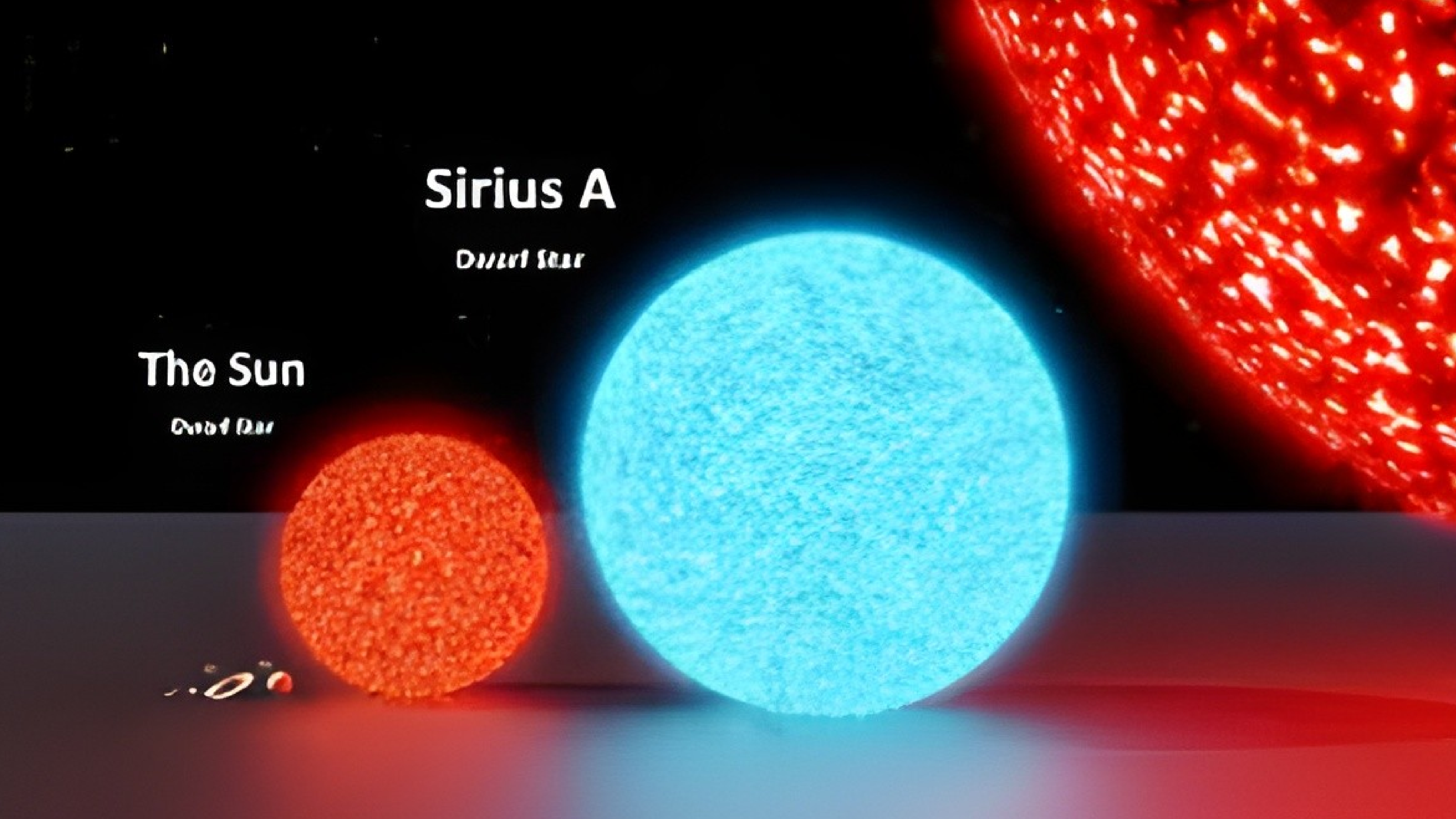
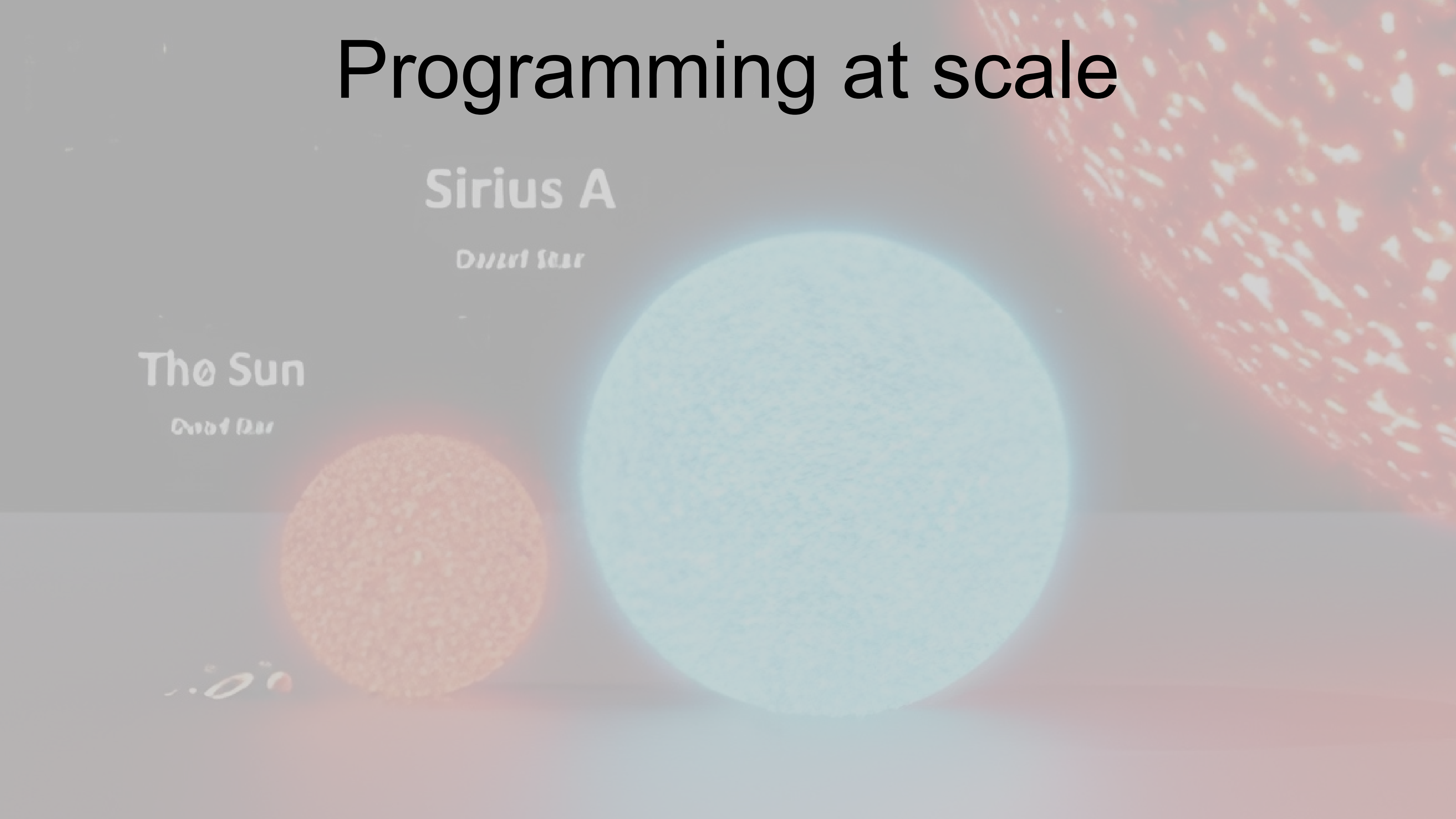
Enforced modular security

The clean garden security model is hopeless
We all live in the dark forest.


Sirius A
Tho Sun

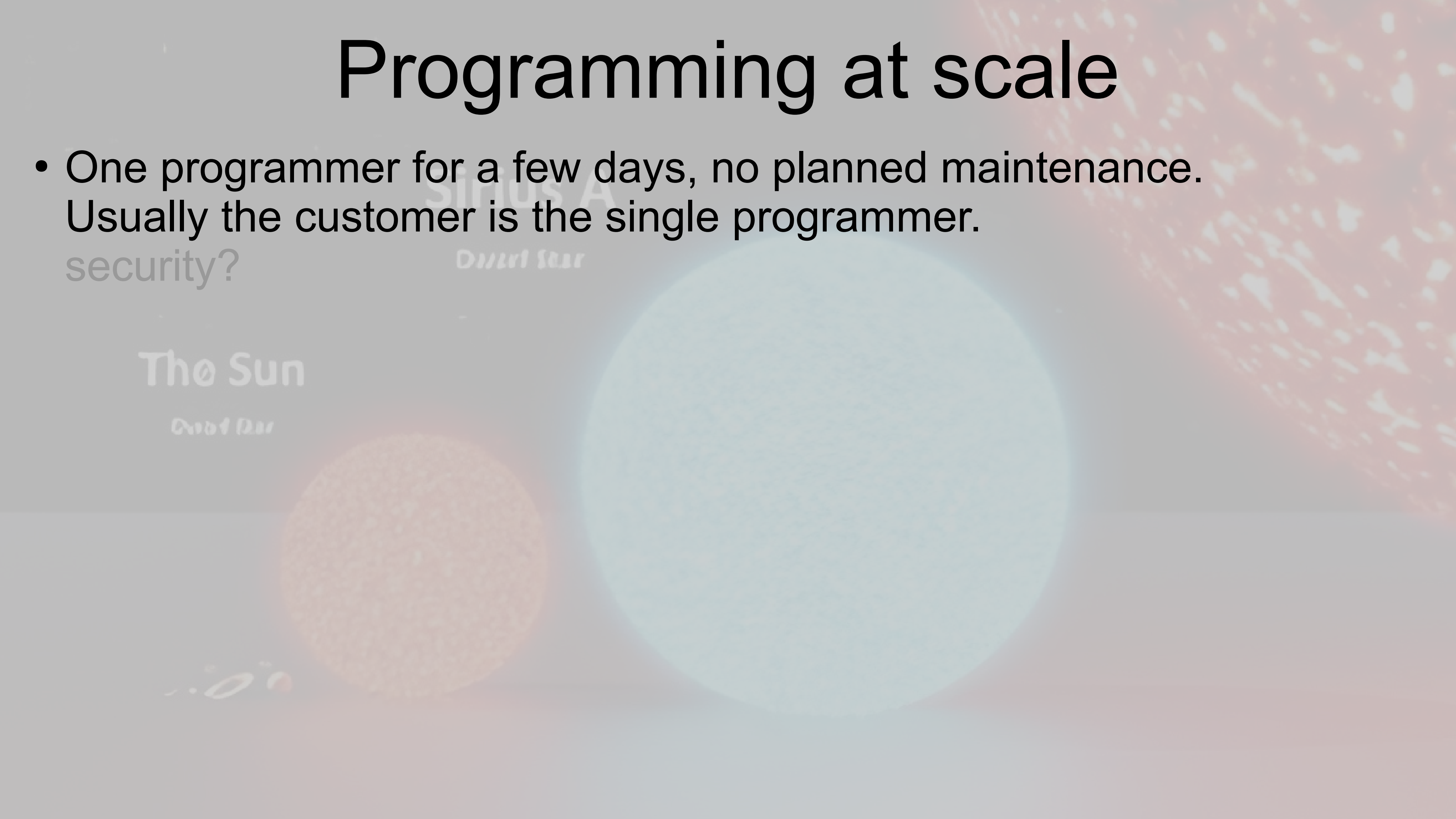
Sirius A
The Sun

Programming at scale

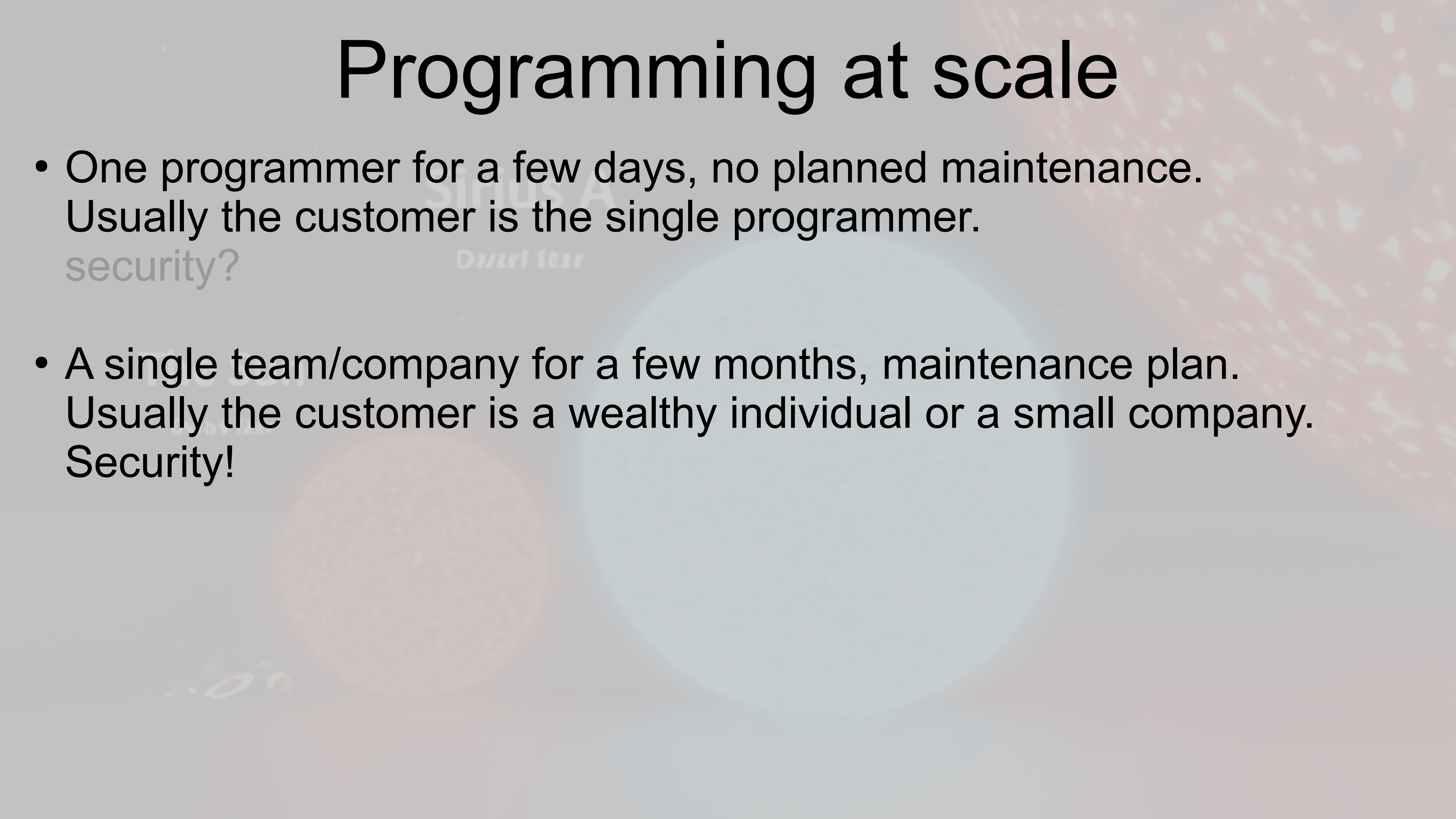


Programming at scale

- One programmer for a few days, no planned maintenance. Usually the customer is the single programmer. security?



Programming at scale



- One programmer for a few days, no planned maintenance. Usually the customer is the single programmer. security?

- A single team/company for a few months, maintenance plan. Usually the customer is a wealthy individual or a small company. Security!

Programming at scale

- One programmer for a few days, no planned maintenance.
  Usually the customer is the single programmer.
  security?

- A single team/company for a few months, maintenance plan.
  Usually the customer is a wealthy individual or a small company.
  Security!

- A community effort for many years, maintaining IS developing.
  Usually there is no clear 'customer', but a large user base.
  Libraries, frameworks, OS, big open source projects..
  SECURITY !

Security is not just network security

- **Fact**
  - We are getting better at network security
  - Compromising a system by sending tailored data is getting harder and harder.

Security is not just network security

- **Fact**

  We are getting better at network security

  Compromising a system by sending tailored data is getting harder and harder.

- **Fact**

  Hackers are not going to simply stop bothering.

  They are going to try to hack our code using other kinds of vulnerabilities.

Security is not just network security

- **Fact**

  We are getting better at network security

  Compromising a system by sending tailored data is getting harder and harder.

- **Fact**

  Hackers are not going to simply stop bothering.

  They are going to try to hack our code using other kinds of vulnerabilities.

Traditionally, security focused a lot on network security. Now that network security is finally getting stronger, hackers are using other ways

Software supply chain attacks

- Attackers can inject malicious code into one of our dependencies.

  For example, they become 'trusted' contributors to one of the open source libraries used by our system. In this way, when updating the library, we will silently import adversarial code in our security critical process.
- Eventually, some malicious code will make its way into our program.

Indeed, my interest in SW security started when as soon as my friend got to work in some bank security critical code, he spotted a back door left from a former programmer. My friend decided to report the issue and get it fixed. Someone else may have not been so honest.

Malicious code in the process.

- Malicious code is now running in our process.
- In most languages, at this point it is game over. The malicious code can do any action.

Malicious code in the process.

- Malicious code is now running in our process.
- In most languages, at this point it is game over.

  The malicious code can do any action.

- Can we modularly enforce security even in the presence of malicious code?

Freedom is Slavery (George Orwell's '1984')

- Small scale:
  programming is mostly to express behavior.
- Larger scale:
  programming is mostly to restrict behavior.

- Freedom is slavery!

Freedom is Slavery (George Orwell's '1984')

- Essential truth in the world of software engineering:
  more freedom in one aspect of the coding experience,
  often enslaves us in other aspects.
- We crave the freedom provided by flexible languages; this freedom can bind us in chains of complexity, vulnerability, and unanticipated interactions.
- Balancing between developer autonomy and the potential for chaos and vulnerability lies at the heart of software engineering and secure programming language design.

Examples:

Dynamic Typing in Programming Languages

Freedom: The ability to pass any object to any method is liberating, In particular no type parameters hell.
Slavery: No Types == more runtime errors, complicates code comprehension, obfuscates dependencies.

Global Variables

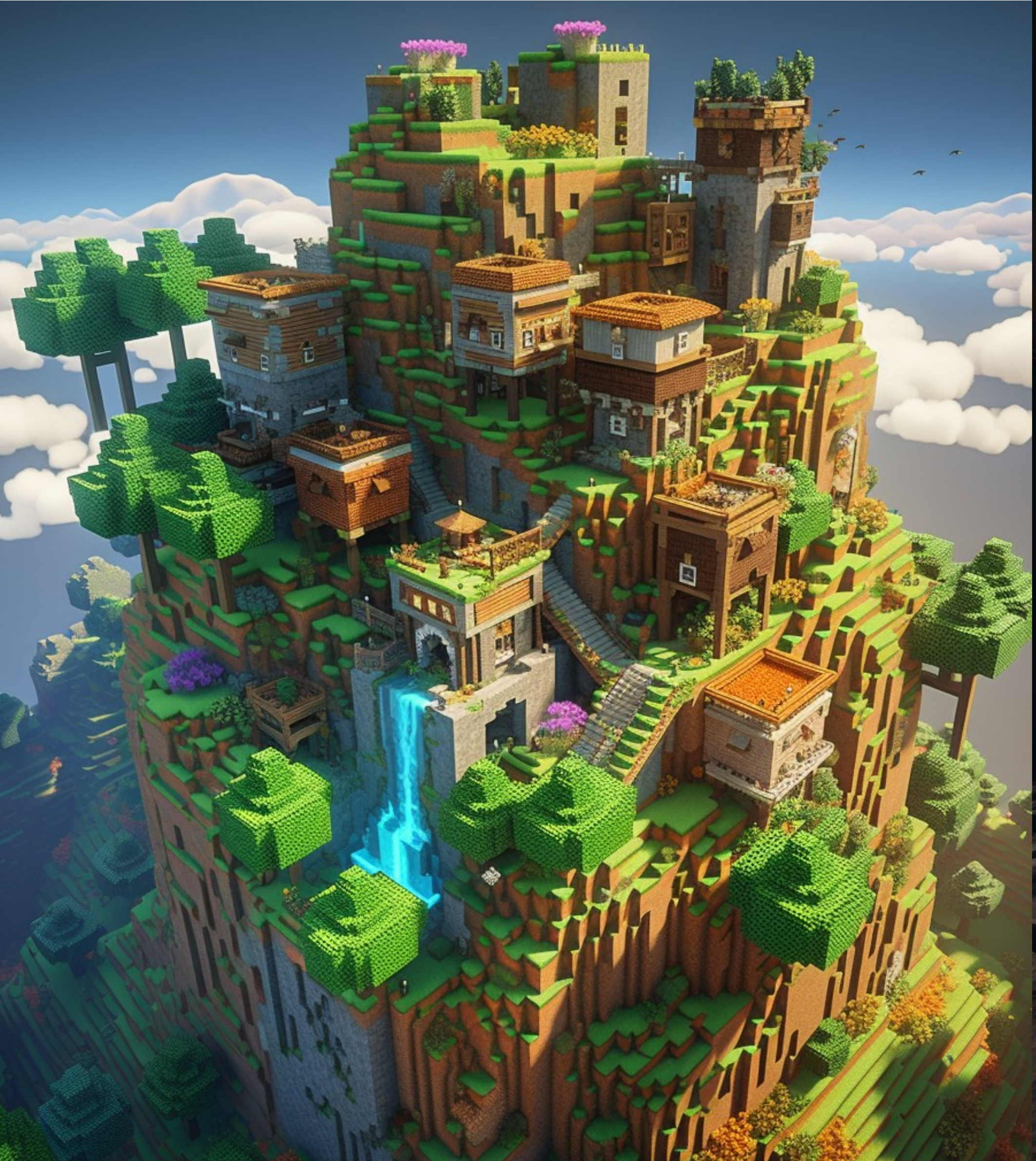
Freedom: Accessed / modified from anywhere, no need to pass data through layers of function parameters.
Slavery: State risks accidental (or intentional) alteration: elusive bugs and (unit) testing challenges.

Object-Oriented Programming (OOP) and Encapsulation

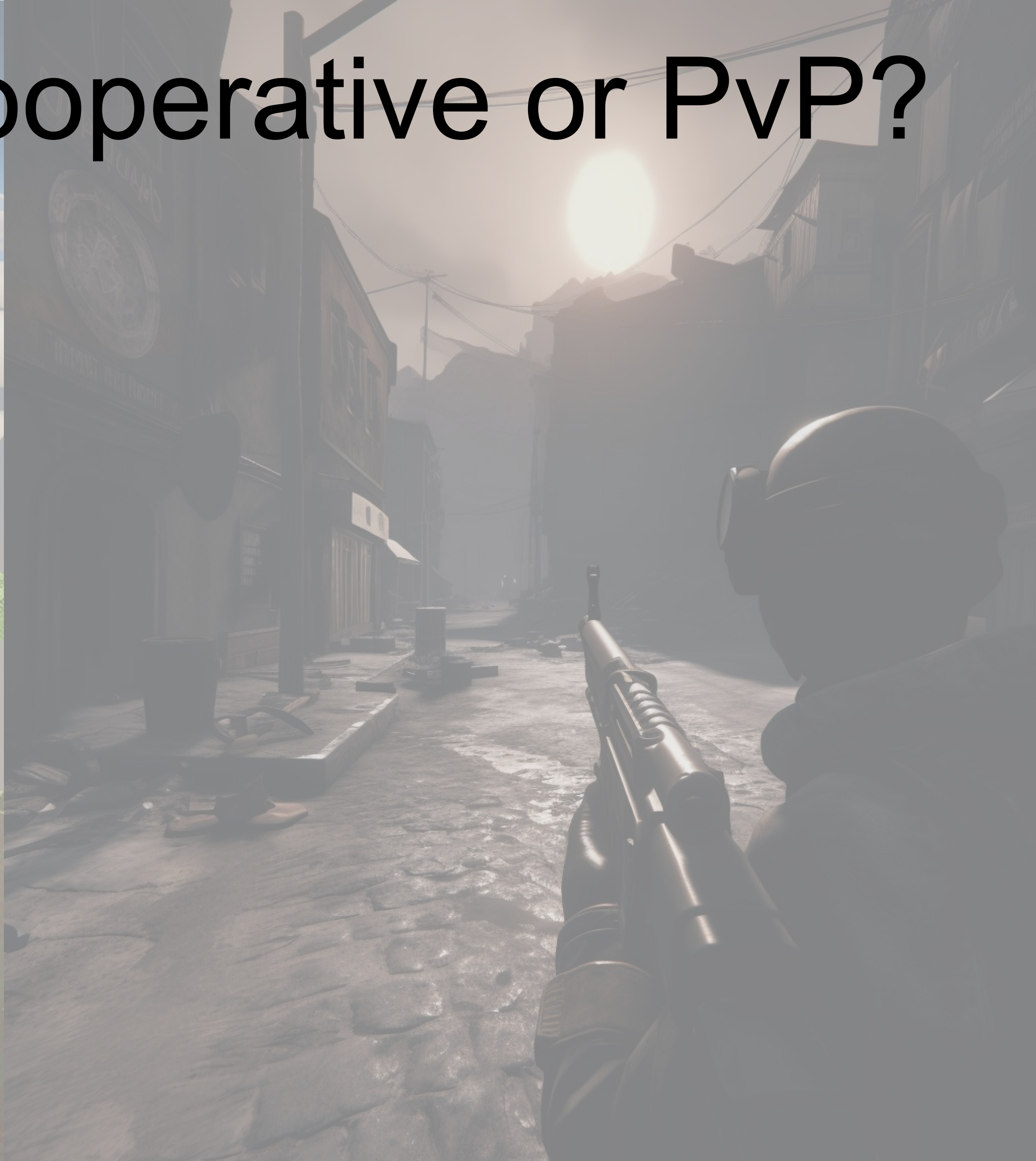
Freedom: With public constructors and static methods we can invoke operations all over the codebase.
Slavery: Any code, anywhere, can do everything. Entangled, difficult to debug code. Challenging in a team!

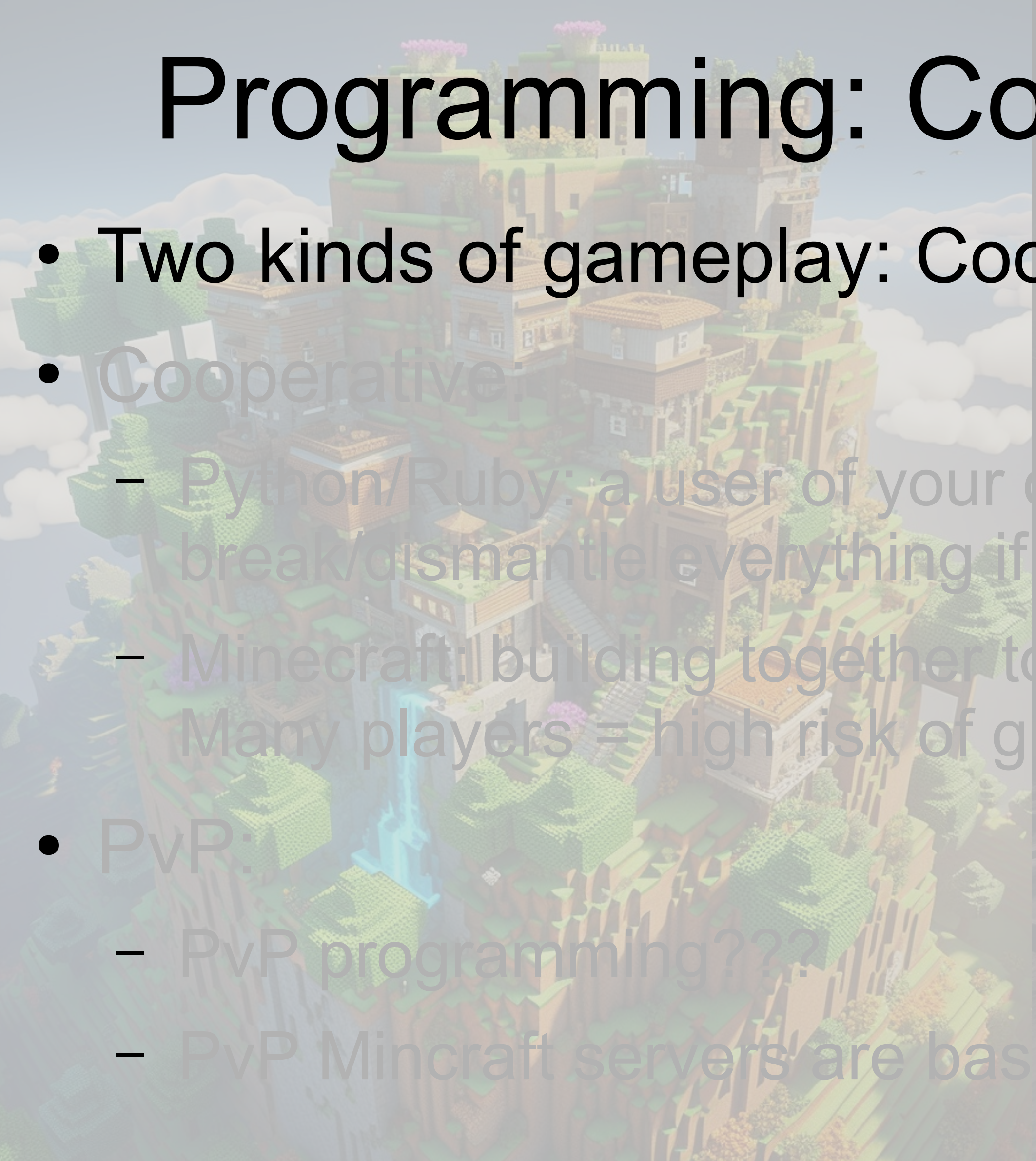
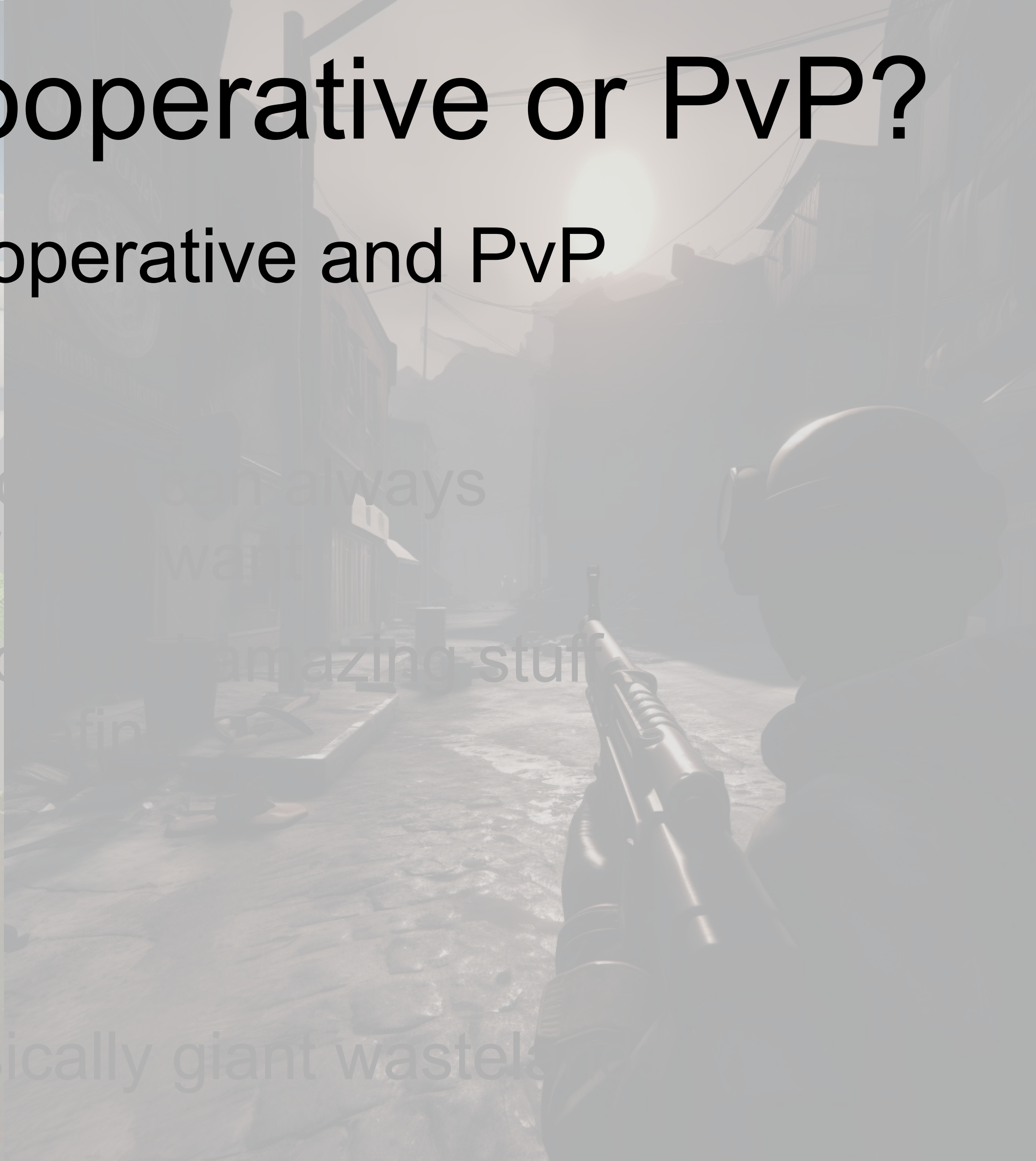
Permissive Access Controls

Freedom: Liberal access controls facilitate versatile code usage and reuse.
Slavery: This ease of access allows for more forms of misuse and is a massive liability during API evolution



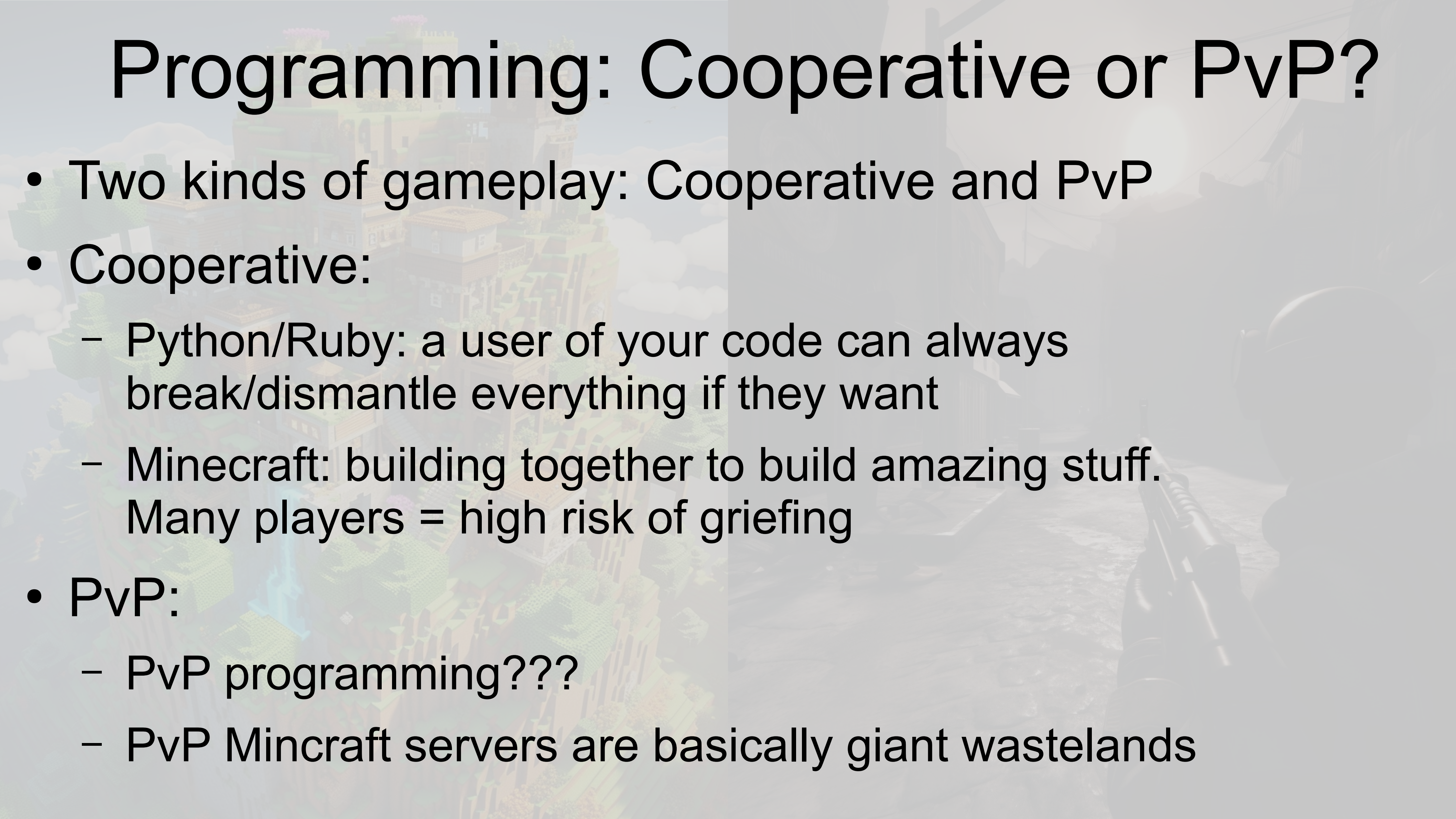
Programming: Cooperative or PvP?



Programming: Cooperative or PvP?

- Two kinds of gameplay: Cooperative and PvP
- Cooperative:
  - Python/Ruby: a user of your c[illegible] can always break/dismantle everything if [illegible] want
  - Minecraft: building together to [illegible] amazing stuff. Many players = high risk of g[illegible]fin[illegible]
- PvP:
  - PvP programming???
  - PvP Mincraft servers are basically giant wastel[illegible]



Programming: Cooperative or PvP?

- Two kinds of gameplay: Cooperative and PvP
- Cooperative:
  - Python/Ruby: a user of your code can always break/dismantle everything if they want
  - Minecraft: building together to build amazing stuff. Many players = high risk of griefing
- PvP:
  - PvP programming???
  - PvP Mincraft servers are basically giant wastelands

The C model delusion

- Ultimately everything is broken: just be sensible, practical and try to cooperate.
  - True in the old C/Shell/Low level-OS programming.

The C model delusion

- Ultimately everything is broken: just be sensible, practical and try to cooperate.
  - True in the old C/Shell/Low level-OS programming.
- In Java/C#/Haskell a bunch of things are actually guaranteed by the language.
  - Java has 'security bugs/fix' because it has a security model: some properties that they promise.
  - C/C++: no such thing as a security fix, nothing is actually promised by the language.

But I do not want restrictions when I code

- It is turtles all the way down!
- An assembly programmer may be against C preventing them from freely decide how to allocate registers.
- A C programmer may be against Python preventing them from freely handle memory management.
- A Python programmer may be against Java preventing them to handle self modifying code (monkey patching) and requiring more strict typing rules.
- A Java programmer may be against Haskell preventing them from freely handling I/O and mutation.
- A Haskell programmer may be against Agda preventing them from handling termination and well foundness of data-structures.

Designing languages for Modular security

- Can we make it **as easy as possible** to do simple tasks?
  - Misguided objective: Freedom is Slavery

Designing languages for Modular security

- Can we make it **as easy as possible** to do simple tasks?
  - Misguided objective: Freedom is Slavery

Instead

- Make it **impossible** to do certain kinds of **errors**!
  - Classify and mathematically model various kinds of errors.
  - The language semantic prevents those errors.
  - **Cost**: this language is now **harder** to use.

Offensive programming A variation of Defensive programming

- Statically enforcing representation invariants via runtime checks
  - As soon as something goes out of the expected behavior, throw error!

```java
/** Invariant:-there are no more then 10 allergies */
class Person {
  private List<String> allergies;
  public List<String> allergies(){ return allergies; }
  public void allergies(List<String> a){
    if(a.size() > 10){ throw new Error(); }
    allergies = List.copyOf(a);
  }
  Person(List<String> a){ allergies(a); }
}
```

Modular Offensive programming → Modular Security

If the user tries to forge an invalid object, the code fails with an error!

Modular Offensive programming → Modular Security

If the user tries to forge an invalid object, the code fails with an error!

- **Modular Offensive programming: (**Here M can be a library and C the user code)
  - The module M can be placed in any context C and code running under the control of M will either answer according to its specification or throw an error with the system being in a consistent state.

Modular Offensive programming → Modular Security

If the user tries to forge an invalid object, the code fails with an error!

- **Modular Offensive programming: (**Here M can be a library and C the user code)
  - The module M can be placed in any context C and code running under the control of M will either answer according to its specification or throw an error with the system being in a consistent state.
- **Modular security:**
  - The module M can be placed in any context C and code running under the control of M will never do the wrong thing/action **(not the same concept of 'computation actions', see later)**

Modular Offensive programming → Modular Security

If the user tries to forge an invalid object, the code fails with an error!

- **Modular Offensive programming: (**Here M can be a library and C the user code)
  - The module M can be placed in any context C and code running under the control of M will either answer according to its specification or throw an error with the system being in a consistent state.
- **Modular security:**
  - The module M can be placed in any context C and code running under the control of M will never do the wrong thing/action **(not the same concept of 'computation actions', see later)**
- **Contrasting forces**
  - Certainty of constraints
  - Simplicity to express constraints
  - Range of expressible constraints

Modular Offensive programming → Modular Security

If the user tries to forge an invalid object, the code fails with an error!

- **Modular Offensive programming: (**Here M can be a library and C the user code)
  - The module M can be placed in any context C and code running under the control of M will either answer according to its specification or throw an error with the system being in a consistent state.
- **Modular security:**
  - The module M can be placed in any context C and code running under the control of M will never do the wrong thing/action **(not the same concept of 'computation actions', see later)**
- **Contrasting forces / My hope**: [SCP Dec 2022]
  - Certainty of constraints → up to uncompromised root
  - Simplicity to express constraints → just write code checking your constraints
  - Range of expressible constraints → must be expressible from the object ROG

Modular Offensive programming → Modular Security

If the user tries to forge an invalid object, the code fails with an error!

- Java: possible*, but the more complex the invariant or the data structure,
  the harder it gets to enforce invariants.
  Python: user code can always disassemble and tweak any other code
  C/C++: user code can always send the whole system into undefined behavior

Modular Offensive programming → Modular Security

If the user tries to forge an invalid object, the code fails with an error!

- Java: possible*, but the more complex the invariant or the data structure,
  the harder it gets to enforce invariants.
  Python: user code can always disassemble and tweak any other code
  C/C++: user code can always send the whole system into undefined behavior

possible*?

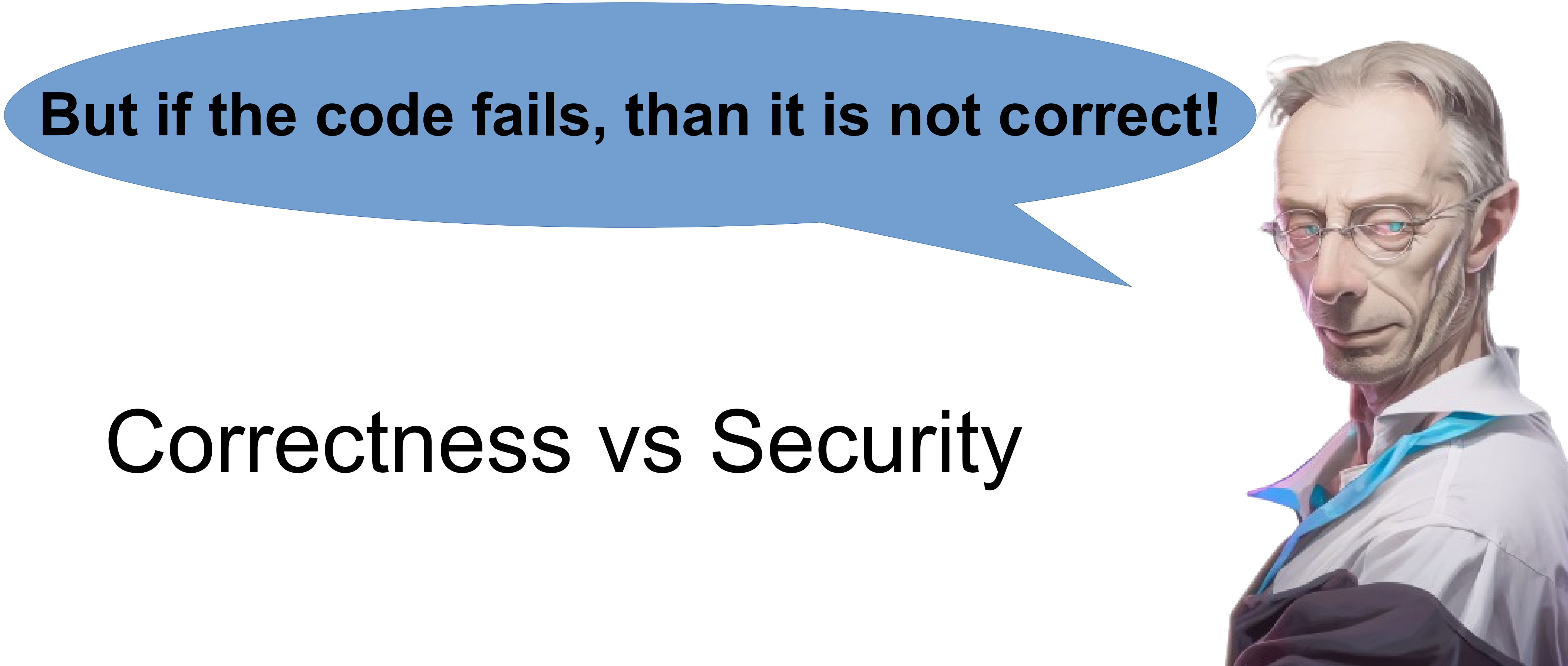
- very unpractical in the general case
- Internal constraints offers no protection from actions (I/O etc)
- False if you load broken C code

Even then, if arbitrary adversarial code enters in the system

A real PvP scenario = The desolate wasteland

Modular Offensive programming → Modular Security

If the user tries to forge an invalid object, the code fails with an error!



Correctness vs Security

Correctness vs Security

- **Correctness**

  the code always does the right thing
- **Security**

  the code never does the wrong thing

  Different scopes; example:
- Correctness: the code is right up to stack/memory overflow
- Security: the code is right up to cosmic ray bitflip/hardware bug

Correctness vs Security

- **Crucial point:** the code may simply not do any action.

Correctness vs Security

- **Crucial point:** the code may simply not do any action.
- Example: a secure but not correct implementation for anything would be

```
throw new Error("Nope");
```

Correctness vs Security

- **Crucial point:** the code may simply not do any action.
- Example: a secure but not correct implementation for anything would be

```
throw new Error("Nope");
```

- Example: a correct but not secure implementation for anything would be

```
int doStuff(int x){
   try{rec(aNumber);}
   catch(StackOverflowError o){ formatHD(); }
   return correctlyDoStuff(x);
   }
void rec(int i){if(i==0){return;} rec(i-1);}
```

What does it mean to do an action?

- In this context, an action is returning a result, but also doing side effects like I/O. Non termination and exceptions are not actions in this setting.

Non termination:
the code is taking an unlimited amount of time to act

Exception: the code refuses to act with a specific reason

What does it mean to do an action?

- In this context, an action is returning a result, but also doing side effects like I/O. Non termination and exceptions are not actions in this setting.
- Most languages allow any piece of code to do any actions (for example I/O).
- Typically this happens by indirectly calling native static methods.
- If static methods/fields are forbidden, we can get a more pure OO setting where all behavior is obtained by method calls to objects.

What does it mean to do an action?

- In this context, an action is returning a result, but also doing side effects like I/O. Non termination and exceptions are not actions in this setting.
- Most languages allow any piece of code to do any actions (for example I/O).
- Typically this happens by indirectly calling native static methods.
- If static methods/fields are forbidden, we can get a more pure OO setting where all behavior is obtained by method calls to objects.
- Put simply, if there’s no object with the capability to do the task, the task can’t be done.

The rule of Object Capabilities

NO OBJECT

NO ACTION

Java with Object capabilities

- For example, in Java we can do

```
System.out.println("...");
```

- This relies on the static field out and can be accessed everywhere
- An Object capability variant of Java could look like

```
void main(System s){ s.out.println("..."); }
```

- And 'System' would have no public constructors.

Object capabilities in a secure language

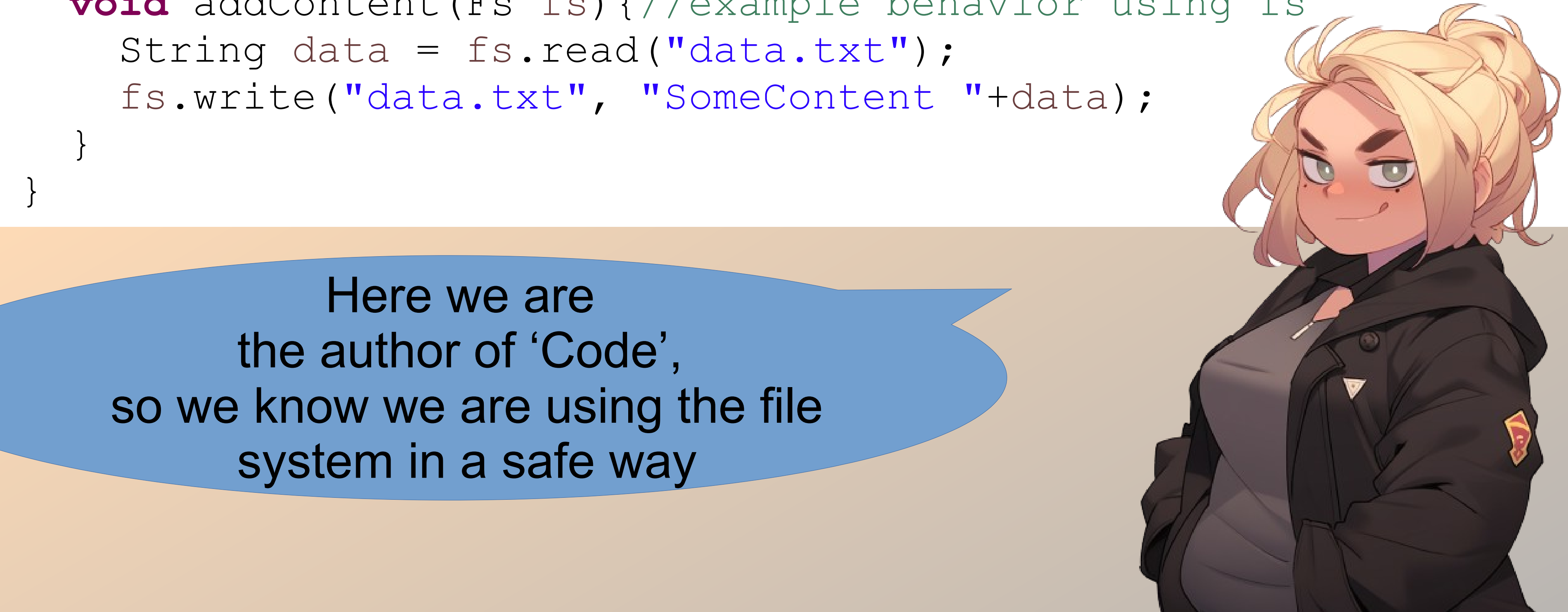
```
interface Fs{
  String read(String fileName);
  void write(String fileName, String content);
  void formatHD();//many more possible methods here
  void close();
}
private class RealFs implements Fs{
  public String read(String fileName){/*magic!*/}
  public void write(String fileName, String content){/*magic!*/}
  public void formatHD(){/*magic!*/}
  public void close(){/*magic!*/}
  private RealFs(){/*magic!*/}
}
```

-Fs is an interface
-RealFs is a private class with a private constructor with native method implementation for the methods; programmers can only get an instance of RealFs by calling methods on other Object capabilities.

Object capabilities in a secure language

- Example: File System access

```
class Code{
  void addContent(Fs fs){//example behavior using fs
    String data = fs.read("data.txt");
    fs.write("data.txt", "SomeContent "+data);
  }
}
```



Object capabilities in a secure language

- Example: File System access

```
class Code{
  void addContent(Fs fs){
    //using a library expecting the same behavior

    new TrustMe().act(fs,     "data.txt","SomeContent");

  }
}
```

We are not the
authors of ‘TrustMe’.
What if we do not trust ‘TrustMe’ with
all the power of the File System?

Object capabilities in a secu age

We can make a wrapper file system to limit it

- Example: File System access

```
class Code{
  void addContent(Fs fs){
    //using a library expecting the same behavior
    Fs safeFs = new GuardFs(fs);
    new TrustMe().act(safeFs,"data.txt","SomeContent");
    safeFs.close();
  }
}
```

We are not the authors of ‘TrustMe’.
What if we do not trust ‘TrustMe’ with all the power of the File System?

Object capabilities in a secure language

```
class GuardFs extends ErrorFs{//ErrorsFs implements Fs and just throws error
  private Optional<String> usedName = Optional.empty();
  private Optional<Fs> inner;
  public GuardFs(Fs fs) { inner = Optional.of(fs); }
  public String read(String fileName){
    assert usedName.isEmpty();
    usedName = Optional.of(fileName);
    return inner.get().read(fileName);
  }
  public void write(String fileName,String content){
    assert usedName.equals(Optional.of(fileName));
    inner.get().write(fileName, content);
  }
  public void close(){ inner = Optional.empty(); }
}
```

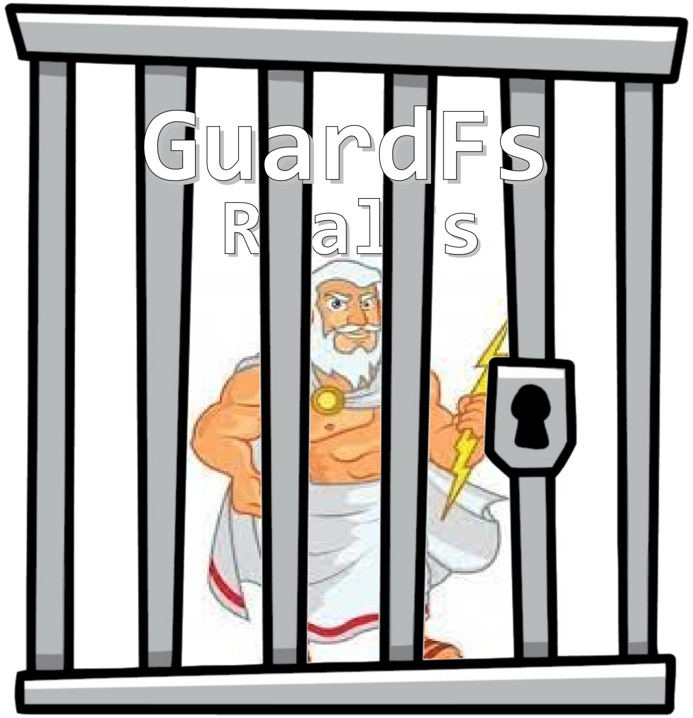
- Our GuardFs could override read and write to make sure they are used as we expect

Restricting the power of Fs

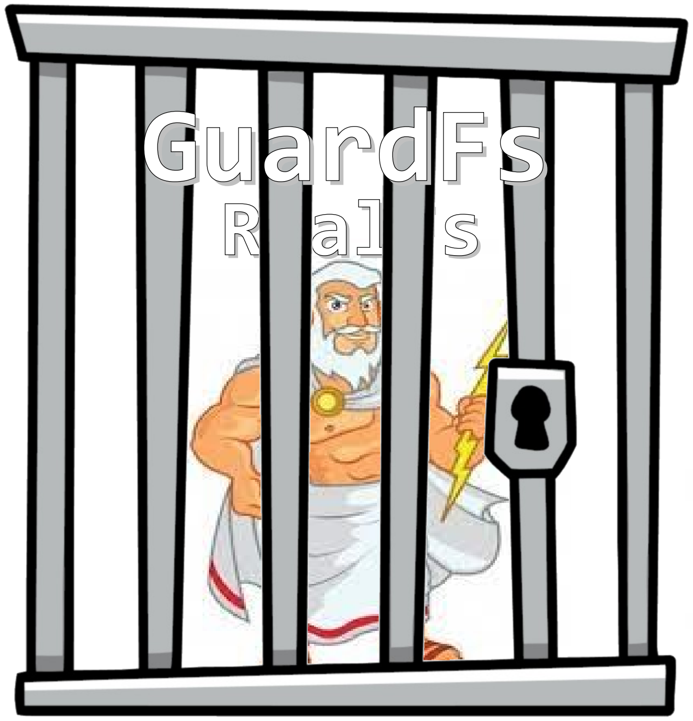
RealFs

Restricting the power of Fs

Restricting the power of Fs

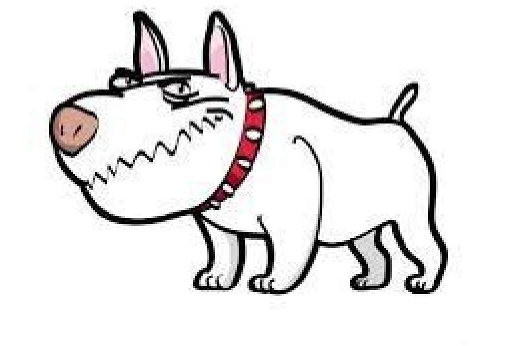


Restricting the power of Fs



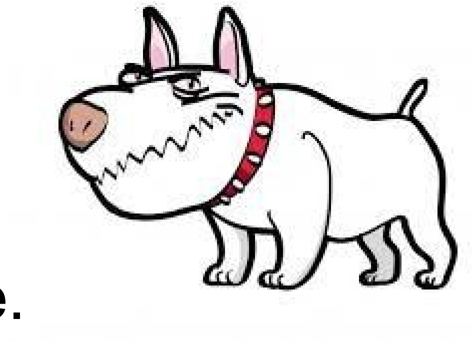
Delegate the methods on the captive capability,

Throw errors as soon as something unexpected happens



Restricting the power of Fs

- Now ‘TrustMe’ does not have the permission to write files. Even if ‘TrustMe’ is malicious, the ‘RealFs’ instance is encapsulated into a watchdog object that only allows them to use it as expected
  - TrustMe is unable to create another RealFs instance.
  - TrustMe is unable to freely access the RealFs instance from the GuardFs instance.
  - If there are no static fields, TrustMe can not use those to access other RealFs instances.
  - TrustMe can not access the FileSystem without going through an RealFs instance.



Three roles for programmers

- **Security architects:** they write capability classes, like ‘GuardFs’.
  Those classes can wrap and refine other capabilities.
  - Any security condition that can be expressed as a check performed by a wrapper object can be modularly enforced.
- **Main programmer:** they instantiate capabilities written by Security architects and they pass it to untrusted code.
- **Regular programmers:** they write most of the code, and they are responsible to write correct$^{\text{(enough)}}$ code but they do not need to worry about any security consideration.
  - If they 'somehow can' do an action, that is a valid action.

Third party libraries/untrusted code

- Most libraries will not contain capability classes, but they will take in input capability objects.
- Those libraries will not be able to do anything outside of the security actions approved by the main programmer.

- Few core libraries will contain capability classes and their code will need to be manually verified.

It can't be done in existing languages

- Even when coding in other languages, having a self contained part of the code defining security actions would be a good programming practice.

  This would statistically reduce the amount of security bugs in your application, but it would be no match against carefully crafted adversarial code.

- Requirement for capabilities:
  - Capability classes must not be freely instantiable
  - Reflection and other tricks must not allow to access hidden fields
  - Static fields may become a hidden communication channel between multiple untrusted code units.
  - Full access to native code must be restricted.

Representation invariants → Security

- If a language enforce that object invariants can never be observed broken, we get a boost:
- Object invariants can enforce security

  If DBAccess has invariant and we need a DBAccess to access the DB,

  then only valid DBAccess objects will access the DB
- Invariants can encode the security requirements

Reference capabilities → Security

- Reference capabilities (eg. mut, read, imm, iso, capsule, lent,…) cooperate greatly with Object Capabilities.
  - A language with reference capabilities can design the standard library so that all ‘capability’ methods of capability objects are ‘mut’. That is, they conceptually ‘mutate’ the outer world.
  - The main will then receive a parameter of type **`mut System`**.
  - All the non deterministic operations will be available as capability objects.

- In this world, we gain amazing reasoning power:
  All methods that take in input only immutable and readonly data are deterministic!

This is the end for me!