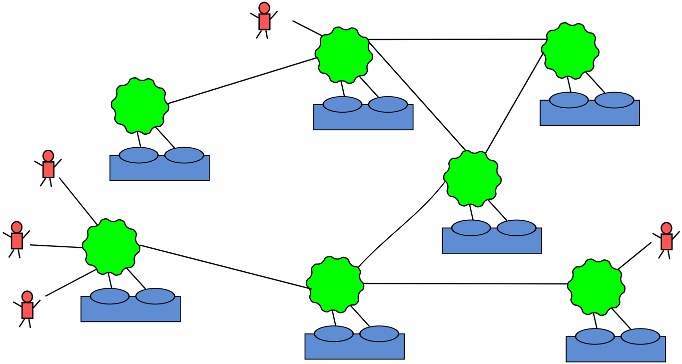
Evidence-based Audit

Jeff Vaughan
Limin Jia, Karl Mazurak, and Steve Zdancewic

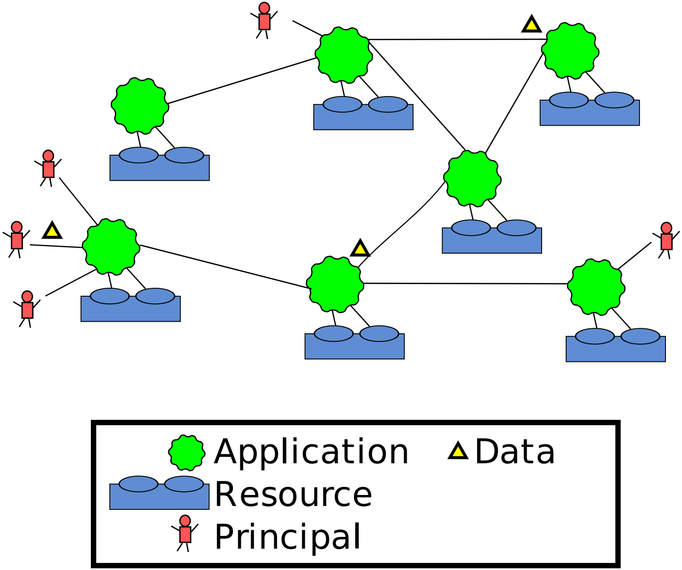
Department of Computer and Information Science
University of Pennsylvania

CSF/LICS Joint Session
June 24, 2008

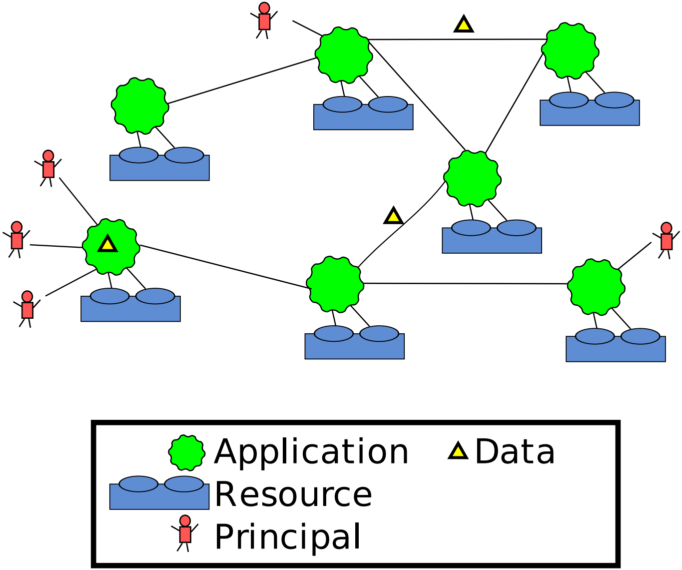
Our Setting: Distributed Access Control



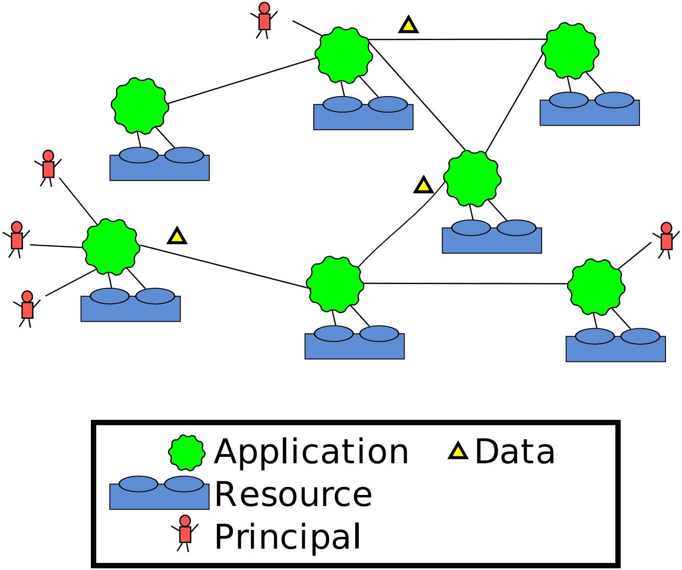
Our Setting: Distributed Access Control



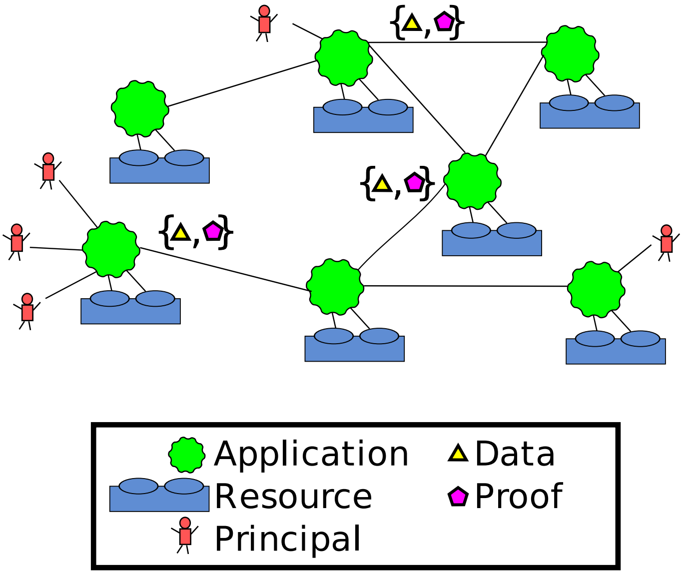
Our Setting: Distributed Access Control



Our Setting: Distributed Access Control



Key Idea: Proofs attest to data integrity.



[Necula+ 98]

The Aura Project

- A programming language called Aura
  - A propositional fragment, modeled here by Aura_0
  - An ML-like computation language [Jia+ 08]
- A security aware programming model
  - active, potentially malicious principals
  - mutual distrust between applications and principals
  - emphasis on access control and audit
- An implementation including complier and .Net-based runtime

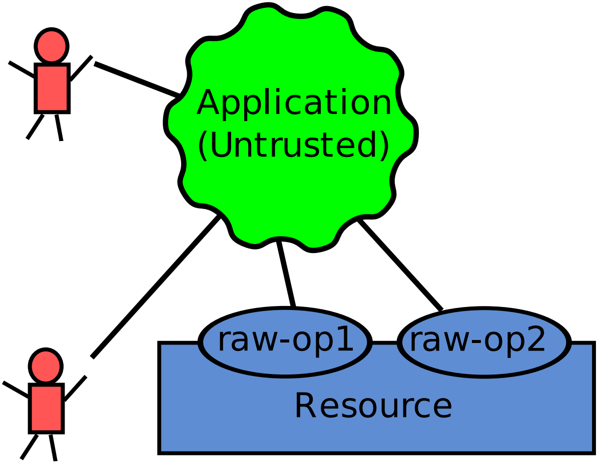
The Aura Project

- A programming language called Aura
  - A propositional fragment, modeled here by Aura$_0$
  - An ML-like computation language [Jia+ 08]
- A security aware programming model
  - active, potentially malicious principals
  - mutual distrust between applications and principals
  - emphasis on access control and audit
- An implementation including complier and .Net-based runtime

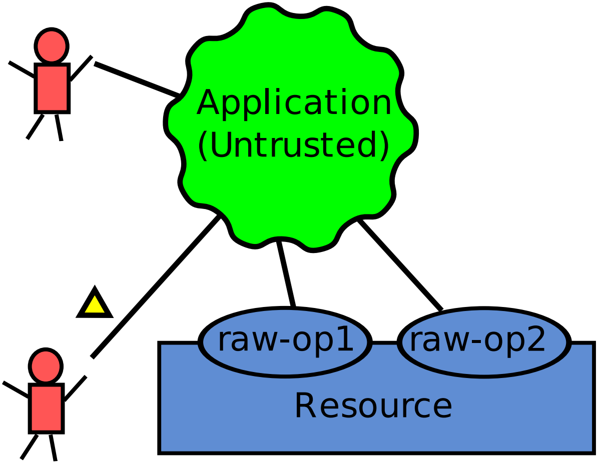
Today's Talk

Analyzing the local security of Aura applications.

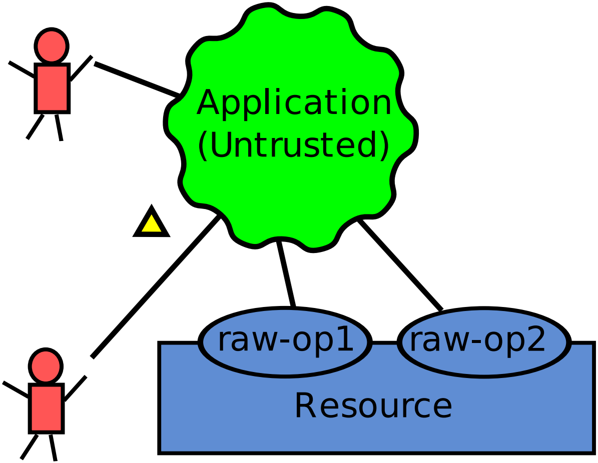
Access control, locally.



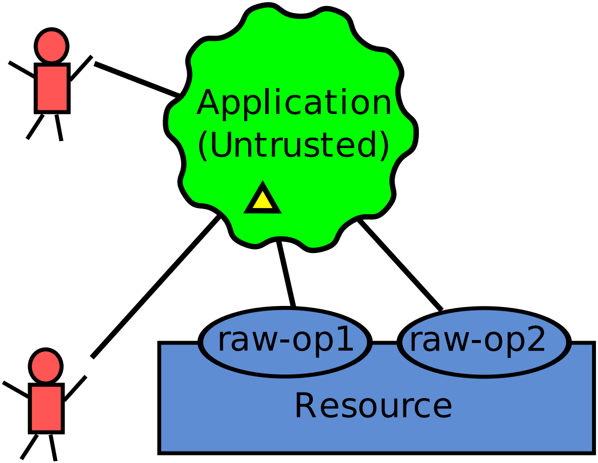
Access control, locally.



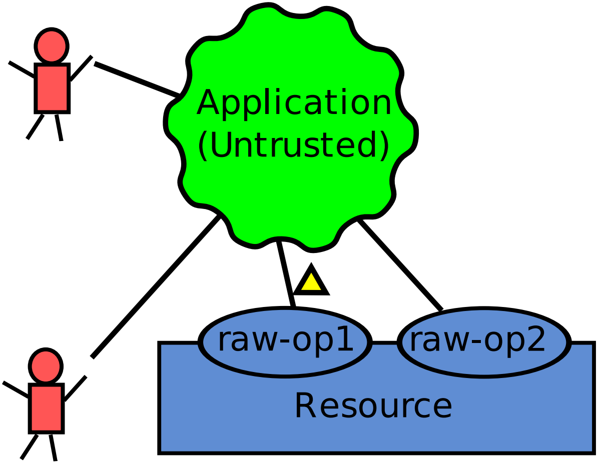
Access control, locally.



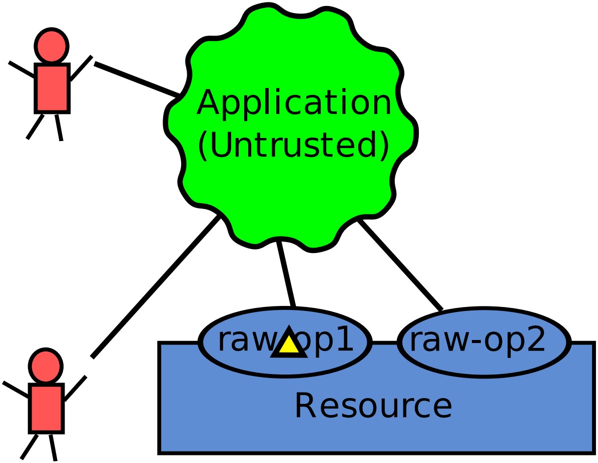
Access control, locally.



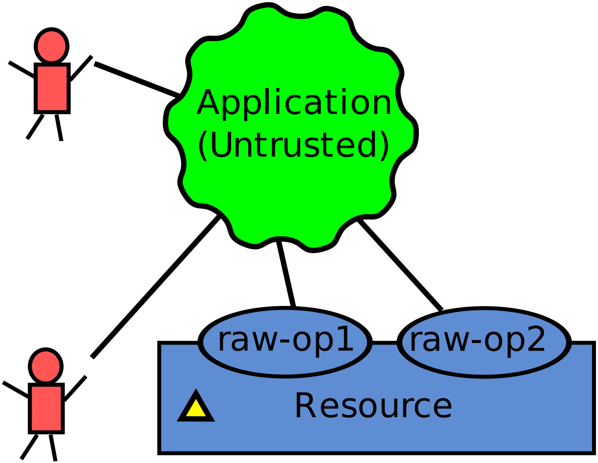
Access control, locally.



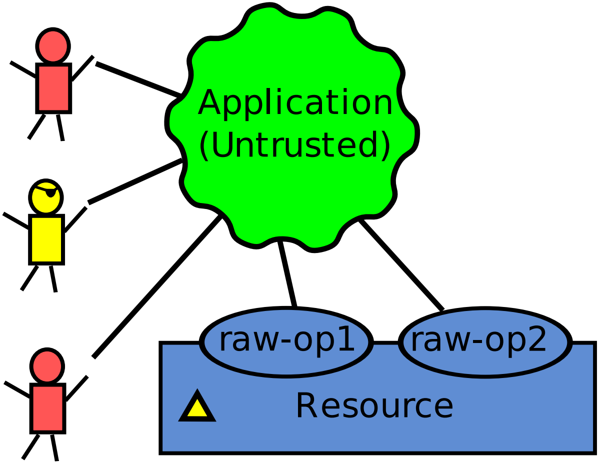
Access control, locally.



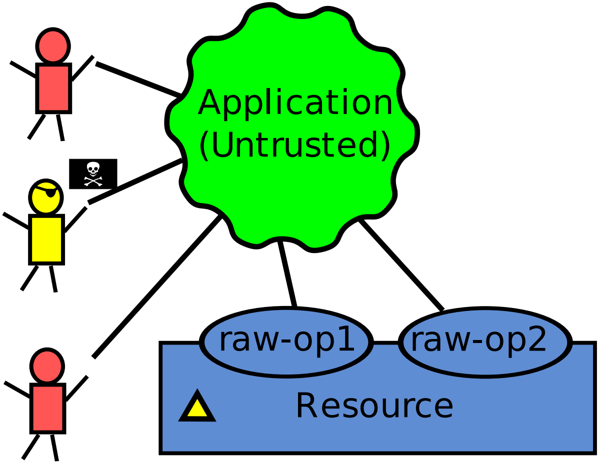
Access control, locally.



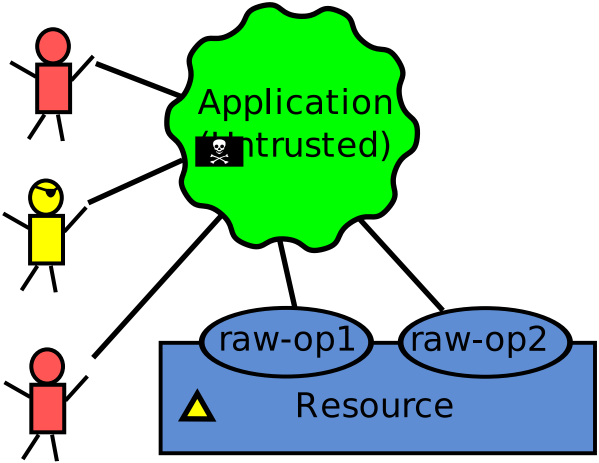
Access control, locally.



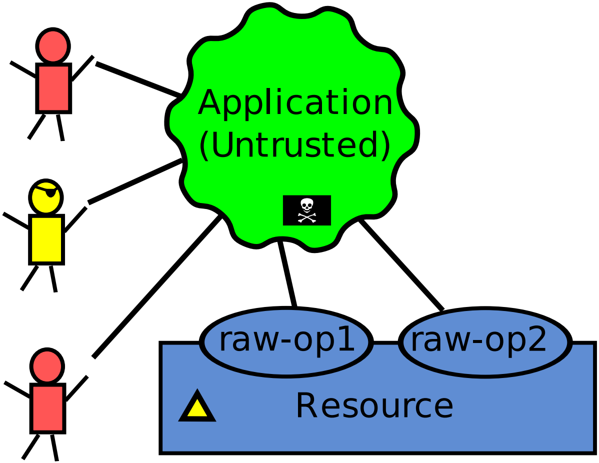
Access control, locally.



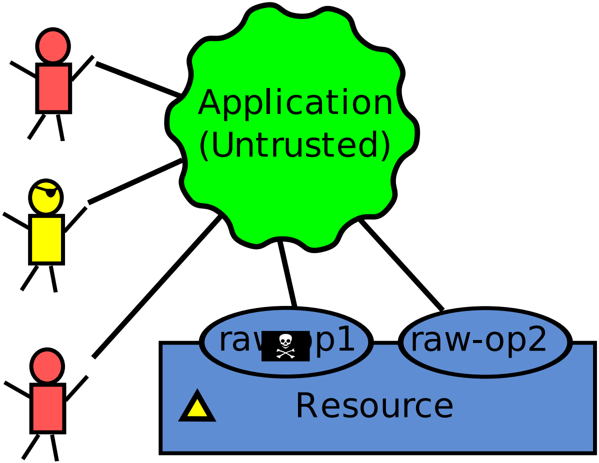
Access control, locally.



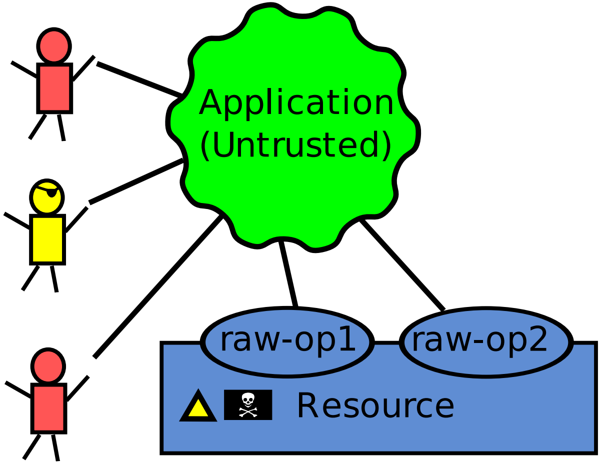
Access control, locally.



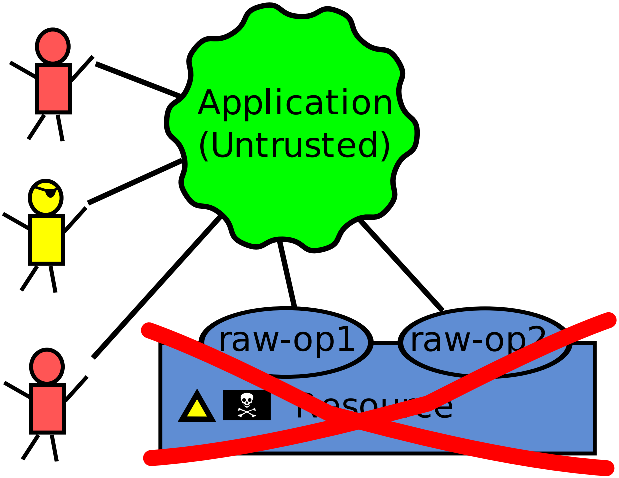
Access control, locally.



Access control, locally.



Access control, locally.



Jargon needed for this talk

Institutional Policy Human-level rules about principals, data values, and resources.

Bad Access A system state change violating institutional policy.

Formal Rules Machine-level encoding of policy.

Jargon needed for this talk

Institutional Policy: Human-level rules about principals, data values, and resources.

Bad Access: A system state change violating institutional policy.

Formal Rules: Machine-level encoding of policy.

Foreshadowing

Users care about *institutional policy*, but technology tries to enforces *formal rules*.

Why do reference monitors allow bad accesses to occur?

Problem 1

The trusted computing base's implementation may be buggy.

Problem 2

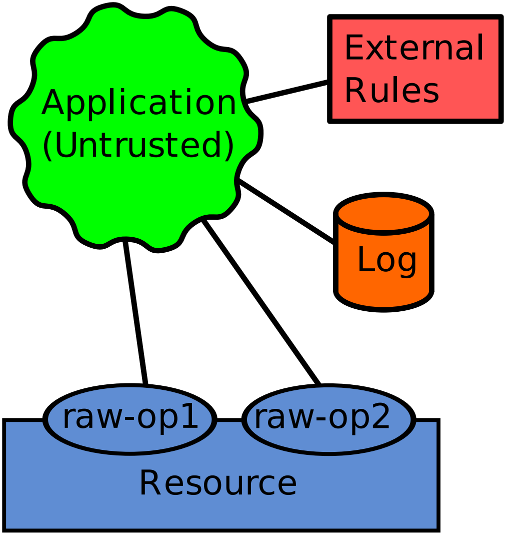
The formal rules language may be too impoverished to express institutional policy.

Problem 3

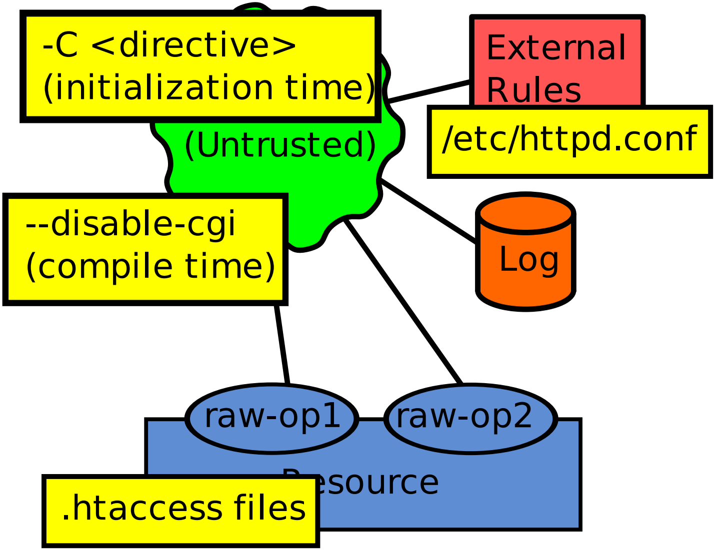
The system may be configured with incorrect formal rules.

And many other reasons not addressed here...

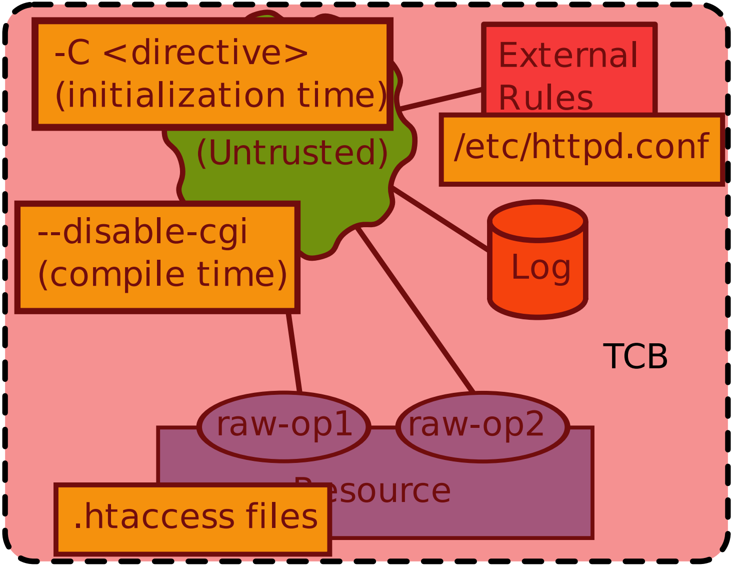
Common application design exacerbates these problems.



Common application design exacerbates these problems.



Common application design exacerbates these problems.



Problem 1

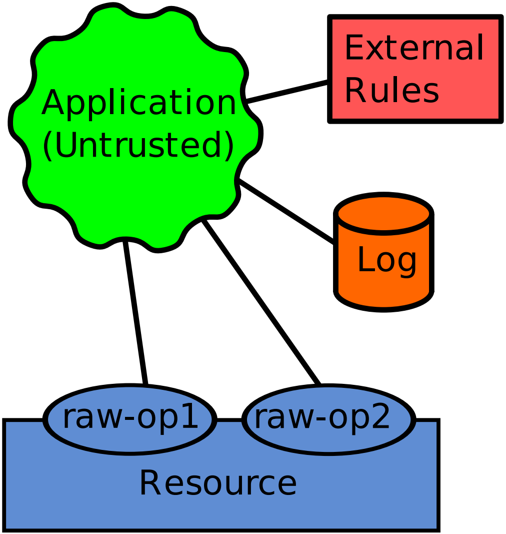
The trusted computing base's implementation may be buggy.

Aura Solution

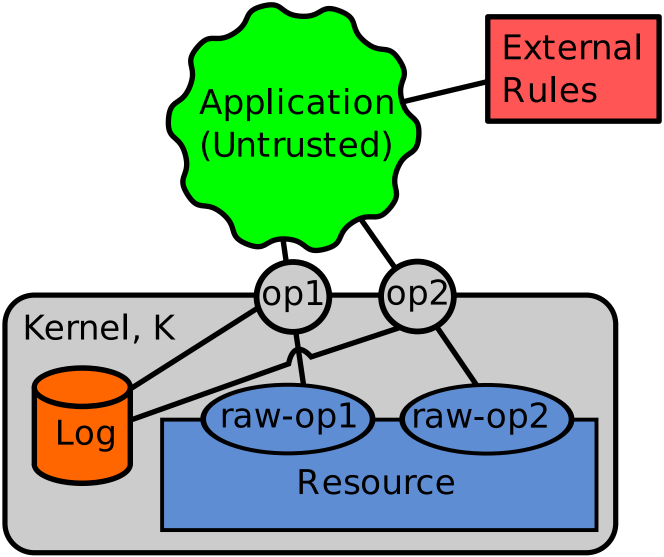
Trust only a small, generic *kernel* that has no application-specific functionality.

[Saltzer+ 75], [Bauer+ 99], [Jia+ 08]

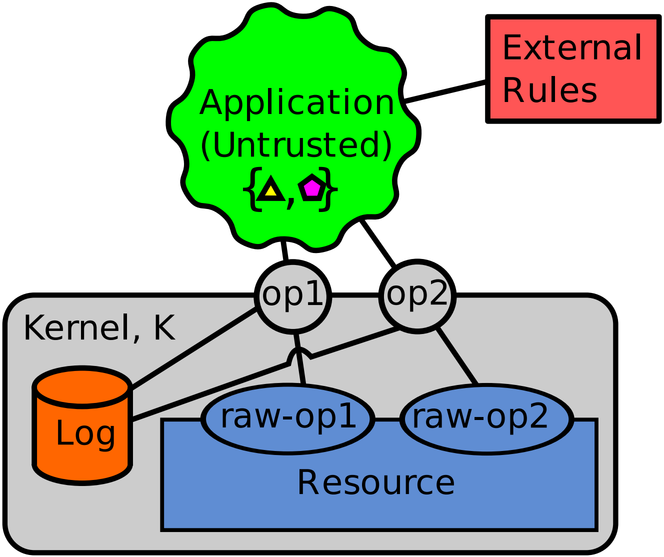
In Aura, a lightweight kernel protects resources.



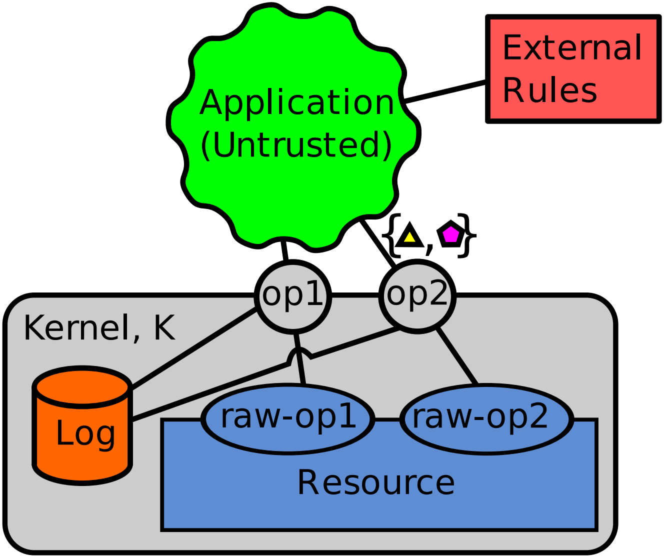
In Aura, a lightweight kernel protects resources.



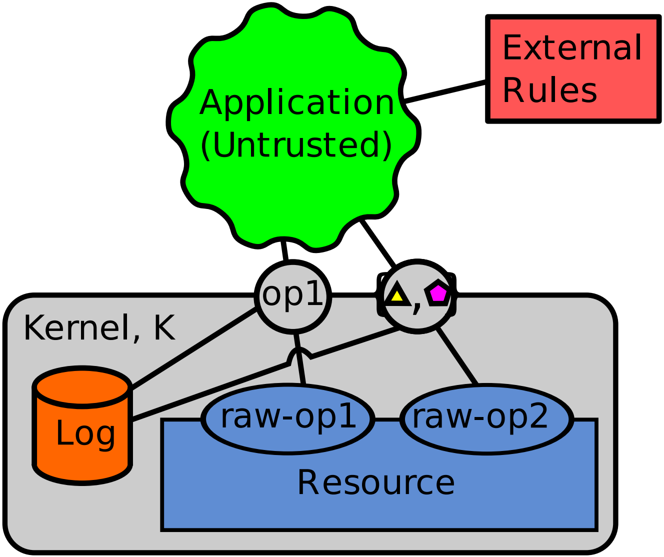
In Aura, a lightweight kernel protects resources.



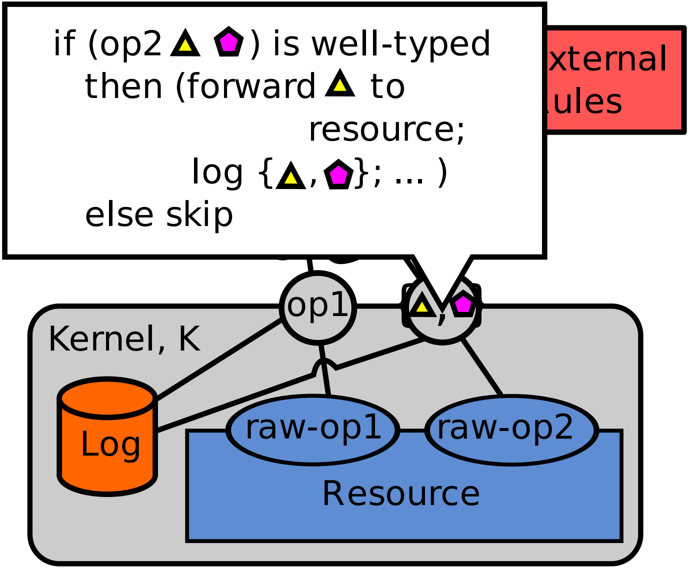
In Aura, a lightweight kernel protects resources.



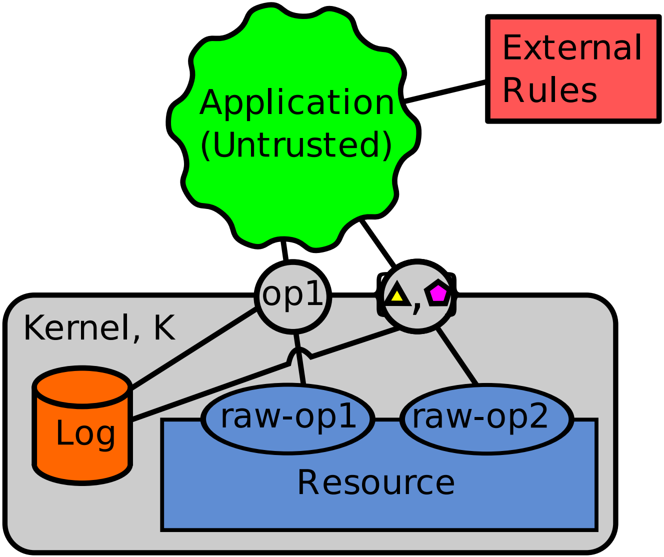
In Aura, a lightweight kernel protects resources.



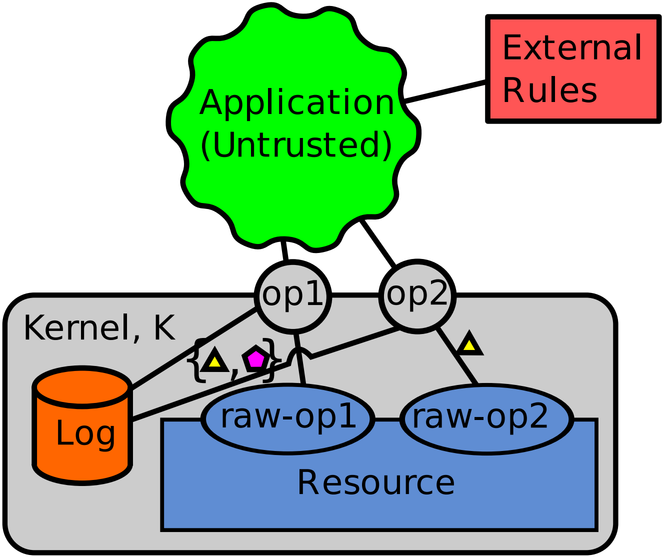
In Aura, a lightweight kernel protects resources.



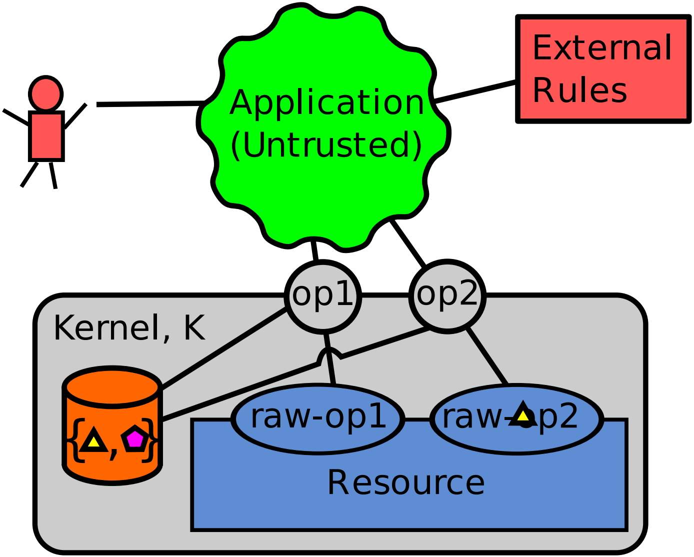
In Aura, a lightweight kernel protects resources.



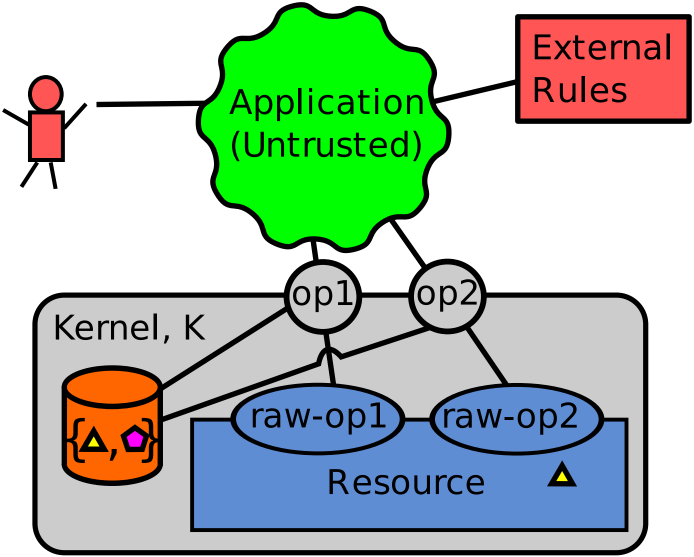
In Aura, a lightweight kernel protects resources.



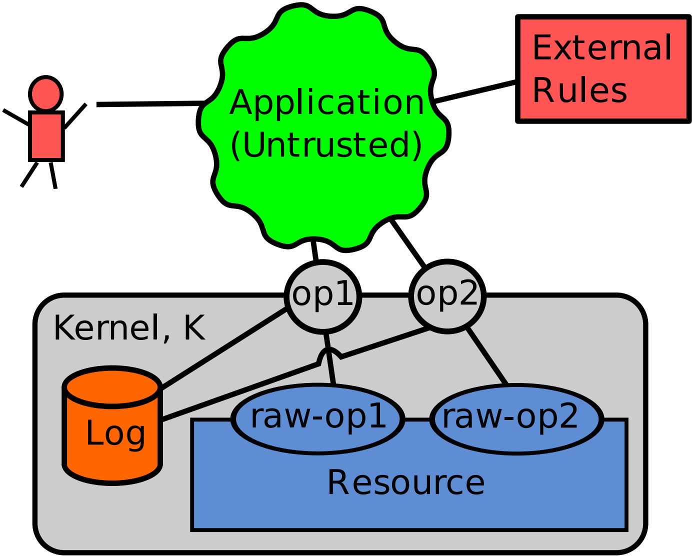
In Aura, a lightweight kernel protects resources.



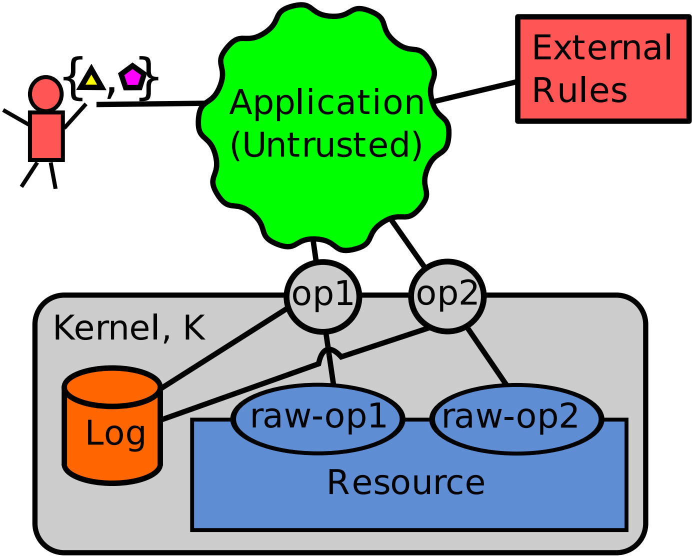
In Aura, a lightweight kernel protects resources.



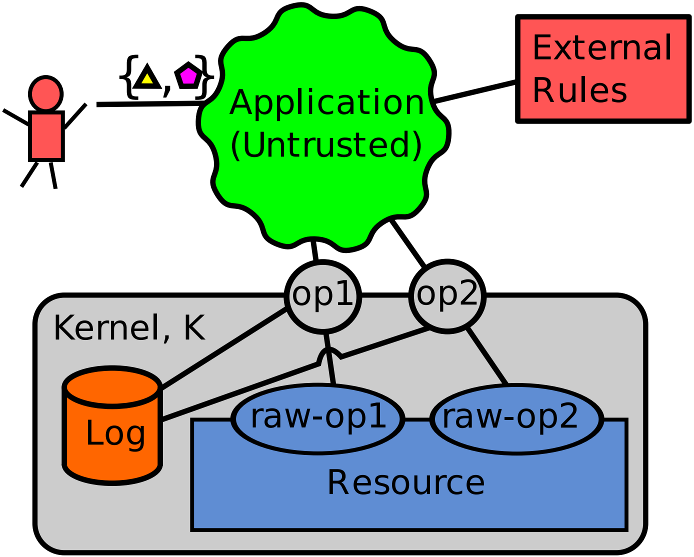
In Aura, a lightweight kernel protects resources.



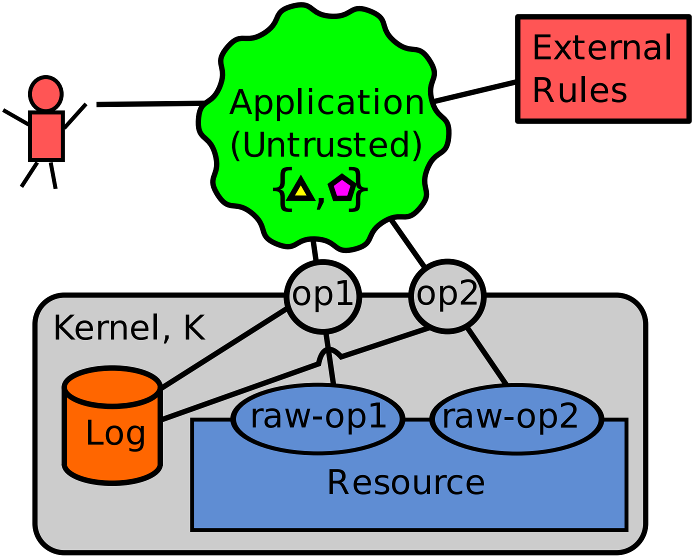
In Aura, a lightweight kernel protects resources.



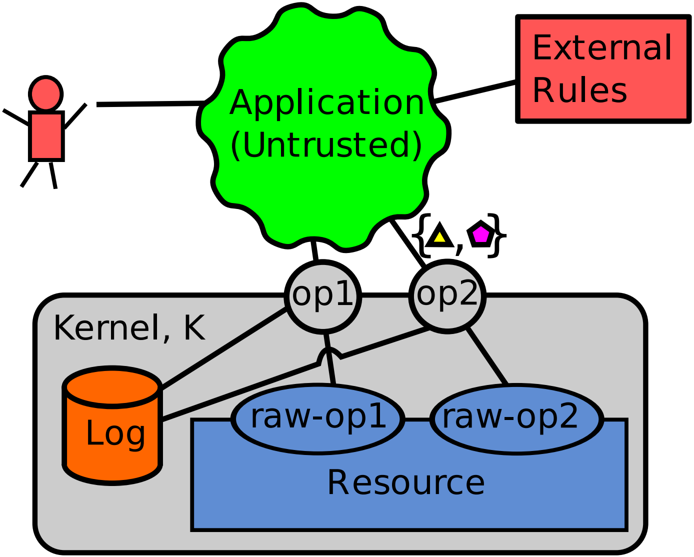
In Aura, a lightweight kernel protects resources.



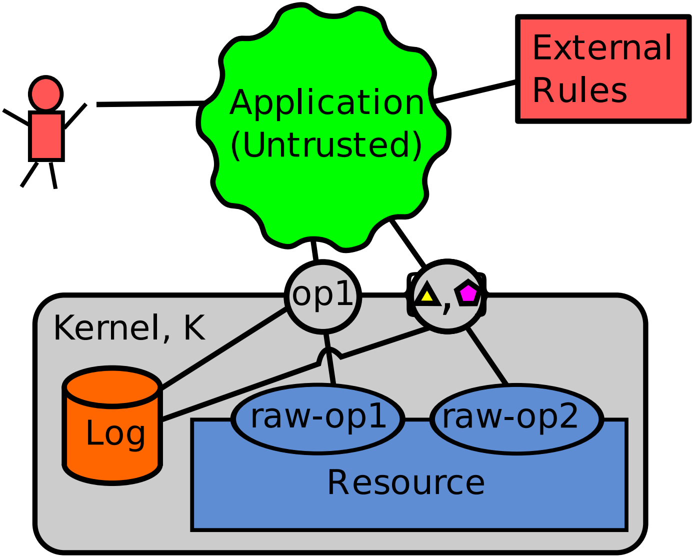
In Aura, a lightweight kernel protects resources.



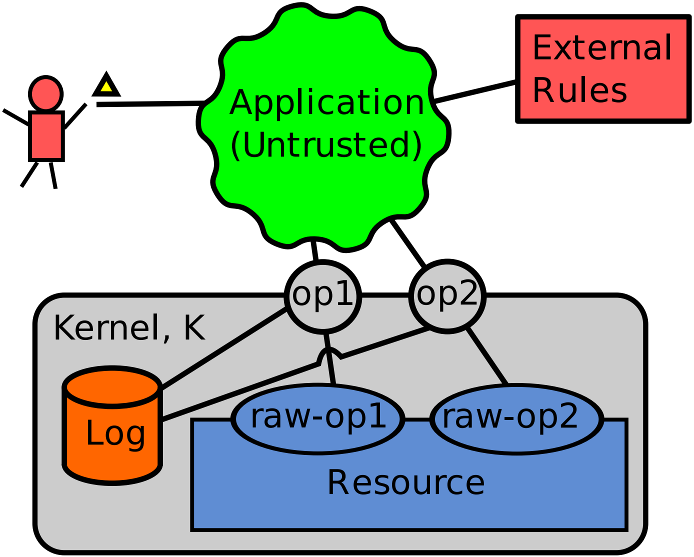
In Aura, a lightweight kernel protects resources.



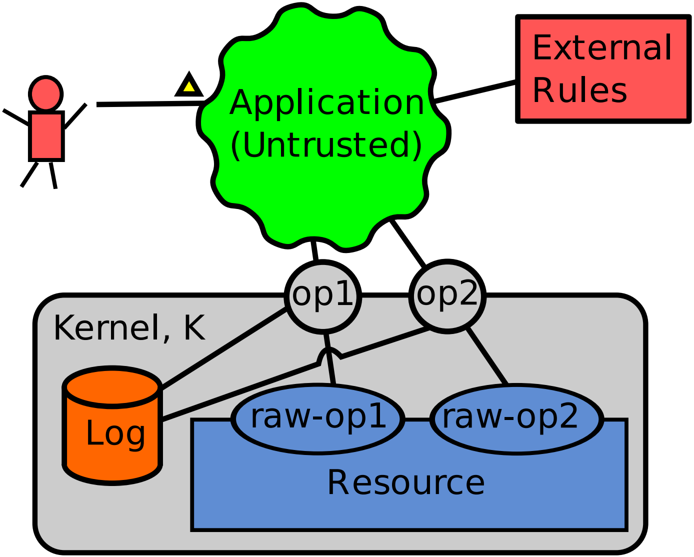
In Aura, a lightweight kernel protects resources.



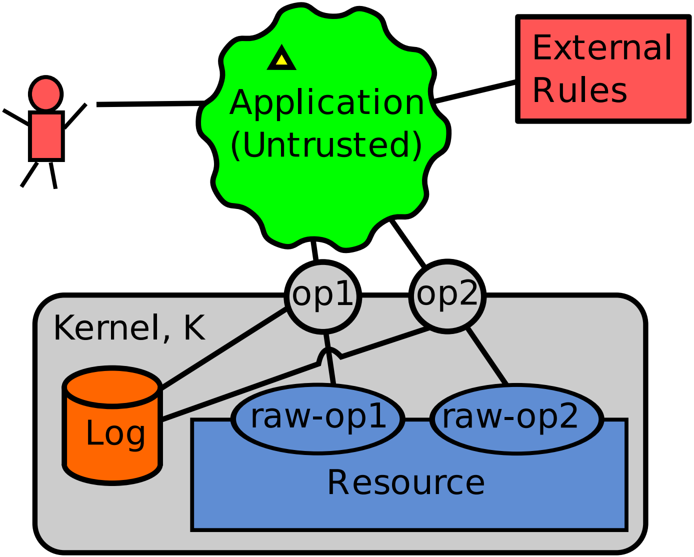
In Aura, a lightweight kernel protects resources.



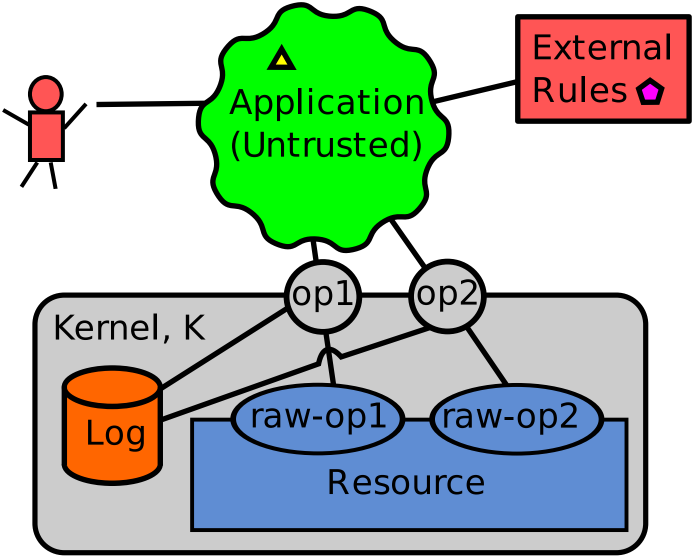
In Aura, a lightweight kernel protects resources.



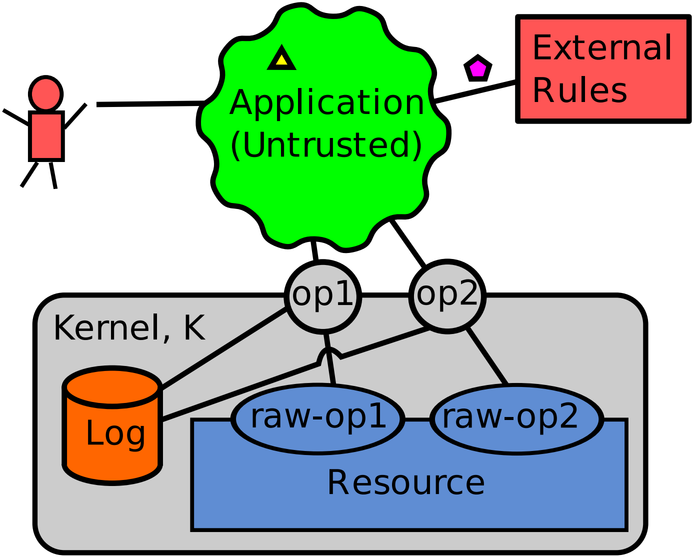
In Aura, a lightweight kernel protects resources.



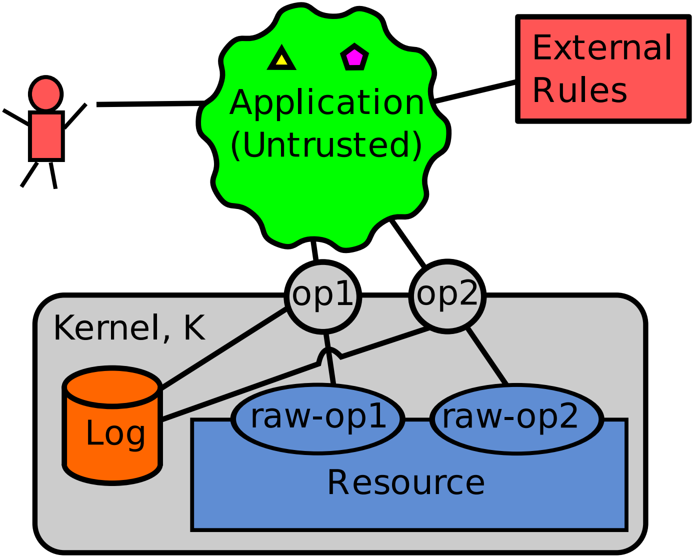
In Aura, a lightweight kernel protects resources.



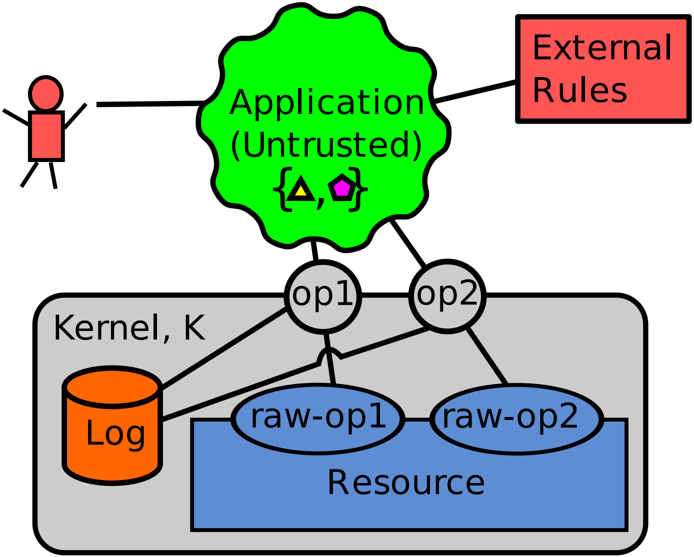
In Aura, a lightweight kernel protects resources.



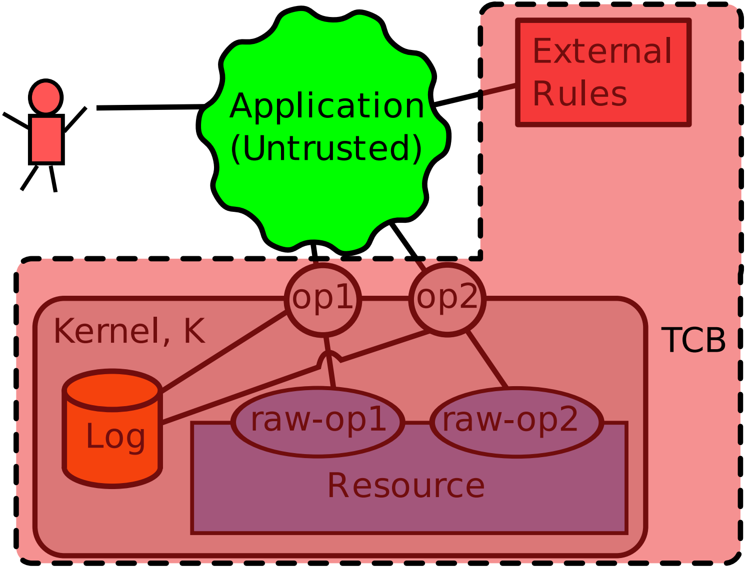
In Aura, a lightweight kernel protects resources.



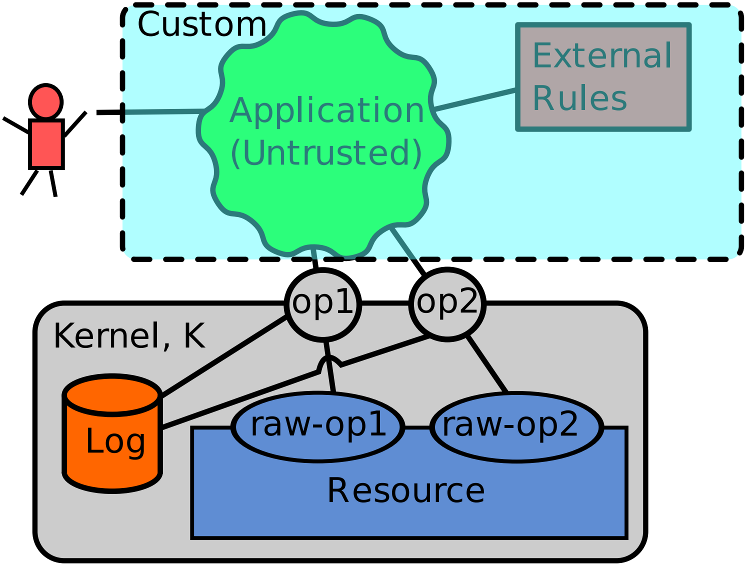
In Aura, a lightweight kernel protects resources.



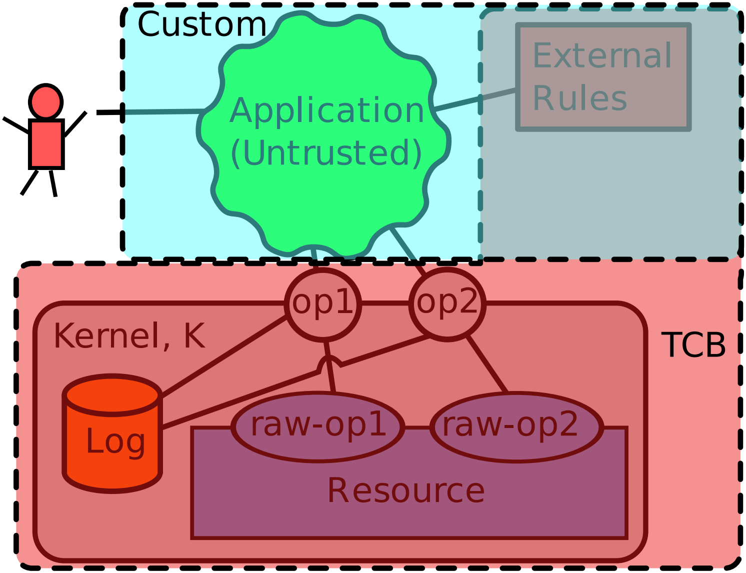
In Aura, a lightweight kernel protects resources.



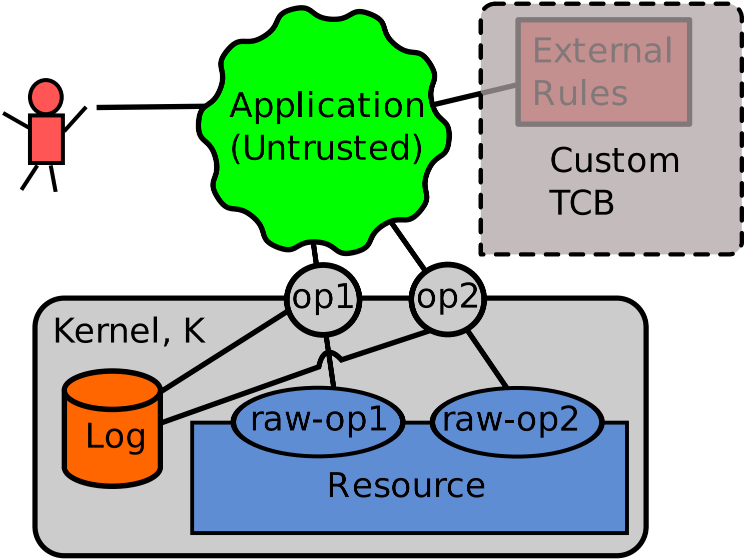
In Aura, a lightweight kernel protects resources.



In Aura, a lightweight kernel protects resources.



In Aura, a lightweight kernel protects resources.



Example: Types for a remote procedure call server.

The RPC server resource has a single raw operation

$$\text{raw-rpc} : \textbf{string} \Rightarrow \textbf{string}$$

The kernel exposes an *extended signature* of propositions.

$$\begin{aligned} \text{OkToRPC} &: \textbf{string} \rightarrow \textbf{Prop} \\ \text{DidRPC} &: \textbf{string} \rightarrow \textbf{string} \rightarrow \textbf{Prop} \end{aligned}$$

Applications are written against the *kernel interface*.

$$\begin{aligned} \text{rpc} \;:\; & (x : \textbf{string}) \Rightarrow (\text{Kernel } \textbf{says } \text{OkToRPC } x) \\ & \Rightarrow \{y : \textbf{string}; \text{Kernel } \textbf{says } \text{DidRPC } x\ y\} \end{aligned}$$

Example: Types for a remote procedure call server.

The RPC server resource has a single raw operation

$$\text{raw-rpc} : \textbf{string} \Rightarrow \textbf{string}$$

The kernel exposes an *extended signature* of propositions.

$$\begin{aligned} \text{OkToRPC} &: \textbf{string} \rightarrow \textbf{Prop} \\ \text{DidRPC} &: \textbf{string} \rightarrow \textbf{string} \rightarrow \textbf{Prop} \end{aligned}$$

Applications are written against the *kernel interface*.

$$\begin{aligned} \text{rpc} \; : \; & (x : \textbf{string}) \Rightarrow (\text{Kernel} \; \textbf{says} \; \text{OkToRPC}\, x) \\ & \Rightarrow \{y : \textbf{string}; \text{Kernel} \; \textbf{says} \; \text{DidRPC}\, x\, y\} \end{aligned}$$

The kernel interface and extended signature
may be mechanically generated.

Problem 2

The formal rules language may be too impoverished to express institutional policy.

Aura Solution

Use dependent DCC with signature objects to specify rules.

[Abadi+ 06], [Fournet+ 07], [Bengtson+ 08] ...

Aura's **says** modality represents affirmation. (1/2)

$$\frac{\Gamma \vdash P : \textbf{Prop}}{\Gamma \vdash A \ \textbf{says} \ P : \textbf{Prop}}$$

The proposition "Principal A affirms proposition P."

$$\frac{\Gamma \vdash P : \textbf{Prop}}{\Gamma \vdash \textbf{sign}(A, P) : A \ \textbf{says} \ P}$$

A's signature on P. P might be unprovable.

Aura's **says** modality represents affirmation. (1/2)

$$\frac{\Gamma \vdash P : \mathbf{Prop}}{\Gamma \vdash A \ \mathbf{says}\ P : \mathbf{Prop}}$$

The proposition "Principal A affirms proposition P."

$$\frac{\Gamma \vdash P : \mathbf{Prop}}{\Gamma \vdash \mathbf{sign}(A, P) : A \ \mathbf{says}\ P}$$

A's signature on P. P might be unprovable.

Aura's **says** modality represents affirmation. (2/2)

$$\frac{\Gamma \vdash p : P}{\Gamma \vdash \textbf{return@}[A]p : A \ \textbf{says} \ P}$$

A affirms proven propositions.

$$\frac{\Gamma \vdash p : A \ \textbf{says} \ P \qquad \Gamma, x : P \vdash q : A \ \textbf{says} \ Q}{\Gamma \vdash \textbf{bind} \ x = p \ \textbf{in} \ q : A \ \textbf{says} \ Q}$$

A affirms the result of hypothetical reasoning. We can "reason from A's point of view."

(These are standard monad rules.)

Aura$_0$'s syntax supports restricted dependent types.

Partial syntax

$$
\begin{array}{lcll}
t & ::= & \textbf{string} \mid \textbf{prin} & \text{Base types} \\
 & \mid & x \mid a & \text{Variables and constants} \\
 & \mid & t \ \textbf{says} \ t & \text{Says modality} \\
 & \mid & (x{:}\, t) \rightarrow \ t & \text{Logical implication/quantification} \\
 & \mid & (x{:}\ t) \ \Rightarrow \ t & \text{Computational arrows} \\
 & \mid & \{x{:}\, t; t\} & \text{Dependent pair type} \\
 & \mid & t \ t & \text{Application} \\
 & & \vdots &
\end{array}
$$

Syntactic separation of computation and logical arrows stop arbitrary resource-effects from polluting the logic.

Dependent types allow for expressive rules.

Example (Bob acts for Alice)

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

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 validity)

$$\text{Alice } \textbf{says } ((x\colon \textbf{string}) \to \text{ Bob } \textbf{says } \text{valid } x \to \text{valid } x)$$

Dependent types allow for expressive rules.

Example (Bob acts for Alice)

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

Example (Bob acts for Alice only regarding validity)

$$\text{Alice } \textbf{says } ((x\colon \textbf{string}) \rightarrow \text{ Bob } \textbf{says } \text{valid } x \rightarrow \text{valid } x)$$

Example (Kernel allows RPC calls on strings endorsed by Alice)

$$\text{Kernel } \textbf{says } ((x\colon \textbf{string}) \rightarrow \text{ Alice } \textbf{says } \text{valid } x \rightarrow \text{OkToRPC } x)$$

Problem 3

The system may be configured with incorrect formal rules.

Aura Solution

Consistently log runtime proof objects for later analysis.

[Wee 95], [Cederquist+ 05]

When something unexpected happens: look at the log.

- System design guarantees a one-to-one correspondence between log entries and resource state changes.
- If a Alice's signature does not appear in a log entry, she could not have caused the associated action.
- Proofs get convoluted, but *proof reduction* can restore clarity.

Example (A convoluted proof)

$$(\lambda x.\lambda y.y)\ (\mathbf{sign}(\mathsf{Alice}, P))\ (\mathbf{sign}(\mathsf{Bob}, Q))$$

Here Alice's signature is "irrelevant."

Reduction relation includes special cases for **bind**.

R-Bind-Specious

$$\frac{x \notin fv(t_2)}{\textbf{bind}\ x = t_1\ \textbf{in}\ t_2 \longrightarrow t_2}$$

Drops unused hypotheses. Most interesting when t_1 is a signature.

R-Bind-Commute

$$\frac{y \notin fv(t_3)}{\begin{array}{c}\textbf{bind}\ x = (\textbf{bind}\ y = t_1\ \textbf{in}\ t_2)\ \textbf{in}\ t_3 \longrightarrow \\ \textbf{bind}\ y = t_1\ \textbf{in}\ \textbf{bind}\ x = t_2\ \textbf{in}\ t_3\end{array}}$$

Commutation rule that can enable further reductions.

. . . plus standard β and structural rules.

Proof reduction

Example (The same convoluted proof)

$$(\lambda x.\lambda y.y)\ (\mathbf{sign}(\mathsf{Alice}, P))\ (\mathbf{sign}(\mathsf{Bob}, Q)) \longrightarrow^{*} \mathbf{sign}(\mathsf{Bob}, Q)$$

Example (Reducing by special bind rules)

$$\begin{aligned}
&\mathbf{bind}\ x = \big(\mathbf{bind}\ y = (f\ \mathbf{sign}(\mathsf{Bob}, Q)) \\
&\qquad\qquad\quad \mathbf{in}\ \mathbf{sign}(\mathsf{Alice}, P)\big) \\
&\quad \mathbf{in}\ x \\
&\longrightarrow^{*} \mathbf{sign}(\mathsf{Alice}, P)
\end{aligned}$$

Logical properties of Aura_0

Theorem (Subject Reduction)

If $p \longrightarrow p'$ and $\Gamma \vdash p : s$ then $\Gamma \vdash p' : s$.

Theorem (Confluence)

If $p \longrightarrow^* p_1$, and $p \longrightarrow^* p_2$, then there exists p_3 such that $p_1 \longrightarrow^* p_3$ and $p_2 \longrightarrow^* p_3$.

Theorem (Strong Normalization)

If $\Gamma \vdash p : s$, then p is strongly normalizing (SN). That is, all reduction sequences starting with p halt.

Aura$_0$ is strongly normalizing.

Fact

The Calculus of Constructions with dependent pairs (CoC) is SN.

[Geuvers 95]

Proof Idea

Show Aura reductions can be simulated in terminating system based on CoC.

$$\begin{array}{ccccccccc}
p_0 & \longrightarrow & p_1 & \longrightarrow & & \longrightarrow & p_2 & \longrightarrow & \cdots \\
\downarrow [\![\cdot]\!] & & \downarrow [\![\cdot]\!] & & & & \downarrow [\![\cdot]\!] & & \\
t_0 & \longrightarrow_{cc} & t_1 & \longrightarrow_{cc} & t_1' & \longrightarrow_{cc} & t_2 & \longrightarrow_{cc} & \cdots
\end{array}$$

Take home message

Aura is a framework for proof carrying authorization and audit.

Aura includes

- a small and generic trusted computing base,
- an expressive authorization logic, and
- a principled audit methodology.

Take home message

Aura is a framework for proof carrying authorization and audit.

Aura includes

- a small and generic trusted computing base,
- an expressive authorization logic, and
- a principled audit methodology.

Code and technical reports available from
`http://www.cis.upenn.edu/~stevez/sol/aura.html`

Or ask us for a demo!

Bonus Slides

- Typing Rules for Pair and Arrow
- Strong Normalization Proof

Typing rules for arrow and pair.

$$\frac{\Sigma;\Gamma \vdash t_1 : k_1 \qquad \Sigma;\Gamma, x : t_1 \vdash t_2 : k_2 \qquad k_1 \in \{\mathbf{Kind}^{\mathrm{P}}, \mathbf{Prop}, \mathbf{Type}\} \qquad k_2 \in \{\mathbf{Prop}, \mathbf{Type}\}}{\Sigma;\Gamma \vdash (x{:}\, t_1) \to\ t_2 : k_2}$$

$$\frac{\Sigma;\Gamma \vdash t_1 : k_1 \qquad \Sigma;\Gamma, x : t_1 \vdash t_2 : k_2 \qquad k_1, k_2 \in \{\mathbf{Prop}, \mathbf{Type}\}}{\Sigma;\Gamma \vdash \{x{:}\, t_1; t_2\} : k_1}$$

Translation from Aura to CoC erases **says**

Definition

$$
\begin{aligned}
[\![A \ \textbf{says} \ P]\!] &\approx [\![P]\!] \\[2em]
[\![\textbf{return@}[A]p]\!] &\approx [\![p]\!] \\
[\![\textbf{bind}\ x = p{:}\ P\ \textbf{in}\ q]\!] &\approx (\lambda x{:}[\![P]\!] \ .\ [\![q]\!] \ )\ [\![p]\!] \\
[\![\textbf{sign}(A, P)]\!] &\approx x \quad \text{fresh}
\end{aligned}
$$

Translation from Aura to CoC erases **says**

Definition

$$[\![A \ \textbf{says} \ P]\!]_\Delta \approx [\![P]\!]_\Delta$$

$$\begin{aligned}
[\![\textbf{return@}[A]p]\!]_\Delta &\approx [\![p]\!]_\Delta \\
[\![\textbf{bind } x = p{:}\ P \textbf{ in } q]\!]_\Delta &\approx (\lambda x{:}[\![P]\!]_\Delta.\ [\![q]\!]_\Delta)\ [\![p]\!]_\Delta \\
[\![\textbf{sign}(A, P)]\!]_\Delta &\approx \Delta(\textbf{sign}(A, P))
\end{aligned}$$

Where Δ maps signatures to unique fresh variables.
(We'll treat the Δ's implicitly from now on.)

Translation from Aura to CoC erases **says**

Definition

$$\llbracket A \ \textbf{says} \ P \rrbracket_\Delta \quad \approx \quad \llbracket P \rrbracket_\Delta$$

$$\begin{aligned}
\llbracket \textbf{return@}[A]p \rrbracket_\Delta &\quad \approx \quad \llbracket p \rrbracket_\Delta \\
\llbracket \textbf{bind } x = p \text{: } P \textbf{ in } q \rrbracket_\Delta &\quad \approx \quad (\lambda x{:}\llbracket P \rrbracket_\Delta .\, \llbracket q \rrbracket_\Delta)\ \llbracket p \rrbracket_\Delta \\
\llbracket \textbf{sign}(A, P) \rrbracket_\Delta &\quad \approx \quad \Delta(\textbf{sign}(A, P))
\end{aligned}$$

Where Δ maps signatures to unique fresh variables.
(We'll treat the Δ's implicitly from now on.)

Lemma

If p is a well typed term in $Aura_0$, then—for an appropriate context—$\llbracket p \rrbracket$ is well typed in CoC.

Constructions needs a new reduction to simulate **bind**.

Definition (CC′ reduction)

The CC′ reduction relation augments the standard Calculus of Construction reduction relation with

$$\frac{}{(\lambda x{:}t.\, t_1)((\lambda y{:}s.\, t_2)u) \longrightarrow (\lambda y{:}s.\, ((\lambda x{:}t.\, t_1)t_2))u}\ \beta'$$

β' reductions simulate R-Bind-Commute reductions:

$$\frac{y \notin fv(t_3)}{\begin{array}{c}\textbf{bind } x = (\textbf{bind } y = t_1 \textbf{ in } t_2) \textbf{ in } t_3 \longrightarrow \\ \textbf{bind } y = t_1 \textbf{ in bind } x = t_2 \textbf{ in } t_3\end{array}}$$

Aura$_0$ is strongly normalizing.

Lemma

Well typed CoC terms are SN under CC$'$ reduction.

[Lindley 05]

Lemma

If p is a well typed Aura$_0$ term and $p \longrightarrow p'$. Then $[\![p]\!] \longrightarrow^{+}_{CC'} [\![p']\!]$ using one or more steps.

Proof of strong normalization.

Imagine p is an looping Aura$_0$ term. Then by the second lemma we can build a term which loops according to CC$'$. This contradicts the first lemma. $\square$