ANOSY:
Approximated K**no**wledge **Sy**nthesis with Refinement Types

Sankha Guria*, **Niki Vazou**#, Marco Guarnieri#, James Parker+

* University of Maryland

IMDEA Software Institute

+ Galois, Inc.

Anosy @ PLDI'22

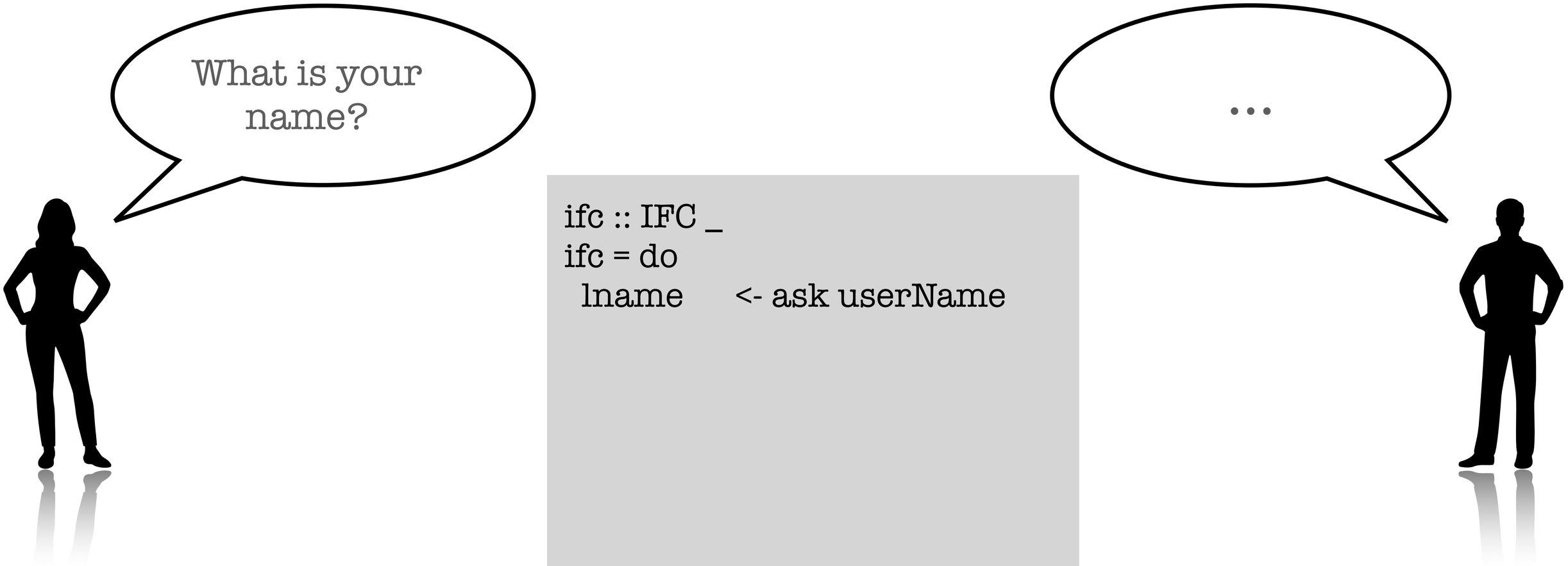
Haskell: Pure with Monads

Haskell: Pure with Monads

Ideal to track information that goes in the monad

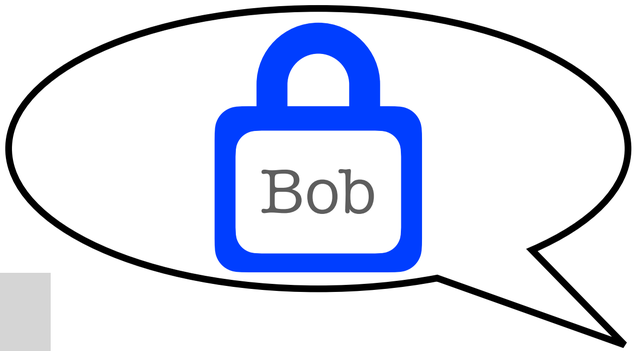
a.k.a. Information Flow Control

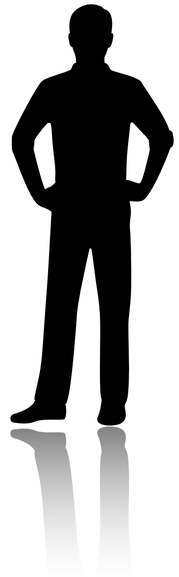
Monadic Information Flow Control




```
ifc :: IFC _
ifc = do
  lname     <- ask userName
```

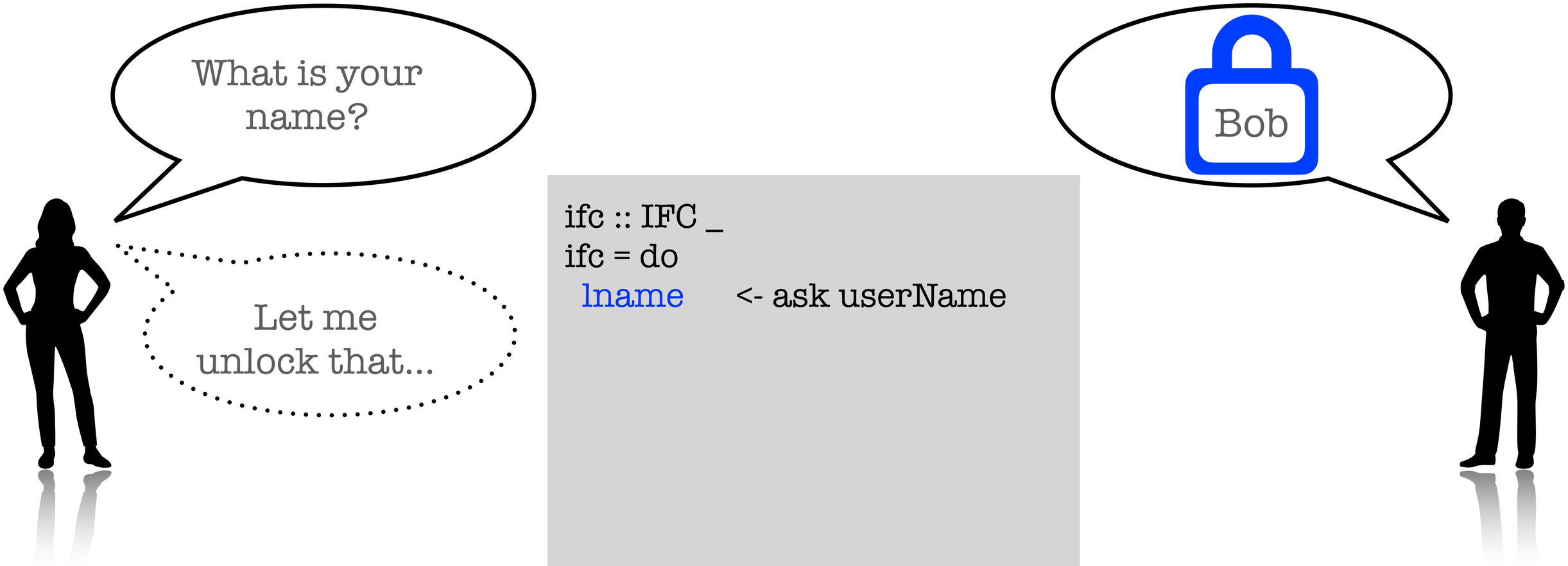
Monadic Information Flow Control





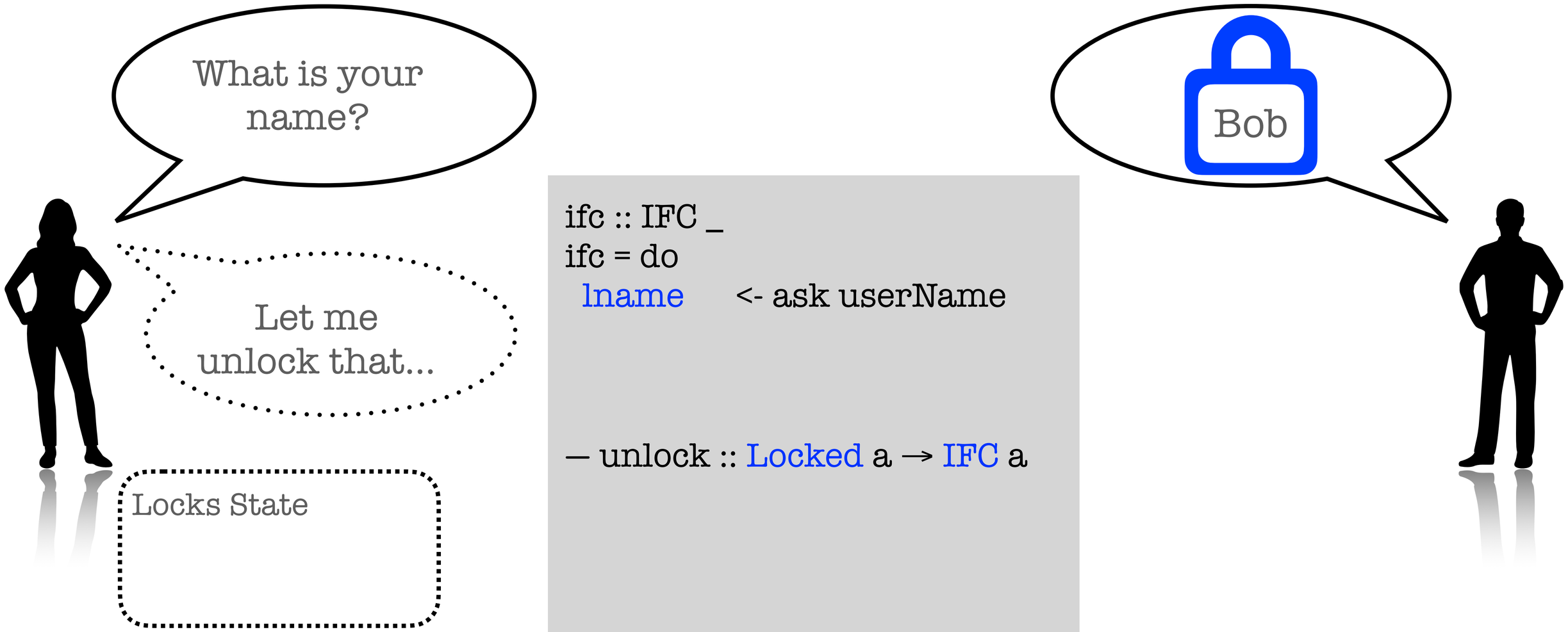
```
ifc :: IFC _
ifc = do
  lname     <- ask userName
```

Monadic Information Flow Control



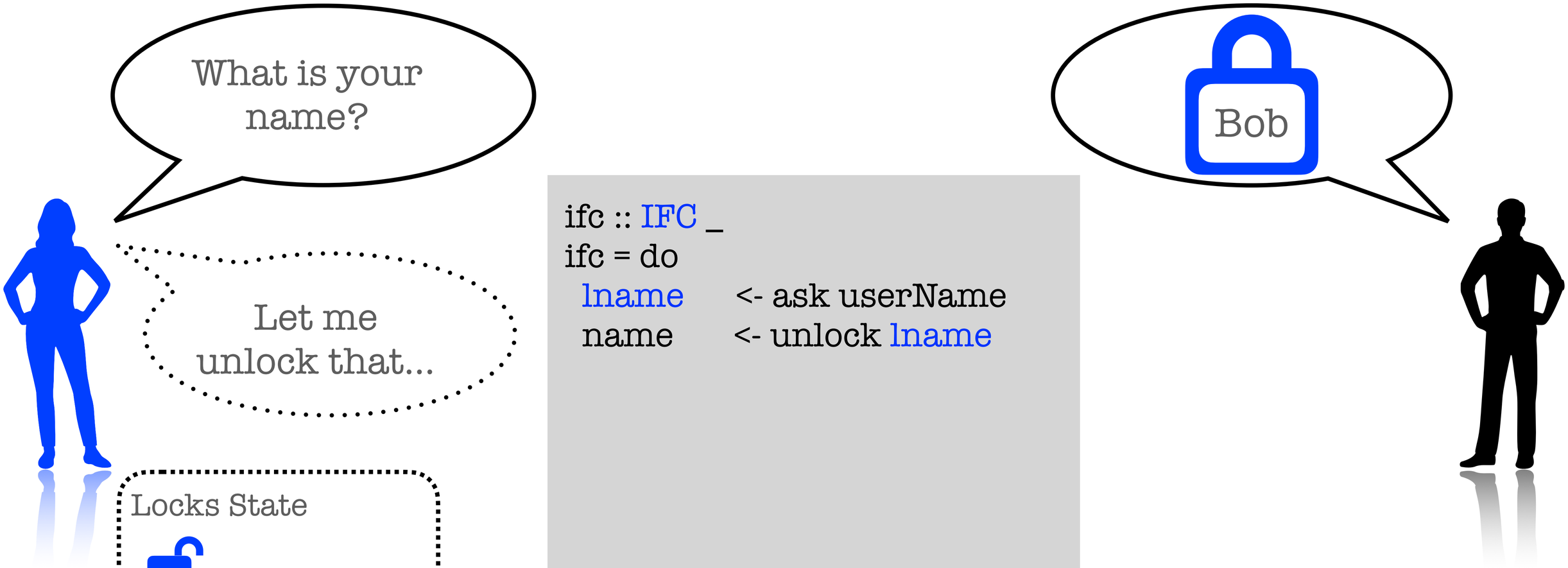

```
ifc :: IFC _
ifc = do
  lname     <- ask userName
```

Monadic Information Flow Control



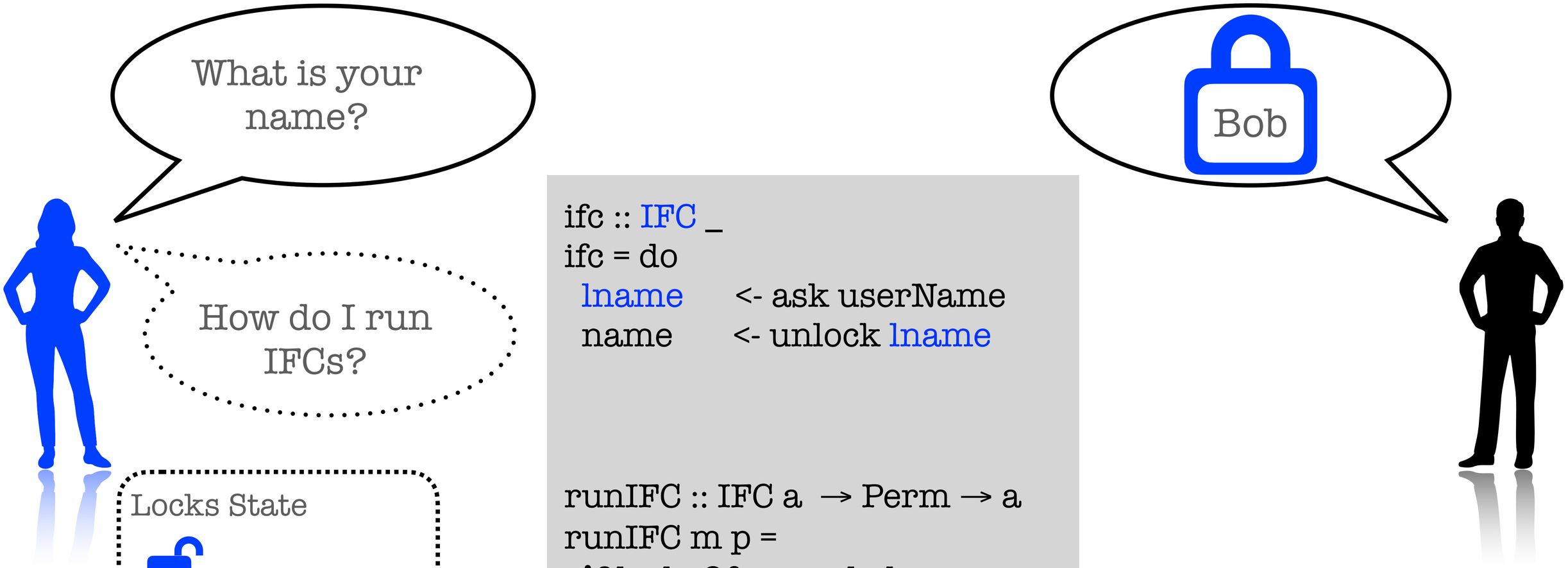

```
ifc :: IFC _
ifc = do
  lname     <- ask userName



— unlock :: Locked a → IFC a
```

Monadic Information Flow Control




```
ifc :: IFC _
ifc = do
  lname     <- ask userName
  name      <- unlock lname
```

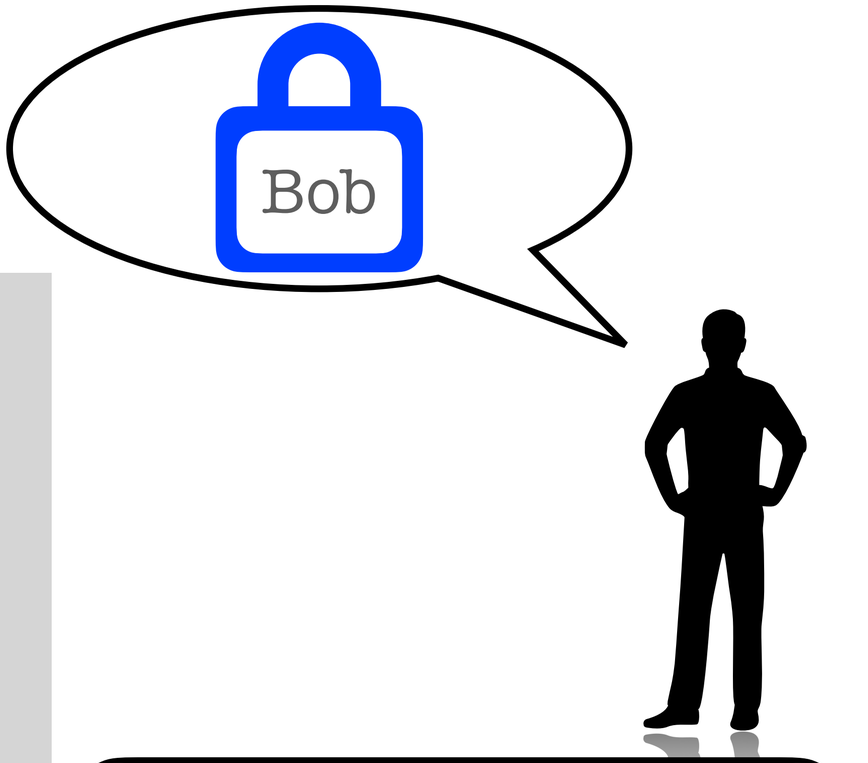
Monadic Information Flow Control




```
ifc :: IFC _
ifc = do
  lname     <- ask userName
  name      <- unlock lname


runIFC :: IFC a  → Perm → a
runIFC m p =
  if locksOf m are below p
    then valueOf  m
    else Permission Error
```

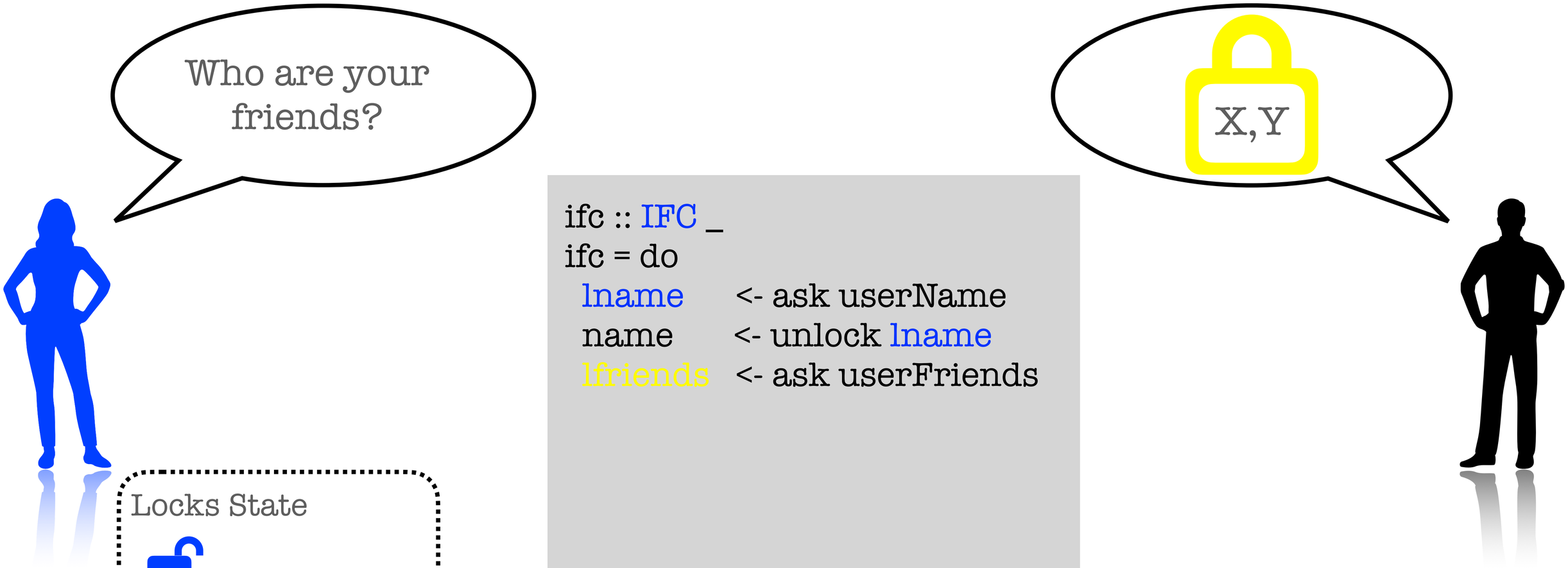
Monadic Information Flow Control


```
ifc :: IFC _
ifc = do
  lname     <- ask userName
  name      <- unlock lname


runIFC :: IFC a  → Perm → a
runIFC m p =
  if locksOf m are below p
    then valueOf  m
    else Permission Error
```



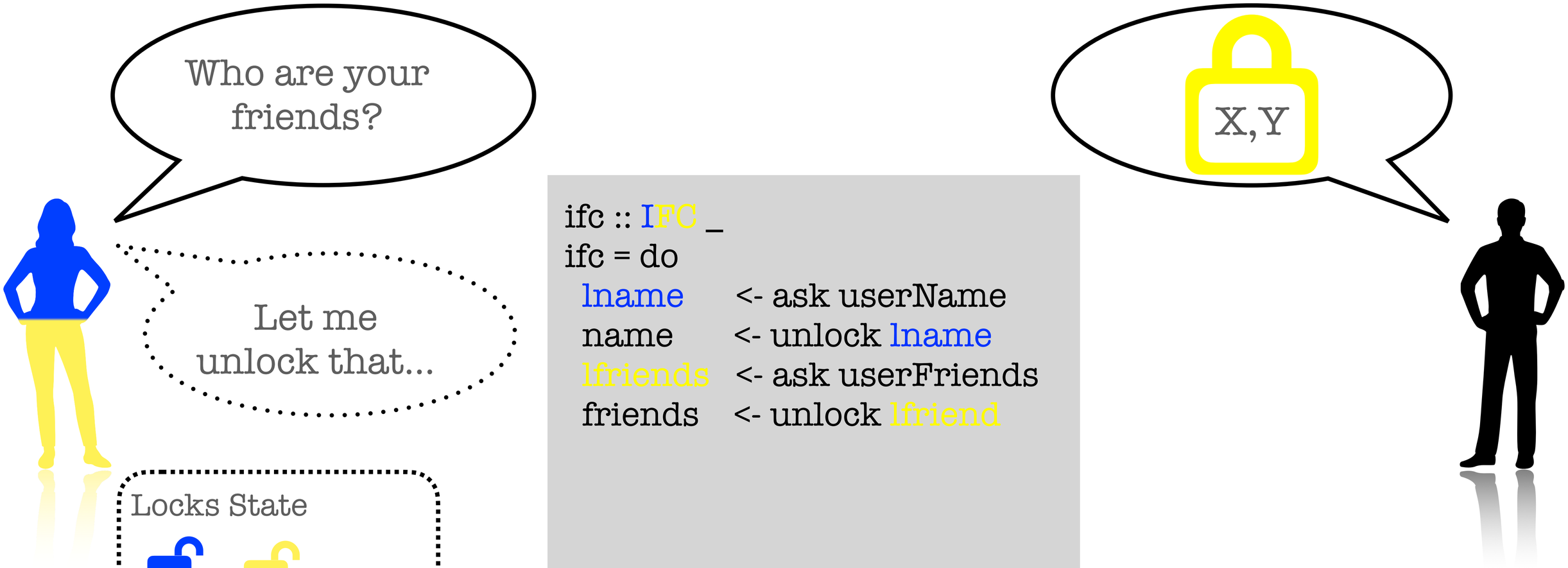
Haskell IFC systems:
LIO, LWeb, STORM

Monadic Information Flow Control



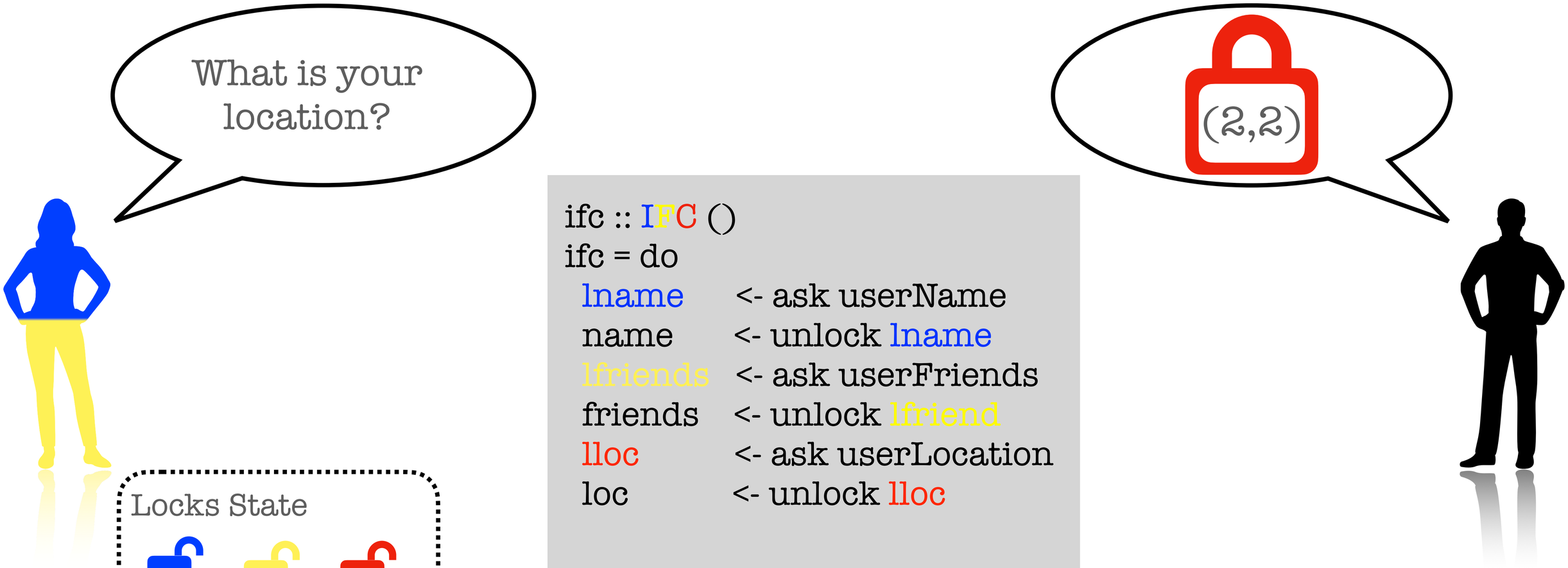

```
ifc :: IFC _
ifc = do
  lname     <- ask userName
  name      <- unlock lname
  lfriends  <- ask userFriends
```

Monadic Information Flow Control



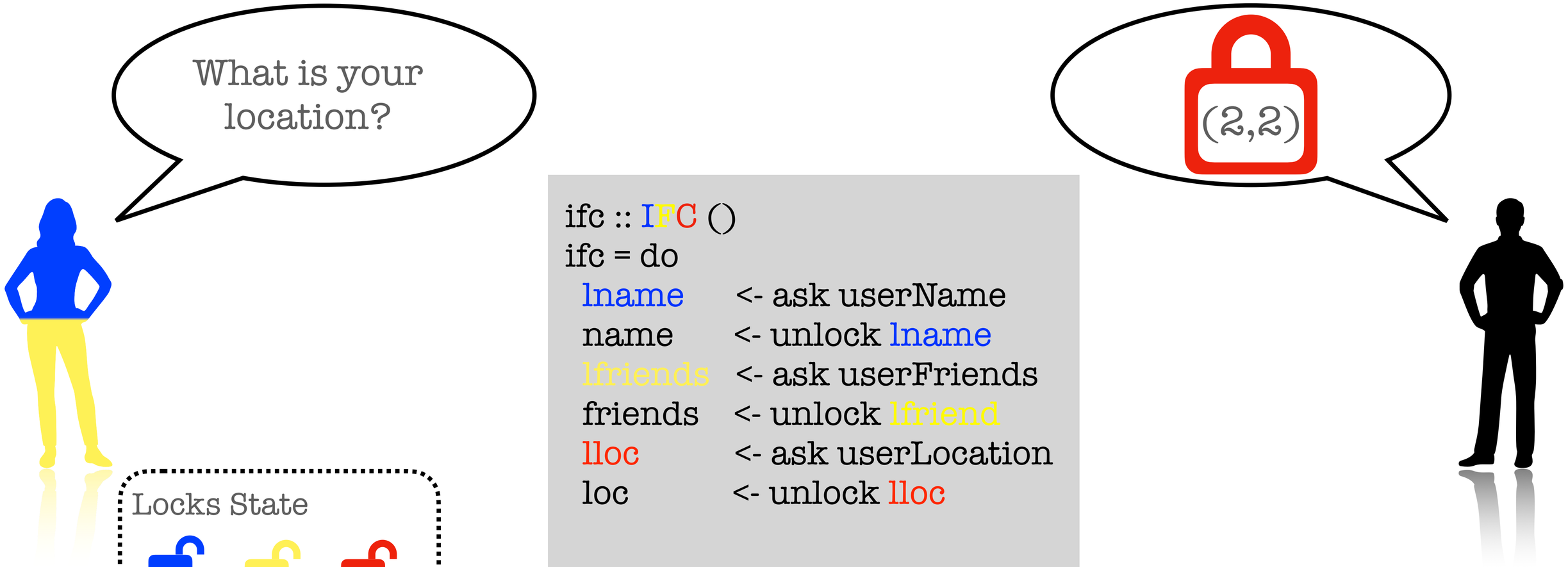

```
ifc :: IFC _
ifc = do
  lname     <- ask userName
  name      <- unlock lname
  lfriends  <- ask userFriends
  friends   <- unlock lfriend
```

Monadic Information Flow Control




```
ifc :: IFC ()
ifc = do
  lname     <- ask userName
  name      <- unlock lname
  lfriends  <- ask userFriends
  friends   <- unlock lfriend
  lloc      <- ask userLocation
  loc       <- unlock lloc
```

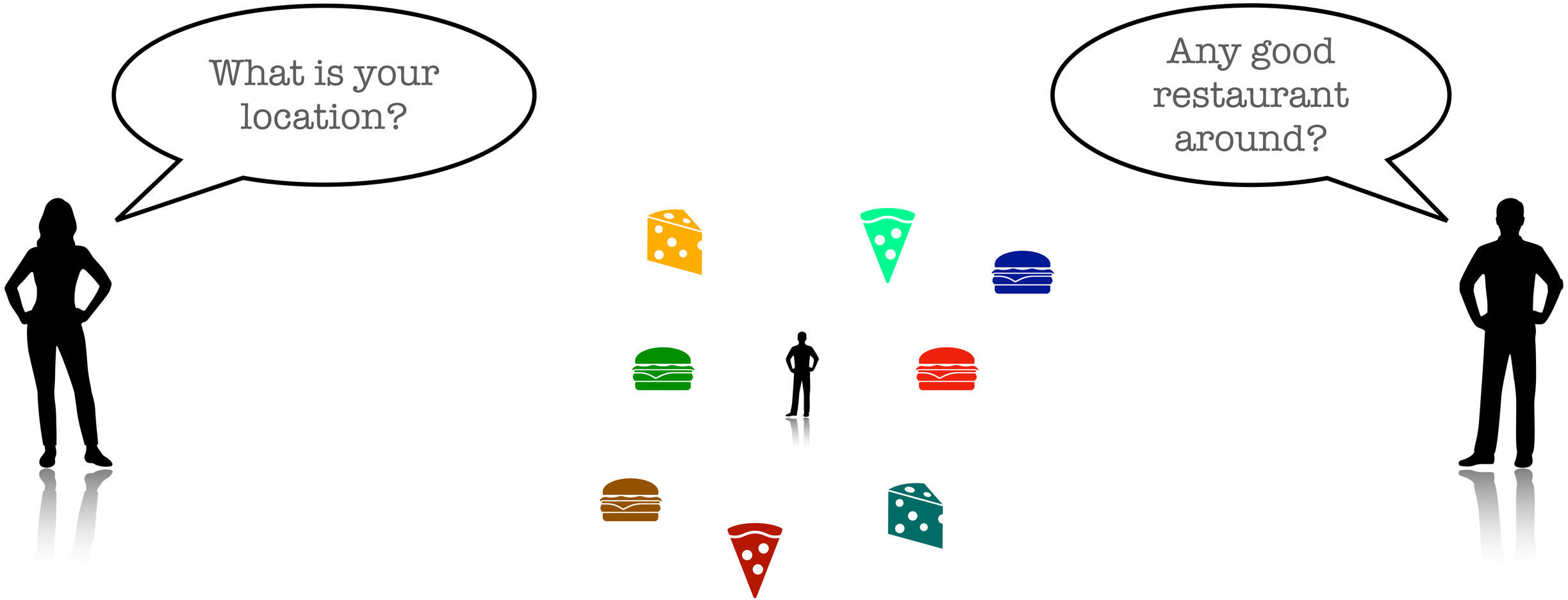
Exact Location is usually Well Protected



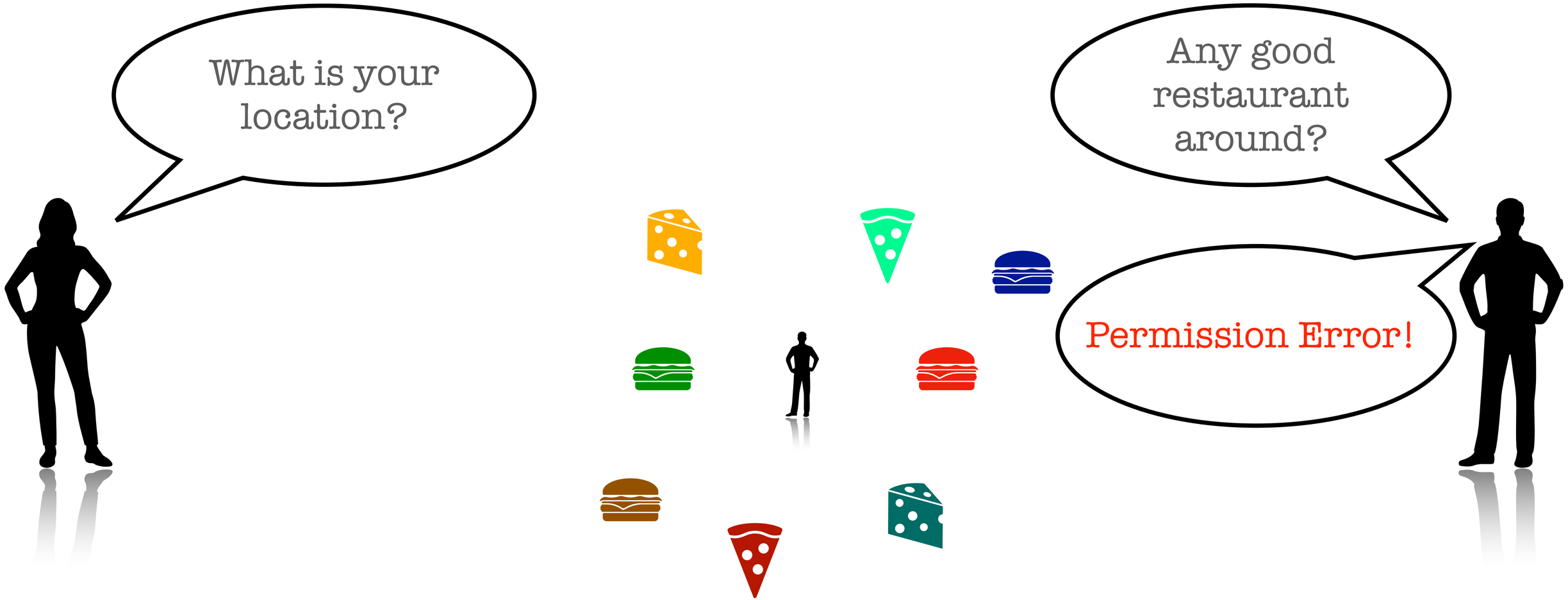

```
ifc :: IFC ()
ifc = do
  lname     <- ask userName
  name      <- unlock lname
  lfriends  <- ask userFriends
  friends   <- unlock lfriend
  lloc      <- ask userLocation
  loc       <- unlock lloc


>> runIFC ifc p
>> Permission Error
```

Approximate Location is Required by Location-Based Apps

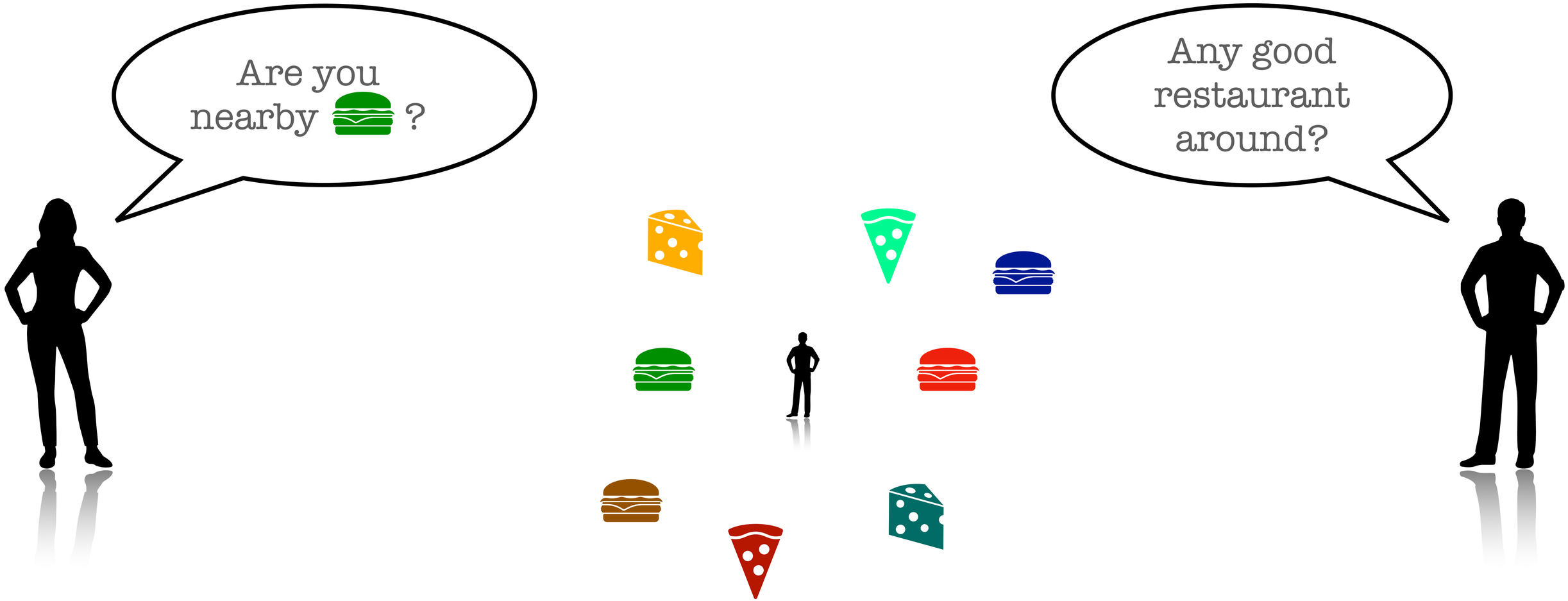
Approximate Location is Required by Location-Based Apps



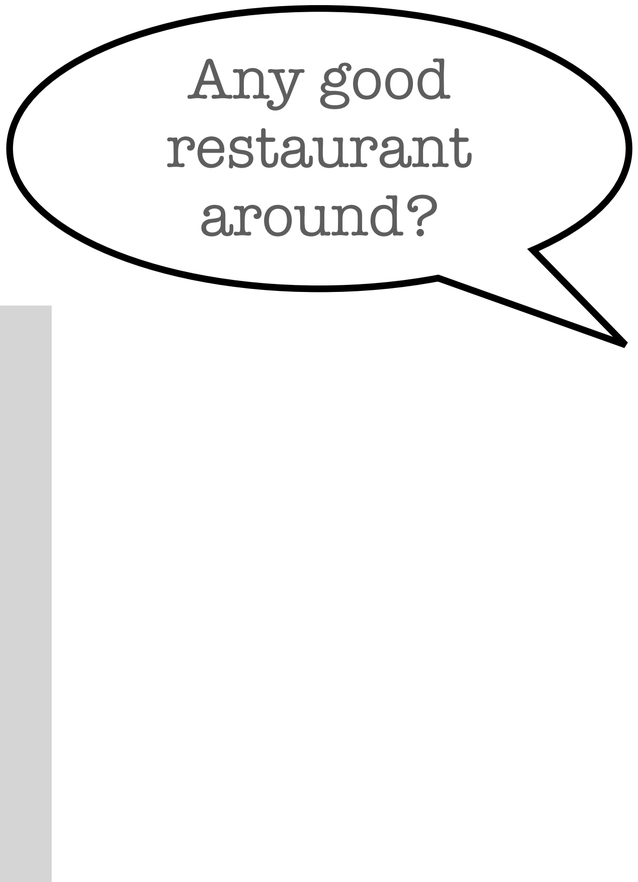
Approximate Location is Required by Location-Based Apps



Approximate Location is Required by Location-Based Apps

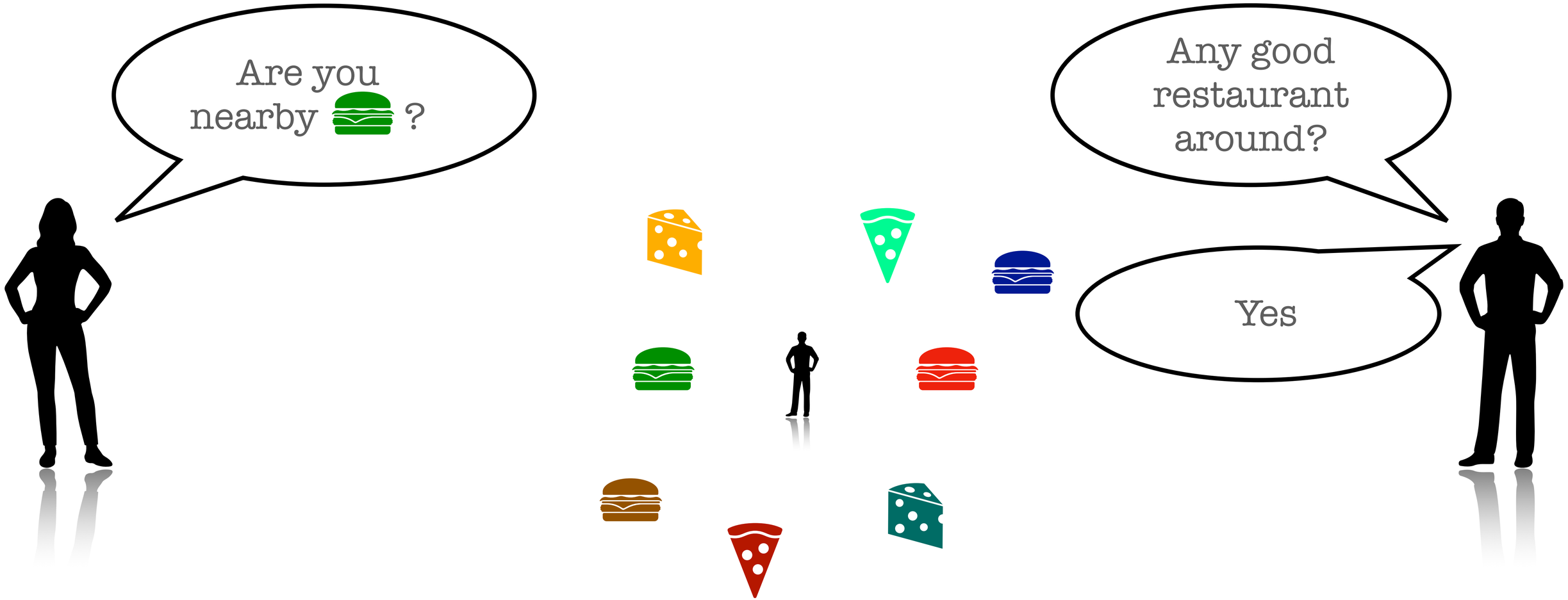


Downgrade for Approximate Location Information

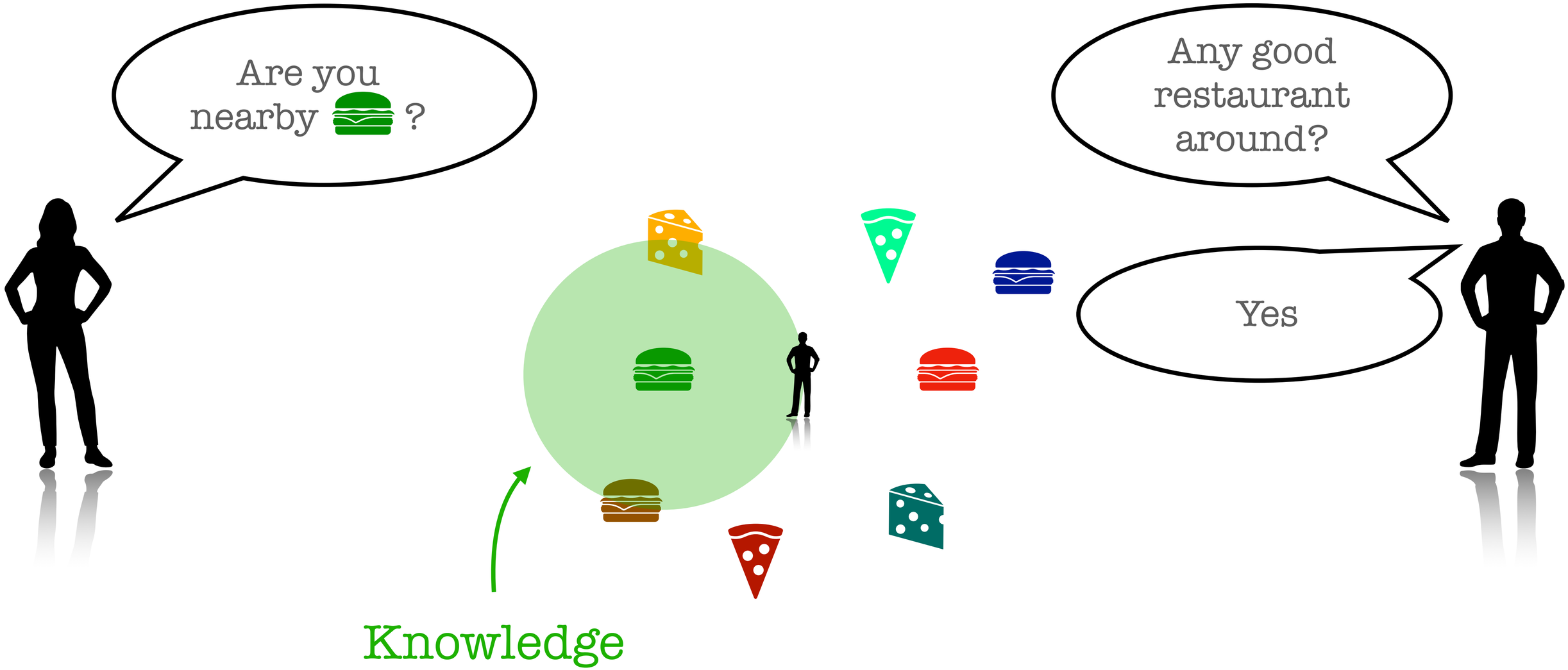



```
ifc :: IFC Bool
ifc = do
  lloc      <- ask userLocation
  isclose   <- downgrade lloc
                         nearbyPlace


downgrade :: Locked s
                  → (s → Bool)
                  → IFC Bool
```

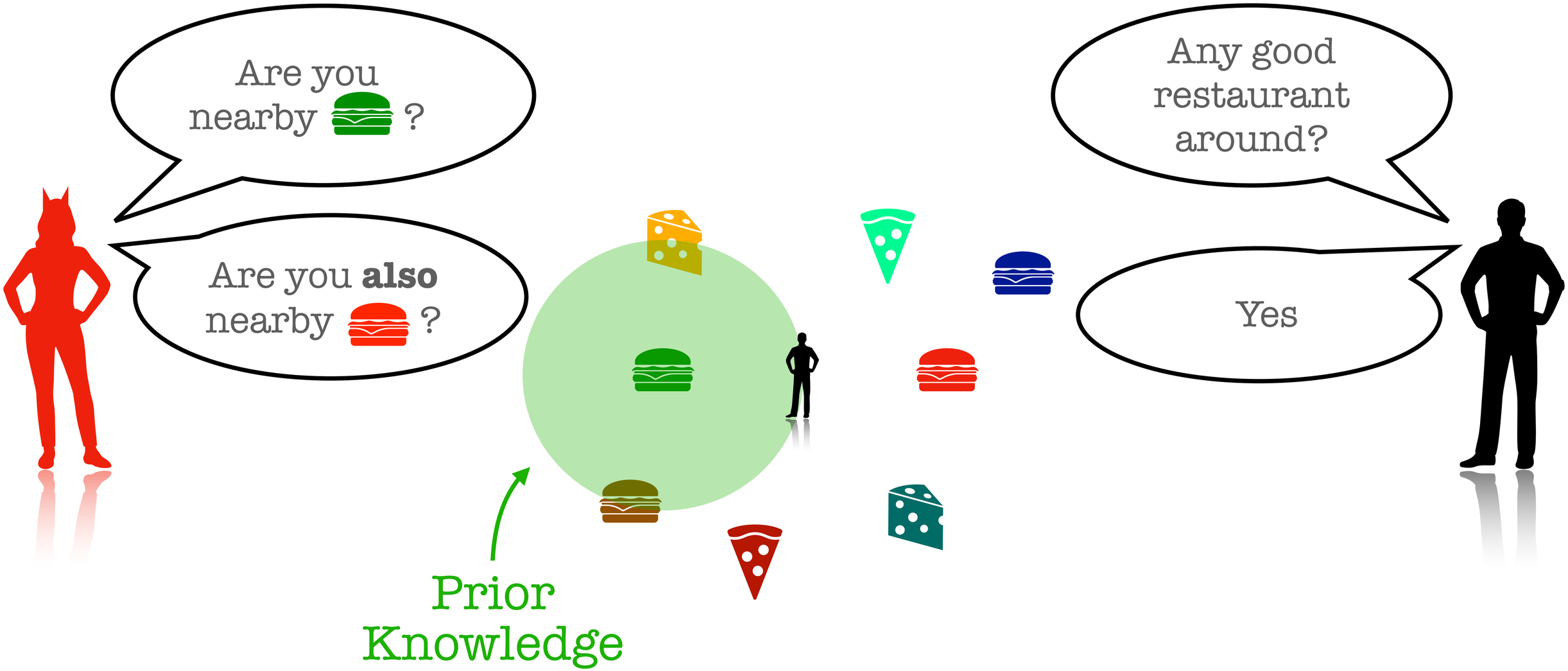
Downgrade for Approximate Location Information



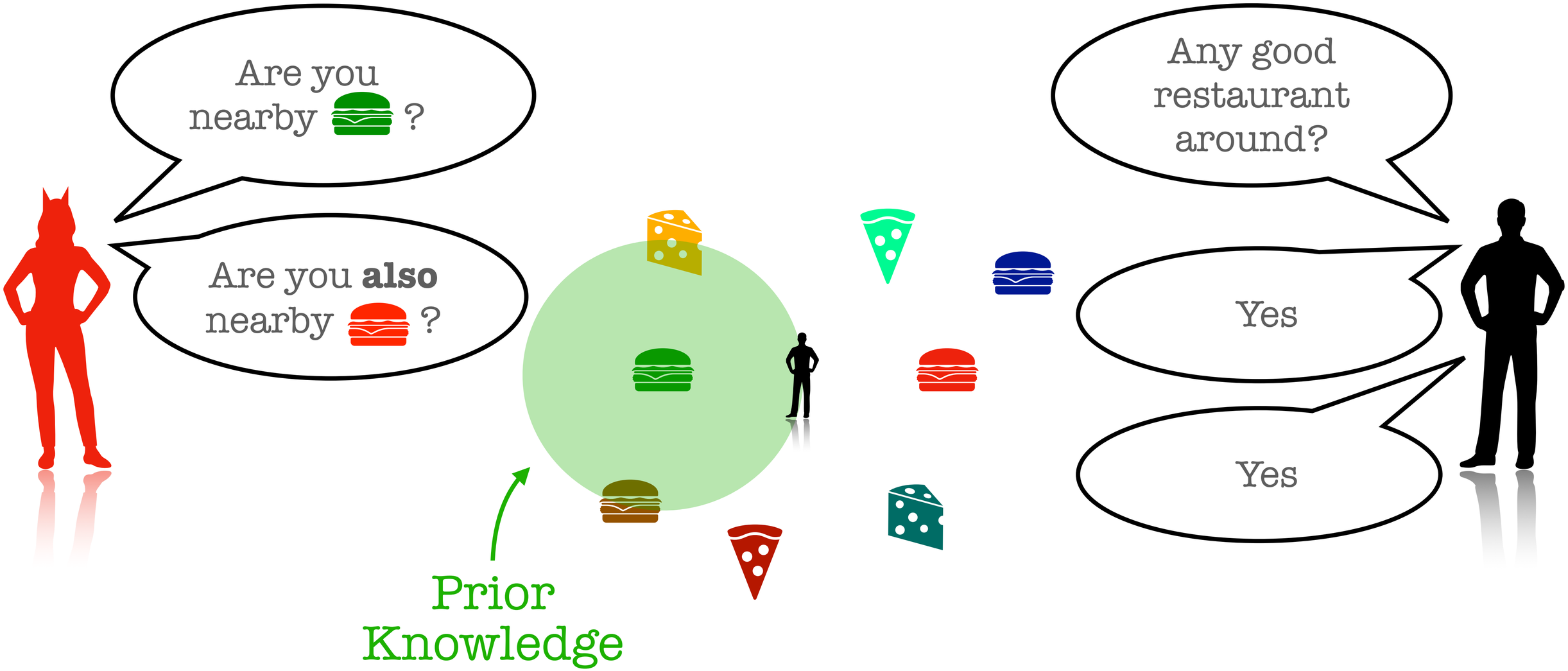
Downgrade for Approximate Location Information



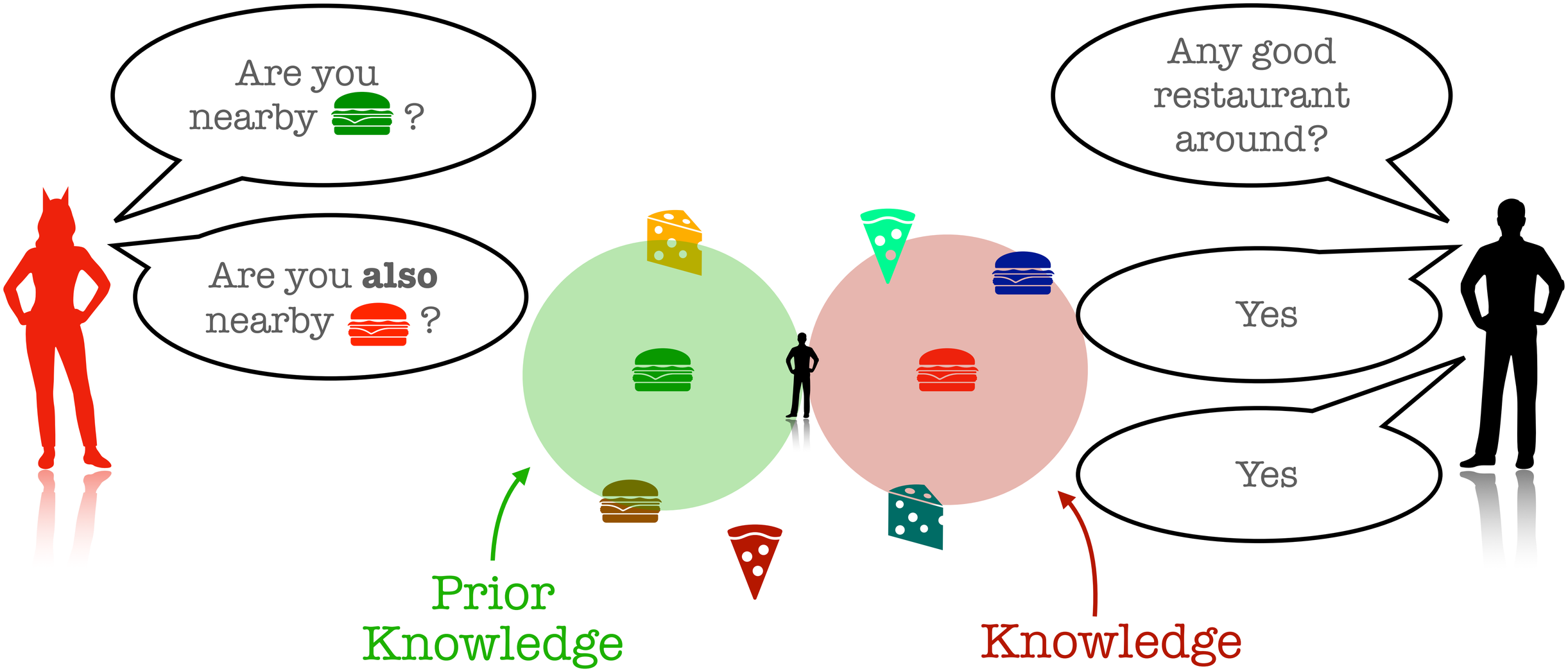
Downgrade for Approximate Location Information



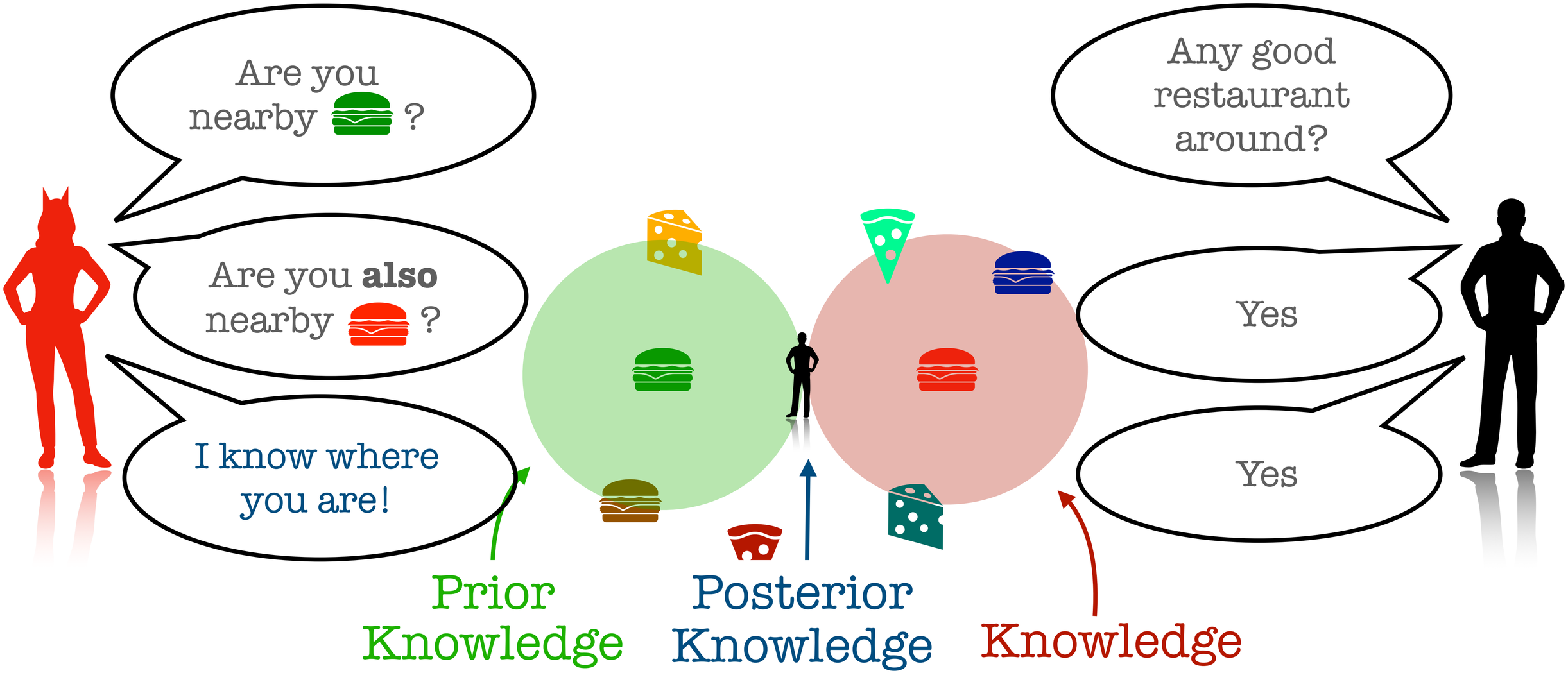
Downgrade for Approximate Location Information



Downgrade for Approximate Location Information



Downgrade for Approximate Location Information



Problem: Downgrade can Leak Exact Location

Bounded Downgrade:
Answer only when posterior is “general enough”

Problem: Downgrade can Leak Exact Location

Bounded Downgrade:
Answer only when posterior is “general enough”

```
policy :: a → Bool
policy a = size a > 100
```

Bounded Downgrade:
Answer only when posterior is “general enough”

State

```
priors :: Map s a
policy :: a -> Bool
```

```
downgrade :: Locked s
                  → (s → Bool)
                  → IFC Bool
downgrade s q = do
  val      <- unprotect s
  prior    <- getPrior val
  let post = posterior q val prior
  if policy post
    then do setPrior val post
                  return $ q val
    else throwError “Policy Violation”
```

Bounded Downgrade:
Answer only when posterior is “general enough”

State

priors :: Map s a
policy :: a -> Bool

```
downgrade :: Locked s
                  → (s → Bool)
                  → IFC Bool
downgrade s q = do
  val      <- unprotect s
  prior    <- getPrior val
  let post = posterior q val prior
  if policy post
    then do setPrior val post
                  return $ q val
    else throwError “Policy Violation”
```

Observations:

No Dependence from IFC!

How to compute posterior?

Bounded Downgrade: AnosyT Monad Tranformer

```
AnosyT a s =
  StateT (StateA a s)
```

```
StateA

  priors :: Map s a
  policy :: a -> Bool
```

```
downgrade :: Knowledge a s, Monad m
                  , Lockable l => l s
                  → (s → Bool)
                  → AnosyT a s m Bool
downgrade s q = do
  val      <- unprotect s
  prior    <- getPrior val
  let post = posterior q val prior
  if policy post
    then do setPrior val post
                  return $ q val
    else throwError “Policy Violation”
```

Observations:

No Dependence from IFC!

How to compute posterior?

Bounded Downgrade: AnosyT Monad Tranformer

```
AnosyT a s =
  StateT (StateA a s)
```

```
StateA

  priors :: Map s a
  policy :: a -> Bool
```

On top of any Haskell IFC system: LIO, LWeb, STORM

```
downgrade :: Knowledge a s, Monad m
                  , Lockable l => l s
                  → (s → Bool)
                  → AnosyT a s m Bool
downgrade s q = do
  val      <- unprotect s
  prior    <- getPrior val
  let post = posterior q val prior
  if policy post
    then do setPrior val post
                  return $ q val
    else throwError “Policy Violation”
```

Observations:

No Dependence from IFC!

How to compute posterior?

Posterior Knowledge Computation

Goal: Definition of

```
posterior :: Knowledge a s
          => (s → Bool)    — query
          → s              — secret
          → a              — prior
          → a
```

Posterior Knowledge Computation

Knowledge: A Set of Secrets

```
class Knowledge a s where
  ⊤ :: a
  ⊥ :: a
  ∈ :: s → a → Bool
  ⊆ :: a → a → Bool
  ∩ :: a → a → a
```

Goal: Definition of

```
posterior :: Knowledge a s
          => (s → Bool)    — query
          → s              — secret
          → a              — prior
          → a
```

Posterior Knowledge Computation

Knowledge: A Set of Secrets

```
class Knowledge a s where
  ⊤ :: a
  ⊥ :: a
  ∈ :: s → a → Bool
  ⊆ :: a → a → Bool
  ∩ :: a → a → a
```

Goal: Definition of

```
posterior :: Knowledge a s
          => (s → Bool)    — query
          → s              — secret
          → a              — prior
          → a
```

Challenge:

Precise Operations can be Uncomputable

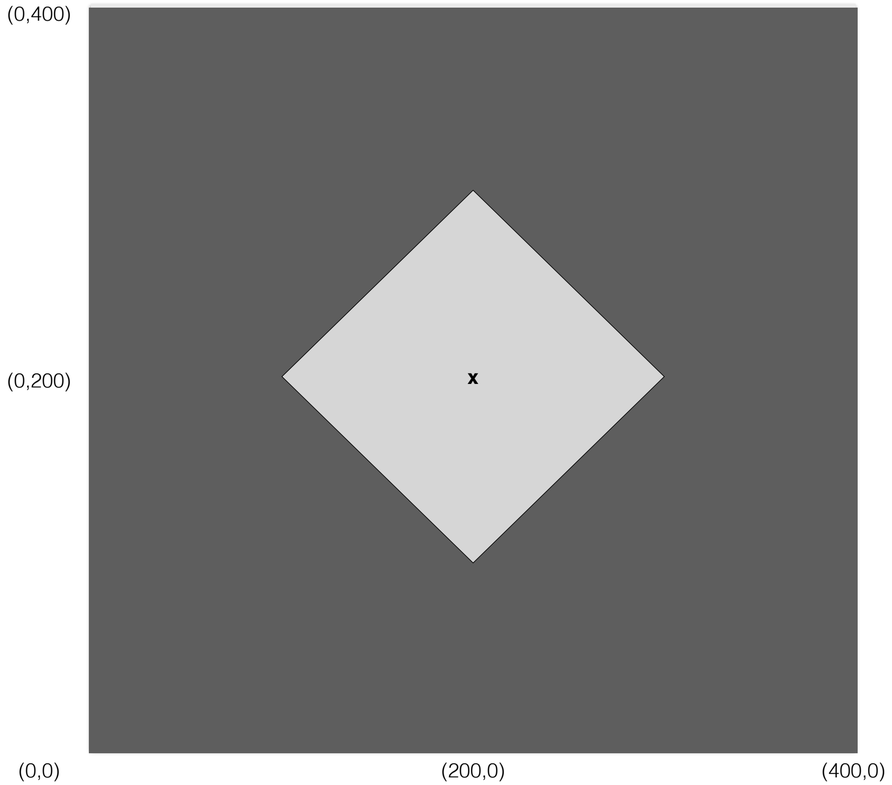
Solution:

Use Abstract Domains!

Knowledge Representation

```
data Loc = L {x :: Int, y :: Int}

nearby :: Loc  → Loc → Bool
nearby (L xo yo) (L x y) =
  abs (x - xo) (y -yo) <= 100
```

```
type S = Loc

query :: S → Bool
query s = nearby (L 200 200)
```



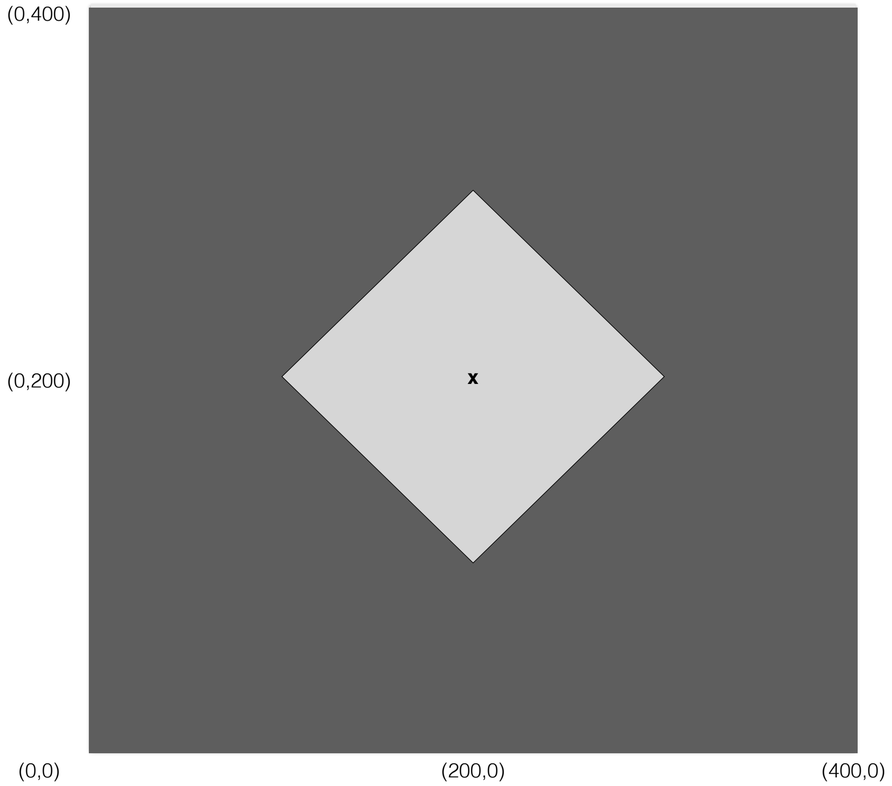
Knowledge on query
(Separation into yes/no areas)

Knowledge Representation

Difficult to Represent

```
data Loc = L {x :: Int, y :: Int}

nearby :: Loc  → Loc → Bool
nearby (L xo yo) (L x y) =
  abs (x - xo) (y -yo) <= 100
```

```
type S = Loc

query :: S → Bool
query s = nearby (L 200 200)
```



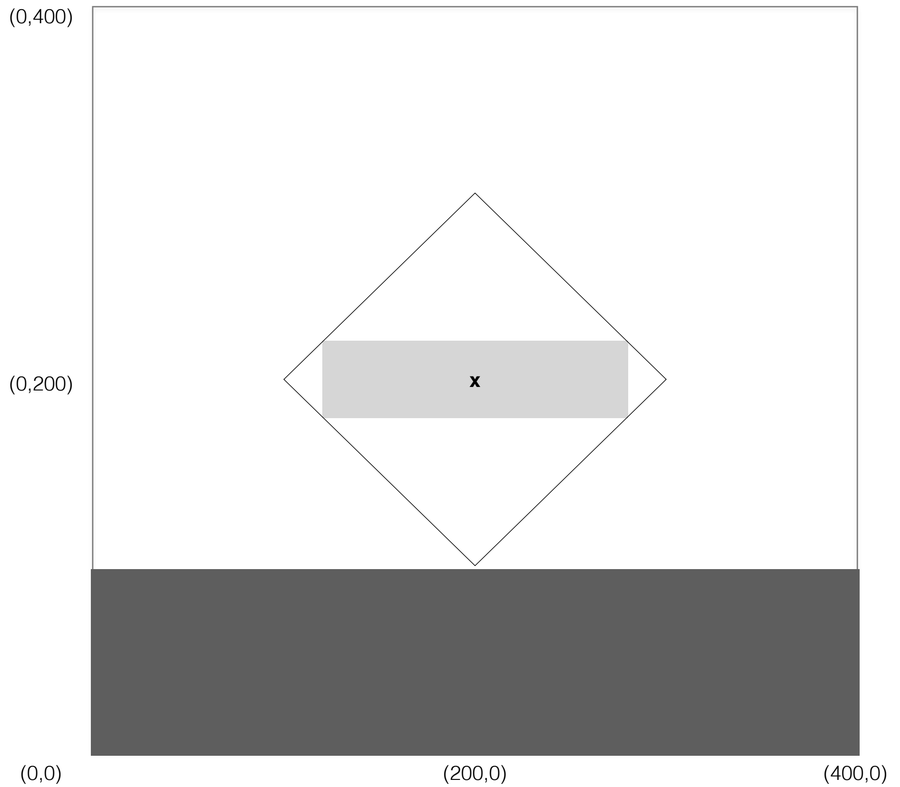
Knowledge on query
(Separation into yes/no areas)

Knowledge Approximation

```
type S = Loc —  = L {x :: Int, y :: Int}
query :: S → Bool
query s = nearby (L 200 200)
```

```
data KInt = KInt {lo :: Int, hi :: Int}
data K    = K {kx :: KInt, ky :: KInt}
instance Knowledge K S where ...
```

```
approx :: (K,K)
approx = (yesK, noK)
yesK = K (KInt 121 279) (KInt 179 221)
noK  = K (KInt 0   400)  (KInt 0   99)
```



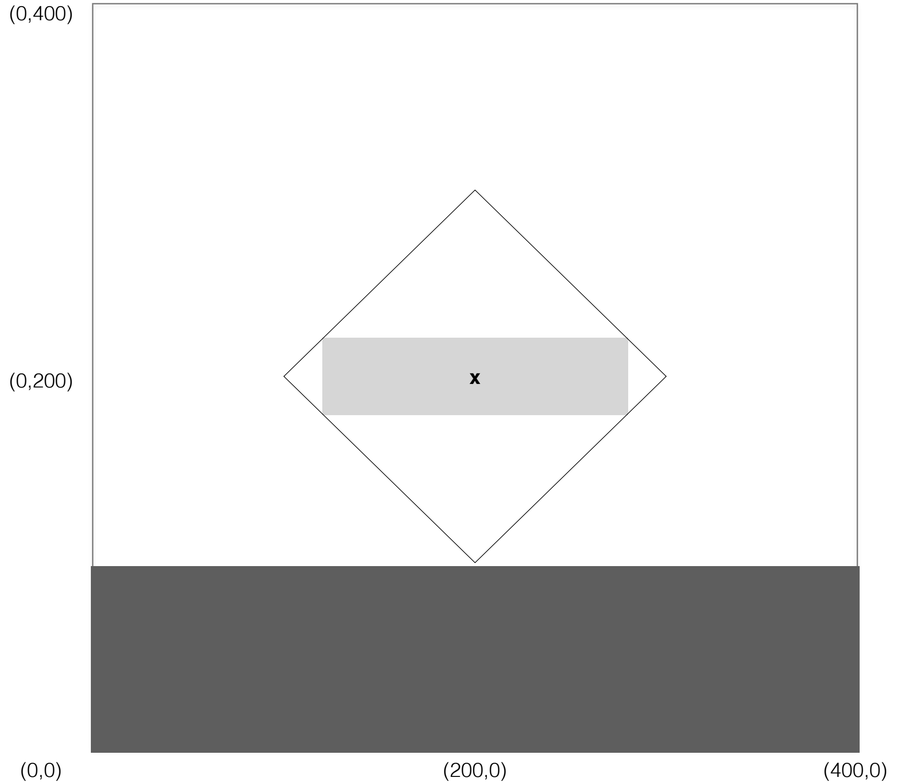
Interval Knowledge Approximation

Knowledge Approximation

```
approx :: (K,K)
approx = (yesK, noK)
yesK = K (KInt 121 279) (KInt 179 221)
noK  = K (KInt 0   400) (KInt 0   99)
```

Back to our Goal, posterior is easy to define

```
posterior :: S → K → K
posterior s p = p ∩
  if query s then  yesK else noK
```



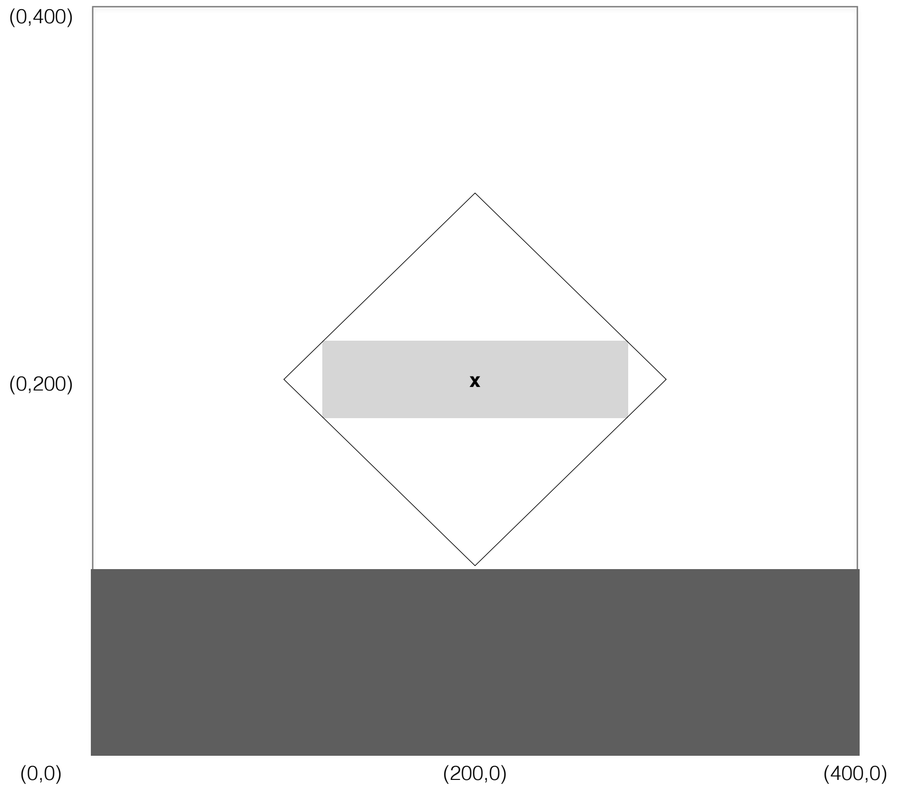
Interval Knowledge Approximation

Knowledge Approximation

```
approx :: (K,K)
approx = (yesK, noK)
yesK = K (KInt 121 279) (KInt 179 221)
noK  = K (KInt 0   400) (KInt 0   99)
```

Back to our Goal, posterior is easy to define

```
posterior :: S → K → K
posterior s p = p ∩
  if query s then  yesK else noK
```



Interval Knowledge Approximation

How do I come up with approx?

How do I know it is correct?

Refinement Types to the Rescue!

```
approx :: (K,K)
approx = (yesK, noK)
yesK = K (KInt 121 279) (KInt 179 221)
noK  = K (KInt 0   400) (KInt 0   99)
```

```
yesK :: K<\s -> query s>
noK  :: K<\s -> not (query s)>
```

k :: K<p>

All elements inside* k satisfy p

* Knowledge contains secrets

LiquidHaskell

Refinement Types for Synthesis!

Goal: yesK :: K<\s -> query s>

Refinement Types for Synthesis!

Goal: yesK :: K<\s -> query s>

Step 1: Syntax Directed

```
yesK = K (KInt xl xh) (KInt yl yh)
```

Refinement Types for Synthesis!

Goal: yesK :: K<\s -> query s>

Step 1: Syntax Directed

yesK = K (KInt xl xh) (KInt yl yh)

Step 2: Constraint Generation

$\forall$ x, y. xl $\leq$ x $\leq$ xh $\wedge$ yl $\leq$ y $\leq$ yh $\Rightarrow$ query (L x y)
maximize xh - xl
maximize yh - yl

Refinement Types for Synthesis!

Goal: yesK :: K<\s -> query s>

Step 1: Syntax Directed

yesK = K (KInt xl xh) (KInt yl yh)

Step 2: Constraint Generation

$\forall$ x, y. xl $\leq$ x $\leq$ xh $\wedge$ yl $\leq$ y $\leq$ yh $\Rightarrow$ query (L x y)
maximize xh - xl
maximize yh - yl

Step 3: SMT solves constraints

xl := 121, xh := 279, yl := 179 , yh := 221

Refinement Types for Synthesis!

Goal: yesK :: K<\s -> query s>

Step 1: Syntax Directed

yesK = K (KInt xl xh) (KInt yl yh)

Step 2: Constraint Generation

$\forall$ x, y. xl $\leq$ x $\leq$ xh $\wedge$ yl $\leq$ y $\leq$ yh $\Rightarrow$ query (L x y)
maximize xh - xl
maximize yh - yl

Step 3: SMT solves constraints

xl := 121, xh := 279, yl := 179 , yh := 221

Finally: Plug in the holes

yesK = K (KInt 121 279) (KInt 179 221)

Anosy: Bounded Downgrade

AnosyT keeps track of prior leaked knowledge

Challenge: compute posterior
Knowledge Abstraction + Synthesis.

For each query, abstract knowledge is
synthesised via SMT
verified via Liquid Haskell

Implementation

AnosyT + GHC plugin to process queries.

Queries are straight line Bool functions

Secrets are products of Ints

Abstract Domains are Intervals and their Powersets

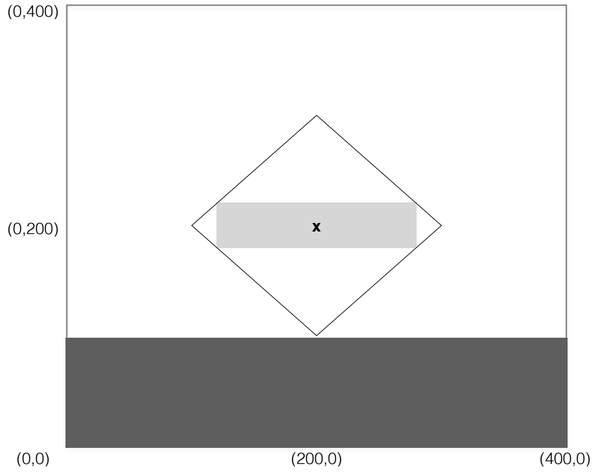
Implementation

AnosyT + GHC plugin to process queries.

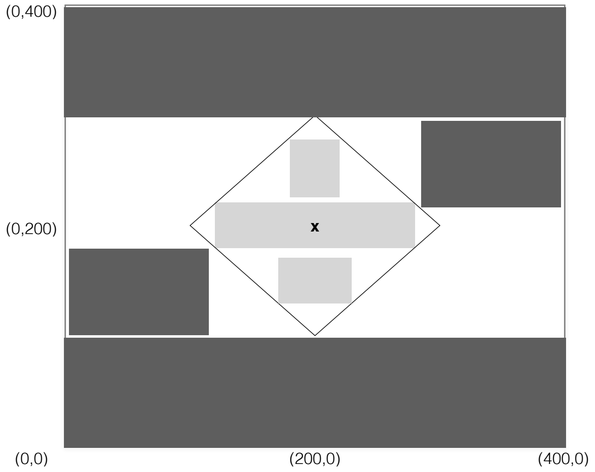
Queries are straight line Bool functions

Secrets are products of Ints

Abstract Domains are Intervals and their Powersets



Interval Knowledge



Powerset Knowledge

Evaluation

#	Name	No. of fields	Size of ind. sets
B1	Birthday	2	259 / 13246
B2	Ship	3	1.01e+06 / 2.43e+07
B3	Photo	3	4 / 884
B4	Pizza	4	1.37e+10 / 2.81e+13
B5	Travel	4	2160 / 6.72e+06

We evaluate with secrets with multiple dimensions

#	Under-approximation							
	Size	% diff.	Verif. time	Synth. time				Synth. time
B1	259 / 9620	0 / 27	2.78 ± 0.03	1.11 ± 0.01	259 / 13505	0 / 2	2.64 ± 0.03	1.07 ± 0.01
B2	2.21e+05 / 1.01e+07	78 / 58	3.62 ± 0.02	9.26 ± 0.04	2.02e+06 / 2.54e+07	100 / 5	3.17 ± 0.02	4.00 ± 0.12
B3	4 / 664	0 / 25	3.12 ± 0.06	0.90 ± 0.07	4 / 888	0 / 0	2.83 ± 0.03	0.90 ± 0.01
B4	3.53e+04 / 1.35e+05	100 / 100	3.66 ± 0.04	20.92 ± 0.11	9.22e+12 / 2.81e+13	67200 / 0	3.29 ± 0.08	10.87 ± 0.01
B5	360 / 5.04e+06	83 / 25	3.81 ± 0.04	1.38 ± 0.04	35460 / 6.72e+06	1542 / 0	3.47 ± 0.04	0.89 ± 0.01

(a) Interval abstract domain

#	Under-approximation				Over-approximation			
	Size	% diff.	Verif. time	Synth. time	Size	% diff.	Verif. time	Synth. time
B1	259 / 13246	0 / 0	4.51 ± 0.05	1.13 ± 0.02	259 / 13505	0 / 2	4.34 ± 0.03	1.08 ± 0.01
B2	6.78e+05 / 1.62e+07	33 / 33	5.32 ± 0.09	14.34 ± 0.11	1.80e+06 / 2.54e+07	78 / 5	5.17 ± 0.02	4.89 ± 0.09
B3	4 / 880	0 / 0	5.29 ± 0.09	1.07 ± 0.03	4 / 888	0 / 0	4.99 ± 0.03	1.03 ± 0.01
B4	3.88e+05 / 4.00e+05	100 / 100	5.78 ± 0.03	54.89 ± 0.23	9.22e+12 / 2.81e+13	67200 / 0	5.48 ± 0.08	30.57 ± 0.07
B5	720 / 6.70e+06	67 / 0	6.02 ± 0.07	13.26 ± 0.09	6300 / 6.72e+06	192 / 0	5.96 ± 0.04	15.25 ± 0.03

(b) Powerset of intervals with size 3

* Mardziel, Magill, Hicks, and Srivatsa. J. Comp. Sec. 2013. Dynamic enforcement of knowledge-based security policies using probabilistic abstract interpretation.

Evaluation

#	Name	No. of fields	Size of ind. sets
B1	Birthday	2	259 / 13246
B2	Ship	3	1.01e+06 / 2.43e+07
B3	Photo	3	4 / 884
B4	Pizza	4	1.37e+10 / 2.81e+13
B5	Travel	4	2160 / 6.72e+06

#	Under-approximation				Over-approximation			
	Size	% diff.	Verif. time	Synth. time	Size	% diff.	Verif. time	Synth. time
B1	259 / 9620	0 / 27	2.78 ± 0.03	1.11 ± 0.01	259 / 13505	0 / 2	2.64 ± 0.03	1.07 ± 0.01
B2	2.21e+05 / 1.01e+07	78 / 58	3.62 ± 0.02	9.26 ± 0.04	2.02e+06 / 2.54e+07	100 / 5	3.17 ± 0.02	4.00 ± 0.12
B3	4 / 664	0 / 25	3.12 ± 0.06	0.90 ± 0.07	4 / 888	0 / 0	2.83 ± 0.03	0.90 ± 0.01
B4	3.53e+04 / 1.35e+05	100 / 100	3.66 ± 0.04	20.92 ± 0.11	9.22e+12 / 2.81e+13	67200 / 0	3.29 ± 0.08	10.87 ± 0.01
B5	360 / 5.04e+06	83 / 25	[illegible]	1.38 ± 0.04	35460 / 6.72e+06	1542 / 0	3.47 ± 0.04	0.89 ± 0.01

Synthesis times are reasonable. Static analysis tools such as Prob* are faster. But ANOSY synthesises knowledge at compile time.

Interval abstract domain

				Over-approximation			
			Synth. time	Size	% diff.	Verif. time	Synth. time
			1.13 ± 0.02	259 / 13505	0 / 2	4.34 ± 0.03	1.08 ± 0.01
			14.34 ± 0.11	1.80e+06 / 2.54e+07	78 / 5	5.17 ± 0.02	4.89 ± 0.09
B3	4 / 880	0 / 0	5.29 ± 0.09	1.07 ± 0.03	4 / 888	0 / 0	4.99 ± 0.03	1.03 ± 0.01
B4	3.88e+05 / 4.00e+05	100 / 100	5.78 ± 0.03	54.89 ± 0.23	9.22e+12 / 2.81e+13	67200 / 0	5.48 ± 0.08	30.57 ± 0.07
B5	720 / 6.70e+06	67 / 0	6.02 ± 0.07	13.26 ± 0.09	6300 / 6.72e+06	192 / 0	5.96 ± 0.04	15.25 ± 0.03

(b) Powerset of intervals with size 3

* Mardziel, Magill, Hicks, and Srivatsa. J. Comp. Sec. 2013. Dynamic enforcement of knowledge-based security policies using probabilistic abstract interpretation.

Evaluation

#	Name	No. of fields	Size of ind. sets
B1	Birthday	2	259 / 13246
B2	Ship	3	1.01e+06 / 2.43e+07
B3	Photo	3	4 / 884
B4	Pizza	4	1.37e+10 / 2.81e+13
B5	Travel	4	2160 / 6.72e+06

#	Under-approximation				Over-approximation			
	Size	% diff.	Verif. time	Synth. time	Size	% diff.	Verif. time	Synth. time
B1	259 / 9620	0 / 27	2.78 ± 0.03	1.11 ± 0.01	259 / 13505	0 / 2	2.64 ± 0.03	1.07 ± 0.01
B2	2.21e+05 / 1.01e+07	78 / 58	3.62 ± 0.02	9.26 ± 0.04	2.02e+06 / 2.54e+07	100 / 5	3.17 ± 0.02	4.00 ± 0.12
B3	4 / 664	0 / 25	3.12 ± 0.06	0.90 ± 0.07	4 / 888	0 / 0	2.83 ± 0.03	0.90 ± 0.01
B4	3.53e+04 / 1.35e+05	100 / 100	3.66 ± 0.04	20.92 ± 0.11	9.22e+12 / 2.81e+13	67200 / 0	3.29 ± 0.08	10.87 ± 0.01
B5	360 / 5.04e+06	83 / 25	3.81 ± 0.04	1.38 ± 0.04	35460 / 6.72e+06	1542 / 0	3.47 ± 0.04	0.89 ± 0.01

Approximation accuracy gets better with increasing the size of the powerset.

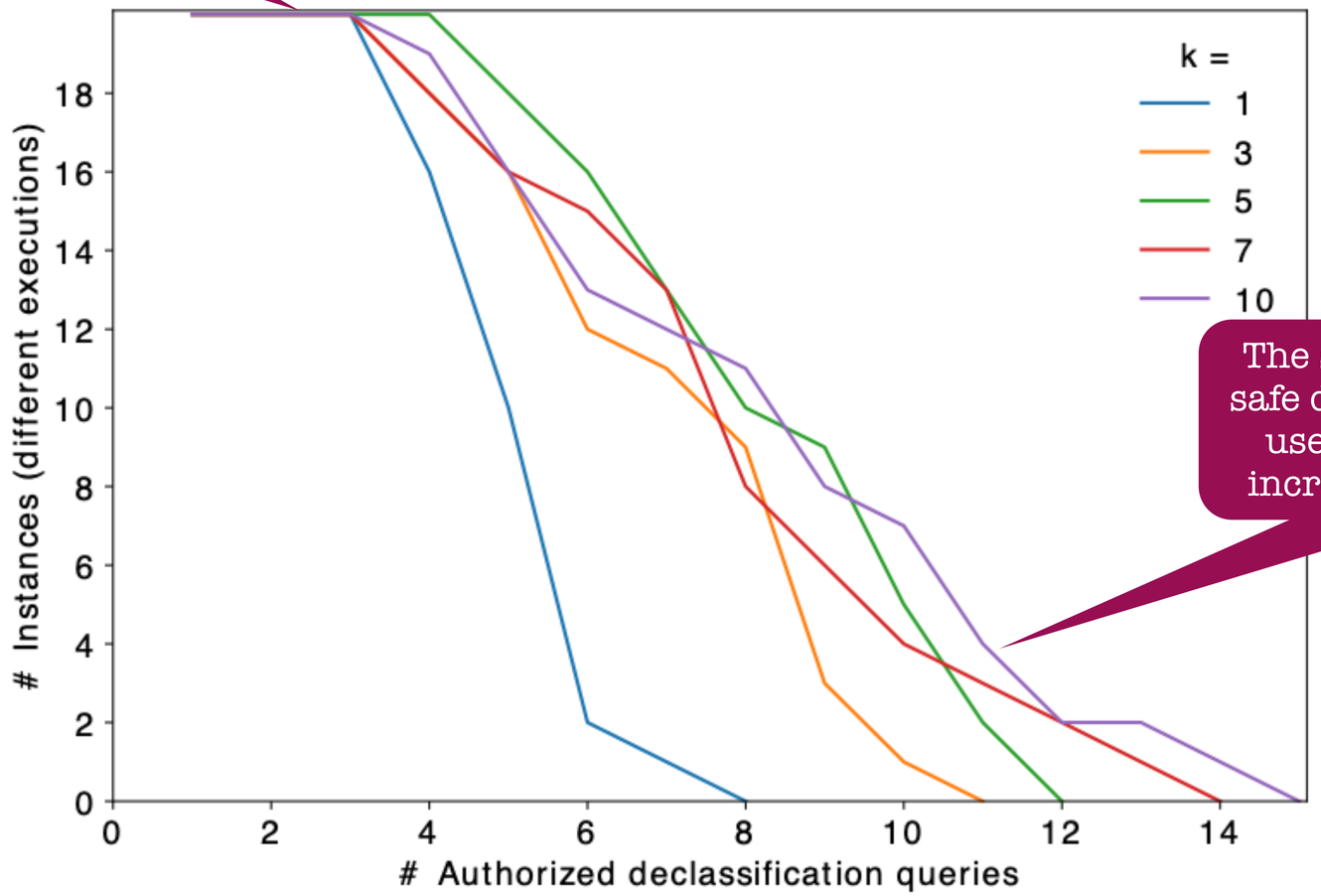
#	Under-approximati				roximation			
	Size	% diff.	Ver		f.	Verif. time	Synth. time	
B1	259 / 13246	0 / 0	4.51 ± 0.05	± 0.02	259 / 13505	0 / 2	4.34 ± 0.03	1.08 ± 0.01
B2	6.78e+05 / 1.62e+07	33 / 33	5.32 ± 0.09	14.34 ± 0.11	1.80e+06 / 2.54e+07	78 / 5	5.17 ± 0.02	4.89 ± 0.09
B3	4 / 880	0 / 0	5.29 ± 0.09	1.07 ± 0.03	4 / 888	0 / 0	4.99 ± 0.03	1.03 ± 0.01
B4	3.88e+05 / 4.00e+05	100 / 100	5.78 ± 0.03	54.89 ± 0.23	9.22e+12 / 2.81e+13	67200 / 0	5.48 ± 0.08	30.57 ± 0.07
B5	720 / 6.70e+06	67 / 0	6.02 ± 0.07	13.26 ± 0.09	6300 / 6.72e+06	192 / 0	5.96 ± 0.04	15.25 ± 0.03

(b) Powerset of intervals with size 3

* Mardziel, Magill, Hicks, and Srivatsa. J. Comp. Sec. 2013. Dynamic enforcement of knowledge-based security policies using probabilistic abstract interpretation.

Evaluation: Location Example

All these queries are verified safe for the nearby query.



The system handles more safe declassifications of if a user is nearby with the increase in powerset size

Summary

Haskell is Ideal for IFC (e.g., LIO, LWeb, STORM)

Realistic IFC apps downgrade (potentially leaking info)

Anosy: Bounded Downgrade

AnosyT keeps track of prior leaked knowledge

For each query, abstract knowledge is
synthesised via SMT
verified via Liquid Haskell

Anosy: Bounded Downgrade

AnosyT keeps track of prior leaked knowledge

For each query, abstract knowledge is
synthesised via SMT
verified via Liquid Haskell

Anosy @ PLDI

nikivazou

Thanks!

institute iMdea software

Looking for
PhDs/interns