

# FHE Transpiler

Miguel Guevara  
Product Manager, Google

- From Theory to Practice  
A product-centric view





# AGENDA

1. FHE
2. Pareto Optimality
3. What can we do?
4. What's out of reach?
5. Our future direction

Hi!

I'm Miguel



**Fully Homomorphic Encryption (FHE)**  
makes computations on encrypted  
data possible.



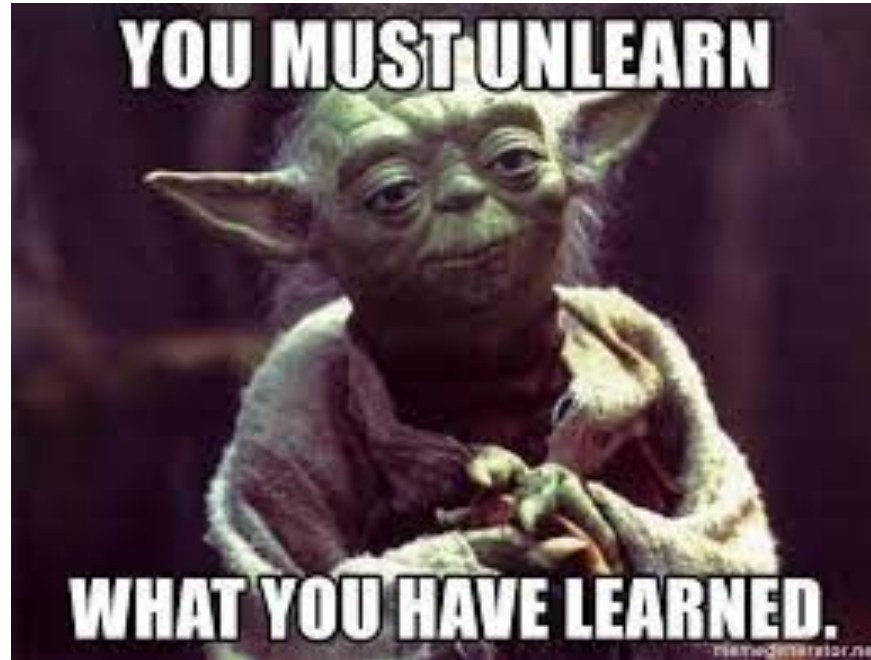
Without FHE



With FHE

A recap of our team's work

# Adding FHE to an existing application

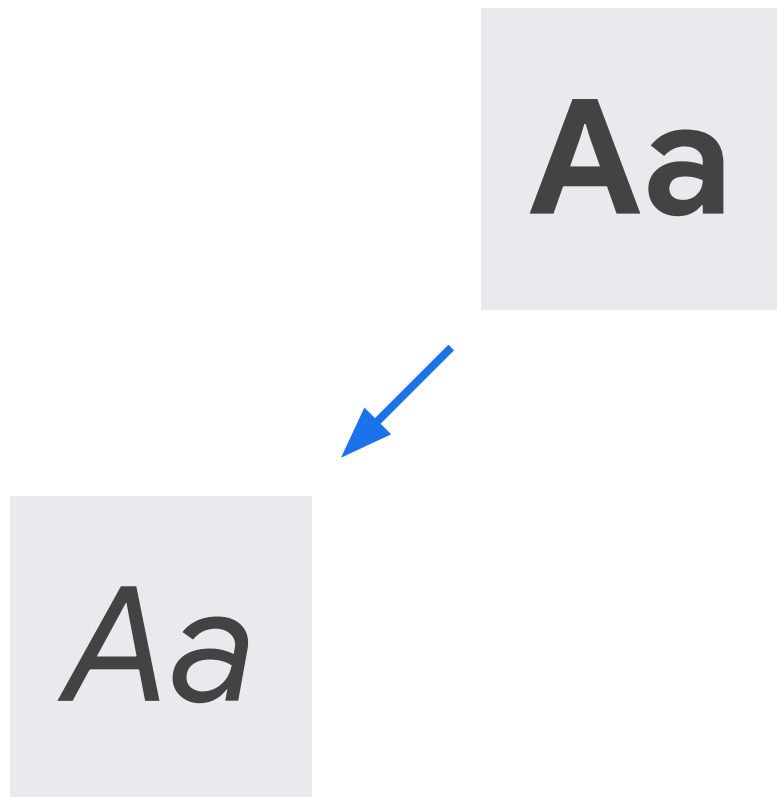






# FHE transpiler

Allows us to easily convert existing code that works on plaintext to work on ciphertext.





# Challenges



Ease of Use



Speed



# The regulatory changes play a key role

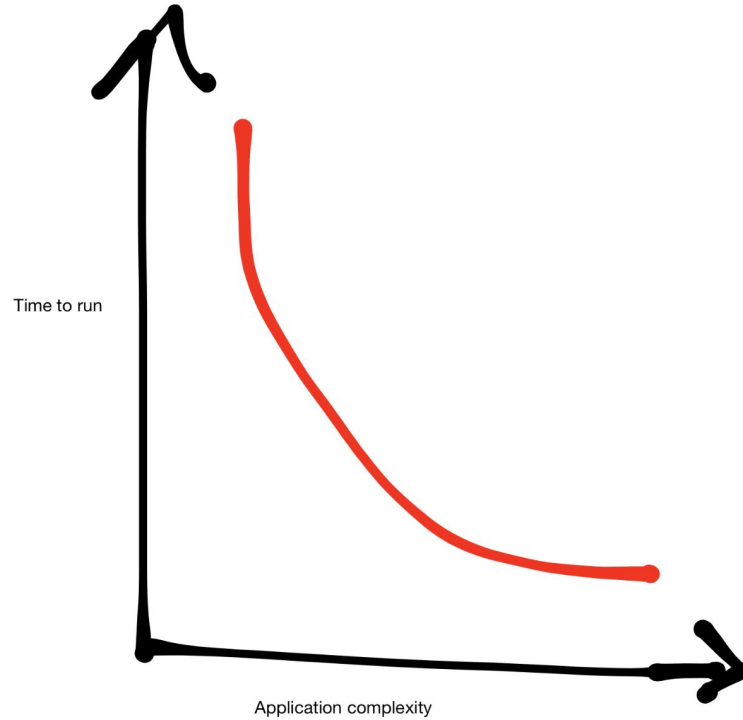
- GDPR
- Global privacy regulation
- Increased privacy awareness



# Finding the right fit



# Pareto Optimality



# We're far from optimality



# Removing the human in the loop



# Removing the human in the loop







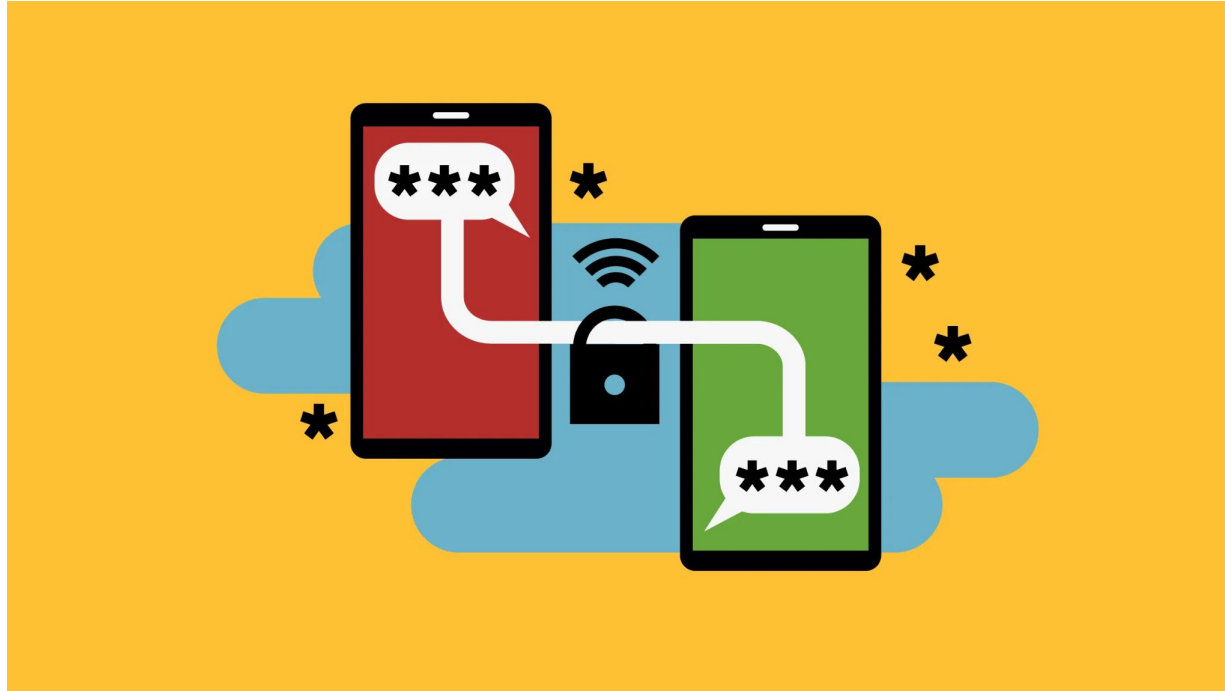
Lesson: the user has a **40 minute** tolerance to computational delays



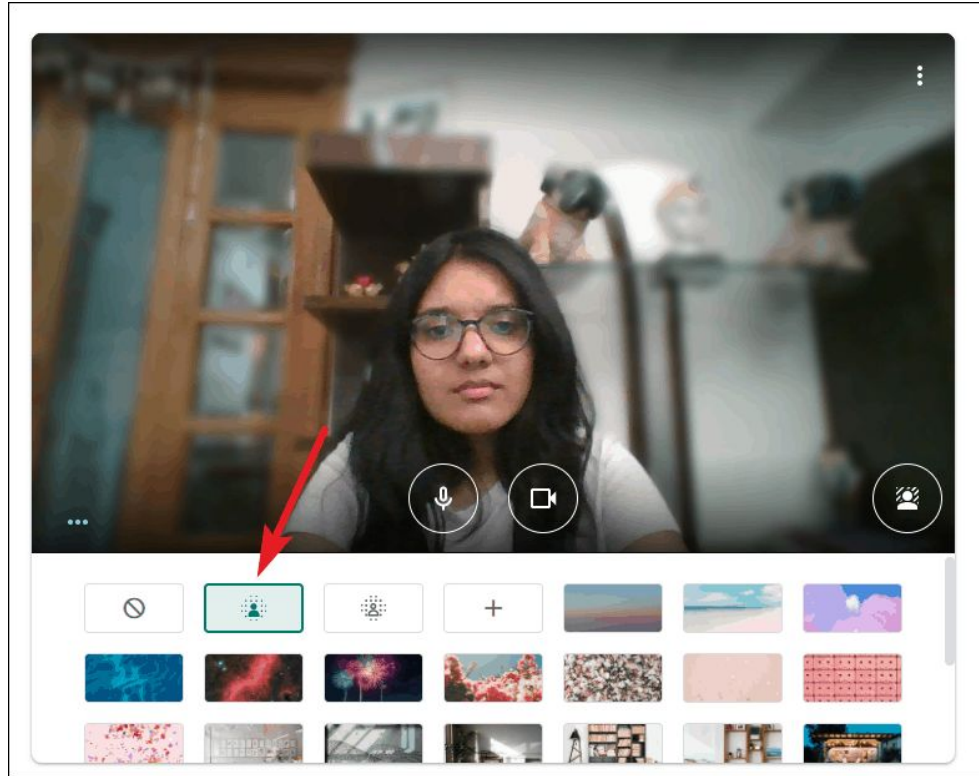
# What can we do in 40 minutes?

- Embedding comparison
- Transcripts
- Image processing

# Another example: loss of functionality due to E2E Encryption



# Blurring



# Spell Check

The image shows a Google Docs interface with a document titled "Sample Document". The document content is the Preamble to the US Constitution. A spell check dialog box is open over the word "perfect", which is highlighted in blue in the document. The dialog box shows the word "perfect" as the suggested correction. The document text is: "Sample Document: Preamble to the Constitution| We the People of the United States, in Order to form a more perfect Justice, insure domestic Tranquility, provide for the common defence, promote the general Welfare, and secure the Blessings of Liberty to ourselves and our posterity, do ordain and establish this Constitution for the United States of America."

Sample Document

File Edit View Insert Format Tools Add-ons Help [All changes saved in Drive](#)

100% Normal text Arial 11 B I U A

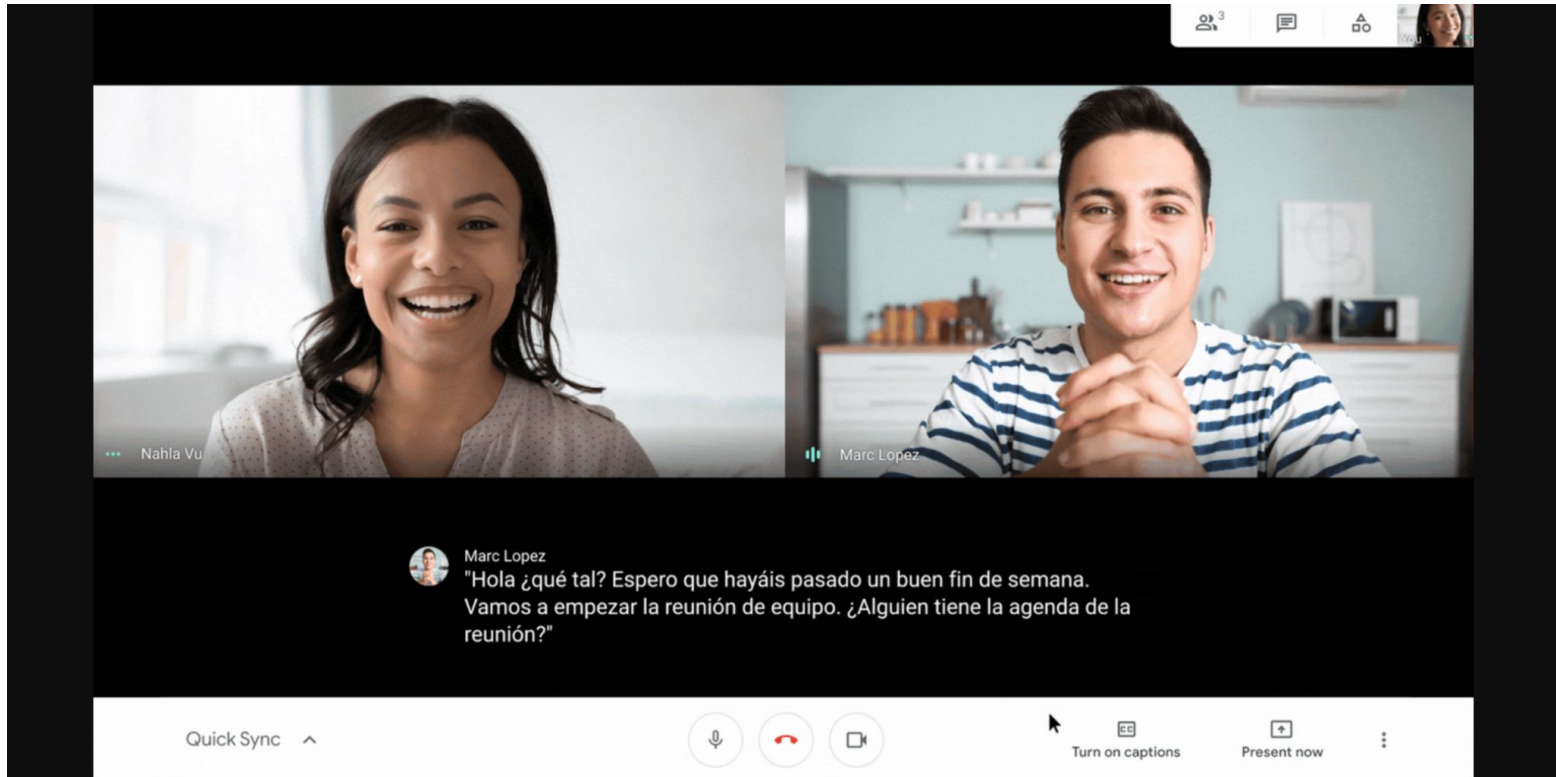
Spelling and grammar

Change perfect to:

perfect

Ignore Accept

# Out of Bounds: Real-time captions



The screenshot shows a Zoom meeting interface. At the top right, there are icons for participants (3), chat, and a small video thumbnail of Nahla Vu. The main area contains two video feeds: Nahla Vu on the left and Marc Lopez on the right. Below the video feeds, a chat bubble from Marc Lopez contains the text: "Hola ¿qué tal? Espero que hayáis pasado un buen fin de semana. Vamos a empezar la reunión de equipo. ¿Alguien tiene la agenda de la reunión?". At the bottom, the Zoom control bar includes "Quick Sync", microphone, video, and screen sharing icons, along with "Turn on captions" and "Present now" buttons.

Nahla Vu

Marc Lopez

Marc Lopez  
"Hola ¿qué tal? Espero que hayáis pasado un buen fin de semana. Vamos a empezar la reunión de equipo. ¿Alguien tiene la agenda de la reunión?"

Quick Sync ^

Turn on captions

Present now



# What's next for our team?

01

Reduce the computational overhead

02

Work to find applications that have a natural fit for FHE

03

Implement solutions for VERY simple use-cases

04

Launch an FHE application

05

Find ways to use FHE in combination with other PPTs



# What do I think we need to work as a community?

01

Reduce the computational overhead

02

Reduce the computational overhead

03

Reduce the computational overhead for an entire application





**Questions?**





**Fin**





# Get involved



[google/fully-homomorphic-encryption](https://github.com/google/fully-homomorphic-encryption)



[fhe-team@google.com](mailto:fhe-team@google.com)





# Challenges



Ease of Use



Speed



# FHE Programming

Need to abstract away cryptographic details from engineers



Cryptography



Engineering

Reference: SoK: Fully Homomorphic Encryption Compilers, Viand et al



# FHE Programming :

# Cryptographic challenges

01

Parameter selection

02

Noise tracking

03

Bootstrapping

04

Circuit depth analysis

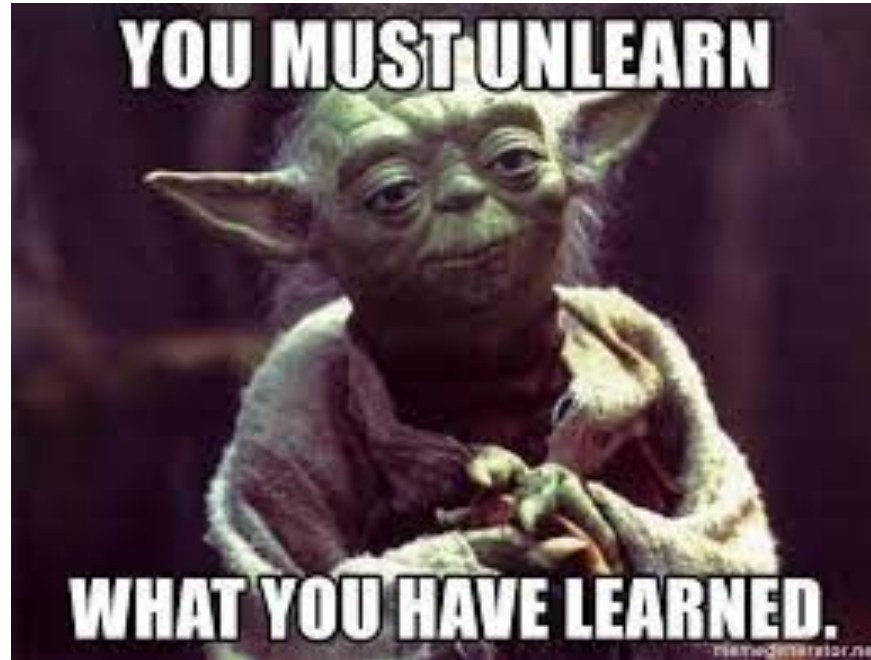
Reference: SoK: Fully Homomorphic Encryption Compilers, Viand et al



# FHE Programming : Engineering challenges

- 01 If / else needs to be written as controlled multiplexer
- 02 No variable length loops and jumps based on data
- 03 No branch and bound optimizations
- 04 Using the right scheme for the right type of computation  
  
Example: approximate computations don't work for string processing

# Adding FHE to an existing application

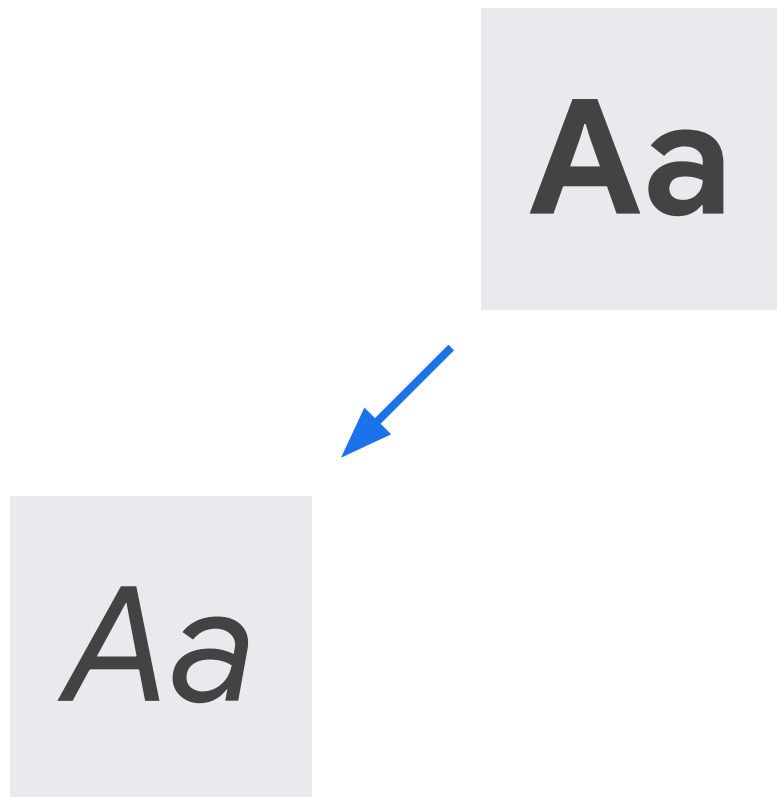






# FHE transpiler

Allows us to easily convert existing code that works on plaintext to work on ciphertext.





# FHE Transpiler Building Blocks





# FHE Schemes



## Levelled schemes

BGV, BFV, CKKS



## Composable schemes

FHEW, TFHE



# General purpose FHE compilers

## Cingulata

Armadillo: a compilation chain for privacy preserving applications

- *Carpov et al*

## E3

A Framework for Compiling C++ Programs with Encrypted Operands

- *Chielle et al*



# General purpose FHE compilers

## Cingulata

Armadillo: a compilation chain for privacy preserving applications

- *Carpov et al*

*Uses boolean circuit*

## E3

A Framework for Compiling C++ Programs with Encrypted Operands

- *Chielle et al*



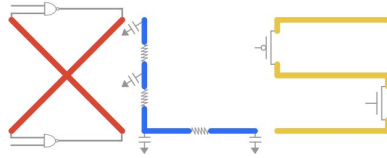
**FHE**

**Programming**

**Hardware**

**Design**

# High-Level Synthesis (HLS) Tools



XLS: Accelerated HW Synthesis



# Composable FHE

## FHEW

FHEW: Bootstrapping Homomorphic Encryption in Less Than a Second

- *Ducas, Micciancio*

## CGGI/TFHE

Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds

- *Chilloti et al*





# TFHE API

```
void bootsNOT(LweSample* result, const LweSample* ca,  
              const TFheGateBootstrappingCloudKeySet* bk);  
void bootsNAND(LweSample* result, const LweSample* ca, const LweSample* cb,  
              const TFheGateBootstrappingCloudKeySet* bk);  
void bootsOR(LweSample* result, const LweSample* ca, const LweSample* cb,  
            const TFheGateBootstrappingCloudKeySet* bk);  
void bootsAND(LweSample* result, const LweSample* ca, const LweSample* cb,  
            const TFheGateBootstrappingCloudKeySet* bk);  
void bootsXOR(LweSample* result, const LweSample* ca, const LweSample* cb,  
            const TFheGateBootstrappingCloudKeySet* bk);  
void bootsNOR(LweSample* result, const LweSample* ca, const LweSample* cb,  
            const TFheGateBootstrappingCloudKeySet* bk);  
void bootsMUX(LweSample* result, const LweSample* a, const LweSample* b,  
            const LweSample* c, const TFheGateBootstrappingCloudKeySet* bk);
```

# FHE transpiler

Convert any C++ code to work on ciphertext.

```
int sum(int a, int b) {  
    return (a + b);  
}
```



```
#include <tfhe.h>  
  
void sum(LweSample* result, const LweSample* a, const LweSample* b,  
const int nb_bits, const TFheKeySet* bk) {  
    LweSample* carry = new_ciphertext(bk->params);  
    LweSample* temp = new_ciphertext(bk->params);  
    // Initialize the carry to 0  
    bootsCONSTANT(&carry, 0, bk);  
    // Compute bitwise addition  
    for (int i = 0; i < nb_bits; i++) {  
        // Compute Sum  
        bootsXOR(&temp, &a[i], &b[i], bk);  
        bootsXOR(&result[i], &temp, &carry, bk);  
        // Compute carry  
        bootsAND(&carry, &carry, &temp, bk);  
        bootsAND(&temp, &a[i], &b[i], bk);  
        bootsOR(&carry, &temp, &carry, bk);  
    }  
    delete_ciphertext(carry);  
    delete_ciphertext(temp);  
}
```



# FHE Transpiler Architecture





# FHE Applications

## Client

- Encryption
- Decryption
- Key generation

## Server

- Homomorphic evaluation of computational circuit

# Transpiler Design



Frontend: Parser

Middle-end: FHE Optimizer

Backend: Execution Engine

# Transpiler Design

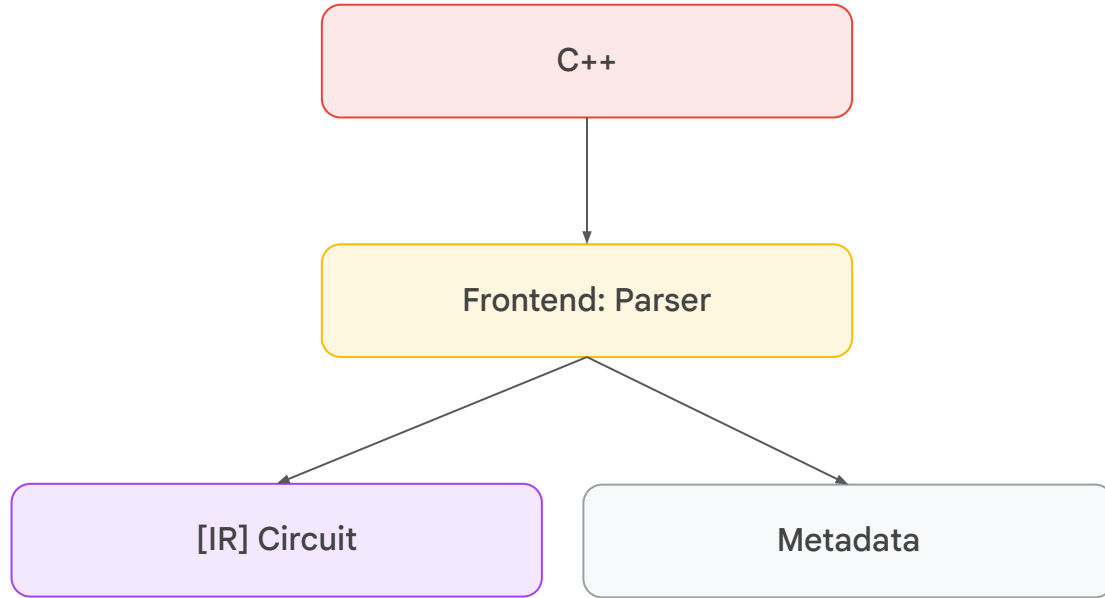


Frontend: Parser

Middle-end: FHE Optimizer

Backend: Execution Engine

# Transpiler Frontend: Parser



## Basic Example: String Capitalization

The following code capitalizes words in a string and is written using standard C++ syntax.

```
#include "string_cap.h"
State::State()
  : last_was_space_(true) {

unsigned char State::process(unsigned char c) {
    unsigned char ret = c;

    if(last_was_space_ && (c >= 'a') && (c <= 'z'))
        ret -= ('a' - 'A');

    last_was_space_ = (c == ' ');
    return ret;
}

// Note: Way to mark State::process() as main function
char my_package(State &st, char c) {
    return st.process(c);
}
```





# XLS IR

## Intermediate representation

```
package my_package
fn _ZN5State7processEh(this: (bits[1]), c: bits[8]) -> (bits[8], (bits[1])) {
  literal.20: bits[8] = literal(value=97, pos=1,10,2)
  literal.31: bits[8] = literal(value=97, pos=1,11,3)
  ...
  ...
  ...
  ret tuple.44: (bits[8], (bits[1])) = tuple(sel.36, tuple.43, pos=1,7,1)
}

fn my_package(st: (bits[1]), c: bits[8]) -> (bits[8], (bits[1])) {
  invoke.45: (bits[8], (bits[1])) = invoke(st, c, to_apply=_ZN5State7processEh,
pos=1,19,2)
  tuple_index.46: bits[8] = tuple_index(invoke.45, index=0, pos=1,19,2)
  tuple_index.47: (bits[1]) = tuple_index(invoke.45, index=1, pos=1,19,2)
  ret tuple.48: (bits[8], (bits[1])) = tuple(tuple_index.46, tuple_index.47,
pos=1,18,1)
}
```

# Transpiler Design

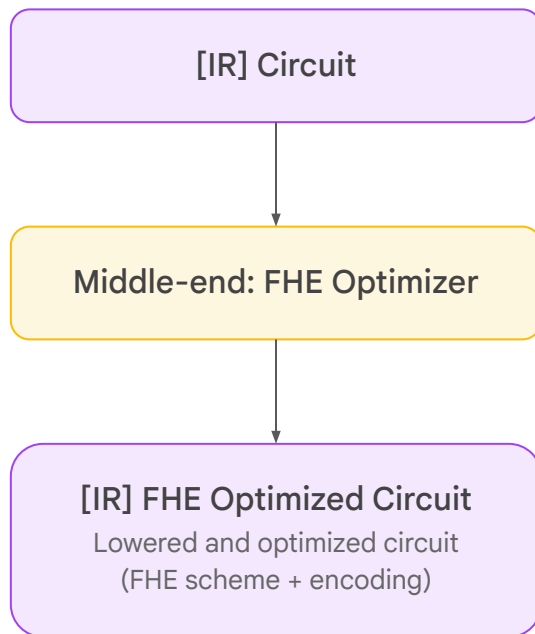


Frontend: Parser

Middle-end: FHE Optimizer

Backend: Execution Engine

# Transpiler Middle-end: FHE Optimizer





## Booleanized XLS IR

IR after running through the booleanifier.

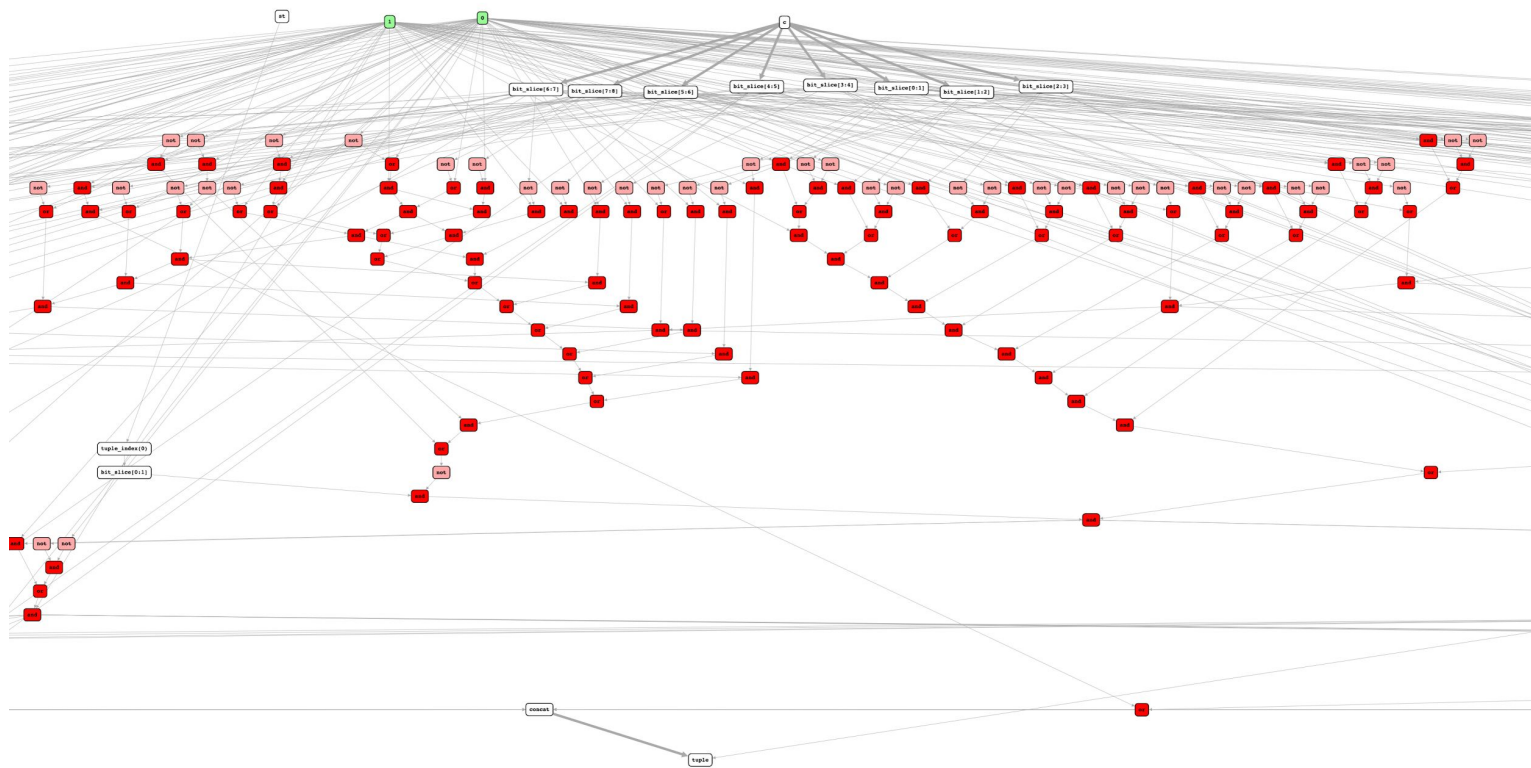
Now it only contains AND, OR, and NOT gates.

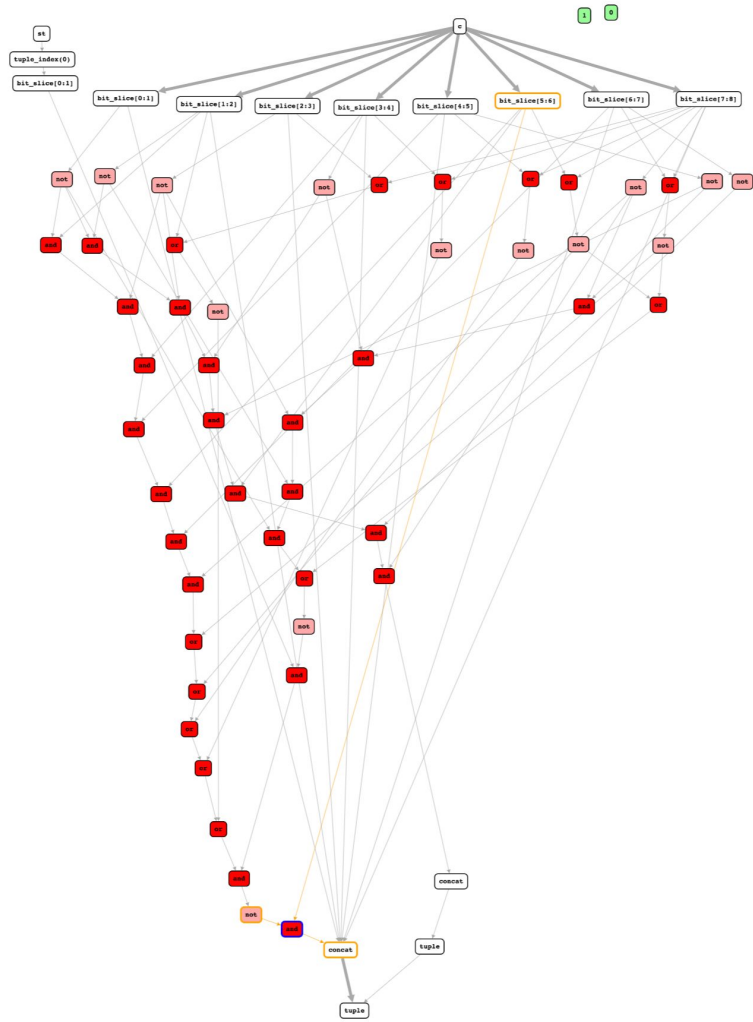
Boolean-ized IR:

```
package p

fn my_package_boolean(st: (bits[1]), c: bits[8]) -> (bits[8], (bits[1])) {
  literal.115: bits[8] = literal(value=1)
  shr1.117: bits[8] = shr1(c, literal.115)
  shr1.119: bits[8] = shr1(shr1.117, literal.115)
  shr1.121: bits[8] = shr1(shr1.119, literal.115)
  shr1.123: bits[8] = shr1(shr1.121, literal.115)
  shr1.125: bits[8] = shr1(shr1.123, literal.115)
  shr1.127: bits[8] = shr1(shr1.125, literal.115)
  ...
  ...
  ...

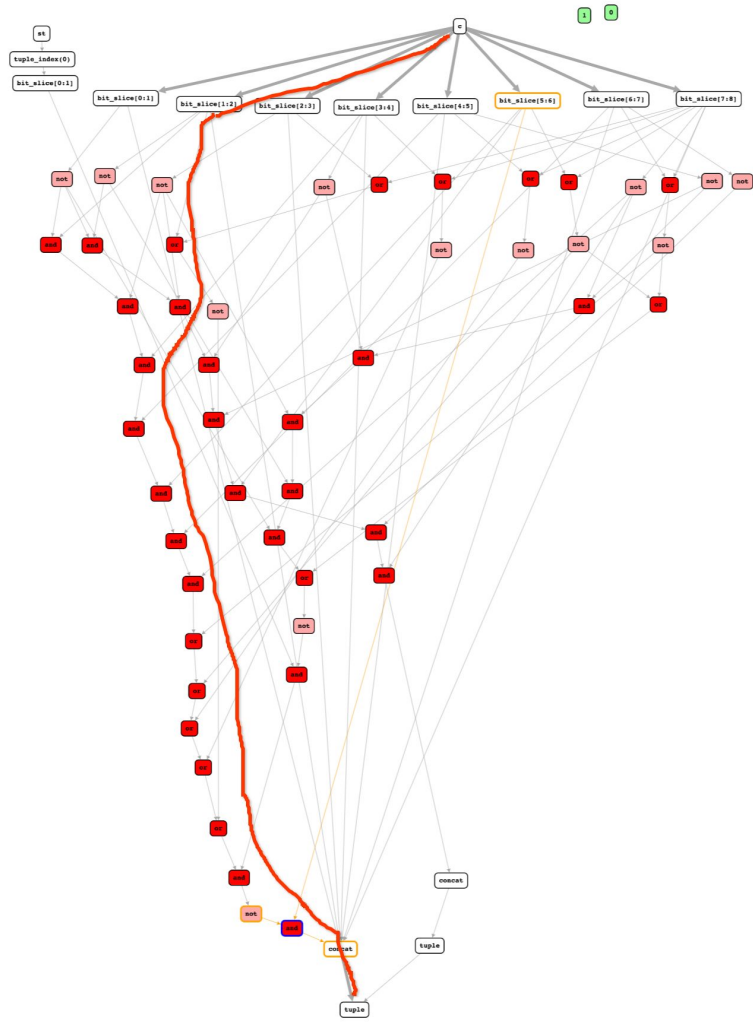
  shr1.131: bits[8] = shr1(shr1.129, literal.115)
  and.196: bits[1] = and(and.189, or.195)
  and.263: bits[1] = and(and.256, or.262)
  or.346: bits[1] = or(or.345, and.342)
  ret tuple.468: (bits[8], (bits[1])) = tuple(concat.465, tuple.467)
}
```





1 0





1 0



JSON Cytoscape graph

load Choose file

Browse

benchmarks...

Opt lev

Load

ckage my\_package

my\_package\_boolean(st: (bits[1]), c: bits[8]) -> (bits[8], (bits[1]))

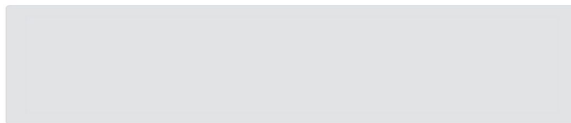
```
bit_slice.1075: bits[1] = bit_slice(c, start=0, width=1, id=1075)
not.1088: bits[1] = not(bit_slice.1075, id=1088)
bit_slice.1076: bits[1] = bit_slice(c, start=1, width=1, id=1076)
bit_slice.1077: bits[1] = bit_slice(c, start=2, width=1, id=1077)
and.1102: bits[1] = and(not.1088, bit_slice.1076, id=1102)
not.1086: bits[1] = not(bit_slice.1077, id=1086)
and.1103: bits[1] = and(and.1102, not.1086, id=1103)
bit_slice.1078: bits[1] = bit_slice(c, start=3, width=1, id=1078)
bit_slice.1082: bits[1] = bit_slice(c, start=7, width=1, id=1082)
bit_slice.1079: bits[1] = bit_slice(c, start=4, width=1, id=1079)
and.1104: bits[1] = and(and.1103, bit_slice.1078, id=1104)
not.1083: bits[1] = not(bit_slice.1082, id=1083)
not.1084: bits[1] = not(bit_slice.1079, id=1084)
and.1105: bits[1] = and(and.1104, bit_slice.1079, id=1105)
bit_slice.1080: bits[1] = bit_slice(c, start=5, width=1, id=1080)
and.1093: bits[1] = and(not.1083, not.1084, id=1093)
not.1085: bits[1] = not(bit_slice.1078, id=1085)
and.1106: bits[1] = and(and.1105, bit_slice.1080, id=1106)
bit_slice.1081: bits[1] = bit_slice(c, start=6, width=1, id=1081)
and.1094: bits[1] = and(and.1093, not.1085, id=1094)
and.1107: bits[1] = and(and.1106, bit_slice.1081, id=1107)
or.1089: bits[1] = or(bit_slice.1082, bit_slice.1081, id=1089)
or.1091: bits[1] = or(bit_slice.1082, bit_slice.1080, id=1091)
and.1095: bits[1] = and(and.1094, not.1086, id=1095)
not.1087: bits[1] = not(bit_slice.1076, id=1087)
and.1108: bits[1] = and(and.1107, not.1083, id=1108)
not.1090: bits[1] = not(or.1089, id=1090)
not.1092: bits[1] = not(or.1091, id=1092)
and.1096: bits[1] = and(and.1095, not.1087, id=1096)
or.1117: bits[1] = or(and.1108, not.1090, id=1117)
or.1109: bits[1] = or(bit_slice.1082, bit_slice.1079, id=1109)
and.1127: bits[1] = and(not.1088, not.1087, id=1127)
or.1100: bits[1] = or(not.1090, not.1092, id=1100)
and.1097: bits[1] = and(and.1096, not.1088, id=1097)
or.1118: bits[1] = or(or.1117, not.1092, id=1118)
not.1110: bits[1] = not(or.1109, id=1110)
or.1111: bits[1] = or(bit_slice.1082, bit_slice.1078, id=1111)
or.1113: bits[1] = or(bit_slice.1082, bit_slice.1077, id=1113)
and.1128: bits[1] = and(and.1127, not.1086, id=1128)
```

K

ew Graph

Critical Path

Show only selected nodes





# Transpiler Design

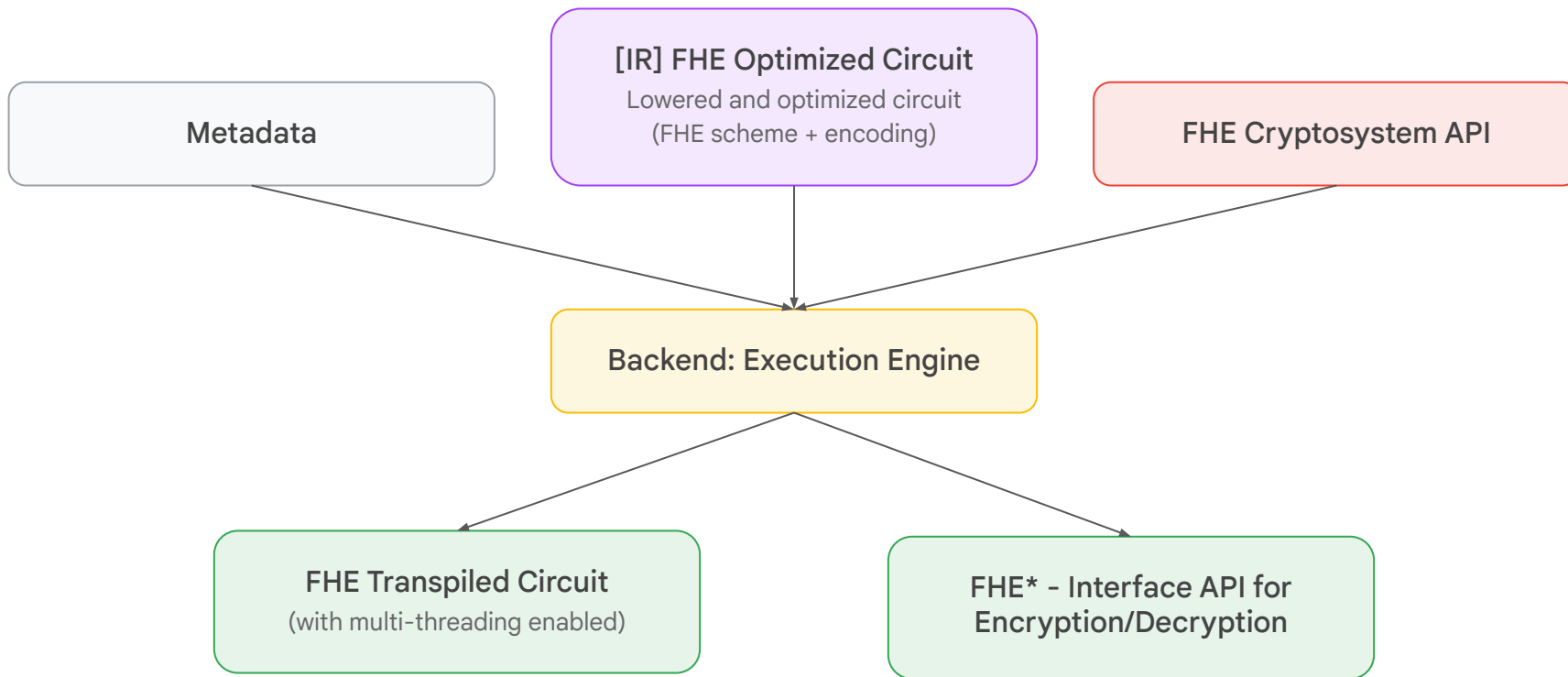


Frontend: Parser

Middle-end: FHE Optimizer

Backend: Execution Engine

# Transpiler Backend: Execution Engine



# FHE C++

C++ that now works on  
ciphertext.

```
#include <tfhe/tfhe.h>
#include <tfhe/tfhe_io.h>

#include <unordered_map>

void my_package_boolean(LweSample* result, LweSample* st, LweSample* c,
                        const TFheGateBootstrappingCloudKeySet* bk) {
    std::unordered_map<int, LweSample*> temp_nodes;

    temp_nodes[115] = new_gate_bootstrapping_ciphertext(bk->params);
    bootsCONSTANT(temp_nodes[115], 1, bk);

    ...
    ...
    ...

    bootsCOPY(&result[2], temp_nodes[459], bk);
    bootsCOPY(&result[3], temp_nodes[460], bk);
    bootsCOPY(&result[4], temp_nodes[461], bk);
    bootsCOPY(&result[5], temp_nodes[462], bk);
    for (auto it = temp_nodes.begin(); it != temp_nodes.end(); ++it) {
        delete_gate_bootstrapping_ciphertext(it->second);
    }
}
```

```
plaintext(28):do or do not there is no try
Encryption done
Sanity check by decryption:
do or do not there is no try
```

---

Server side computation:

```
char 0 done.
char 1 done.
char 2 done.
char 3 done.
char 4 done.
char 5 done.
char 6 done.
char 7 done.
char 8 done.
char 9 done.
char 10 done.
char 11 done.
char 12 done.
char 13 done.
char 14 done.
char 15 done.
char 16 done.
char 17 done.
char 18 done.
char 19 done.
char 20 done.
char 21 done.
char 22 done.
char 23 done.
char 24 done.
char 25 done.
char 26 done.
char 27 done.
Total time : 28 secs
Computation done
```

```
Decrypted result: Do Or Do Not There Is No Try
Decryption done
```

---

```
plaintext(28):do or do not there is no try
Encryption done
Sanity check by decryption:
do or do not there is no try
```

```
Server side computation:
```

```
char 0 done.
char 1 done.
char 2 done.
char 3 done.
char 4 done.
char 5 done.
char 6 done.
char 7 done.
char 8 done.
char 9 done.
char 10 done.
char 11 done.
char 12 done.
char 13 done.
char 14 done.
char 15 done.
char 16 done.
char 17 done.
char 18 done.
char 19 done.
char 20 done.
char 21 done.
char 22 done.
char 23 done.
char 24 done.
char 25 done.
char 26 done.
char 27 done.
Total time : 28 secs
Computation done
```

```
Decrypted result: Do Or Do Not There Is No Try
Decryption done
```

# Execution engine: Multi-threaded Interpreter

```
plaintext size: 32  
plaintext: 'do or do not there is no try'  
Encryption done  
Initial state check by decryption:  
do or do not there is no try
```

Starting!

Server side computation:

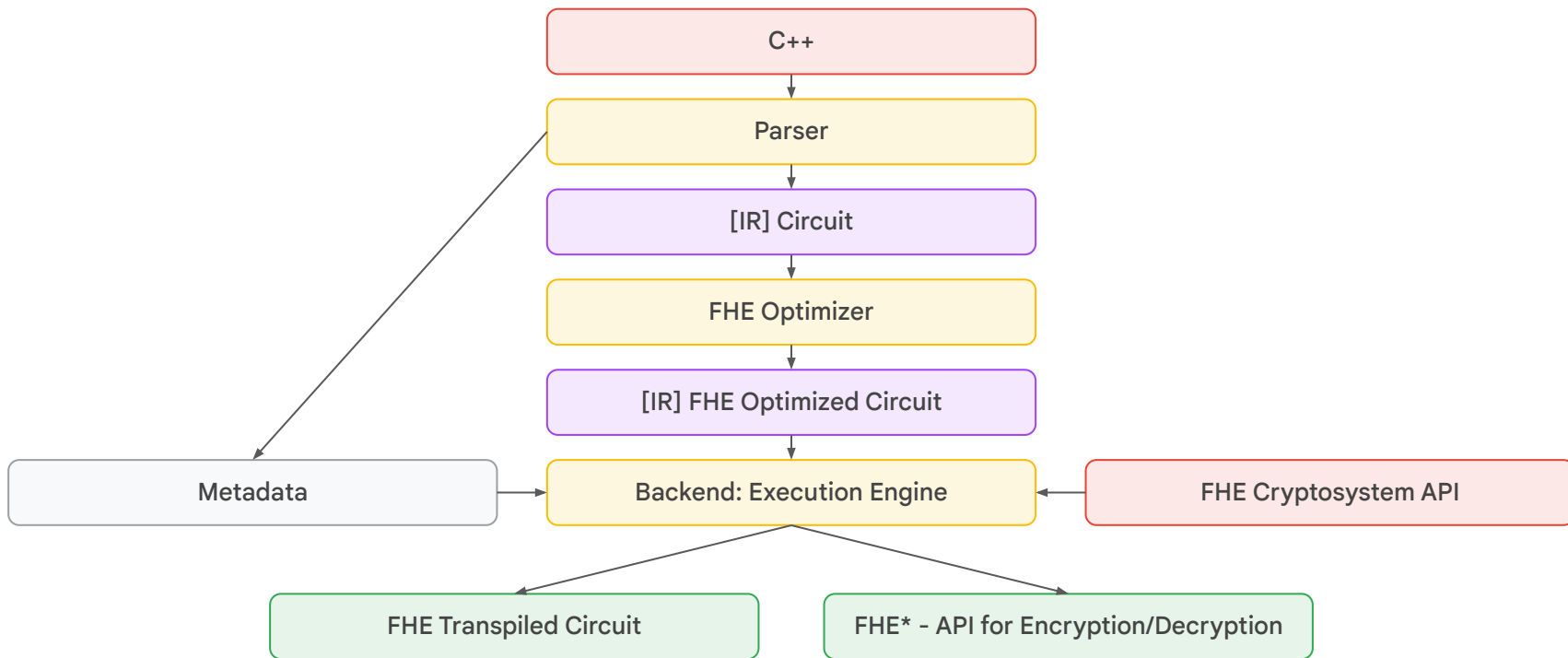
Total time: 0.756137 secs

CPU time: 46.4172 secs

Computation done

```
Decrypted result: Do Or Do Not There Is No Try  
Decryption done
```

# Transpiler: Putting it all together



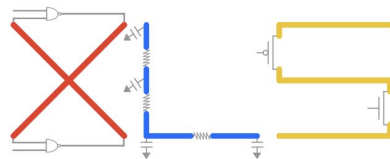


# Modular architecture

## FHE cryptosystems



## Optimizers



XLS: Accelerated HW Synthesis





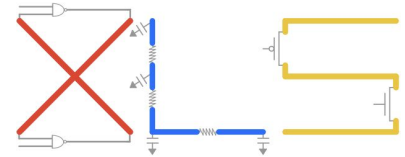
# Modular architecture

Present

## FHE cryptosystems



## Optimizers



XLS: Accelerated HW Synthesis



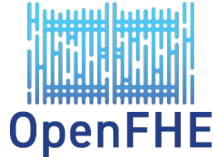
Yosys: Yosys Open SYNthesis Suite



# Modular architecture

Future

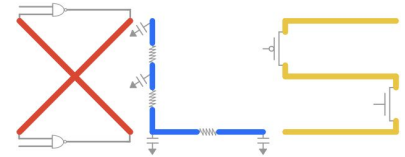
## FHE cryptosystems



ZAMA  
**Concrete**

**CuFHE**

## Optimizers



XLS: Accelerated HW Synthesis



Yosys: Yosys Open  
SYnthesis Suite



# Future Directions Supporting Arithmetic Programs

## Parser

Tensorflow

## FHE Optimizer : Ciphertext maintenance

Rescaling, Relinearization, Bootstrapping

## Intermediate representations

MLIR, EVA-IR

## Execution Engine: Levelled schemes

BGV, BFV, CKKS



# Transpiler as Framework for Benchmarks



Applications



FHE libraries



Hardware



Optimizers