AURA: A programming language for authorization and audit

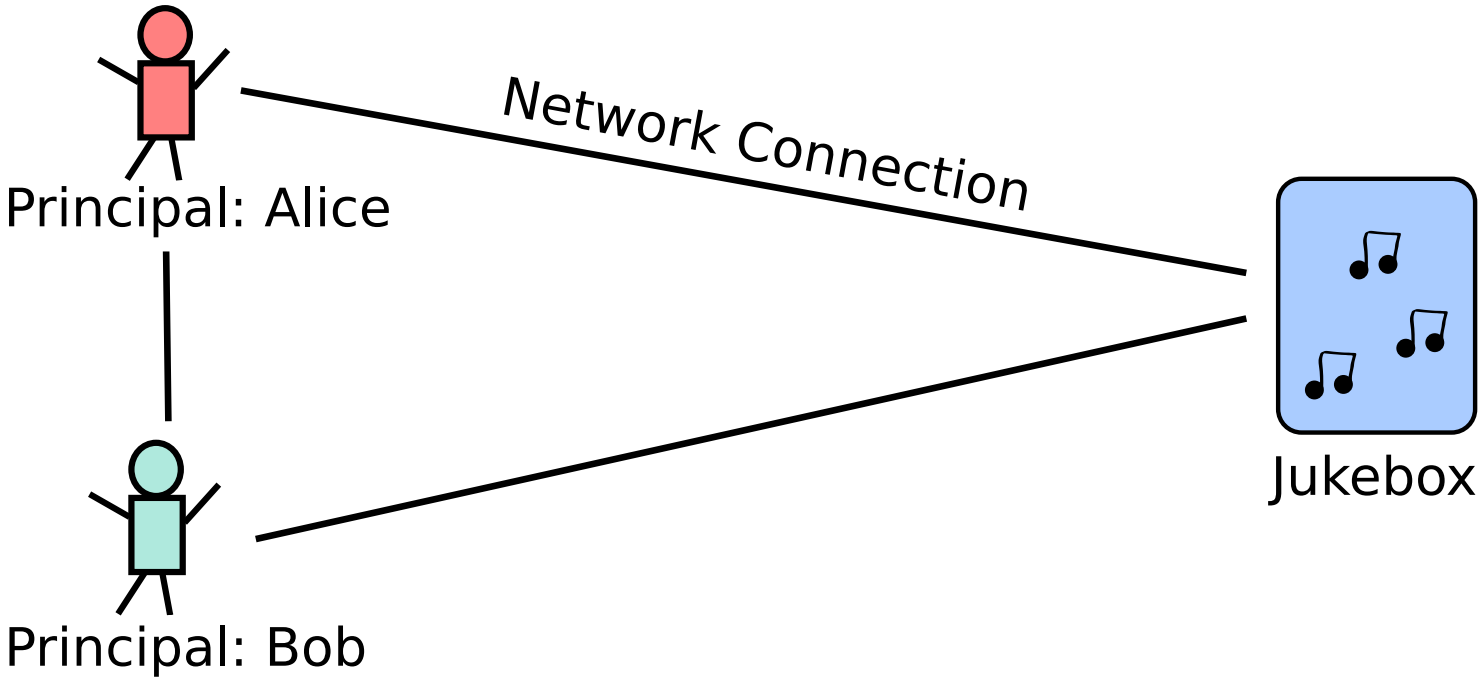
Jeff Vaughan
Limin Jia, Karl Mazurak, Jianzhou Zhao, Luke Zarko,
Joseph Schorr, and Steve Zdancewic

Department of Computer and Information Science
University of Pennsylvania

ICFP
September 22, 2008

PENN PL

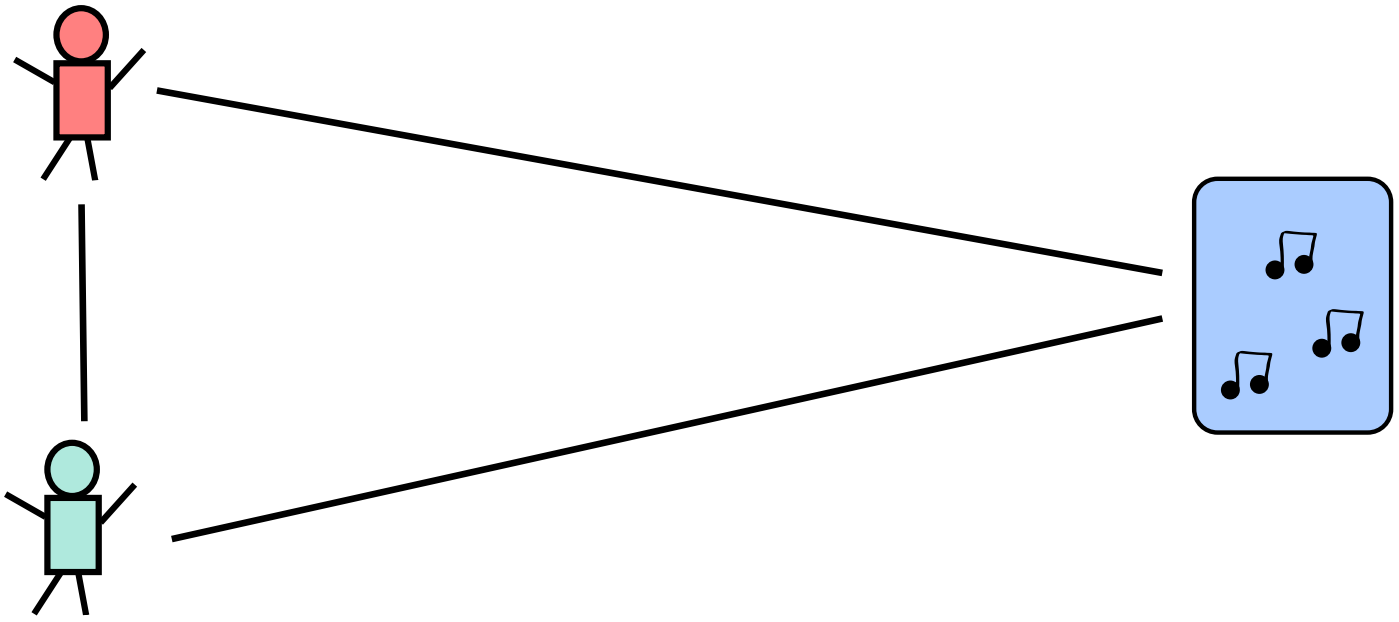
A distributed access control example



Jukebox's signature:

playFor_raw: (s: Song) → (p: **prin**) → Unit

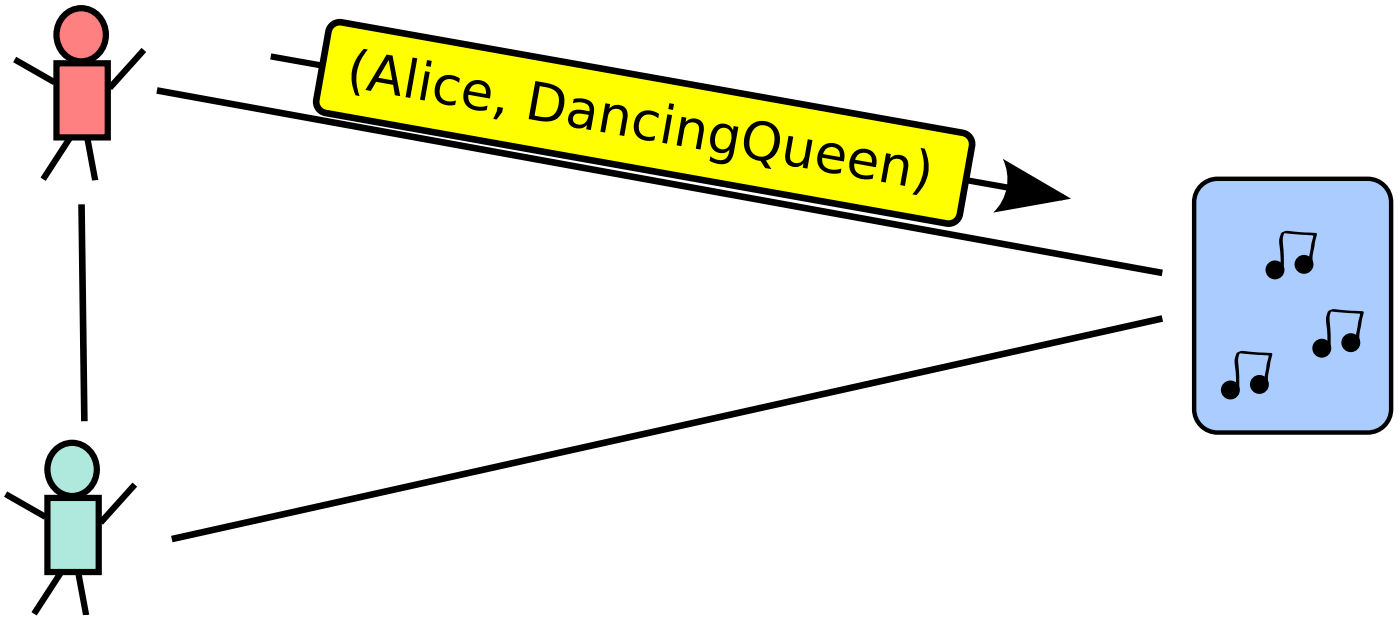
A distributed access control example



Jukebox's signature:

playFor_raw: (s: Song) → (p: **prin**) → Unit

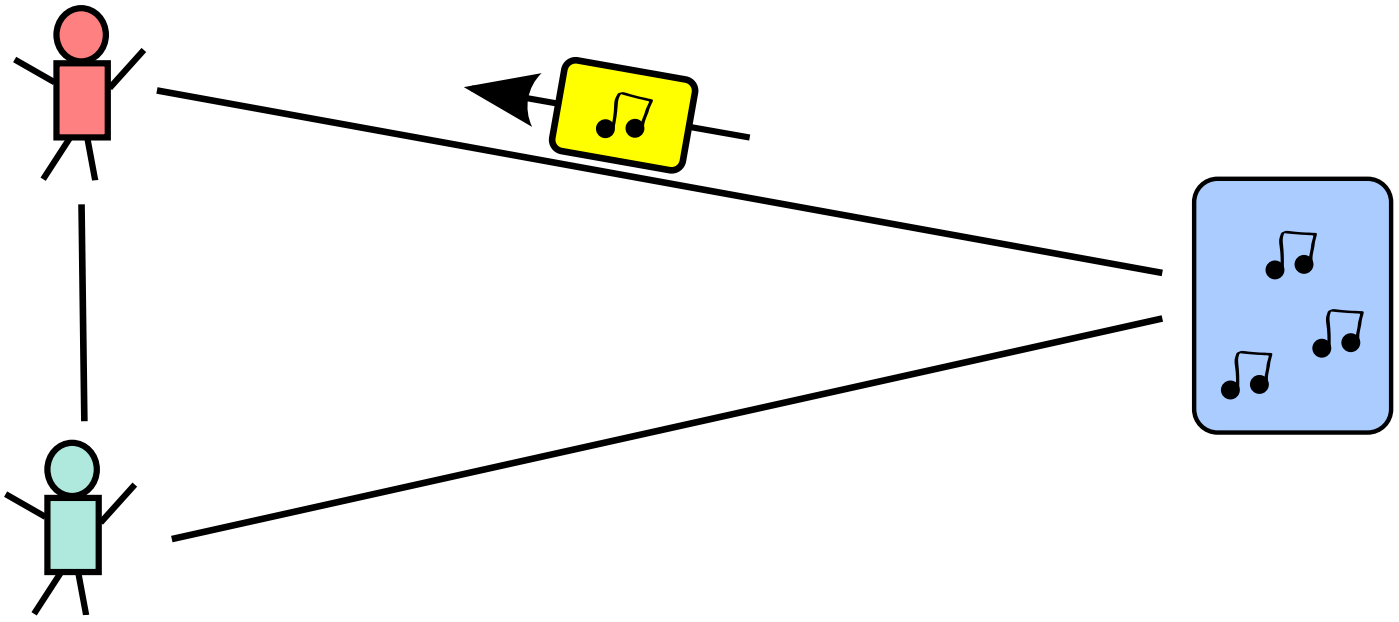
A distributed access control example



Jukebox's signature:

playFor_raw: (s: Song) → (p: **prin**) → Unit

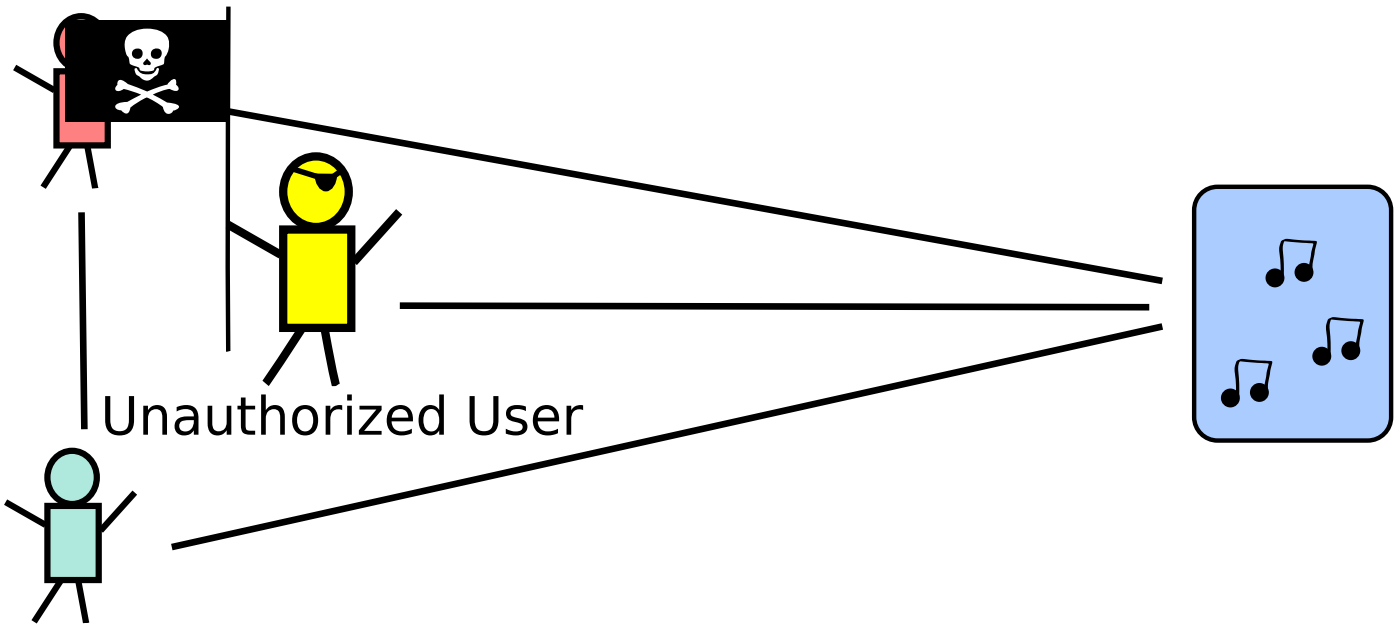
A distributed access control example



Jukebox's signature:

playFor_raw: (s: Song) → (p: **prin**) → Unit

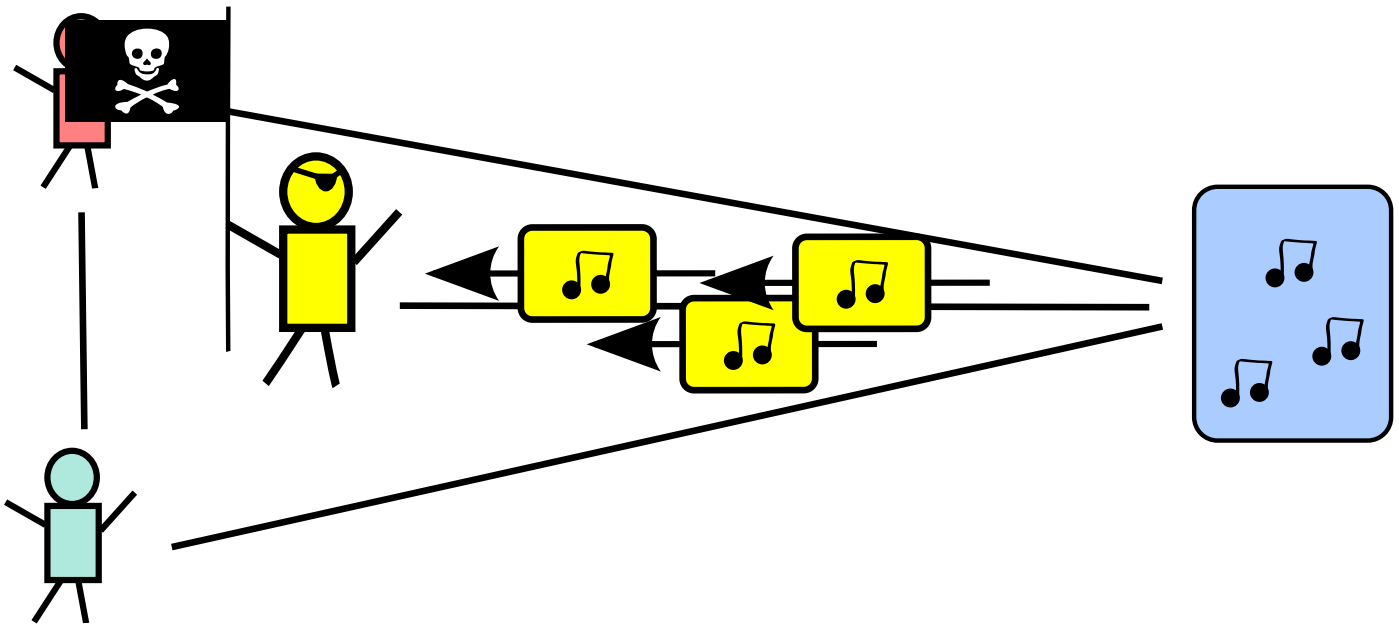
A distributed access control example



Jukebox's signature:

$$\text{playFor\_raw: } (\text{s: Song}) \rightarrow (\text{p: } \textbf{prin}) \rightarrow \text{Unit}$$

A distributed access control example



Jukebox's signature:

playFor_raw: (s: Song) → (p: **prin**) → Unit

International Cartel for Fonograph Players Policy

Policy Statement (Simple):

- Songs have one or more owners.
- An owner may authorize principals to play songs he owns.

International Cartel for Fonograph Players Policy

Policy Statement (Simple):

- Songs have one or more owners.
- An owner may authorize principals to play songs he owns.

Policy Enforcement Problems (Hard):

- distributed decision making
- mutual distrust
- prominent use of delegation

AURA: Enforce policy with proof carrying access control.

- Programs build *proofs* attesting to their access rights.
- Proof components
  - standard rules of inference
  - *evidence* capturing principal intent (e.g. signatures)
- AURA runtime:
  - checks proof structure (well-typedness)
  - logs appropriate proofs for later audit

 Proof Carrying Code [Necula+ 98], Grey Project [Bauer+ 05], Protocol Analysis [Fournet+ 07], Evidence-Based Audit [CSF 08]

Encoding policy at the ICFP server

shareRule ≡ **self says** (
(o: **prin**) → (s: Song) → (r: **prin**) →
(Owns o s) →
(o **says** (MayPlay r s)) →
(MayPlay r s)))

playFor: (s: Song) → (p: **prin**) →
pf (**self says** (MayPlay p s)) → Unit

AURA features above: **pf**, **self**, **says**, dependency, effects...

Encoding policy at the ICFP server

shareRule ≡ **self says** (
(o: **prin**) → (s: Song) → (r: **prin**) →
(Owns o s) →
(o **says** (MayPlay r s)) →
(MayPlay r s)))

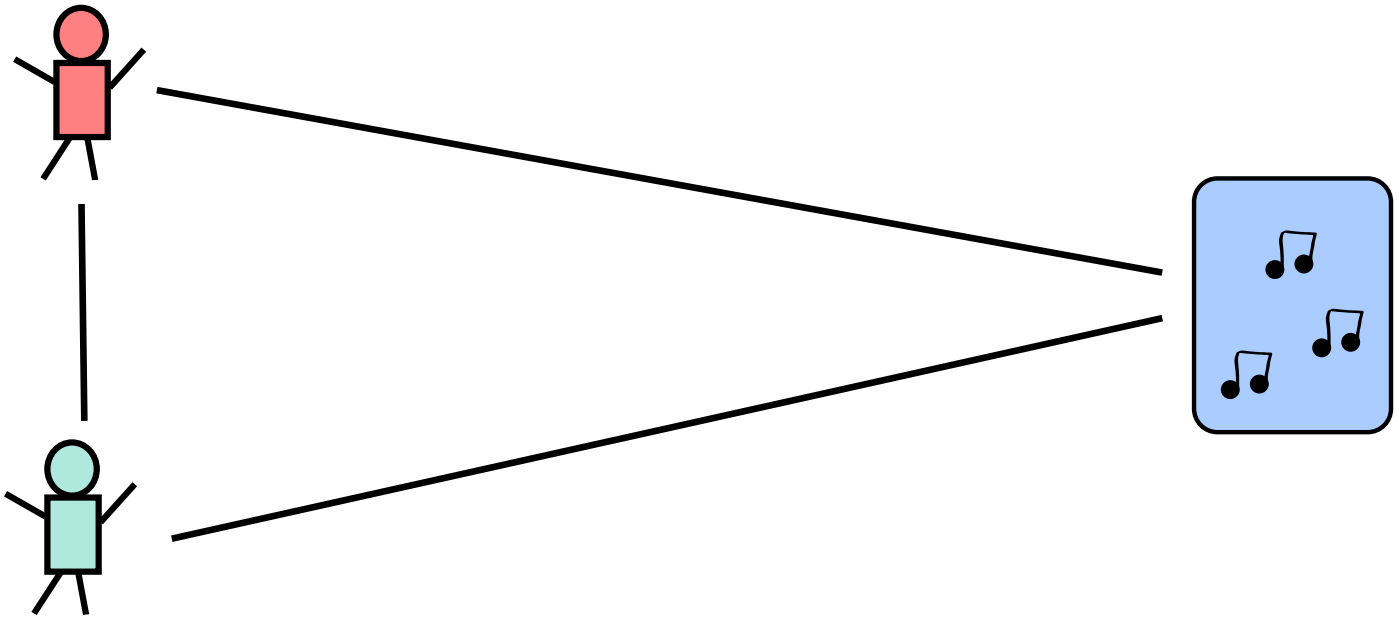
playFor: (s: Song) → (p: **prin**) →
pf (**self says** (MayPlay p s)) → Unit

Key Property

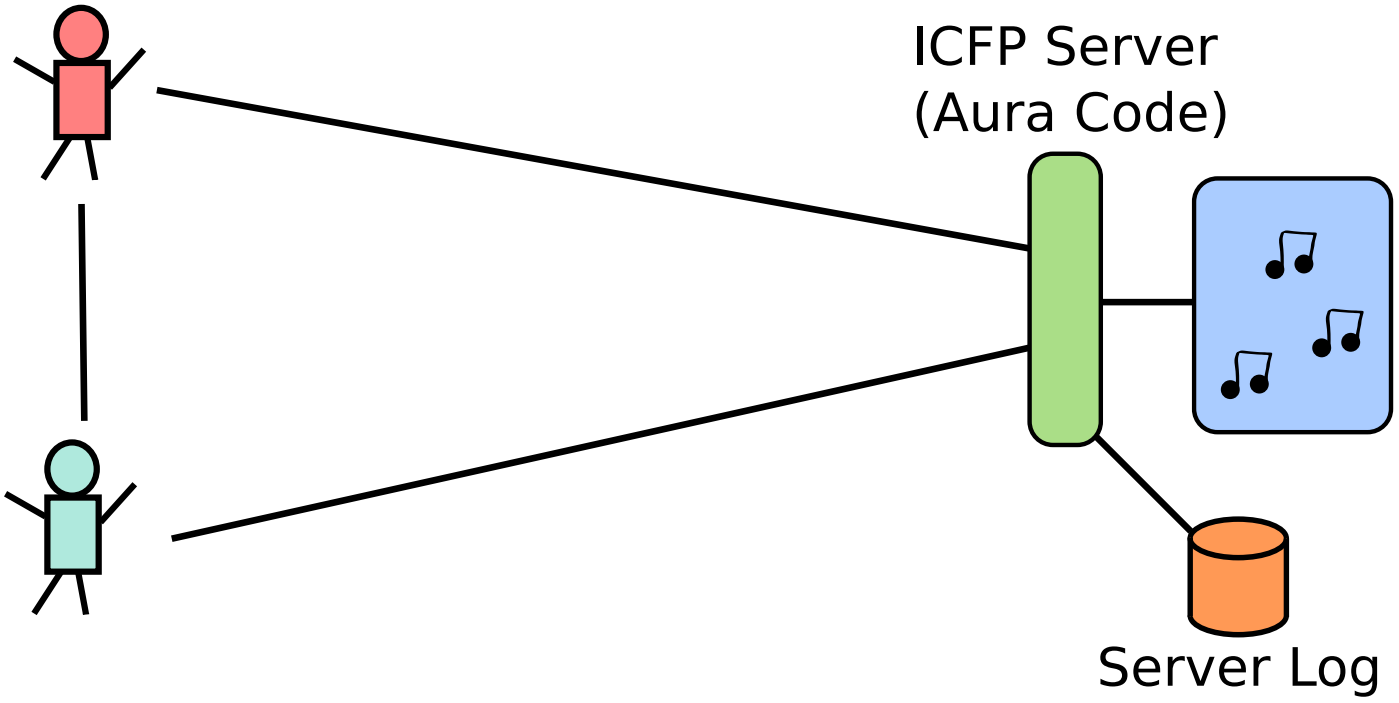
A program can only call playFor when it has an appropriate access control proof.

AURA features above: **pf**, **self**, **says**, dependency, effects...

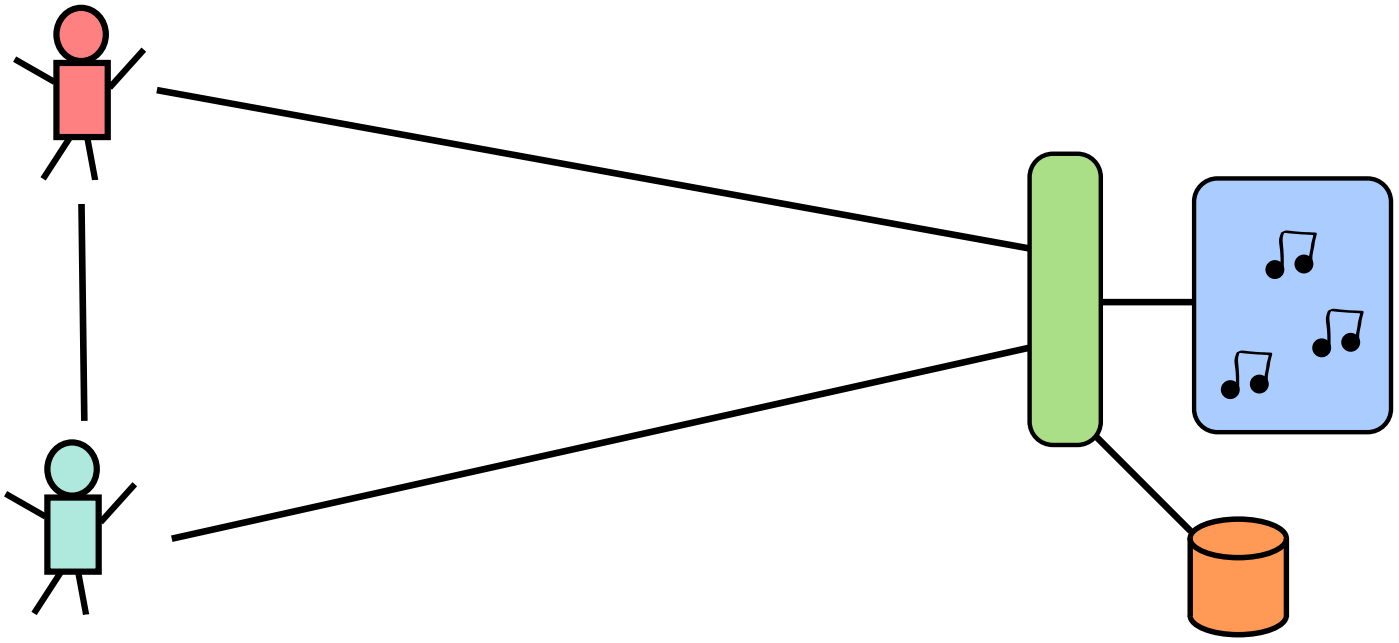
Using the ICFP policy.



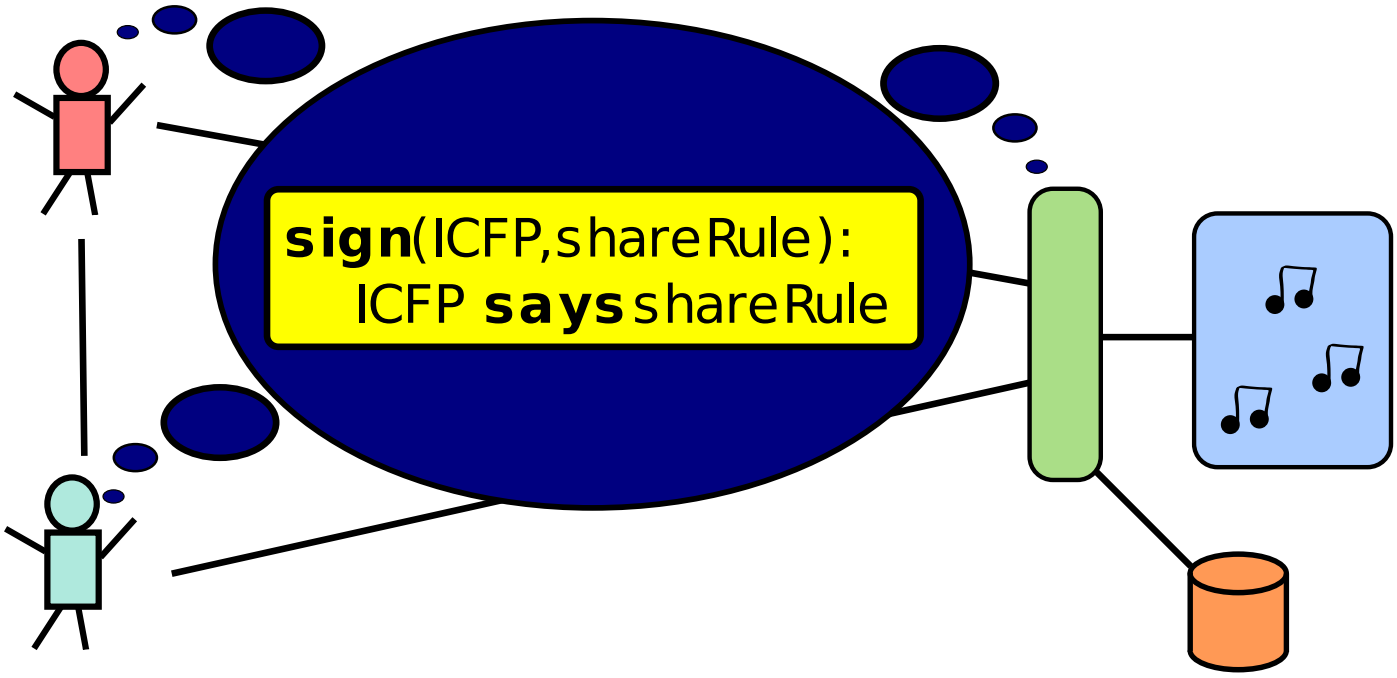
Using the ICFP policy.



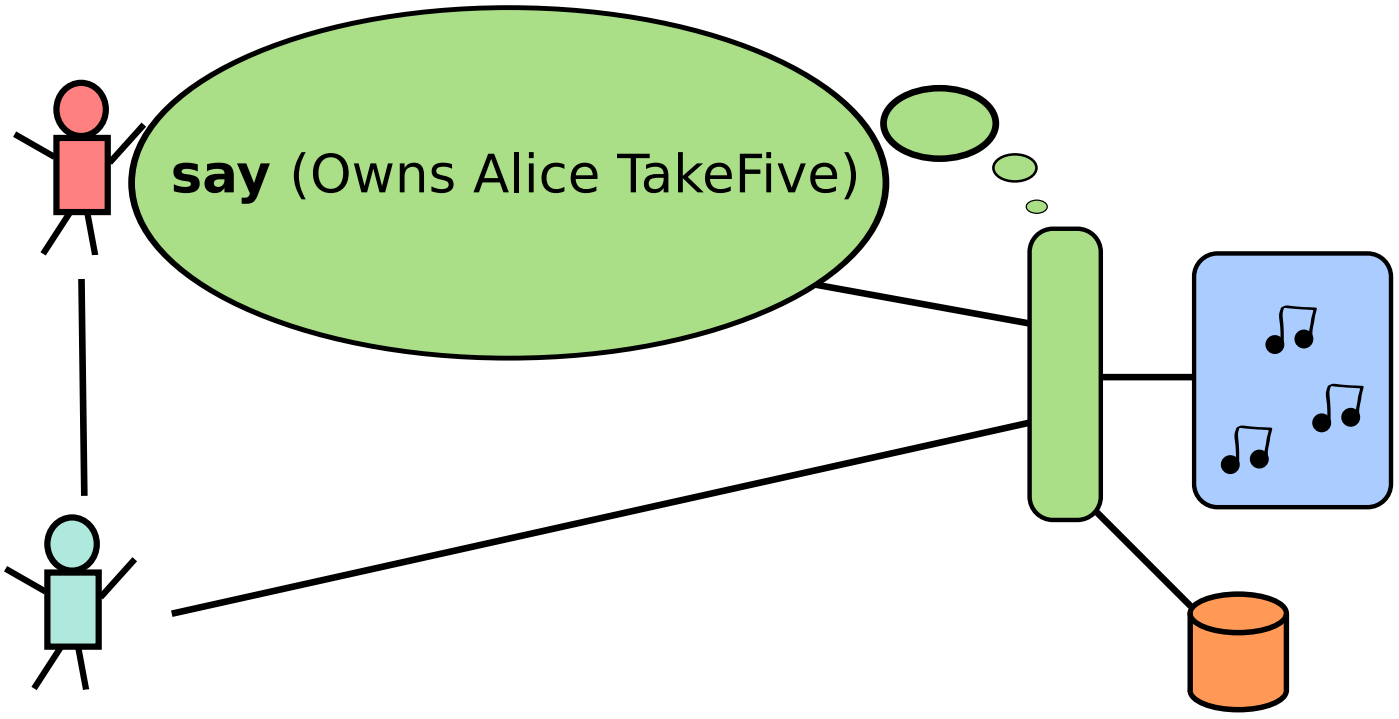
Using the ICFP policy.



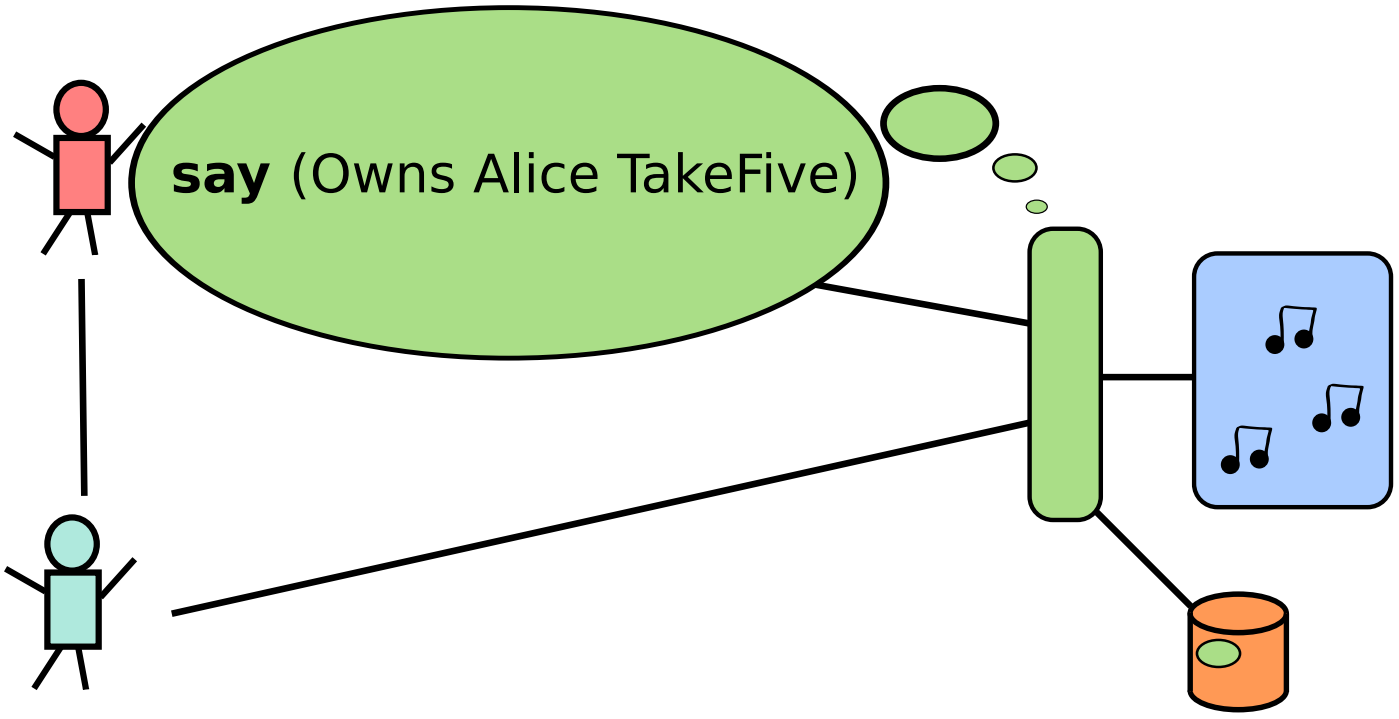
Using the ICFP policy.



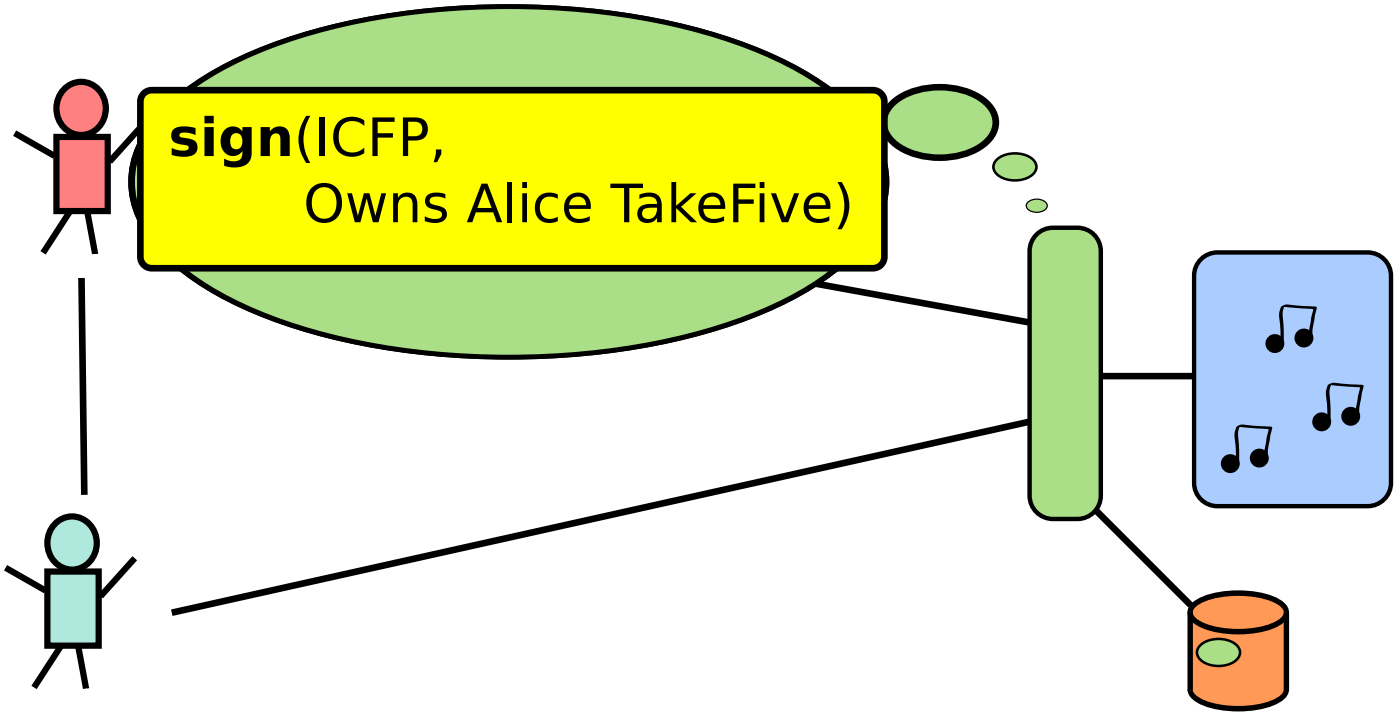
Using the ICFP policy.



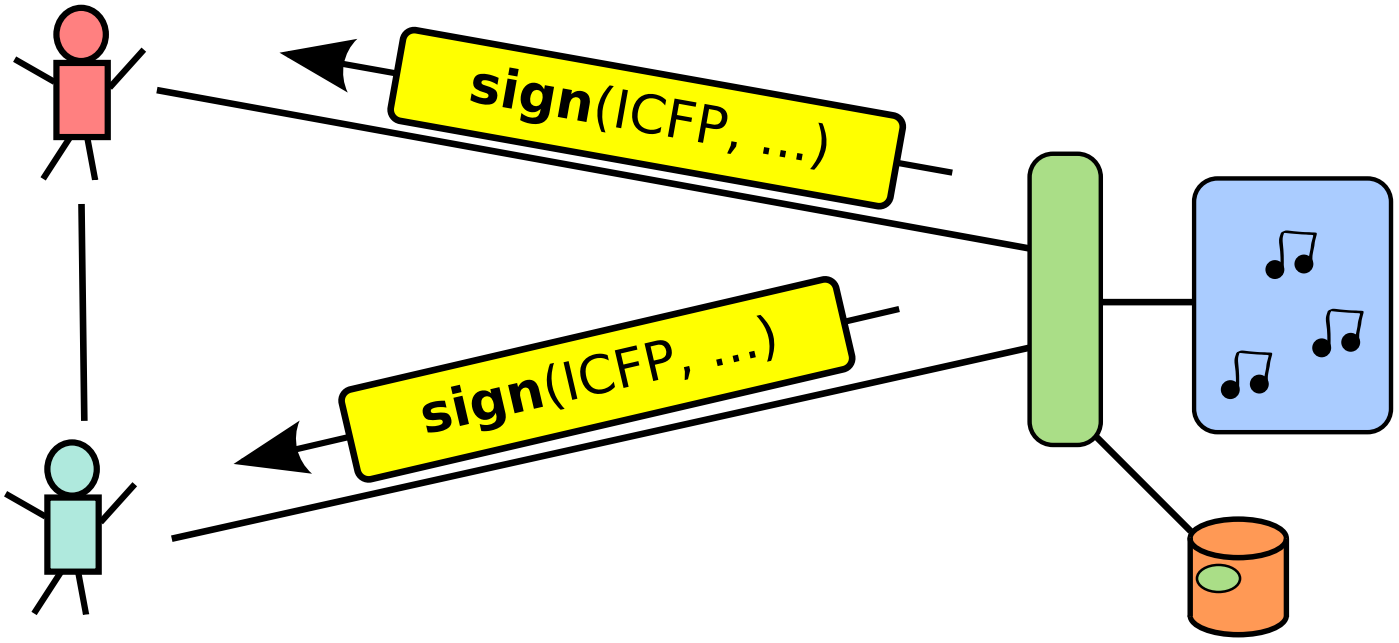
Using the ICFP policy.



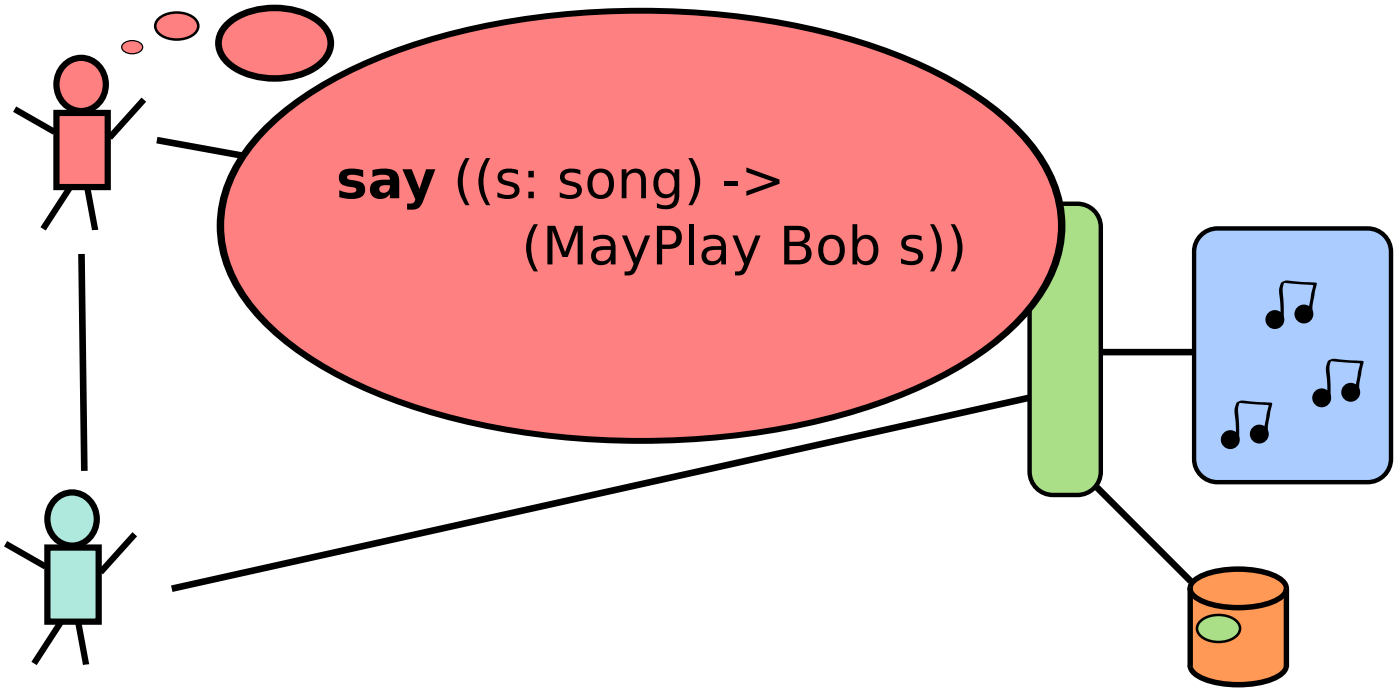
Using the ICFP policy.



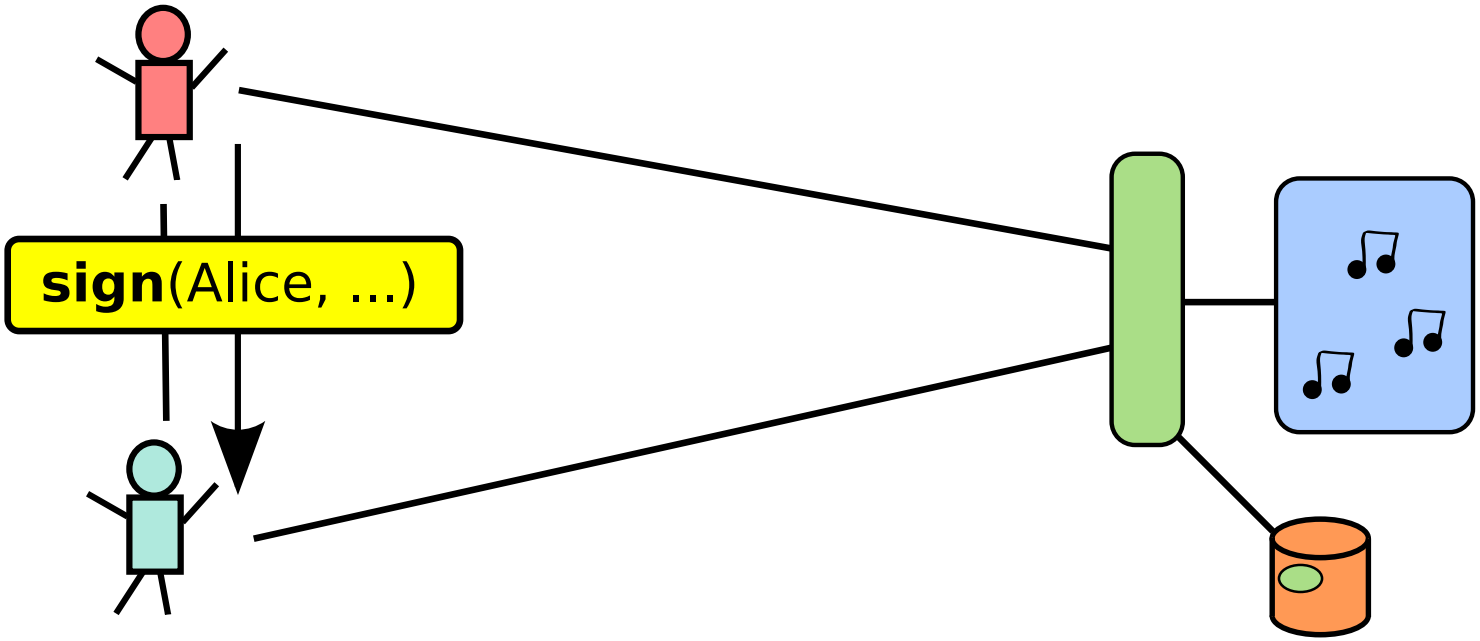
Using the ICFP policy.



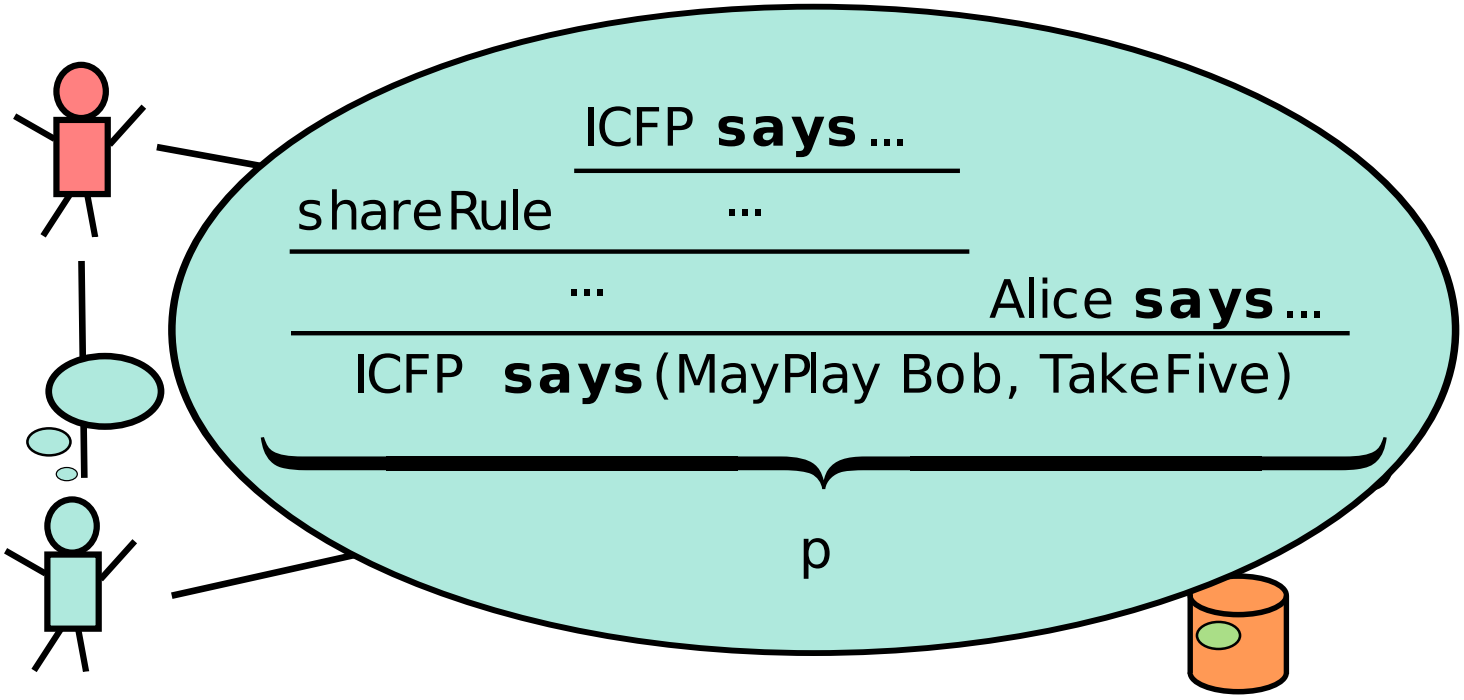
Using the ICFP policy.



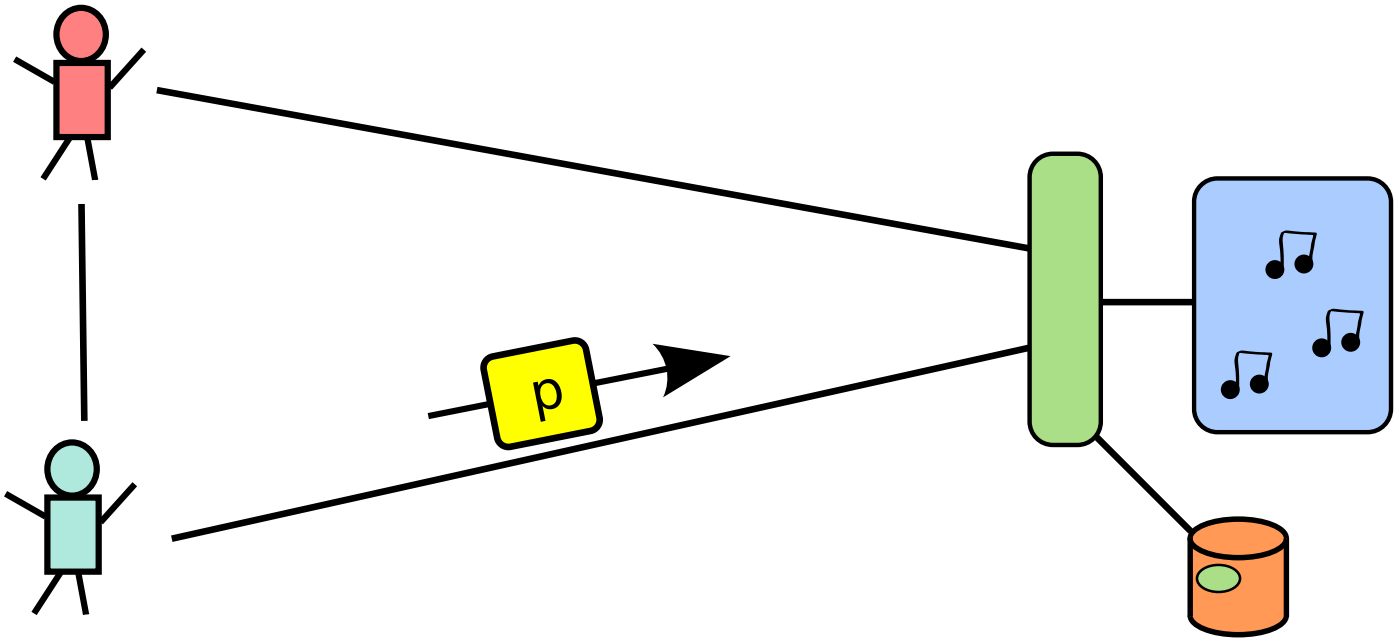
Using the ICFP policy.



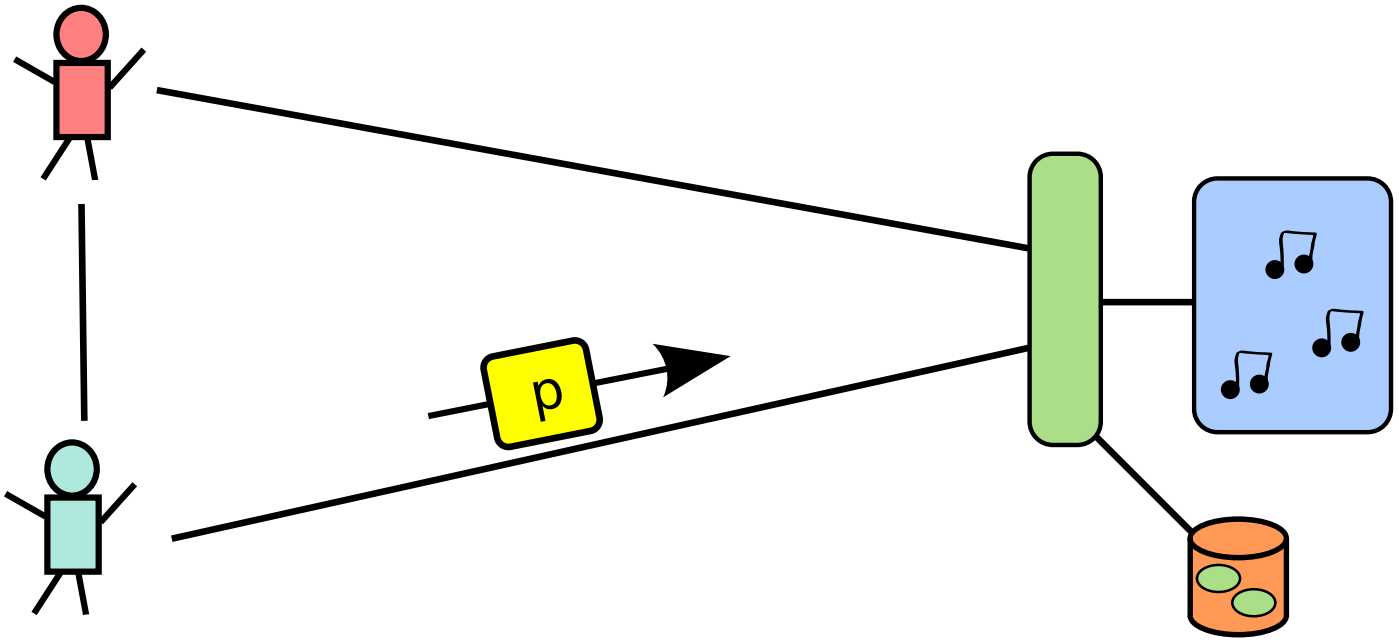
Using the ICFP policy.



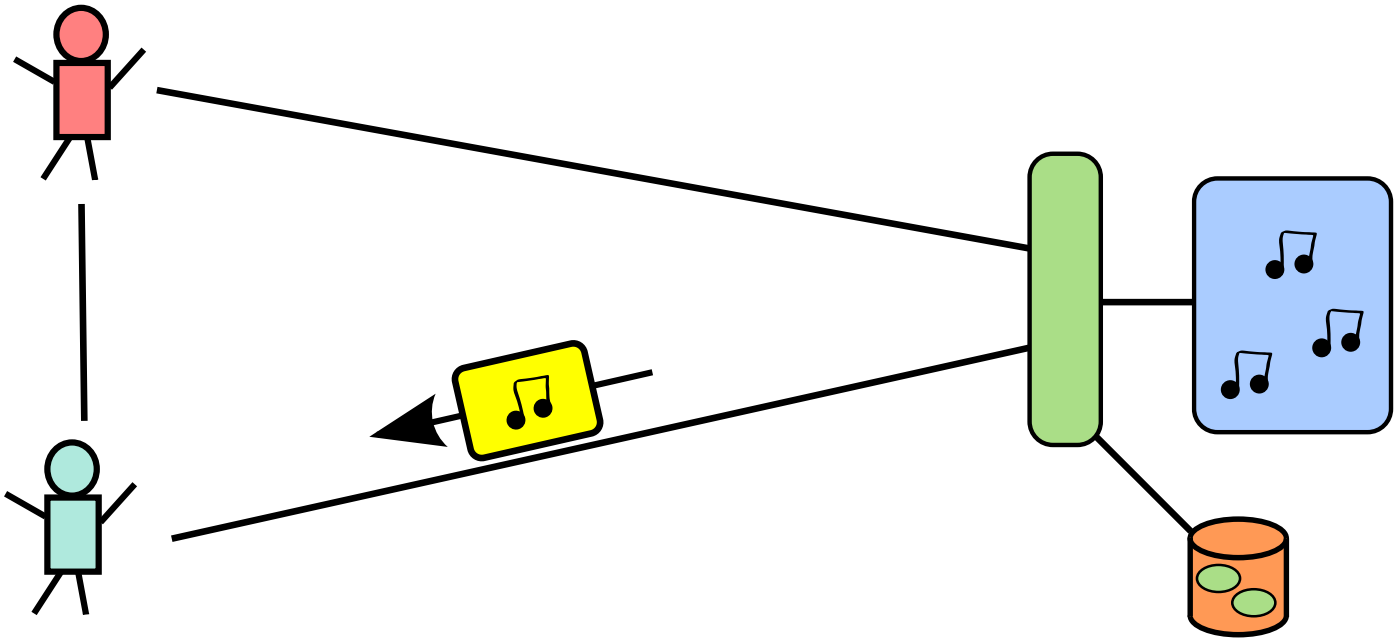
Using the ICFP policy.



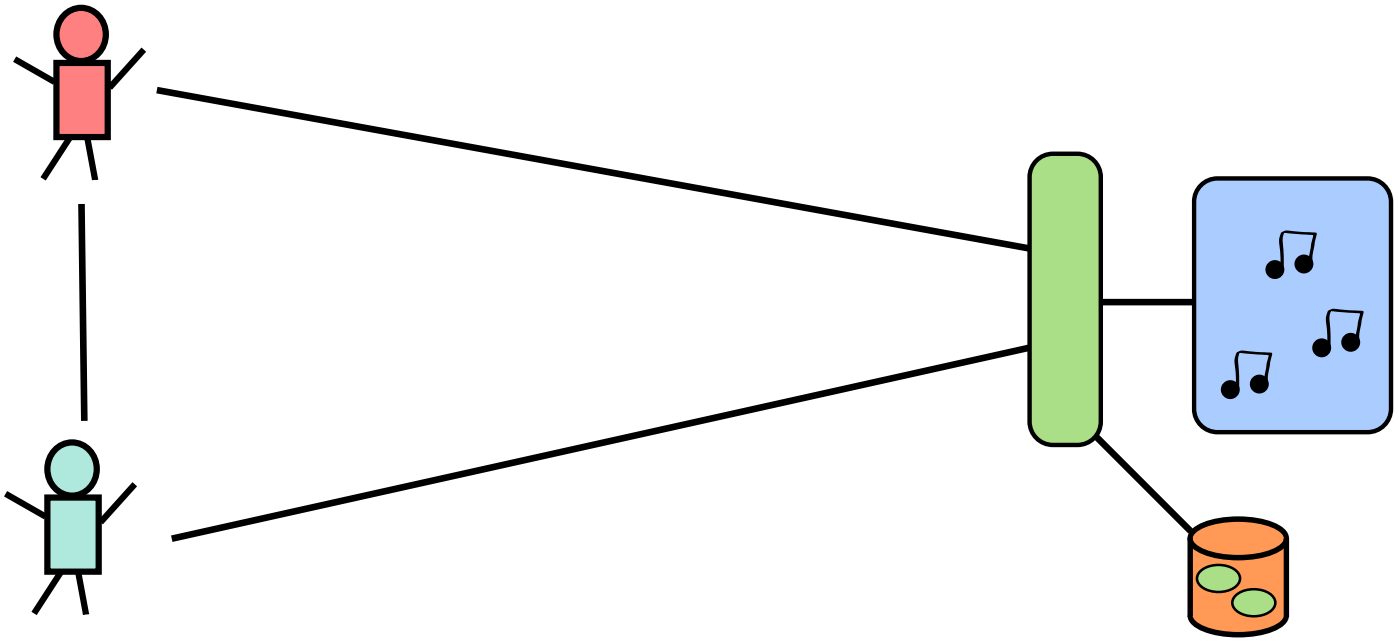
Using the ICFP policy.



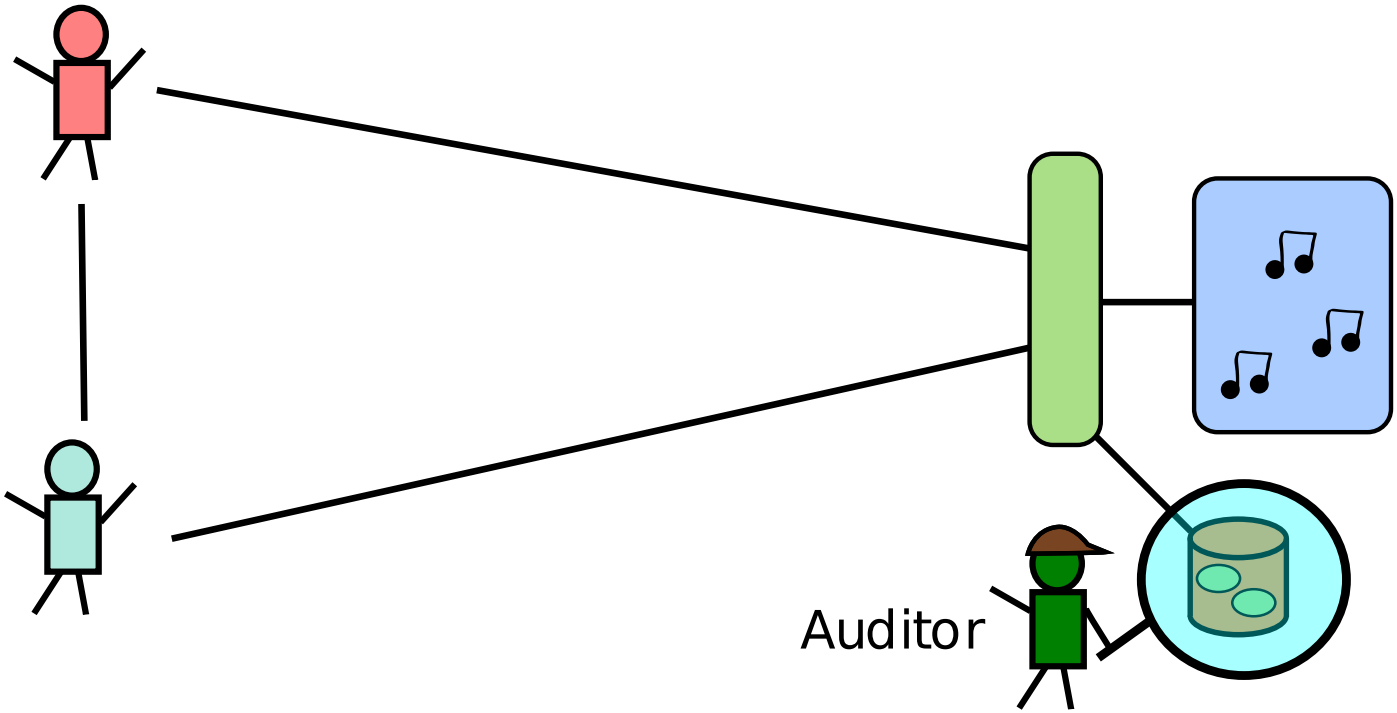
Using the ICFP policy.



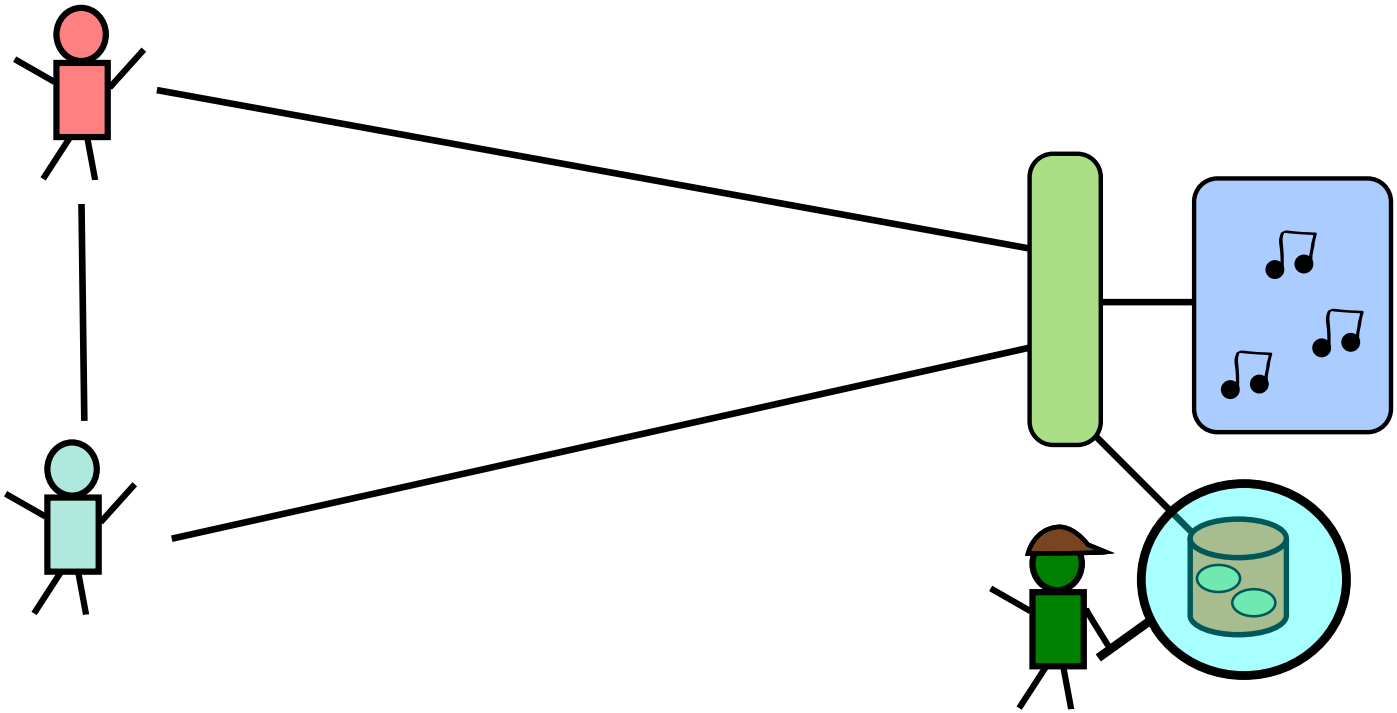
Using the ICFP policy.



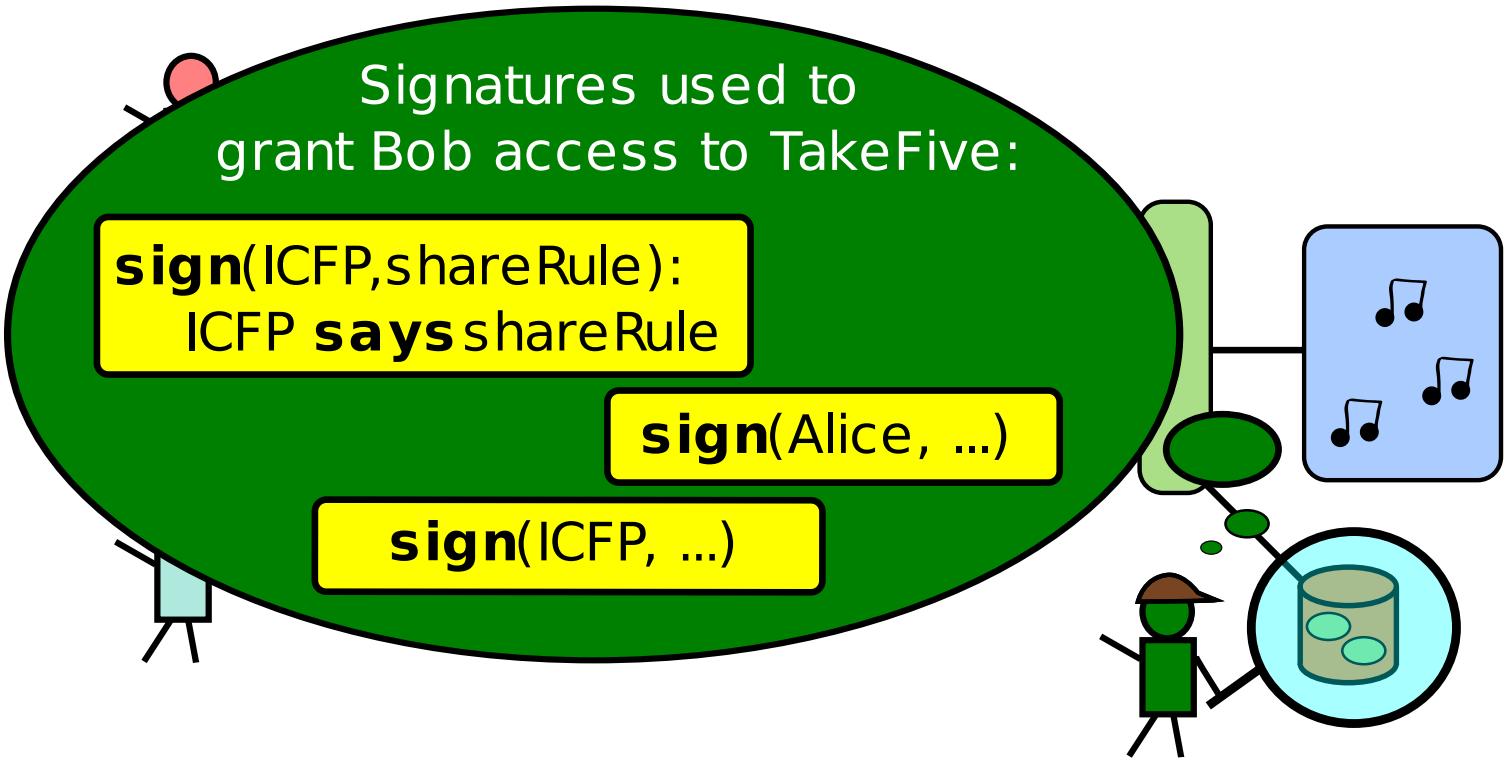
Using the ICFP policy.



Using the ICFP policy.



Using the ICFP policy.

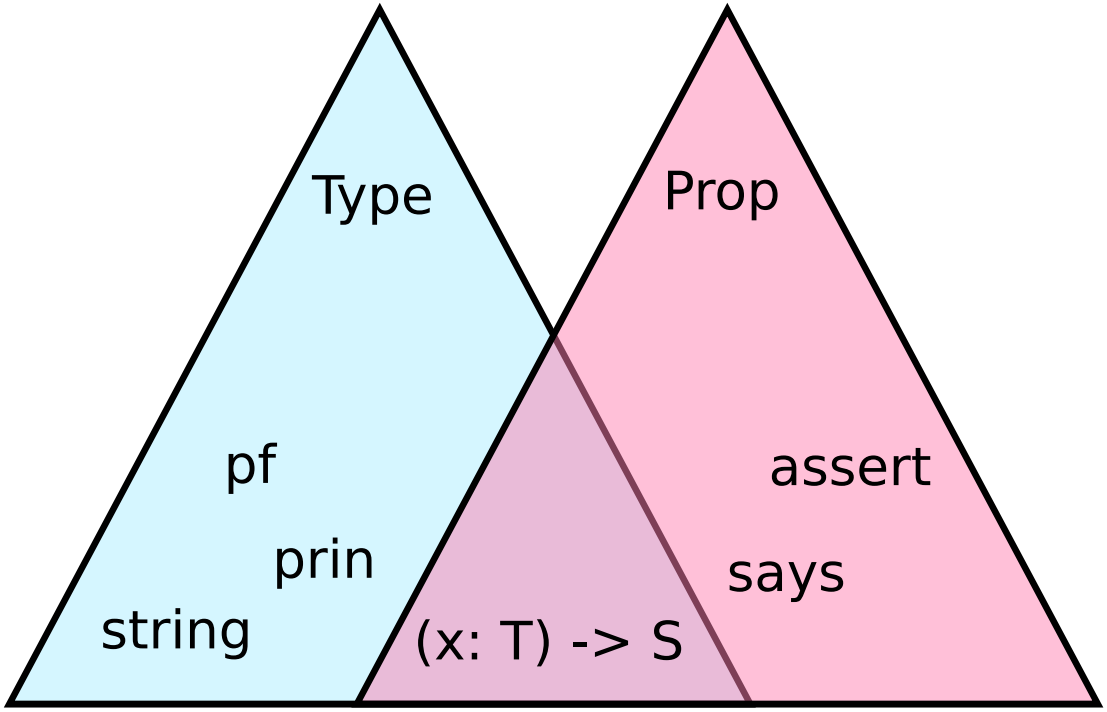


Language Design and Features

AURA's type system is divided into two universes.

Type Contains computation expressions. Includes non-termination and world effects.

Prop Contains pure expressions with a clear interpretation as proofs.



Aura's says modality represents affirmation.

- The proposition "principal Alice affirms proposition P."

  Alice **says** P: **Prop**

- Principals may actively affirm propositions with signatures.

  sign(Alice, P): Alice **says** P

- Principals affirm "true" propositions

  return Alice p: Alice **says** P

  when p: P.

DCC [Abadi+ 06], Logic with Explicit Time [DeYoung+ 08]

Dependent types allow for expressive rules.

Example (Bob acts for Alice)

$$\text{Alice } \textbf{says } ((\text{P: } \textbf{Prop}) \rightarrow \text{Bob } \textbf{says } \text{P} \rightarrow \text{P})$$

Dependent types allow for expressive rules.

Example (Bob acts for Alice)

$$\text{Alice } \textbf{says } ((P\colon \textbf{Prop}) \to \text{Bob } \textbf{says } P \to P)$$

Example (Bob acts for Alice only regarding jazz)

$$\begin{aligned}\text{Alice } \textbf{says } ((s\colon \text{Song}) \to \text{isJazz } s \to \\ \text{Bob } \textbf{says } (\text{MayPlay Bob } s) \to \text{MayPlay Bob } s)\end{aligned}$$

Dependent types allow for expressive rules.

Example (Bob acts for Alice)

$$\text{Alice } \textbf{says } ((\text{P}: \textbf{Prop}) \rightarrow \text{Bob } \textbf{says } \text{P} \rightarrow \text{P})$$

Example (Bob acts for Alice only regarding jazz)

$$\text{Alice } \textbf{says } ((\text{s}: \text{Song}) \rightarrow \text{isJazz s} \rightarrow \text{Bob } \textbf{says } (\text{MayPlay Bob s}) \rightarrow \text{MayPlay Bob s})$$

Restricted formulation of dependent types:

- expressive enough for access control
- too weak for general correctness properties
- AURA feels more like ML than Coq

Effect **say** reifies a program's authority as a signature.

- Programs manufacture new **sign** objects with **say**.
- Intuitively **say** uses the program's (e.g. current user's) private key to generate the signature.
- Special principal **self** stands in for the program.

say P: **self says** P

say P $\mapsto$ **sign**(**self**, P)

Effect **say** reifies a program's authority as a signature.

- Programs manufacture new **sign** objects with **say**.
- Intuitively **say** uses the program's (e.g. current user's) private key to generate the signature.
- Special principal **self** stands in for the program.

say P: **pf**(**self says** P)

say P $\mapsto$ **return**(**sign**(**self**, P))

Technical Point

The **pf** monad protects the **Prop** universe from **say**'s world effect.

AURA contains inductive types and *assertions*.

- Inductive Types define complex data structures.

  data List : **Type** → **Type** {
  | nil : (t :**Type**) → (List t)
  | cons : (t :**Type**) → t → (List t) → (List t) }

- Inductive Props define simple inference systems subject to a (draconian) positivity constraint.

  data And: **Prop** → **Prop** → **Prop** {
  | both: (P: **Prop**) → (Q: **Prop**) → P → Q → And P Q }

  data False: **Prop** { }

- Assertions define access control predicates

  assert Owns: **prin** → Song → **Prop**

Assertion types are uninhabited, but not false.

Inductive types admit pattern matching.

Example

$\lambda f\colon$ Alice **says** False. $\lambda P\colon$ **Prop**. . . .
match f **with** (P) $\{\}$ . . .
$\quad :$ Alice **says** False $\rightarrow$ (P: **Prop**) $\rightarrow$ Alice **says** P

Assertions have no elimination form.

- Intuition: Assertions $\approx$ type variables.
- There is no analogous function of type

ICFP **says** (Owns Bob Thriller) $\rightarrow$
(P:**Prop**) $\rightarrow$ ICFP **says** P.

Theory and Practice

AURA's metatheory: the view from 30,000 feet.

- AURA is defined in a Pure-Type-Systems style.

$$
\begin{array}{rcl}
t & ::= & \textbf{Prop} \mid \textbf{Type}\ldots \\
 & \mid & (\mathrm{x}\colon \mathrm{t}) \rightarrow \mathrm{t} \mid \mathrm{t}\ \textbf{says}\ \mathrm{t}\ldots \\
 & \mid & \lambda \mathrm{x}\colon \mathrm{t}.\ \mathrm{t} \mid \textbf{sign}(\mathrm{t},\ \mathrm{t})\ldots
\end{array}
$$

- Call-by-value reduction ensures $\bot$ isn't confused for a proof.

Theorem (Syntactic Soundness)

Reduction preserves typing; well-typed terms don't get stuck.

Theorem (Decidability of typechecking)

Either $\Sigma; \Gamma \vdash t_1 : t_2$ *or* $\Sigma; \Gamma \nvdash t_1 : t_2$*, constructively.*

Aura's core metatheory formalized in Coq.

- Terms *locally nameless*, with DeBruijn indexed bound variables and named free variables.
- Formalized features: inductive data types, **Prop** and **Type** language fragments, **says** and **pf** modalities. . . .

Development Size (in lines of commented Coq code)	
Definitions	1400
Type Soundness	6000
Decidability of Typechecking	5000

Engineering Formal Metatheory [Aydemir+ 08]

Aura is real.

- Current Features:
  - Interpreter and typechecker for full language
  - Foreign function interface
- Coming Soon:
  - Cryptographic implementation of **sign**
  - Automatic logging
- Future Research:
  - Type inference?
  - Surface syntax?
  - Information flow?
  - Effects tracking?

Aura is real.

- Current Features:
  - Interpreter and typechecker for full language
  - Foreign function interface
- Coming Soon:
  - Cryptographic implementation of **sign**
  - Automatic logging
- Future Research:
  - Type inference?
  - Surface syntax?
  - Information flow?
  - Effects tracking?

Demo

Conclusion

The AURA language ...

- unifies access control and computation.
- supports arbitrary domain-specific authorization policies.
- mixes weak dependency, effects, and authorization logic in a compelling way.

Conclusion

The AURA language ...

- unifies access control and computation.
- supports arbitrary domain-specific authorization policies.
- mixes weak dependency, effects, and authorization logic in a compelling way.

Interpreter, Coq scripts, and papers available from
`http://www.cis.upenn.edu/~stevez/sol/aura.html`

Bonus Slides

- Access Control Matrices and Capabilities
- Mechanizing AURAwas a positive experience.

Conventional techniques handle the ICFP policy poorly.

Access control matrices

- ICFP server stores the list of owners and delegations.
- Owner must contact ICFP server directly to delegate.
- All participants must trust server's records re: delegation.

Atomic capabilities

- Unforgeable, atomic tokens serve as tickets to play songs.
- Who issues the tokens?

Mechanizing AURA was a positive experience.

- Aura is large.
  - 21 syntactic forms
  - 15 judgments
  - 63 inference rules
- Mechanization helped us manage AURA's complexity. Coq proofs...
  - provided assurance that we hadn't make mistakes.
  - enabled us to experiment without rechecking pages of unaffected proofs.
  - simplified collaboration (source control, etc.).

PENN PL