Automatic Enforcement of Expressive Security Policies using Enclaves

Anitha Gollamudi Stephen Chong
Harvard University

OOPSLA'16

Application-level Security

Application-level Security

```
x = secret_info
//compute with x
…
x = public_info
output x
```

Example Programs

```
x = key
//encrypt with x
encrypt(message, x)
x = 0
output x
```

Encryption

```
x = key
//sign with x
sign(message, x)
x = 0
output x
```

Signature

Application-level Security

```
x = secret_info
//compute with x
…
x = public_info
output x
```

Application-level Security

```
x = secret_info
//compute with x
…
x = public_info
output x
```

Application-level Security

```
x = secret_info
//compute with x
…
x = public_info
output x
```

Language-based Security

Application-level Security

```
x = secret_info
//compute with x
…
x = public_info
output x
```

Language-based Security

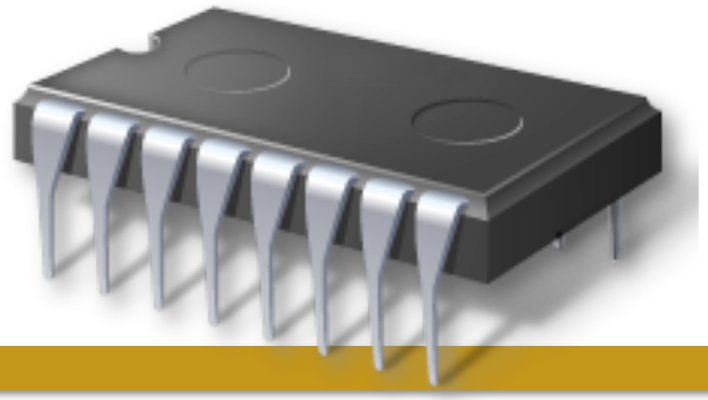
Real World Scenario: Application running a real machine

Application-level Security

```
x = secret_info
//compute with x
…
x = public_info
output x
```

Language-based Security

Operating System

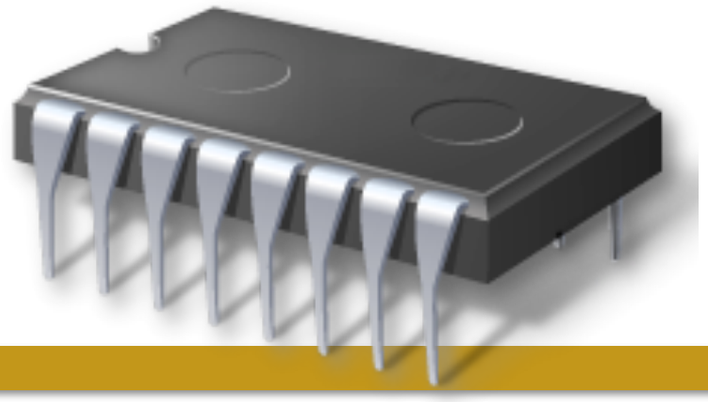


Application-level Security

```
x = secret_info
//compute with x
…
x = public_info
output x
```

Language-based Security

Operating System

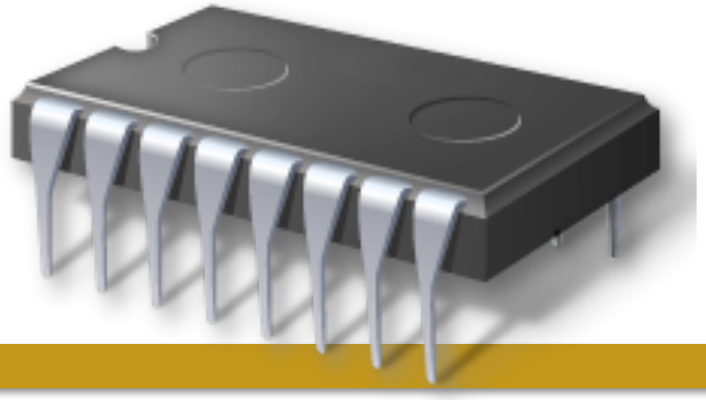


Application-level Security

```
x = secret_info
//compute with x
…
x = public_info
output x
```

Language-based Security

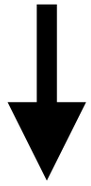
Operating System



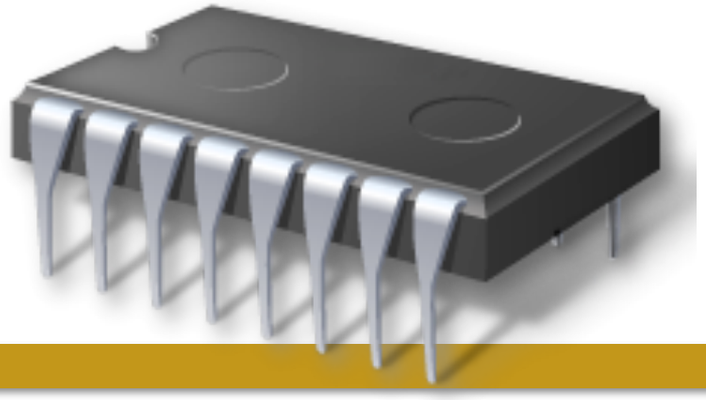
Application-level Security

```
x = secret_info
//compute with x
…
x = public_info
output x
```

Language-based Security



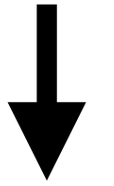
Operating System



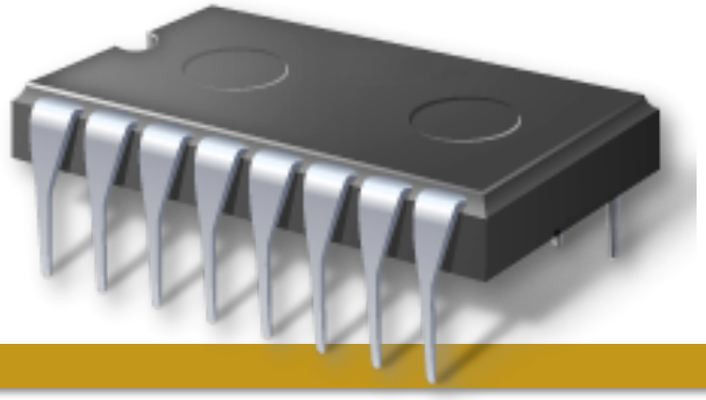
Application-level Security

```
x = secret_info
//compute with x
…
x = public_info
output x
```

Language-based Security



Operating System



Application-level Security

```
x = secret_info
//compute with x
…
x = public_info
output x
```

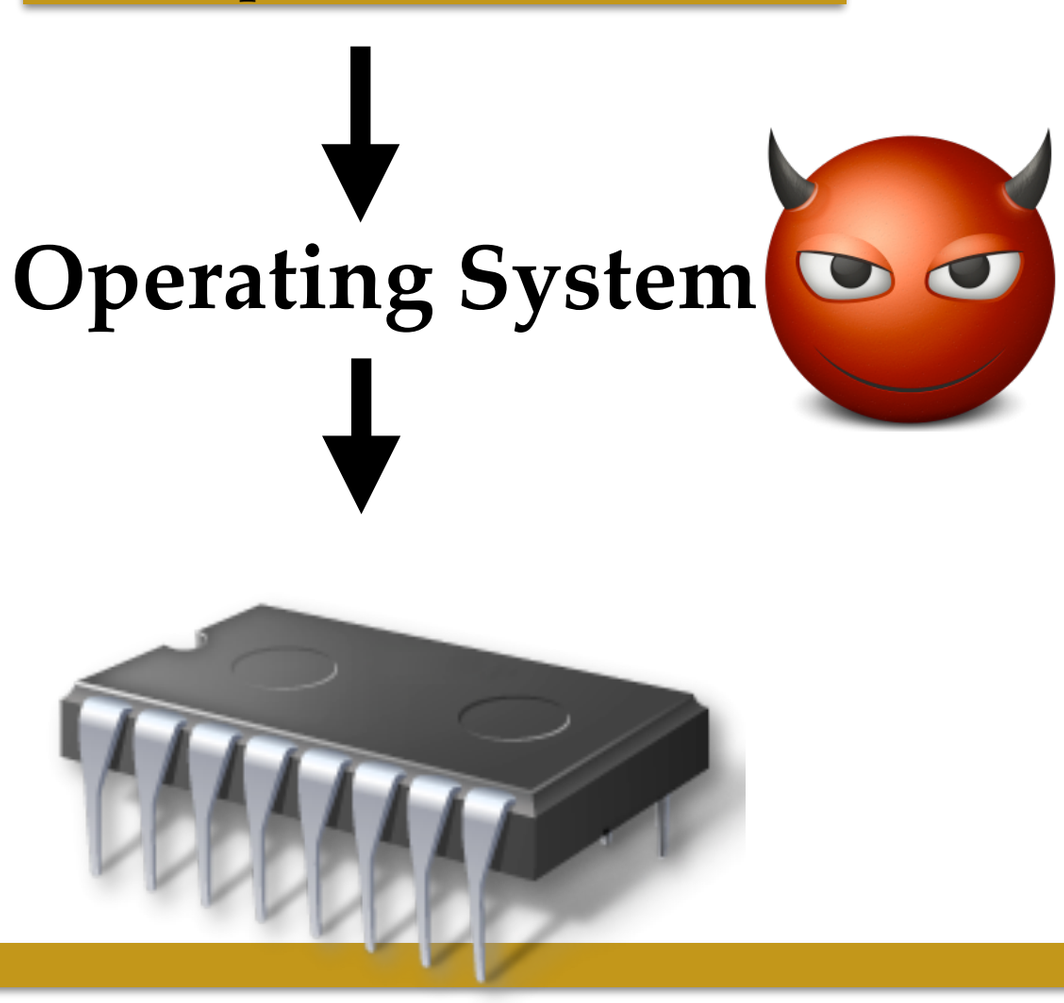
Language-based Security

Operating System

Question: How to enforce application security guarantees against low-level attackers?

Application-level Security

```
x = secret_info
//compute with x
…
x = public_info
output x
```

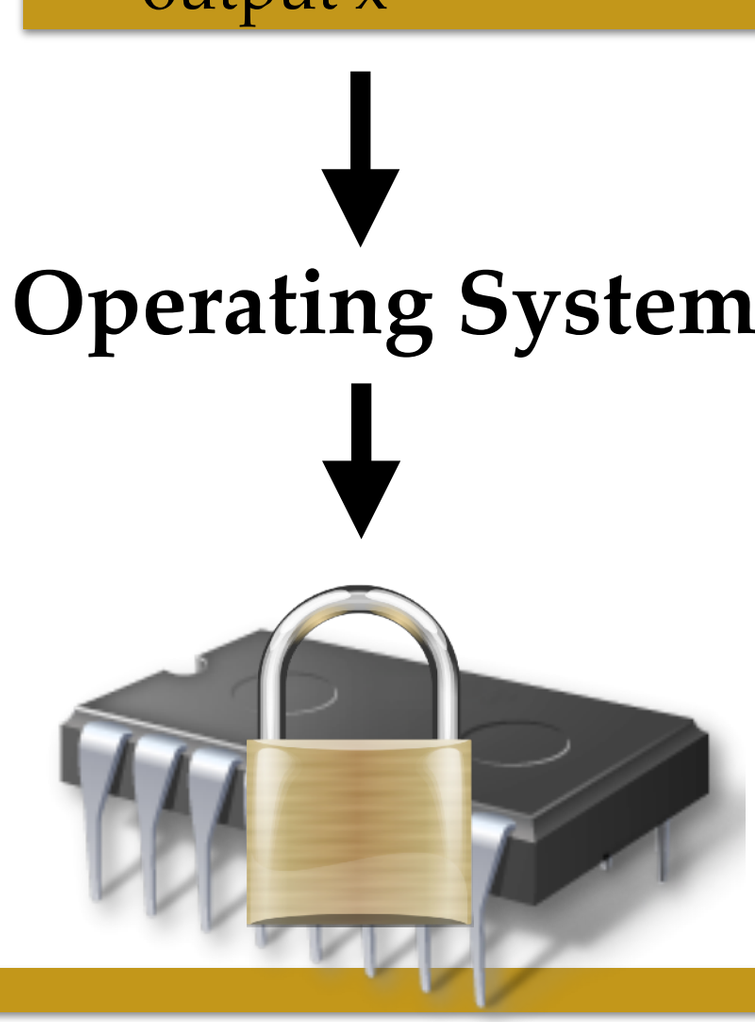


Our Solution

- Extend the Language-based Security with hardware protection mechanisms (Intel SGX, ARM TrustZone)
- Enforce security against low-level attackers

Hardware Protection Mechanisms

```
x = secret_info
//compute with x
…
x = public_info
output x
```

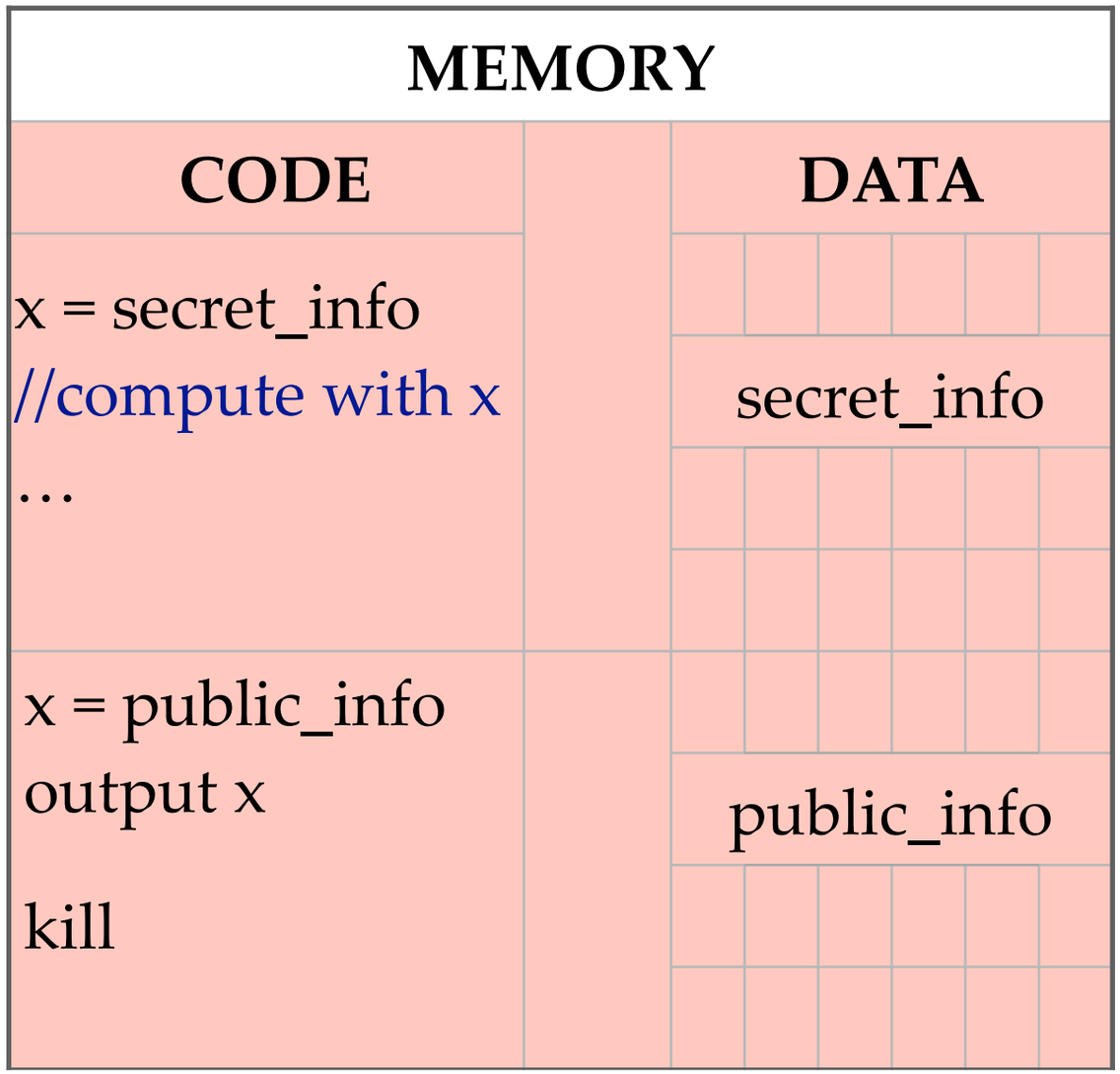


- Intel SGX enables
  - Applications to build enclaves: protected memory containers
    - Isolated execution
    - Restricted access
- ARM TrustZone

Overview of Enclaves

```
x = secret_info
//compute with x
 …

x = public_info
output x
```

Program

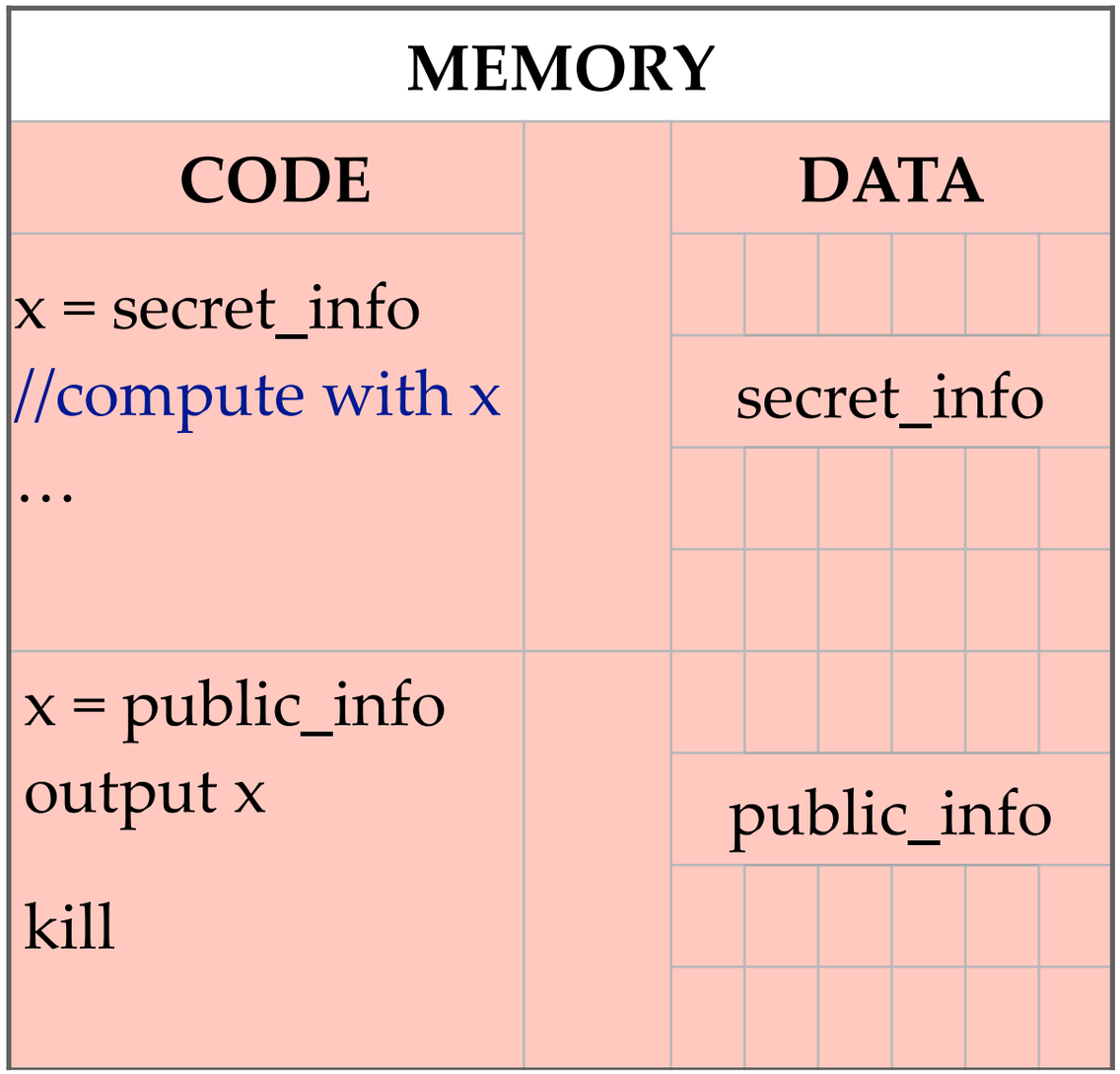


Execution

Overview of Enclaves

```
enclave {
  x = secret_info
  //compute with x
  …

}
x = public_info
output x

kill
```

Program

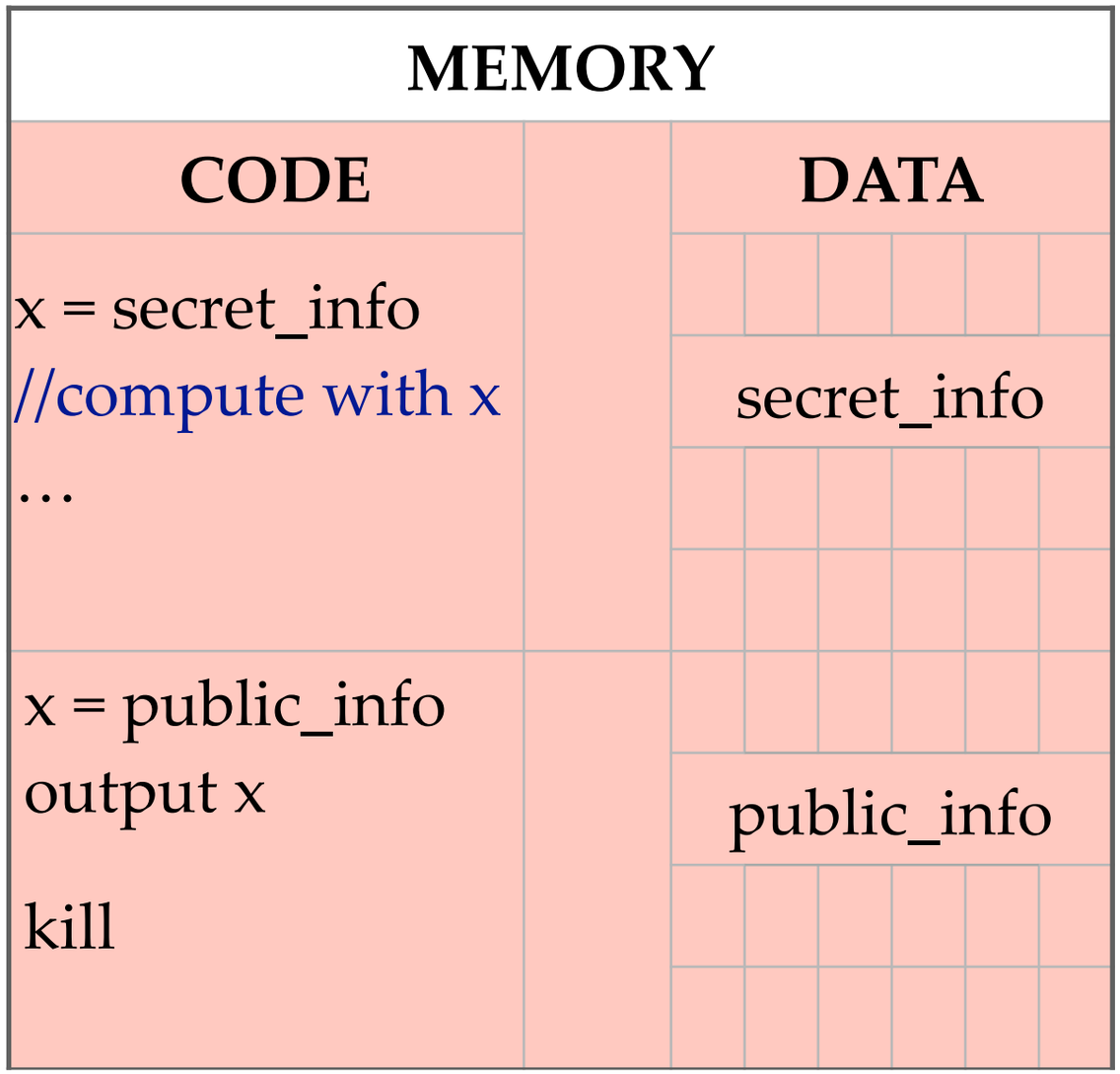


Execution

Overview of Enclaves

```
enclave {
  x = secret_info
  //compute with x
  …

}
x = public_info
output x

kill
```

Program

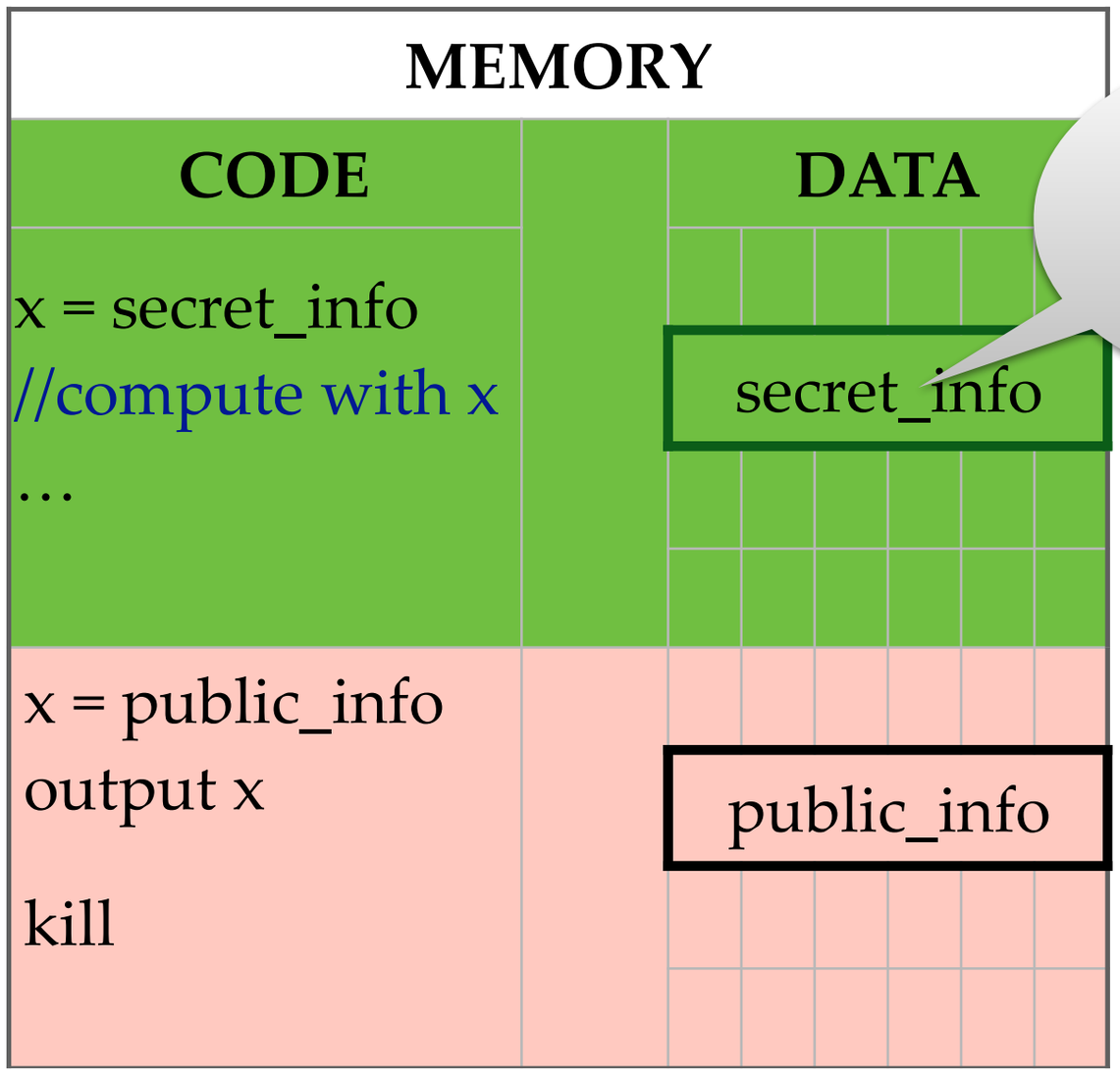


Execution

Overview of Enclaves

```
enclave {
  x = secret_info
  //compute with x
  …

}
x = public_info
output x

kill
```

Program

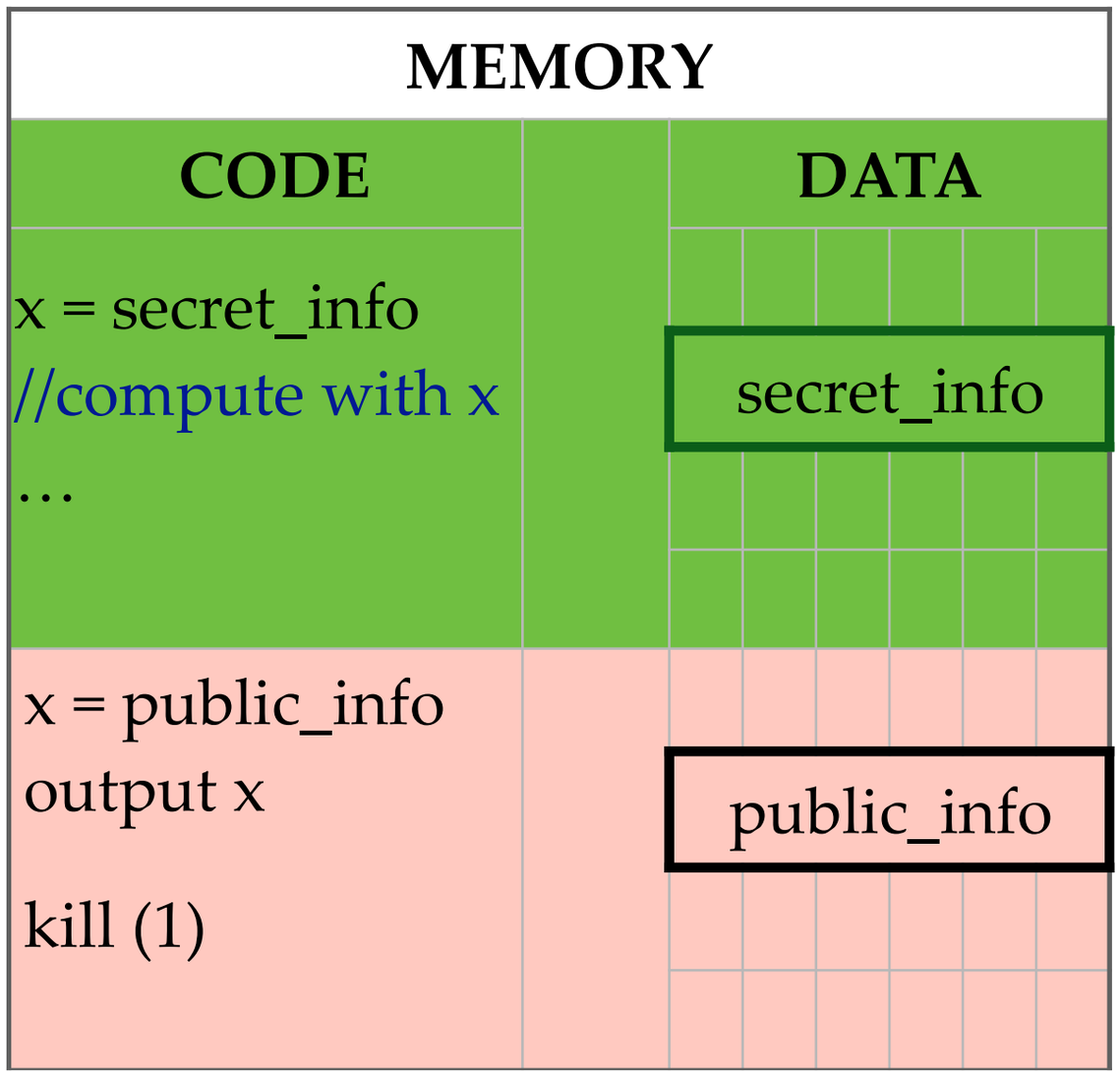


Execution

Overview of Enclaves

```
enclave {
 x = secret_info
 //compute with x
 …

}
x = public_info
output x

kill
```

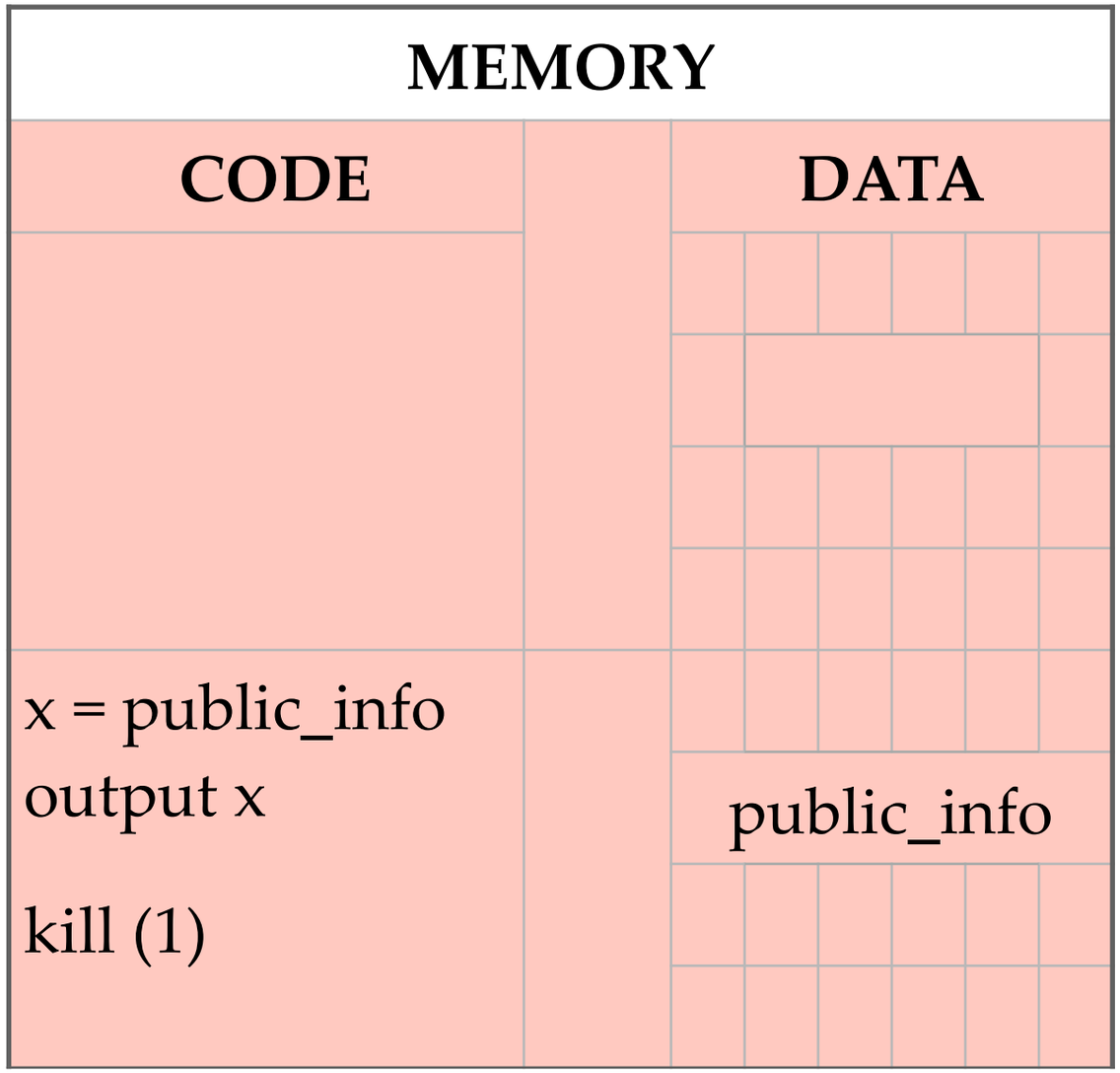
Program



Overview of Enclaves

```
enclave {
 x = secret_info
 //compute with x
  …

}
x = public_info
output x

kill
```

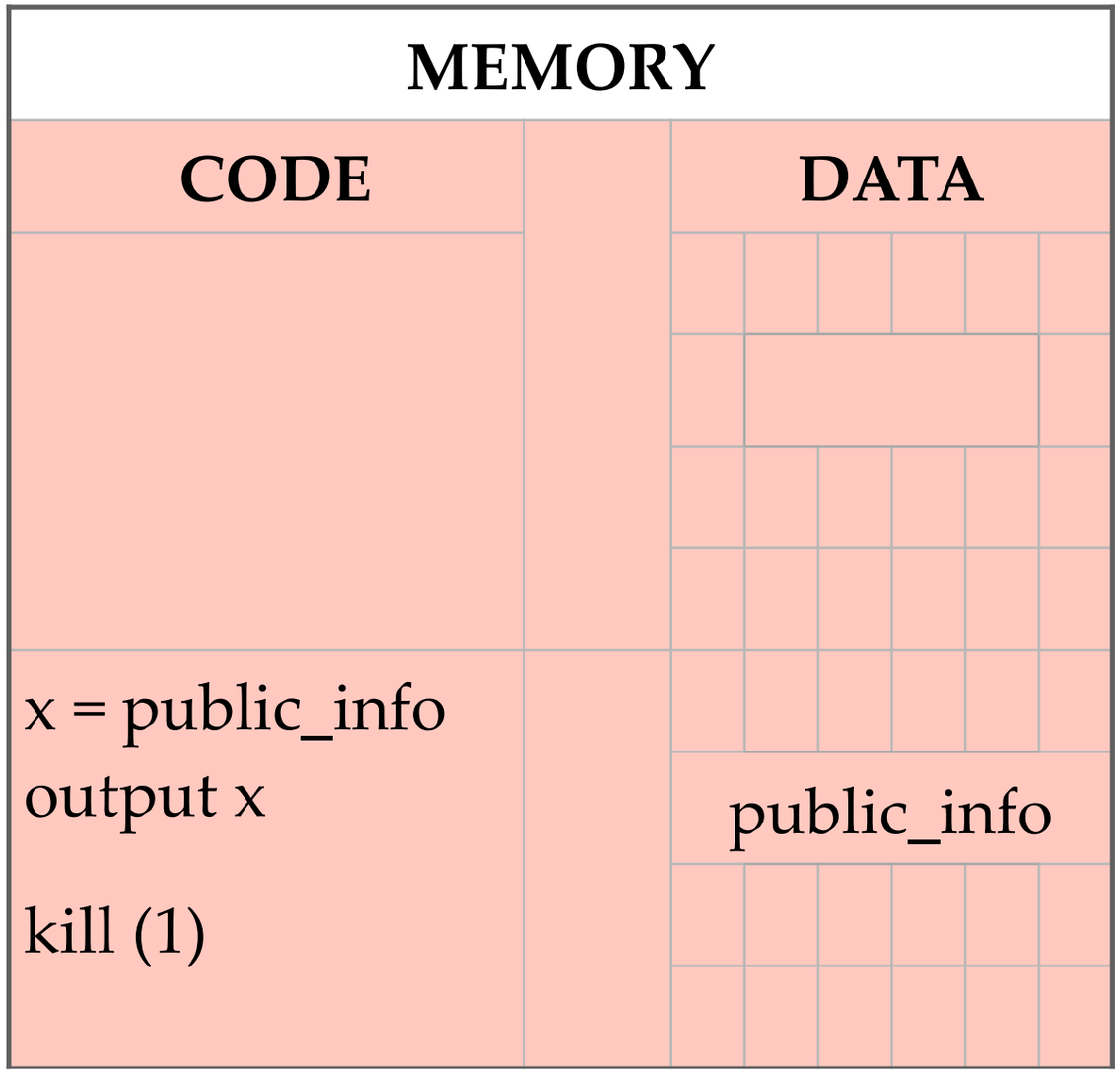
Program



Execution

Overview of Enclaves

```
enclave {
 x = secret_info
 //compute with x
 …

}
x = public_info
output x

kill
```

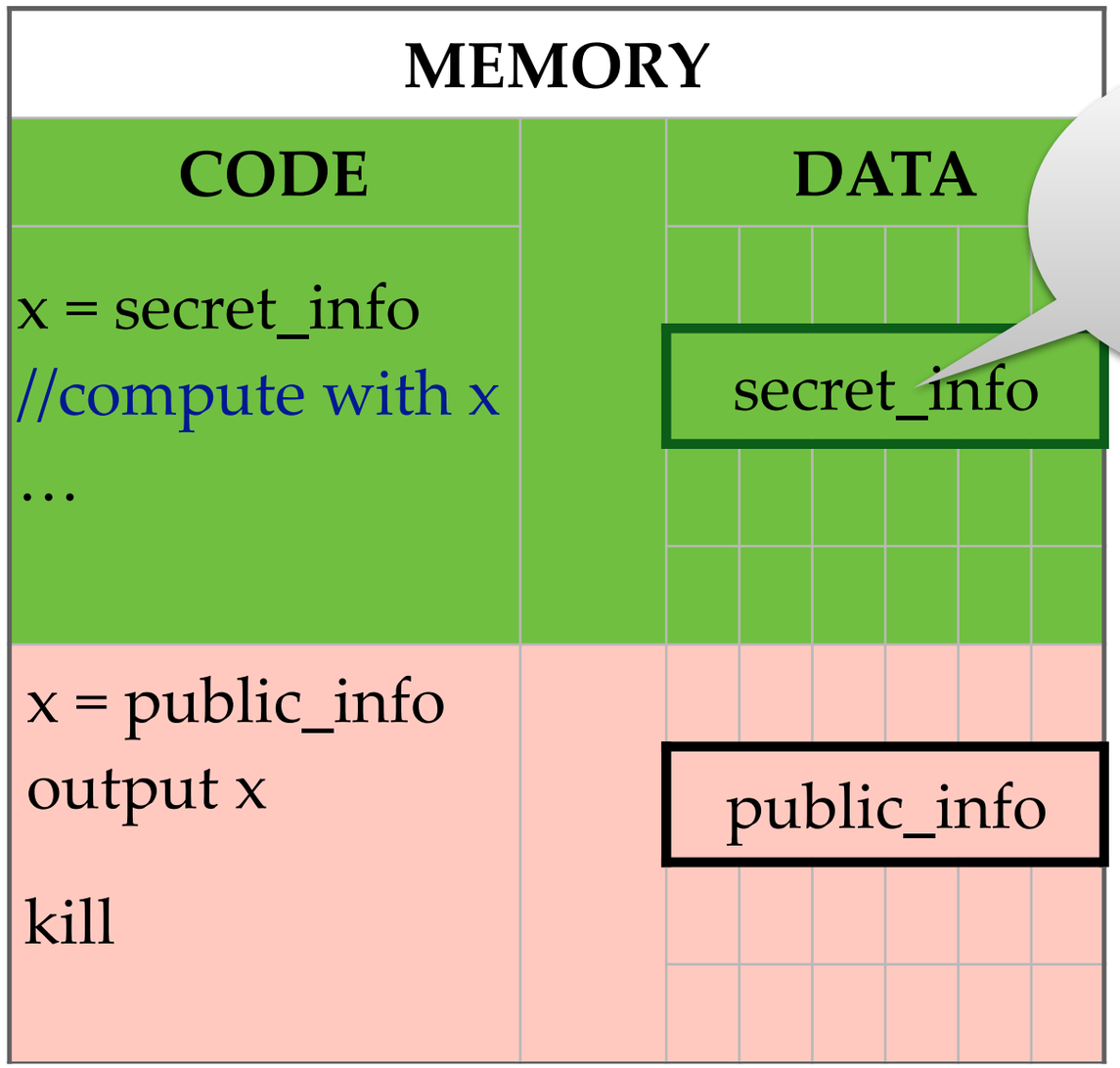


Why are enclaves useful for enforcing security?

Overview of Enclaves

```
enclave {
 x = secret_info
 //compute with x
 …

}
x = public_info
output x

kill
```

Program

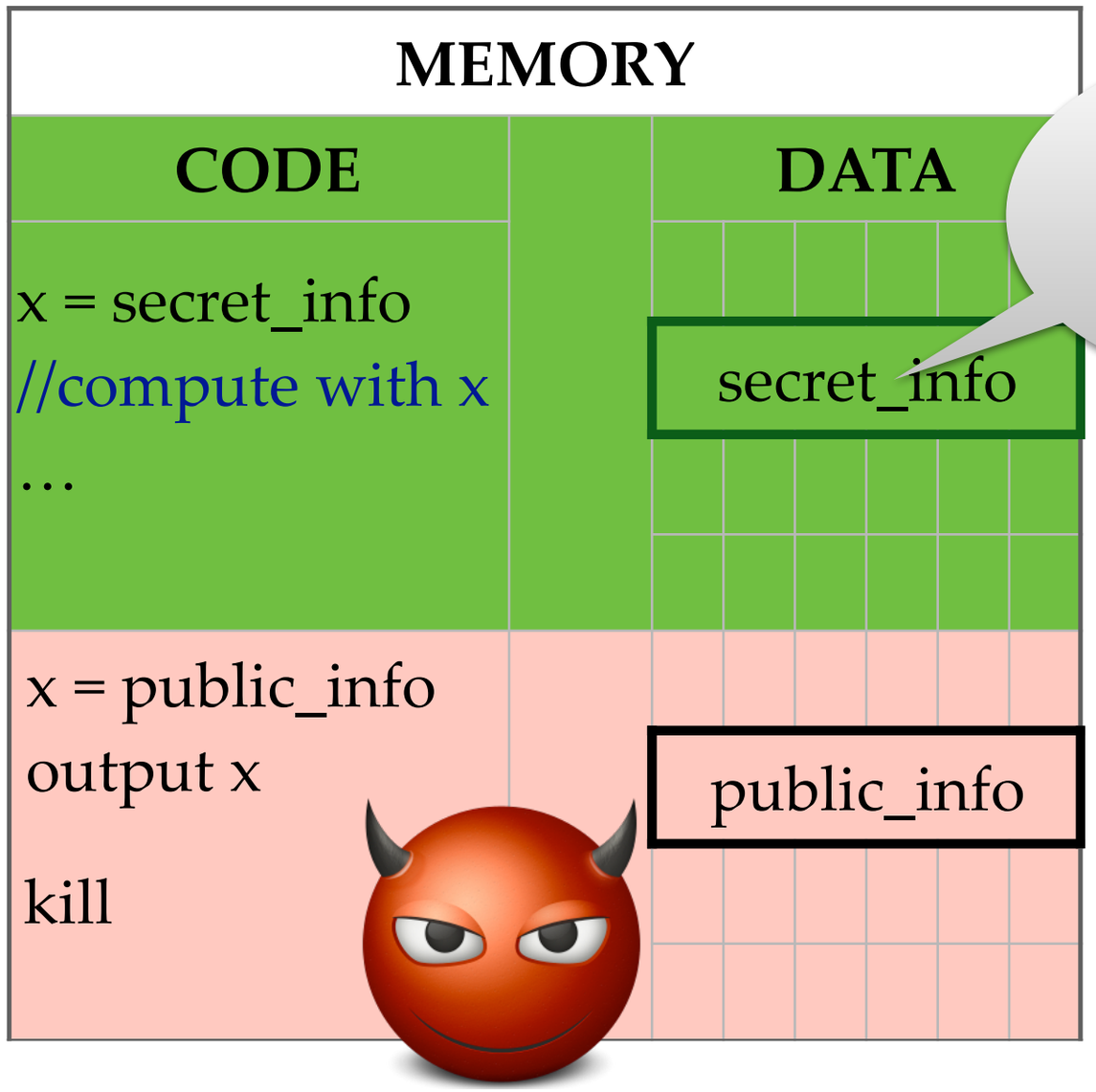


Execution

Overview of Enclaves

```
enclave {
 x = secret_info
 //compute with x
 …

}
x = public_info
output x

kill
```

Program

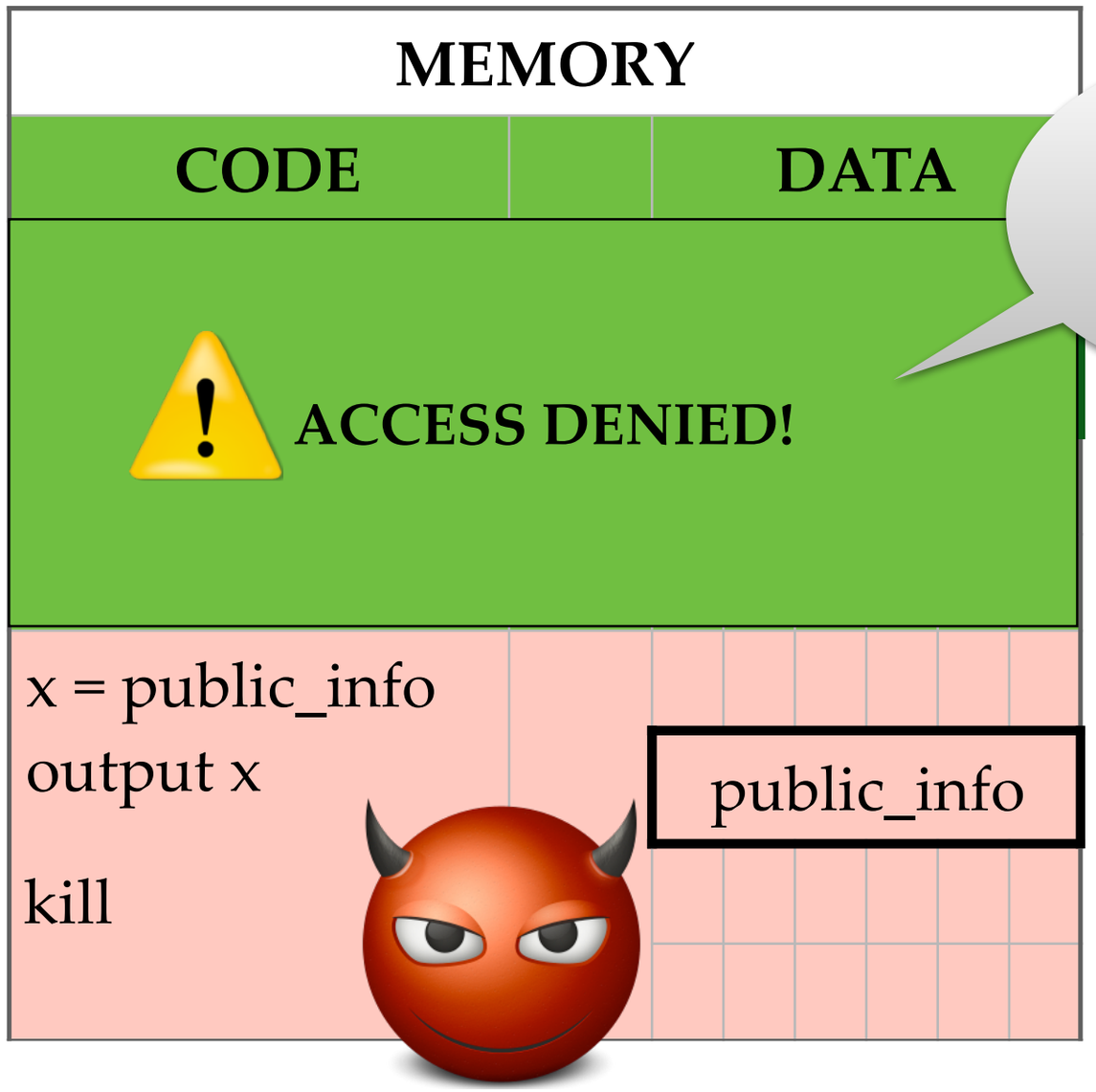


Execution

Overview of Enclaves

```
enclave {
 x = secret_info
 //compute with x
 …

}
x = public_info
output x

kill
```

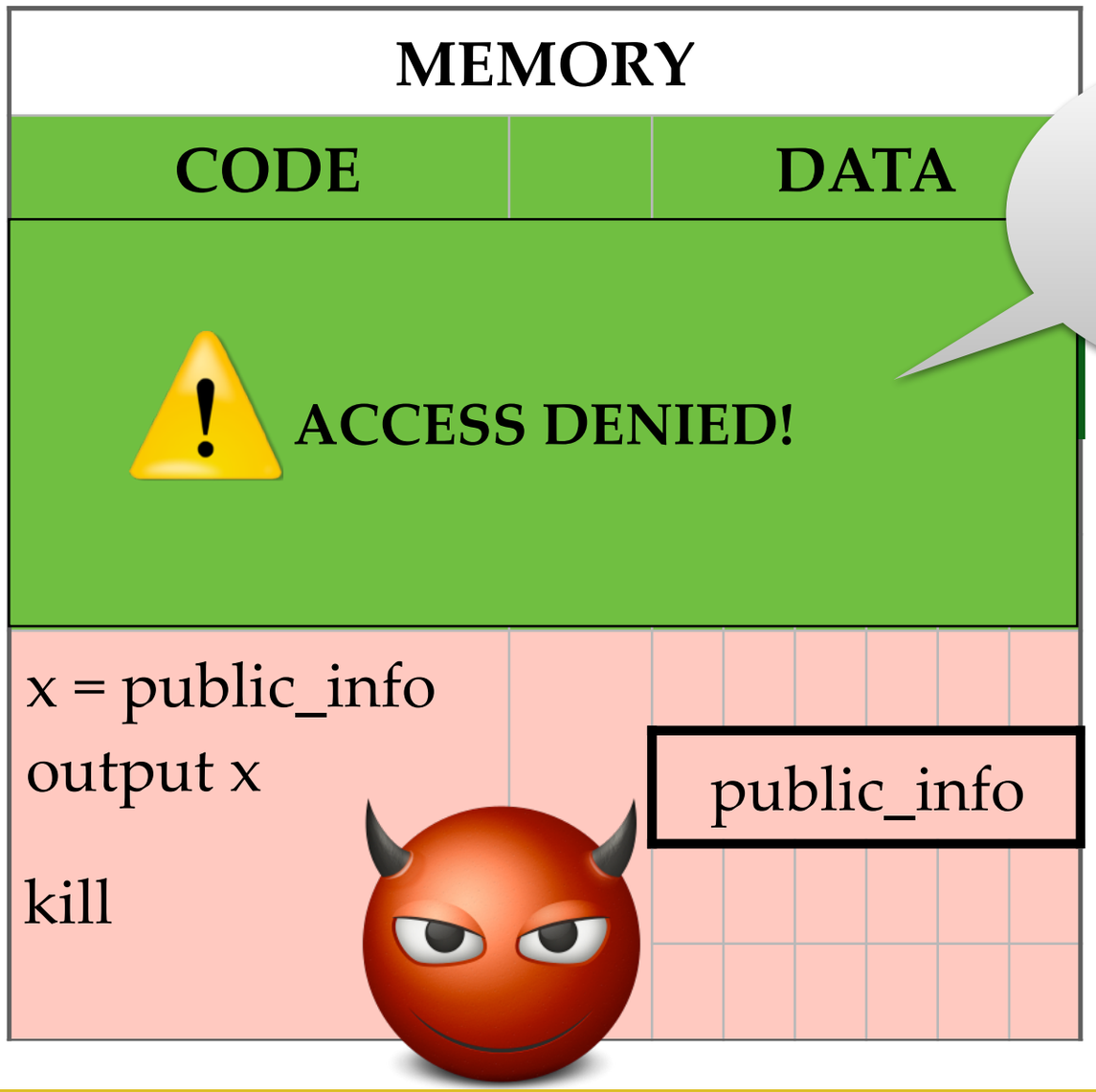
Program



Execution

Overview of Enclaves

```
enclave {
 x = secret_info
 //compute with x
 …

}
x = public_info
output x

kill
```

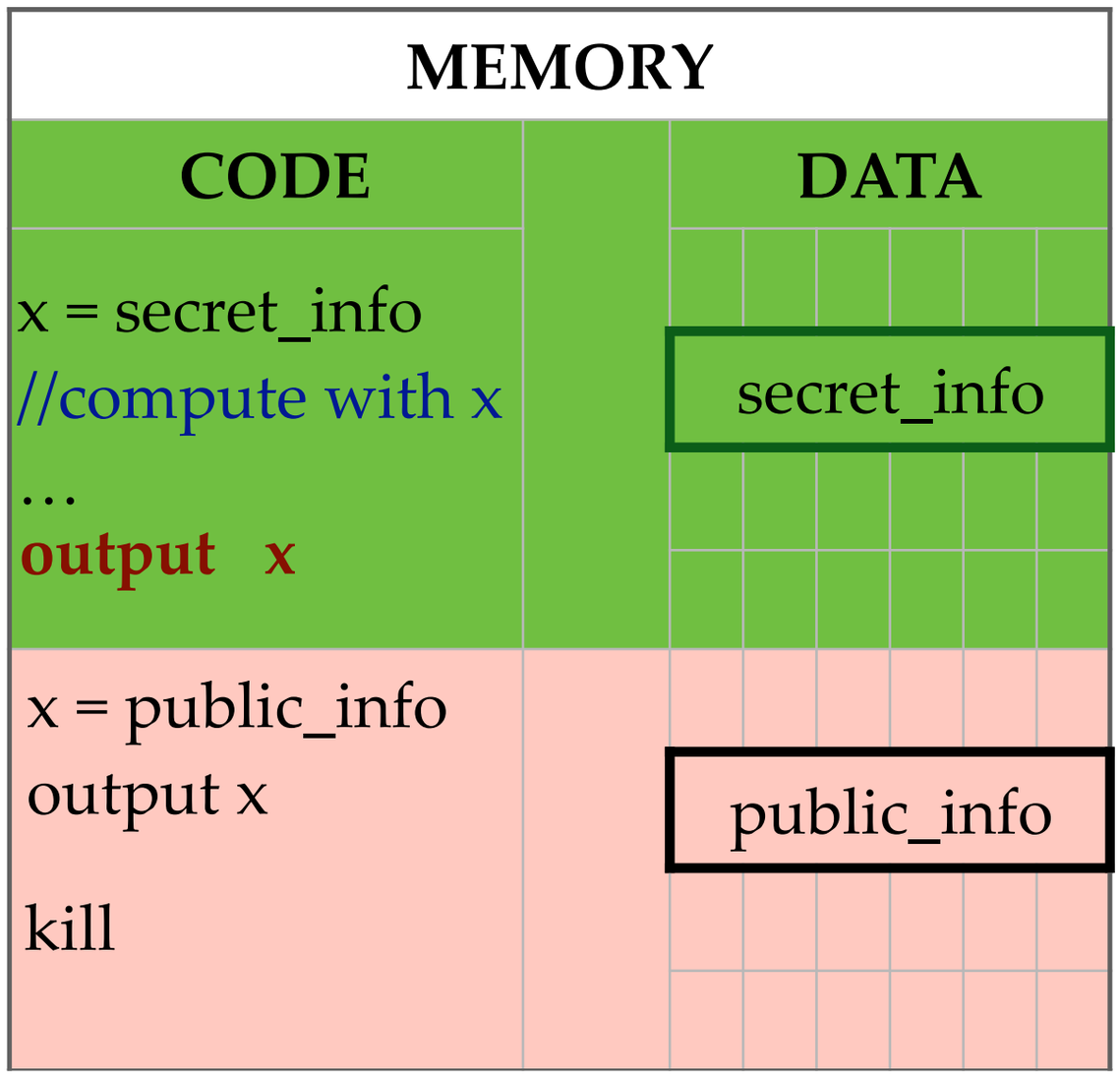


However, enclaves by themselves are insufficient!

Enclaves are Insufficient

```
enclave {
 x = secret_info
 //compute with x
 …
output x
}
x = public_info
output x

kill
```

Program

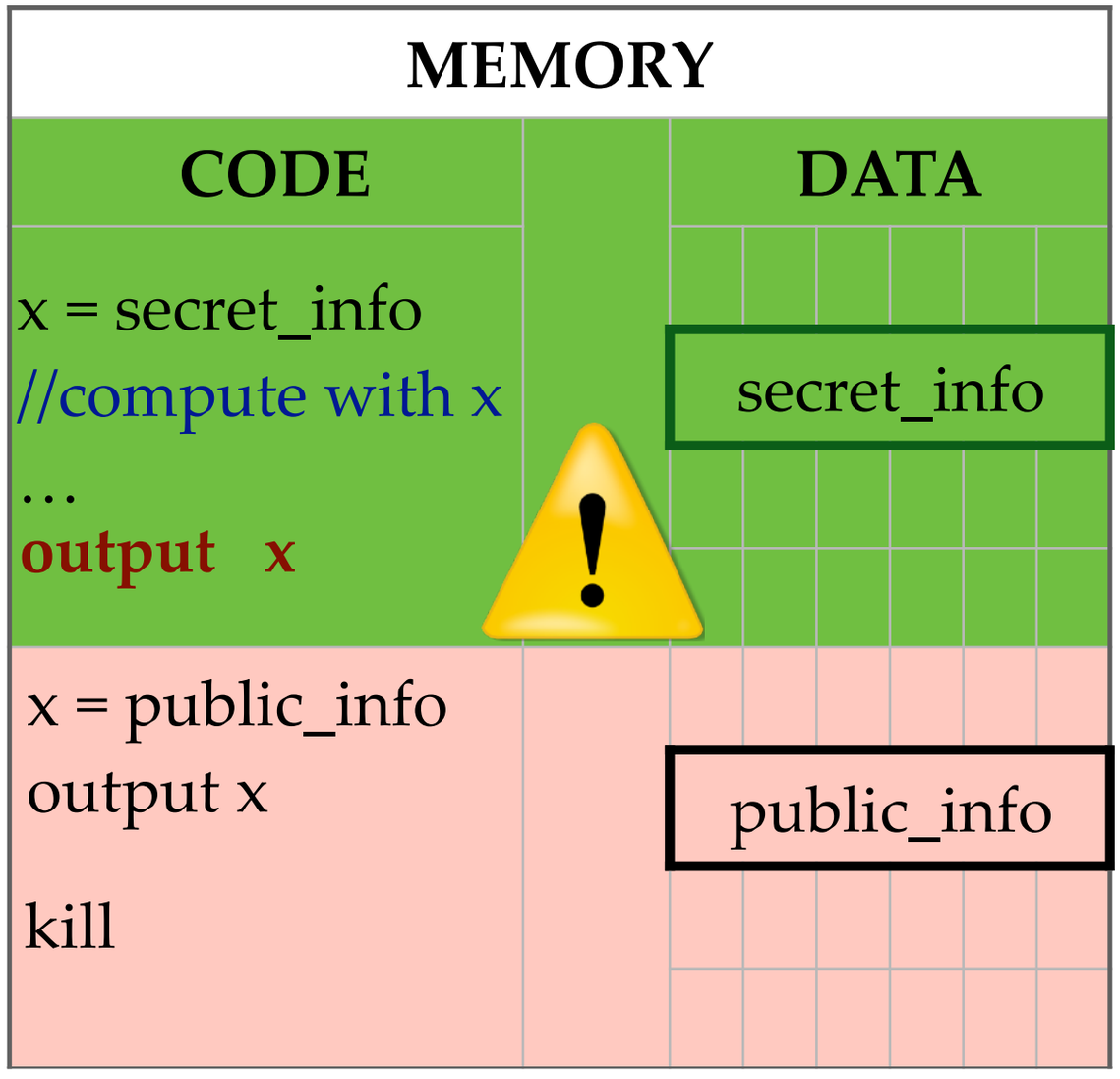


Execution

Enclaves are Insufficient

```
enclave {
 x = secret_info
 //compute with x
 …
output x
}
x = public_info
output x

kill
```

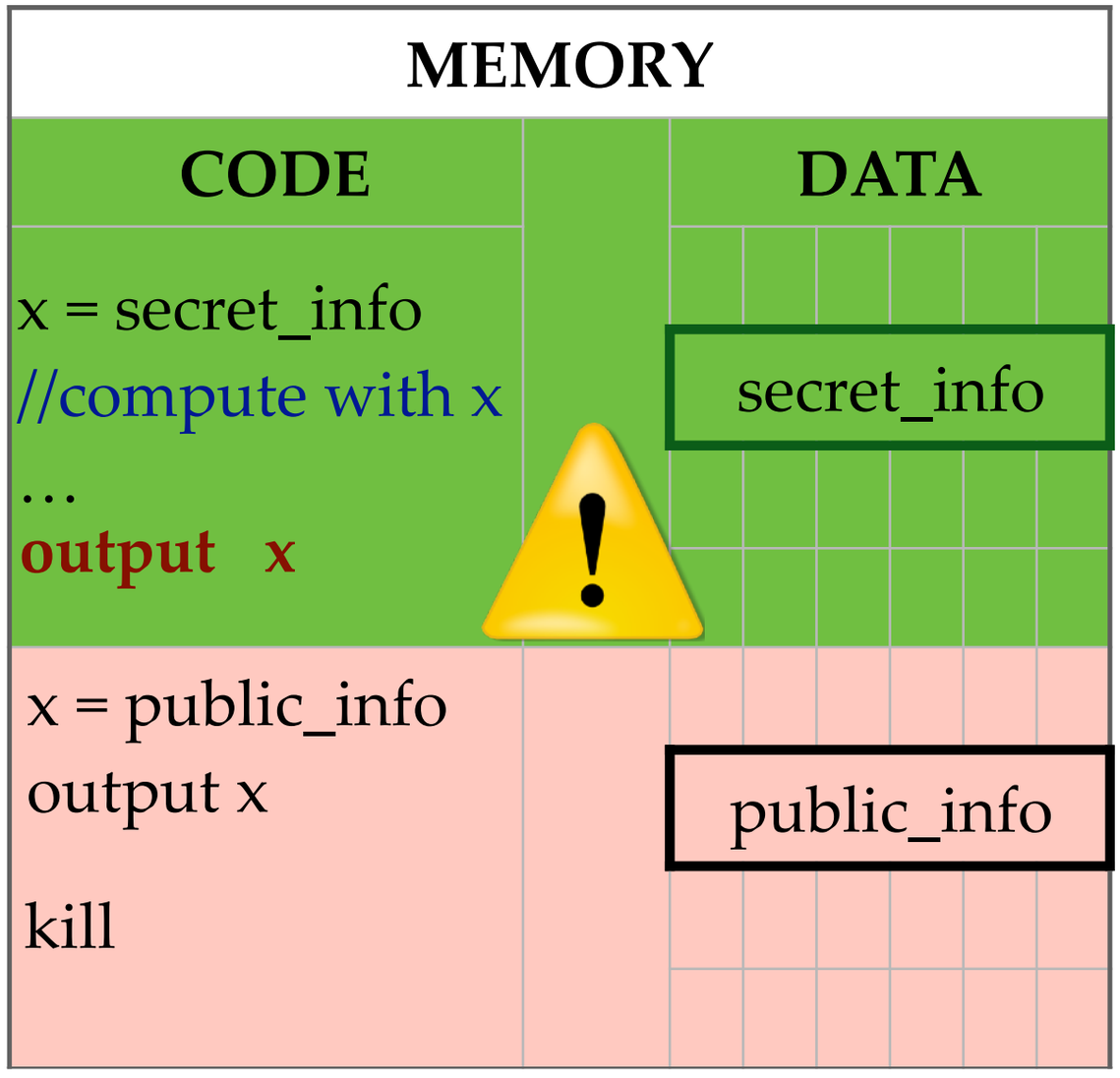
Program



Execution

Enclaves are Insufficient

```
enclave {
 x = secret_info
 //compute with x
 …
output x
}
x = public_info
output x

kill
```



Extend the language-based security mechanisms with enclaves!

IMPE

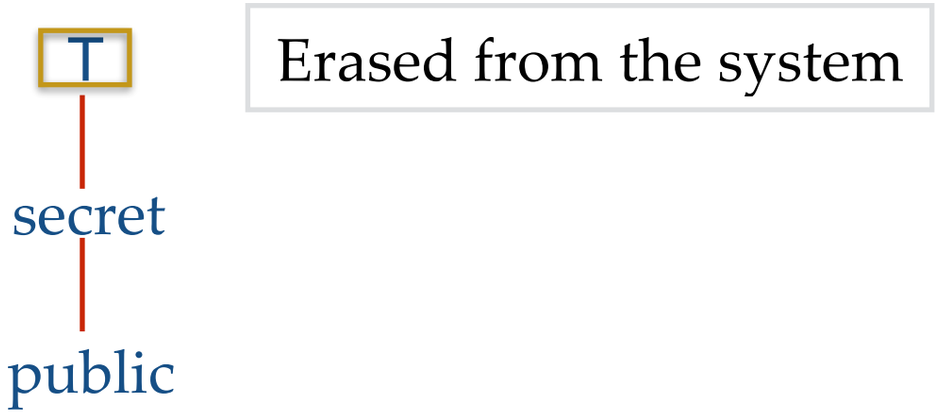
- An expressive formal calculus
- Extends a standard imperative calculus with
  - Enclaves
  - First-class functions
  - Security policies
    - Express application-specific security requirements

Security Policies

- Confidentiality levels
  - form a linear order

Security Policies

- Confidentiality levels
  - form a linear order



Security Policies

- Confidentiality levels
  - form a linear order
- Erasure policy : change of confidentiality level during the program execution

Erased from the system

- e.g. secret_info : 

Security Policies

- Confidentiality levels
  - form a linear order
- Erasure policy : change of confidentiality level during the program execution
  - e.g. secret_info : secret → end → T

Erased from the system

```
enclave {
  x = secret_info
  //compute with x
  …
}
x = public_info
set(end)
output x
kill
```

Security Policies

- Confidentiality levels
  - form a linear order
- Erasure policy : change of confidentiality level during the program execution
  - e.g. secret_info : secret → end → T

Erased from the system

```
enclave {
  x = secret_info
  //compute with x
  …
}
x = public_info
set(end)
output x
kill
```

Security policies are enforced w.r.t. a threat model

	Attacker	Observe Output	Modify Non-Enclave Memory	Modify Enclave Memory	Example
	Passive	✔	✘	✘	Network monitoring
	Non-Enclave Active	✔	✔	✘	Malware

	Attacker	Observe Output	Modify Non-Enclave Memory	Modify Enclave Memory	Example
	Passive	✔	✘	✘	Network monitoring
	Non-Enclave Active	✔	✔	✘	Malware
	Enclave Active	✔	✔	✔	Vulnerabilities in enclave code

	Attacker	Observe Output	Modify Non-Enclave Memory	Modify Enclave Memory	Example
	Passive	✓	✗	✗	Network monitoring
	Non-Enclave Active	✓	✓	✗	Malware
	Enclave Active	✓	✓	✓	Vulnerabilities in enclave code

When is a program secure against these attackers?

Security

- Formally defined as a non-interference property
  - Public outputs are not influenced by private inputs
- Parameterized by the kind of attacker

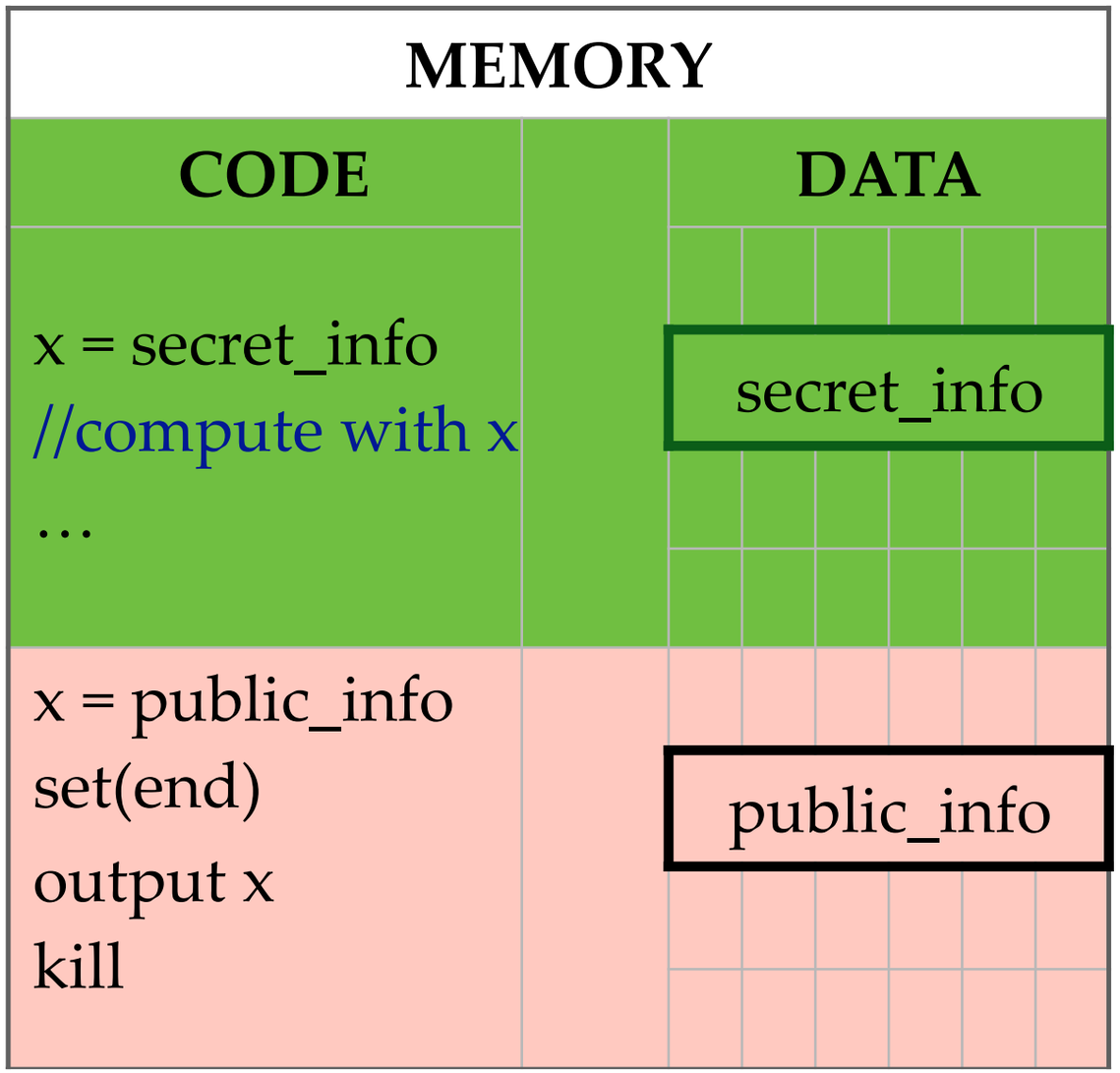
Security

- Formally defined as a non-interference property
  - Public outputs are not influenced by private inputs
- Parameterized by the kind of attacker

Security against weaker attacker $\not\Rightarrow$ security against powerful attacker

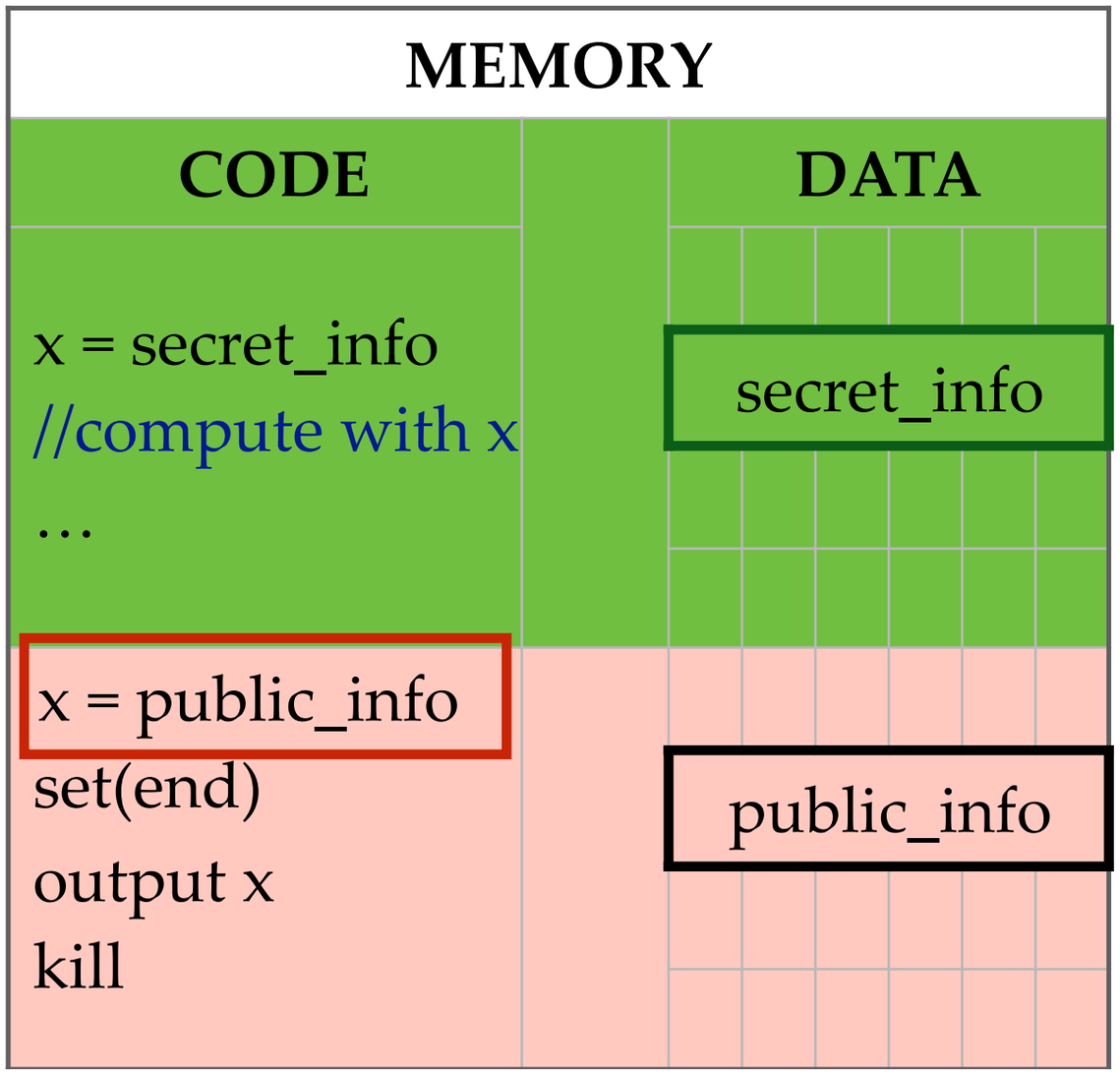
Security Against Passive Attacker

enclave {
x = secret_info
//compute with x
…
}
x = public_info
set(end)
output x
kill



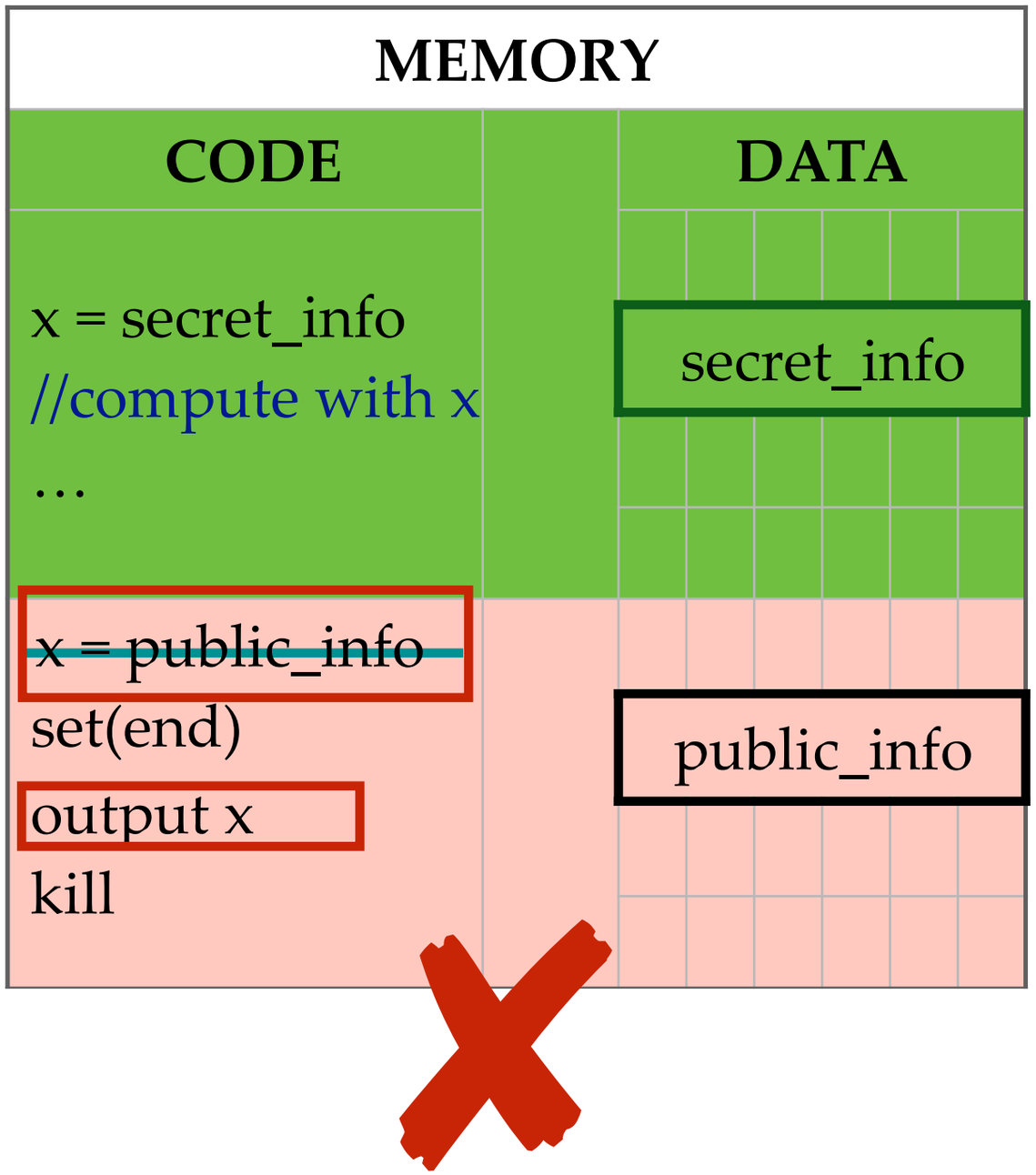
Security Against Non-enclave Active Attacker

```
enclave {
  x = secret_info
  //compute with x
  …
}
x = public_info
set(end)
output x
kill
```



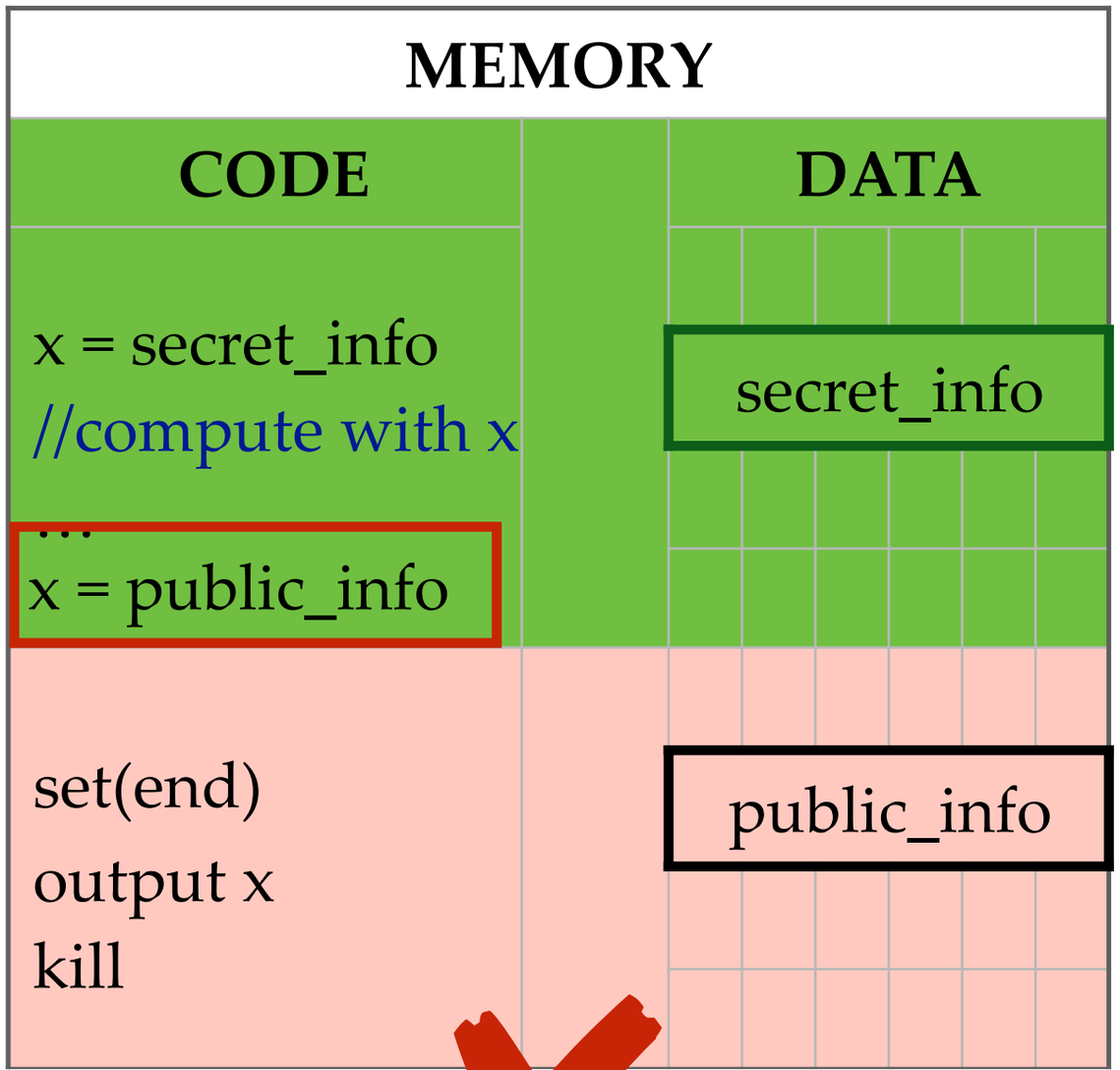
Security Against Non-enclave Active Attacker

```
enclave {
  x = secret_info
  //compute with x
  …
}
x = public_info
set(end)
output x
kill
```



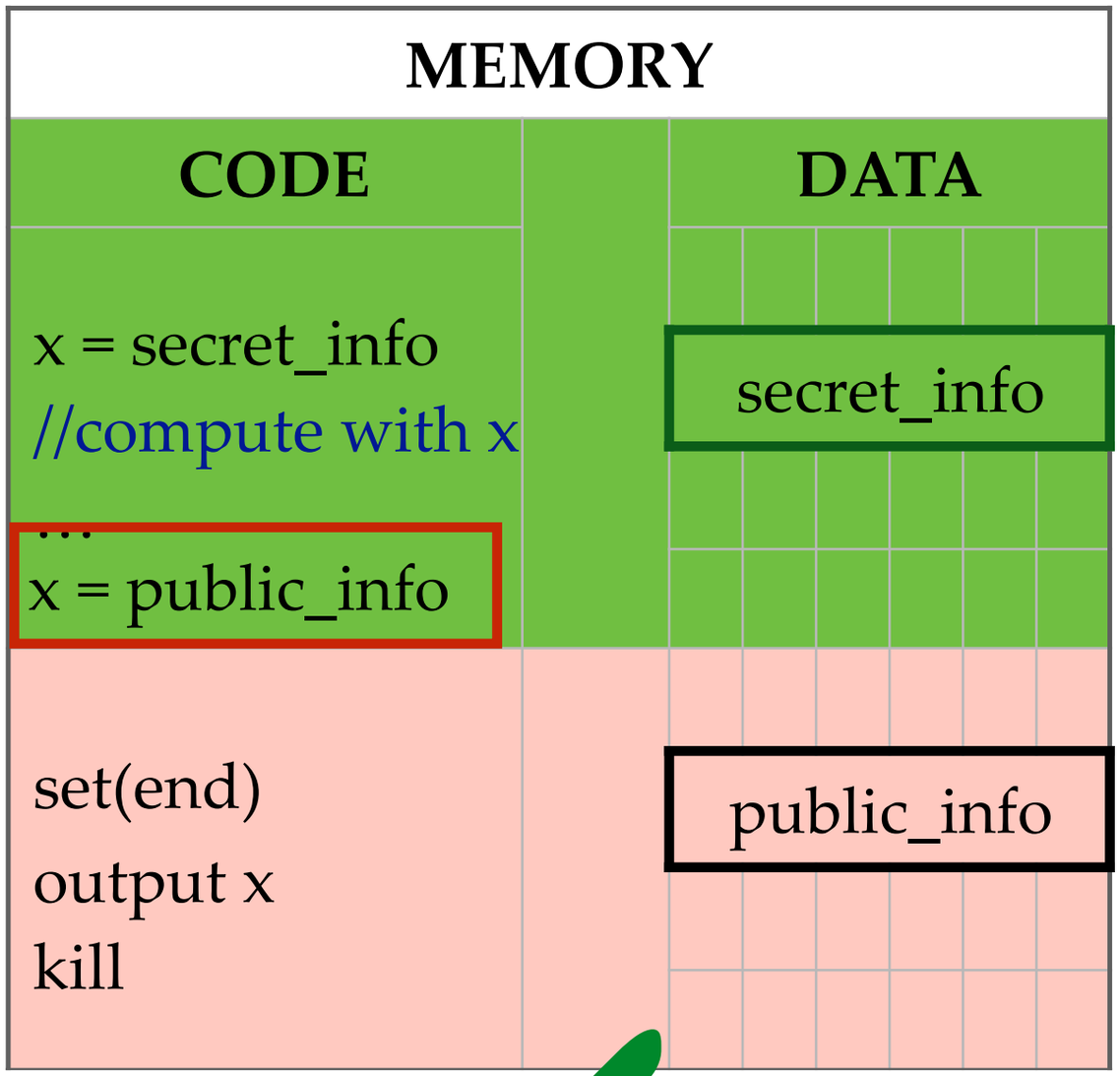
Security Against Non-enclave Active Attacker

```
enclave {
  x = secret_info
  //compute with x
  …
  x = public_info
}
set(end)
output x
kill
```



Security Against Non-enclave Active Attacker

```
enclave {
  x = secret_info
  //compute with x
  …
  x = public_info
}
set(end)
output x
kill
```



Enforcing Security

- Security Type System
  - secret information is placed only in enclaves
  - code that manipulates the secret information is placed in the same enclave

Enforcing Security

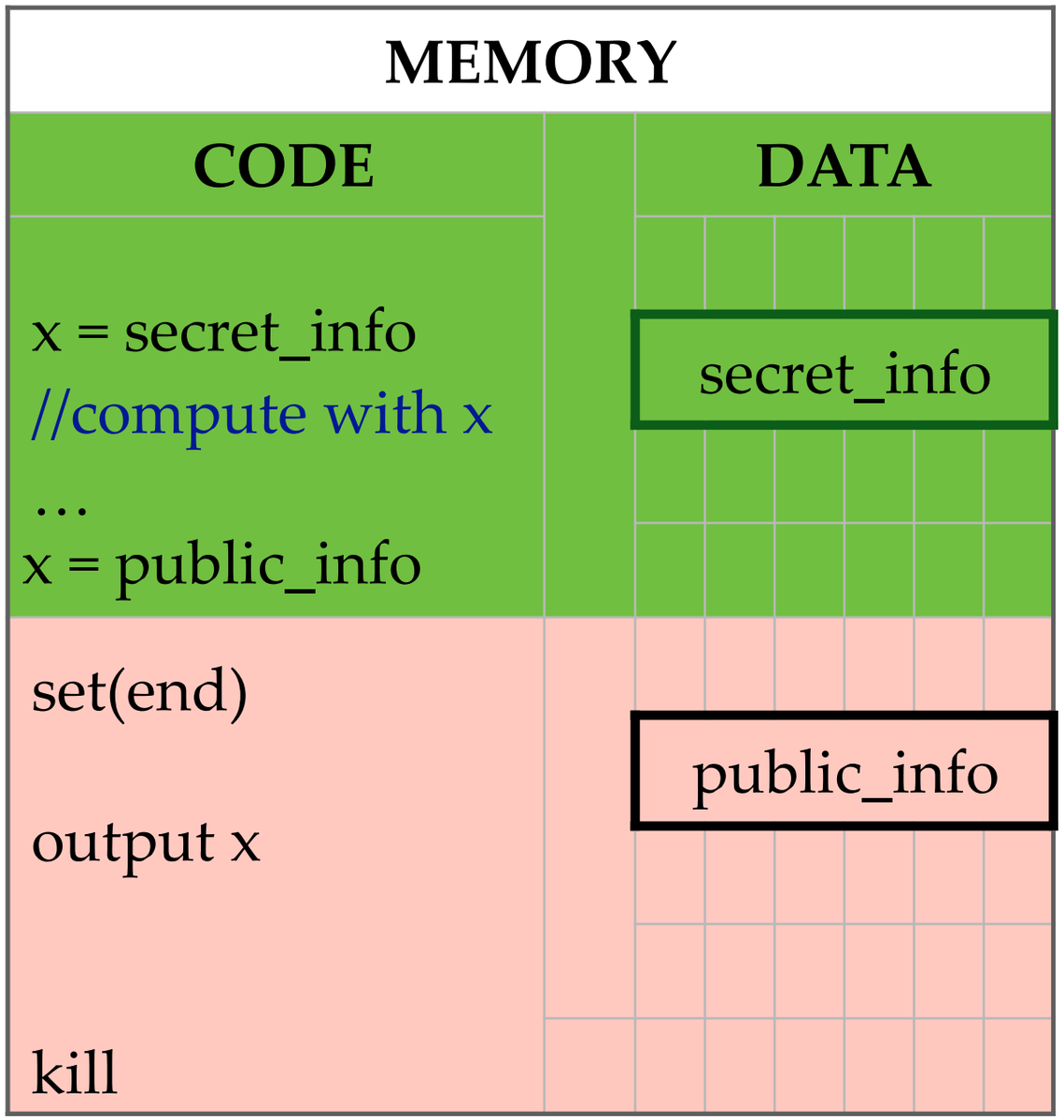
- **Theorem:** Well-typed IMPE programs are secure against
  - Passive attacker
  - Non-enclave active attacker

Enforcing Security

- **Theorem:** Well-typed IMPE programs are secure against
  - Passive attacker
  - Non-enclave active attacker
- Is the program secure for **enclave active attacker**?

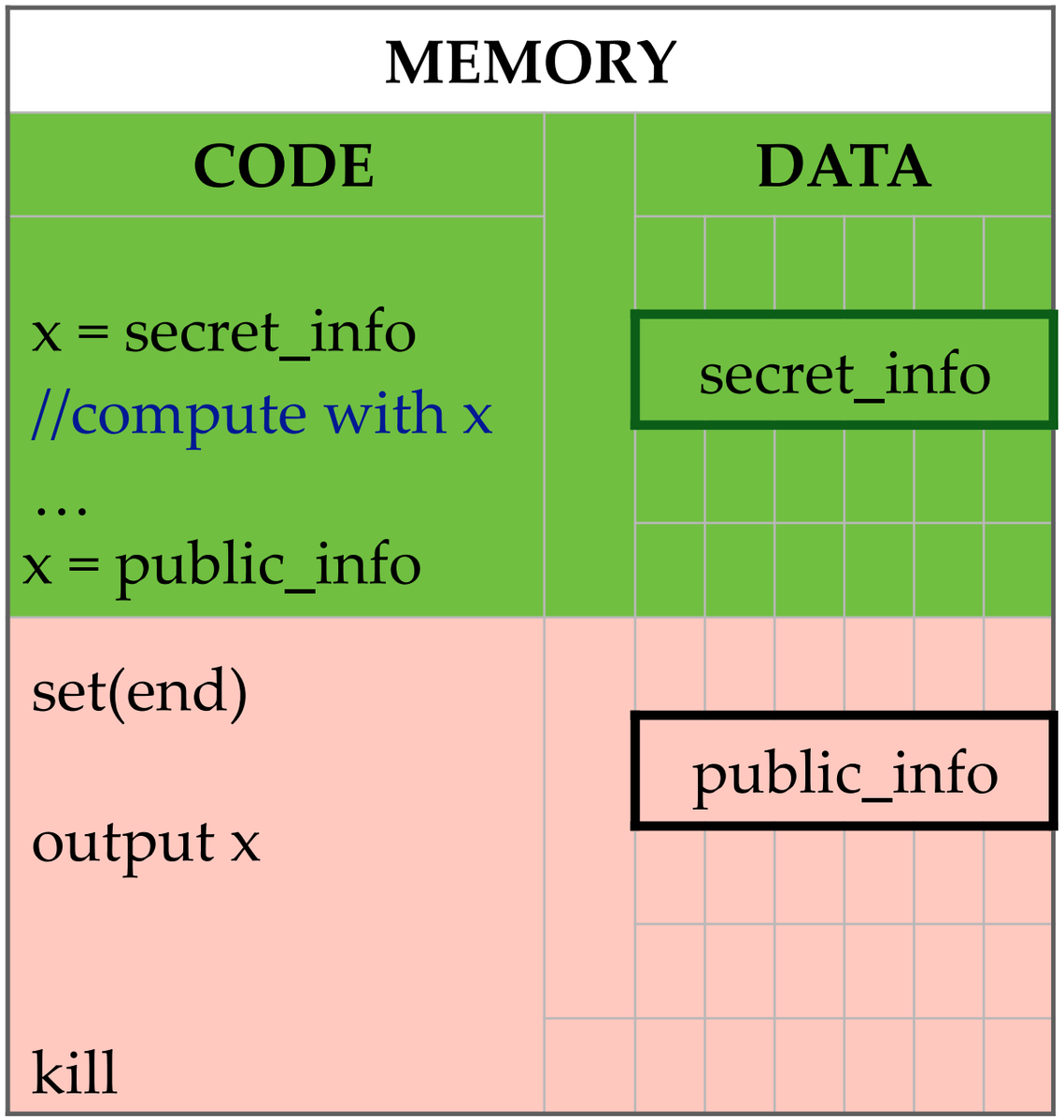
Security Against Enclave Active Attacker

```
enclave {
  x = secret_info
  //compute with x
  …
  x = public_info
}
set(end)


   output x


kill
```

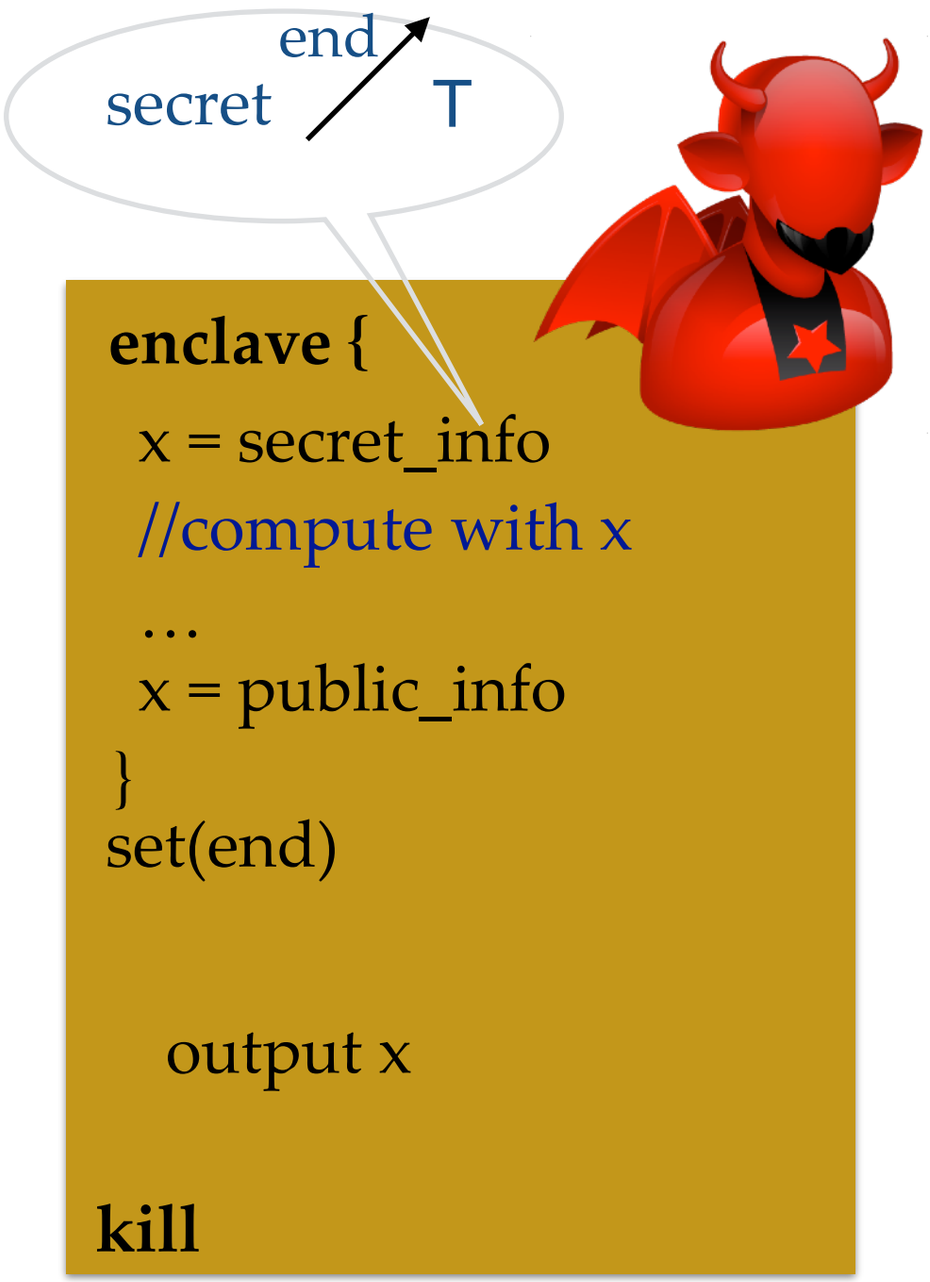


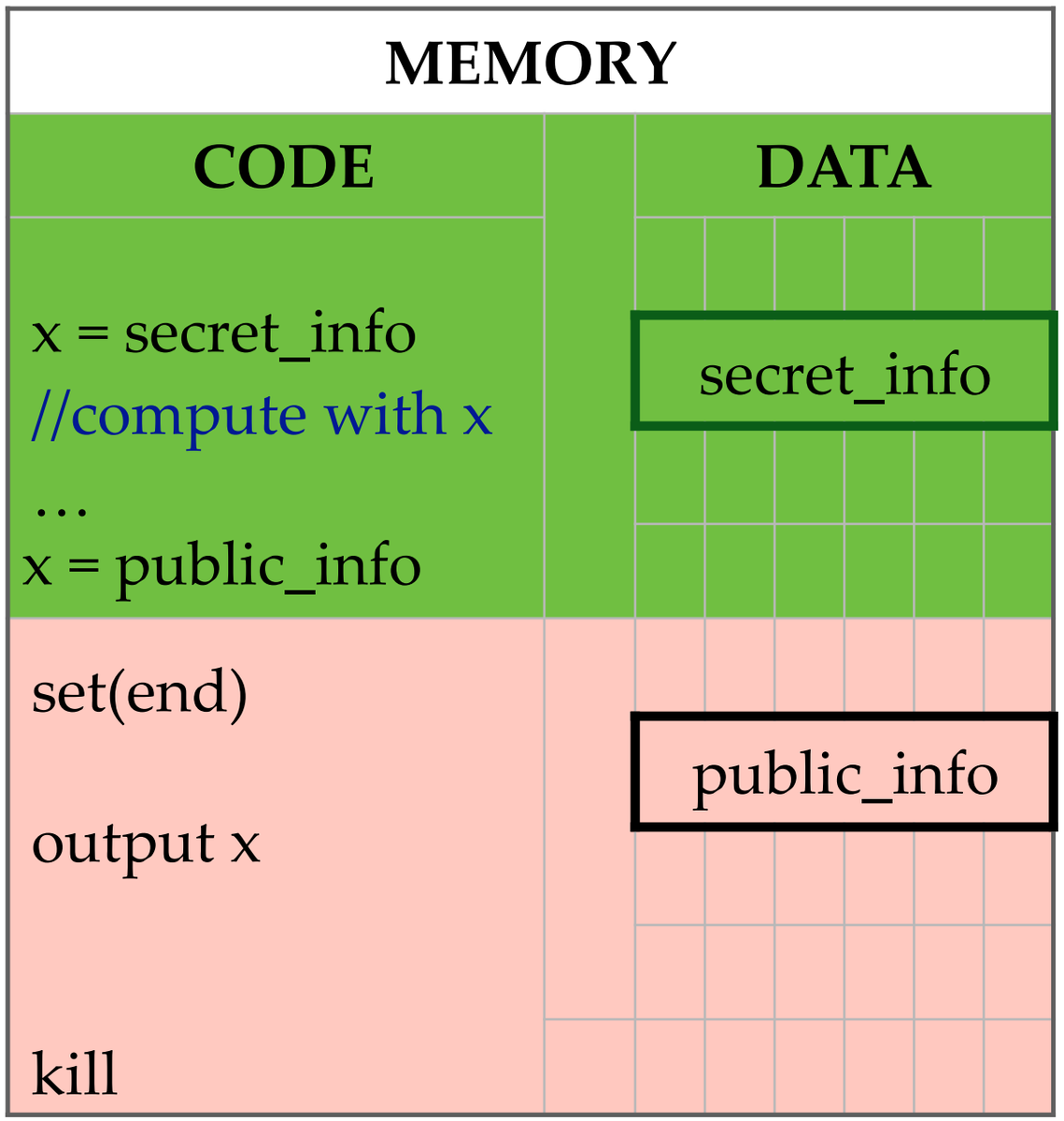
Security Against Enclave Active Attacker

```
enclave {
  x = secret_info
  //compute with x
  …
  x = public_info
}
set(end)


   output x


kill
```

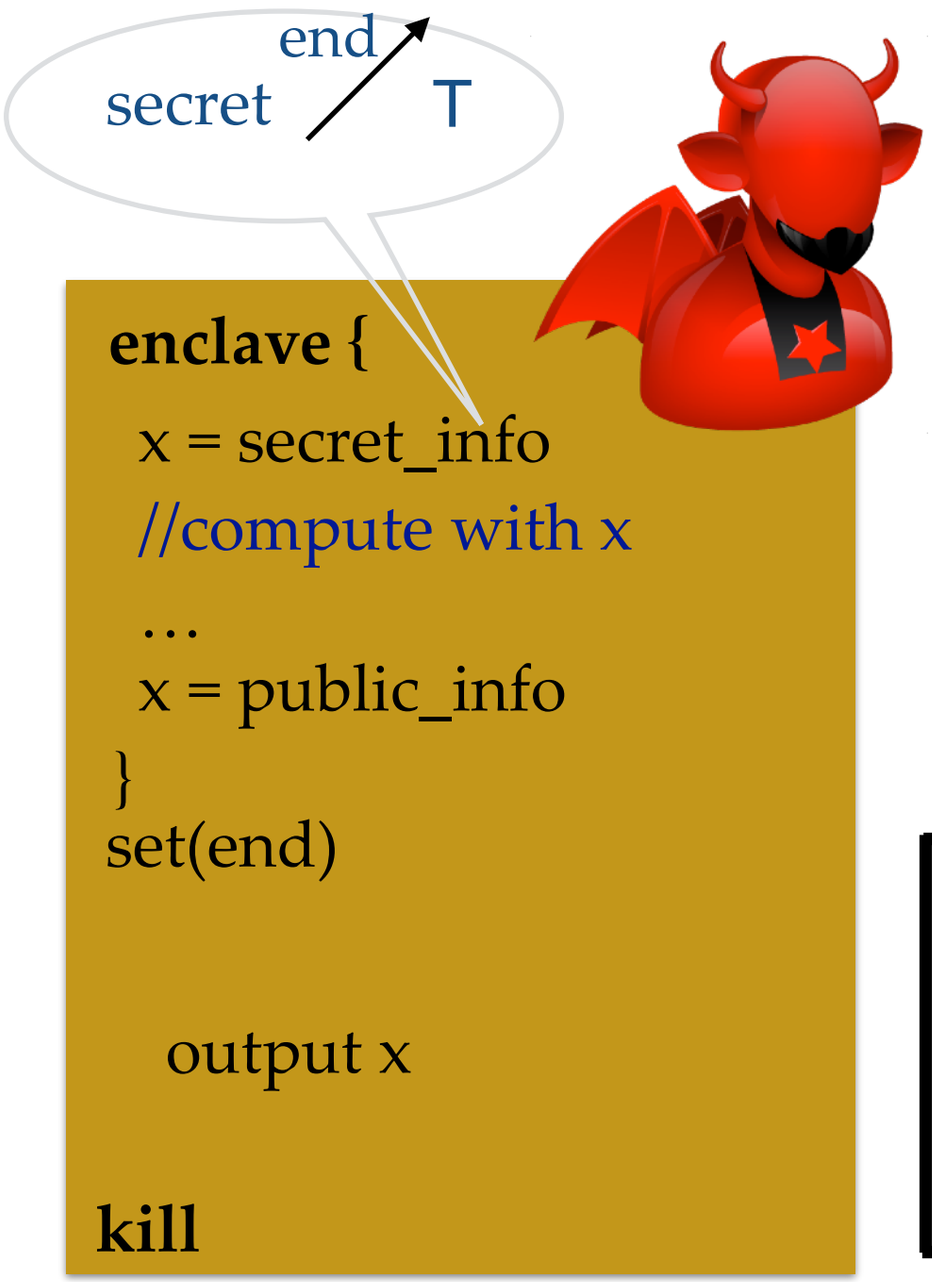


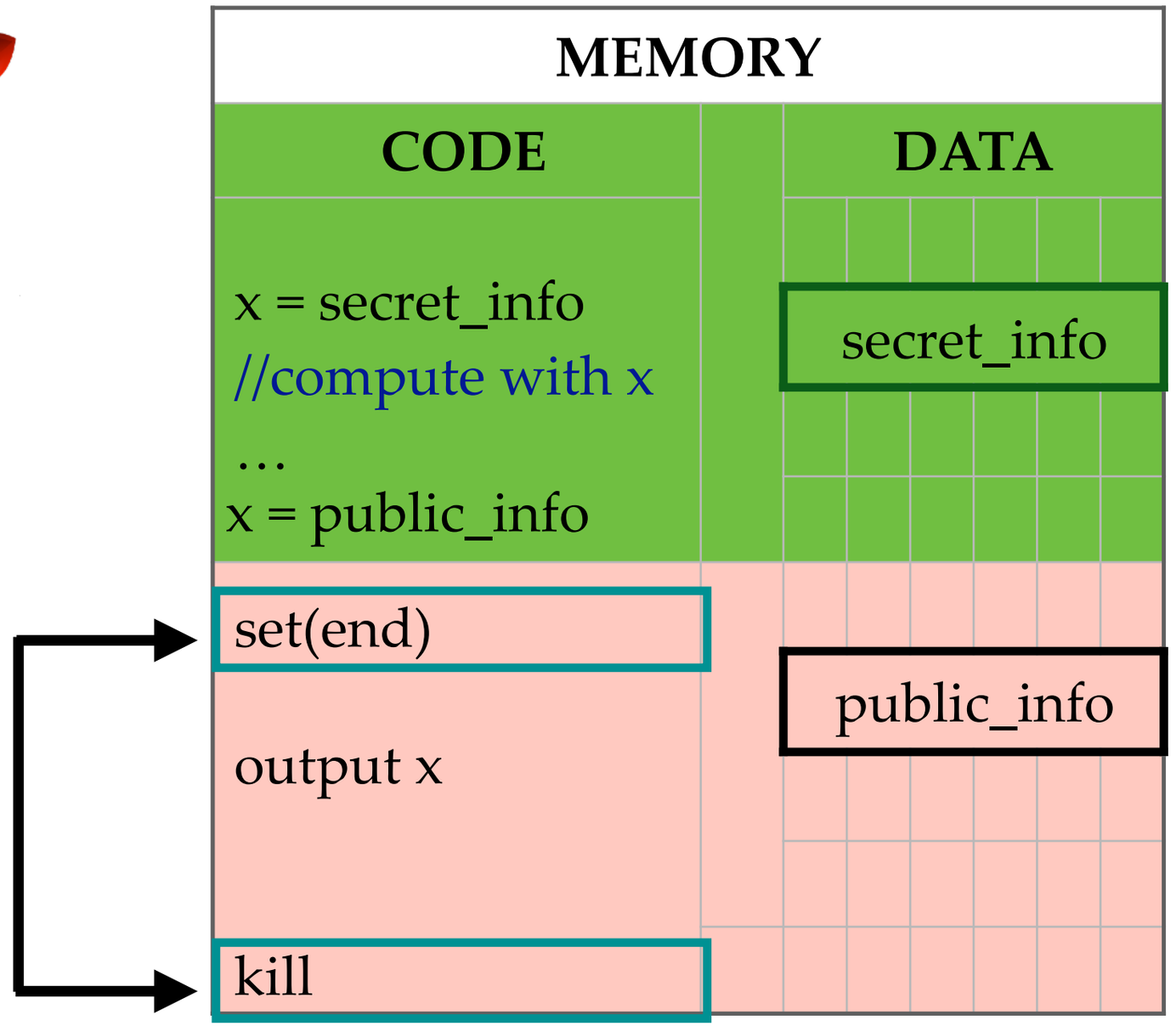
Security Against Enclave Active Attacker





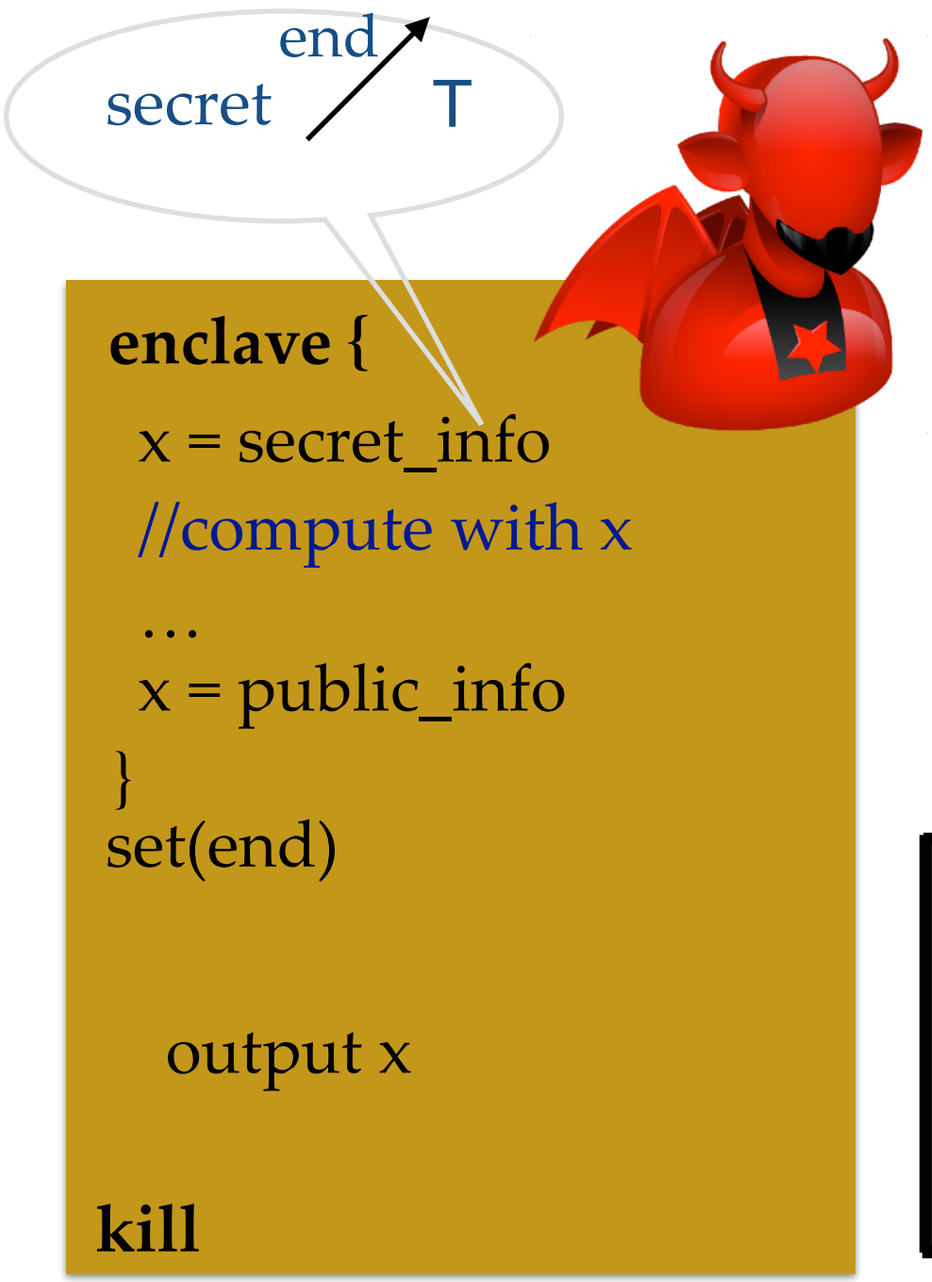
Security Against Enclave Active Attacker

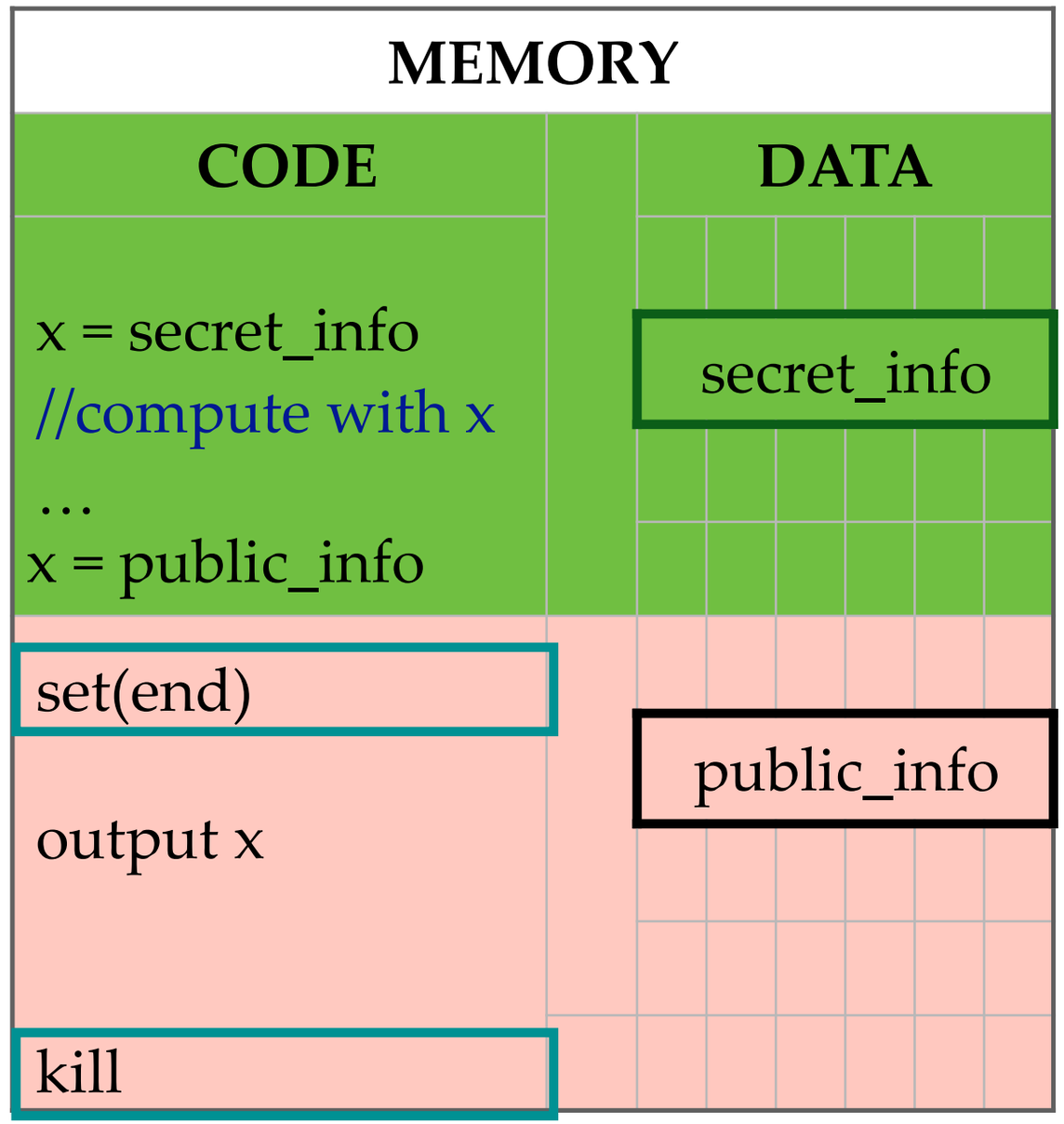




- Window of vulnerability

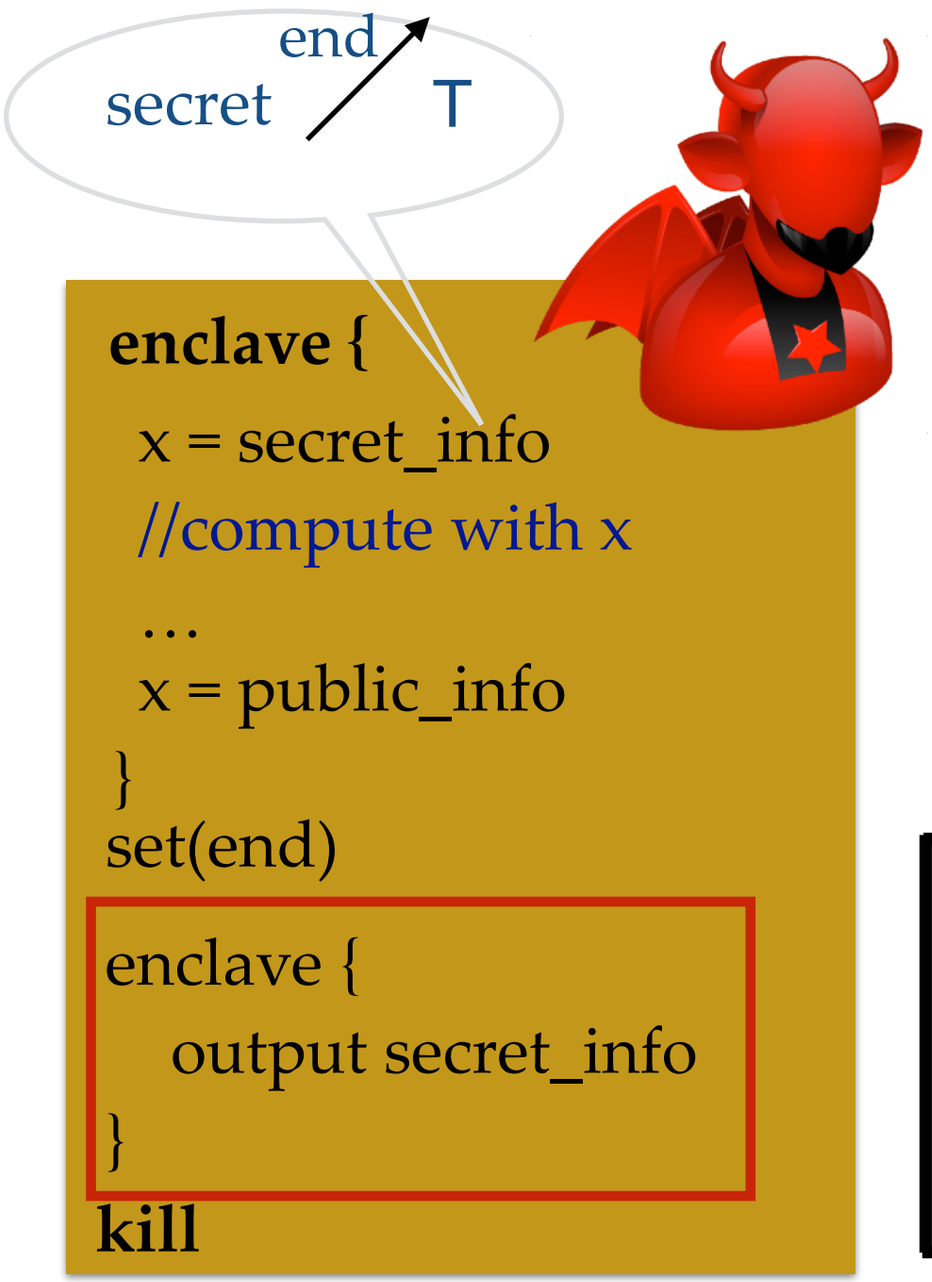
Security Against Enclave Active Attacker

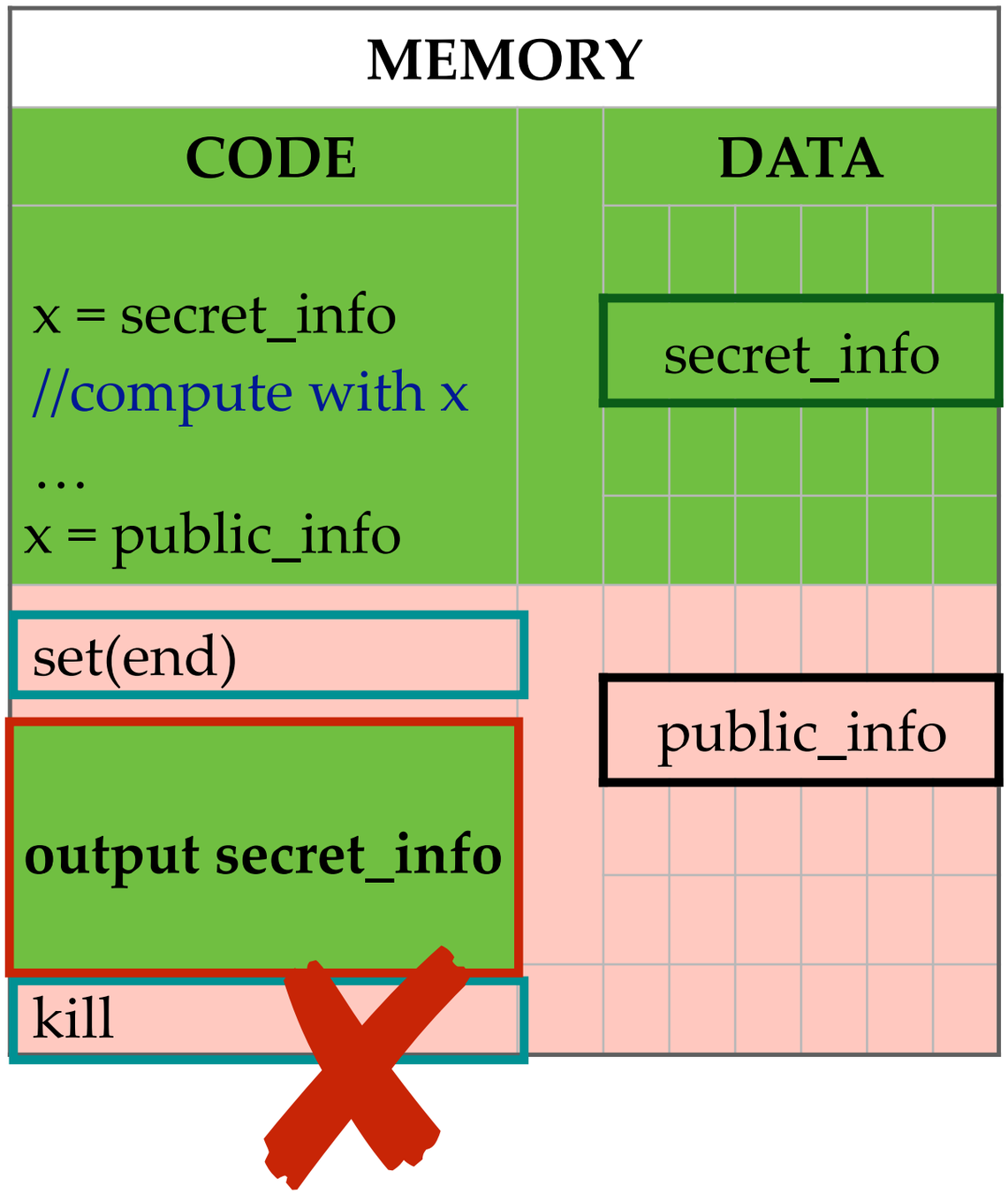




- Window of vulnerability

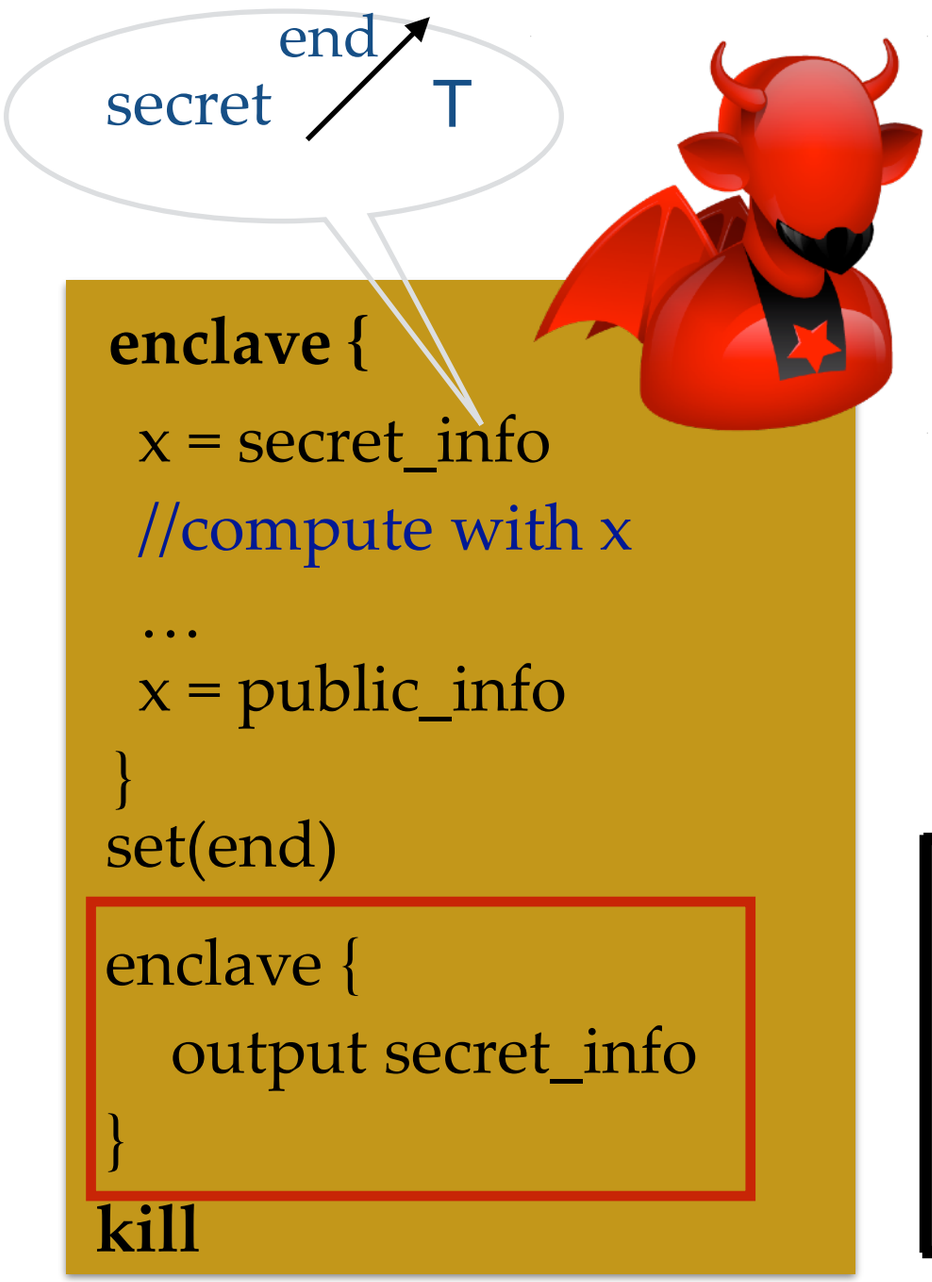
Security Against Enclave Active Attacker

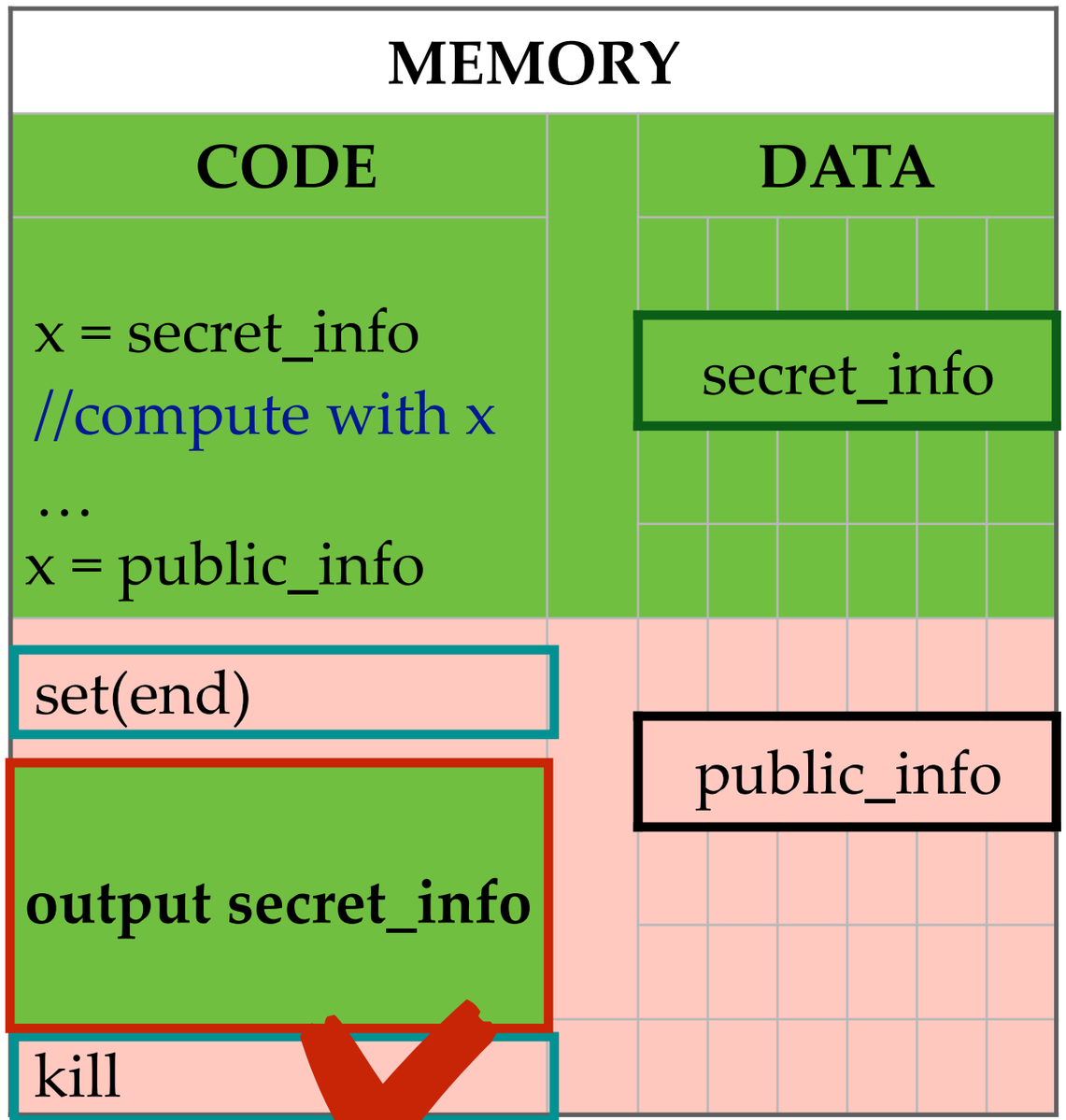




- Window of vulnerability

Security Against Enclave Active Attacker





- Window of vulnerability
- **Smaller the window, better the security**

Enforcing Security against Enclave Active Attackers

- **Theorem:** Well-typed IMPE programs are secure against enclave active attacker

- Only for attacks launched after the enclaves (containing the data to be erased) are killed

Recap

Recap

- Well-typed IMPE programs are:
  - Secure against passive attacker
  - Secure against non-enclave active attacker
  - Partially secure against enclave active attacker

Recap

- Well-typed IMPE programs are:
  - Secure against passive attacker
  - Secure against non-enclave active attacker
  - Partially secure against enclave active attacker

How to partition IMPE programs into enclaves

How to Partition Program into Enclaves?

Trivial solution: Place entire application inside an enclave

How to Partition Program into Enclaves?

Trivial solution: Place entire application inside an enclave

- Increases trusted computing base (TCB)
  - For non-enclave active attacker
    - enclave code is assumed to have no vulnerabilities
  - More enclave code = more assumptions

How to Partition Program into Enclaves?

Trivial solution: Place entire application inside an enclave

- Increases trusted computing base (TCB)
  - For non-enclave active attacker
    - enclave code is assumed to have no vulnerabilities
  - More enclave code = more assumptions
- Increases window of vulnerability
  - Can't kill an enclave until the end
  - Data to be erased lives longer

How to Partition Program into Enclaves?

Using multiple enclaves:

- Fine-grained partitioning leads to smaller enclaves
  - Reduces the lifetime of data to be erased
- Tedious and error-prone

How to Partition Program into Enclaves?

Using multiple enclaves:

- Fine-grained partitioning leads to smaller enclaves
  - Reduces the lifetime of data to be erased
- Tedious and error-prone

We can automatically infer enclave placement!

Enclave Inference as Constraint Satisfaction

Enclave Inference as Constraint Satisfaction

Program w/out Enclaves

Enclave Inference as Constraint Satisfaction

Program w/out Enclaves

- Sensitive data should be in some enclave
- A killed enclave cannot be accessed

constraints

Enclave Inference as Constraint Satisfaction

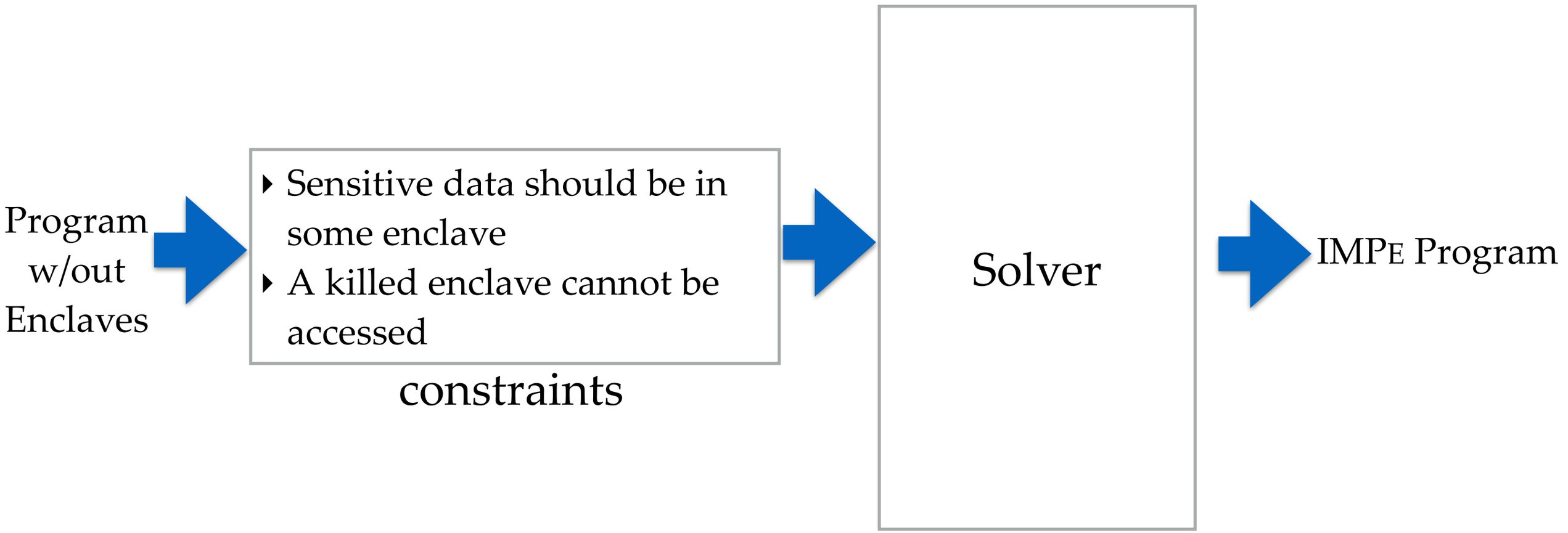
Program w/out Enclaves

- Sensitive data should be in some enclave
- A killed enclave cannot be accessed

constraints

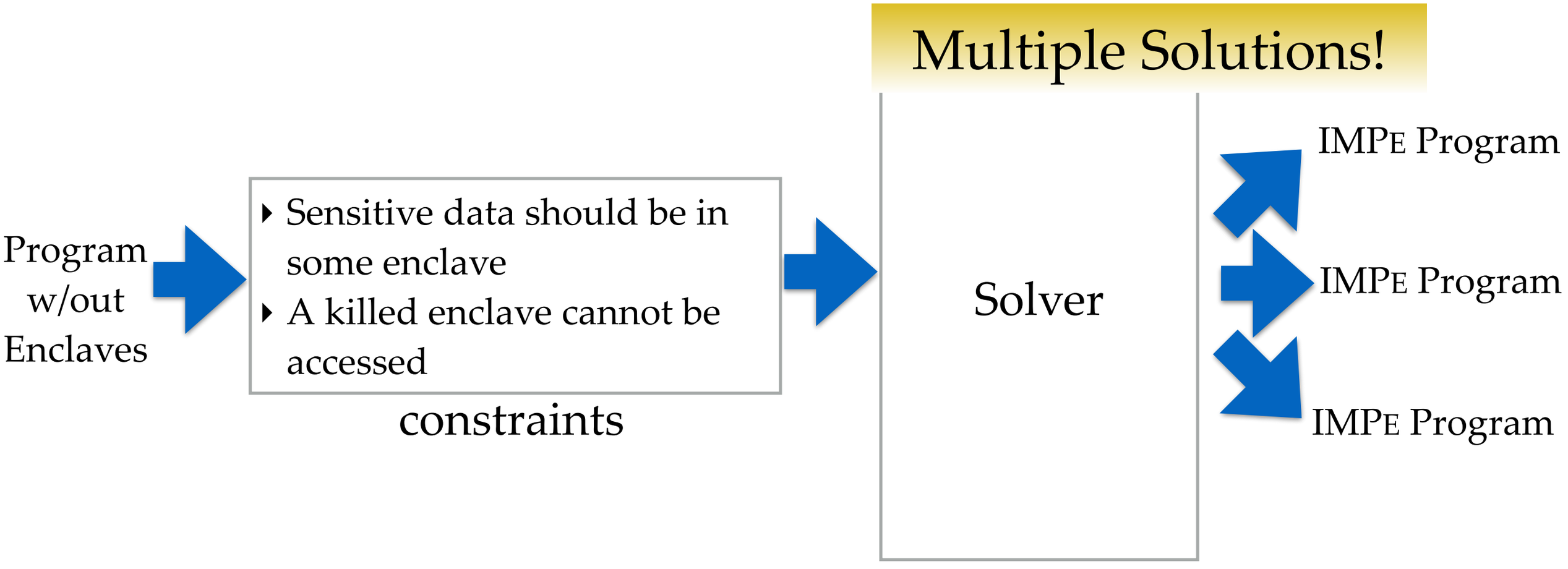
Solver

Enclave Inference as Constraint Satisfaction



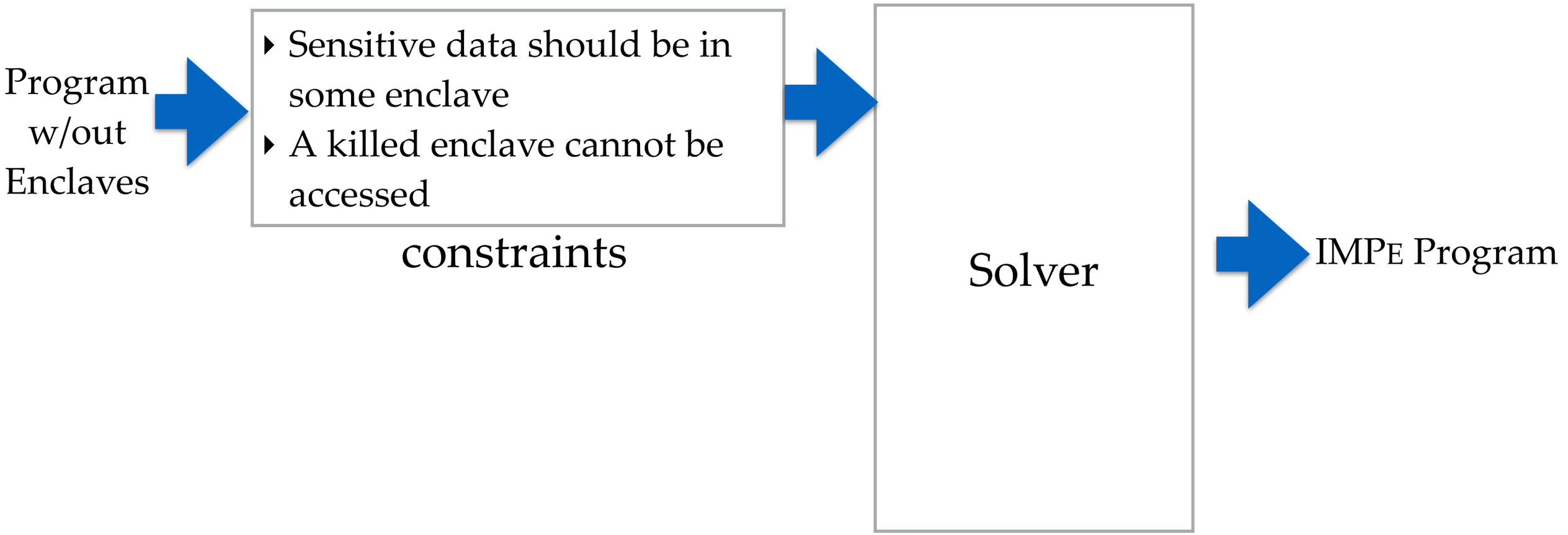
Solution is a well-typed IMPE program

Enclave Inference as Constraint Satisfaction

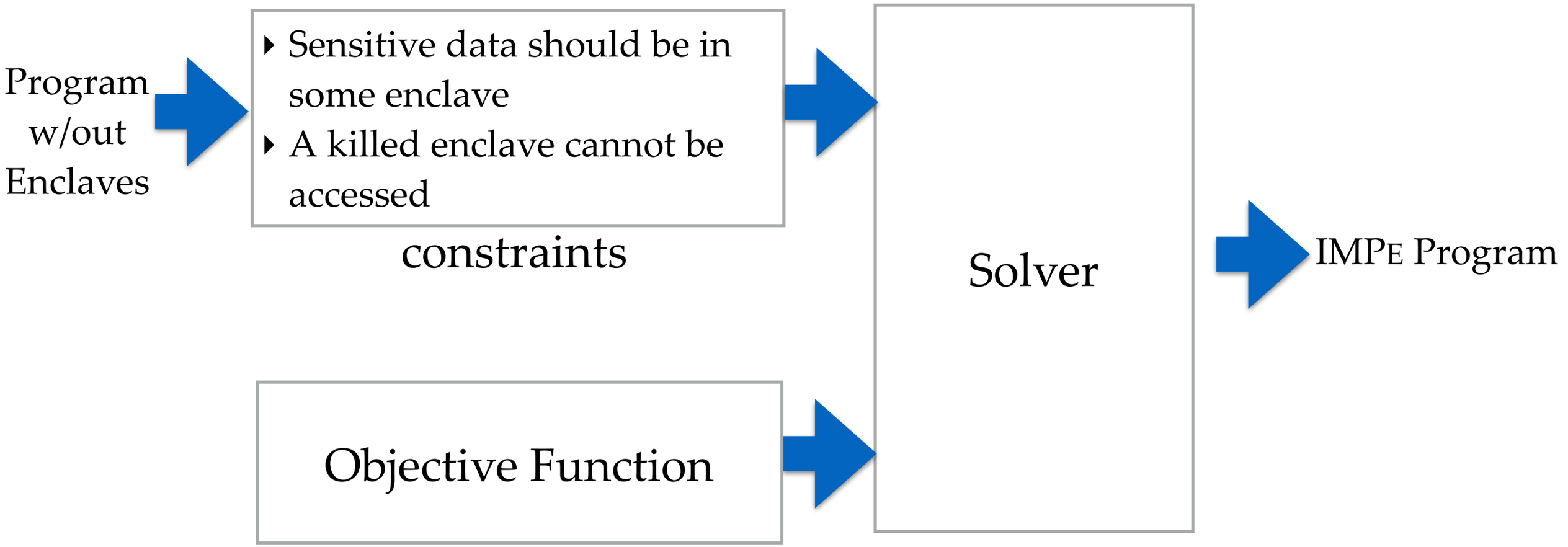


Solution is a well-typed IMPE program

Enclave Inference as Constraint **Optimization**



Enclave Inference as Constraint **Optimization**



Example Objective Functions

1. Minimize the TCB

   - Reduce the number of statements inside enclaves

Example Objective Functions

1. Minimize the TCB

   - Reduce the number of statements inside enclaves

2. Minimize the window of vulnerability

   - Place code and data in as many different enclaves as possible

   - Kill an enclave as soon as possible

Summary

Summary

1. Strong information-flow guarantees using hardware protection mechanisms (enclaves)

2. Automatically infer enclave placement in an application relieving the programmers' burden