Crystal, el lenguaje de programación sin burocracia

Beta Ziliani

Universidad ORT Uruguay

Crystal Core Team Member

Un poco de mi

Academia

Demostración interactiva de teoremas

Semántica de lenguajes de programación

Industria

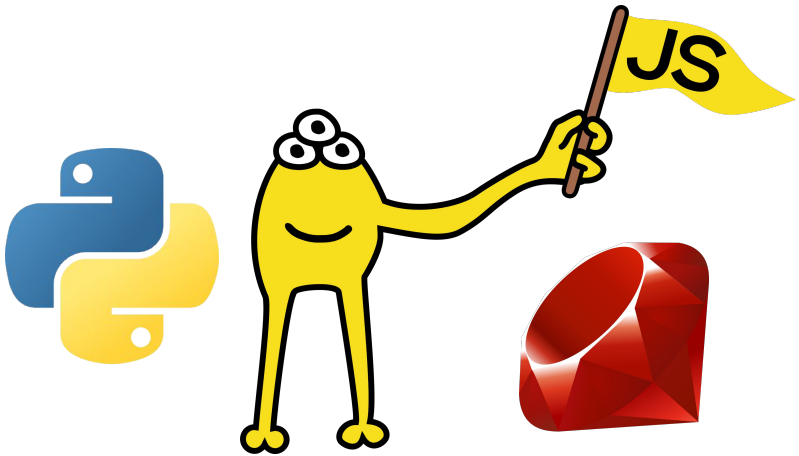
Compilador Crystal y su ecosistema

Herramientas para Solidity

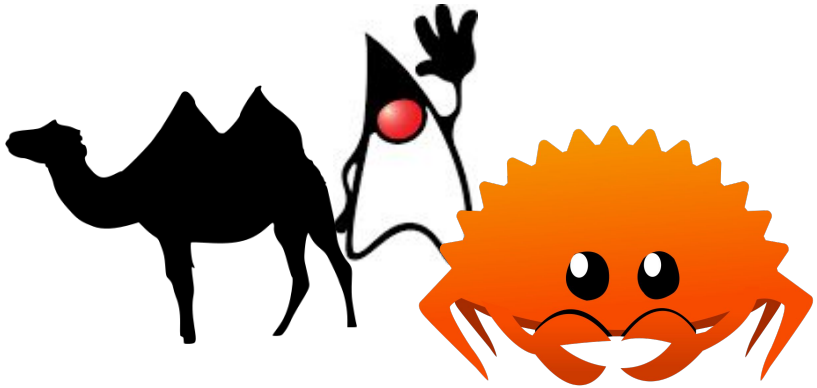
CRYSTAL

El zoológico de lenguajes de programación

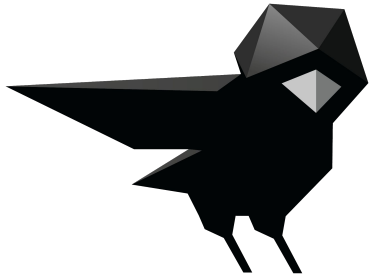
Lenguajes dinámicos



Lenguajes estáticos



CRYSTAL



Un nuevo lenguaje entró al zoológico, ¿dónde lo ponemos?

CRYSTAL

La ventaja de los lenguajes dinámicos (Ruby)

CRYSTAL

Comportamiento dinámico: *duck typing*

```ruby
class Pato
  def cuack
    "🦆 hace cuack!"
  end
end

class Loro
  def cuack
    "🦜 hace cuack!"
  end
end

if rand(2) >= 1     # 👍🪙
  animal = Pato.new
else
  animal = Loro.new
end

puts animal.cuack
```

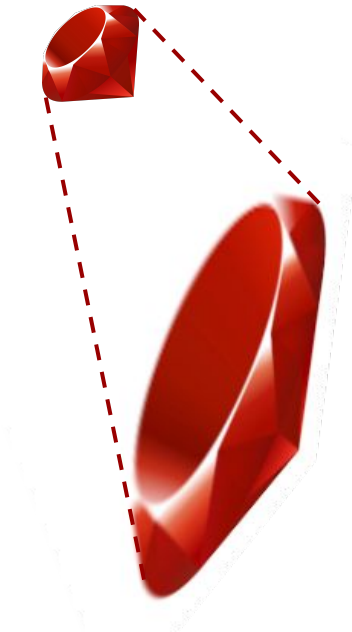
“Si camina como pato y habla como pato… ¡es un pato!”

CRYSTAL

Los lenguajes dinámicos son flexibles

Prototipado rápido

- 🦆 Duck typing
- 🐒 Monkey patching



¿Está todo bien con esto?

CRYSTAL

Duck typing — ¡mas bien, no typing!

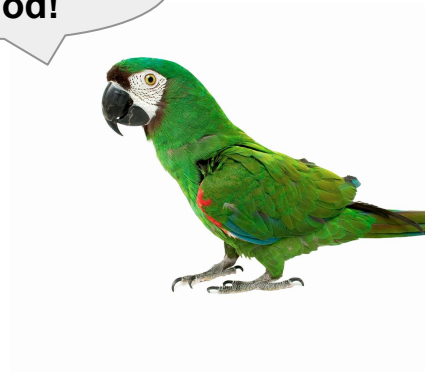
¡Puede no hacer cuack!

```
class Pato
  def cuack
    "🦆 hace cuack!"
  end
end


if rand(2) >= 1
  animal = Pato.new
else
  animal = Loro.new
end

puts animal.cuack
```

```
class Loro
#  def cuack
#    "🦜 hace cuack!"
#  end
# Loro no hace cuack!
# Tests pass ✅
end
```

in `<main>': undefined method `cuack for #<Loro:0x000000010984cad8> (NoMethodError)

CRYSTAL

El dinamismo es divertido pero inseguro

CRYSTAL

La seguridad de los lenguajes estáticos (Java)

CRYSTAL

“No hiciste tu papeleo”

- Si se quiere hacer cuack:
  - Declarar interface *HaceCuack*
  - Declarar animal como *HaceCuack*

CRYSTAL

“No hiciste tu papeleo”

```java
interface HaceCuack {
    String cuack();
}

class Pato implements HaceCuack {
  public String cuack() {
    return "🦆 hace cuack!";
  }
}

class Loro implements HaceCuack {
  public String cuack() {
    return "🦜 hace cuack!";
  }
}

public class Main {
  public static void main(String args[]) {
    HaceCuack animal = null;
    if (new java.util.Random().nextBoolean()) {  // 👍🪙
      animal = new Pato();
    } else {
      animal = new Loro();
    }
    System.out.println(animal.cuack());
  }
}
```

(se puede hacer esto en Ruby también)

CRYSTAL

El papeleo salva vidas

```
interface HaceCuack {
   String cuack();
}

class Pato implements HaceCuack {
 public String cuack() {
   return "🦆 hace cuack!";
 }
}

public class Main {
 public static void main(String args[]) {
   HaceCuack animal = null;
   if (new java.util.Random().nextBoolean()) {  // 👍🪙
     animal = new Pato();
   } else {
     animal = new Loro();
   }
   System.out.println(animal.cuack());
 }
}
```

```
class Loro implements HaceCuack {
^ error: Loro is not abstract and does not
override abstract method cuack()
 // public String cuack() {
 //   return "🦜 hace cuack!";
 // }
}
```

Error de compilación

CRYSTAL

… ¡y tiempo de compilación! – Compilación modular

```java
class Pato implements HaceCuack {
 public String cuack() {
   return "🦆 hace cuack!";
 }
}
```

pato.java

01110000111000110110000 1

pato.o

```java
class Loro implements HaceCuack {
 public String cuack() {
   return "🦜 hace cuack!";
 }
}
```

loro.java

01010100011001110010010 1

loro.o

```java
public class Main {
 public static void main(String args[]) {
   ...
 }
}
```

main.java

10011100010110010010100 0

main.o

… ¡y tiempo de compilación! – Compilación modular

```
class Pato implements HaceCuack {
 public String cuack() {
   return "🦆 hace cuack!!!";
 }
}
```

pato.java

01110000111001001100001

pato.o

```
class Loro implements HaceCuack {
 public String cuack() {
   return "🦜 hace cuack!";
 }
}
```

loro.java

01010100011001110010010 1

loro.o

```
public class Main {
 public static void main(String args[]) {
   ...
 }
}
```

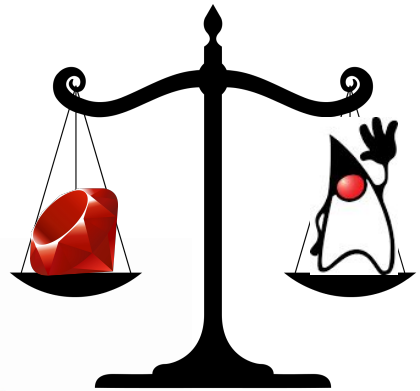
main.java

100111000101100100101000

main.o

Entonces...

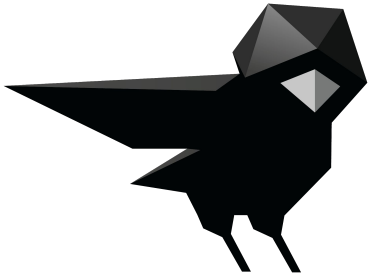
- Los lenguajes dinámicos son flexibles pero inseguros
  - No tienen compilación



- Los lenguajes estáticos son restrictivos pero seguros
  - Admiten compilación modular

¿Son estas las únicas dos opciones?

CRYSTAL



¡Conozcan a Crystal!

CRYSTAL

Un lenguaje tipado y compilado

¡Flexible, pero con una jaula segura!

CRYSTAL

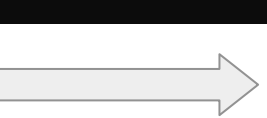
Duck typing con *Tipos Union*

```
class Pato
  def cuack
    "🦆 hace cuack!"
  end
end
```

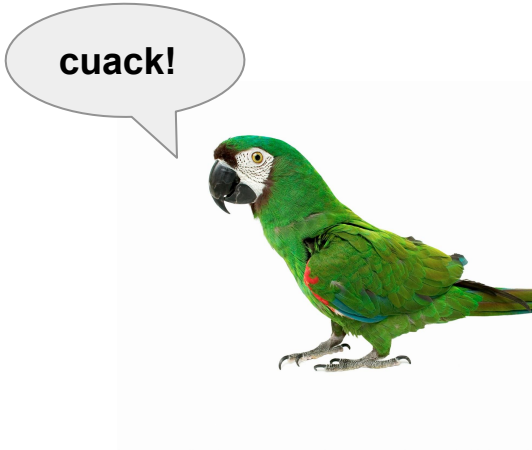
```
class Loro
  def cuack
    "🦜 hace cuack!"
  end
end
```

```
if rand(2) >= 1    # 👍🪙
  animal = Pato.new
else
  animal = Loro.new
end

puts animal.cuack
```



```
animal : Pato | Loro
```



“Si camina como pato y habla como pato… ¡es un pato!”

CRYSTAL

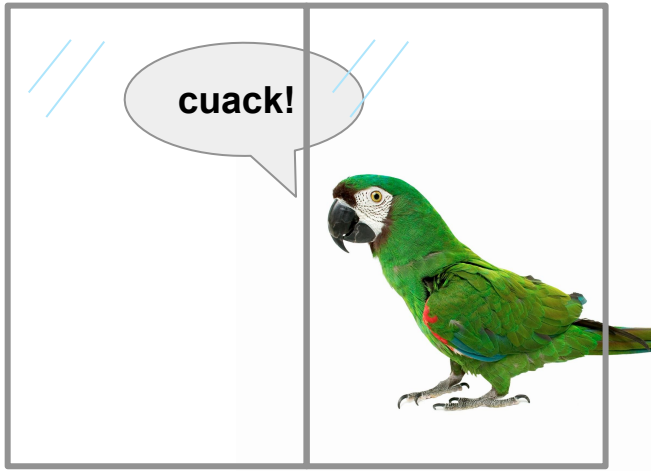
Duck *safe* typing

```
class Pato
  def cuack
    "🦆 hace cuack!"
  end
end
```

```
class Loro
#  def cuack
#    "🦜 hace cuack!"
#  end
# Loro no hace cuack!
end
```

```
if rand(2) >= 1
  animal = Pato.new
else
  animal = Loro.new
end

puts animal.cuack
     ^----
undefined method 'cuack' for Loro (type is (Pato | Loro))
```



Error de compilación

CRYSTAL

¡Flexible, seguro, y eficiente!

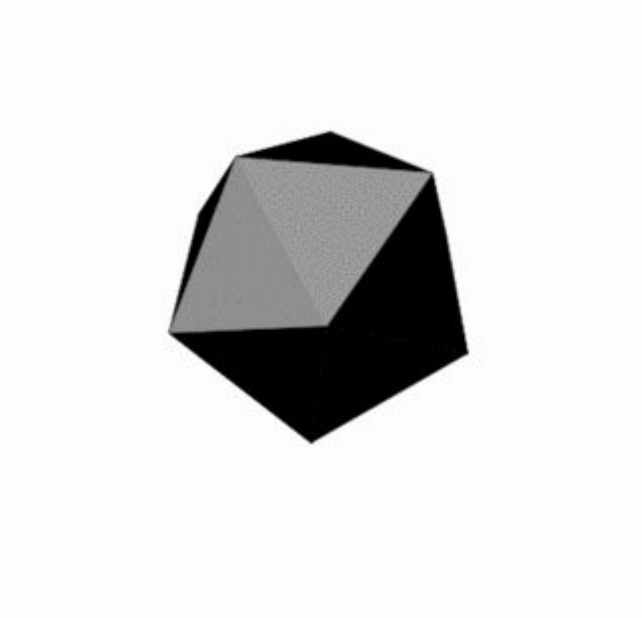
java	2.java	452ms	2.2ms	574.3MB	543ms	87ms	openjdk 21
java	2.java	463ms	5.2ms	586.3MB	583ms	70ms	openjdk 23
kotlin	1.kt	483ms	25ms	1072.1MB	547ms	97ms	kotlin/jvm 21
java	2-m.java	509ms	28ms	649.0MB	737ms	97ms	graal/jvm 17.0.8
dart	1.dart	705ms	2.1ms	81.1MB	670ms	40ms	dart/exe 3.2.4
v	1.v	811ms	3.9ms	37.1MB	790ms	7ms	v/clang+gc 0.4.3
nim	2.nim	837ms	7.9ms	34.4MB	820ms	3ms	nim 2.0.2
javascript	1-m.js	928ms	12ms	211.9MB	1397ms	303ms	bun 1.0.20
nim	2.nim	1181ms	11ms	34.4MB	1157ms	7ms	nim/clang 2.0.2
java	2.java	1191ms	8.1ms	1047.2MB	580ms	773ms	openjdk/zgc 21
rust	4.rs	1303ms	14ms	33.8MB	1277ms	13ms	rustc 1.75.0
csharp	1.cs	1307ms	155ms	130.5MB	1503ms	210ms	dotnet/aot 8.0.100
crystal	1.cr	1319ms	10.0ms	64.4MB	1300ms	3ms	crystal 1.10.1

+30

CRYSTAL

Source https://programming-language-benchmarks.vercel.app/problem/binarytrees

Nota: ¡Desconfiar en los benchmarks!



El secreto de Crystal

CRYSTAL

Tipos union y *method dispatch*


```
if rand(2) >= 1
 animal = Pato.new
else
 animal = Loro.new
end

puts animal.cuack
```

se traduce a

```
if animal.type_id == Pato.type_id
 _tmp = __Pato_cuack(animal.as(Pato))
else  # not Pato ≡ Loro
 _tmp = __Loro_cuack(animal.as(Loro))
end

puts _tmp
```

Sin vtable

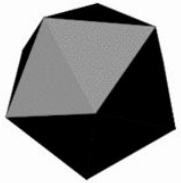
`animal : Pato | Loro`

type_id
fields (max size)

(similar a `union` en C)

CRYSTAL

Inferencia de tipos en llamada a función



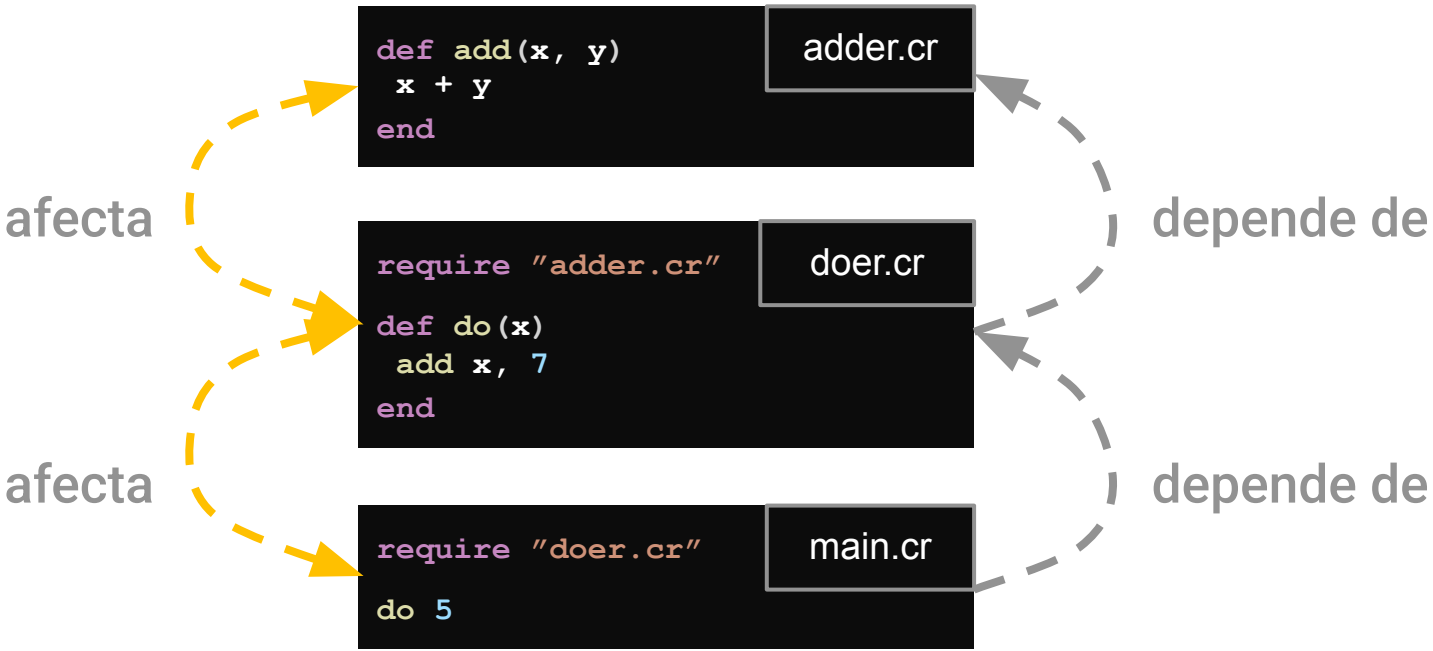
```
def add(x, y)
 x + y
end

add 5, 7                        # add(x : Int, y : Int) : Int
add "crystal ", "es genial"     # add(x : String, y : String) : String
```

No importa como se *define*
la función, si no como *se invoca*

CRYSTAL

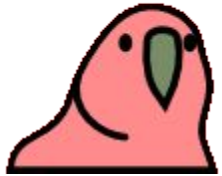
Análisis semántico no modular



CRYSTAL

Crystal negocia modularidad por expresividad

CRYSTAL

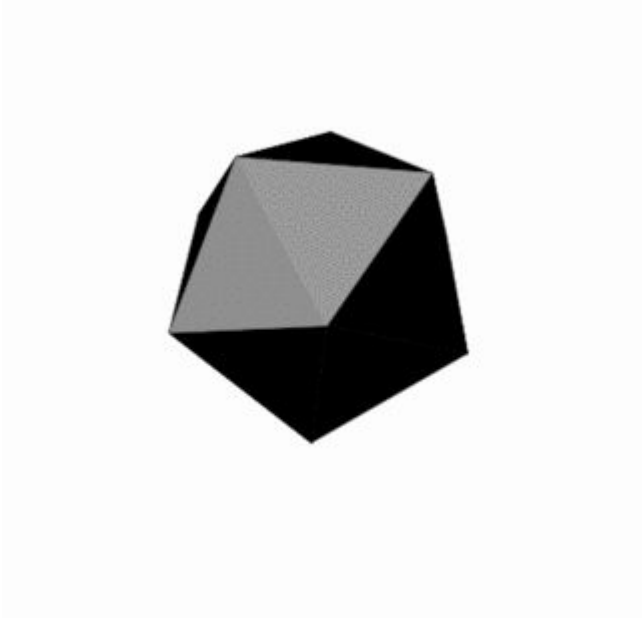


¿¿¿Crystal negocia modularidad???

CRYSTAL

Crystal se siente dinámico, pero es seguro

CRYSTAL



Qué estudiar

CRYSTAL

¿Por qué estudiar Crystal?

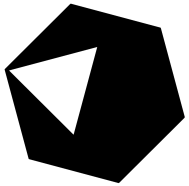
- El typechecker baila tango 💃
  - El flujo de tipos baila desde las llamadas de función hasta los módulos
  - ¡Nacido en Buenos Aires! (Pero de alcance global)
- Lenguaje industrial con problemas concretos a resolver
- Con una comunidad vibrante
  - +20k estrellas en github ⭐⭐⭐

CRYSTAL

Montón de problemas interesantes

microCrystal

Semantic process caching

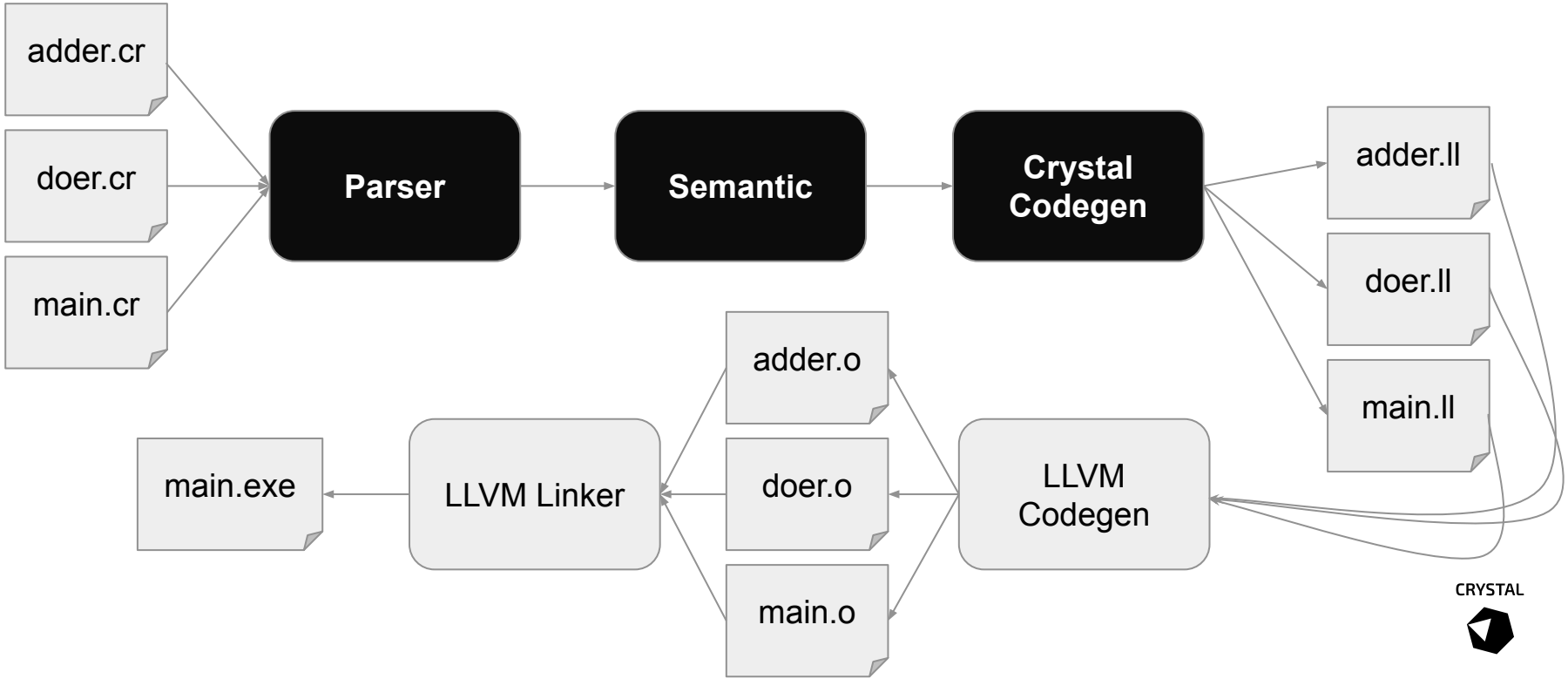


Características funcionales

Análisis de escape

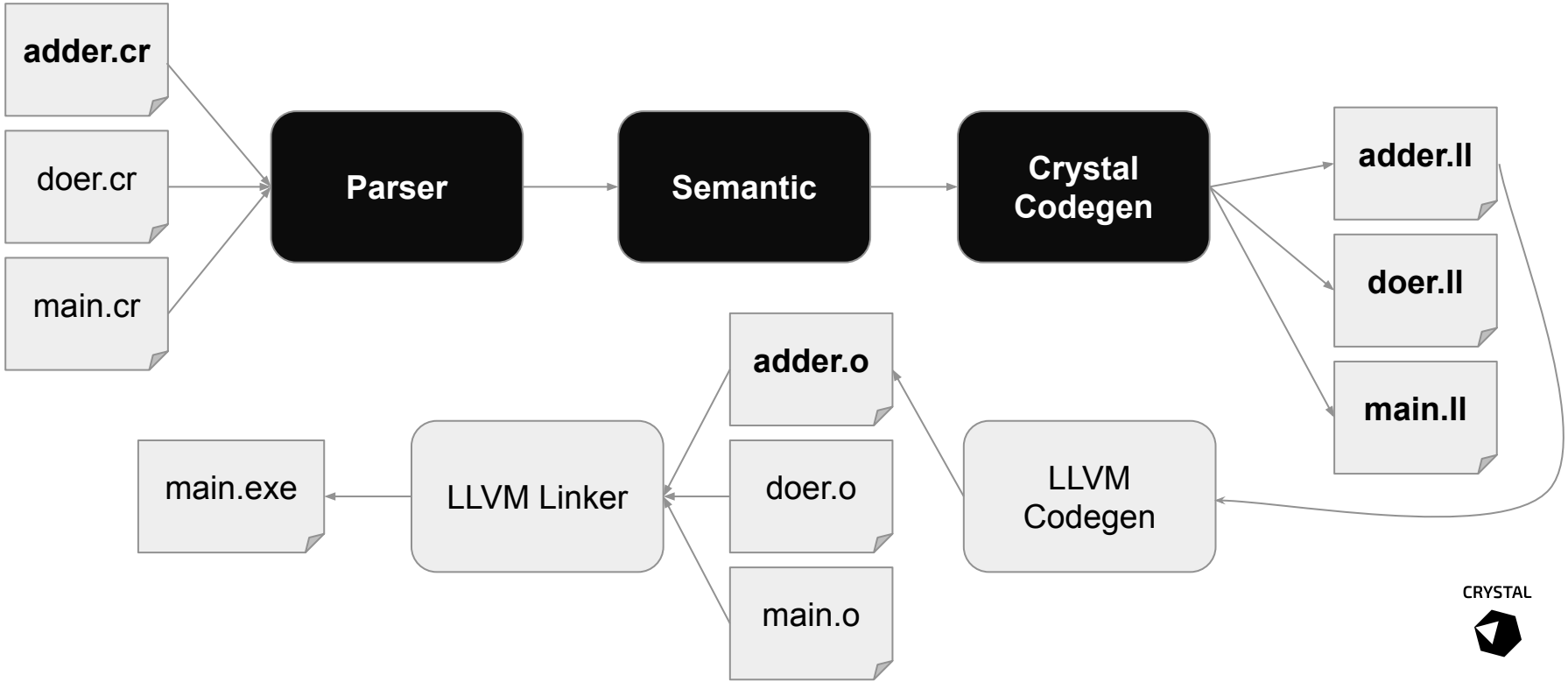
CRYSTAL

El compilador en un slide



CRYSTAL

Cambia un archivo – Se analiza todo



CRYSTAL

www.crystal-lang.org

CRYSTAL