

Profunctor Optics



Categorical View

Bartosz Milewski

Optics

```
type Lens s t a b = forall p. Strong p => p a b -> p s t
```

```
type Prism s t a b = forall p. Choice p => p a b -> p s t
```

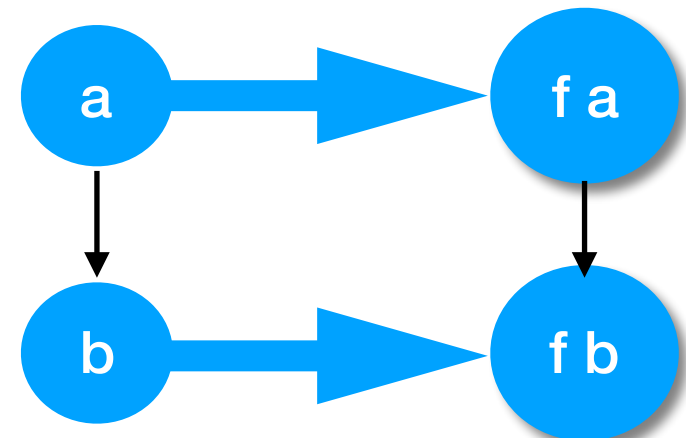
```
class Profunctor p => Strong p where  
    first' :: p a b -> p (a, c) (b, c)
```

```
class Profunctor p => Choice p where  
    left' :: p a b -> p (Either a c) (Either b c)
```

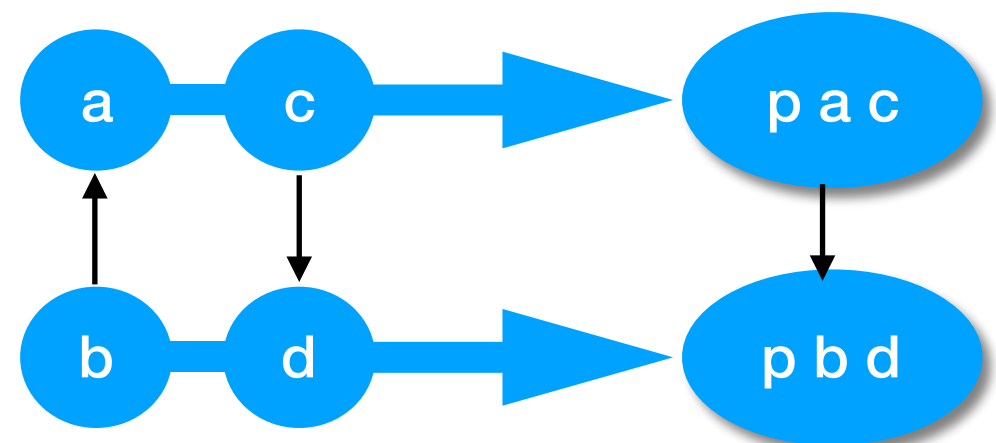
```
class Profunctor p where  
    dimap :: (a -> b) -> (c -> d) -> (p b c -> p a d)
```

Functors

```
class Functor f where  
  fmap :: (a -> b) -> (f a -> f b)
```



```
class Profunctor p where  
  dimap :: (b -> a) -> (c -> d) -> (p a c -> p b d)
```

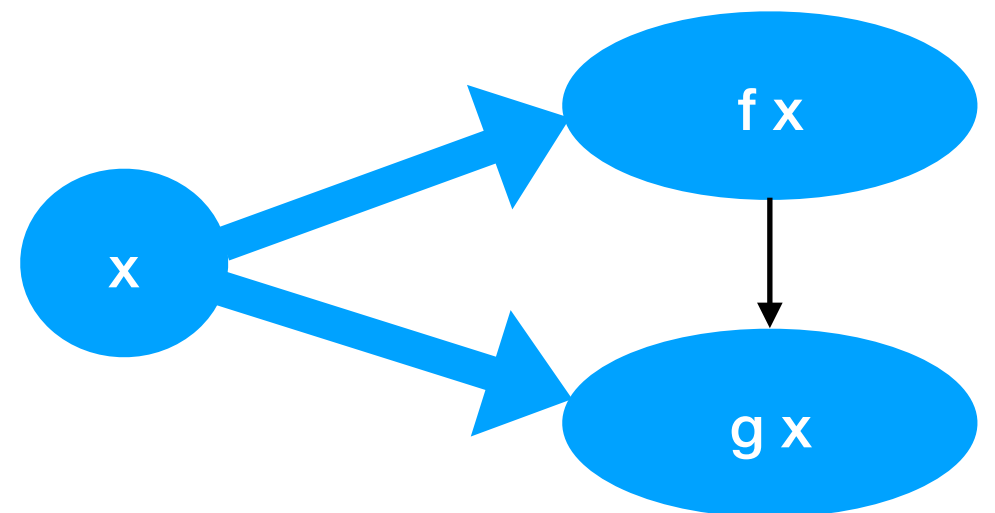


Functor from $C^{op} \times C \rightarrow Set$

Natural Transformations

```
type f ~> g = forall x. f x -> g x
```

```
safeHead :: [] ~> Maybe  
safeHead [] = Nothing  
safeHead (a:as) = Just a
```



```
type Lens s t a b = forall p. Strong p => p a b -> p s t
```

Yoneda Lemma

```
type Reader a x = a -> x
```

```
instance Functor (Reader a) where  
  fmap f atox = f . atox
```

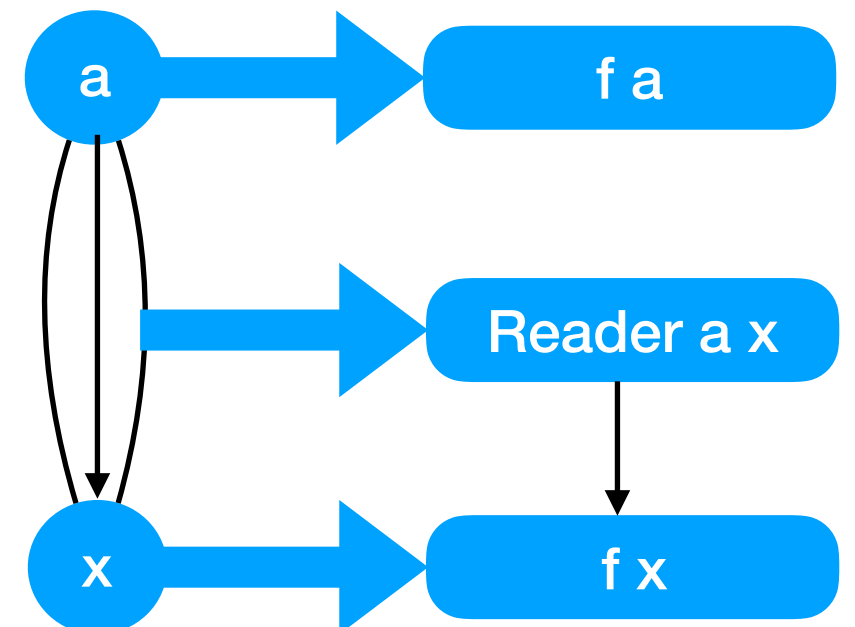
```
type Yo f a = Functor f => Reader a ~> f
```

Yo f a ~ f a

forall x. (a -> x) -> f x ~ f a

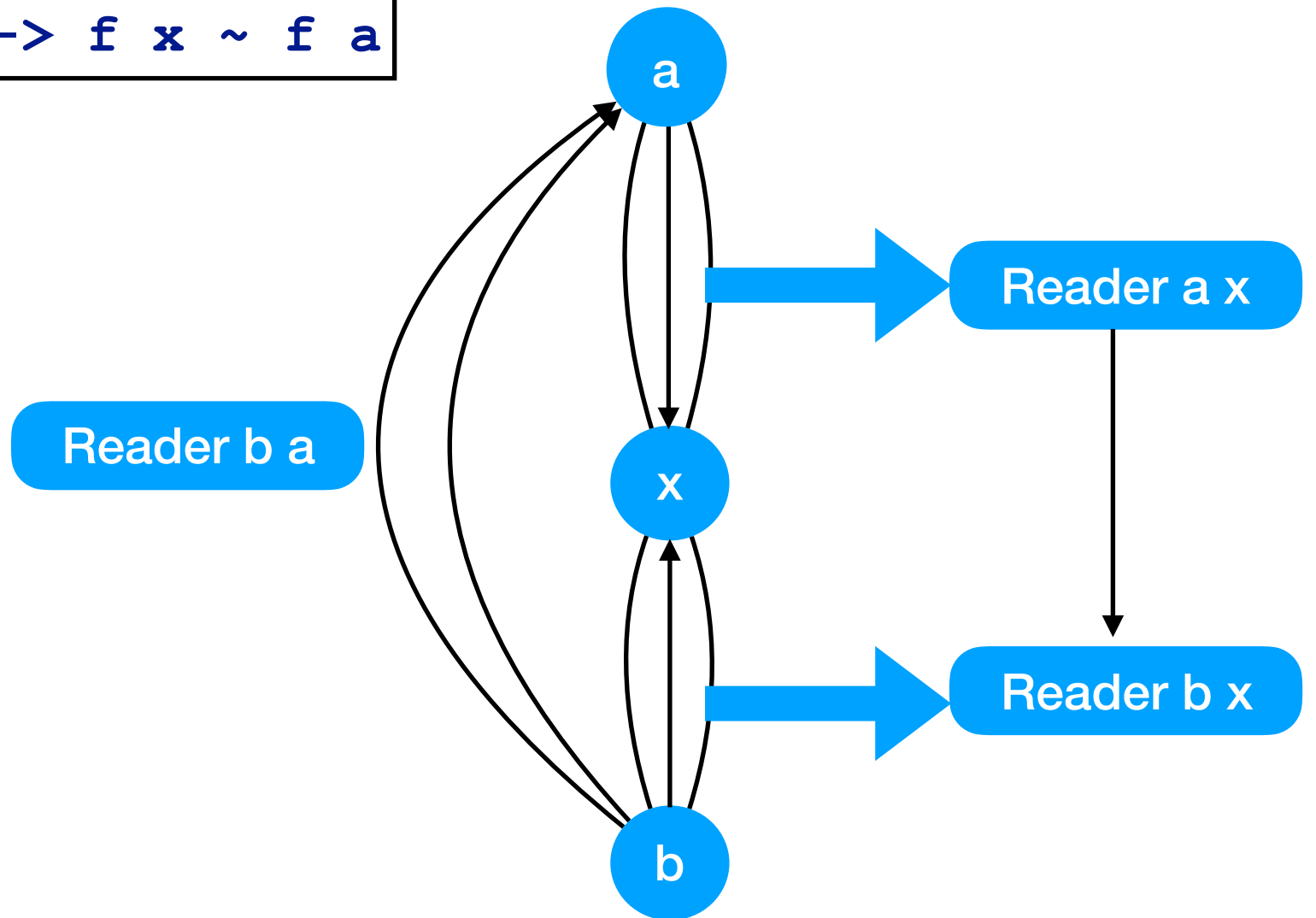
```
toYo :: Functor f => f a -> Yo f a  
toYo fa = \atox -> fmap atox fa
```

```
fromYo :: Functor f => Yo f a -> f a  
fromYo alpha = alpha id
```



Yoneda Embedding

`forall x. (a -> x) -> f x ~ f a`

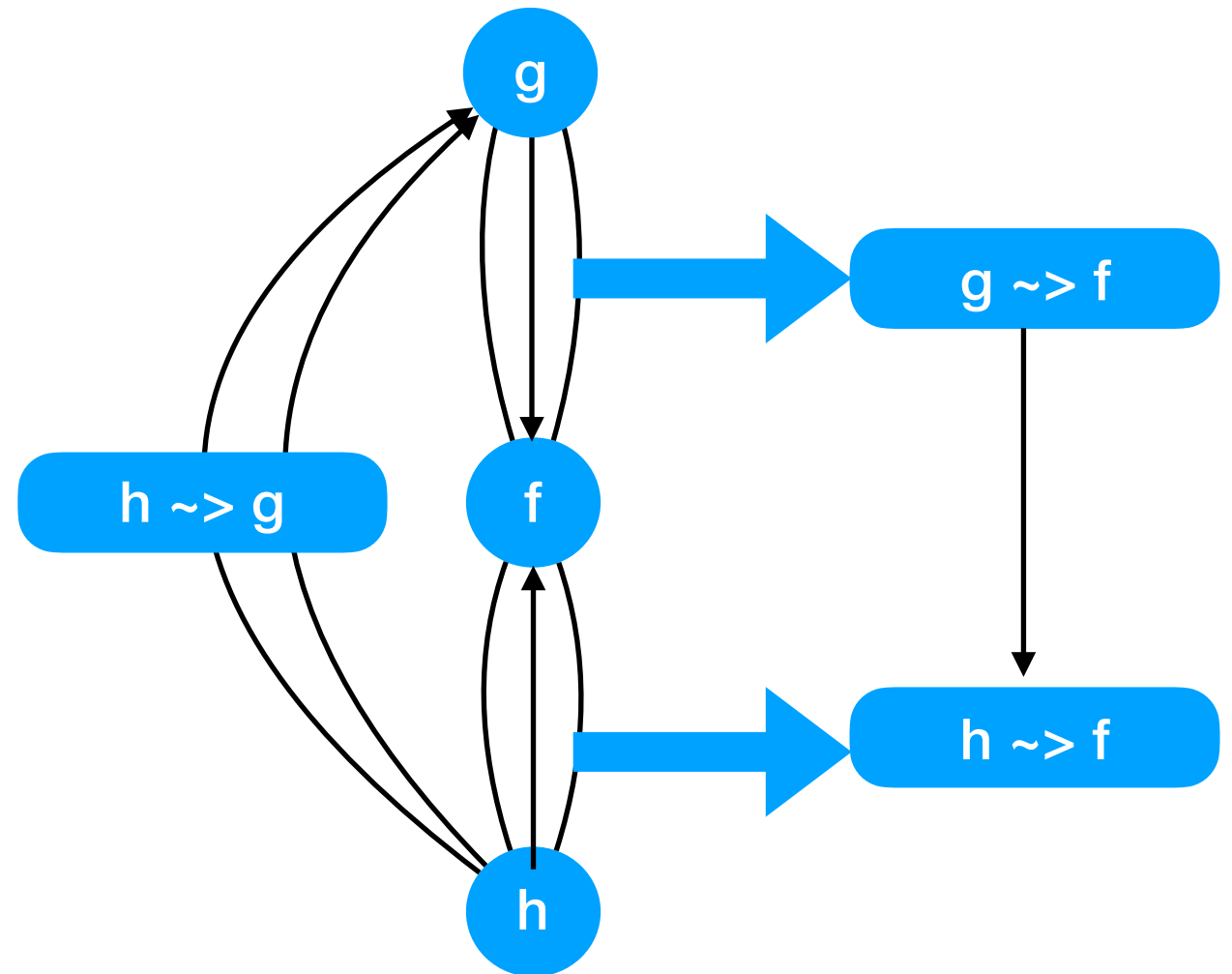


`f x = (b -> x)`

`forall x. (a -> x) -> (b -> x) ~ (b -> a)`

Yoneda on Functors

```
forall x. (a -> x) -> (b -> x)
~ (b -> a)
```



```
forall f. Functor f => (g ~> f) -> (h ~> f)
~ (h ~> g)
```

Toy Optic

```
forall f. Functor f. (g ~> f) -> (h ~> f)  
~ (h ~> g)
```

```
forall f. (forall x. g x -> f x) -> (forall x. h x -> f x)  
~ (forall x. h x -> g x)
```

```
g x = a -> x  
h x = s -> x
```

```
forall f. (forall x. (a->x) -> f x) -> (forall x. (s->x) -> f x)  
~ (forall x. (s->x) -> (a->x))
```

```
forall f. f a -> f s ~ (a -> s)
```

```
type Toy s a = forall f. Functor f => f a -> f s  
Toy s a ~ (a -> s)
```


Yoneda for Profunctors

```
type ProReader a b x y = (x -> a, b -> y)
```

```
instance Profunctor (ProReader a b) where  
  dimap from to (f, g) = (f.from, to.g)
```

```
type ProYo p a b =  
  Profunctor p => forall x y. (x -> a, b -> y) -> p x y
```

ProYo p a b ~ p a b

forall x y. (x -> a) -> (b -> y) -> p x y ~ p a b

Yoneda on Profunctors

```
forall p. Profunctor p => (q ~> p) -> (r ~> p) ~ (r ~> q)
```

```
forall p. Profunctor p => (forall x y. q x y -> p x y) ->
                          (forall x y. r x y -> p x y)
  ~ (forall x y. r x y -> q x y)
```

```
q x y = (x->a, b->y)
r x y = (x->s, t->y)
```

```
forall p. (forall x y. (x->a, b->y) -> p x y) ->
          (forall x y. (x->s, t->y) -> p x y)
  ~ (forall x y. (x->s, t->y) -> q x y)
```

```
forall p. Profunctor p => p a b -> p s t
~ q s t ~ (s -> a, b -> t) ~ Iso s t a b
```

Optics

```
type Iso s t a b = forall p. Profunctor p => p a b -> p s t
```

```
type Lens s t a b = forall p. Strong p => p a b -> p s t
```

```
type Prism s t a b = forall p. Choice p => p a b -> p s t
```

```
class Profunctor p => Strong p where  
  first' :: p a b -> p (a, c) (b, c)
```

```
class Profunctor p => Choice p where  
  left' :: p a b -> p (Either a c) (Either b c)
```

```
class Profunctor p => Tambara p where  
  left' :: p a b -> p (a  $\otimes$  c) (b  $\otimes$  c)
```

Adjoint Functors

F left adjoint to U

$F a \rightarrow b \sim a \rightarrow U b$

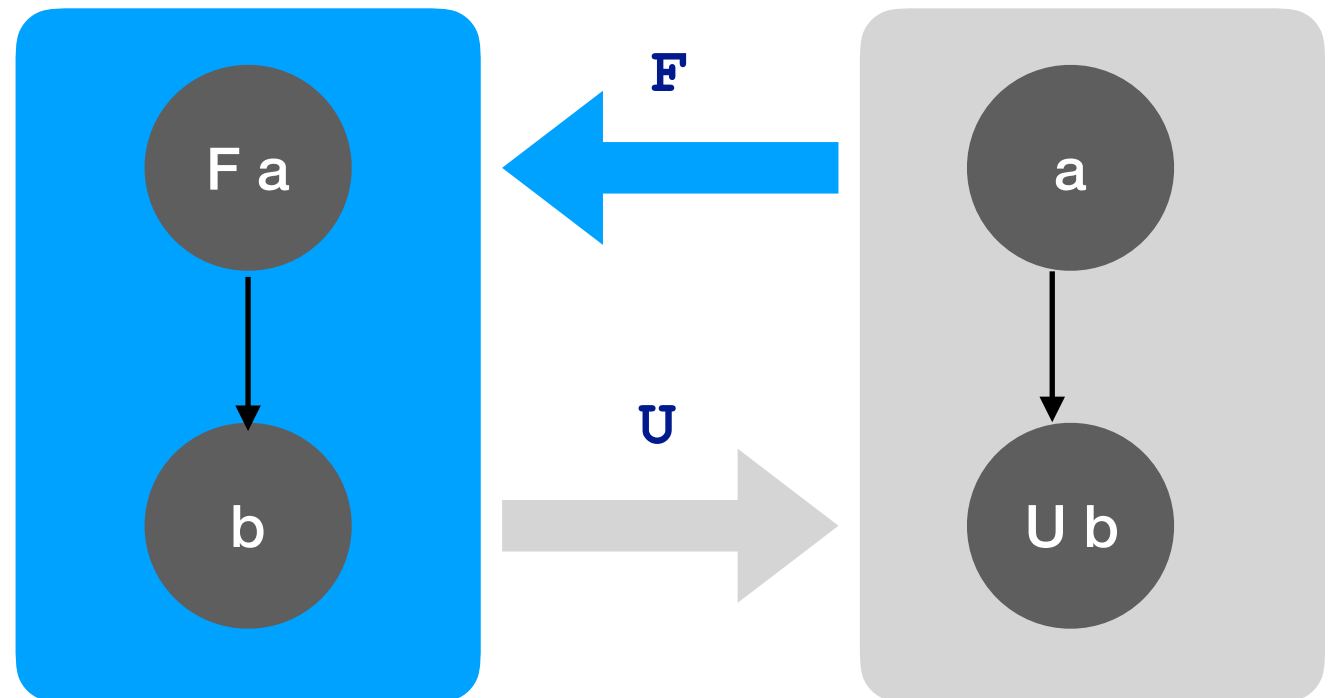
$F a = (a, x)$

$U b = (x \rightarrow b)$

$(a, x) \rightarrow b \sim a \rightarrow (x \rightarrow b)$

Yoneda

$\text{exists } x. (f x, x \rightarrow a) \sim f a$



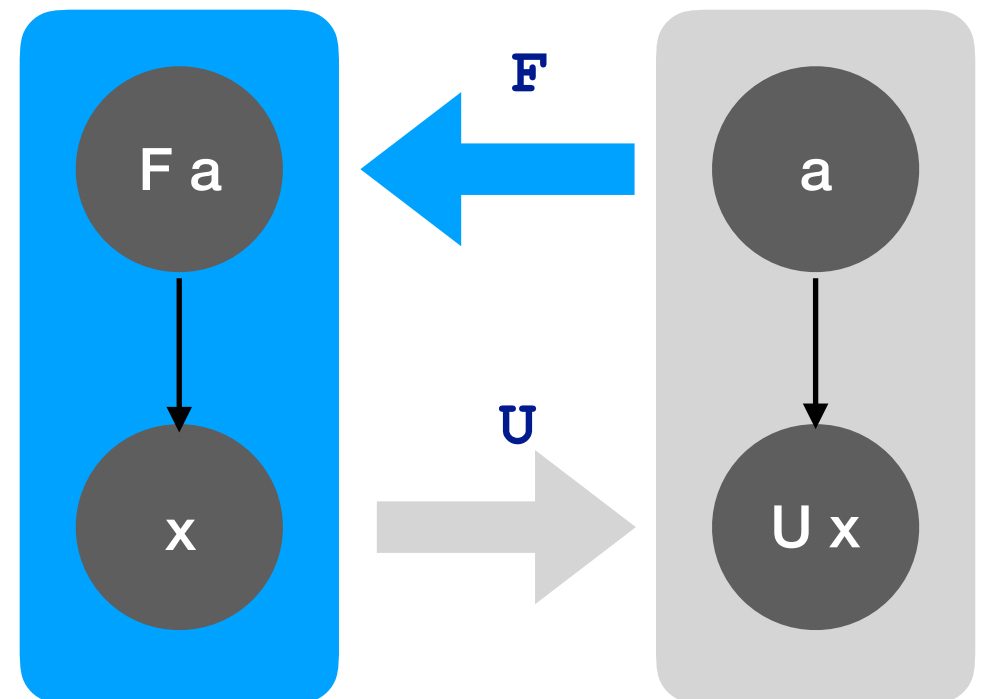
Yoneda + Adjunction

```
forall x. (a -> x) -> (b -> x) ~ (b -> a)
```

```
forall x. (F a -> x) -> (F b -> x)  
~ (F b -> F a)
```

```
forall x. (a -> U x) -> (b -> U x)  
~ (b -> (U ◦ F) a)
```

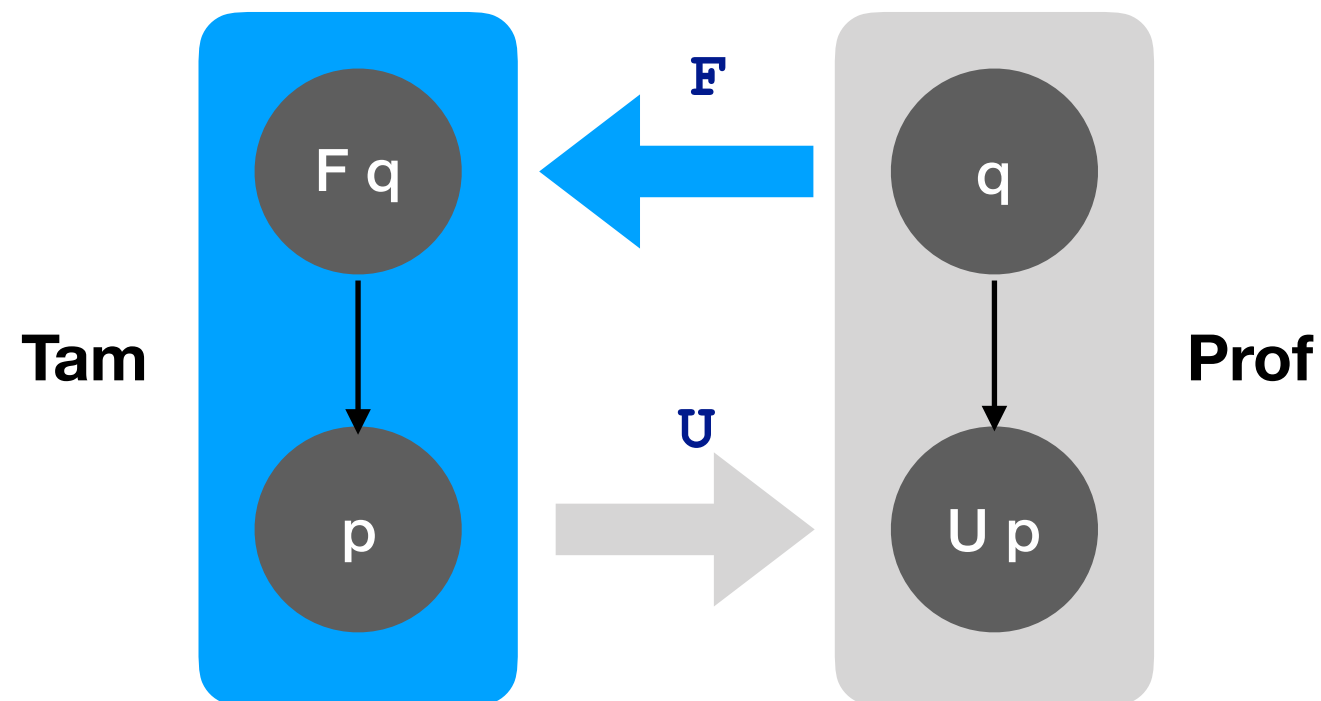
U ◦ F is a monad



$F\ a \rightarrow x \sim a \rightarrow U\ x$

Profunctor Adjunction

```
forall p. Profunctor p => (forall x y. q x y -> p x y) ->
                           (forall x y. r x y -> p x y)
  ~ (forall x y. r x y -> q x y)
```



```
forall p. Tambara p => (forall x y. q x y -> (U p) x y) ->
                       (forall x y. r x y -> (U p) x y)
  ~ (forall x y. r x y -> ((U o F) q) x y)
```

Optics + Adjunction

$$\text{forall } p. \text{ Tambara } p \Rightarrow (\text{forall } x \ y. q \ x \ y \rightarrow (U \ p) \ x \ y) \rightarrow$$
$$(\text{forall } x \ y. r \ x \ y \rightarrow (U \ p) \ x \ y)$$
$$\sim (\text{forall } x \ y. r \ x \ y \rightarrow ((U \circ F) \ q) \ x \ y)$$

$$q \ x \ y = (x \rightarrow a, b \rightarrow y)$$
$$r \ x \ y = (x \rightarrow s, t \rightarrow y)$$

$$\text{forall } p. \text{ Tambara } p \Rightarrow (U \ p) \ a \ b \rightarrow (U \ p) \ s \ t$$
$$\sim ((U \circ F) \ q) \ s \ t$$

$$\text{forall } p. \text{ Tambara } p \Rightarrow p \ a \ b \rightarrow p \ s \ t$$
$$\sim (\Phi \ q) \ s \ t$$

Free Tambara

$$(\Phi \ q) \ s \ t = \int c \times y \ A(s, \ c \otimes x) \otimes A(c \otimes y, \ t) \otimes q \ x \ y$$

$$(\Phi_i \ q) \ s \ t = \text{exists } c \times y. \ ((s \rightarrow c \otimes x, \ c \otimes y \rightarrow t), \ q \ x \ y)$$

$$q \ x \ y = (x \rightarrow a, \ b \rightarrow y)$$

$$(\Phi_i \ q) \ s \ t = \text{exists } c. \ (s \rightarrow c \otimes a, \ c \otimes b \rightarrow t)$$

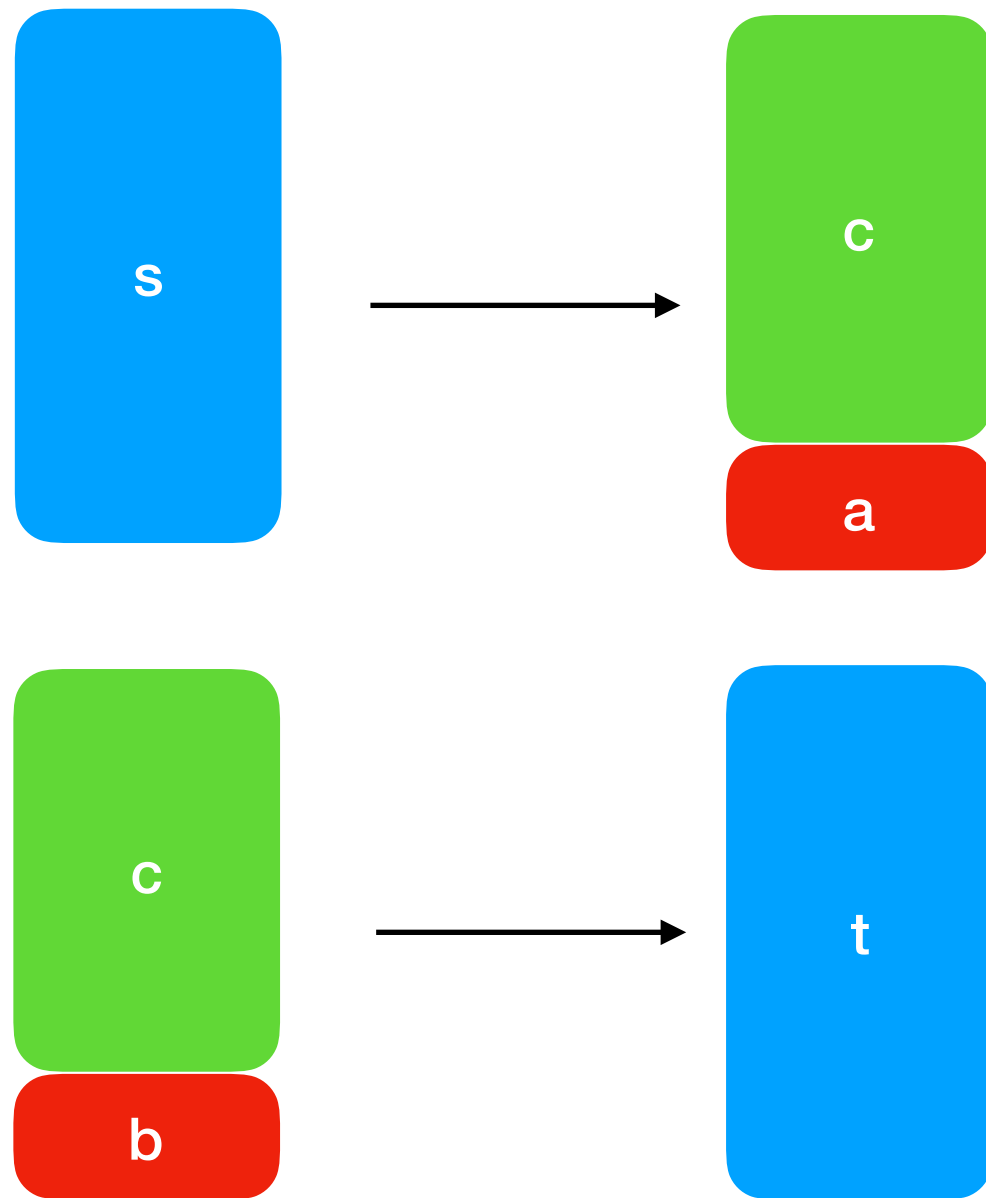
$$\text{type Lens } s \ t \ a \ b = \text{forall } p. \text{ Strong } p \Rightarrow p \ a \ b \rightarrow p \ s \ t$$

$$\text{Lens } s \ t \ a \ b = \text{exists } c. \ (s \rightarrow (c, \ a), \ (c, \ b) \rightarrow t)$$

$$\text{type Prism } s \ t \ a \ b = \text{forall } p. \text{ Choice } p \Rightarrow p \ a \ b \rightarrow p \ s \ t$$

$$\text{Prism } s \ t \ a \ b = \text{exists } c. \ (s \rightarrow \text{Either } c \ a, \ \text{Either } c \ b \rightarrow t)$$

`Lens s t a b = exists c. (s -> (c, a), (c, b) -> t)`



`Prism s t a b = exists c. (s -> Either c a, Either c b -> t)`