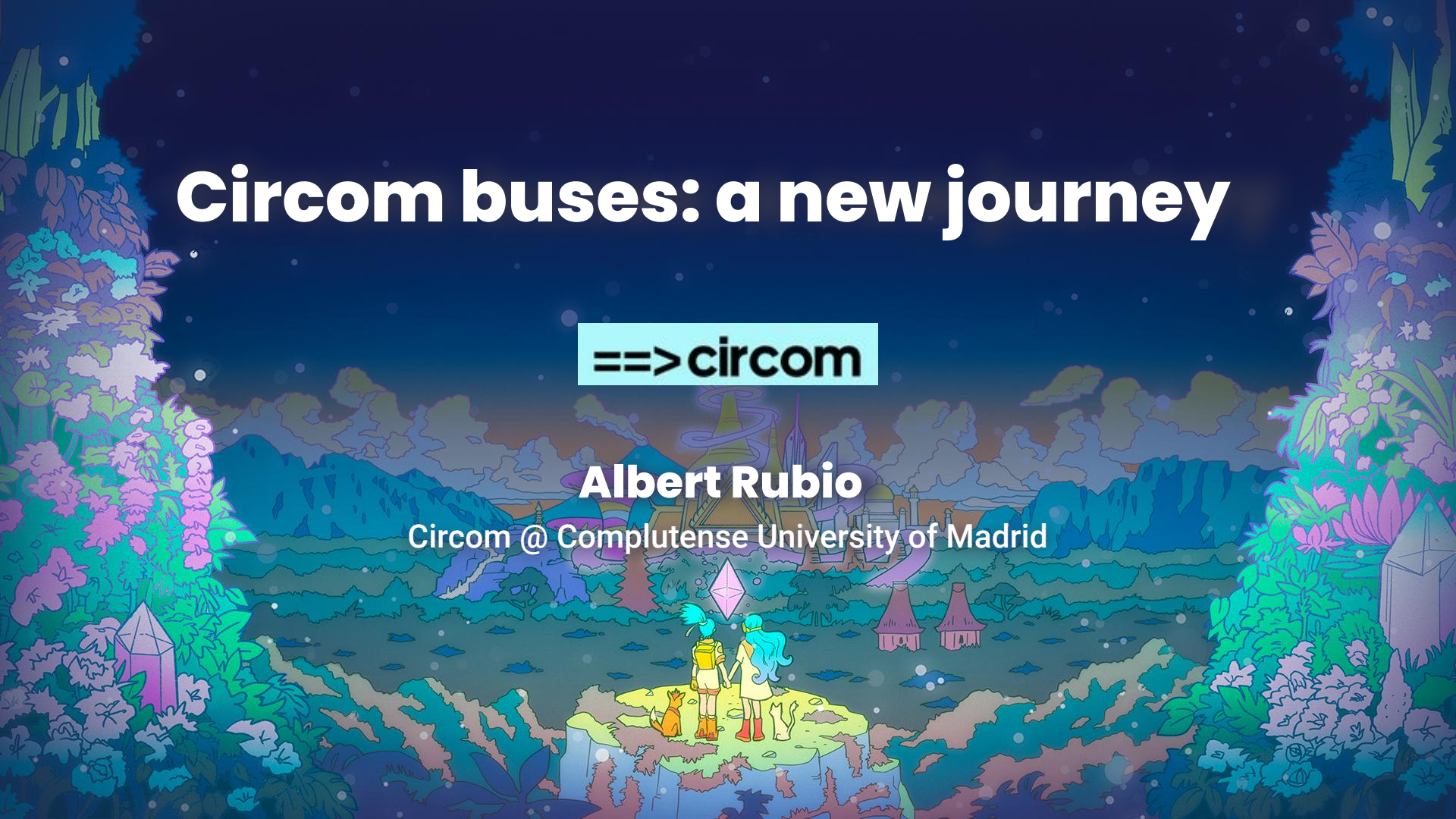
Circom buses: a new journey

==>circom

Albert Rubio

Circom @ Complutense University of Madrid



==> Circom programming language and compiler

In the circom DSL:

- The programmer explicitly provides the constraints defining the circuit.
- The programmer also provides an efficient way to compute the witness.

Using library subcircuits is encouraged (circomlib)

The circom compiler produces:

- The circuit as a constraint system (in R1CS)
- The program code to execute the circuit (compute the witness)

circom has a too simple type system

==> circom buses example 1(basic)

Assume we want to build a circuit to compare dates

```
template smaller_date( ) {
   signal input xday;
   signal input xmonth;
   signal input xyear;
   signal input yday;
   signal input ymonth;
   signal input yyear;
   signal output out;

   signal aux <== LessThan(5)(xyear, yyear);
   …
}
```

==> circom buses example 1(basic)

Assume we want to build a circuit to compare dates

```
template smaller_date( ) {
   signal input x[3];
   signal input y[3];
   signal output out;

   signal aux <== LessThan(5)(x[2],y[2]);
   …
}
```

==> circom buses example 1

Assume we want to build a circuit to compare dates

```
bus Date(){
   signal day;
   signal month;
   signal year;
}

template smaller_date( ) {
   input Date() x;
   input Date() y;
   output signal out;

   signal aux1 <== LessThan(5)(x.year,y.year);
   …
}
```

==> Buses in circom 2.2

- A bus is a collection of signals (like a struct in other languages).

  We call them buses because we are programming circuits!

  A bus can be defined using signals, arrays or other buses

  Recursive definitions are not allowed yet (but soon!)

- A bus defines a new type.

  Type checking prevents us from mixing different types of buses.

  circomlib2 (beta) is defined using buses

==> circom buses example 2

```
bus Date(){
   signal day;
   signal month;
   signal year;
}
```

```
bus Profile(n){
   signal id;
   Date() transaction_dates[n];
}
```

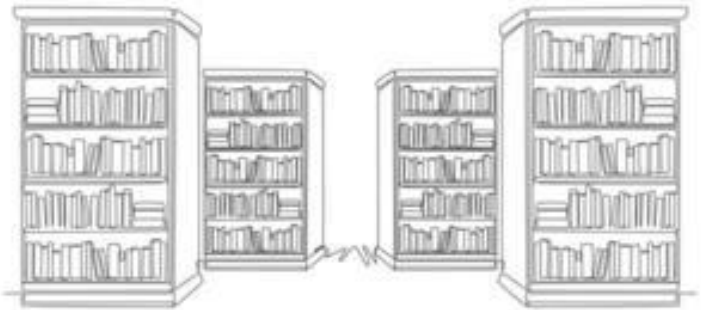
```
template check_profile(n) {
   input Profile(n) member;
   Date() first <== first_date(n)(member.transaction_dates);
   …
      next === member.transaction[i].day + 1;
   …
}
template first_date(n) {
   input Date() transaction_dates[n];
   output Date() out;
   …
}
```

==> Buses and tags

- Tags provide a simple form of subtyping
  - create new types with buses and
  - define a subtype relation using tags
- Tags can occur inside and outside a bus.
  - Inside restricts the fields of the bus.

    Applies to any instance of the bus
  - Outside restricts the particular instance of the bus.
- Adding tags to buses allows us to express properties relating all signals in the bus

The new circomlib2



==> circomlib example without buses

Twisted Edwards Form

Baby Jub Elliptic curve

Conversions

Montgomery Form

```
template BabyAdd() {
   signal input x1;
   signal input y1;
   signal input x2;
   signal input y2;
   signal output xout;
   signal output yout;
   …
}
```

```
template BabyCheck() {
   signal input x;
   signal input y;
   var a = 168700;
   var d = 168696;
   signal x2 <== x * x;
   signal y2 <== y * y;
   a*x2 + y2 === 1 + d*x2 * y2;
}
```

```
template Edwards2Montgomery() {
   signal input in[2];
   signal output out[2];

   out[0] <-- (1 + in[1]) / (1 - in[1]);
   out[1] <-- out[0] / in[0];
   out[0] * (1 - in[1]) === (1 + in[1]);
   out[1] * in[0] === out[0];
}
```

```
template Montgomery2Edwards() {
   signal input in[2];
   signal output out[2];

   out[0] <-- in[0] / in[1];
   out[1] <-- (in[0] - 1) / (in[0] + 1);
   out[0] * in[1] === in[0];
   out[1] * (in[0] + 1) === in[0] - 1;
}
```

```
template MontgomeryAdd() {
   signal input in1[2];
   signal input in2[2];
   signal output out[2];
   var a = 168700;
   var d = 168696;
   var A = (2 * (a + d)) / (a - d);
   var B = 4 / (a - d);
   signal lamda <-- (in2[1] - in1[1]) / (in2[0] - in1[0]);
   lamda * (in2[0] - in1[0]) === (in2[1] - in1[1]);
   out[0] <== B*lamda*lamda - A - in1[0] -in2[0];
   out[1] <== lamda * (in1[0] - out[0]) - in1[1];
}
```

```
template MontgomeryDouble() {
   signal input in[2];
   signal output out[2];
   …
}
```

LOW READABILITY

ERROR PRONE

==> circomlib example with buses

Twisted Edwards Form

```
bus EdwardsPoint() {
  signal x;
  signal y;
}

template BabyAdd() {
  input EdwardsPoint() p1;
  input EdwardsPoint() p2;
  output EdwardsPoint() out;
  …
}

template BabyCheck() {
  input EdwardsPoint() p;
  var a = 168700;
  var d = 168696;
  signal x2 <== p.x * p.x;
  signal y2 <== p.y * p.y;
  a*x2 + y2 === 1 + d*x2 * y2;
}
```

Baby Jub Elliptic curve

Conversions

```
template Edwards2Montgomery() {
  input EdwardsPoint() in;
  output MontgomeryPoint() out;

  out.x <-- (1 + in.y) / (1 - in.y);
  out.y <-- out.x / in.x;
  out[0] * (1-in.y) === (1 + in.y);
  out.y * in.x === out.x;
}

template Montgomery2Edwards() {
  input MontgomeryPoint() in;
  output EdwardsPoint() out;

  out.x <-- in.x / in.y;
  out.y <-- (in.x - 1) / (in.x + 1);
  out.x * in.y === in.x;
  out.y * (in.x + 1) === in.x - 1;
}
```

Montgomery Form

```
bus MontgomeryPoint() {
  signal x;
  signal y;
}

template MontgomeryAdd() {
  input MontgomeryPoint() in1, in2;
  output MontgomeryPoint() out;
  var a = 168700;
  var d = 168696;
  var A = (2 * (a + d)) / (a - d);
  var B = 4 / (a - d);
  signal lamda <-- (in2.y - in1.y) / (in2.x - in1.x);
  lamda * (in2.x - in1.x) === (in2.y - in1.y);
  out.x <== B*lamda*lamda - A - in1.x -in2.x;
  out.y <== lamda * (in1.x - out.x) - in1.y;
}
template MontgomeryDouble() {
  input MontgomeryPoint() in;
  output MontgomeryPoint() out;
  …
}
```

READABLE

TYPESAFE

==> circomlib example with buses and tags

Twisted Edwards Form

```
bus Point() {
  signal x;
  signal y;
}

template BabyAdd() {
  input {edwards} Point() p1;
  input {edwards} Point() p2;
  output {edwards} Point() out;
  …
}

template BabyCheck() {
  input {edwards} Point() p;
  var a = 168700;
  var d = 168696;
  signal x2 <== p.x * p.x;
  signal y2 <== p.y * p.y;
  a*x2 + y2 === 1 + d*x2 * y2;
}
```

Baby Jub Elliptic curve

Conversions

```
template Edwards2Montgomery() {
  input {edwards} Point() in;
  output {montgomery} Point() out;
  out.x <-- (1 + in.y) / (1 - in.y);
  out.y <-- out.x / in.x;
  out[0] * (1-in.y) === (1 + in.y);
  out.y * in.x === out.x;
}

template Montgomery2Edwards() {
  input {montgomery} Point() in;
  output {edwards} Point() out;
  out.x <-- in.x / in.y;
  out.y <-- (in.x - 1) / (in.x + 1);
  out.x * in.y === in.x;
  out.y * (in.x + 1) === in.x - 1;
}
```

Montgomery Form

```
template MontgomeryAdd() {
  input {montgomery} Point() in1, in2;
  output {montgomery} Point() out;
  var a = 168700;
  var d = 168696;
  var A = (2 * (a + d)) / (a - d);
  var B = 4 / (a - d);
  signal lamda <-- (in2.y - in1.y) / (in2.x - in1.x);
  lamda * (in2.x - in1.x) === (in2.y - in1.y);
  out.x <== B*lamda*lamda - A - in1.x -in2.x;
  out.y <== lamda * (in1.x - out.x) - in1.y;
}

template MontgomeryDouble() {
  input {montgomery} Point() in;
  output {montgomery} Point() out;
  …
}
```

READABLE

TYPE SAFE

REUSABLE

==> Revised circom library: circomlib2 (beta)

- Most circuits rewritten:
  - Adding buses
  - Adding tags
  - Compatible with the use of different prime fields.
- No need to audit again: we have formally verified that the new version produces the same constraints.
- Ensure safety when types and tags are used correctly
  - Use library templates to create buses and add tags!

==> Conclusions

- circom is a prominent ZK DSL used in production projects
  - provides an ecosystem of tools to program and test circuits and generate ZK proofs

and now …

- circom 2.2.* introduces buses to structure circuits
- Provides a far more convenient type system: Type safety
- Increase readability and security: helps audit zk projects
- The new circomlib2 takes advantage of all these features

==> Coming soon ....

- Finish the new circomlib 2
- Add recursive types/buses (recursive definitions)
- circom virtual machine (CVM)
  - IR for the witness generation code
  - CVM interpreters written different languages:
    - in Rust as a crate
    - In C++ as a static library
  - From CVM to new back-ends (independent)

UNIVERSIDAD COMPLUTENSE MADRID

==>circom

polygon

Thank you!

Clara Rodríguez, Miguel Isabel, Lucas Cuesta ...