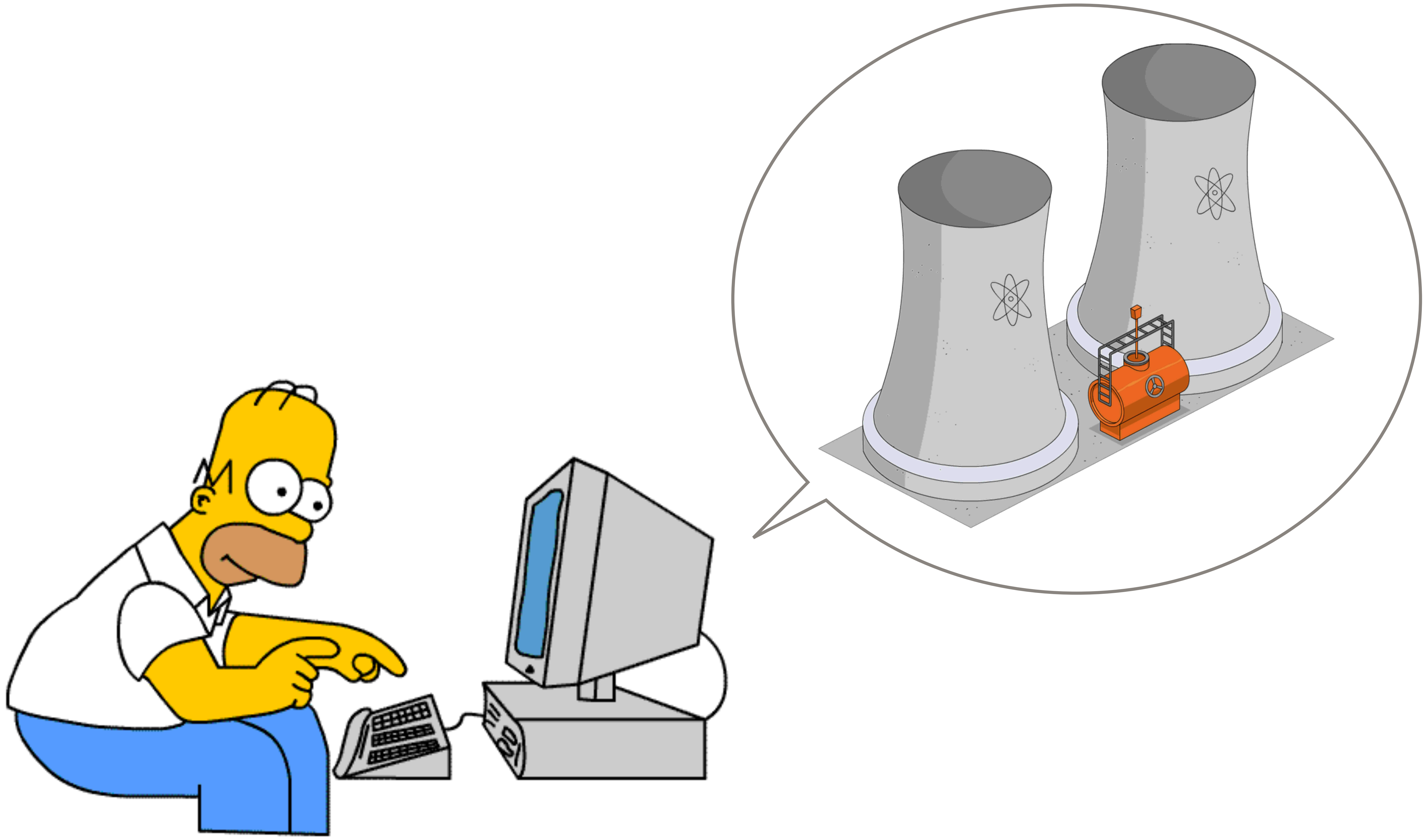
Expressive Authorization Policies using Computation Principals

Anitha Gollamudi

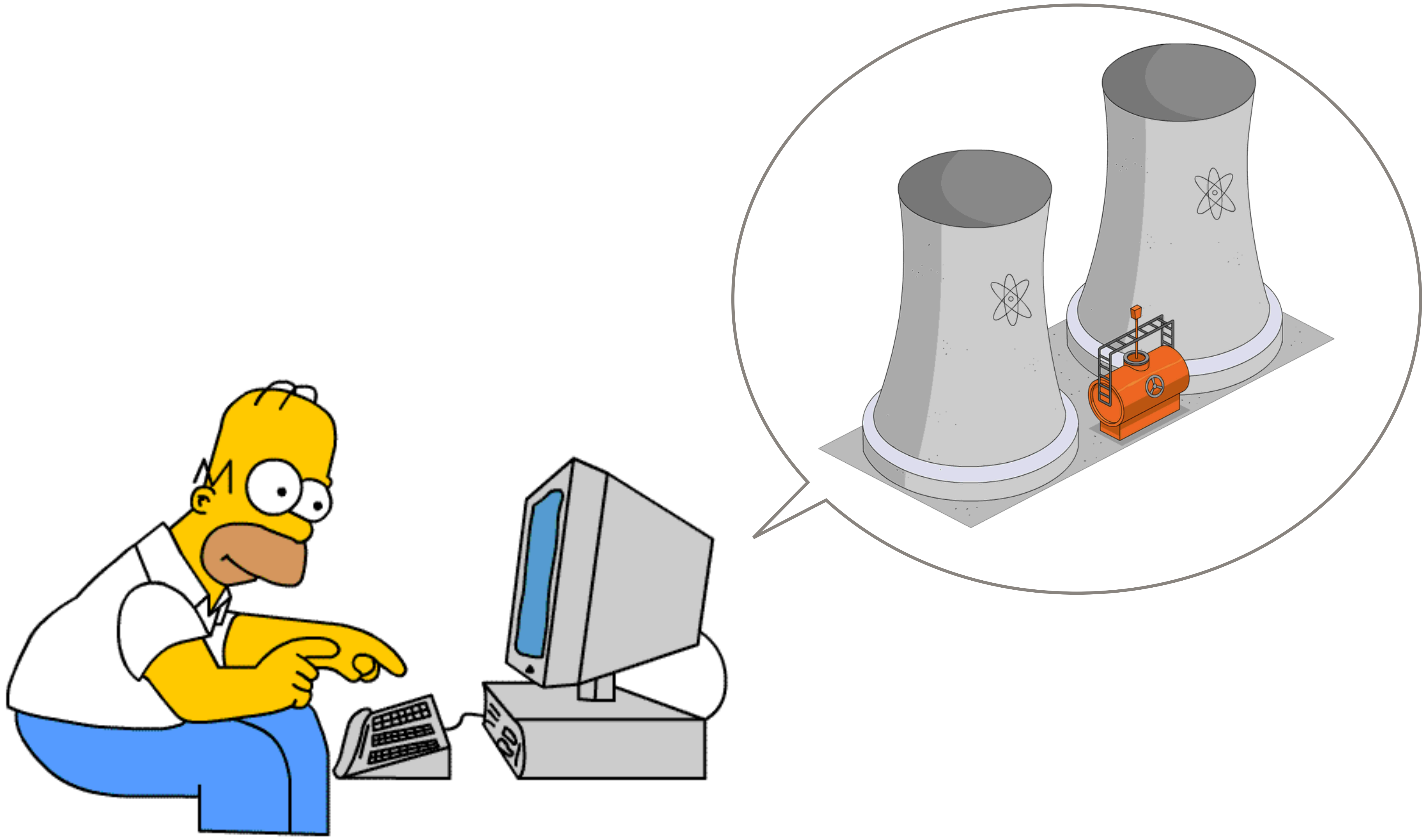
UMASS LOWELL

Stephen Chong

HARVARD

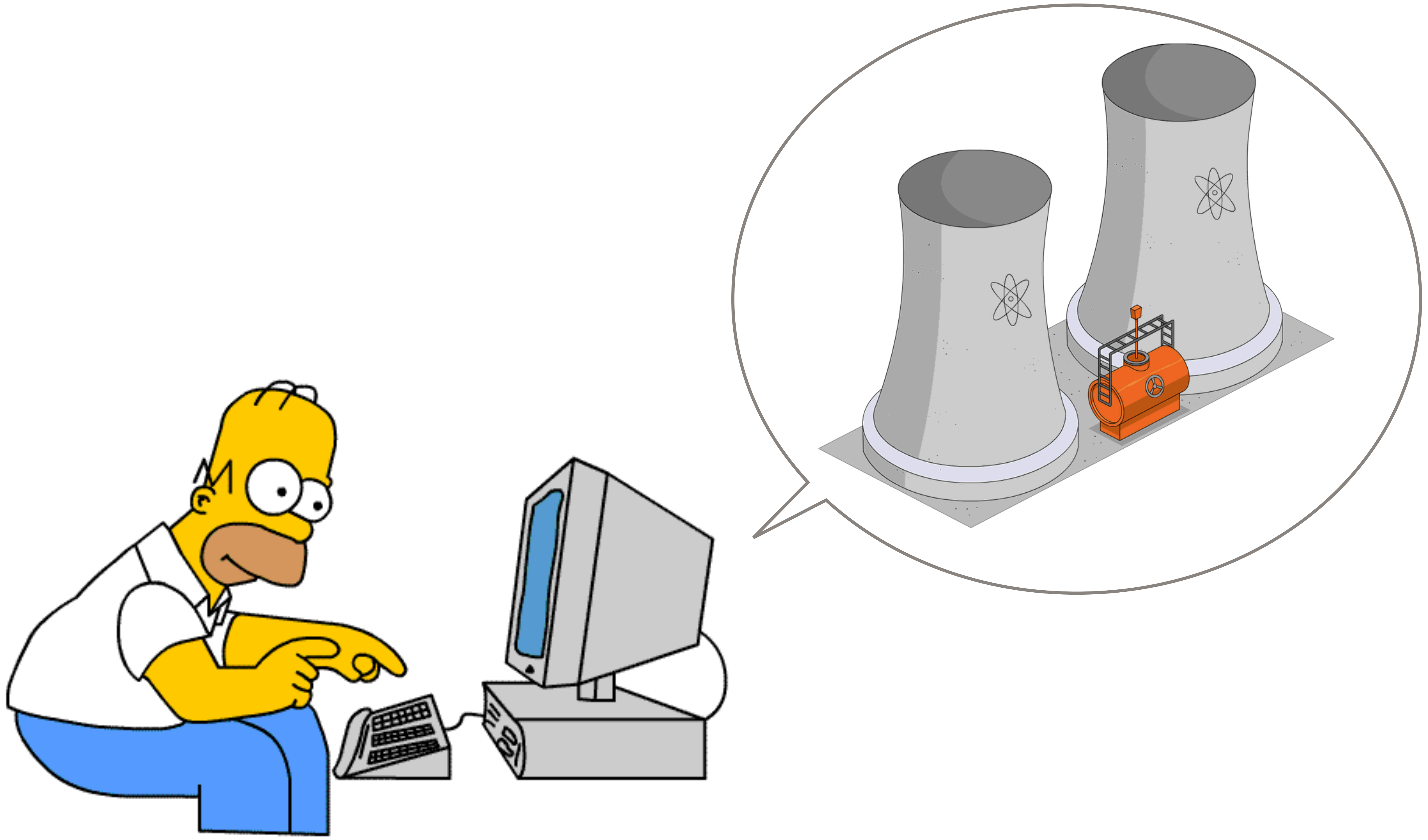


Homer can access nuclear_data



Homer can access nuclear_data

Homer trusts Carl



Homer can access nuclear_data

Homer trusts Carl

Can Carl Access Nuclear_Data?

Authorization Logic

Principled reasoning about authorization decisions

Carl ***speaks for*** Homer

$\forall$P, P ***speaks for*** Homer $\Rightarrow$ P can access nuclear_data

---

Carl can access nuclear_data

Authorization Logic

Principled reasoning about authorization decisions

Carl ***speaks for*** Homer

∀P, P ***speaks for*** Homer ⇒ P can access nuclear_data

Homer trusts/delegates to Carl

Carl can access nuclear_data ✓

Authorization Logic

Principled reasoning about authorization decisions

Access Control Policy

Carl ***speaks for*** Homer

$\forall$P, P ***speaks for*** Homer $\Rightarrow$ P can access nuclear_data

---

Homer trusts/delegates to Carl

Carl can access nuclear_data ✔

Authorization Logic

Principals play a central role

Carl ***speaks for*** Homer

$\forall$P, P ***speaks for*** Homer $\Rightarrow$ P can access nuclear_data

---

Carl can access nuclear_data

Authorization Logic

Principals play a central role

Principal

Carl ***speaks for*** Homer

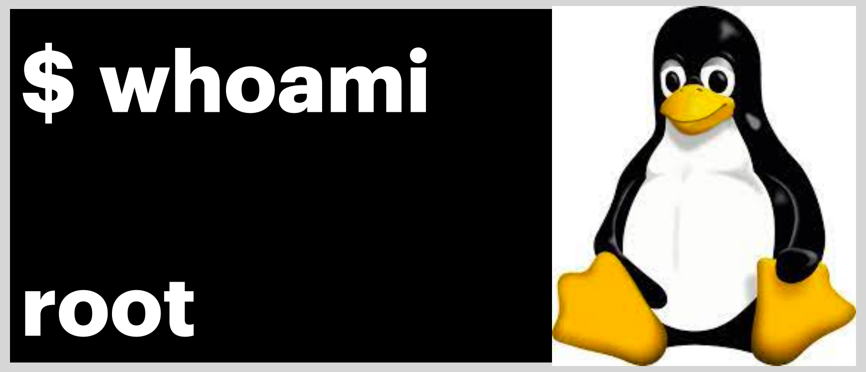
Principal

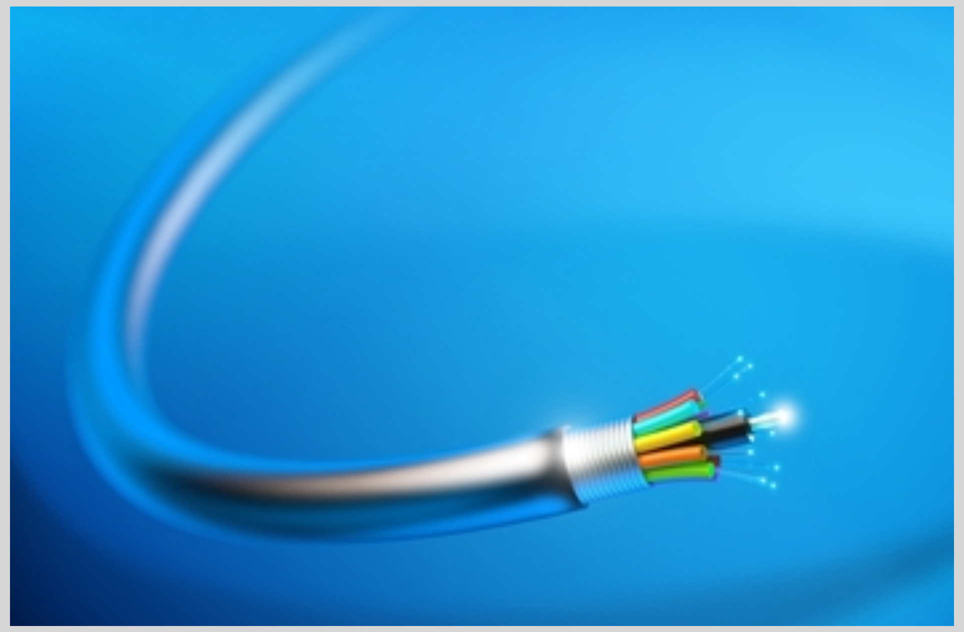
∀P, P ***speaks for*** Homer ⇒ P can access nuclear_data

Principal

Carl can access nuclear_data

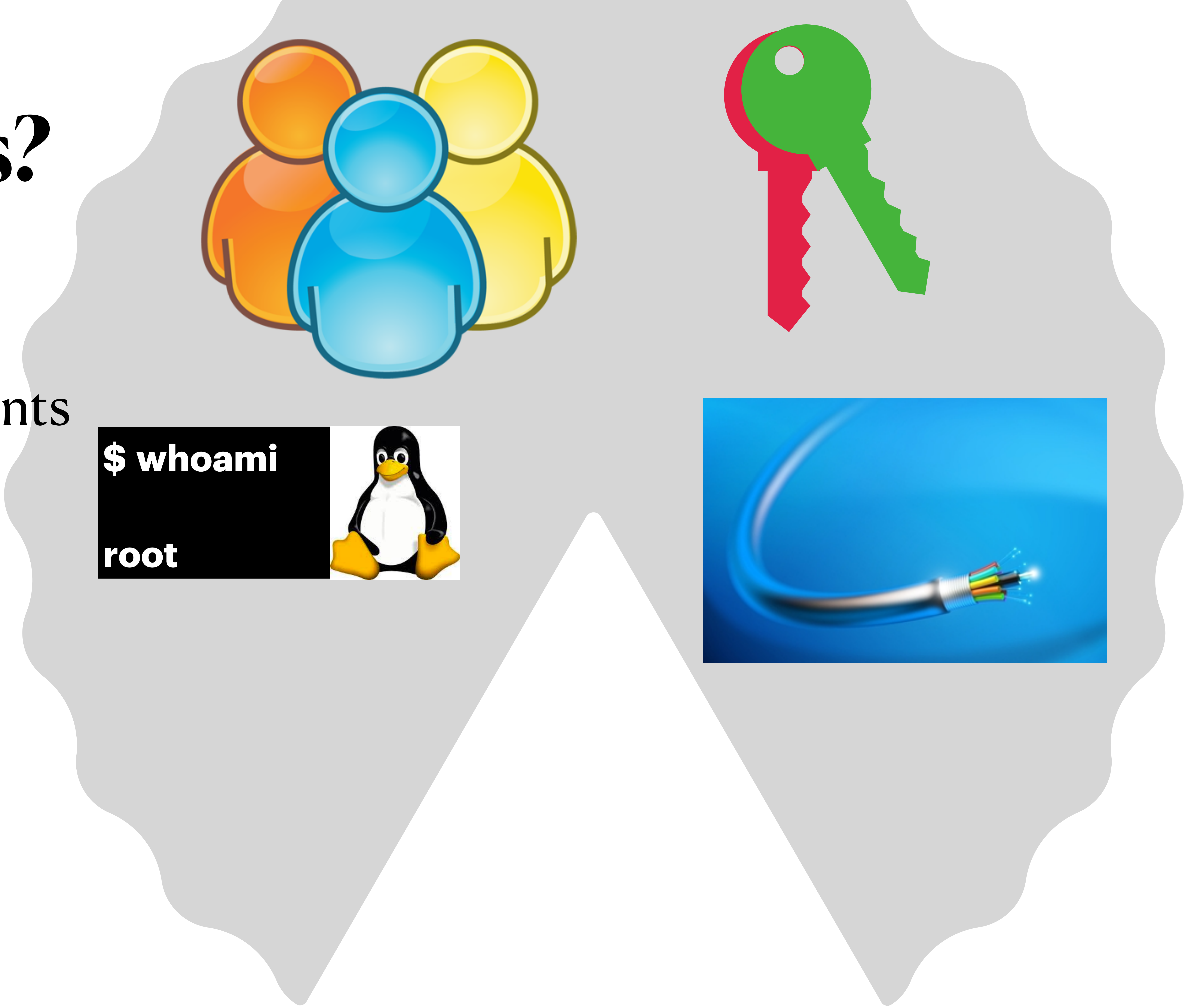
What are Principals?





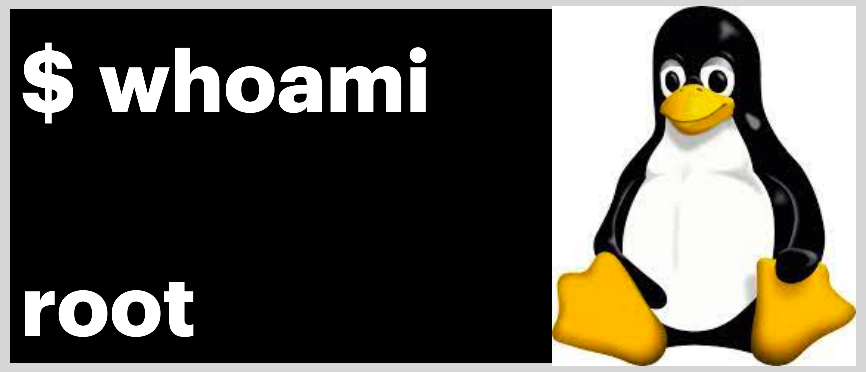
What are Principals?

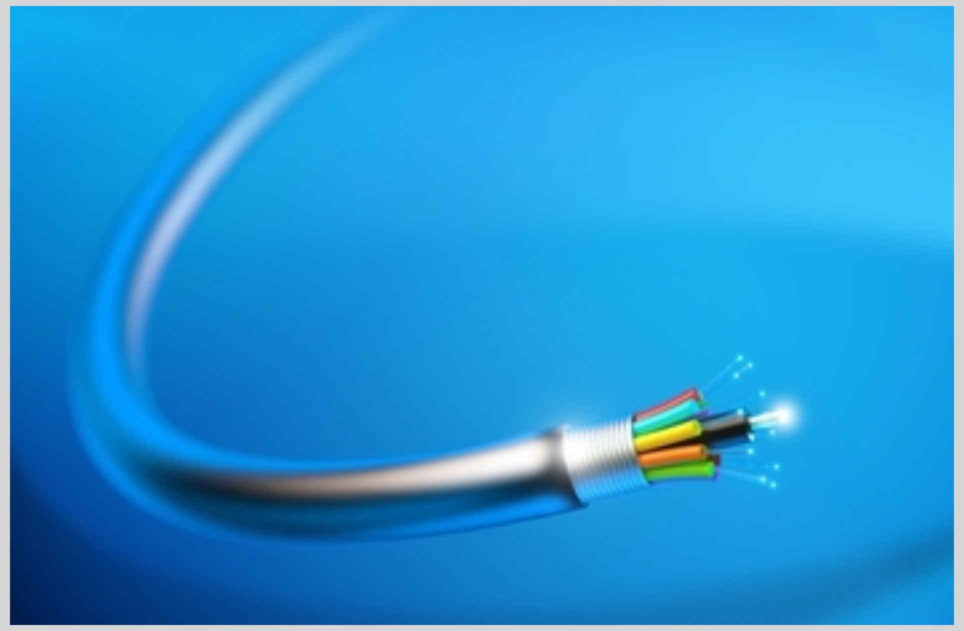
- Entities that can express statements about access control policies



What are Principals?

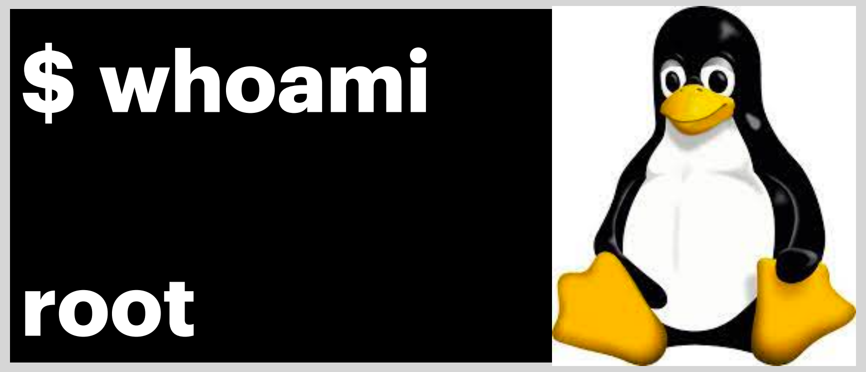
- Entities that can express statements about access control policies
- Examples
  - Users
  - Public keys
  - OS processes
  - Secure channels

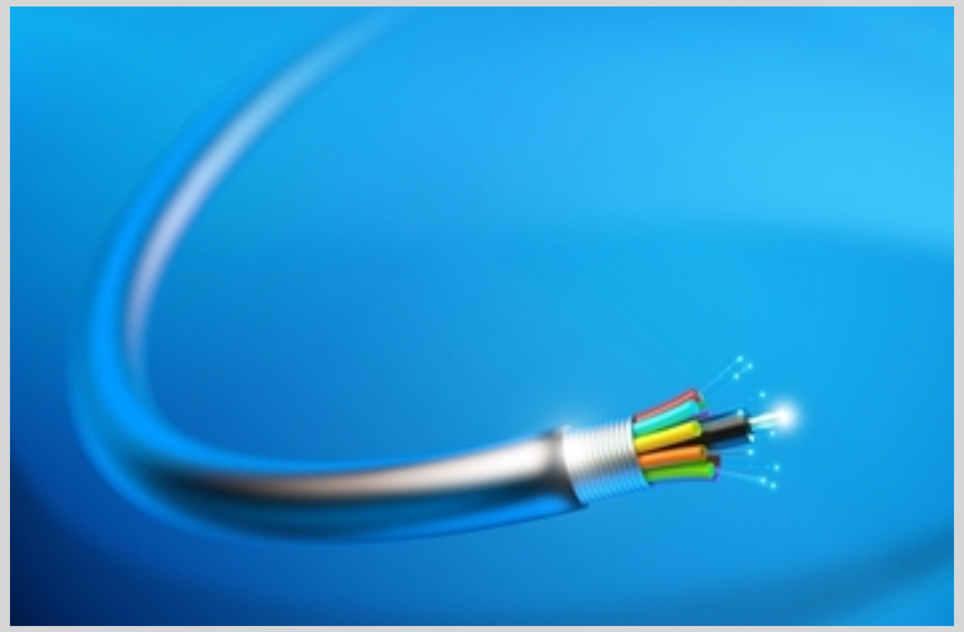




What are Principals?

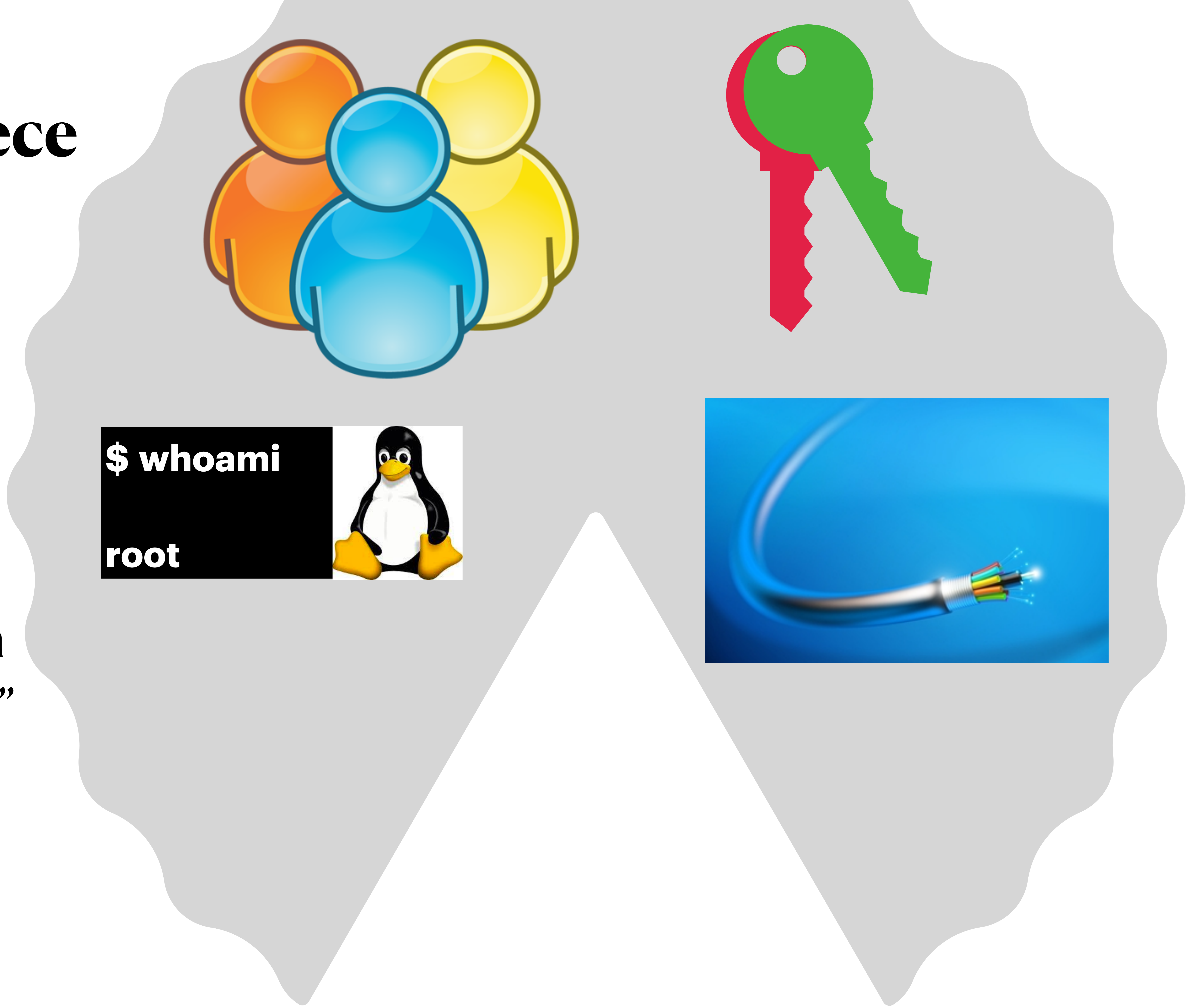
- Entities that can express statements about access control policies
- Examples
  - Users
  - Public keys
  - OS processes
  - Secure channels
- Atomic Principals





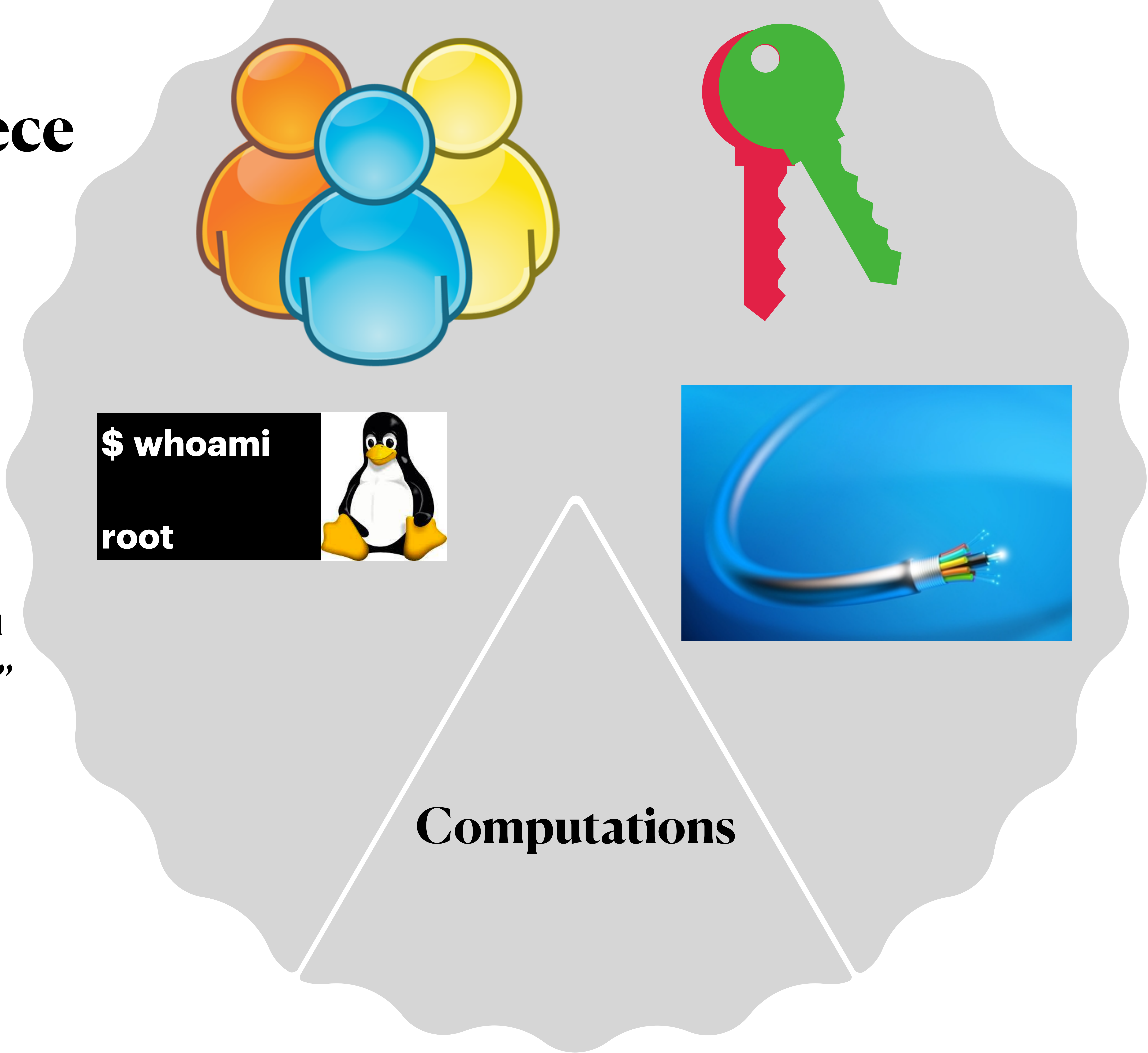
Computations: Missing Piece

- Programs or Computations can also express statements about access control policies
- E.g. Program {P} says “Lenny can access nuclear_data on Tuesday”



Computations: Missing Piece

- Programs or Computations can also express statements about access control policies
- E.g. Program {P} says “Lenny can access nuclear_data on Tuesday”



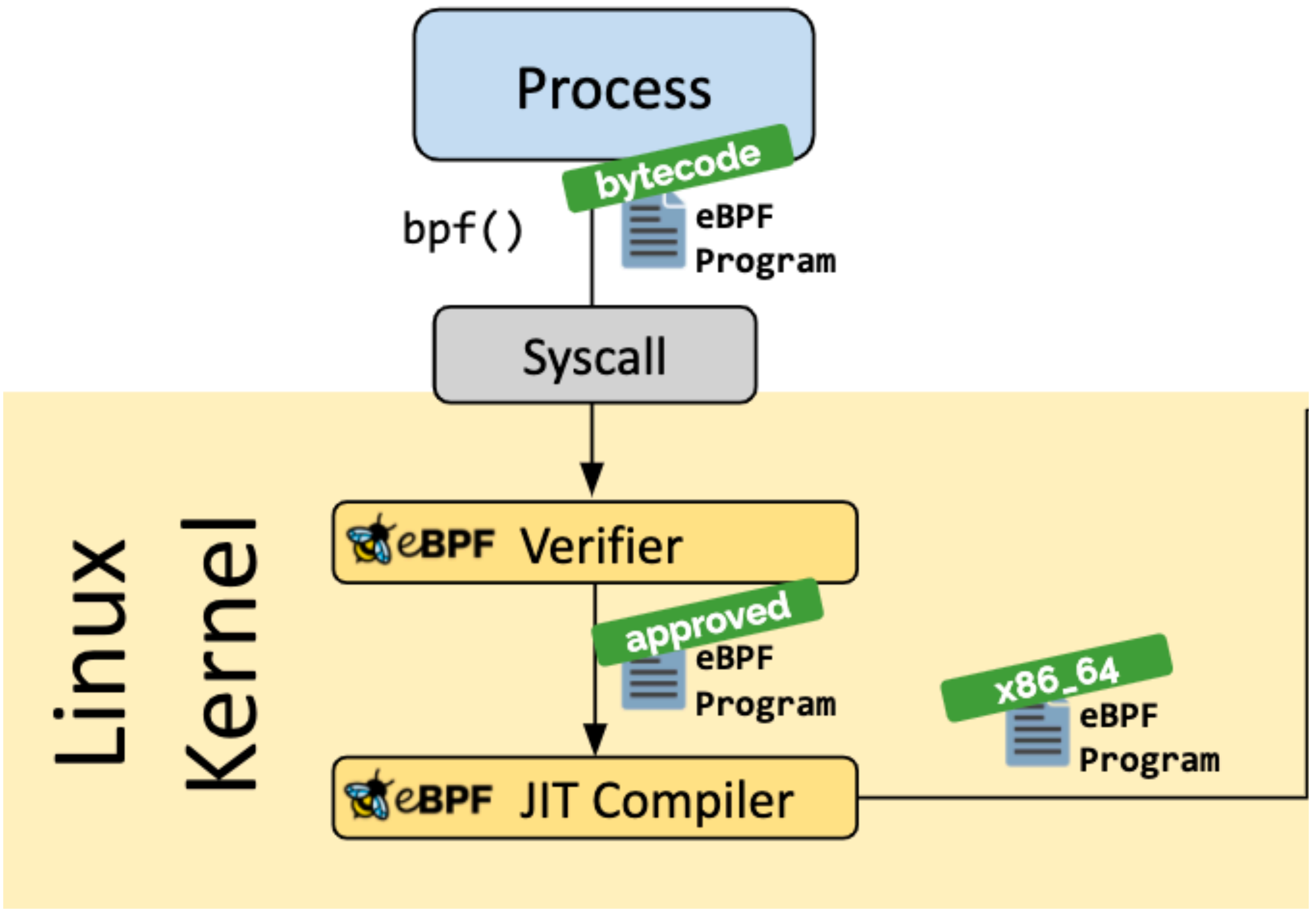
Principals representing computations are Computation Principals

Examples of Computation Principals

Examples of Computation Principals

Mobile code

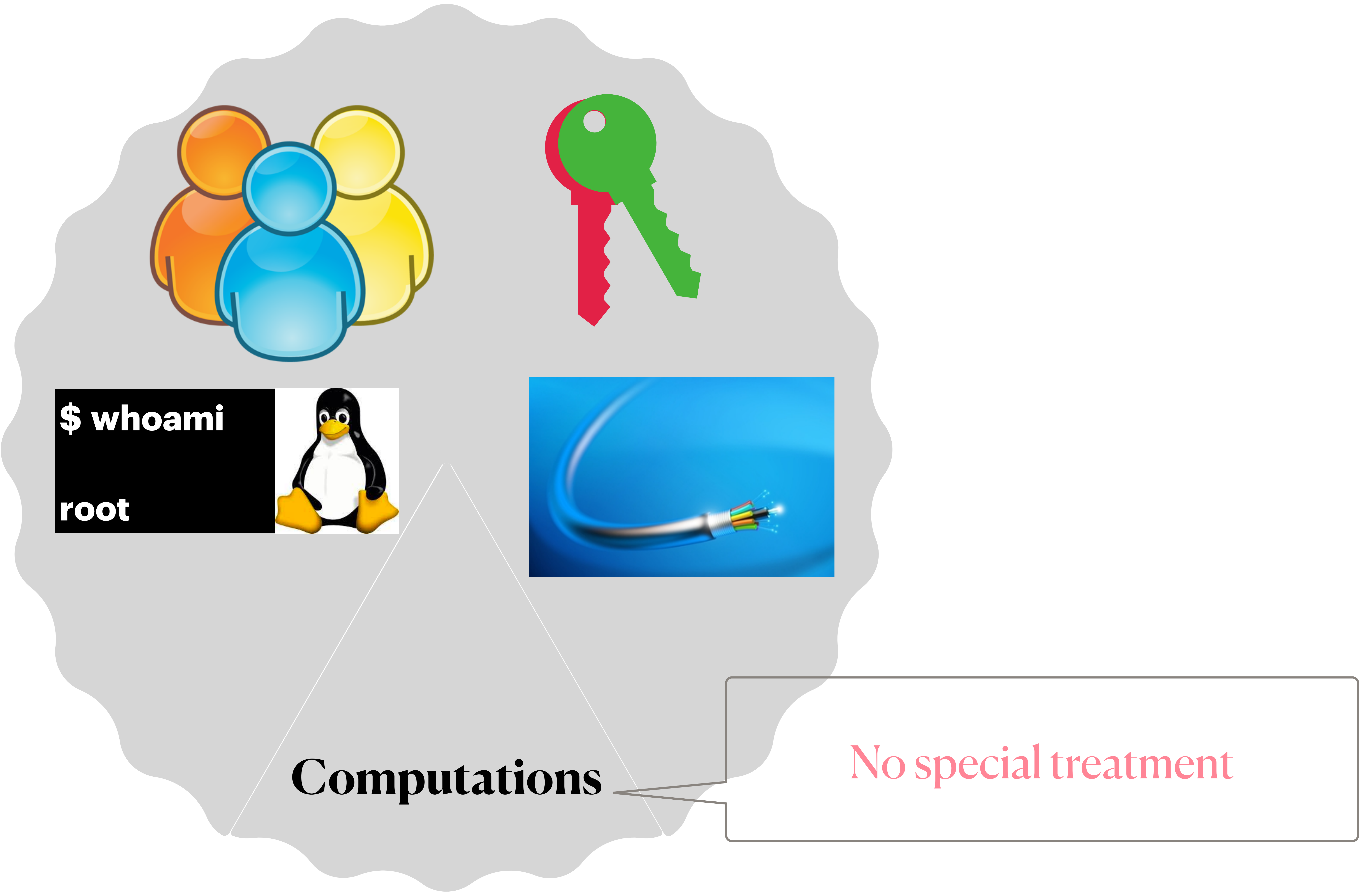
Smart Contracts



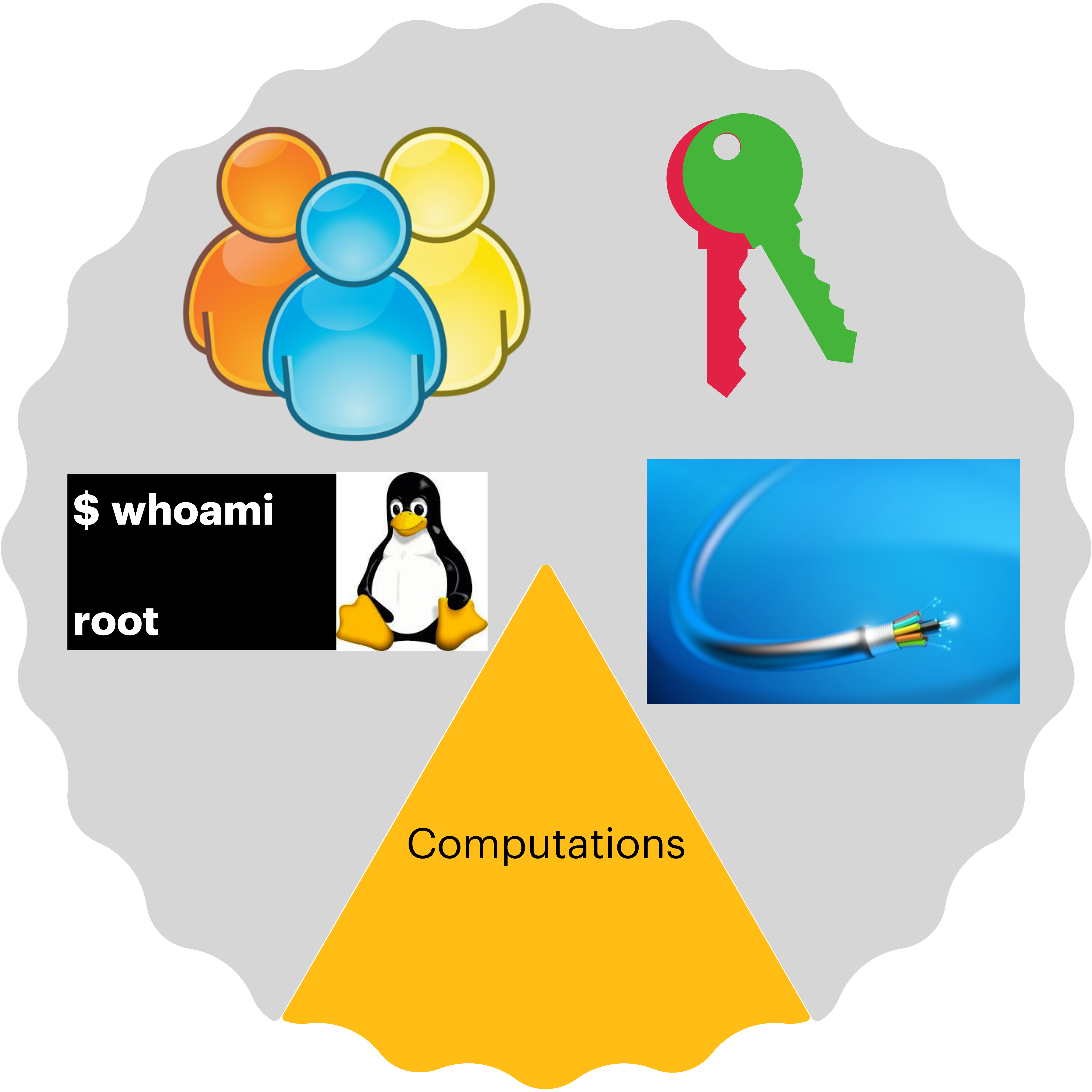
eBPF programs

Trusted Execution Environments

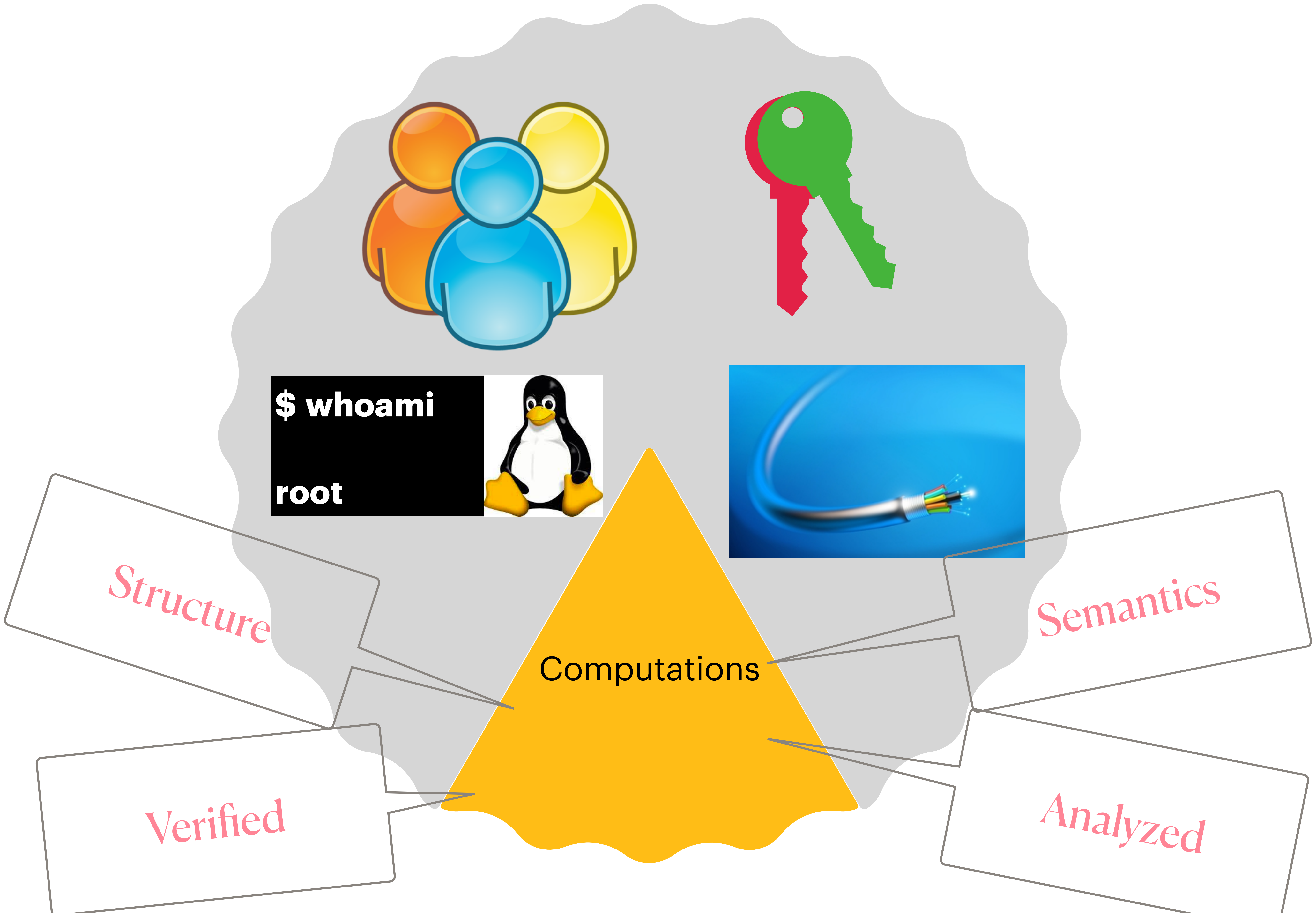
Existing Authorization Logics



But Computation Principals are Distinct

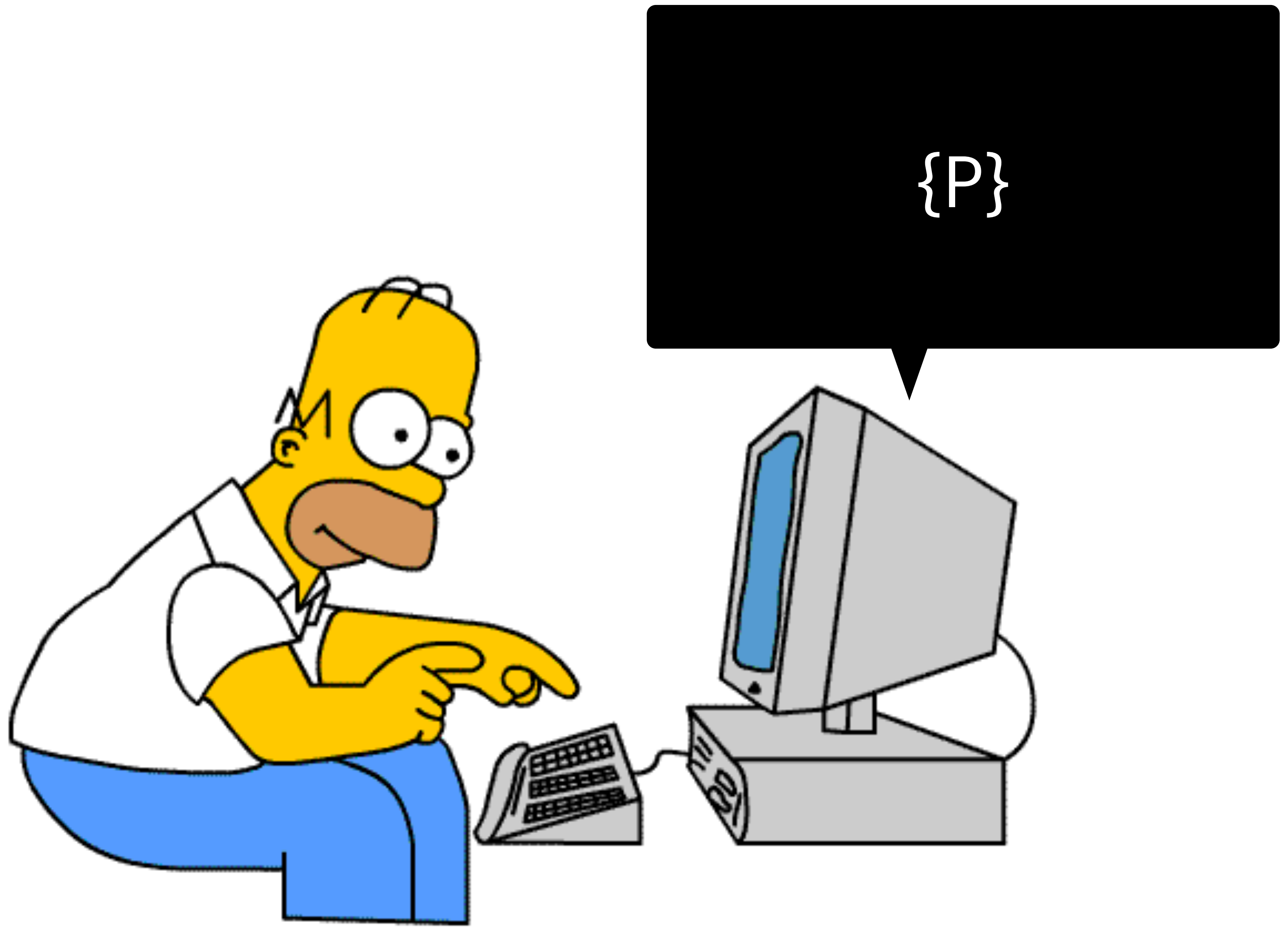


But Computation Principals are Distinct



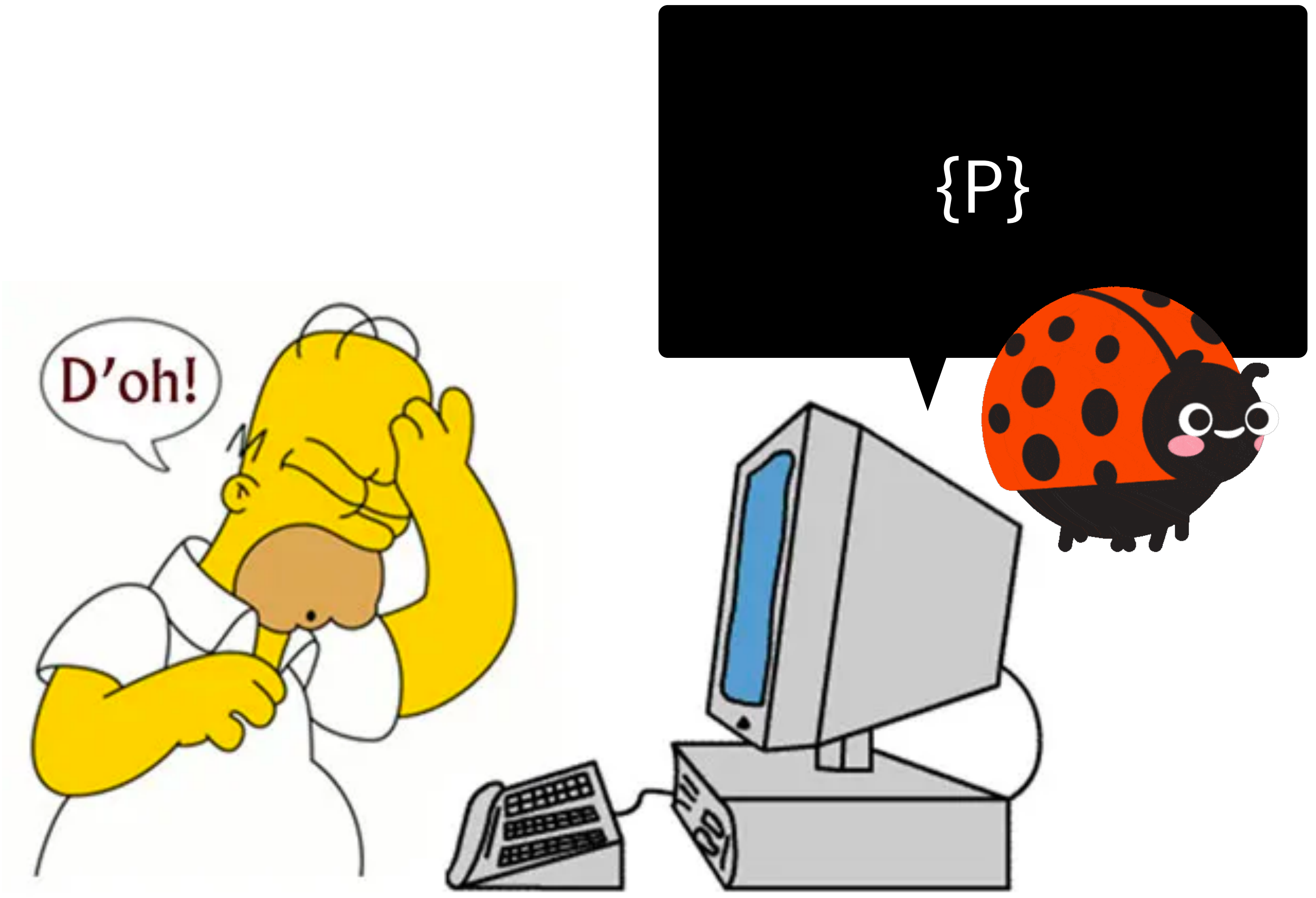
Coal: Authorization logic that distinguishes computation principals from other principals

Express Trust Directly in a Computation



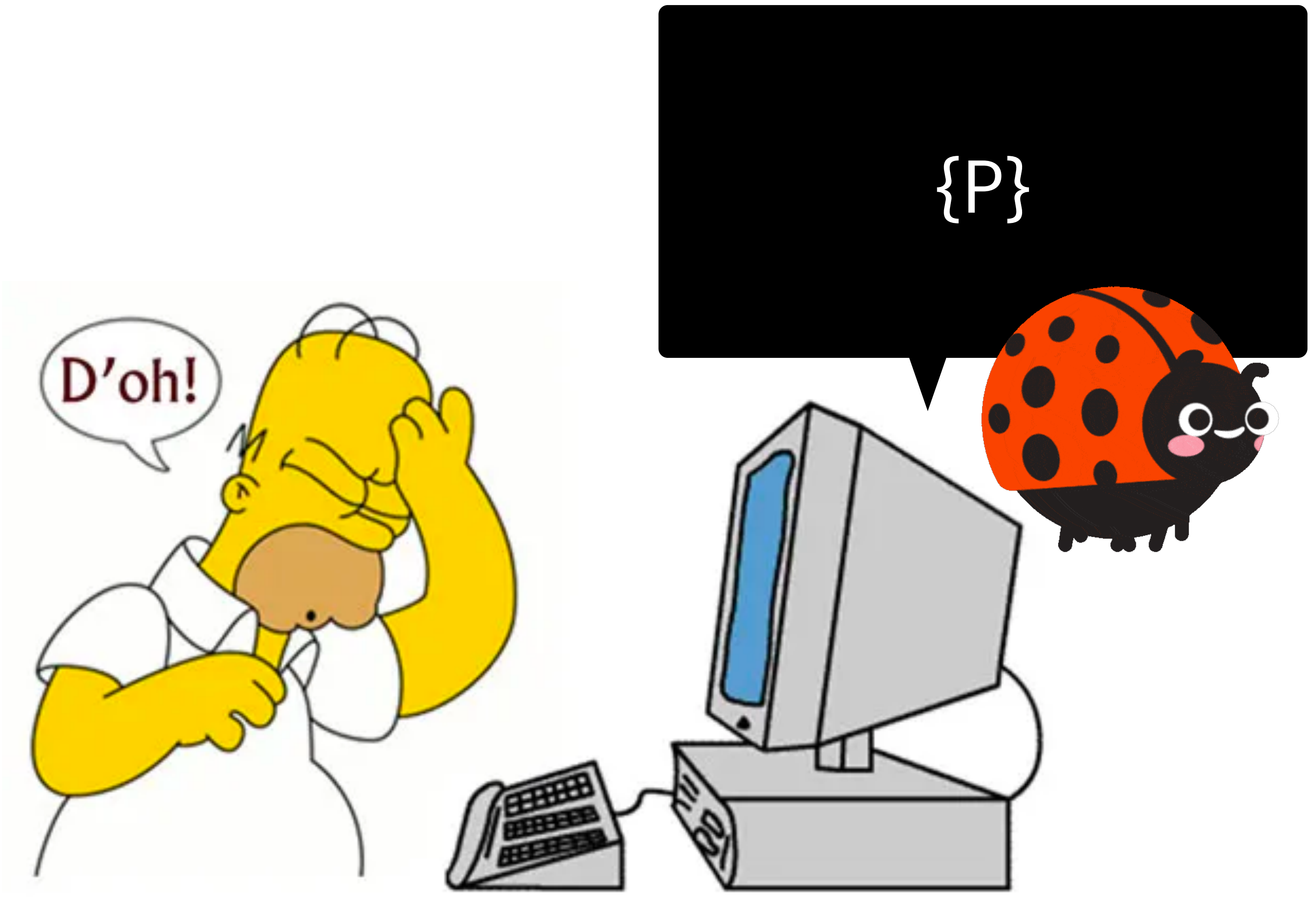
Homer trusts **{P}**

Express Trust Directly in a Computation



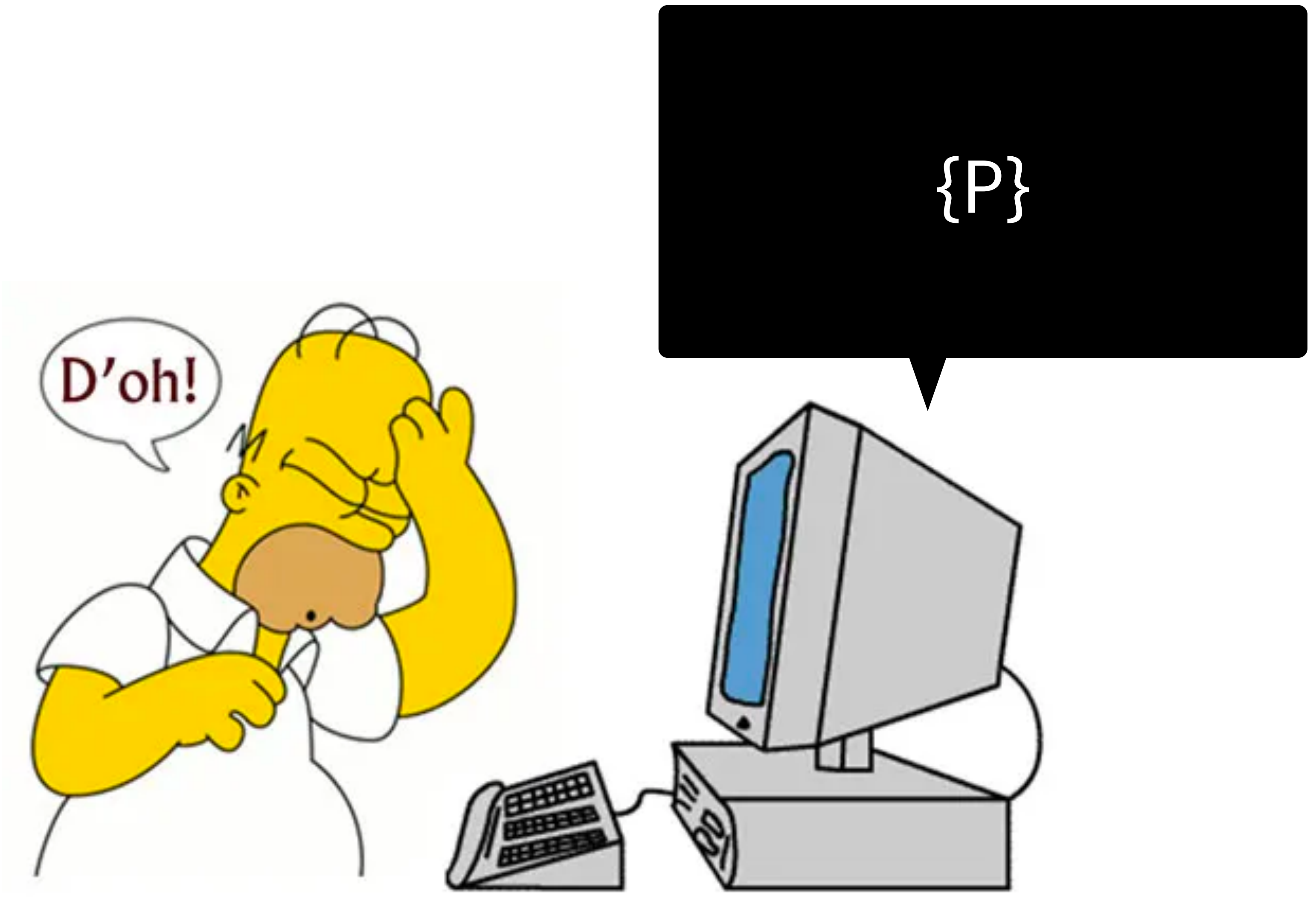
Homer trusts **{P}**

Express Trust Directly in a Computation

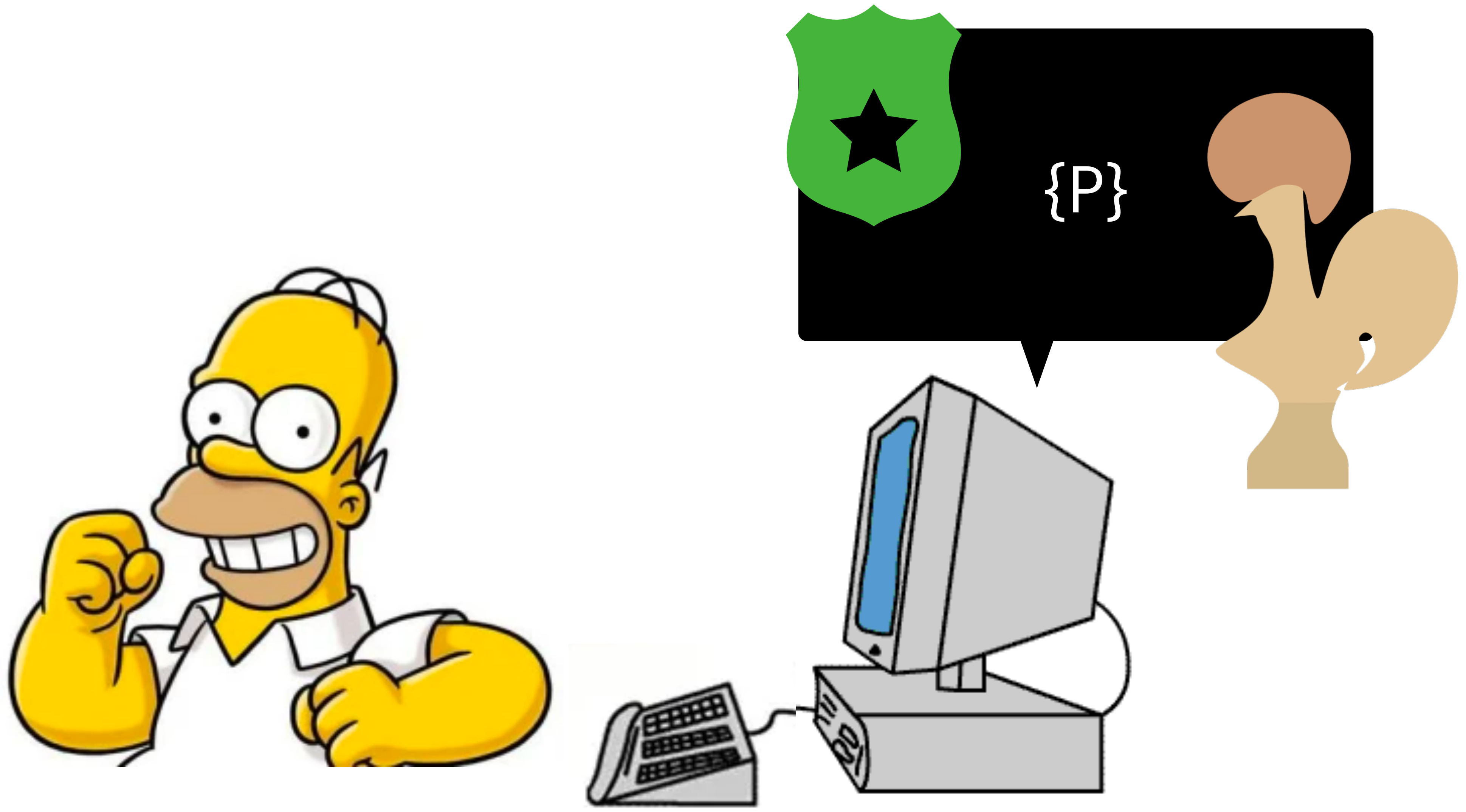


Homer trusts **{P}**

Express Trust Directly in a Computation



Homer trusts **{P}**



Homer trusts **{P}** if **{P}** is verified to be secure (e.g., differentially private)

Challenge: How to Represent a Computation Principal

Can I use the ***Hash Digest*** of the computation?

Why Hash Representation is Not Suitable

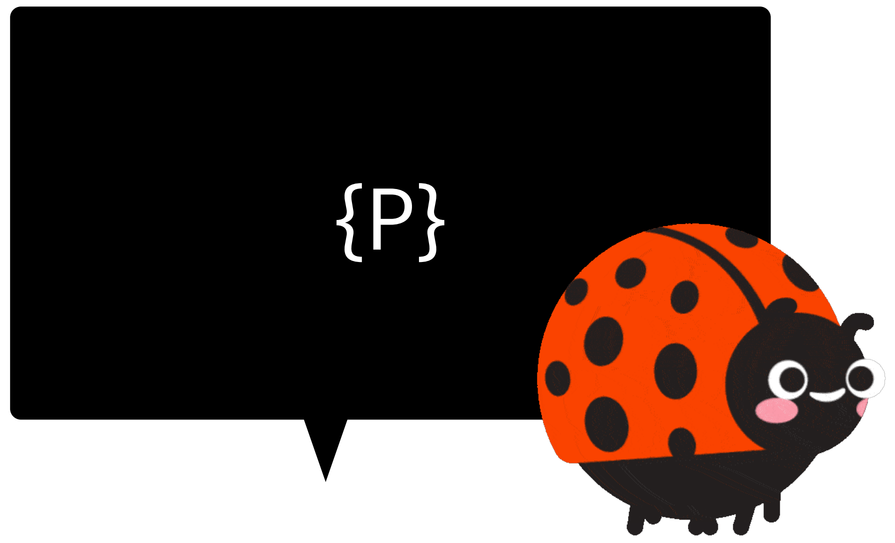
Opaque

Brittle

Why Hash Representation is Opaque? 

Recall that computations have

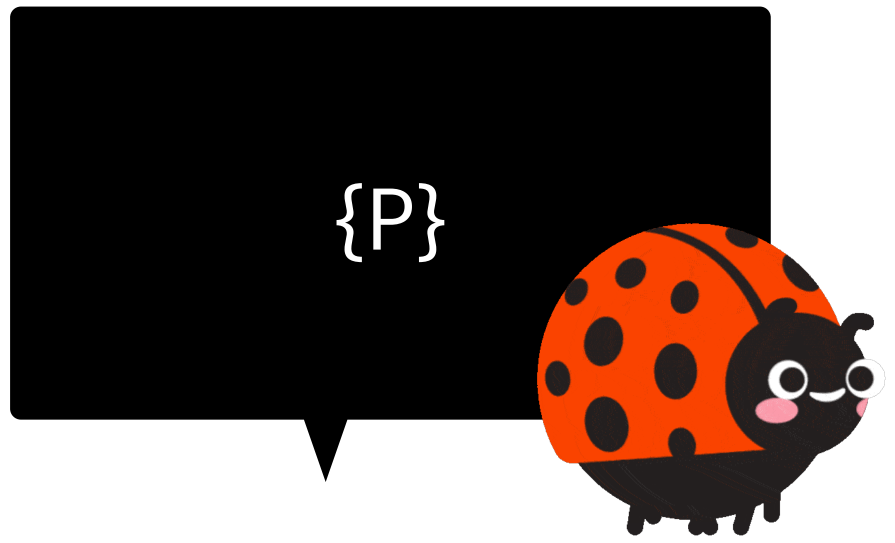
- ✓ Structure
- ✓ Semantics
- ✓ Analyzed
- ✓ Verified



Why Hash Representation is Opaque? 

Recall that computations have

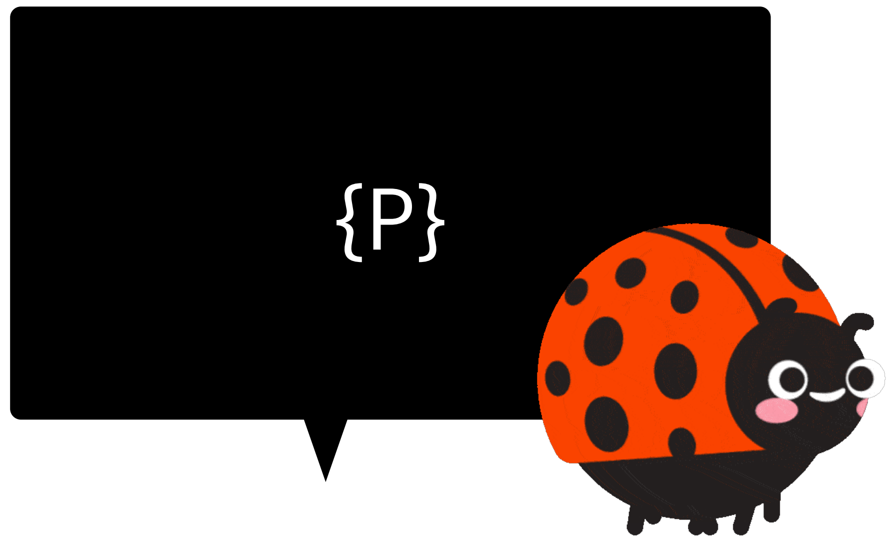
- ✓ Structure
- ✓ Semantics
- ✓ Analyzed
- ✓ Verified



Why Hash Representation is Opaque? 

Recall that computations have

- ✓ Structure
- ✓ Semantics
- ✓ Analyzed
- ✓ Verified

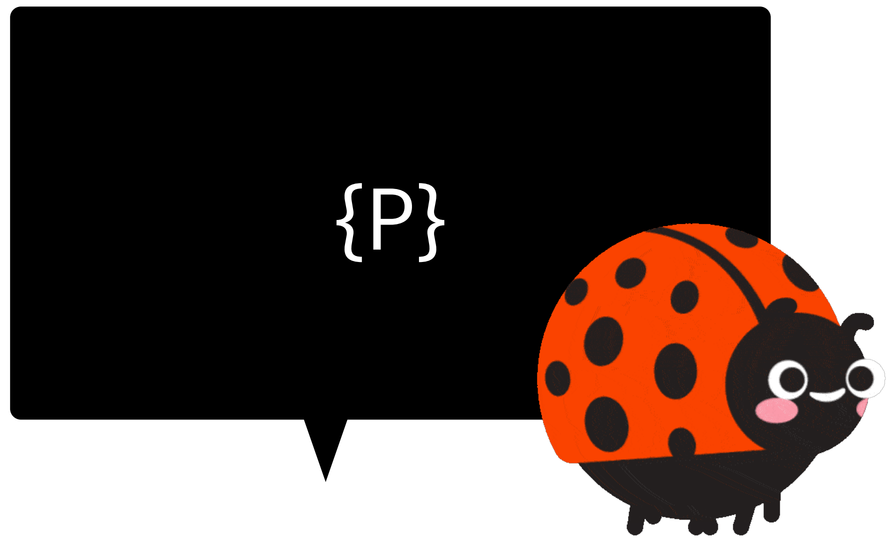


Trust Policy: Homer trusts Hash(**{P}**) if **{P}** is secure

Why Hash Representation is Opaque? 

Recall that computations have

- ✓ Structure
- ✓ Semantics
- ✓ Analyzed
- ✓ Verified



Trust Policy: Homer trusts Hash(**{P}**) if **{P}** is secure

Why Hash Representation is Opaque? 

Recall that computations have

- ✓ Structure
- ✓ Semantics
- ✓ Analyzed
- ✓ Verified

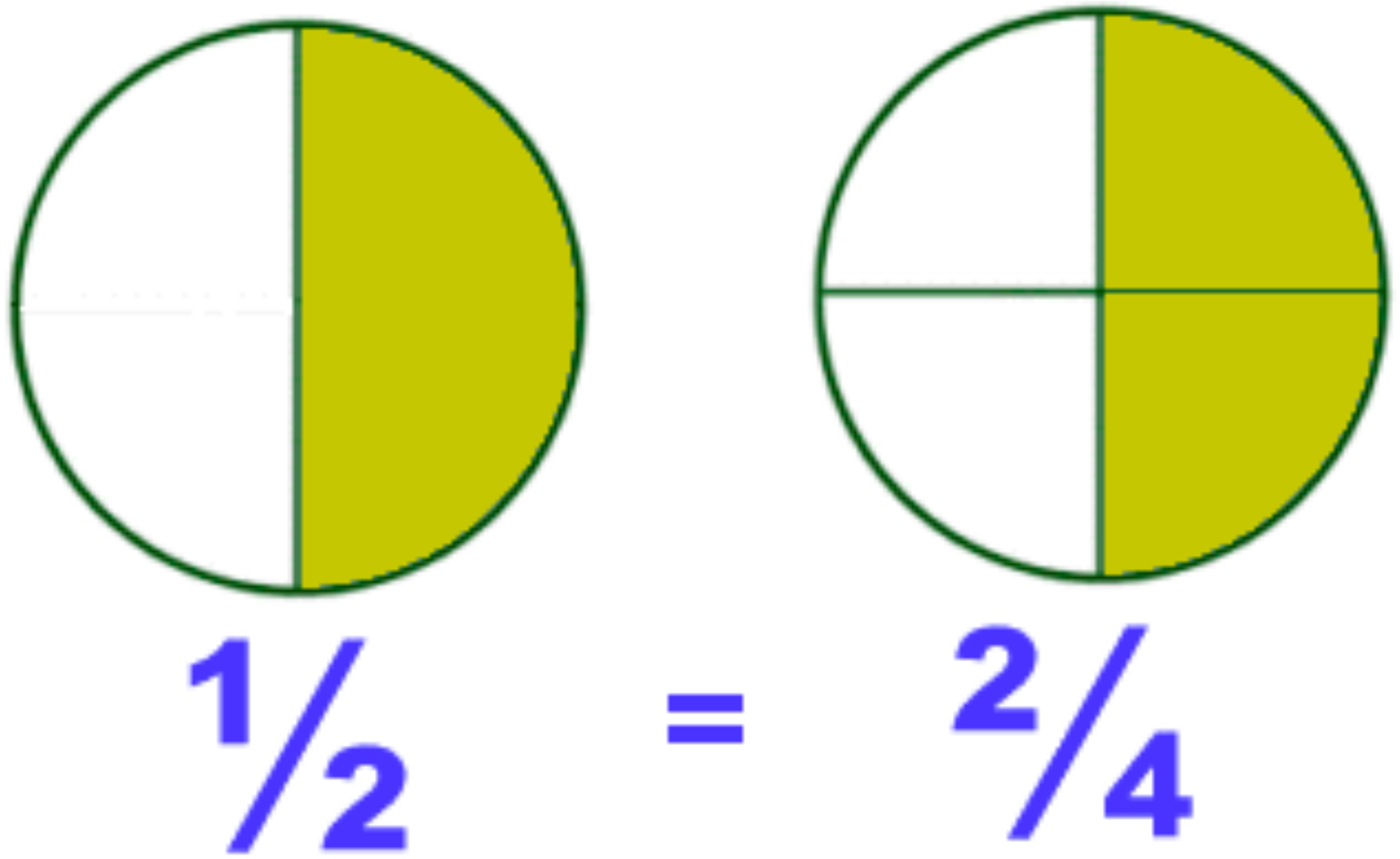
Hash representation loses

- ✗ Structure
- ✗ Semantics
- ✗ Analyzed
- ✗ Verified

Why Hash Representation is Brittle?

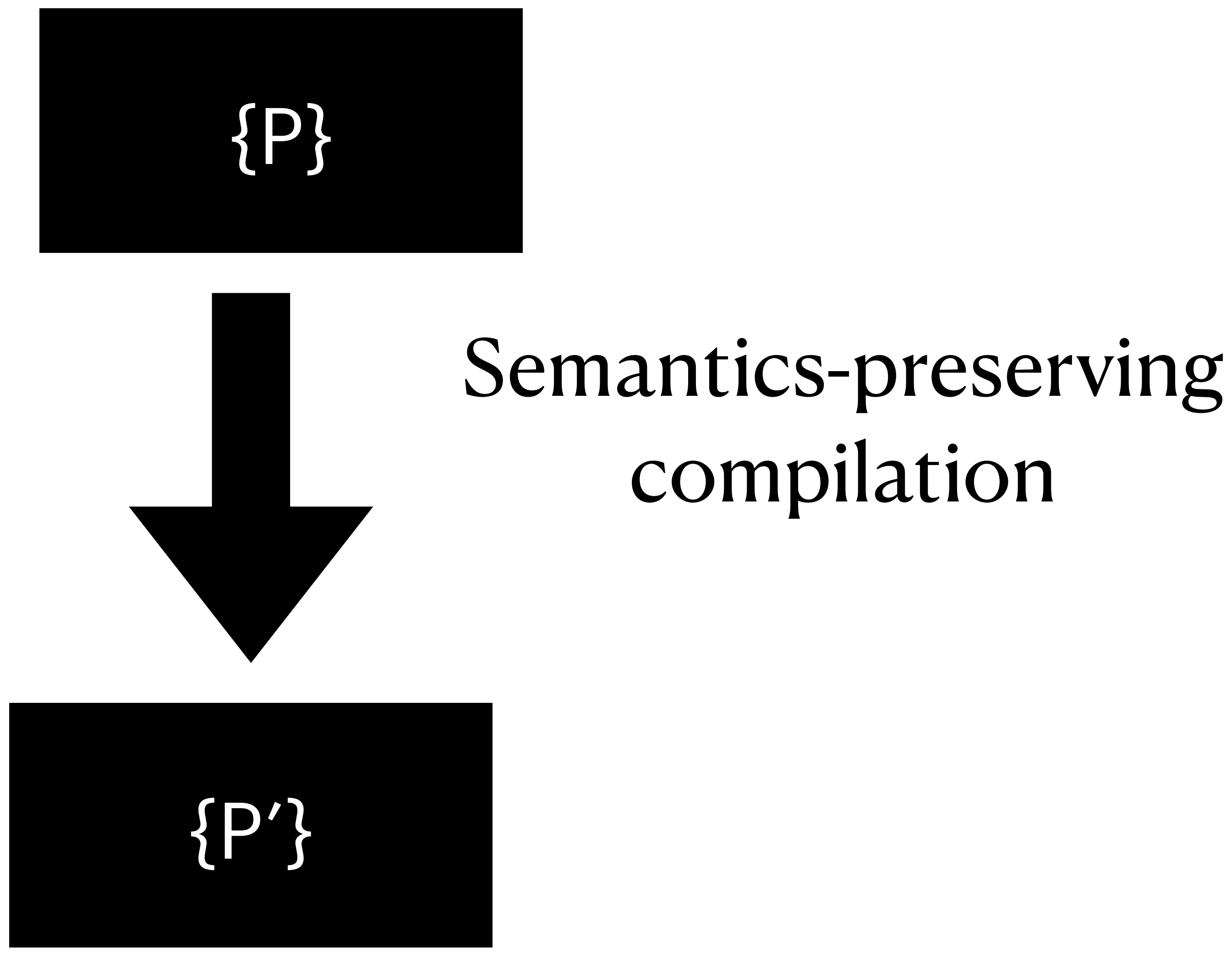
Recall that computations have

- ✓ Structure
- ✓ Semantics
- ✓ Analyzed
- ✓ Verified

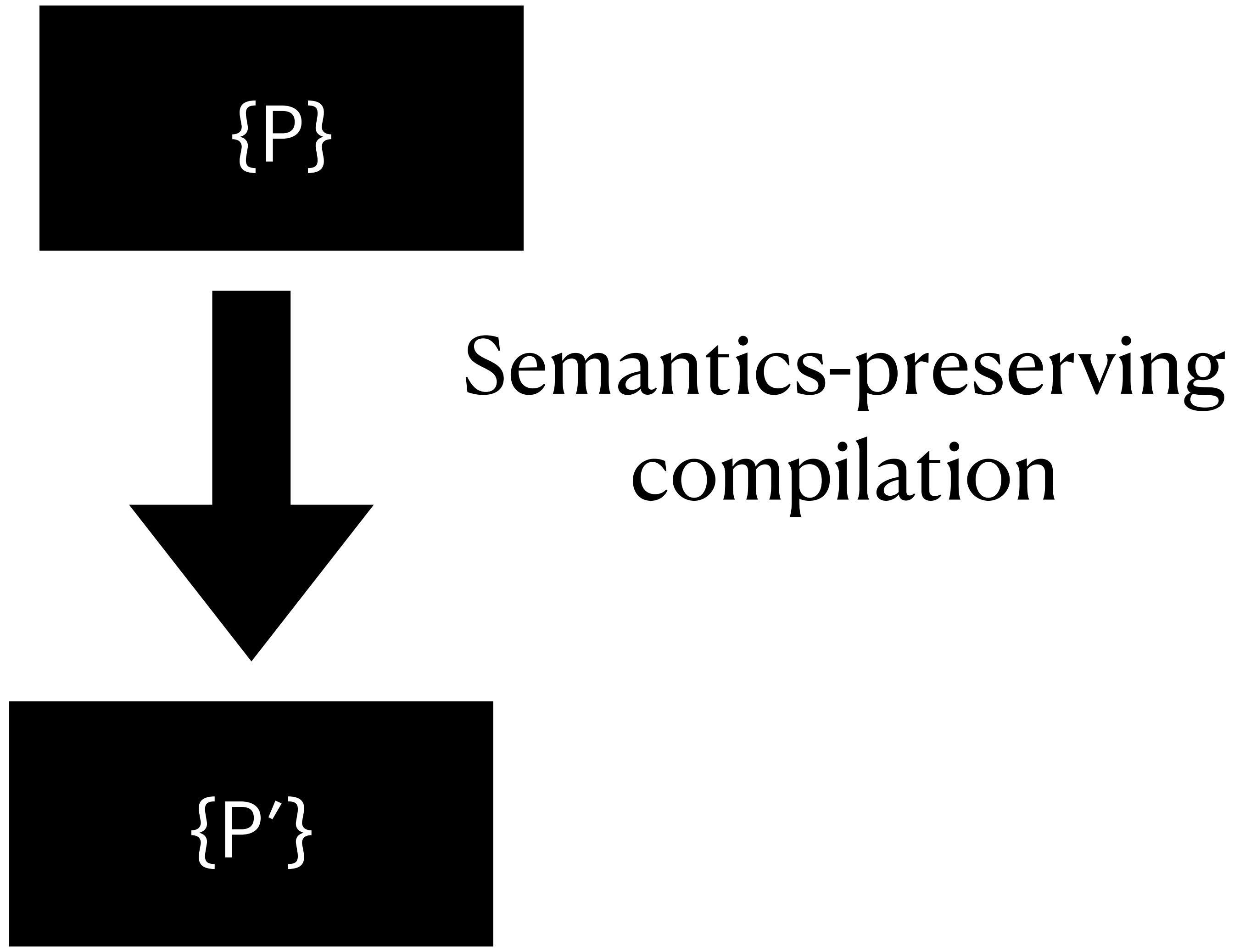


No equational reasoning between computation principals

Why Hash Representation is Brittle?



Why Hash Representation is Brittle?



No equational reasoning

- $P \approx P' \nRightarrow \text{Hash}(P) = \text{Hash}(P')$
- Equivalent programs are treated as **different** principals
- Whenever the computation changes, trust policy changes

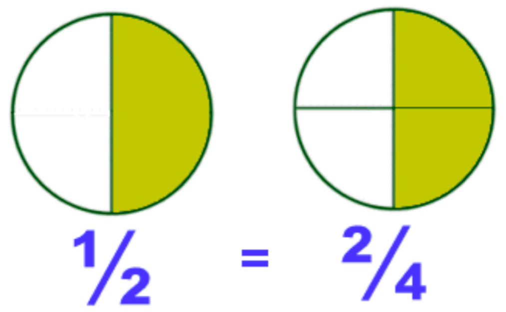
Coal addresses both the challenges

Coal addresses both the challenges

Computation principals can be analyzed for intensional properties

Coal addresses both the challenges

Computation principals can be analyzed for intensional properties



Equivalent computations are treated as equivalent principals

Overview

$$e ::= \ \ldots \ | \ \boldsymbol{\mu} T.e \quad | \quad \mathrm{exec}(e)$$

$$\boldsymbol{\tau} ::= \ \ldots \ | \ p \text{ says } \boldsymbol{\tau} \ | \ \mathrm{code}\{\boldsymbol{\mu} T.e\}$$

Overview

ML/DCC-like

$$e ::= \ \ldots \ | \ \boldsymbol{\mu}T.e \ | \ \mathrm{exec}(e)$$

$$\boldsymbol{\tau} ::= \ \ldots \ | \ p \text{ says } \boldsymbol{\tau} \ | \ \mathrm{code}\{\boldsymbol{\mu}T.e\}$$

Overview

ML/DCC-like

e ::= … | μT.e | exec(e)

τ ::= … | p says τ | code{μT.e}

Principal p supports proposition τ

Overview

ML/DCC-like

Computation Expression

$$e ::= \ \dots \ \mid \ \mu T.e \ \mid \ exec(e)$$

$$\tau ::= \ \dots \mid p \text{ says } \tau \ \mid \ code\{\mu T.e\}$$

Principal p supports proposition τ

Overview

ML/DCC-like

Computation Expression

$$e ::= \ \ldots \ | \ \mu T.e \ | \ exec(e)$$

$$\tau ::= \ \ldots \ | \ p \text{ says } \tau \ | \ code\{\mu T.e\}$$

Principal p supports proposition τ

Computation Principal

Overview

ML/DCC-like

Computation Expression

Run a Computation Principal

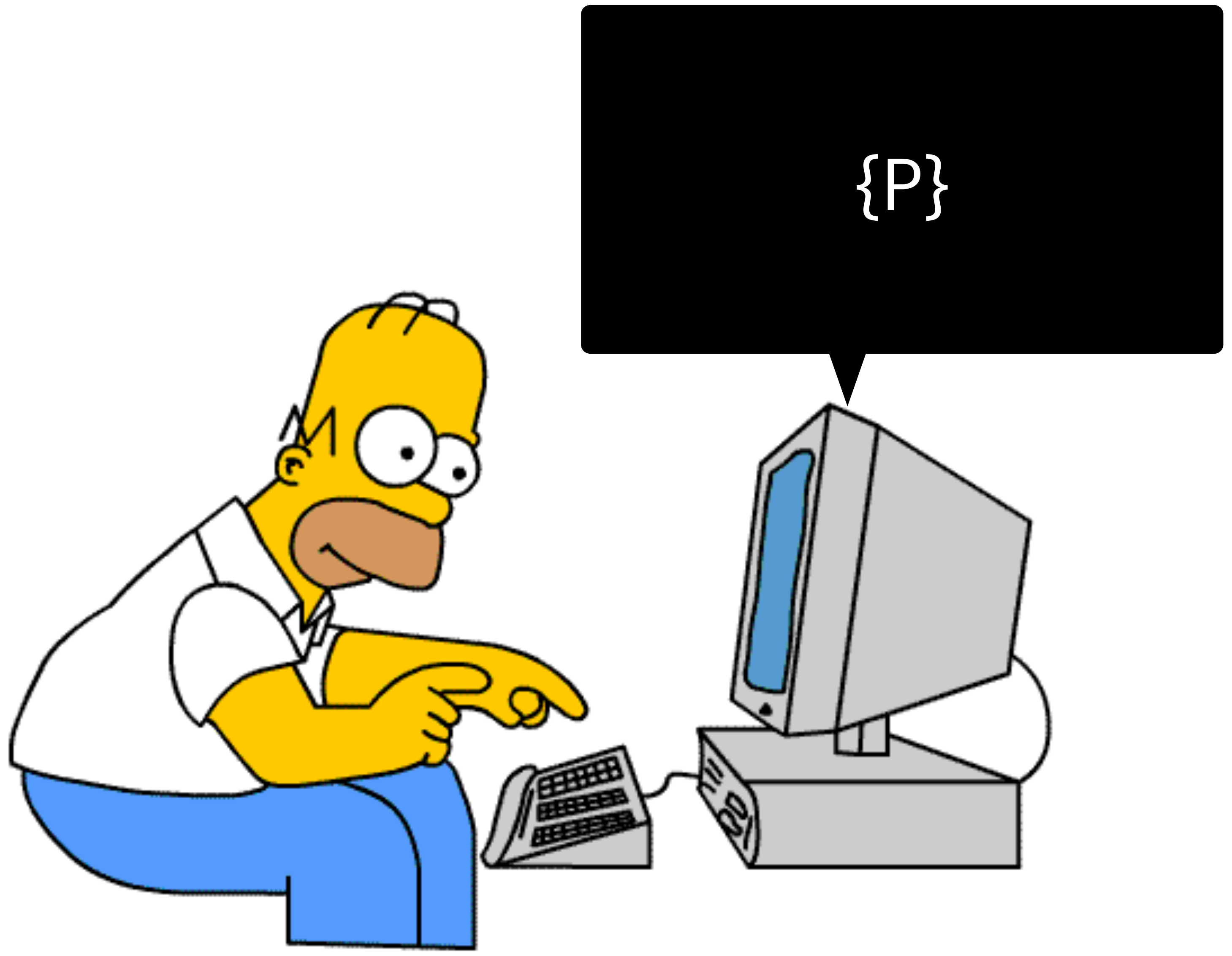
$$e ::= \ldots \mid \mu T.e \mid exec(e)$$

$$\tau ::= \ldots \mid p \text{ says } \tau \mid code\{\mu T.e\}$$

Principal p supports proposition τ

Computation Principal

Assume {P} = **μ**T.e



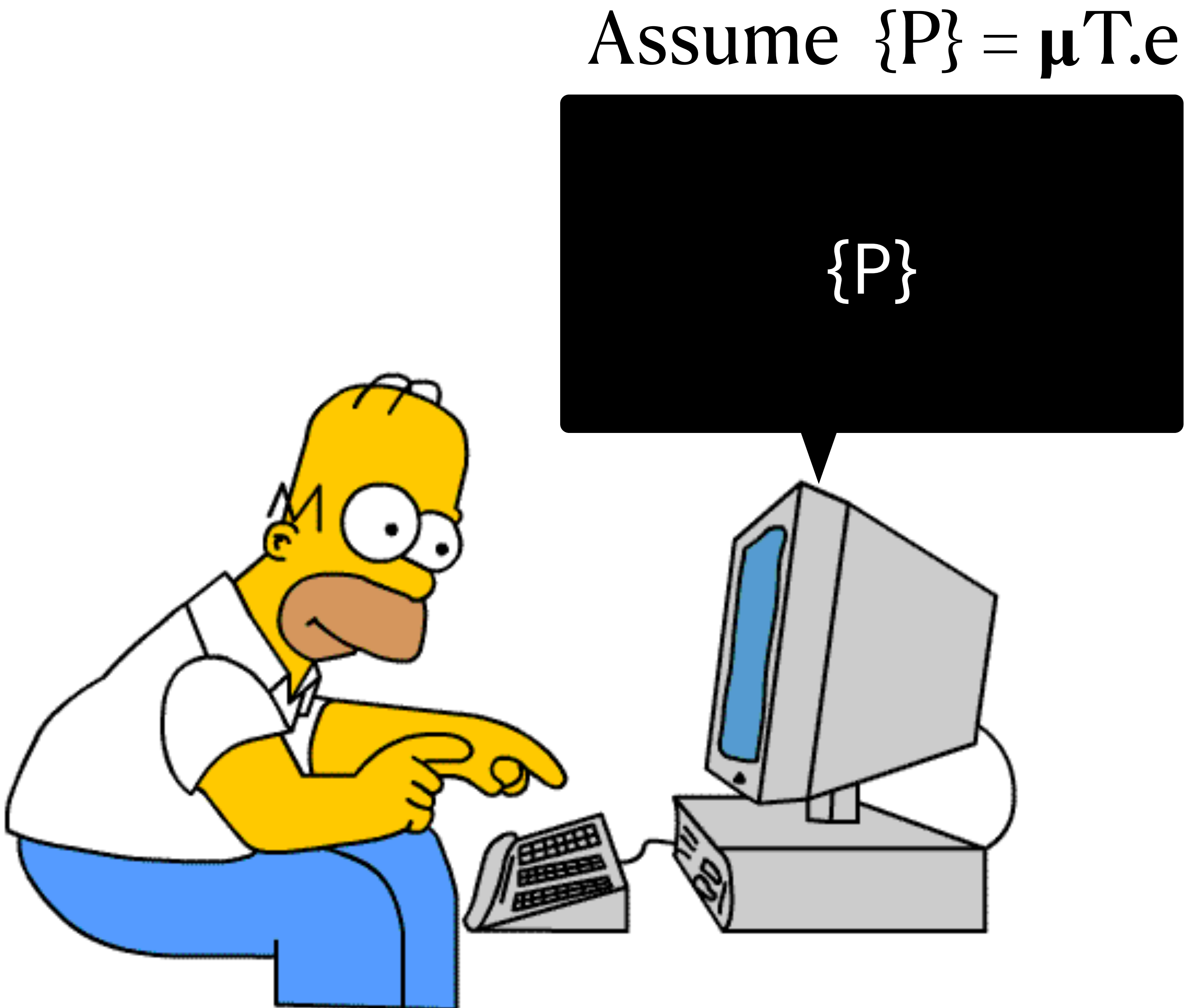
Homer trusts **{P}**

Specifying Trust in a Computation

Atomic Principal

$\forall X.\ \mathrm{code}\{\mu T.e\}\ \mathrm{says}\ X \rightarrow \mathrm{Homer\ says}\ X$

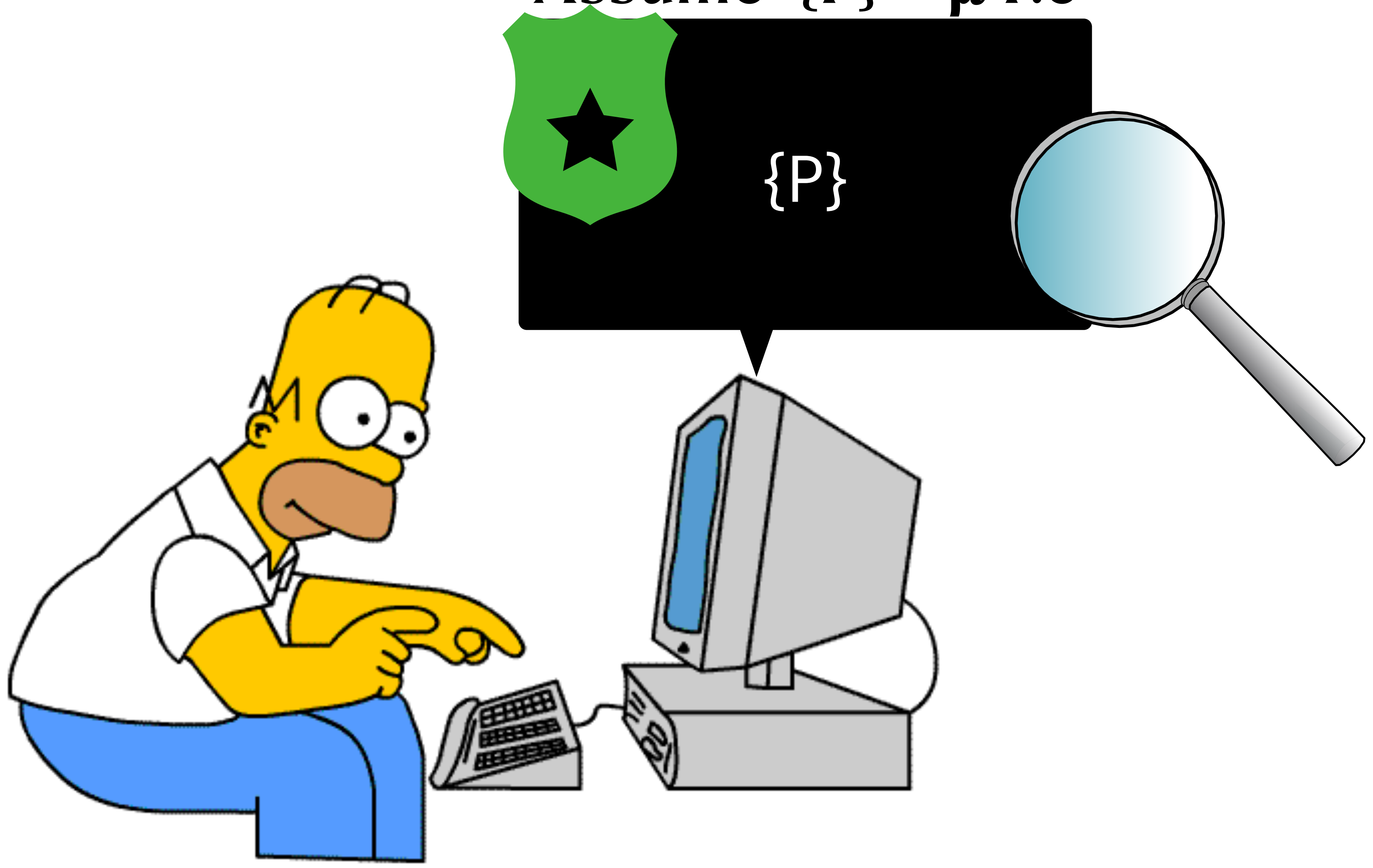
Computation Principal




Chain of Trust

Specifying Chain of Trust

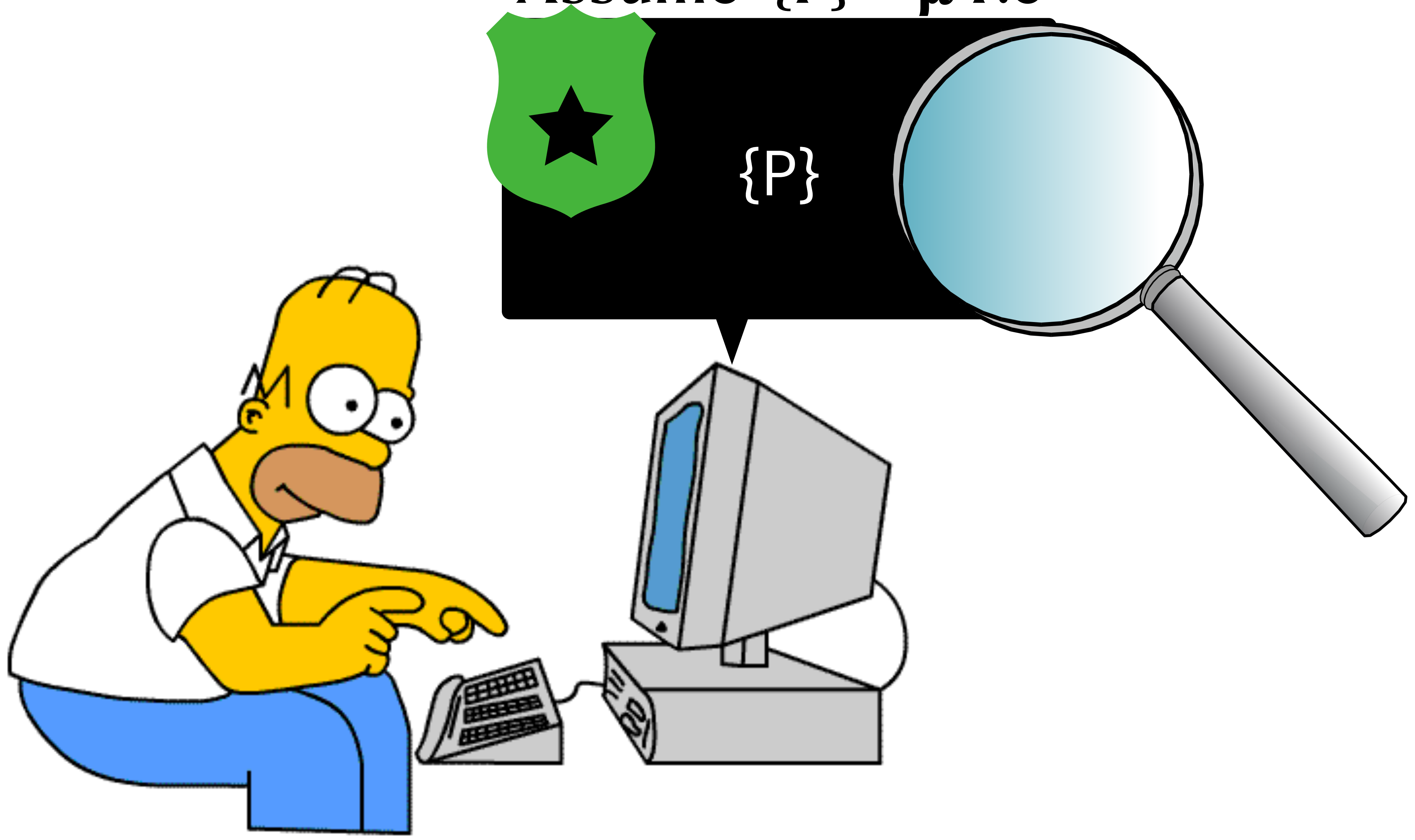
Assume {P} = **μ**T.e



Homer trusts **{P}** that is analyzed to be differentially private

Specifying Chain of Trust

Assume {P} = **μ**T.e



Homer trusts **{P}** that is analyzed to be differentially private

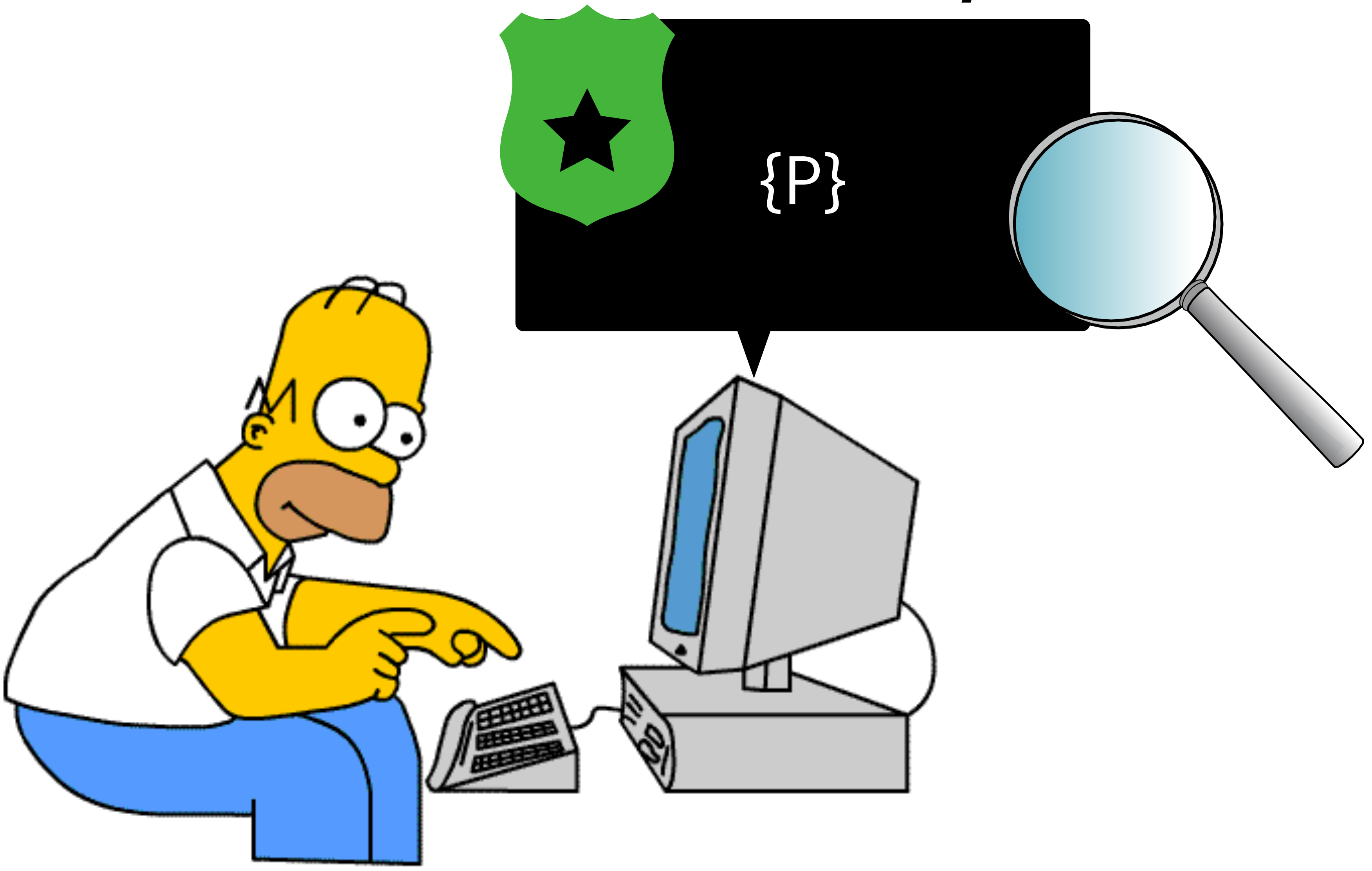
Specifying Trust Chain in a Computation

Atomic Principal

$\forall X.\ \text{code}\{\boldsymbol{\mu} T.e\}\ \text{says}\ X \rightarrow \text{Homer says}\ X$

Computation Principal

Assume {P} = $\boldsymbol{\mu}$T.e



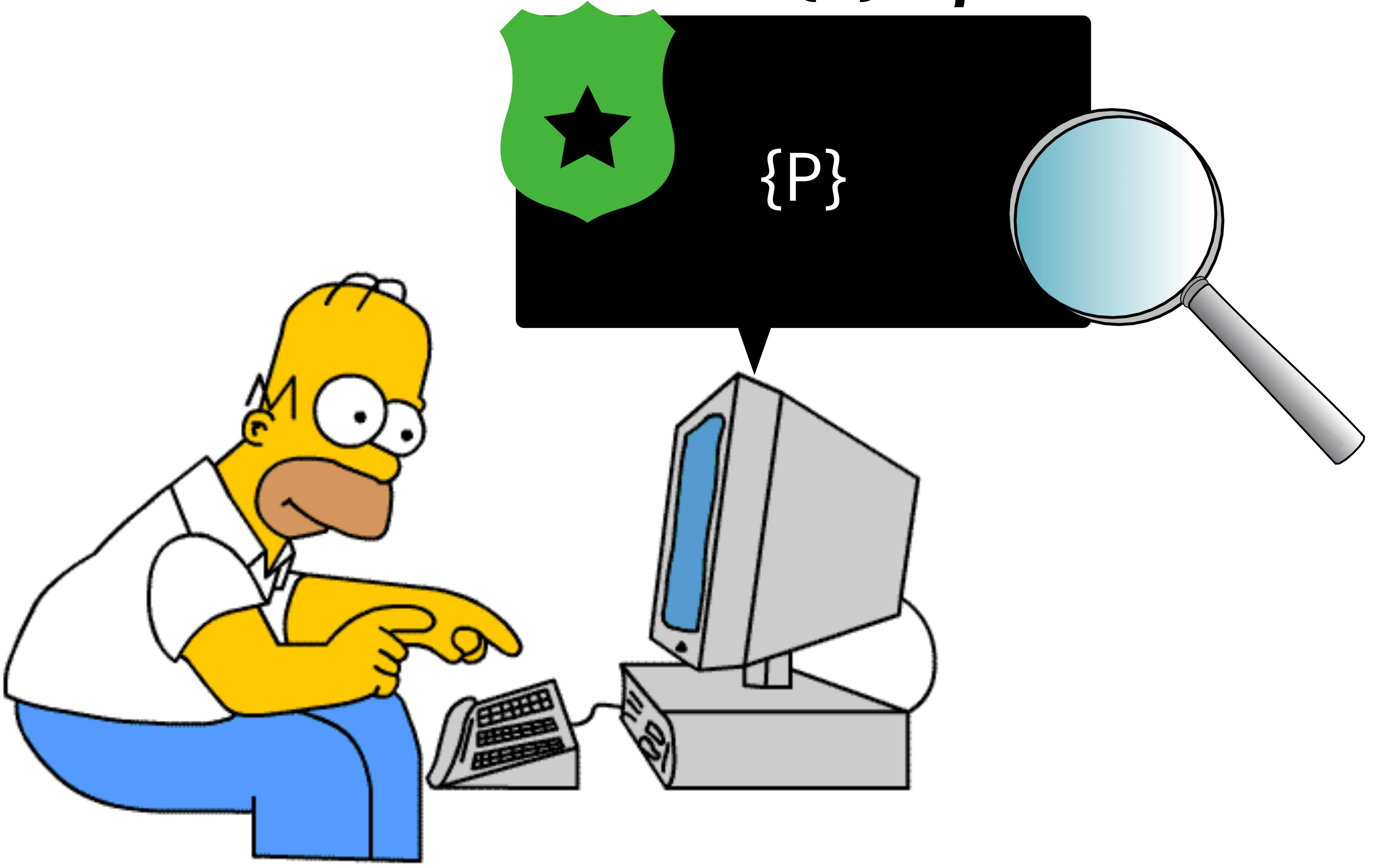
Specifying Trust Chain in a Computation

Atomic Principal

Code{DPAnalyzer} says (isDP μT.e) → ∀X. code{μT.e} says X → Homer says X

Computation Principal

Assume {P} = μT.e



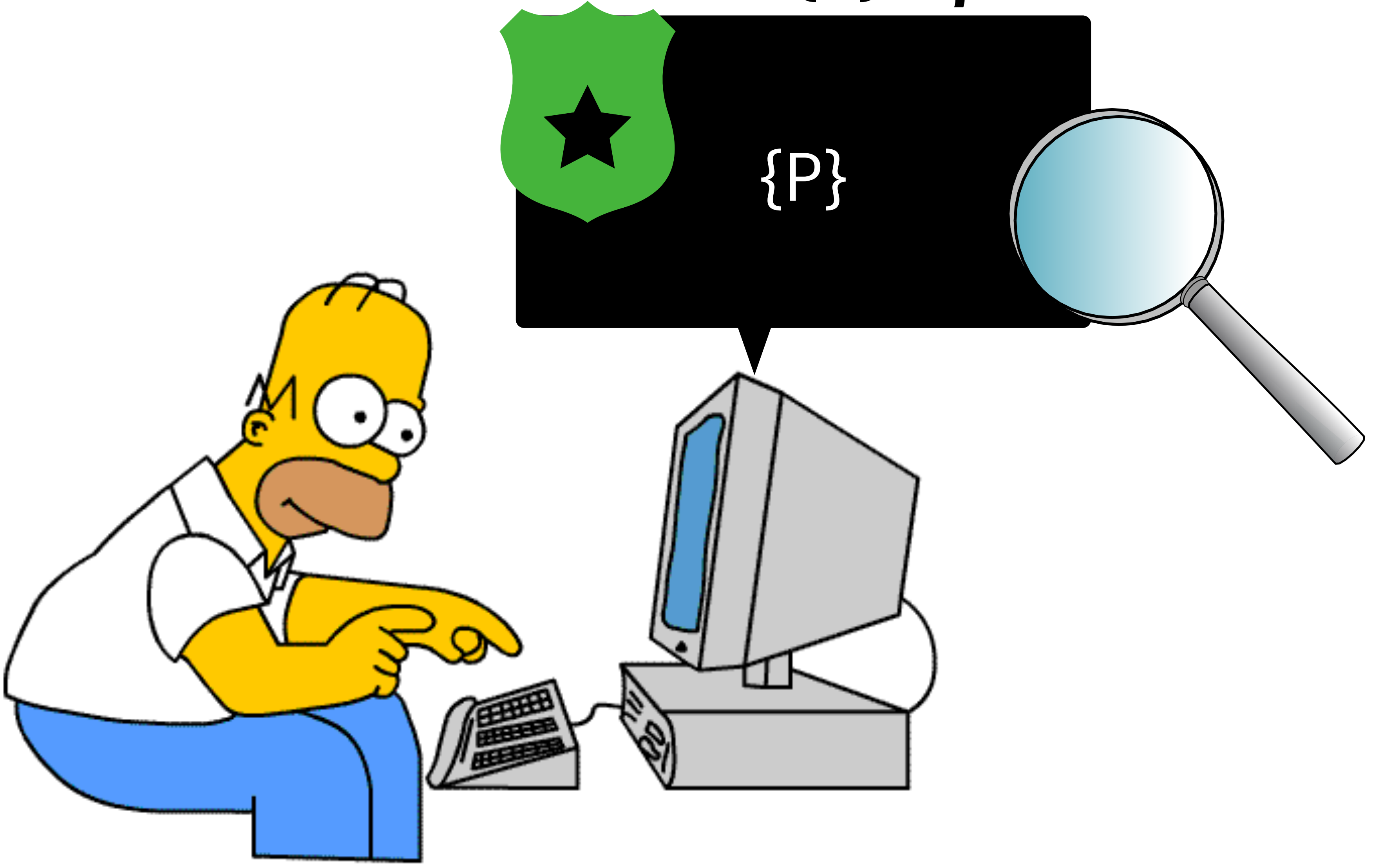
Specifying Trust Chain in a Computation

Atomic Principal

Code{DPAnalyzer} says (isDP μT.e) → ∀X. code{μT.e} says X → Homer says X

Computation Principal

Assume {P} = μT.e



Specifying Trust Chain in a Computation

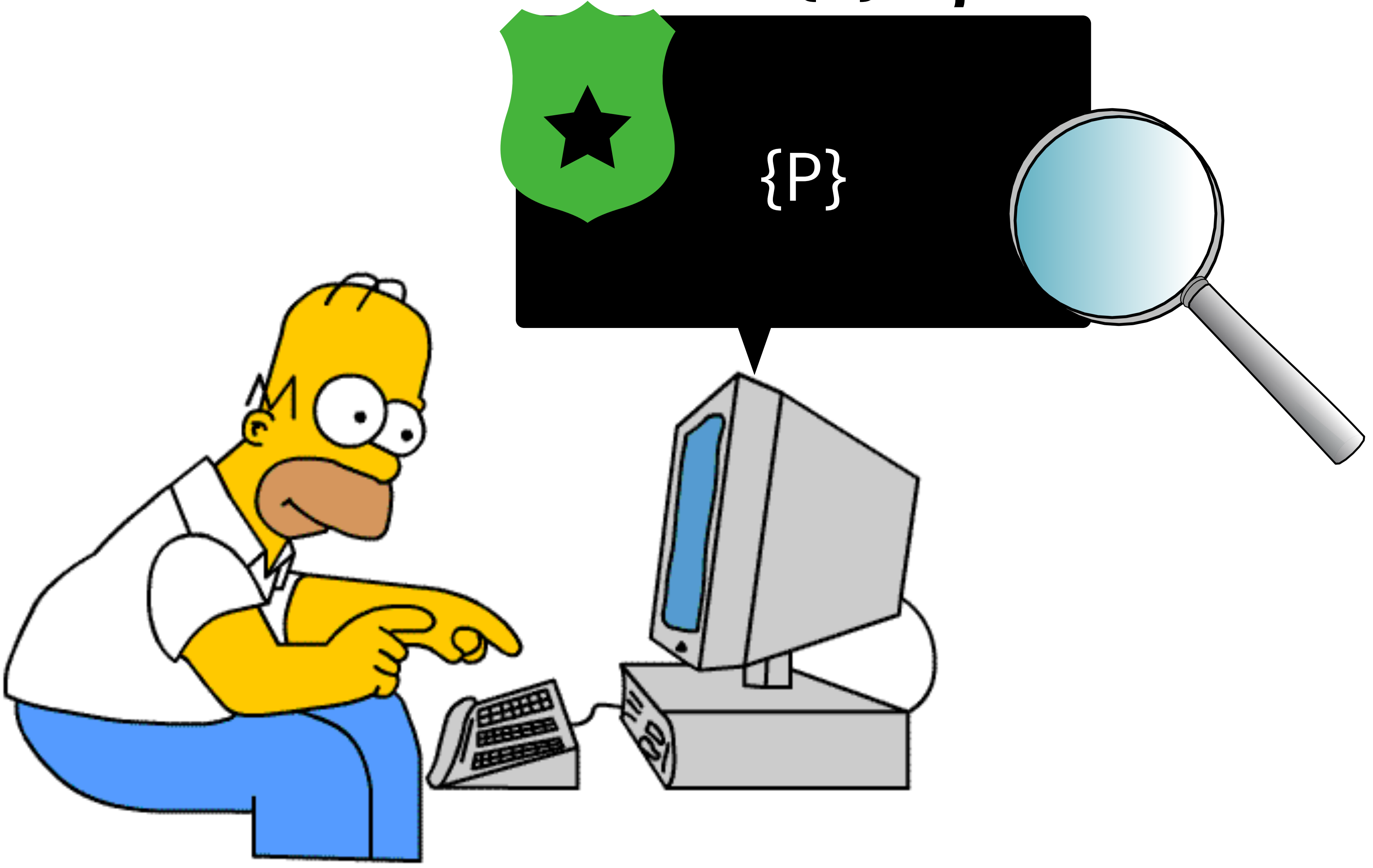
Predicate

Atomic Principal

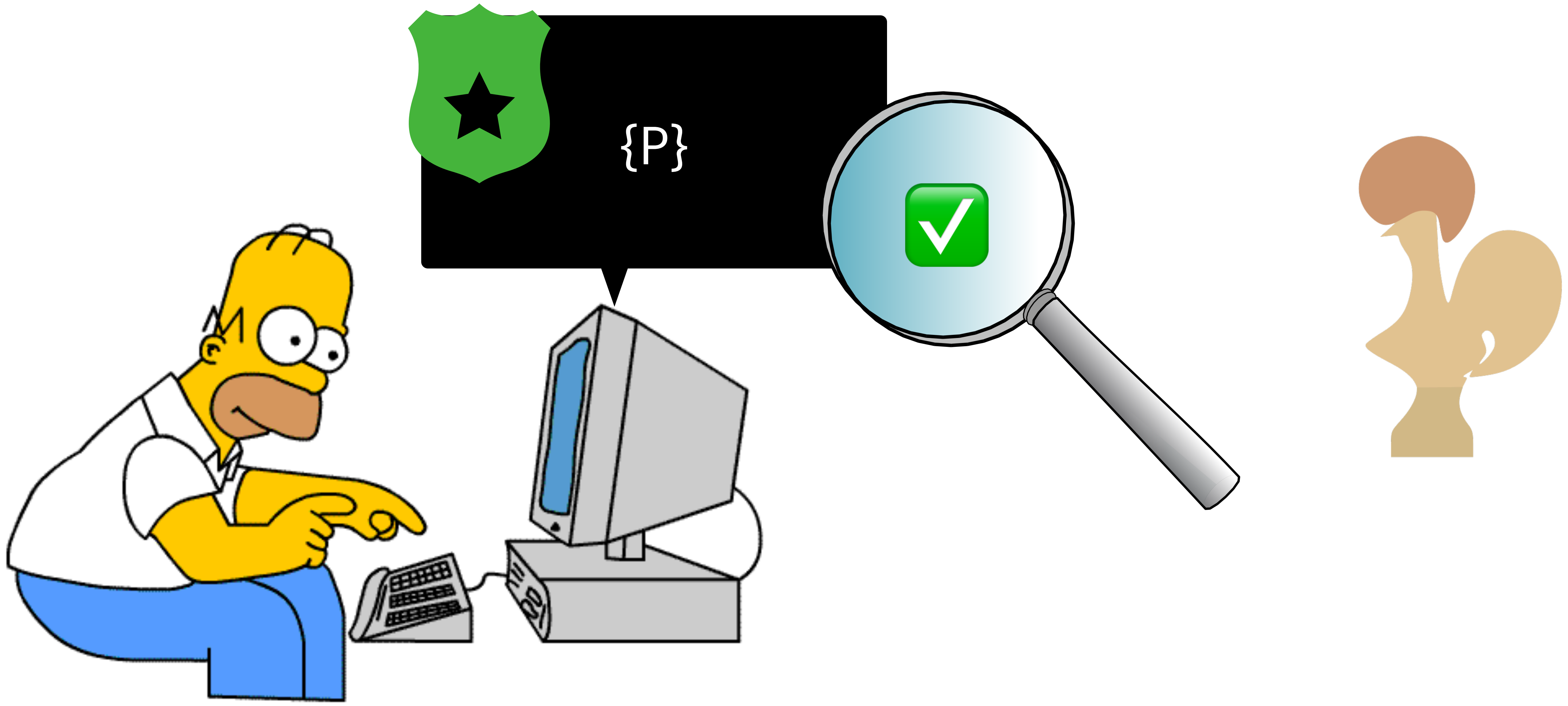
Code{DPAnalyzer} says (isDP μT.e) → ∀X. code{μT.e} says X → Homer says X

Computation Principal

Assume {P} = μT.e



Specifying Chain of Trust



Homer trusts **{P}** that is analyzed to be differentially private by a verified (differential privacy) analyzer

Specifying **Trust Chain** in a Computation

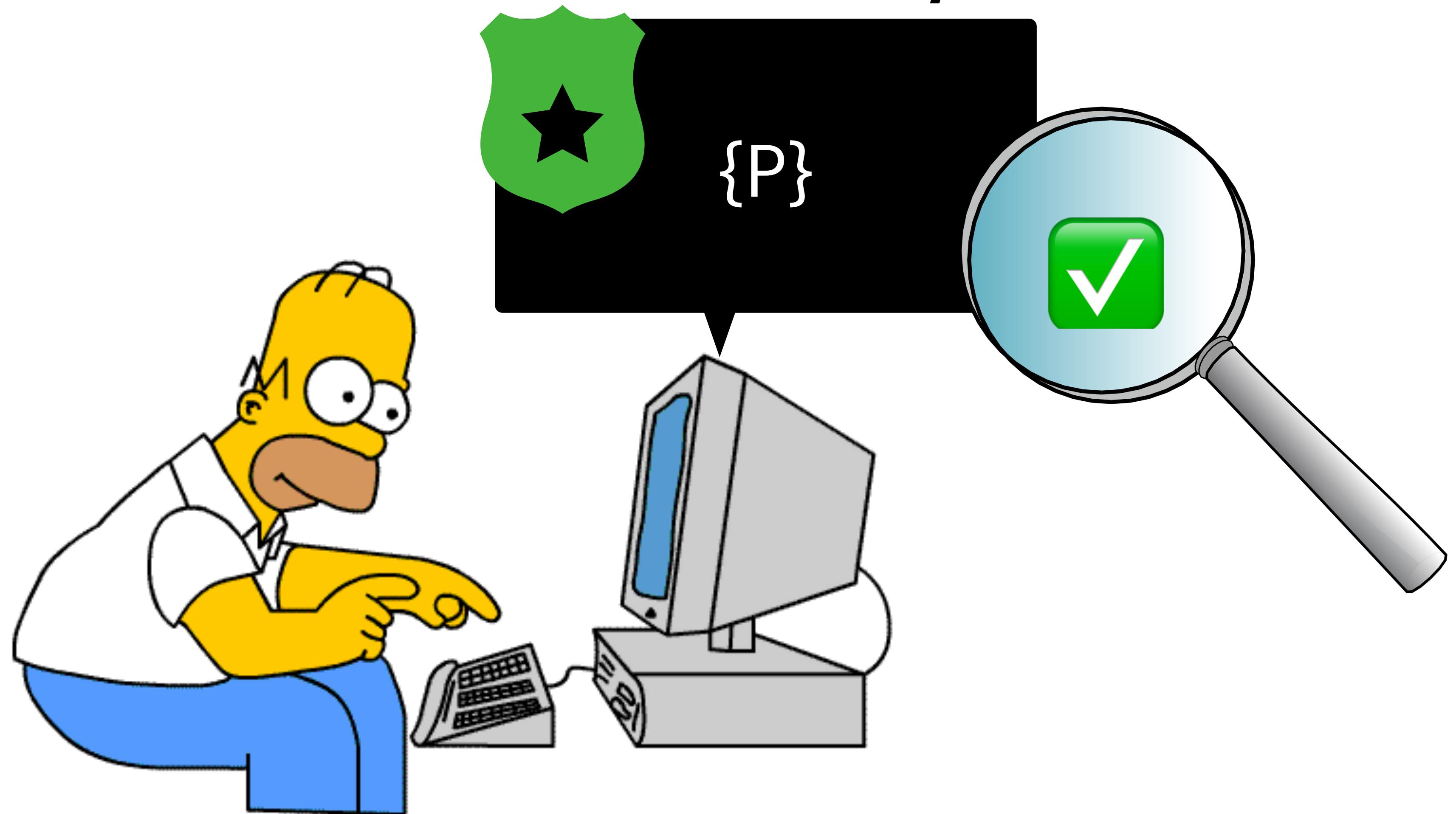
Predicate

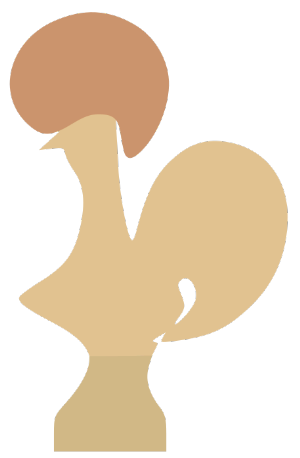
Atomic Principal

code{DPAnalyzer} says (isDP **μ**T.e) →∀X. code{**μ**T.e} says X → Homer says X

Computation Principal

Assume {P} = **μ**T.e





Specifying Trust Chain in a Computation

Predicate

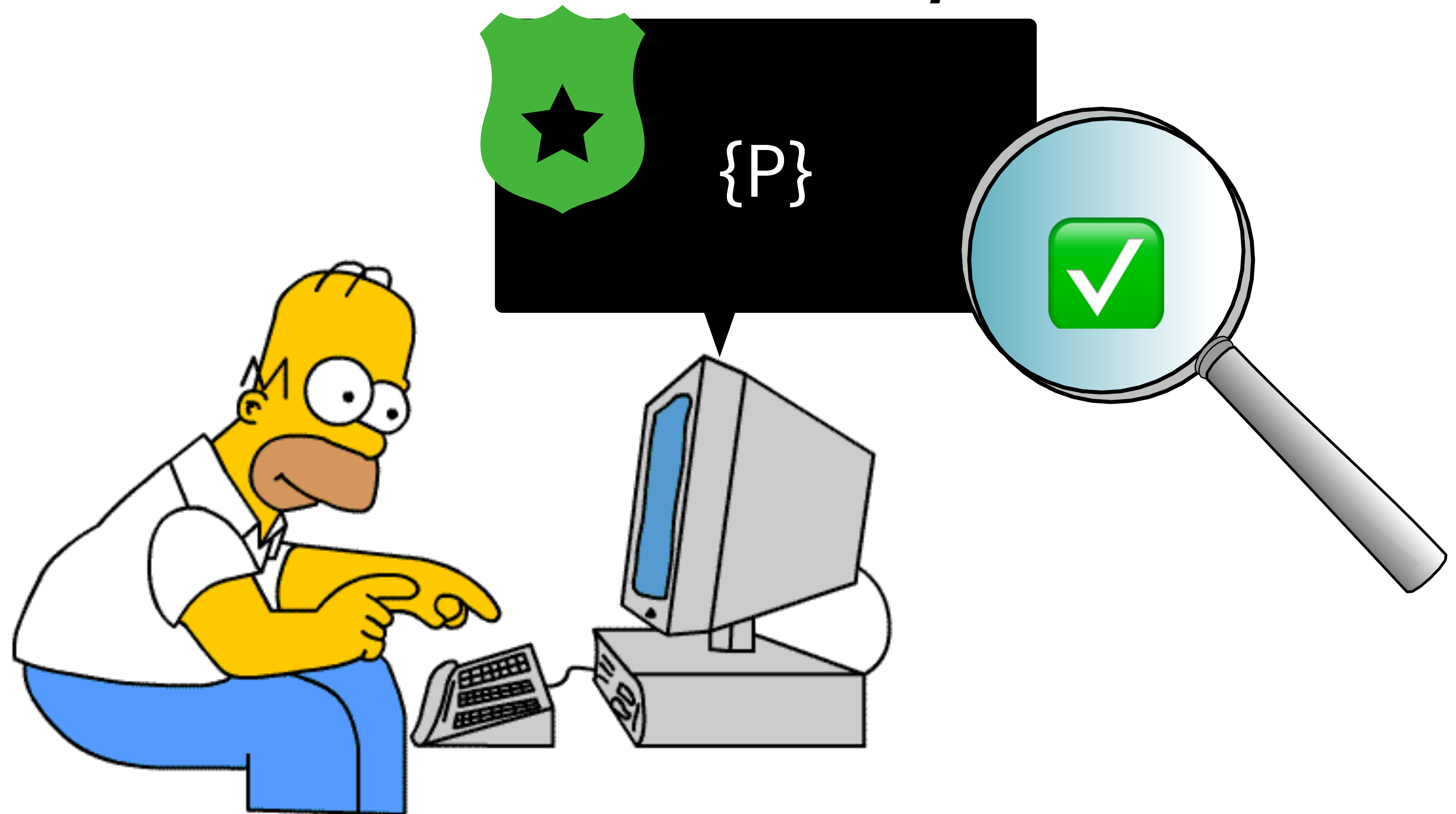
Atomic Principal

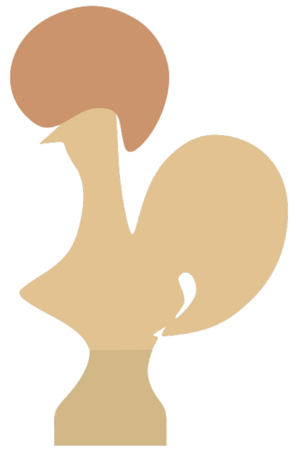
Coq says (✅ DPAnalyzer)

↳code{DPAnalyzer} says (isDP **μ**T.e) →∀X. code{**μ**T.e} says X → Homer says X

Computation Principal

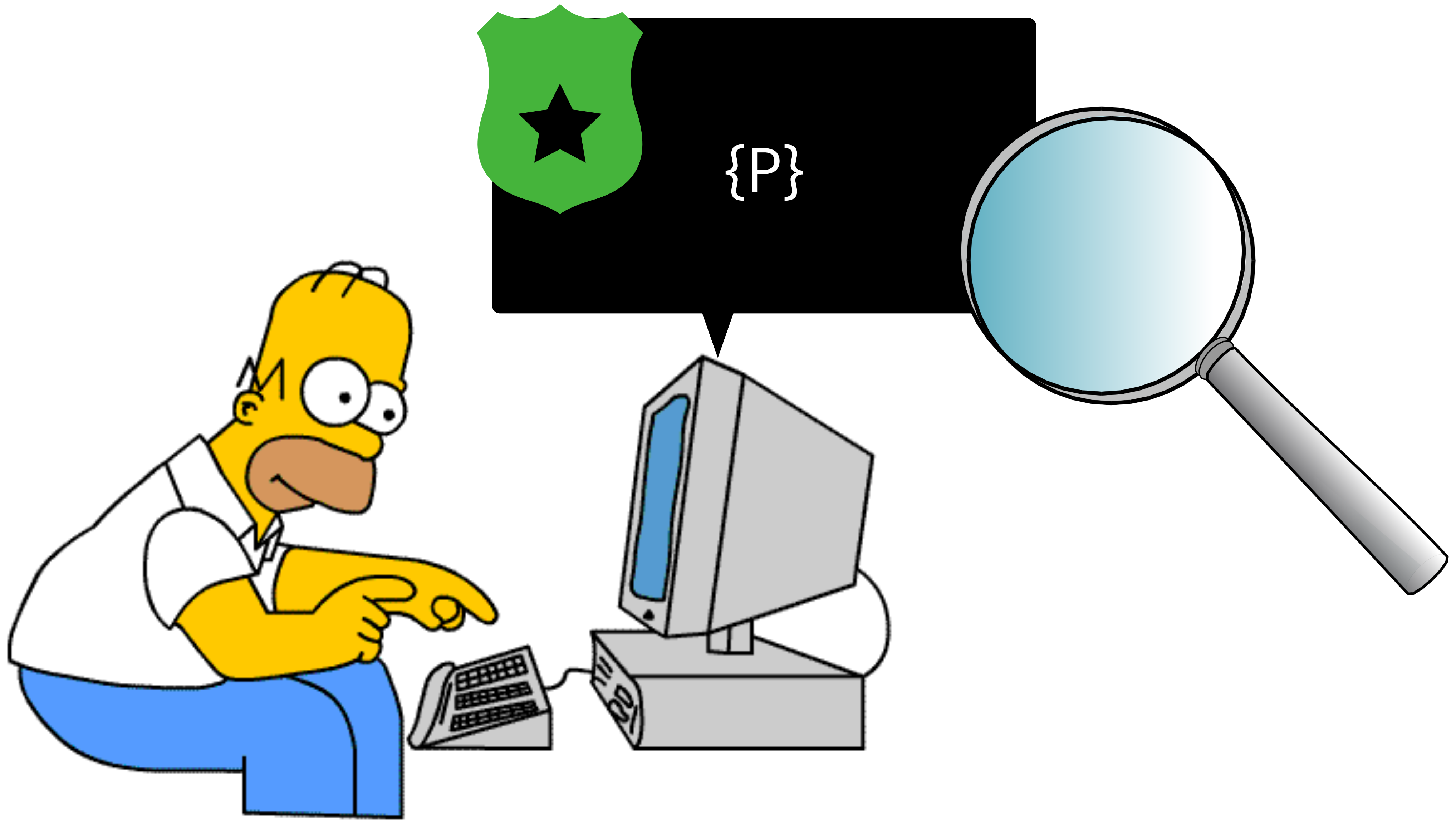
Assume {P} = **μ**T.e





Specifying Trust in Equivalent Computations

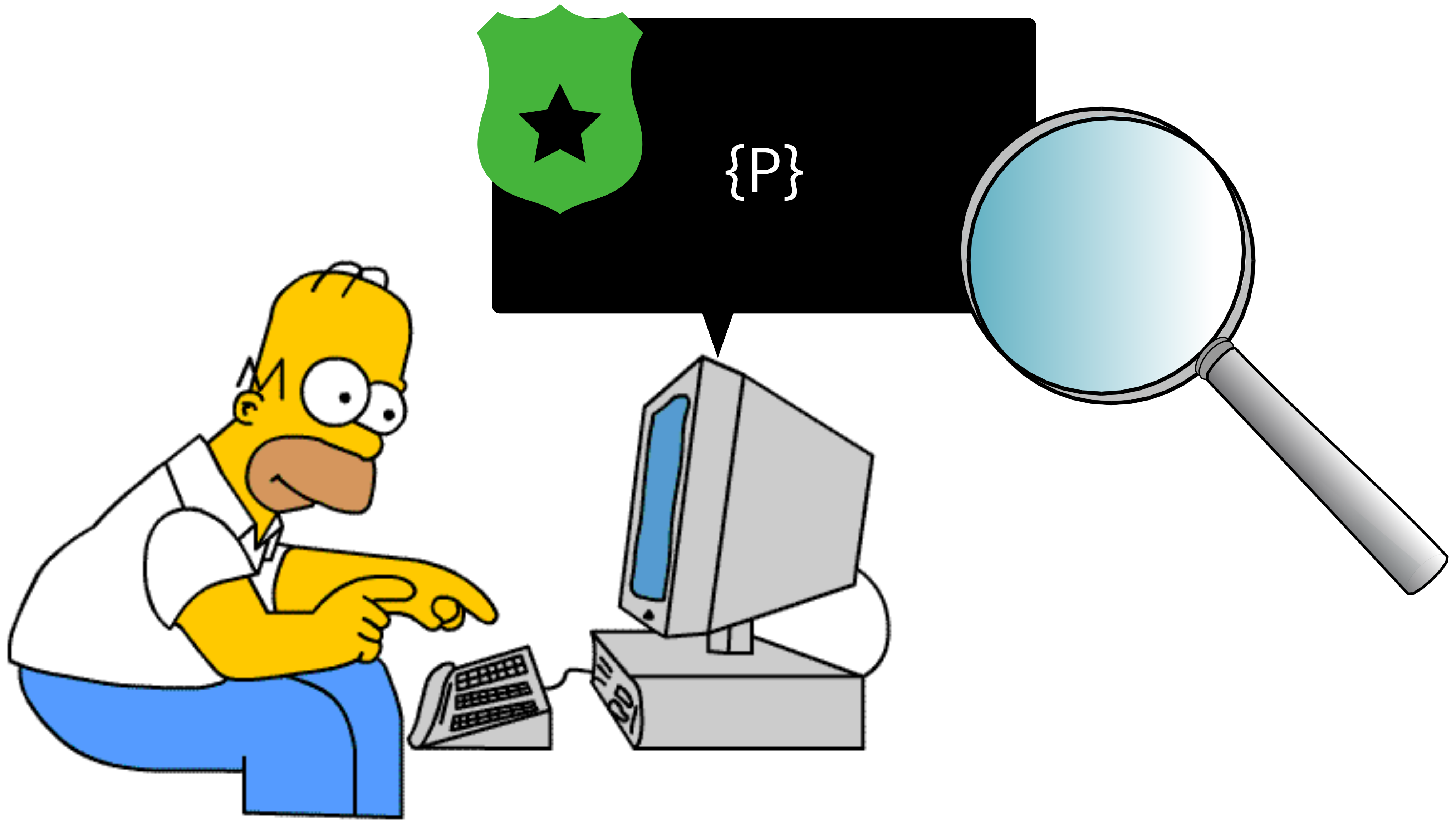
Assume {P} = **μ**T.e

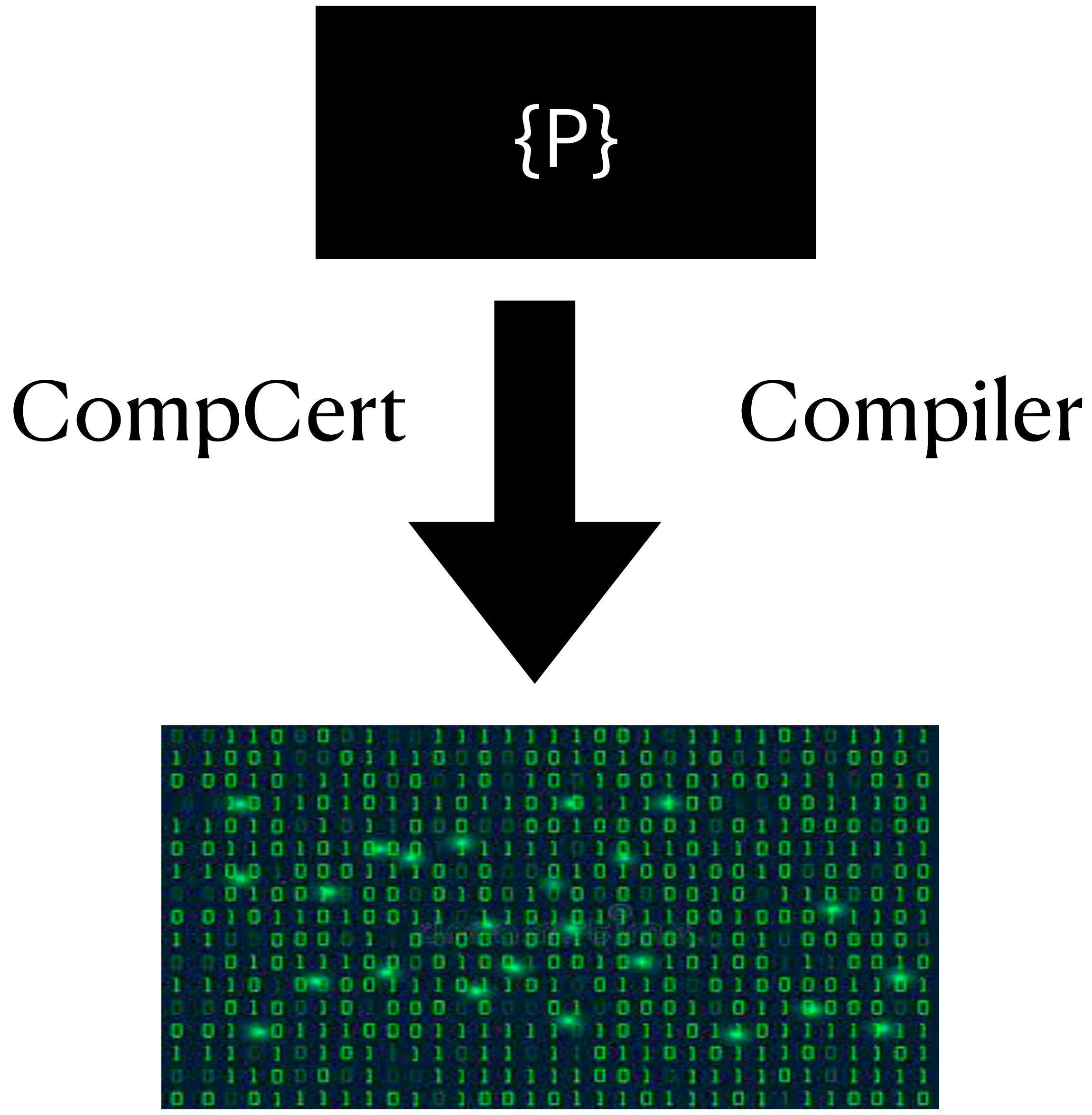


Homer trusts **{P}** that is analyzed to be differentially private

Specifying Trust in Equivalent Computations

Assume {P} = **μ**T.e

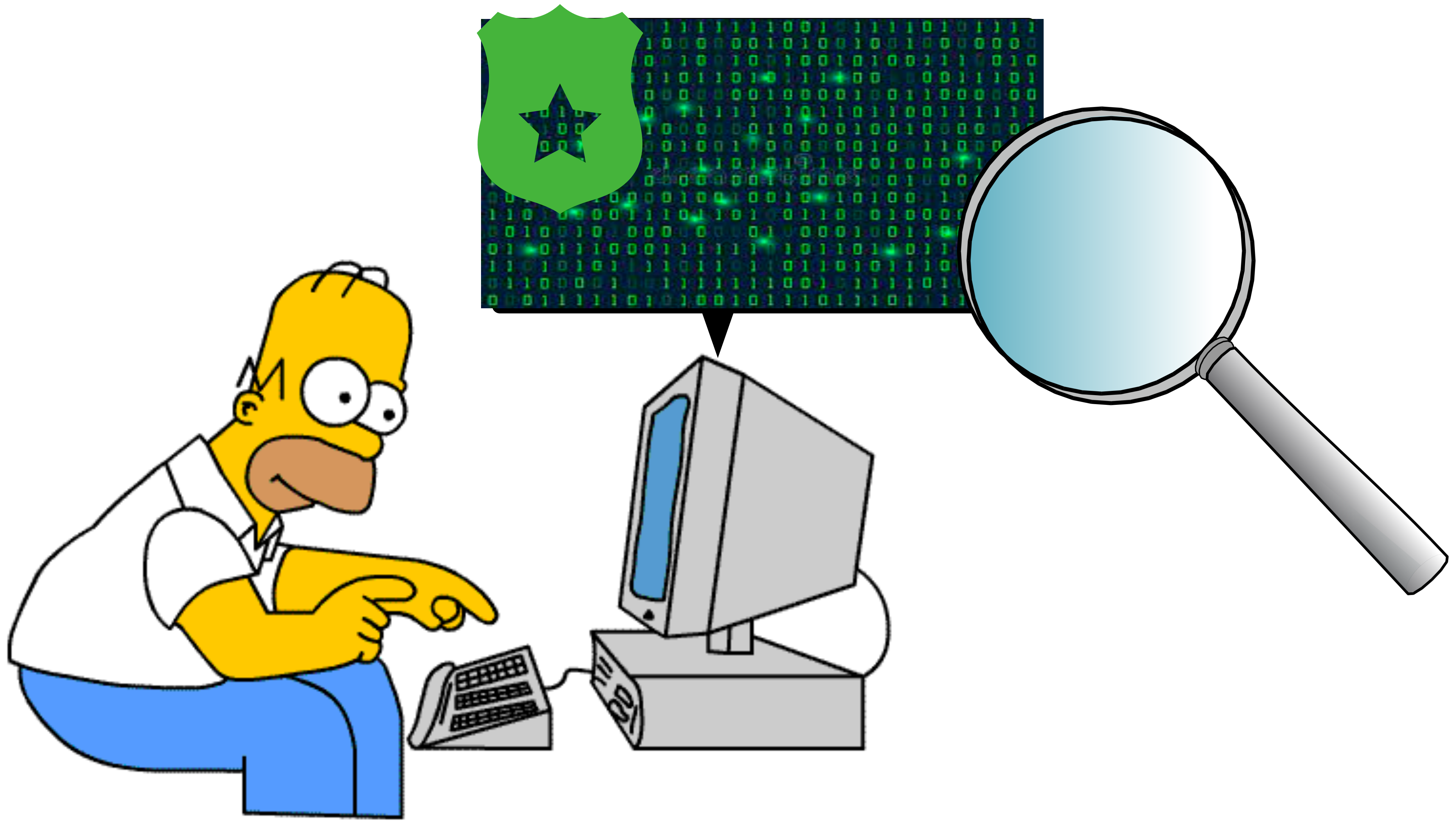


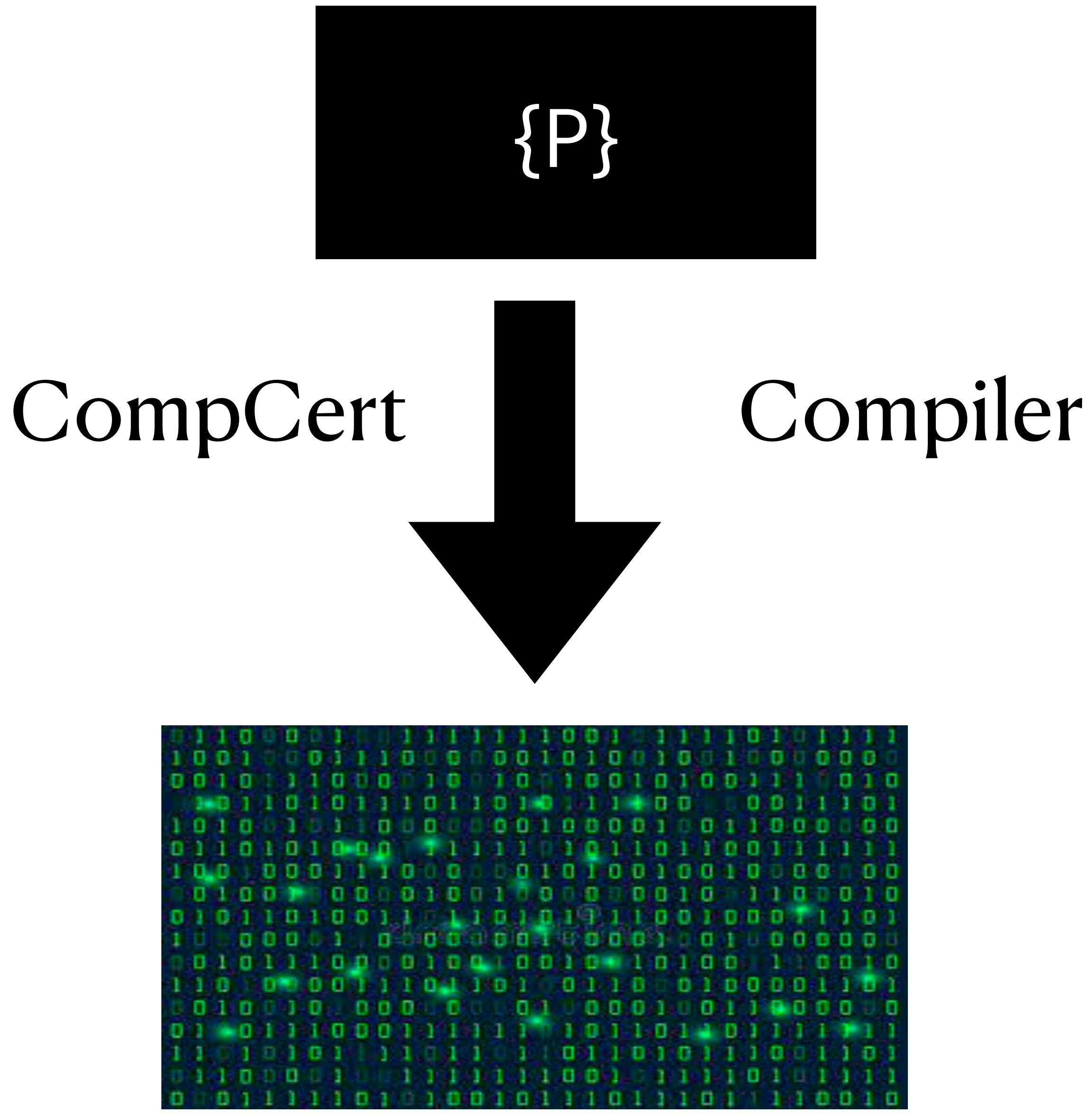


How to specify that Homer trusts compiled **{P}** ?

Specifying Trust in Equivalent Computations

Assume {P} = **μ**T.e





How to specify that Homer trusts compiled **{P}** ?

Type System, Briefly

Key features are to ensure that

- ✓ Computation principals are well-formed
- ✓ Proofs and computations are separate
  - Mixing proofs and computations is meaningless
- ✓ Decidable type inference
- ✓ Equivalent programs are treated as equivalent computation principals

Equivalent Computations

$$\frac{\Gamma \vdash e_1 \equiv e_2}{\Gamma \vdash \text{code}\{e_1\} \equiv \text{code}\{e_2\}}$$

Equivalent Computations

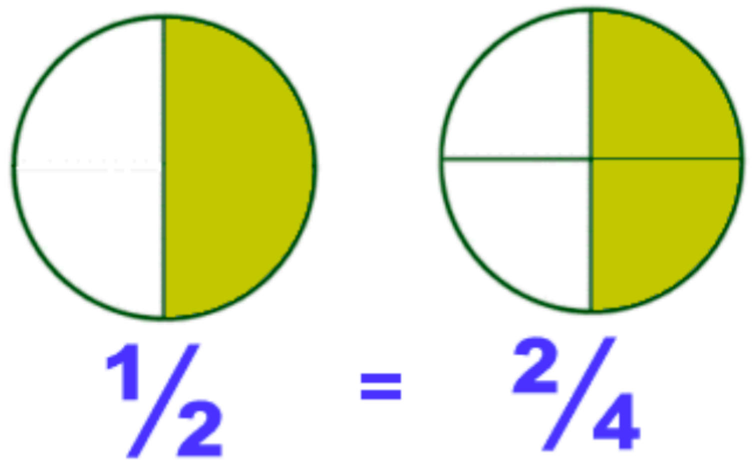
Equivalent Programs

$$\frac{\Gamma \vdash e_1 \equiv e_2}{\Gamma \vdash \text{code}\{e_1\} \equiv \text{code}\{e_2\}}$$

Equivalent Computations

Equivalent Programs

$$\frac{\Gamma \vdash e_1 \equiv e_2}{\Gamma \vdash \mathrm{code}\{e_1\} \equiv \mathrm{code}\{e_2\}}$$



Equivalent computations are treated as equivalent principals

Specifying Trust in Equivalent Computations

code{DPAnalyzer} says (isDP **µ**T.e) →∀X. code{**µ**T.e} says X → Homer says X

Specifying Trust in Equivalent Computations

$$[\![\boldsymbol{\mu} \mathrm{T.e}]\!] = e'$$

code{DPAnalyzer} says (isDP **μ**T.e) →∀X. code{**μ**T.e} says X → Homer says X

Specifying Trust in Equivalent Computations

Secure Compilation

$$[\![\mu T.e]\!] = e' \Rightarrow \mu T.e \equiv e'$$

code{DPAnalyzer} says (isDP μT.e) → ∀X. code{μT.e} says X → Homer says X

Specifying Trust in Equivalent Computations

Secure Compilation

$$\frac{[\![\boldsymbol{\mu}\mathrm{T.e}]\!] = \mathrm{e}' \Rightarrow \boldsymbol{\mu}\mathrm{T.e} \equiv \mathrm{e}'}{\mathrm{code}\{\boldsymbol{\mu}\mathrm{T.e}\} \equiv \mathrm{code}\{\mathrm{e}'\}}$$

Equivalent Principals

code{DPAnalyzer} says (isDP $\boldsymbol{\mu}$T.e) $\rightarrow \forall$X. code{$\boldsymbol{\mu}$T.e} says X $\rightarrow$ Homer says X

Specifying Trust in Equivalent Computations

Secure Compilation

Equivalent Principals

$$\frac{[\![\mu T.e]\!] = e' \Rightarrow \mu T.e \equiv e'}{\text{code}\{\mu T.e\} \equiv \text{code}\{e'\}}$$

$$\text{code}\{\text{DPAnalyzer}\} \text{ says } (\text{isDP } \mu T.e) \rightarrow \forall X.\ \text{code}\{\mu T.e\} \text{ says } X \rightarrow \text{Homer says } X$$

$$\equiv$$

$$\text{code}\{\text{DPAnalyzer}\} \text{ says } (\text{isDP } \mu T.e) \rightarrow \forall X.\ \text{code}\{e'\} \text{ says } X \rightarrow \text{Homer says } X$$

Specifying Trust in Equivalent Computations

Secure Compilation

$$\llbracket \boldsymbol{\mu}T.e \rrbracket = e' \Rightarrow \boldsymbol{\mu}T.e \equiv e'$$

Equivalent Principals

$$\frac{\llbracket \boldsymbol{\mu}T.e \rrbracket = e' \Rightarrow \boldsymbol{\mu}T.e \equiv e'}{\text{code}\{\boldsymbol{\mu}T.e\} \equiv \text{code}\{e'\}}$$

code{DPAnalyzer} says (isDP $\boldsymbol{\mu}$T.e) → ∀X. code{$\boldsymbol{\mu}$T.e} says X → Homer says X

≡

code{DPAnalyzer} says (isDP $\boldsymbol{\mu}$T.e) → ∀X. code{e'} says X → Homer says X

Source

Target

Coal: Next Steps

Coal: Next Steps

- Towards a real language: A secure programming language based on Coal
  - Realize Coal abstractions (e.g., Intel SGX as a computation principal)
  - Information-flow control guarantees
    - E.g. strong integrity guarantees for computation principals (they do not err)

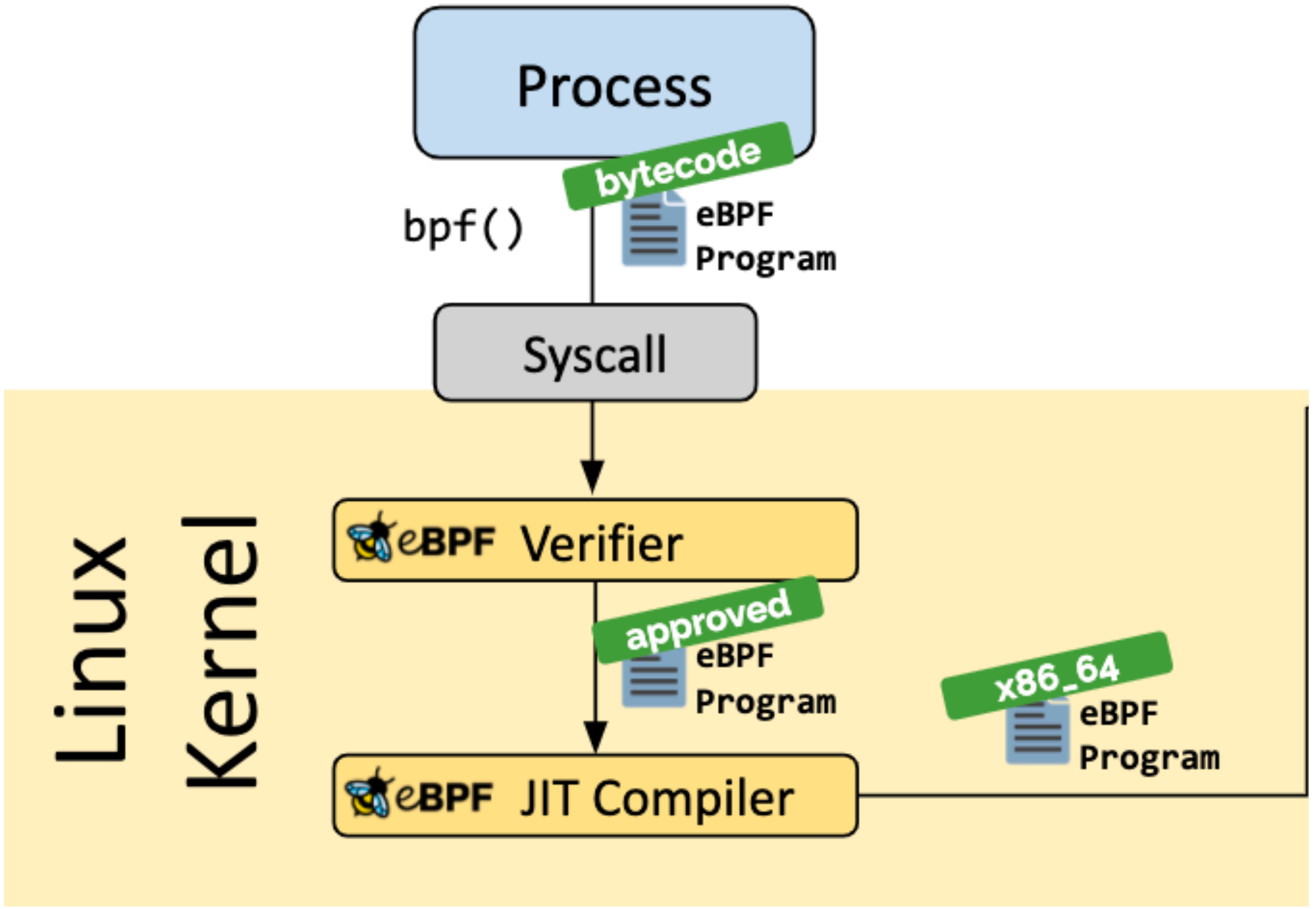
Coal: Next Steps

- ❖Towards a real language: A secure programming language based on Coal
  - Realize Coal abstractions (e.g., Intel SGX as a computation principal)
  - Information-flow control guarantees
    - E.g. strong integrity guarantees for computation principals (they do not err)
- ❖Explore various notions of program equivalence to get equivalent principals
  - Introduces functional dependent types
  - Type checking could be undecidable

Coal: Enables expressive authorization policies using **computation principals**

Backup

Case Study: eBPF Authorization



eBPF Authorization Policy using Coal

U terminates and is safe

Kernel says (∀U. Verifier says (terminates U ∧ safeSysCalls U)) → (U ⇒ Kernel)

Computation Principal

U *speaks for* Kernel

Specifying Trust Chain in Equivalent Computations

Coq says (✅ DPAnalyzer)

↳code{DPAnalyzer} says (isDP **µ**T.e)→ ∀X. code{**µ**T.e} says X→ Homer says X

Specifying Trust Chain in Equivalent Computations

Equivalent Principals

code{DPAnalyzer}≡code{ ⟦DPAnalyzer⟧ }

Coq says (✅ DPAnalyzer)

↳code{DPAnalyzer} says (isDP **μ**T.e)→ ∀X. code{**μ**T.e} says X→ Homer says X

Specifying Trust Chain in Equivalent Computations

Equivalent Principals

code{DPAnalyzer}≡code{ ⟦DPAnalyzer⟧ }

Coq says (✅ DPAnalyzer)

↳code{DPAnalyzer} says (isDP **μ**T.e)→ ∀X. code{**μ**T.e} says X→ Homer says X

⇓

Coq says (✅ DPAnalyzer)

↳code{ ⟦DPAnalyzer⟧ } says (isDP **μ**T.e)→∀X. code{**μ**T.e} says X→Homer says X