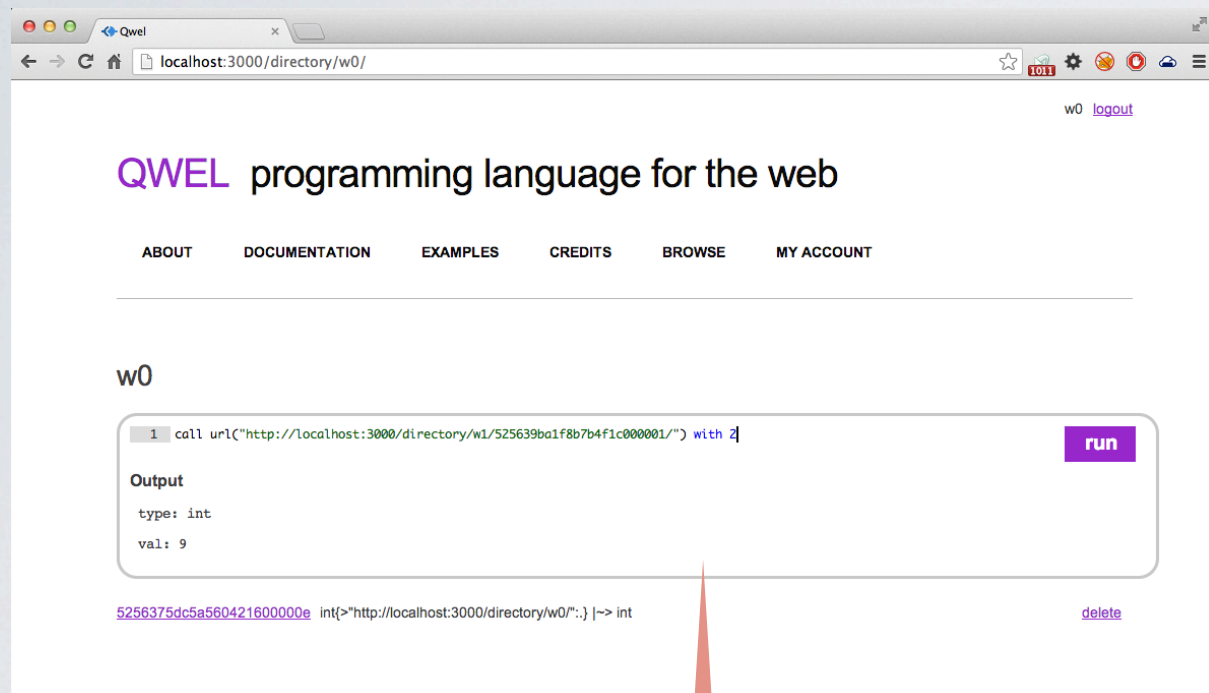
delete

w₁/u₁

publish x : τ . e

Programming with Qwel

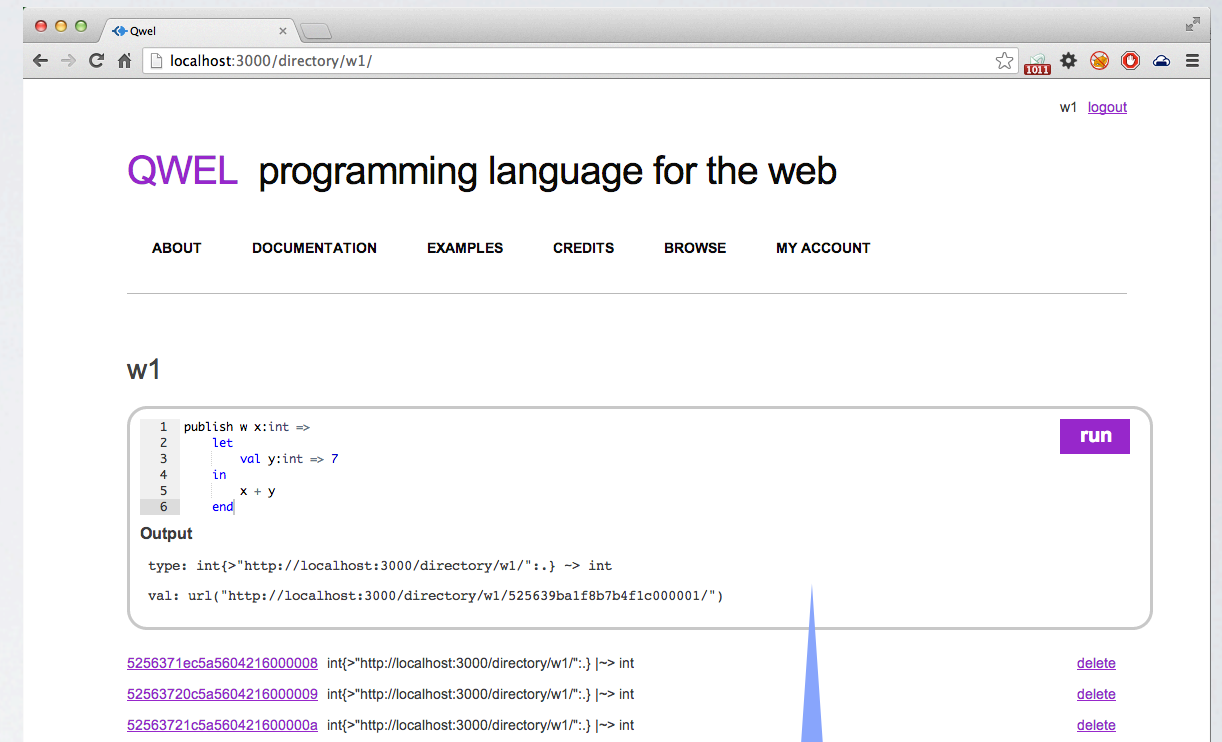
@w₀



The screenshot shows the Qwel web interface. The header includes the Qwel logo and navigation links: ABOUT, DOCUMENTATION, EXAMPLES, CREDITS, BROWSE, and MY ACCOUNT. The main content area is titled 'w0' and contains a code editor with a single line of code: `1 call url("http://localhost:3000/directory/w1/525639ba1f8b7b4f1c000001/") with 2`. Below the code editor is an 'Output' section showing the type `int` and the value `9`. At the bottom of the code editor is a 'run' button. Below the output section is a list of three entries, each with a unique ID and a 'delete' button.

call w₁/u₁ with input

@w₁



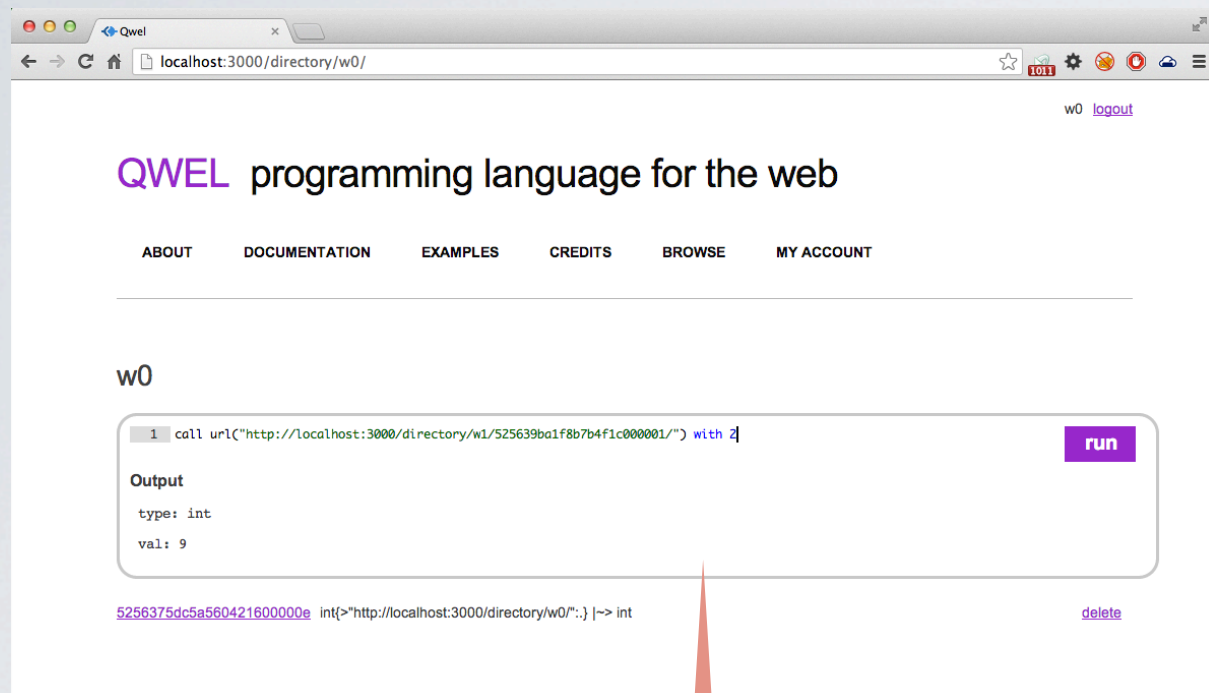
The screenshot shows the Qwel web interface. The header includes the Qwel logo and navigation links: ABOUT, DOCUMENTATION, EXAMPLES, CREDITS, BROWSE, and MY ACCOUNT. The main content area is titled 'w1' and contains a code editor with six lines of code: `1 publish w x:int =>`, `2 let`, `3 val y:int => 7`, `4 in`, `5 x + y`, and `6 end`. Below the code editor is an 'Output' section showing the type `int` and the value `url("http://localhost:3000/directory/w1/525639ba1f8b7b4f1c000001/")`. At the bottom of the code editor is a 'run' button. Below the output section is a list of three entries, each with a unique ID and a 'delete' button.

w₁/u₁

publish x : τ . e

Programming with Qwel

@w₀



The screenshot shows the Qwel web interface at `localhost:3000/directory/w0/`. The page title is "QWEL programming language for the web". The navigation bar includes links for ABOUT, DOCUMENTATION, EXAMPLES, CREDITS, BROWSE, and MY ACCOUNT. The main content area shows a code editor with the following code:

```
1 call url("http://localhost:3000/directory/w1/525639ba1f8b7b4f1c000001/") with 2
```

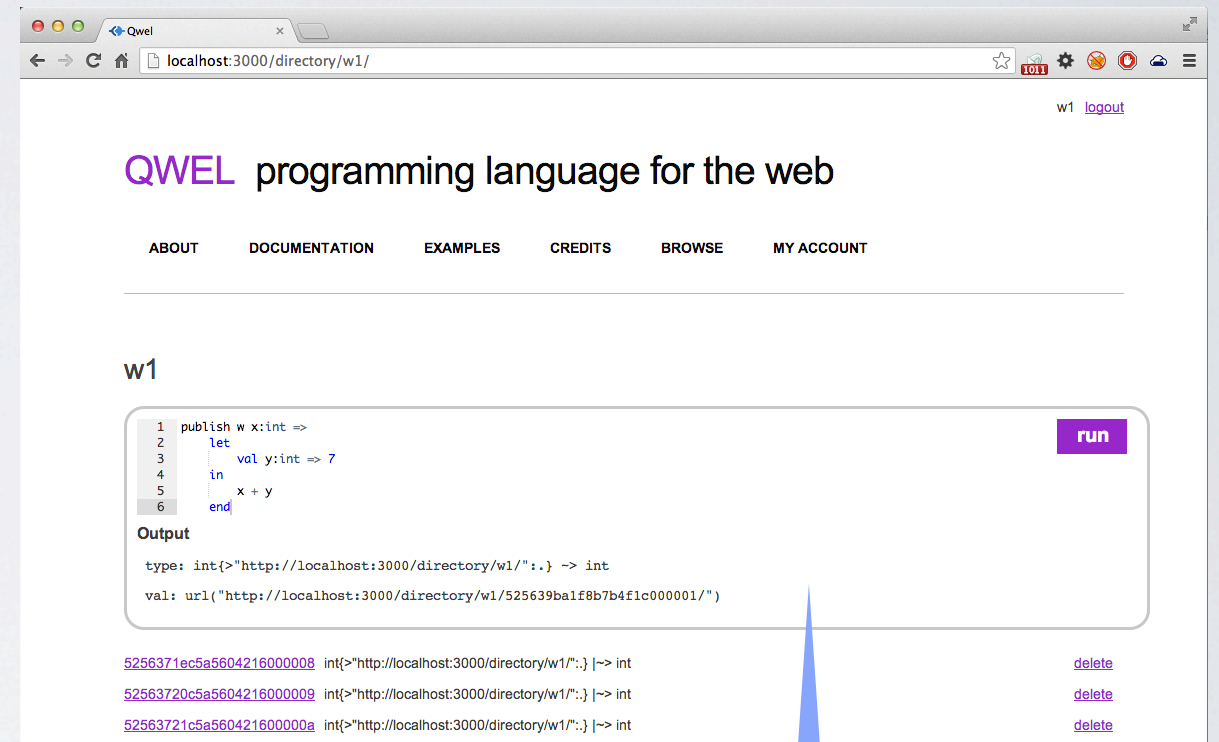
Below the code editor, the output is displayed:

```
type: int
val: 9
```

At the bottom of the code editor, there is a URL: `5256375dc5a560421600000e int{>"http://localhost:3000/directory/w0/":.} |-> int` and a "delete" button.

call w₁/u₁ with input

@w₁



The screenshot shows the Qwel web interface at `localhost:3000/directory/w1/`. The page title is "QWEL programming language for the web". The navigation bar includes links for ABOUT, DOCUMENTATION, EXAMPLES, CREDITS, BROWSE, and MY ACCOUNT. The main content area shows a code editor with the following code:

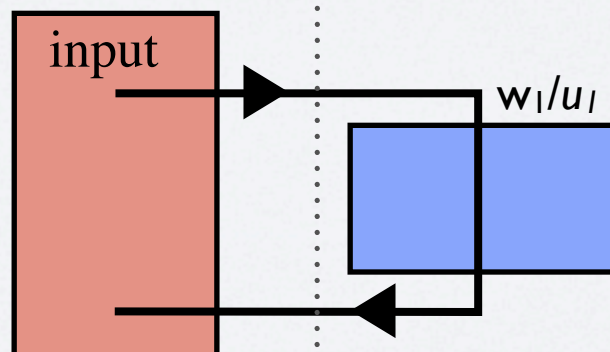
```
1 publish w x:int =>
2   let
3     val y:int => 7
4   in
5     x + y
6   end
```

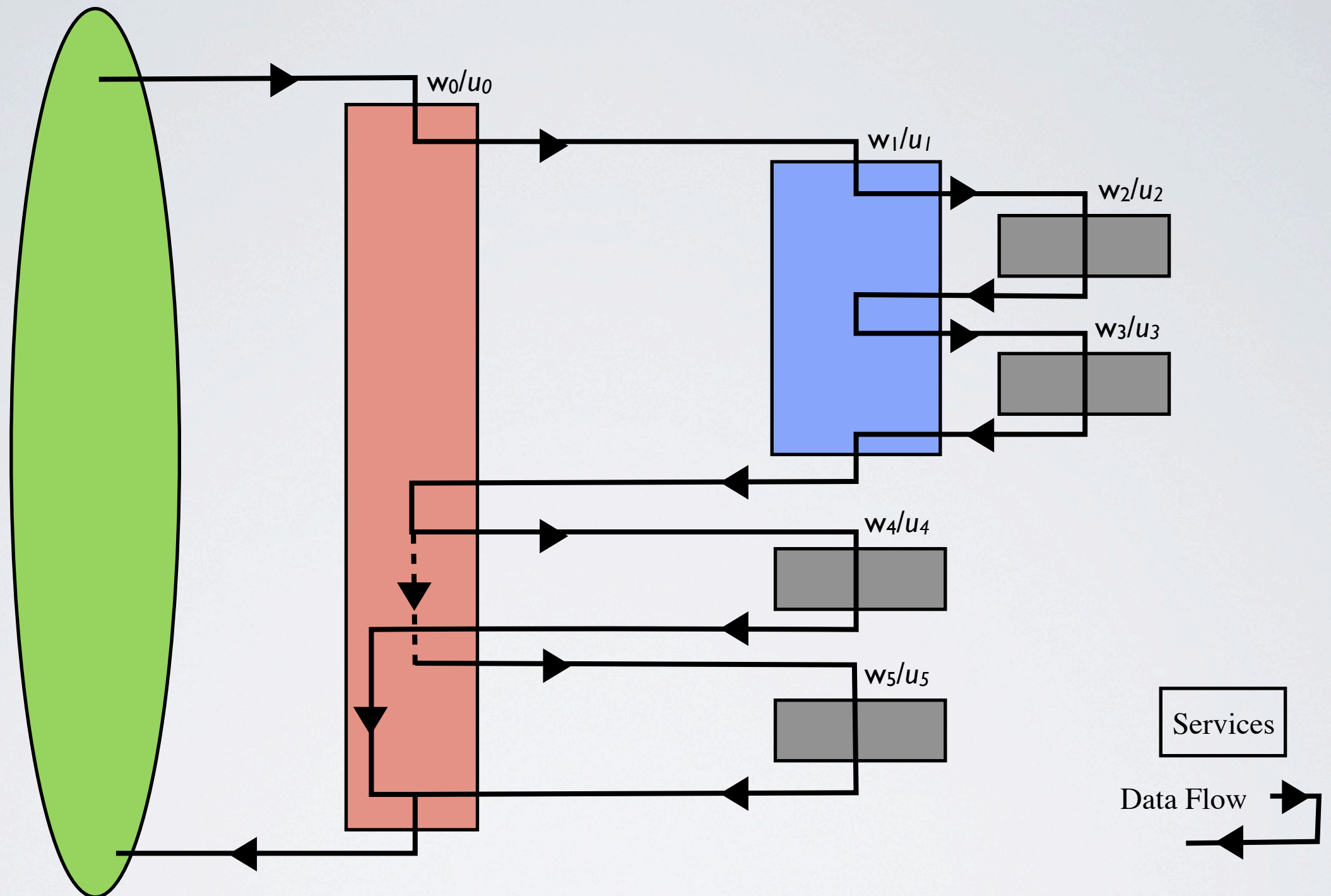
Below the code editor, the output is displayed:

```
type: int{>"http://localhost:3000/directory/w1/":.} -> int
val: url("http://localhost:3000/directory/w1/525639ba1f8b7b4f1c000001/")
```

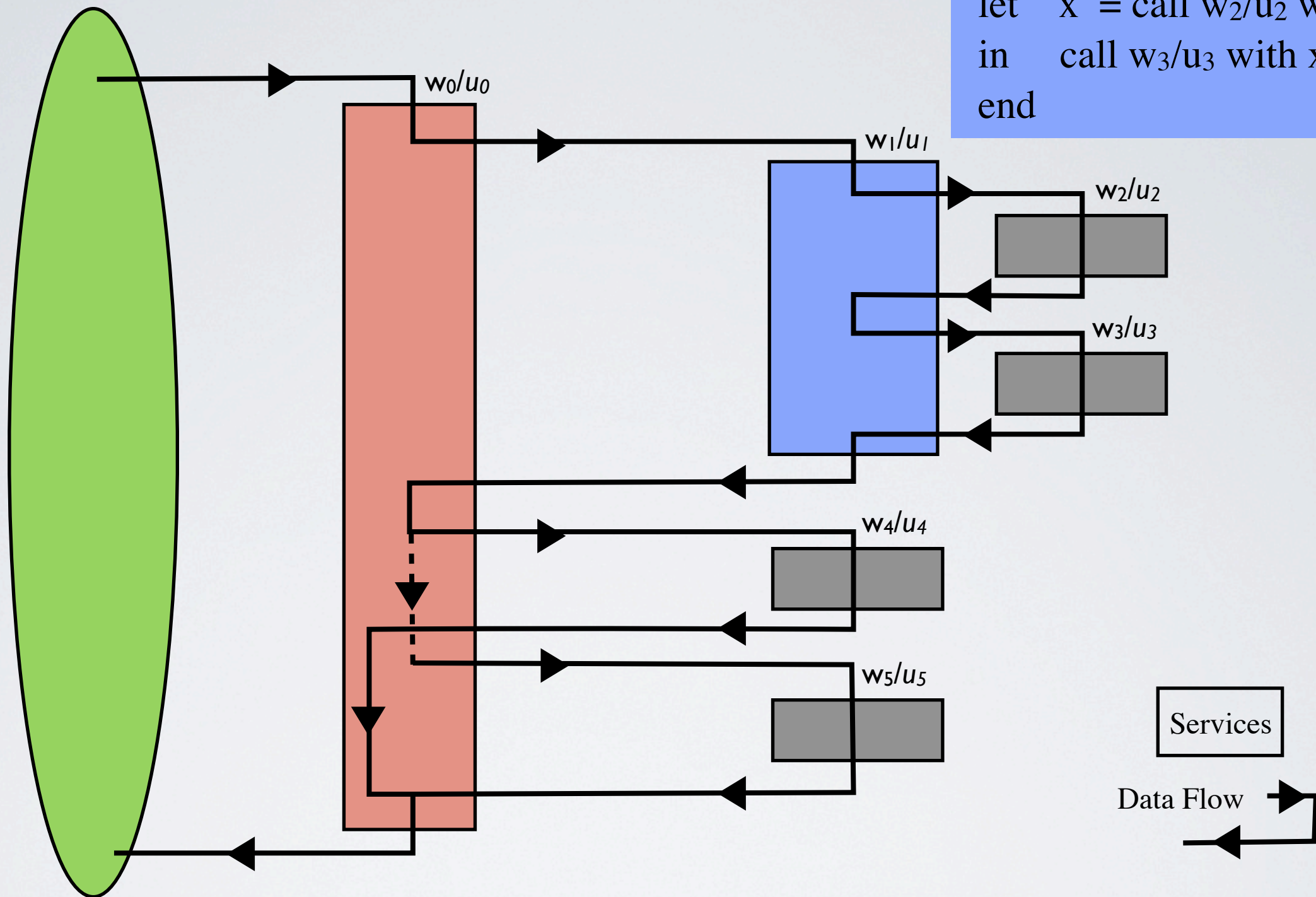
At the bottom of the code editor, there are three URLs: `5256371ec5a5604216000008 int{>"http://localhost:3000/directory/w1/":.} |-> int`, `52563720c5a5604216000009 int{>"http://localhost:3000/directory/w1/":.} |-> int`, and `52563721c5a560421600000a int{>"http://localhost:3000/directory/w1/":.} |-> int`, each with a "delete" button.

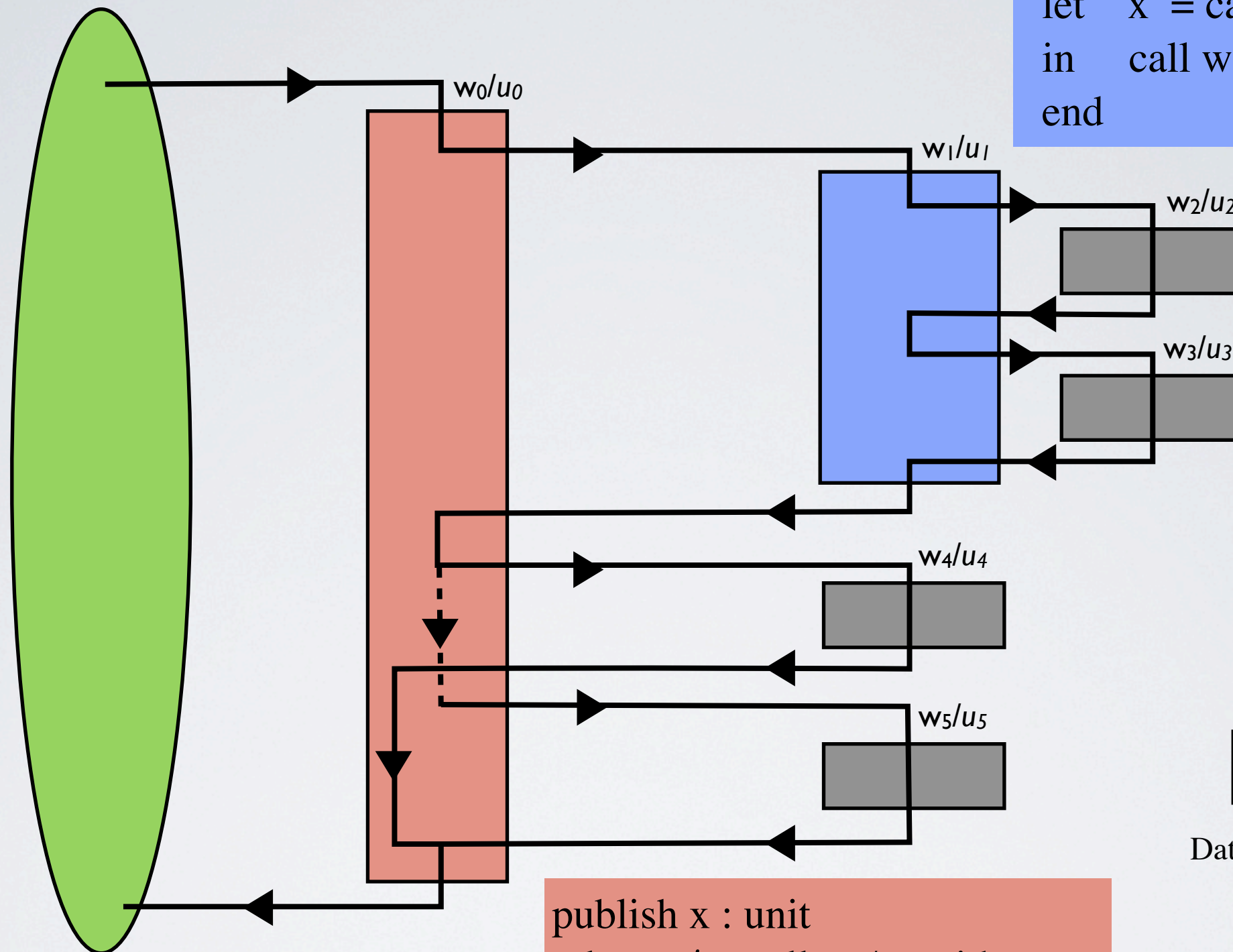
publish $x : \tau . e$





publish x : unit.
let x' = call w₂/u₂ with x
in call w₃/u₃ with x'
end





```

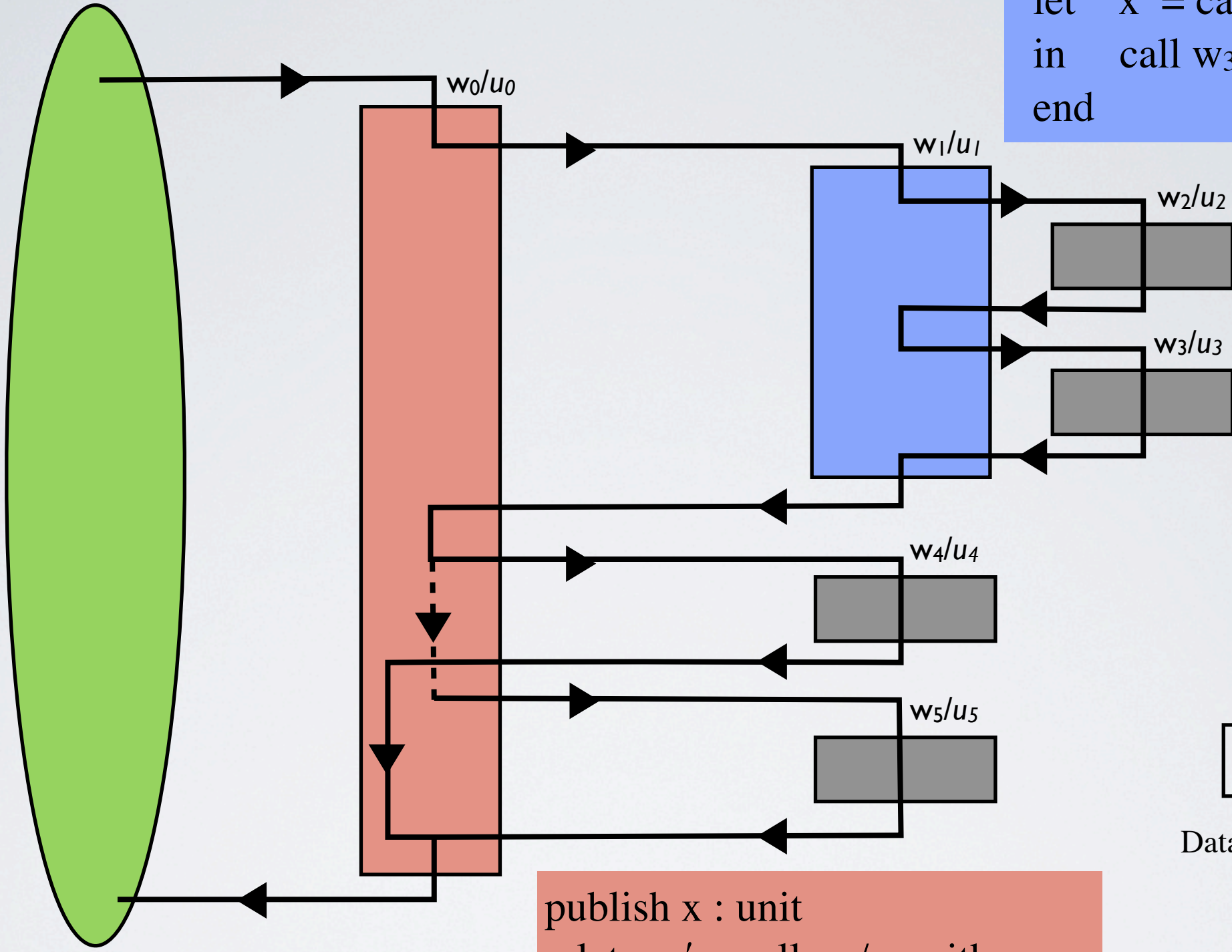
publish x : unit.
  let  x' = call w2/u2 with x
  in   call w3/u3 with x'
end
  
```

```

publish x : unit
  let  x' = call w1/u1 with x
      x'' = call w4/u4 with x'
      x''' = call w5/u5 with x'
  in   ⟨x'', x'''⟩
end
  
```

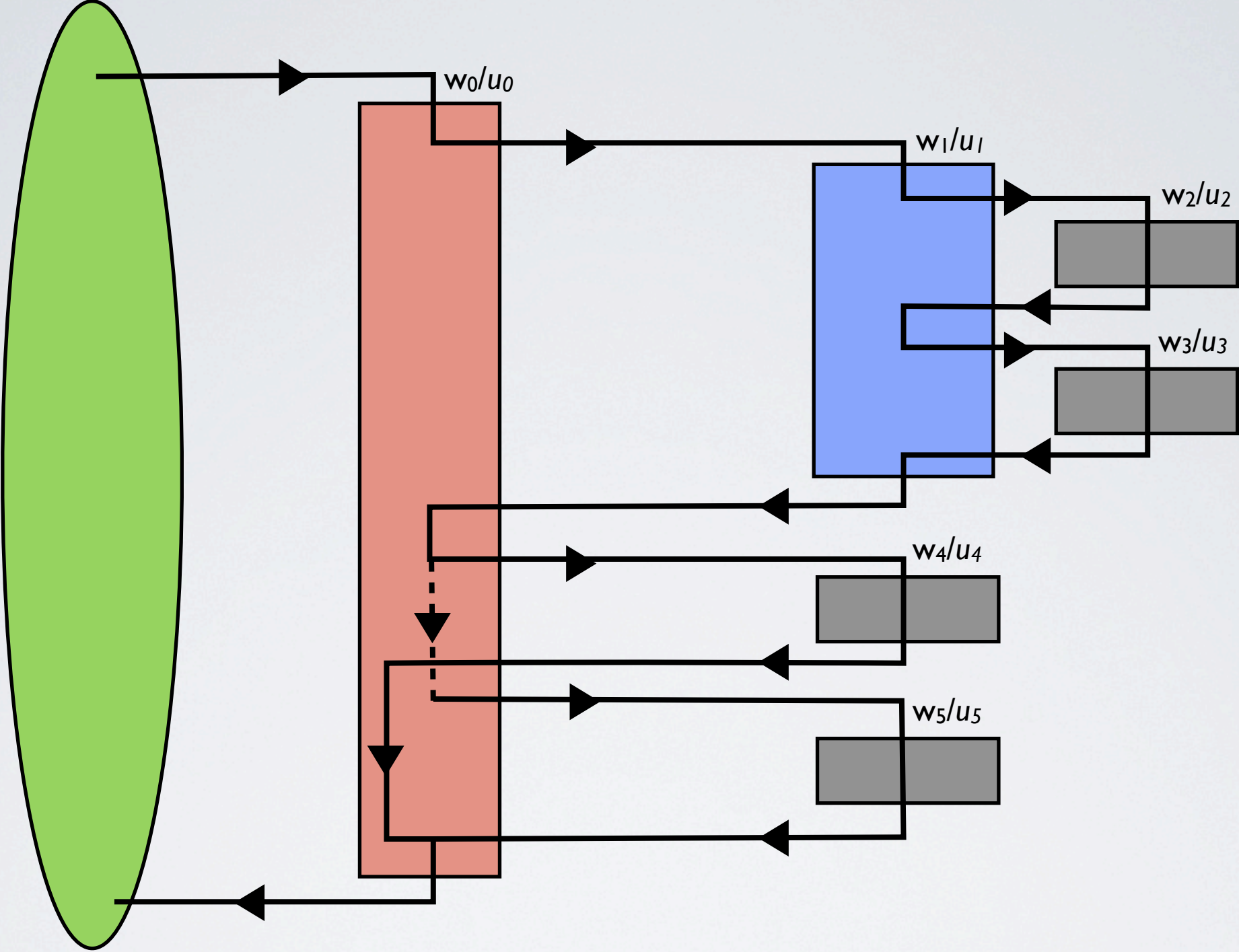
λ input:unit.(call w_0/u_0 with input)

publish $x : \text{unit}$.
let $x' = \text{call } w_2/u_2 \text{ with } x$
in $\text{call } w_3/u_3 \text{ with } x'$
end



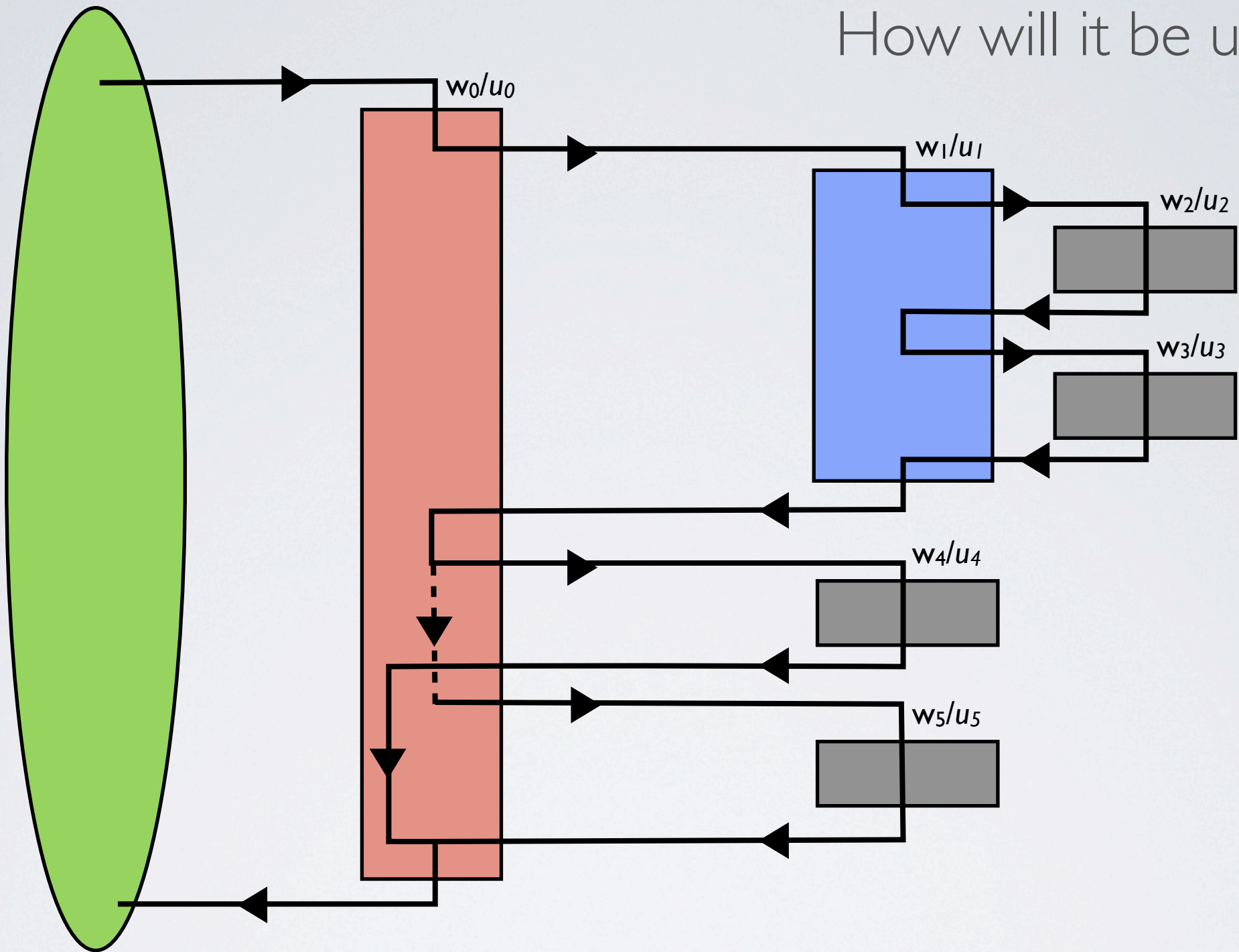
publish $x : \text{unit}$
let $x' = \text{call } w_1/u_1 \text{ with } x$
 $x'' = \text{call } w_4/u_4 \text{ with } x'$
 $x''' = \text{call } w_5/u_5 \text{ with } x'$
in $\langle x'', x''' \rangle$
end

λ input:unit.(call w_0/u_0 with input)



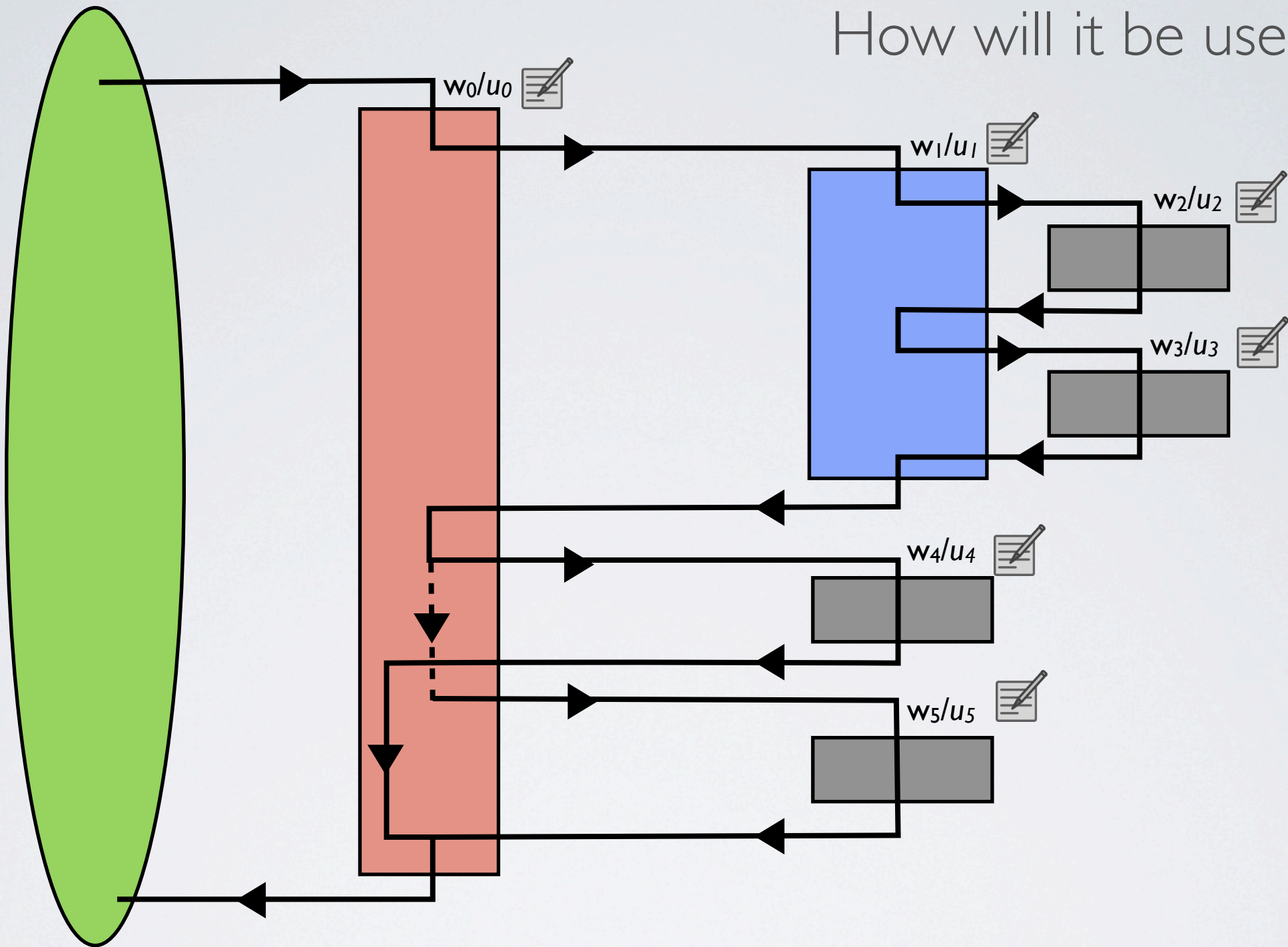
λ input:unit.(call w_0/u_0 with input)

Where is **input** going to?
How will it be used?



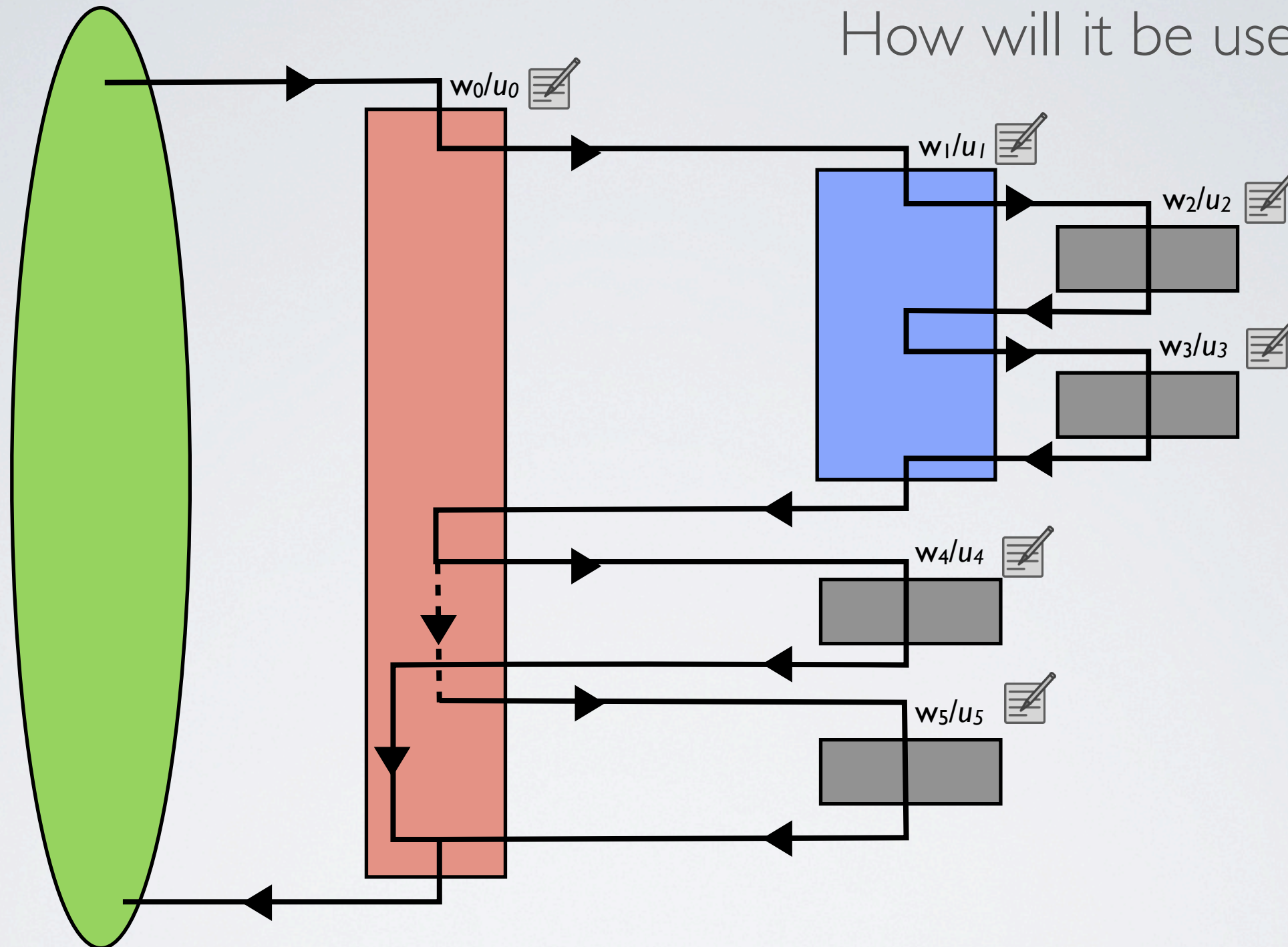
λ input:unit.(call w_0/u_0 with input)

Where is **input** going to?
How will it be used?



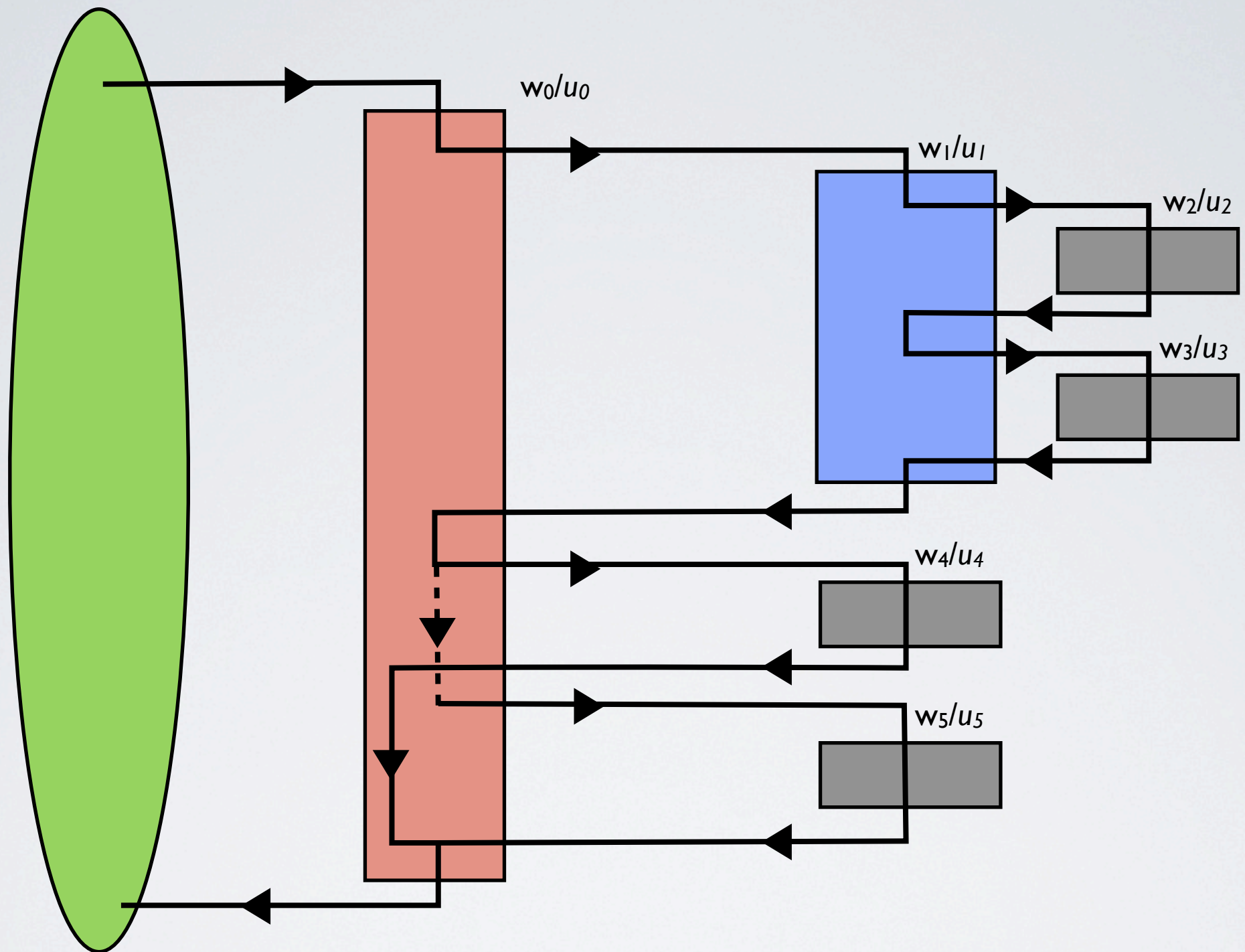
$\lambda \text{ input:unit.}(\text{call } w_0/u_0 \text{ with input})$

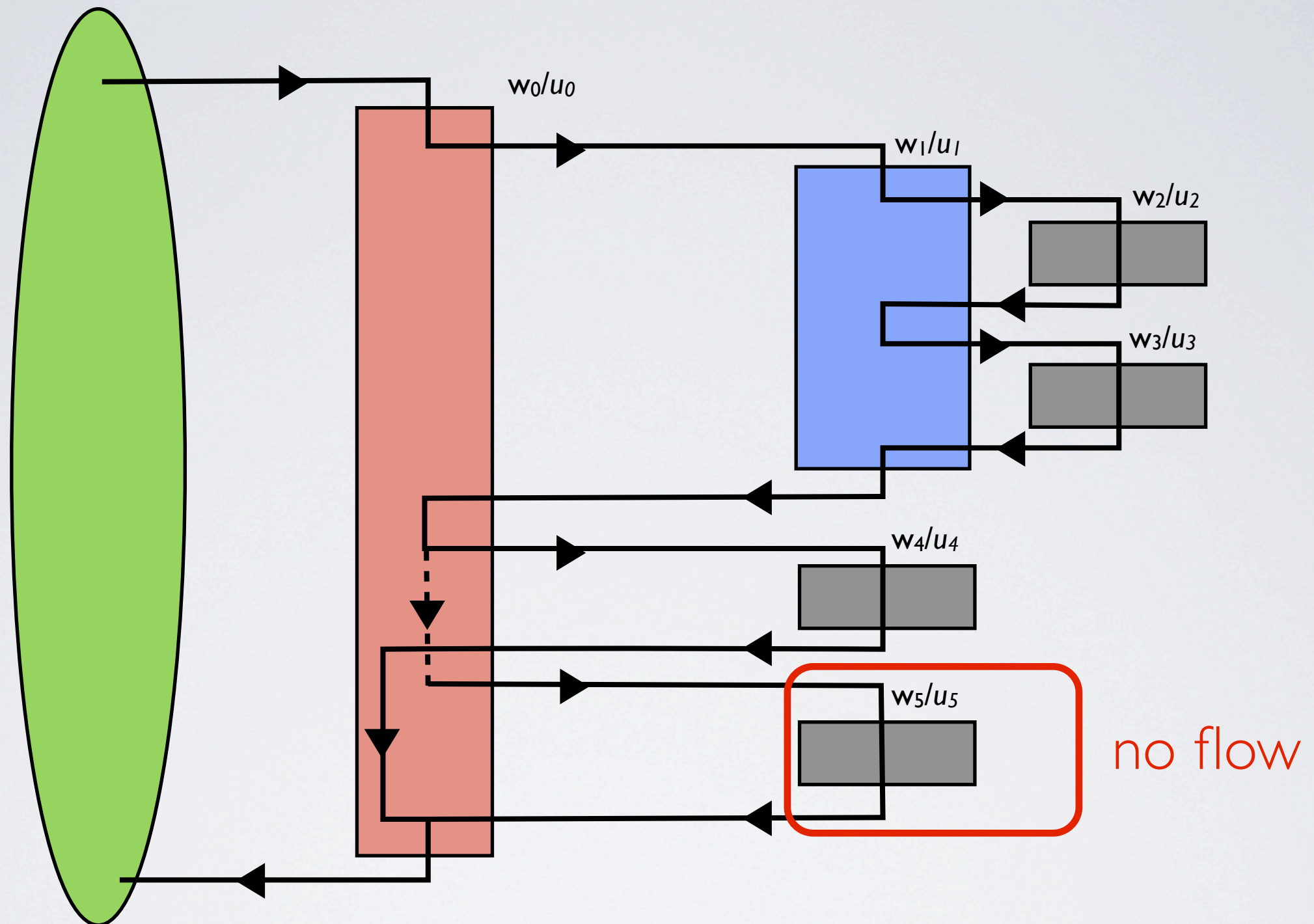
Where is **input** going to?
How will it be used?

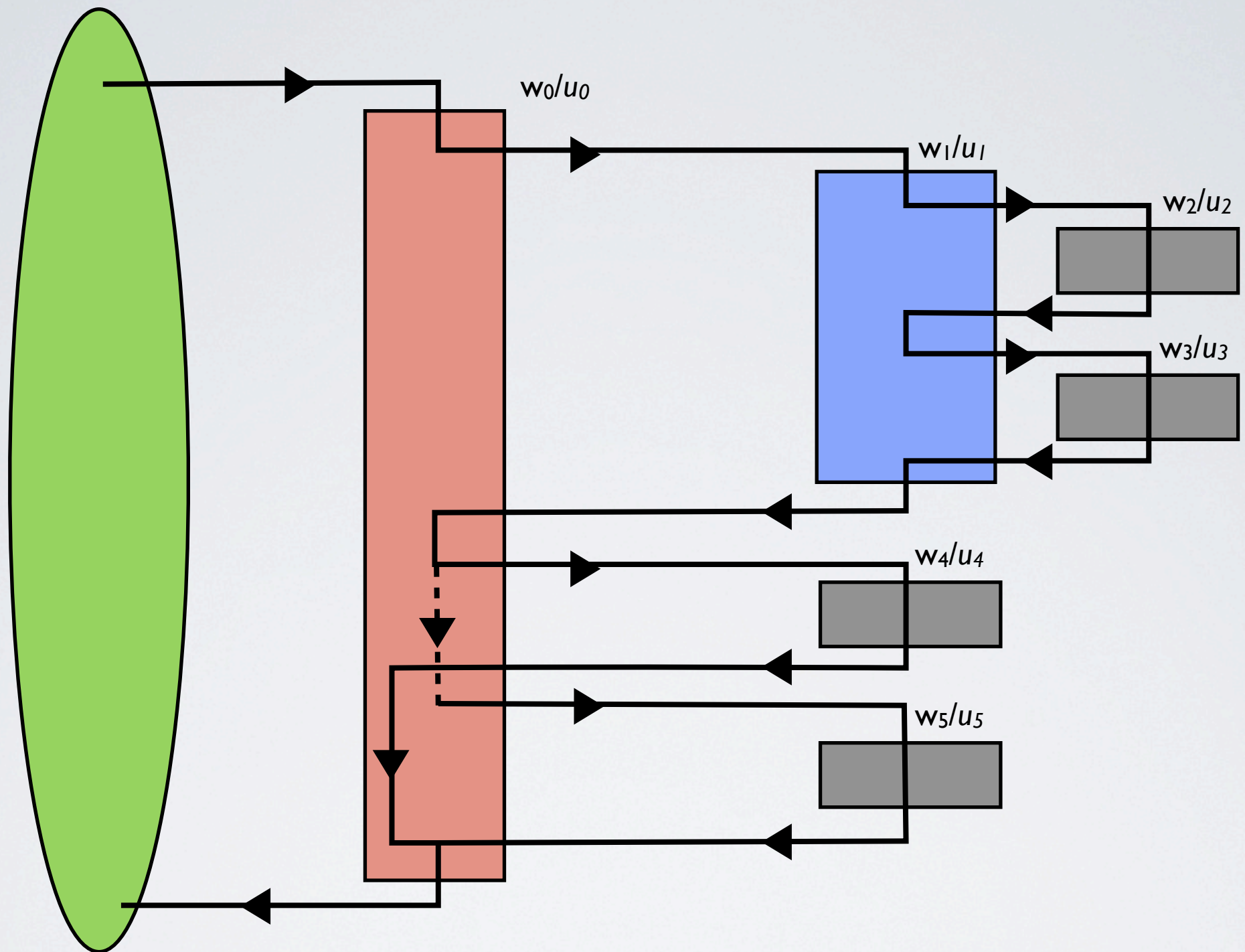


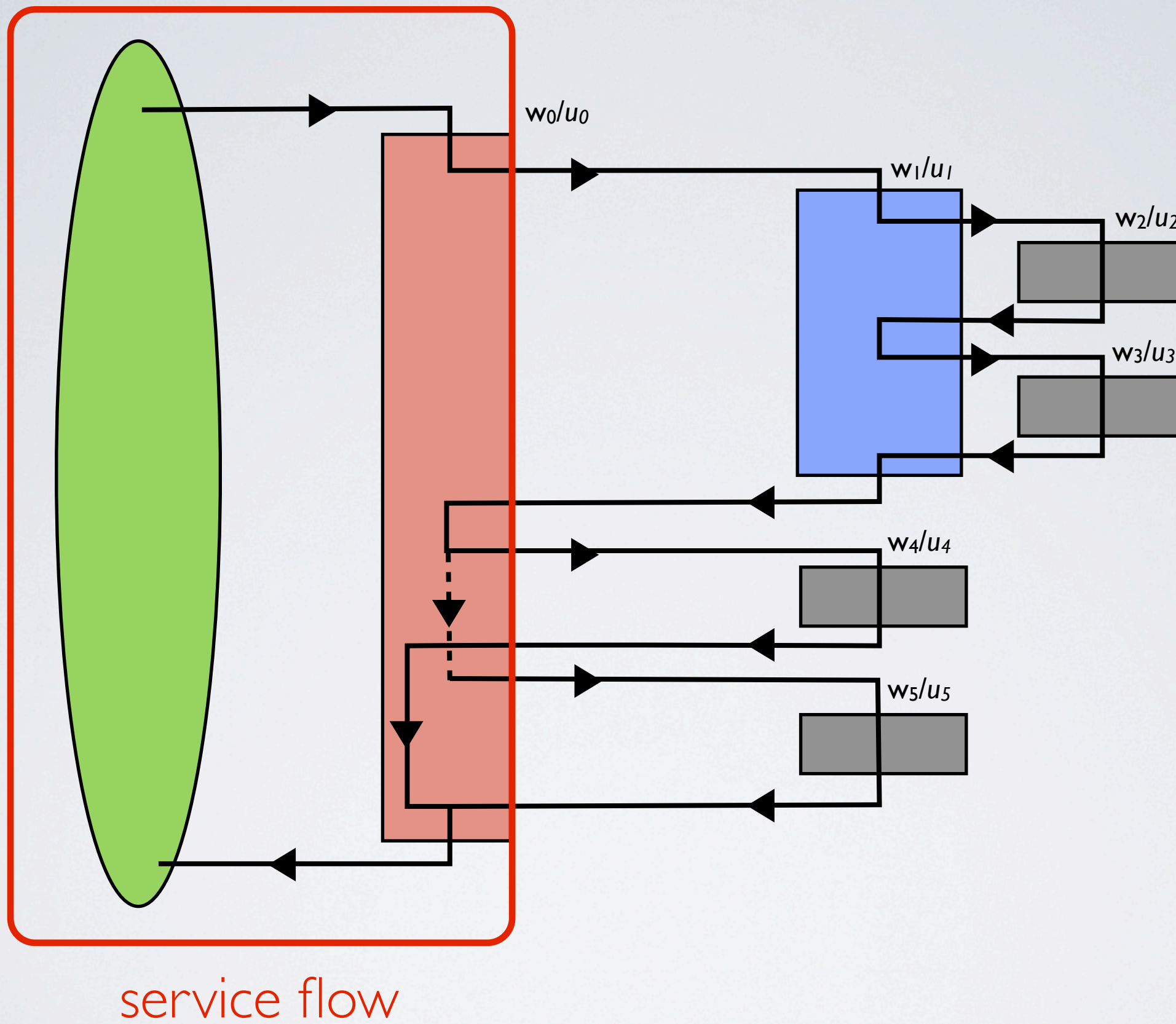
Can we get a symbolic expression that represents how **input** is used?

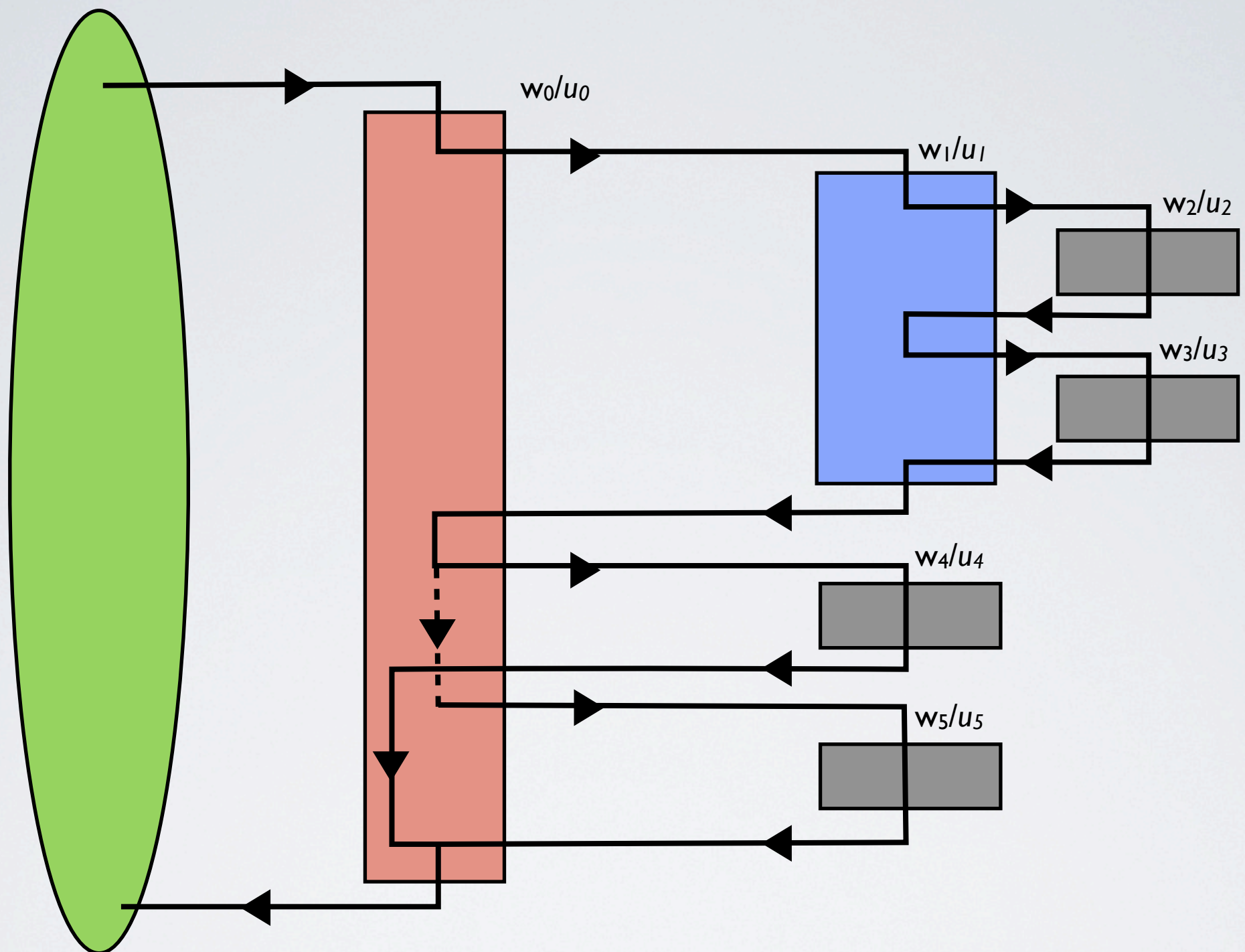
➡ The **data flow** of input

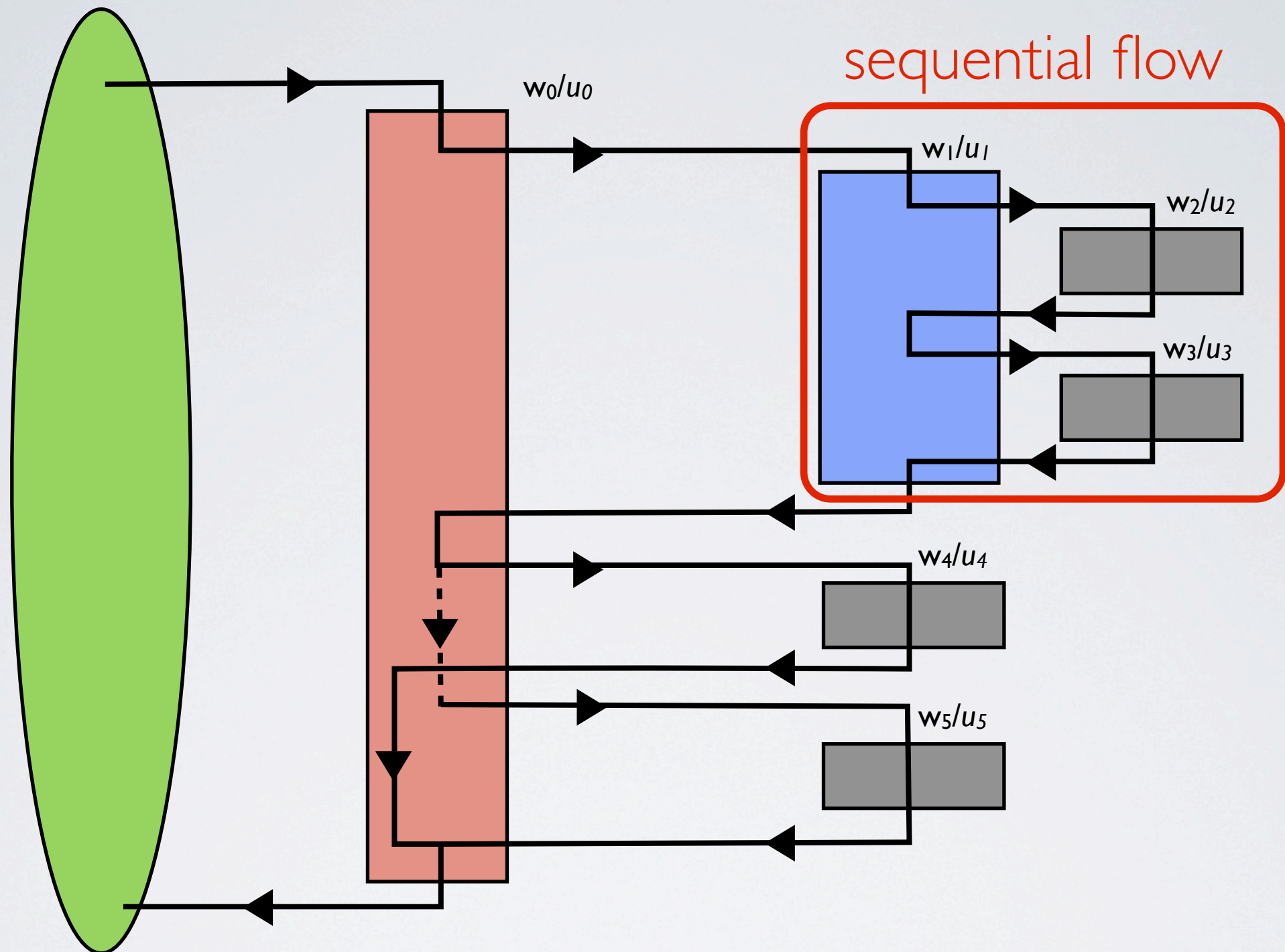


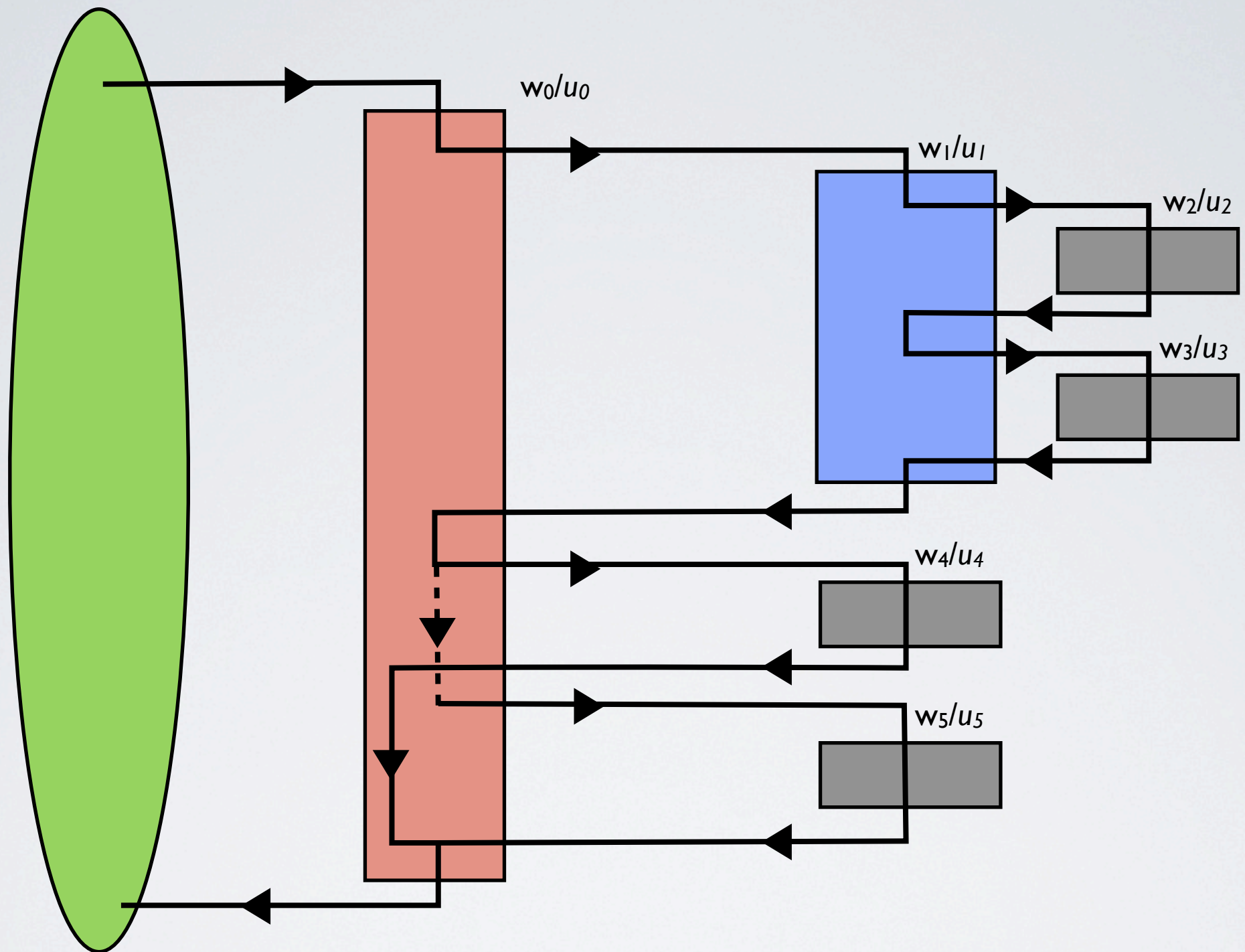


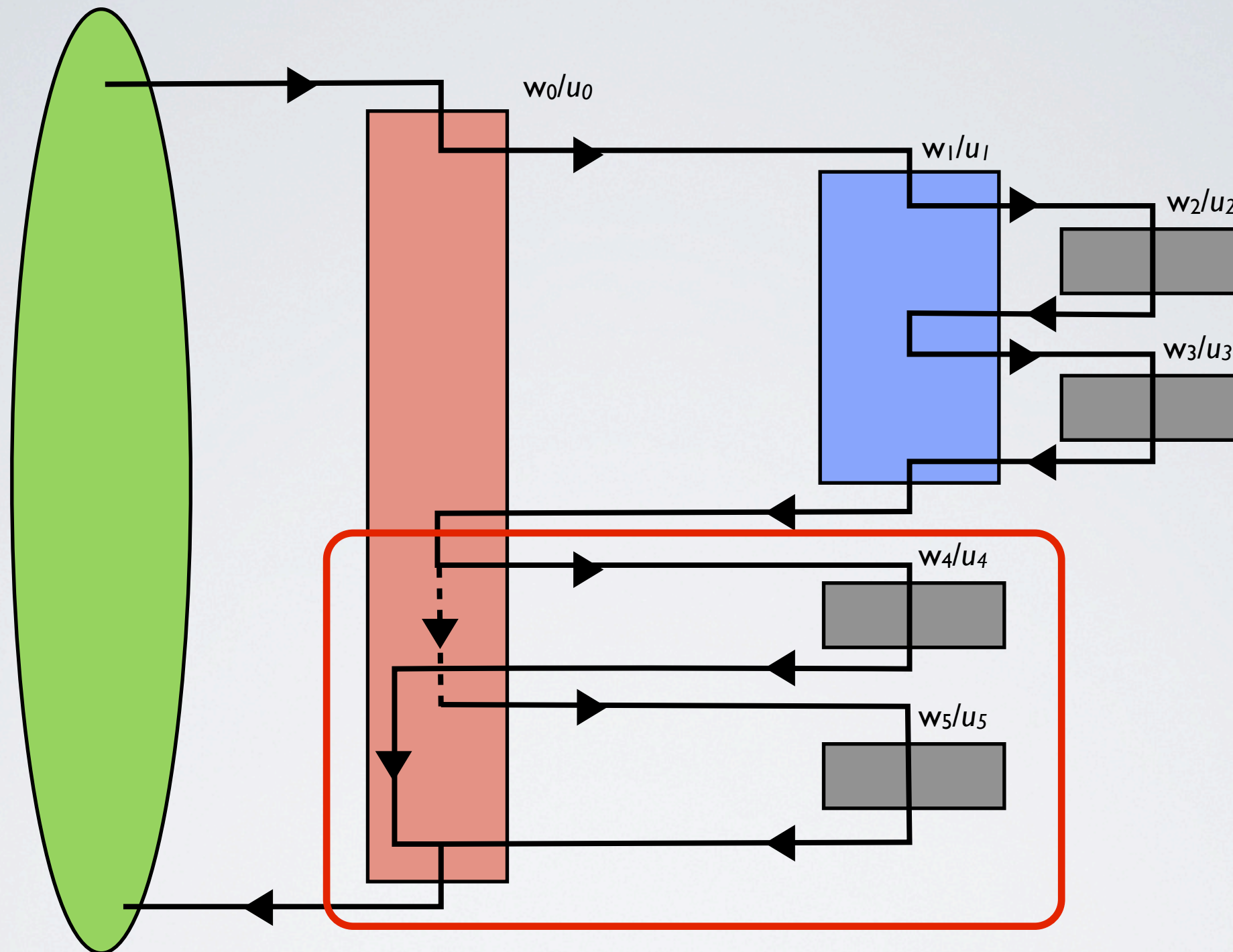












parallel flow

Qwel with Flow

Fragment of the language syntax

Types $\tau ::= \text{unit} \mid \tau \times \tau' \mid \tau \{\mu\} \rightarrow \tau' \mid \tau \{\mu\} \Rightarrow \tau'$

Expressions $e ::= x \mid \lambda x:\tau.e \mid (e_1 e_2) \mid \langle e_1, e_2 \rangle \mid \text{fst } e \mid \text{snd } e \mid ()$
 $\mid w/u \mid \text{publish } x:\tau.e \mid \text{call } e_1 \text{ with } e_2$

Flow $\mu ::= \bullet \mid w > \mu \mid \mu ; \mu' \mid \mu \parallel \mu'$

Qwel with Flow

Fragment of the language syntax

Types $\tau ::= \text{unit} \mid \tau \times \tau' \mid \tau \{\mu\} \rightarrow \tau' \mid \tau \{\mu\} \Rightarrow \tau'$

Expressions $e ::= x \mid \lambda x:\tau.e \mid (e_1 e_2) \mid \langle e_1, e_2 \rangle \mid \text{fst } e \mid \text{snd } e \mid ()$
 $\mid w/u \mid \text{publish } x:\tau.e \mid \text{call } e_1 \text{ with } e_2$

Flow $\mu ::= \bullet \mid w > \mu \mid \mu ; \mu' \mid \mu \parallel \mu'$



no

Qwel with Flow

Fragment of the language syntax

Types $\tau ::= \text{unit} \mid \tau \times \tau' \mid \tau \{\mu\} \rightarrow \tau' \mid \tau \{\mu\} \Rightarrow \tau'$

Expressions $e ::= x \mid \lambda x:\tau.e \mid (e_1 e_2) \mid \langle e_1, e_2 \rangle \mid \text{fst } e \mid \text{snd } e \mid ()$
 $\mid w/u \mid \text{publish } x:\tau.e \mid \text{call } e_1 \text{ with } e_2$

Flow $\mu ::= \bullet \mid w > \mu \mid \mu ; \mu' \mid \mu \parallel \mu'$

no

service

Qwel with Flow

Fragment of the language syntax

Types $\tau ::= \text{unit} \mid \tau \times \tau' \mid \tau \{\mu\} \rightarrow \tau' \mid \tau \{\mu\} \Rightarrow \tau'$

Expressions $e ::= x \mid \lambda x:\tau.e \mid (e_1 e_2) \mid \langle e_1, e_2 \rangle \mid \text{fst } e \mid \text{snd } e \mid ()$
 $\mid w/u \mid \text{publish } x:\tau.e \mid \text{call } e_1 \text{ with } e_2$

Flow $\mu ::= \bullet \mid w > \mu \mid \mu ; \mu' \mid \mu \parallel \mu'$

no

service

composition

Qwel with Flow

Fragment of the language syntax

Types $\tau ::= \text{unit} \mid \tau \times \tau' \mid \tau \{\mu\} \rightarrow \tau' \mid \tau \{\mu\} \Rightarrow \tau'$

Expressions $e ::= x \mid \lambda x:\tau.e \mid (e_1 e_2) \mid \langle e_1, e_2 \rangle \mid \text{fst } e \mid \text{snd } e \mid ()$
 $\mid w/u \mid \text{publish } x:\tau.e \mid \text{call } e_1 \text{ with } e_2$

Flow $\mu ::= \bullet \mid w > \mu \mid \mu ; \mu' \mid \mu \parallel \mu'$

no

service

composition

parallel

Qwel with Flow

Fragment of the language syntax

Types $\tau ::= \text{unit} \mid \tau \times \tau' \mid \tau \{\mu\} \rightarrow \tau' \mid \tau \{\mu\} \Rightarrow \tau'$

Expressions $e ::= x \mid \lambda x:\tau.e \mid (e_1 e_2) \mid \langle e_1, e_2 \rangle \mid \text{fst } e \mid \text{snd } e \mid ()$
 $\mid w/u \mid \text{publish } x:\tau.e \mid \text{call } e_1 \text{ with } e_2$

Flow $\mu ::= \bullet \mid w > \mu \mid \mu ; \mu' \mid \mu \parallel \mu'$

no

service

composition

parallel

Typing

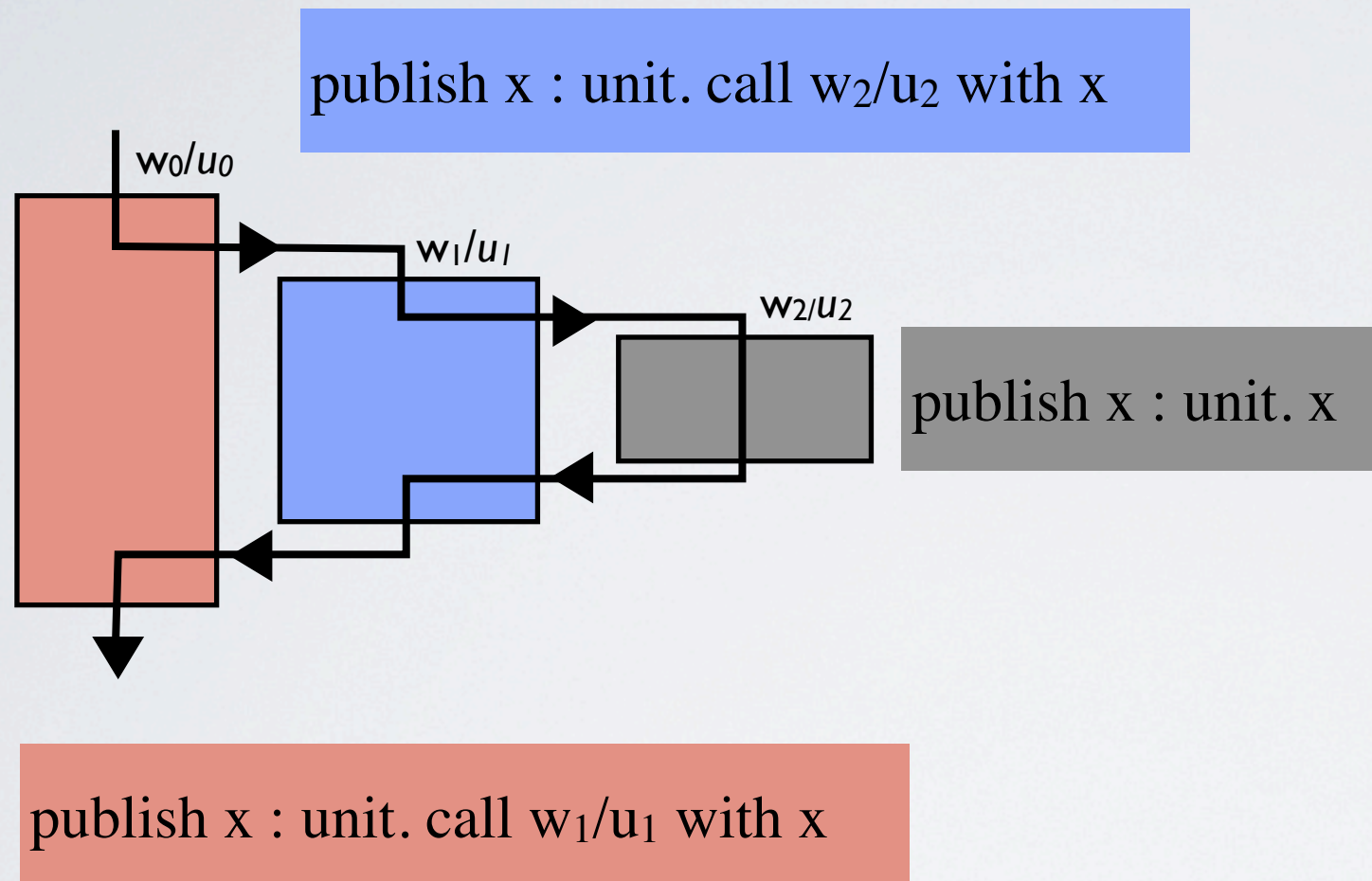
$\Sigma \mid \Gamma \vdash_w e : \tau$

“e has type τ at w w.r.t. Σ and Γ ”

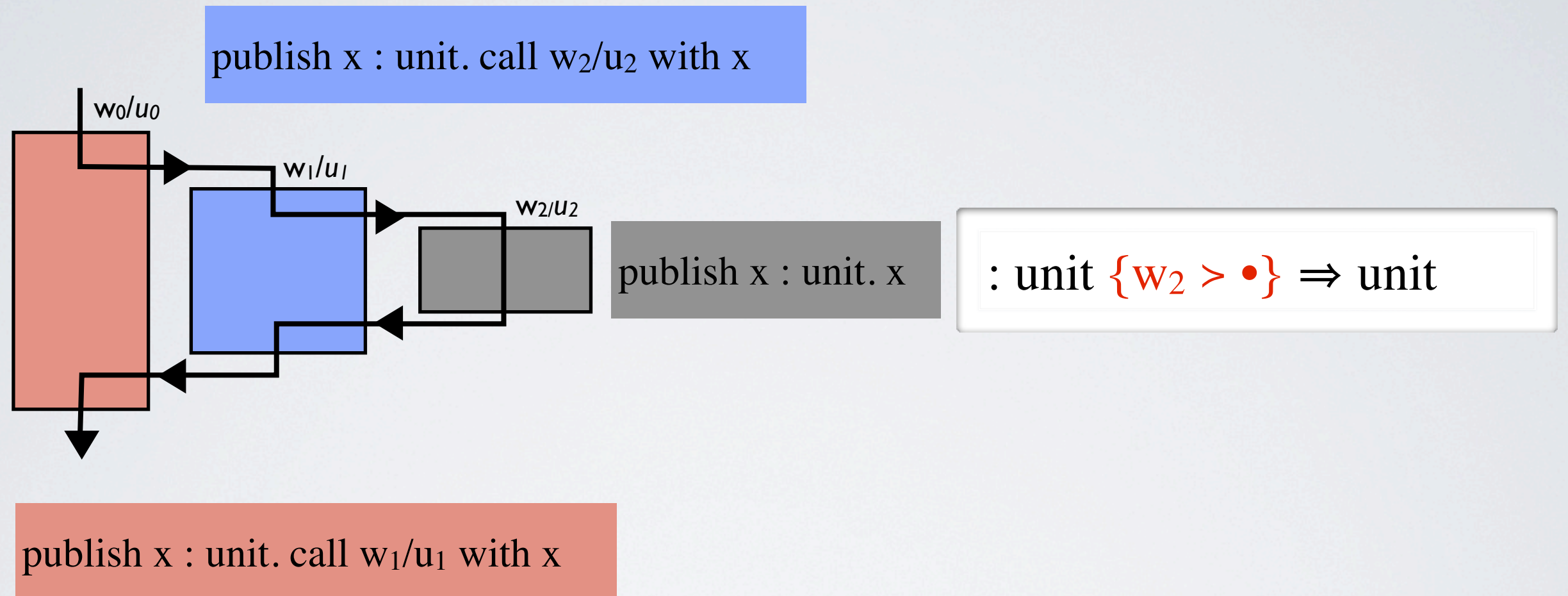
$\Gamma ::= \cdot \mid \Gamma, x : \tau \{\mu\}$

$\Sigma ::= \cdot \mid \Sigma, w/u : \tau \{\mu\} \Rightarrow \tau$

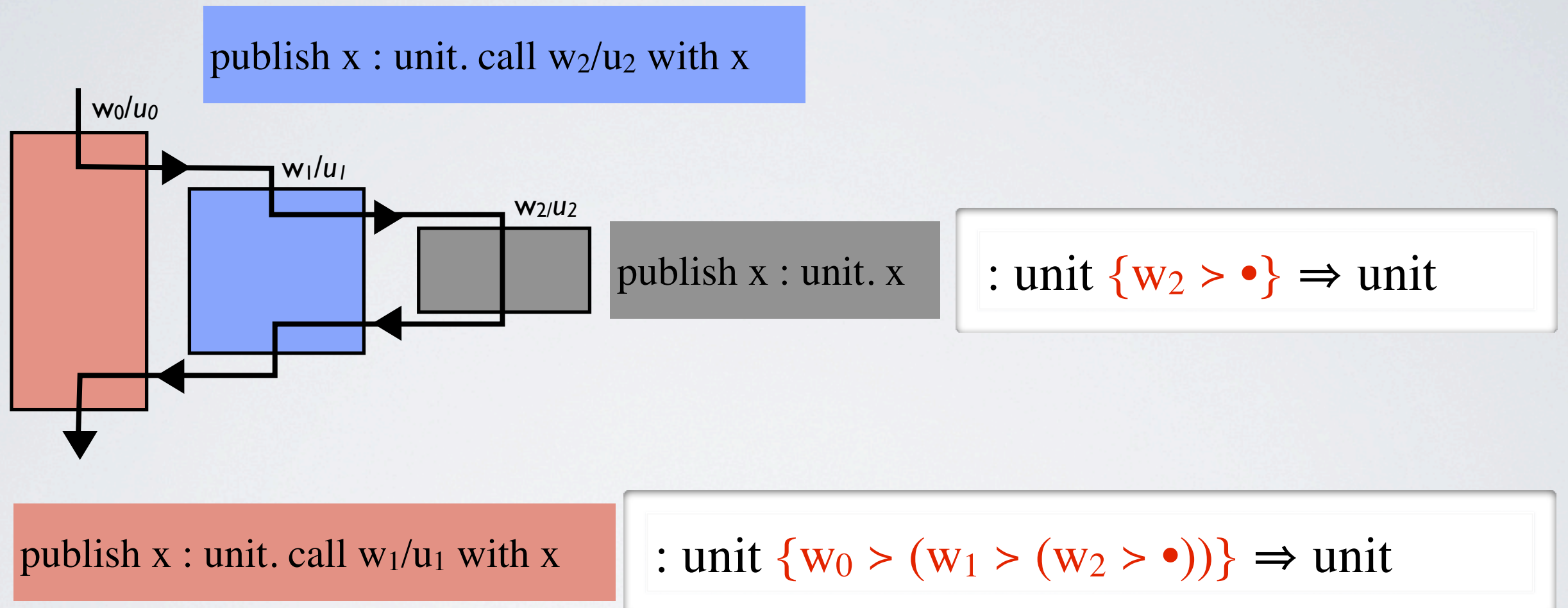
No Flow and Service flow



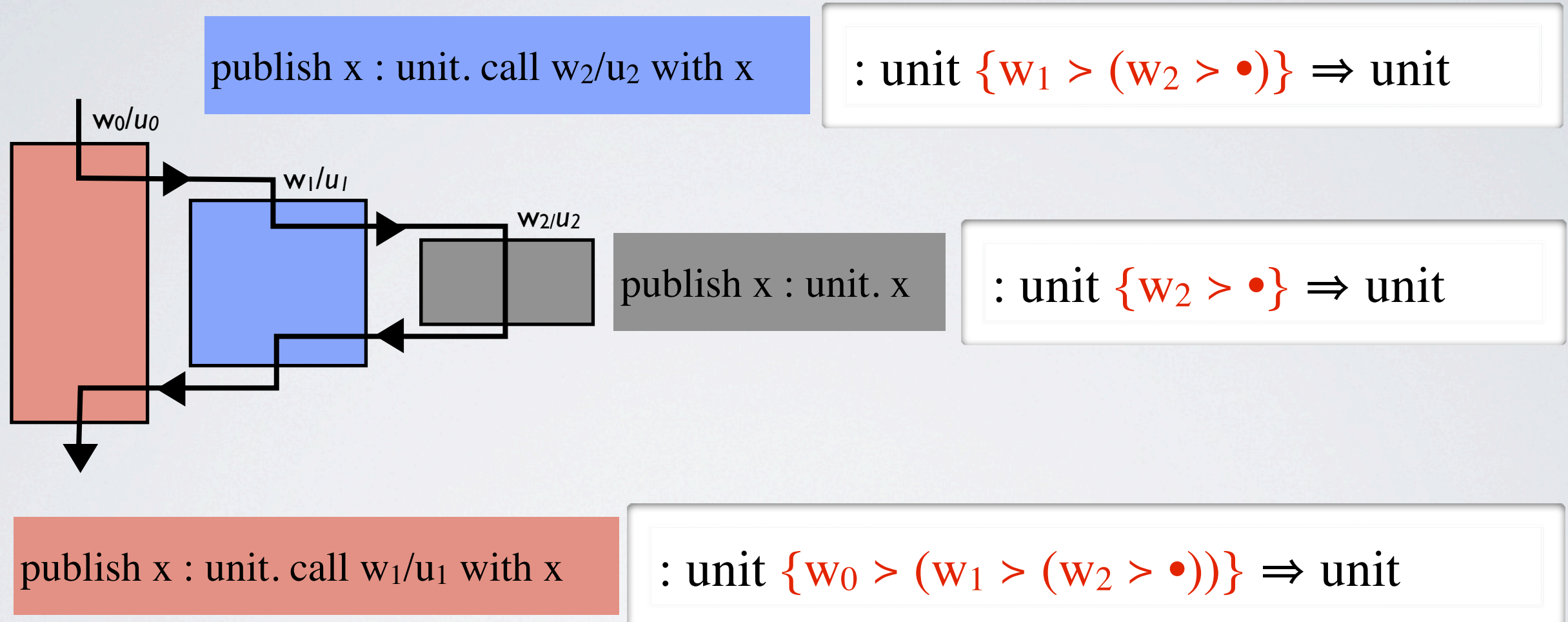
No Flow and Service flow



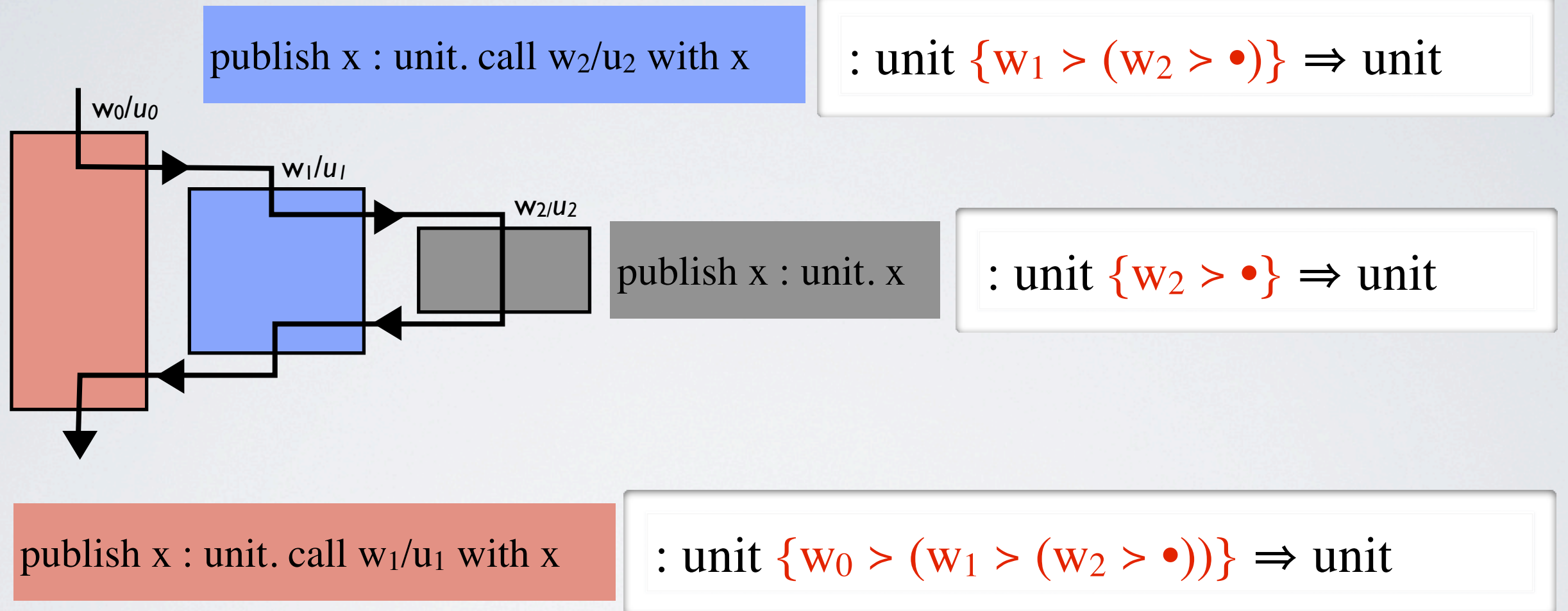
No Flow and Service flow



No Flow and Service flow

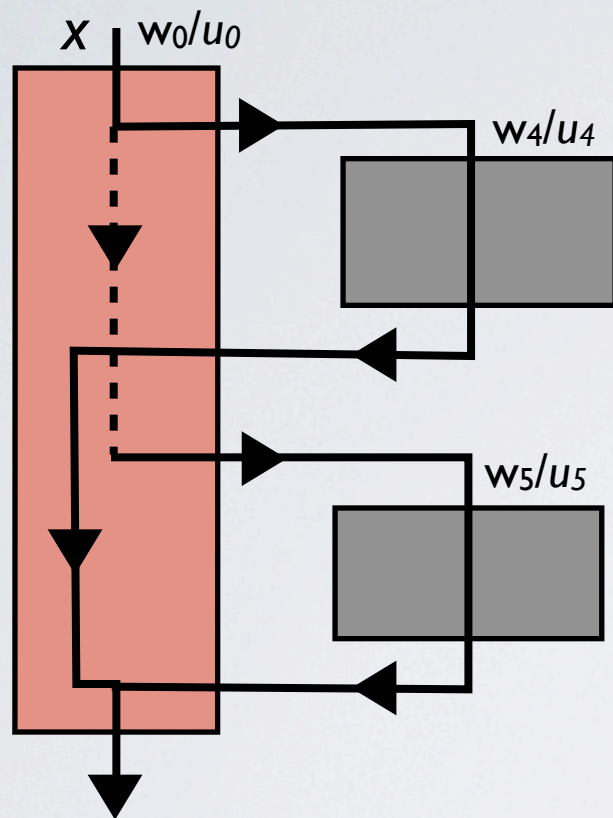


No Flow and Service flow



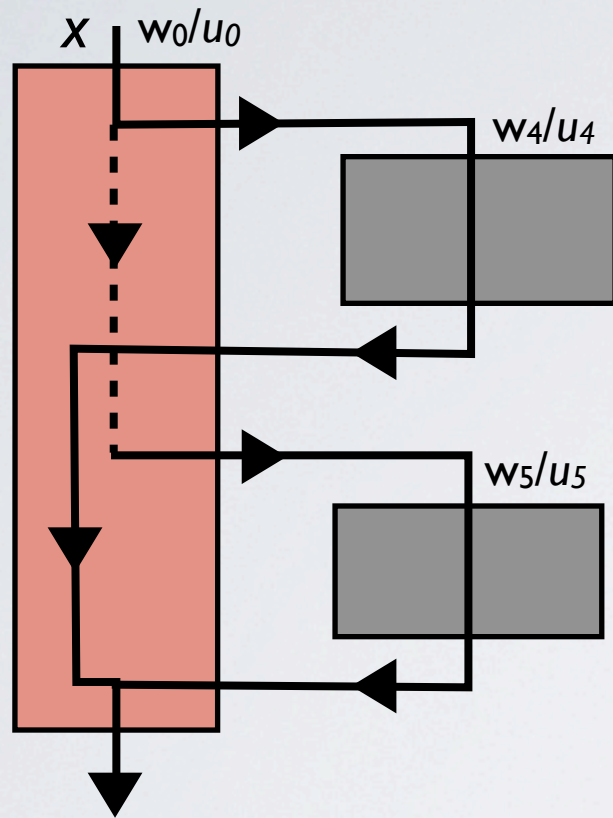
$$\frac{\Sigma \mid \Gamma, x : \tau \{ \mu \} \vdash_w e : \tau'}{\Sigma \mid \Gamma \vdash_w \text{publish } x : \tau . e : \tau \{ w > \mu \} \Rightarrow \tau}$$

Parallel flow



```
publish x : unit
  let  x' = call w4/u4 with x
      x'' = call w5/u5 with x
  in  ⟨x', x'⟩
end
```


Parallel flow

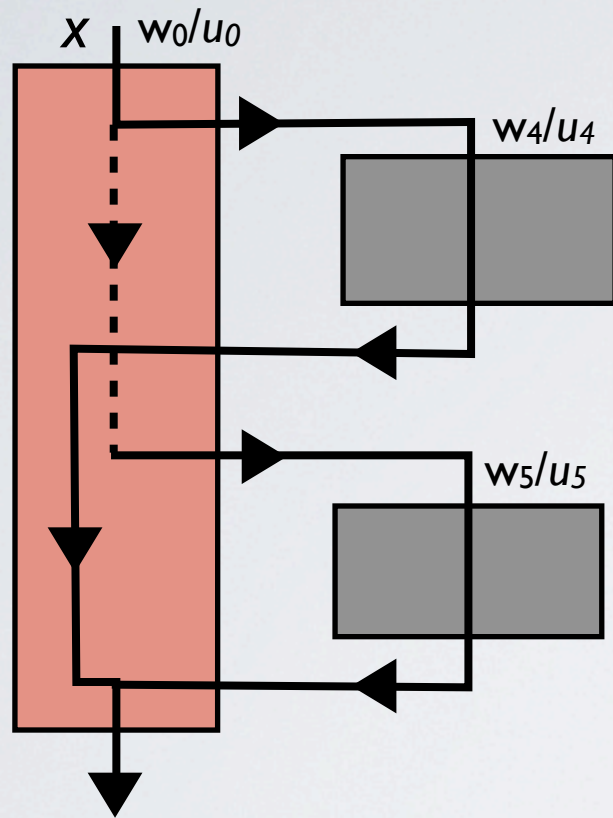


```

publish x : unit
  let  x' = call w4/u4 with x
      x'' = call w5/u5 with x
  in  ⟨x', x'⟩
end
    
```

$: \text{unit } \{w_0 > ((w_4 > \bullet) \parallel (w_5 > \bullet))\} \Rightarrow \text{unit}$

Parallel flow



```

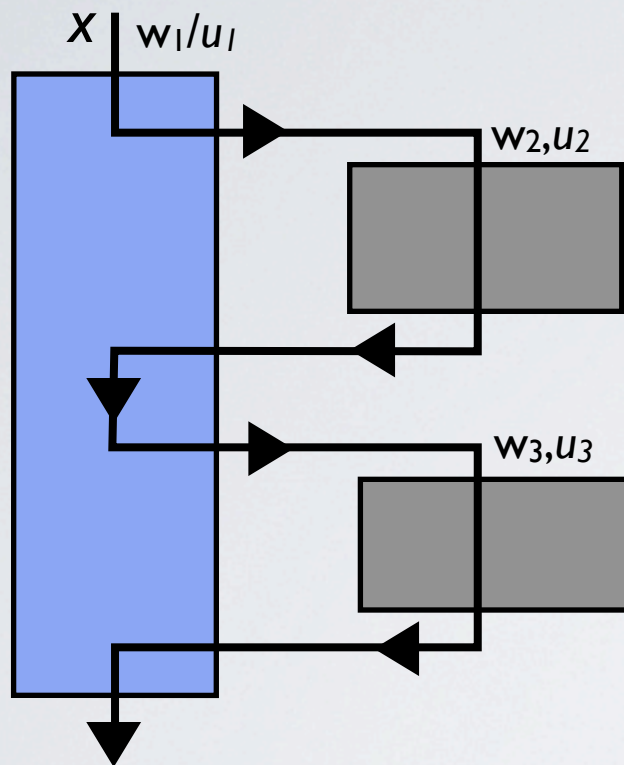
publish x : unit
  let  x' = call w4/u4 with x
      x'' = call w5/u5 with x
  in  <x', x''>
end

```

$: \text{unit } \{w_0 > ((w_4 > \bullet) \parallel (w_5 > \bullet))\} \Rightarrow \text{unit}$

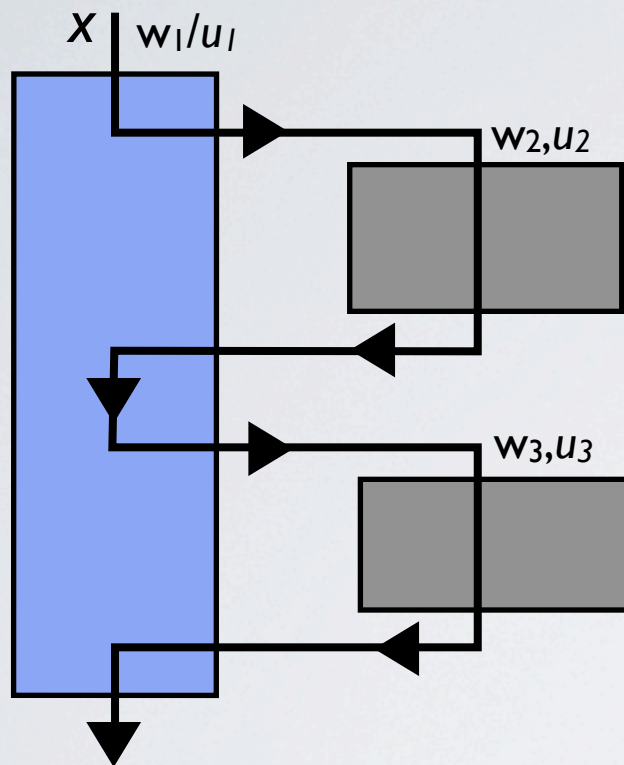
$$\frac{\Sigma \mid \Gamma \vdash_w e_1 : \tau \quad \Sigma \mid \Gamma' \vdash_w e_2 : \tau'}{\Sigma \mid (\Gamma \parallel \Gamma') \vdash_w \langle e_1, e_2 \rangle : \tau \times \tau'}$$

Sequential flow



```
publish x : unit
  let  x' = call w2/u2 with x
  in   call w3/u3 with x'
end
```

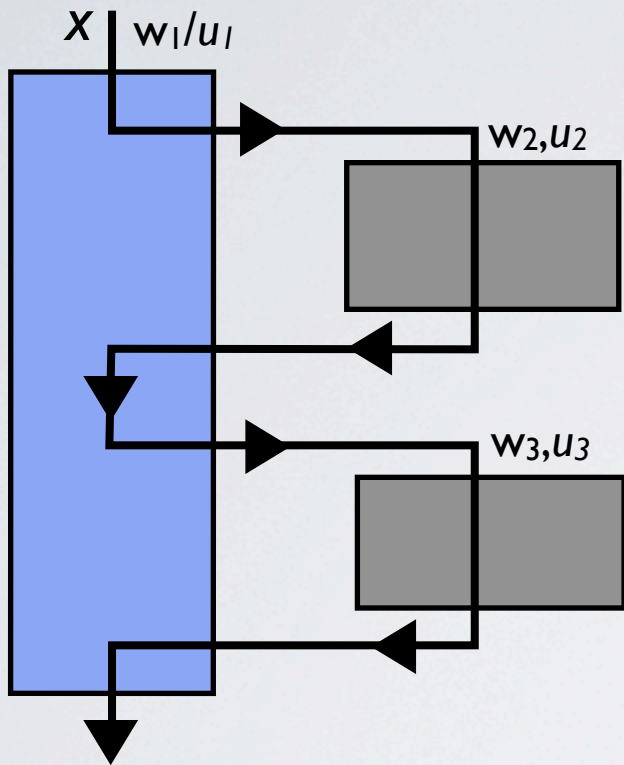
Sequential flow



```
publish x : unit
  let  x' = call w2/u2 with x
  in   call w3/u3 with x'
end
```

$: \text{unit } \{w_1 > ((w_2 > \bullet) ; (w_3 > \bullet))\} \Rightarrow \text{unit}$

Sequential flow



publish $x : \text{unit}$
 let $x' = \text{call } w_2/u_2 \text{ with } x$
 in $\text{call } w_3/u_3 \text{ with } x'$
 end

$: \text{unit } \{w_1 > ((w_2 > \bullet) ; (w_3 > \bullet))\} \Rightarrow \text{unit}$

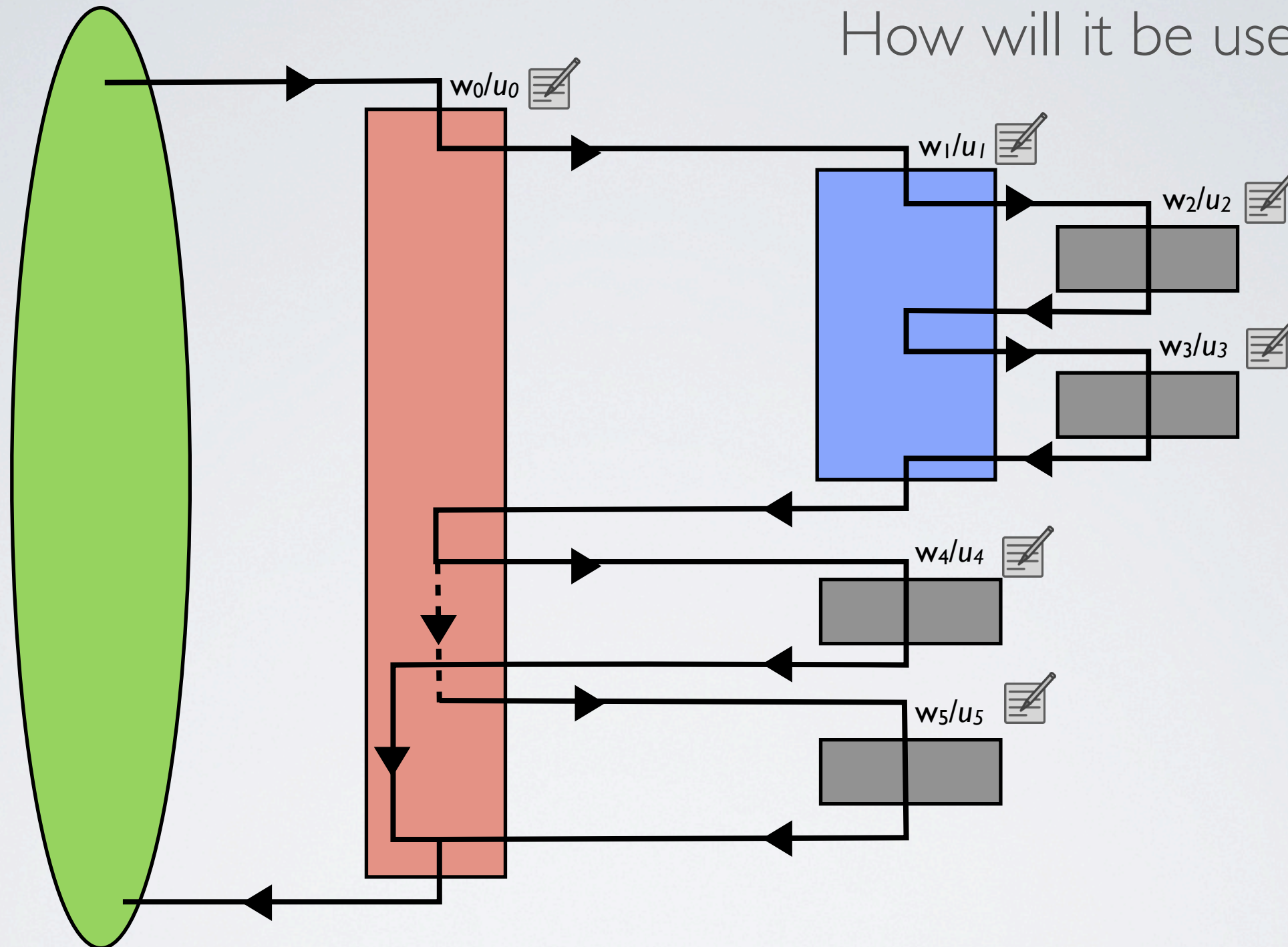
$$\Sigma \mid \Gamma \vdash_w e_1 : \tau \{ \mu \} \Rightarrow \tau'$$

$$\Sigma \mid \Gamma' \vdash_w e_2 : \tau$$

$$\Sigma \mid (\Gamma \parallel (\Gamma' ; \mu)) \vdash_w \text{call } e_1 \text{ with } e_2 : \tau'$$

$\lambda \text{ input:unit.}(\text{call } w_0/u_0 \text{ with input})$

Where is **input** going to?
How will it be used?

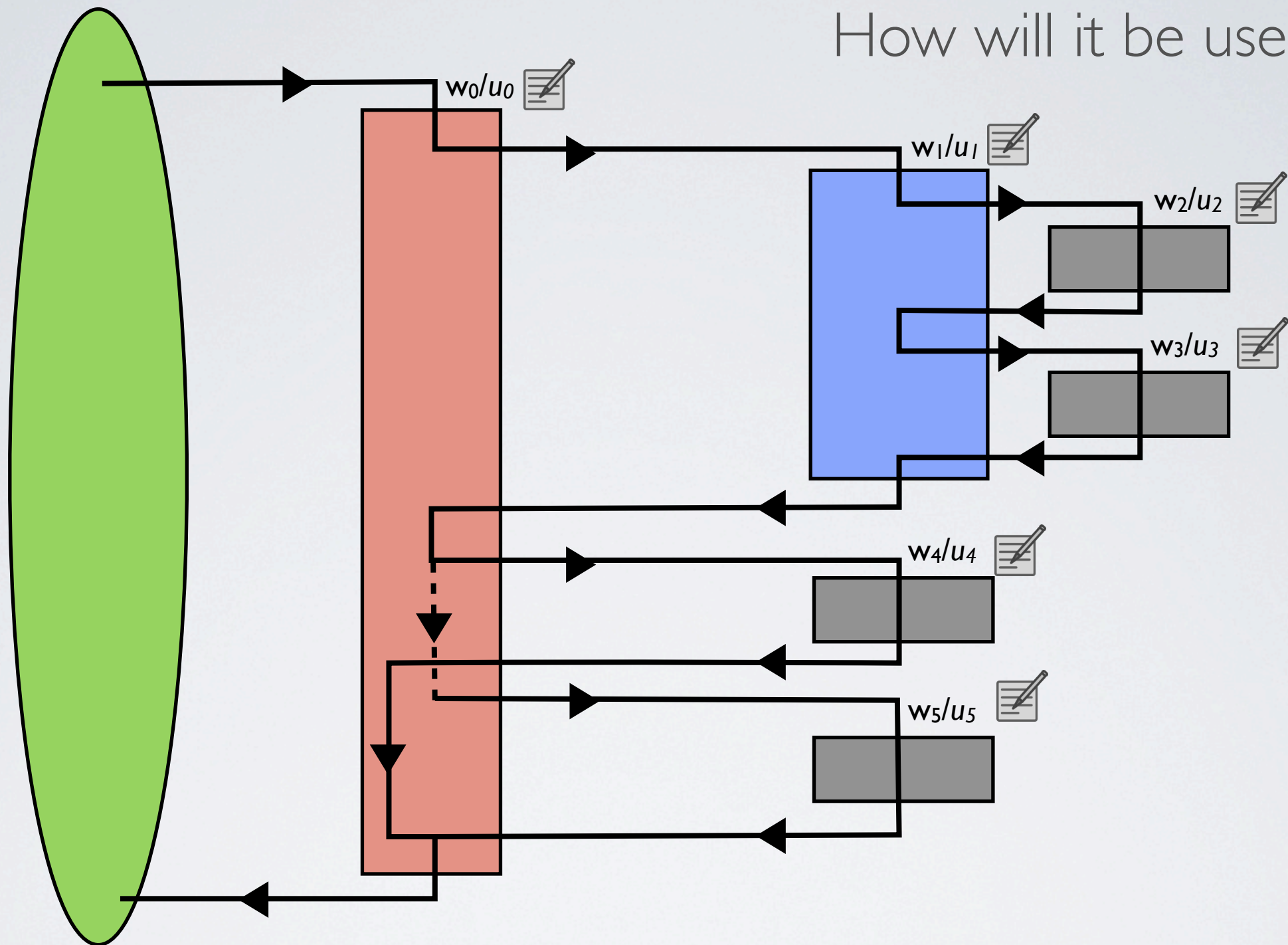


Can we get a symbolic expression that represent how **input** is used?

➡ The **data flow** of input

$\lambda \text{ input:unit.}(\text{call } w_0/u_0 \text{ with input})$

Where is **input** going to?
How will it be used?

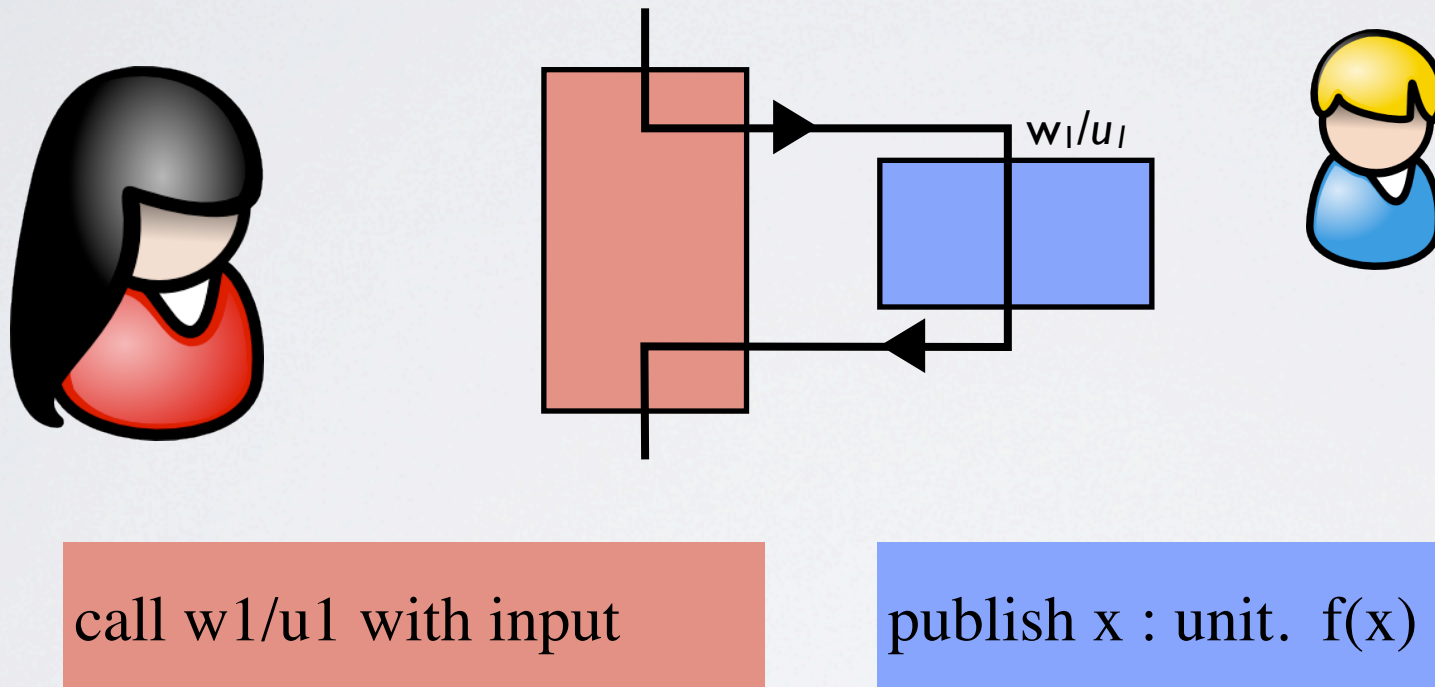


Can we get a symbolic expression that represent how **input** is used?

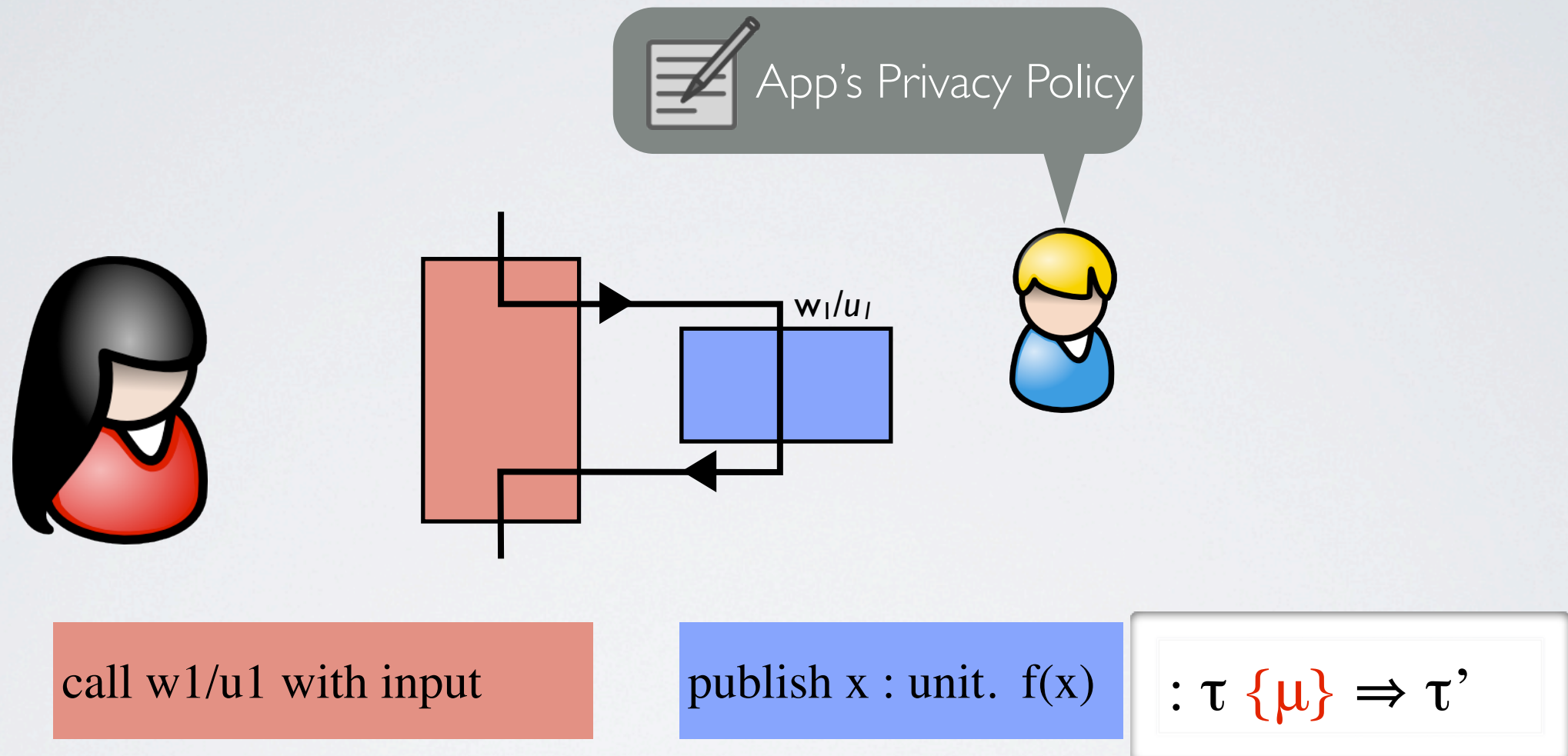
➡ The **data flow** of input

$$w_0 > ((w_1 > (w_2 ; w_3)) ; (w_4 \parallel w_5))$$

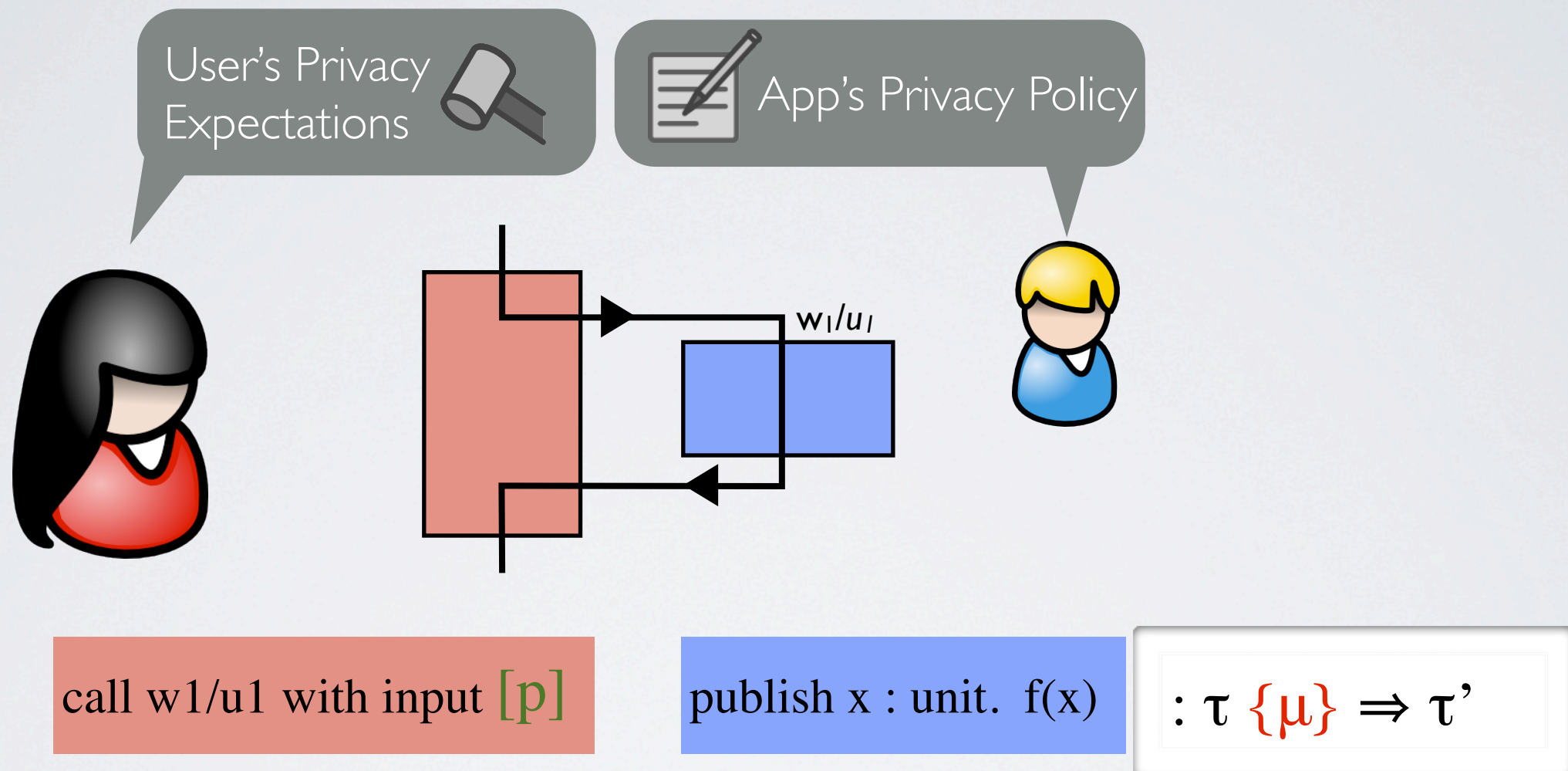
From data flow to flow policies



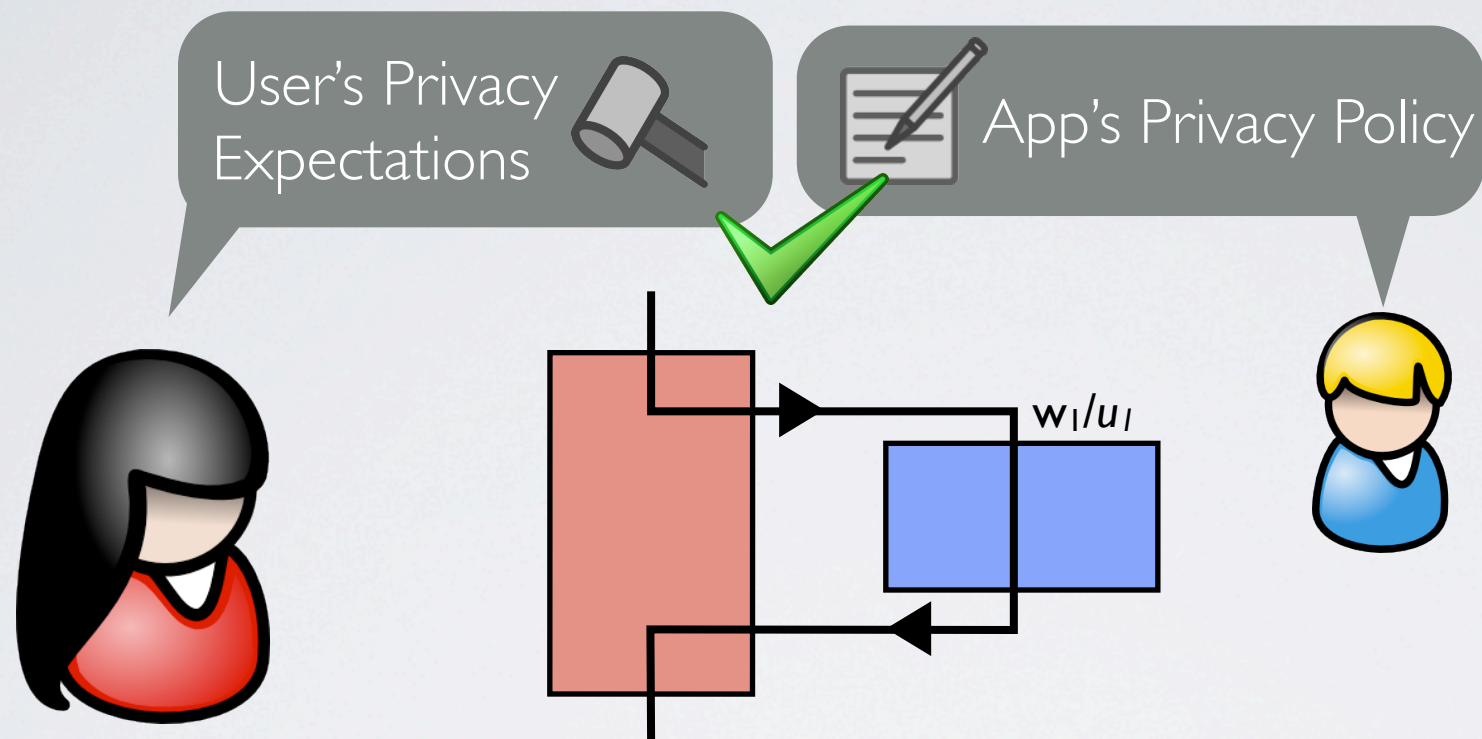
From data flow to flow policies



From data flow to flow policies



From data flow to flow policies



call w_1/u_1 with input $[p]$

publish $x : \text{unit. } f(x)$

$: \tau \{ \mu \} \Rightarrow \tau'$

$\mu \models p$

The policy language

Fragment of the language syntax

Types $\tau ::= \text{unit} \mid \tau \times \tau' \mid \tau \{\mu\} \rightarrow \tau' \mid \tau \{\mu\} \Rightarrow \tau'$

Expressions $e ::= x \mid \lambda x:\tau.e \mid (e_1 e_2) \mid \langle e_1, e_2 \rangle \mid \text{fst } e \mid \text{snd } e \mid ()$
 $\mid w/u \mid \text{publish } x:\tau.e \mid \text{call } e_1 \text{ with } e_2 [p]$

Flow $\mu ::= \bullet \mid w \succ \mu \mid \mu ; \mu' \mid \mu \parallel \mu'$

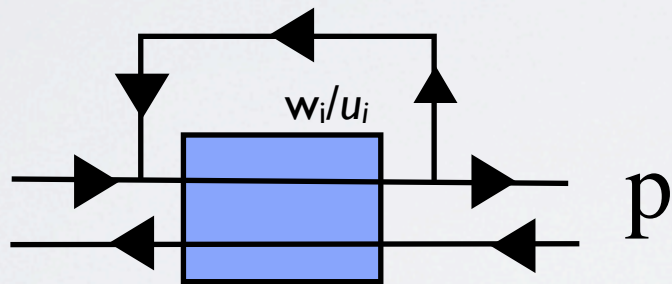
Policies $p = \top \mid \perp \mid \neg p \mid p_1 \wedge p_2 \mid p_1 \vee p_2 \mid \bullet \mid w \succ p \mid p_1 ; p_2$

Policy Evaluation

- | | |
|---|--|
| (1) $\mu \models \top$ | (5) $\bullet \models \bullet$ |
| (2) $\mu \models \neg \rho$ if $\mu \not\models \rho$ | (6) $w \succ \mu \models w \succ \rho$ if $\mu \models \rho$ |
| (3) $\mu \models \rho \wedge \rho'$ if $\mu \models \rho$ and $\mu \models \rho'$ | (7) $\mu ; \mu' \models \rho ; \rho'$ if $\mu \models \rho$ and $\mu' \models \rho'$ |
| (4) $\mu \models \rho \vee \rho'$ if $\mu \models \rho$ or $\mu \models \rho'$ | (8) $\mu \parallel \mu' \models \rho$ if $\mu \models \rho$ and $\mu' \models \rho$ |

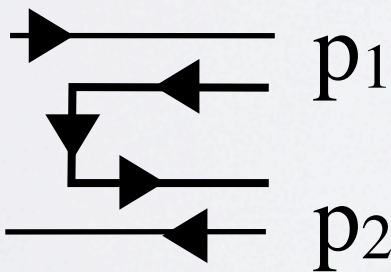
Derived Operators

$ws \succ^* p$



$w_i \in ws$

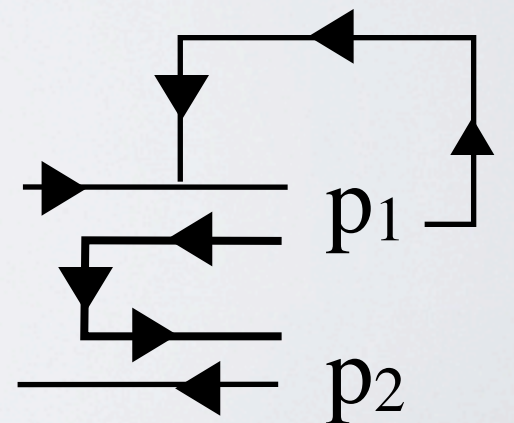
$p_1 ;^? p_2$



or

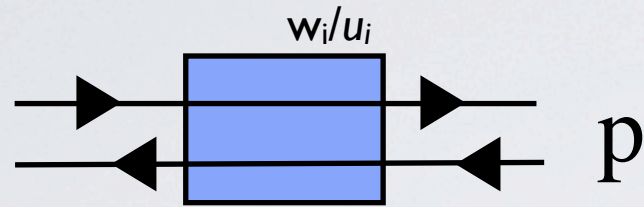


$p_1 ;^* p_2$



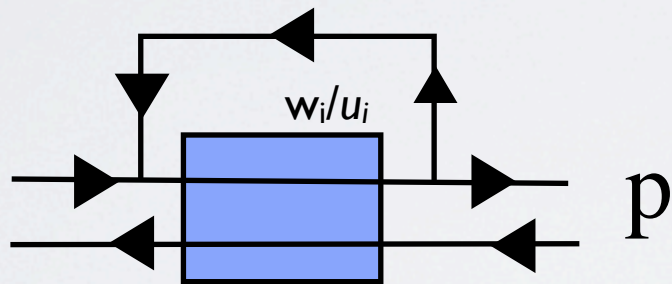
Derived Operators

$ws \succ p$



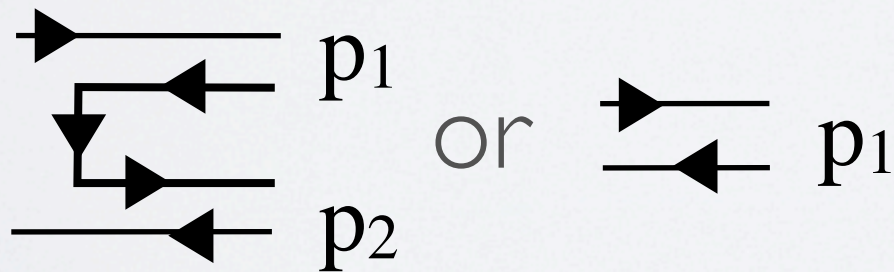
$w_i \in ws$

$ws \succ^* p$

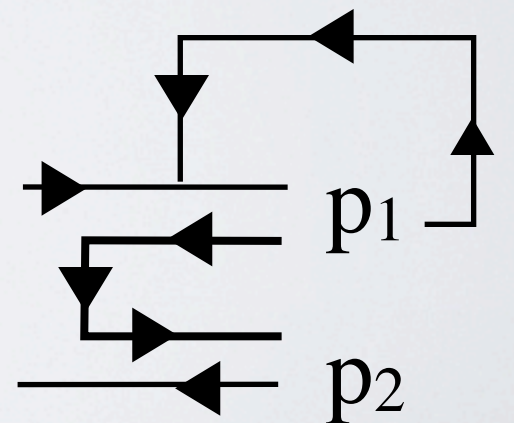


$w_i \in ws$

$p_1 ;^? p_2$

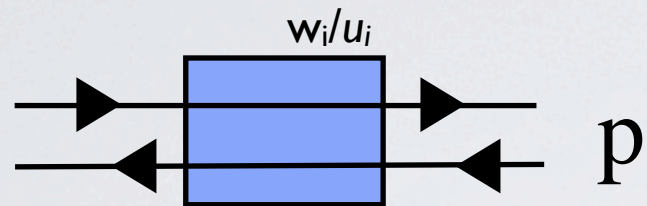


$p_1 ;^* p_2$



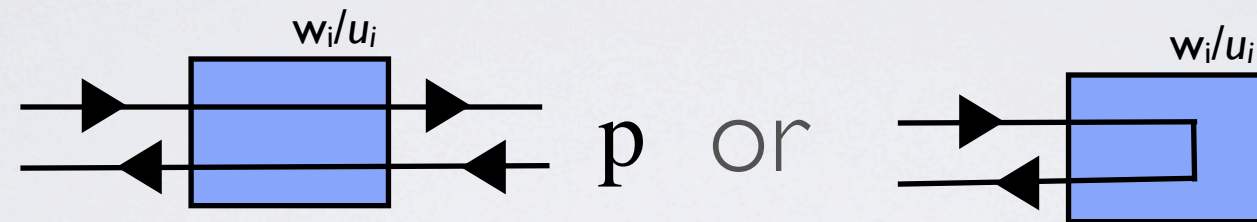
Derived Operators

$ws > p$



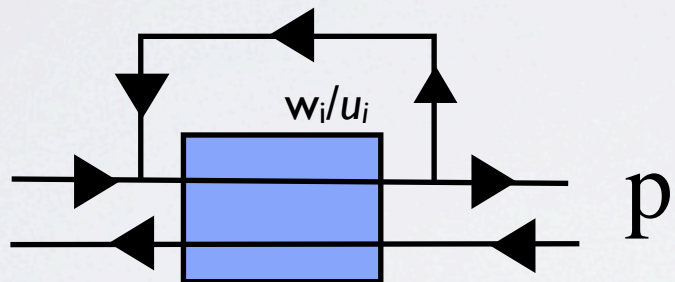
$w_i \in ws$

$ws >^? p$



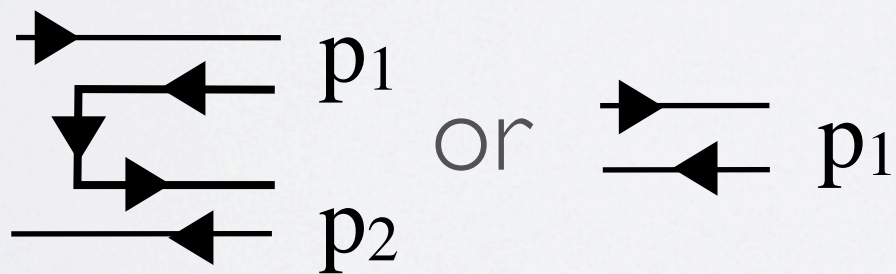
$w_i \in ws$

$ws >^* p$

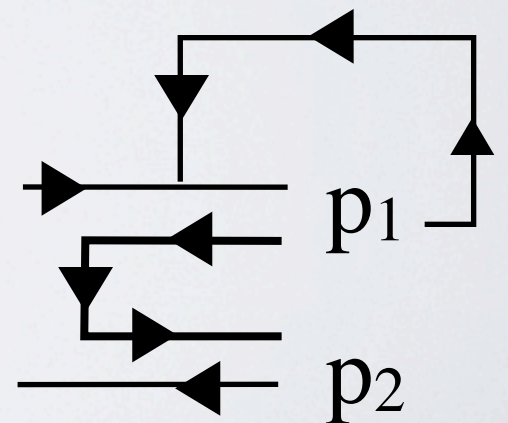


$w_i \in ws$

$p_1 ;^? p_2$



$p_1 ;^* p_2$



Policy examples

Alice trusts w_1 calling any services as needed, and w_0 may send the result coming from w_1 to w_4 and/or w_5 if needed

$$w_0 > ((w_1 > ((w_2 > \top) ; \top)) ; \{w_4, w_5\})$$

Alice trusts w_0, w_1, w_2, w_3, w_4 and w_5 only

$$(\{w_0, w_1, w_2, w_3, w_4, w_5\} >^* \perp) ;^* \perp$$

Alice trusts trusts any node except w_6

$$\overline{\overline{\{w_6\}}} = (\overline{\{w_6\}} >^* \perp) ;^* \perp$$

Chinese Wall security policy

As the value is sent to w_4 any nested service call or further composition should not involve w_5 and vice versa

$$p_4 = (w_4 \succ \overline{\overline{\{w_5\}}}) ;^? \overline{\overline{\{w_5\}}}$$

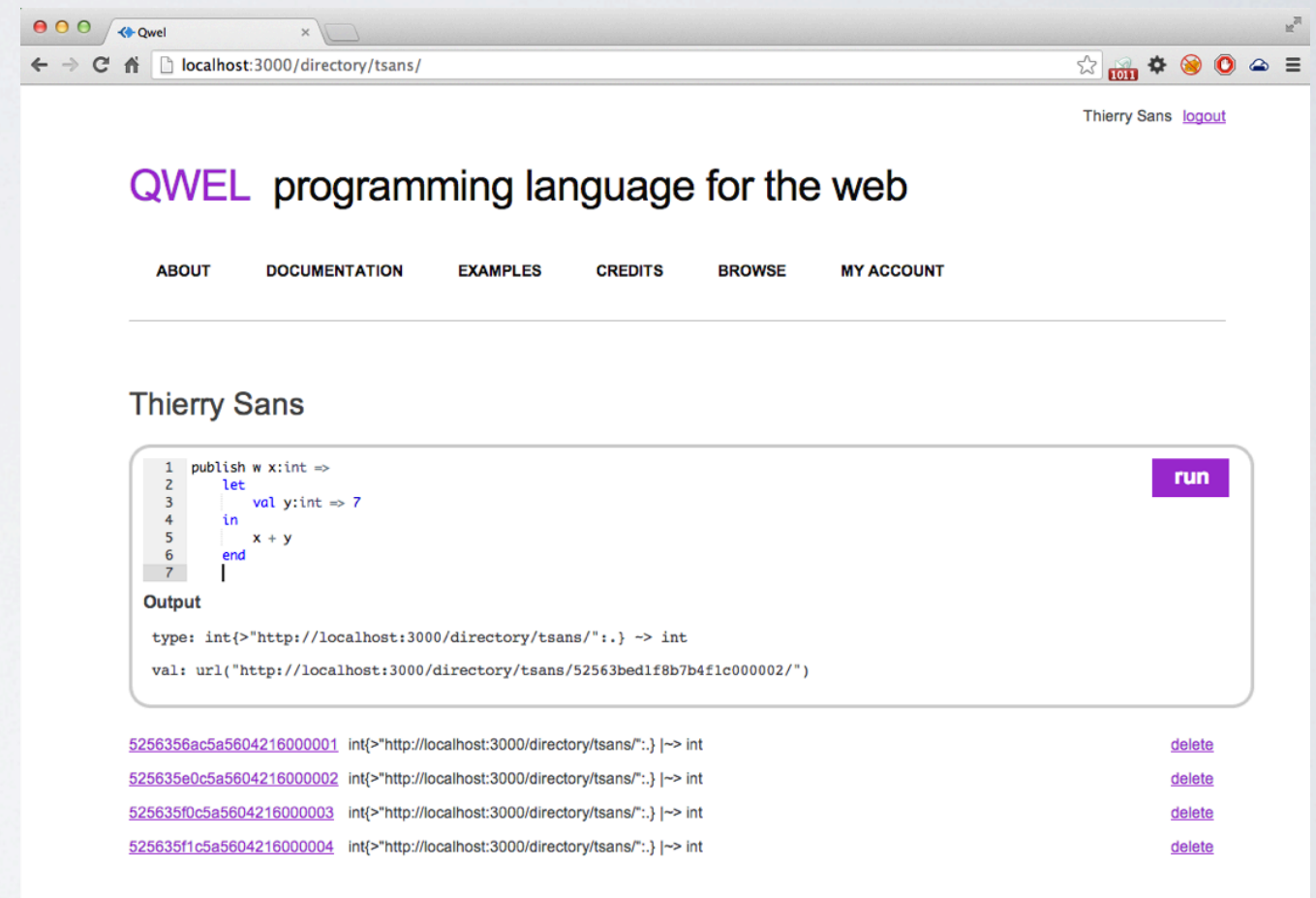
$$p_5 = (w_5 \succ \overline{\overline{\{w_4\}}}) ;^? \overline{\overline{\{w_4\}}}$$

$$p = \overline{\overline{\{w_4, w_5\}}} \vee ((\overline{\overline{\{w_4, w_5\}}} \succ^* p_4) ;^* \overline{\overline{\{w_5\}}}) \\ \vee ((\overline{\overline{\{w_4, w_5\}}} \succ^* p_5) ;^* \overline{\overline{\{w_4\}}})$$

The Qwel prototype

A web application

- Qwel interpreter written in Javascript (browser/server)
- Remote procedure calls are HTTP requests
- Services are stored locally in a MongoDB database



Conclusion

- Qwel a **type-safe language** for web programming
- Types annotated with **flow** inferred **statically**
- **Policy** language to constraint flows

Future work

- Extend the data flow model to discriminate between services from the same domain
- Support higher-order functions by dealing with hypothetical flow (polymorphic flow types)
- Extend the data flow model with support for mobile code (javascript code)
- Prove that executing the code generate a trace that matches the data flow predicted statically

Thank You

On the problem with higher-order functions



@W₀

```
publish (f, x): ((int → int) * int) = (f x)
```


On the problem with higher-order functions



What is the flow of x ?

@w₀

```
publish (f, x): ((int → int) * int) = (f x)
```

On the problem with higher-order functions



What is the flow of x ?

@W₀

```
publish (f, x): ((int → int) * int) = (f x)
```

It depends on f !

```
f : int {μ} → int
```


On the problem with higher-order functions



What is the flow of x ?

@w₀

publish (f, x): ((int \rightarrow int) * int) = (f x)

: int {w₀ > μ } \Rightarrow int

It depends on f!

f : int { μ } \rightarrow int

On the problem with higher-order functions



What is the flow of x ?

@w₀

publish (f, x): ((int \rightarrow int) * int) = (f x)

: int {w₀ > μ } \Rightarrow int

It depends on f !

f : int { μ } \rightarrow int

We need account for hypothetical flow with higher-order functions

➡ Polymorphic flow types

Related Work

Dependencies between program building blocks

- History-based type systems - *Abadi and al* [1]
 - Data Flow Graph - *Ferrante and al* [12]
- ➔ In Qwel, services can be created dynamically as opposed to having static libraries

Acceptable composition of shared resources

- Algebra and logic for access control - *Pym and al* [4, 5]
- ➔ Qwel outlines the specific context of web programming

Related Work

Verifying policy constraints statically

- Fable - *Swamy and al* [16]
 - Jif - *Myers and al* [17]
- ➔ Similar idea explored with Qwel but not based on labeling

Data flow vs non-interference

In non-interference models

- flow is controlled by the mean of type labels and a lattice
- but no explicit data flow representation

We want to understand better how they are related

- Are these two approaches orthogonal?
- Can we use them in conjunction to provide a robust security model?