OperatorToC++: Transpiling Matched EFT Coefficients to Low-Level Routines

Suraj Prakash

(IFIC, University of Valencia - CSIC)

based on work in collaboration with

Sabine Kraml (CNRS, LPSC Grenoble), Andre Lessa (UFABC) and Felix Wilsch (RWTH Aachen)

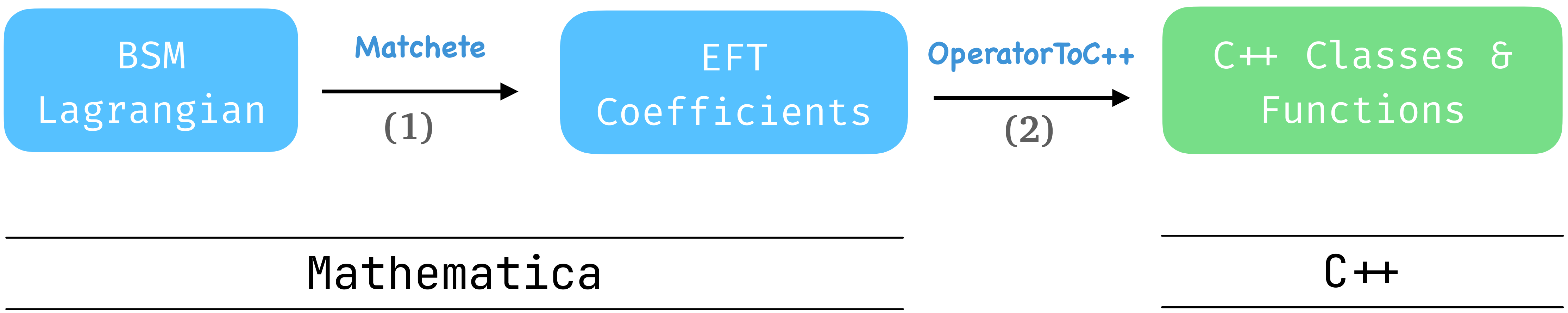
Overview

OperatorToC++

- ***What is it and what does it do?***
- ***The motivations for development and some technical niceties***
- ***Plans for further improvements and extensions***

OperatorToC++: A transpiler

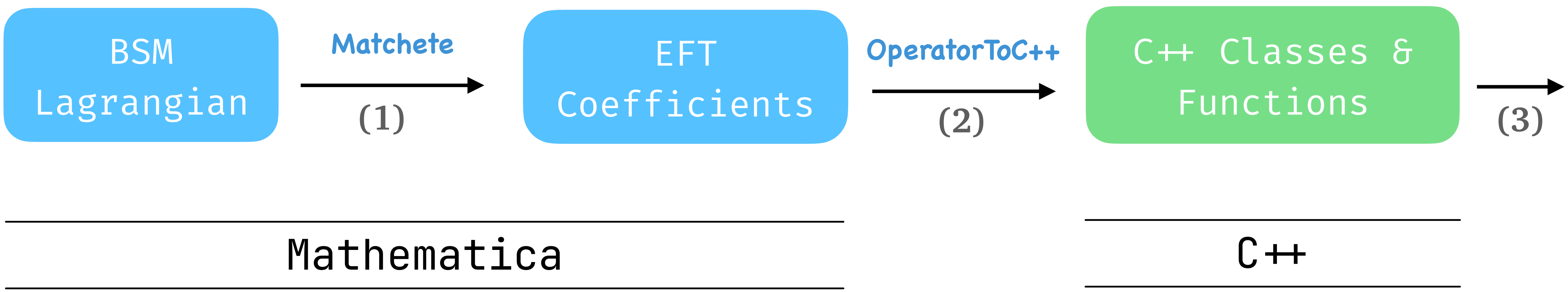
OperatorToC++: A transpiler



(1) Analytical operation

(2) Translation from Matchete API & Mathematica syntax –> C++ syntax

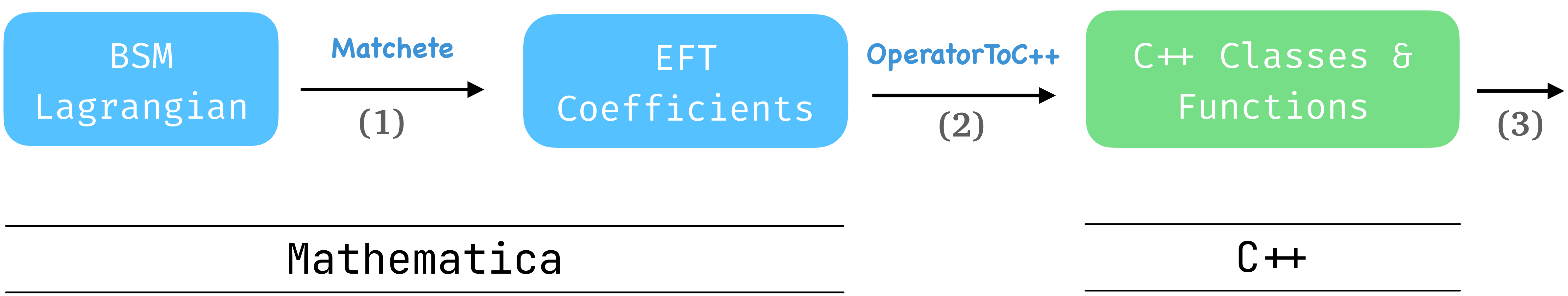
OperatorToC++: A transpiler



(1) Analytical operation

(2) Translation from Matchete API & Mathematica syntax —> C++ syntax

OperatorToC++: A transpiler

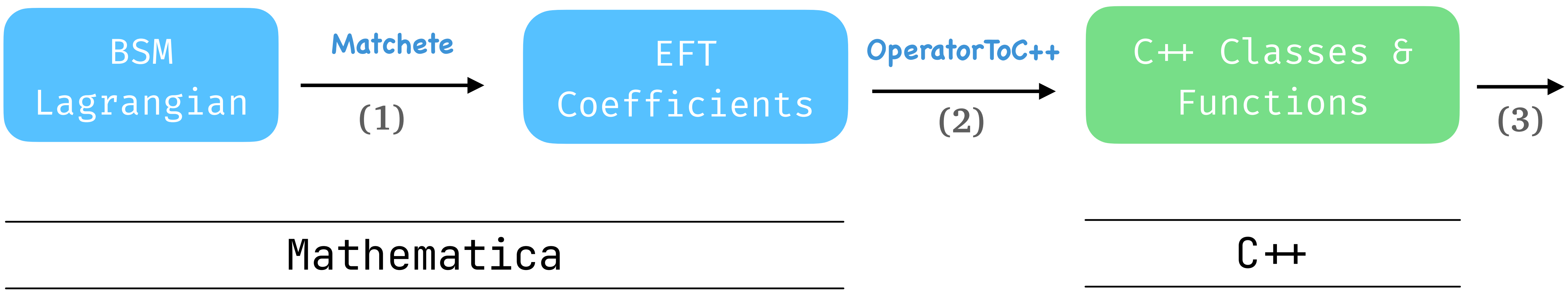


(1) Analytical operation

(2) Translation from Matchete API & Mathematica syntax –> C++ syntax

(3) Further numerical analyses

OperatorToC++: A transpiler



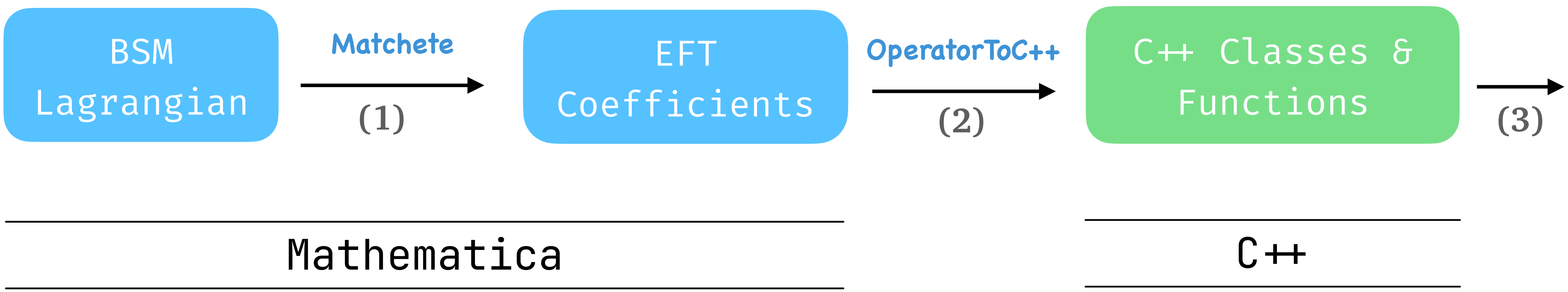
(1) Analytical operation

(2) Translation from Matchete API & Mathematica syntax –> C++ syntax

(3) Further numerical analyses

OperatorToC++ ⟶ A Mathematica & C++ based hybrid tool

OperatorToC++: A transpiler



(1) Analytical operation

(2) Translation from Matchete API & Mathematica syntax –> C++ syntax

(3) Further numerical analyses

OperatorToC++ ⟶ A Mathematica & C++ based hybrid tool

https://github.com/BSM-EFT/OperatorToCpp

OperatorToC++: A transpiler

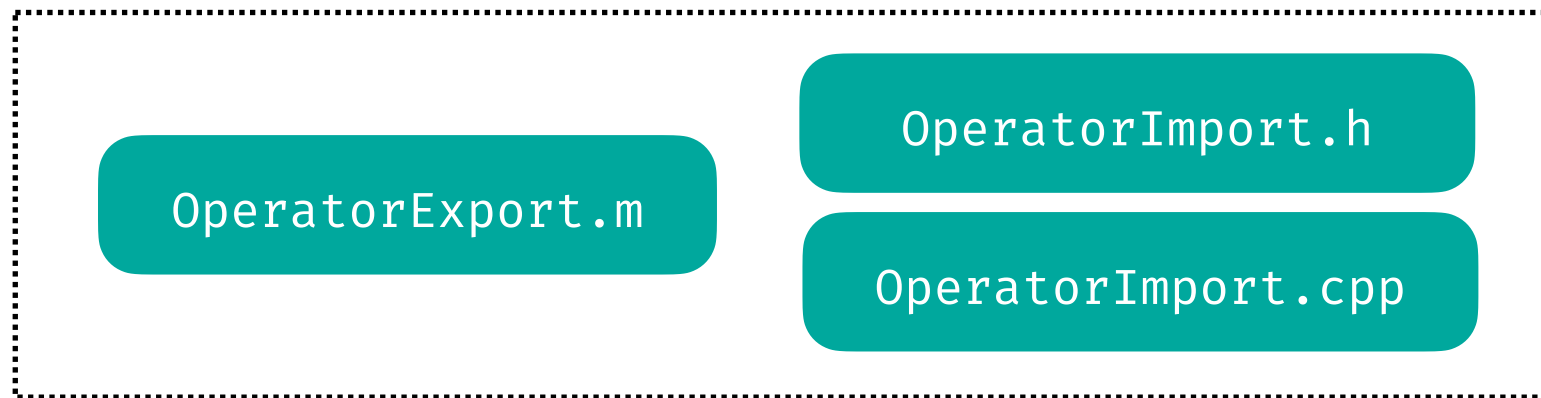
OperatorToC++: A transpiler

OperatorExport.m

OperatorImport.h

OperatorImport.cpp

OperatorToC++: A transpiler



OperatorToC++: A transpiler

`Matching-conditions.m`

file storing Matchete output
as key-value pairs

`OperatorExport.m`

`OperatorImport.h`

`OperatorImport.cpp`

OperatorToC++: A transpiler

`Matching-conditions.m`

file storing Matchete output
as key-value pairs

`OperatorExport.m`

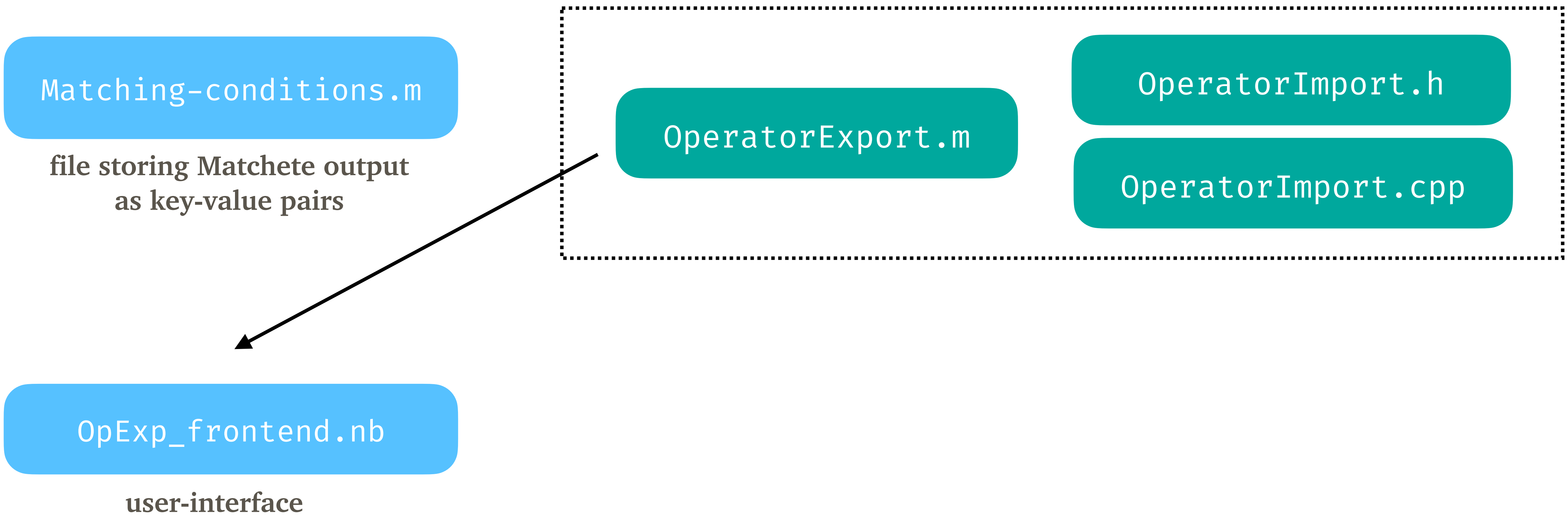
`OperatorImport.h`

`OperatorImport.cpp`

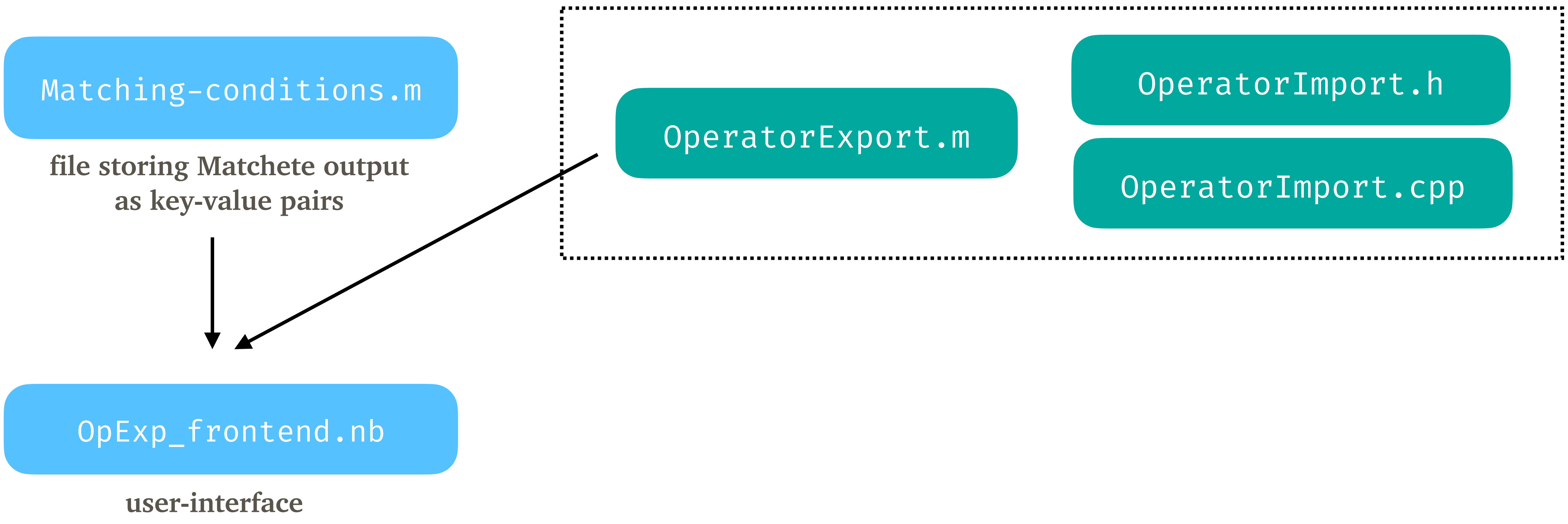
`OpExp_frontend.nb`

user-interface

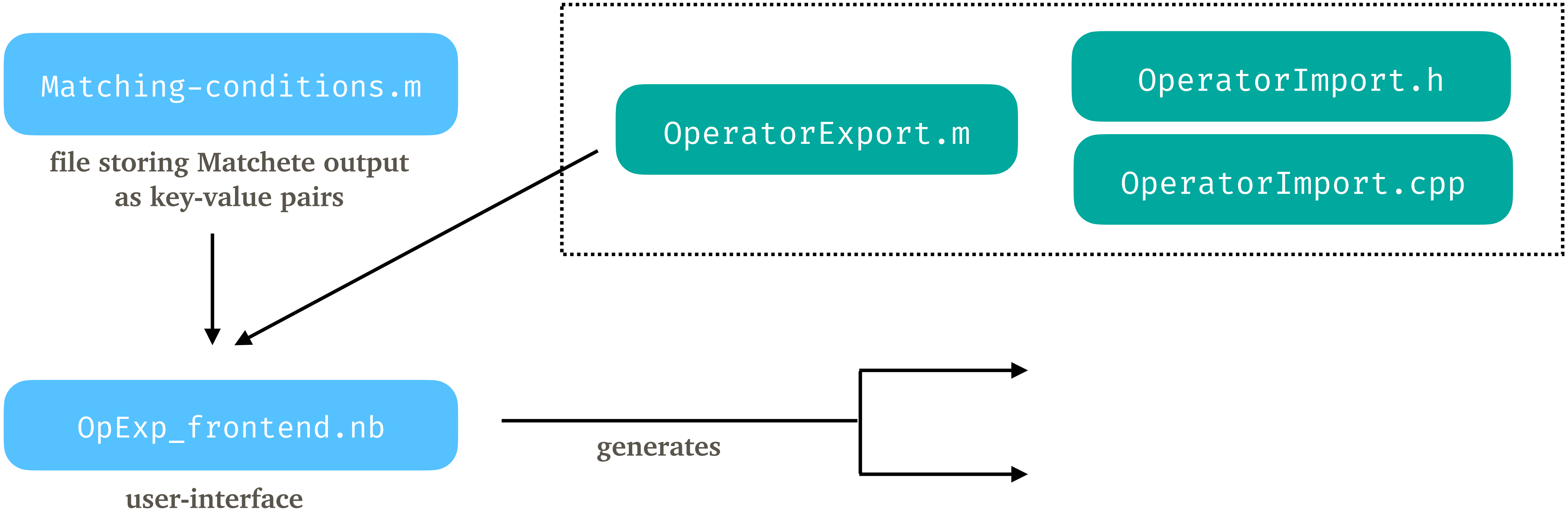
OperatorToC++: A transpiler



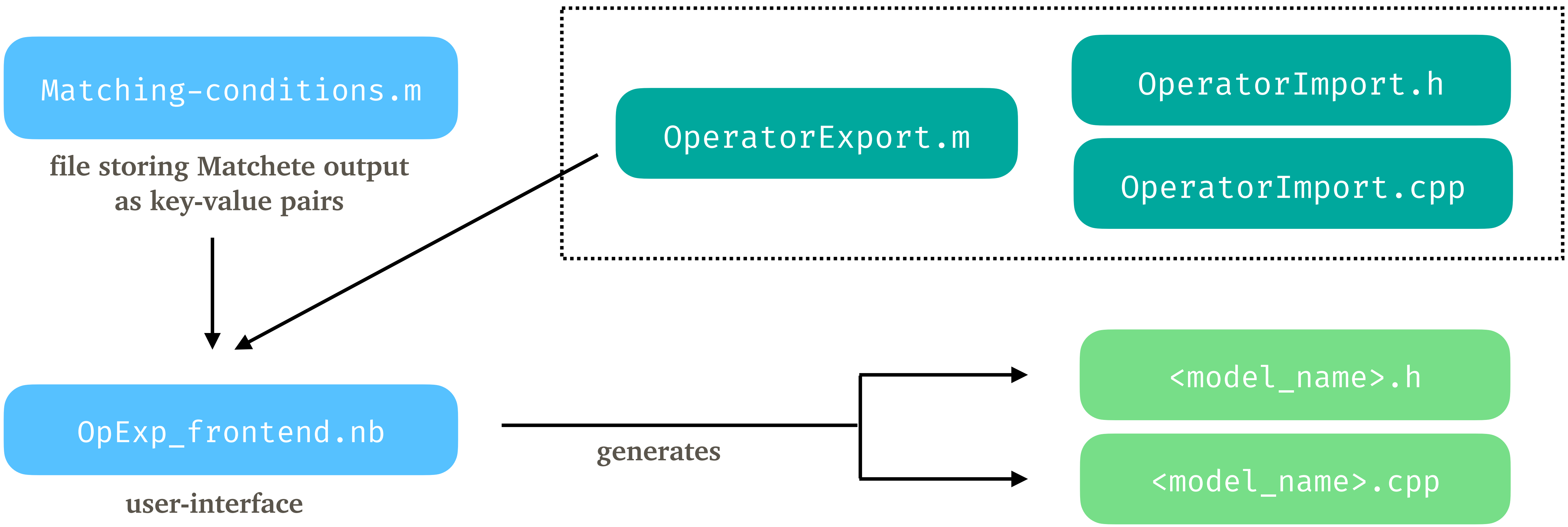
OperatorToC++: A transpiler



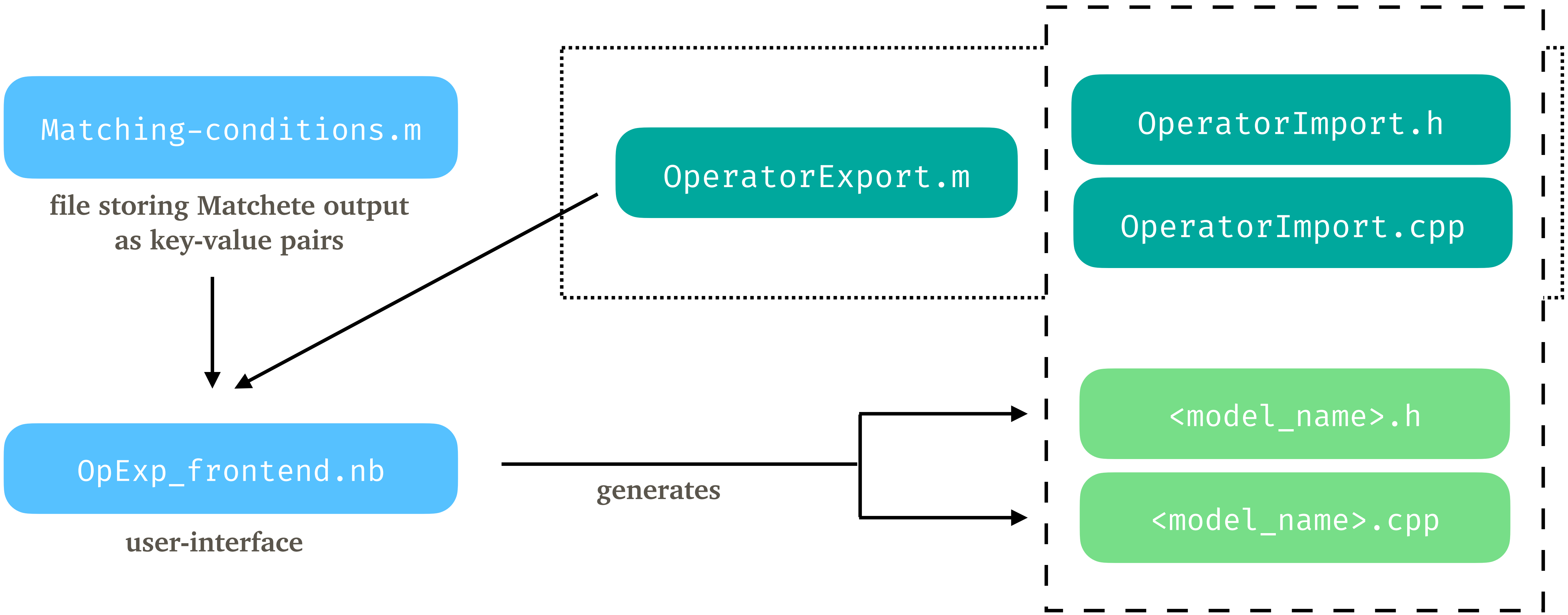
OperatorToC++: A transpiler



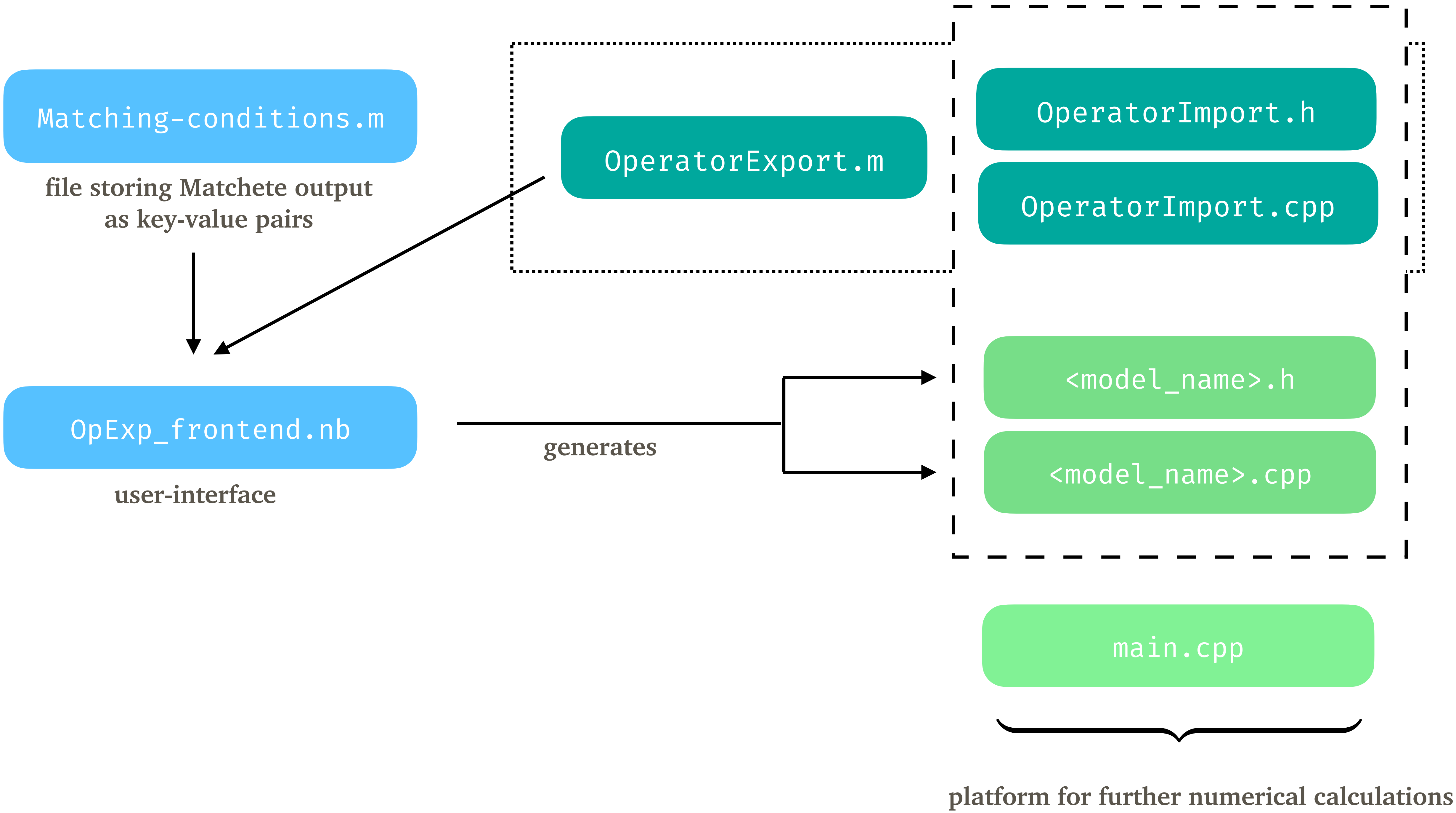
OperatorToC++: A transpiler



OperatorToC++: A transpiler



OperatorToC++: A transpiler



The User-Interface

The User-Interface

Load the OperatorExport module and the saved Machete output

```
In[1]:= OperatorExportPath = FileNameJoin[{NotebookDirectory[], "OperatorExport.m"}];

In[2]:= ImportPath = FileNameJoin[{NotebookDirectory[], "MSSM-matching-conditions.m"}];

In[3]:= Get[OperatorExportPath]

In[4]:= matchedResult = Import[ImportPath];
```

The User-Interface

Load the OperatorExport module and the saved Machete output

```
In[1]:= OperatorExportPath = FileNameJoin[{NotebookDirectory[], "OperatorExport.m"}];

In[2]:= ImportPath = FileNameJoin[{NotebookDirectory[], "MSSM-matching-conditions.m"}];

In[3]:= Get[OperatorExportPath]

In[4]:= matchedResult = Import[ImportPath];
```

Rename variables (model-dependent step) and simplification

```
In[5]:= ReplaceTrigCouplings = {
          sγ → Sqrt[(1 - cγ^2)], c2γ → (2 * cγ^2 - 1), s2γ → 2 * Sqrt[(1 - cγ^2)] * cγ,
          c4γ → (cγ^4 - 6 * cγ^2 * (1 - cγ^2) + (1 - cγ^2)^2), s4γ → 4 * cγ * Sqrt[(1 - cγ^2)] * (2 * cγ^2 - 1)};

In[6]:= variableReplacement = {
         mΦ → mPhi, MH2 → MHsq, mH2 → mHsq, μbar2 → mubarsq, μt → muTilde, λ → lmbd, cγ → cgamma,
         cHqu → yu, cHqd → yd, cHle → ye, cH2 → (-mHsq), cB2 → g1, cW2 → g2, cG2 → g3};

In[7]:= matchedResult = matchedResult /. ReplaceTrigCouplings /. variableReplacement;

In[8]:= SimplifiedOutput = EchoTiming@SimplifyOutput[matchedResult];

  152.42
```

The User-Interface (continued)

The User-Interface (continued)

Parameter extraction and file creation

```
In[9]:= ComplexParams = {ad, ae, au, yd, ye, yu};

In[10]:= ModelParams = SegregateParams[SimplifiedOutput, ComplexParams]

Out[10]= {{cgamma, g1, g2, g3, m1, m2, m3, mHsq, mPhi, muTilde}, {mdt, met, mlt, mqt, mut}, {ad, ae, au, yd, ye, yu}}

In[11]:= EchoTiming@HeaderFileBuilder["MSSM", ModelParams]

0.046271

In[12]:= EchoTiming@SourceFileBuilder["MSSM", ModelParams, ComplexParams, SimplifiedOutput]

5.75139
```

The User-Interface (continued)

Parameter extraction and file creation

```
In[9]:= ComplexParams = {ad, ae, au, yd, ye, yu};

In[10]:= ModelParams = SegregateParams[SimplifiedOutput, ComplexParams]

Out[10]= {{cgamma, g1, g2, g3, m1, m2, m3, mHsq, mPhi, muTilde}, {mdt, met, mlt, mqt, mut}, {ad, ae, au, yd, ye, yu}}

In[11]:= EchoTiming@HeaderFileBuilder["MSSM", ModelParams]

    0.046271

In[12]:= EchoTiming@SourceFileBuilder["MSSM", ModelParams, ComplexParams, SimplifiedOutput]

    5.75139
```

For the MSSM, the entire operation takes less than 3 minutes on an M3 MBP

The User-Interface (continued)

Parameter extraction and file creation

```
In[9]:= ComplexParams = {ad, ae, au, yd, ye, yu};

In[10]:= ModelParams = SegregateParams[SimplifiedOutput, ComplexParams]

Out[10]= {{cgamma, g1, g2, g3, m1, m2, m3, mHsq, mPhi, muTilde}, {mdt, met, mlt, mqt, mut}, {ad, ae, au, yd, ye, yu}}

In[11]:= EchoTiming@HeaderFileBuilder["MSSM", ModelParams]

    0.046271

In[12]:= EchoTiming@SourceFileBuilder["MSSM", ModelParams, ComplexParams, SimplifiedOutput]

    5.75139
```

For the MSSM, the entire operation takes less than 3 minutes on an M3 MBP

For minimal extensions of the SM, it takes only a few seconds.

Generated C++ (header) file

```
class MSSM {
  private:
    double cgamma = 0.0;
    double g1 = 0.0;
    double g2 = 0.0;
    double g3 = 0.0;
    double m1 = 0.0;
    double m2 = 0.0;
    double m3 = 0.0;
    double mHsq = 0.0;
    double mPhi = 0.0;
    double muTilde = 0.0;

    std::vector<double> mdt = {0.0, 0.0, 0.0};
    std::vector<double> met = {0.0, 0.0, 0.0};
    std::vector<double> mlt = {0.0, 0.0, 0.0};
    std::vector<double> mqt = {0.0, 0.0, 0.0};
    std::vector<double> mut = {0.0, 0.0, 0.0};

    std::vector<std::vector<double> > ad = {{0.0, 0.0, 0.0},{0.0, 0.0, 0.0},{0.0, 0.0, 0.0}};
    std::vector<std::vector<double> > ae = {{0.0, 0.0, 0.0},{0.0, 0.0, 0.0},{0.0, 0.0, 0.0}};
    std::vector<std::vector<double> > au = {{0.0, 0.0, 0.0},{0.0, 0.0, 0.0},{0.0, 0.0, 0.0}};
    std::vector<std::vector<double> > yd = {{0.0, 0.0, 0.0},{0.0, 0.0, 0.0},{0.0, 0.0, 0.0}};
    std::vector<std::vector<double> > ye = {{0.0, 0.0, 0.0},{0.0, 0.0, 0.0},{0.0, 0.0, 0.0}};
    std::vector<std::vector<double> > yu = {{0.0, 0.0, 0.0},{0.0, 0.0, 0.0},{0.0, 0.0, 0.0}};
```

```
public:
    MSSM() = default;

    MSSM(std::map<std::string, double> params);

    void updateParams(std::map<std::string, double> params);

    void printParamNames();

    void printParams();

    double cllHH(int i1, int i2, double mubarsq);
    double cG(double mubarsq);
    double cW(double mubarsq);
    double cGt(double mubarsq);
    double cWt(double mubarsq);
    double cH(double mubarsq);
    double cHBox(double mubarsq);
    double cHD(double mubarsq);
    double cHG(double mubarsq);
    double cHW(double mubarsq);
    double cHB(double mubarsq);
    double cHWB(double mubarsq);
    double cHGt(double mubarsq);
    double cHWt(double mubarsq);
    double cHBt(double mubarsq);
    double cHWtB(double mubarsq);
    double ceH(int i1, int i2, double mubarsq);
    double cuH(int i1, int i2, double mubarsq);
    double cdH(int i1, int i2, double mubarsq);
    double ceW(int i1, int i2, double mubarsq);
    double ceB(int i1, int i2, double mubarsq);
    double cuG(int i1, int i2, double mubarsq);
    double cuW(int i1, int i2, double mubarsq);
    double cuB(int i1, int i2, double mubarsq);
    double cdG(int i1, int i2, double mubarsq);
    double cdW(int i1, int i2, double mubarsq);
    double cdB(int i1, int i2, double mubarsq);
```

```
double cHl1(int i1, int i2, double mubarsq);
double cHl3(int i1, int i2, double mubarsq);
double cHe(int i1, int i2, double mubarsq);
double cHq1(int i1, int i2, double mubarsq);
double cHq3(int i1, int i2, double mubarsq);
double cHu(int i1, int i2, double mubarsq);
double cHd(int i1, int i2, double mubarsq);
double cHud(int i1, int i2, double mubarsq);
double cll(int i1, int i2, int i3, int i4, double mubarsq);
double cqq1(int i1, int i2, int i3, int i4, double mubarsq);
double cqq3(int i1, int i2, int i3, int i4, double mubarsq);
double clq1(int i1, int i2, int i3, int i4, double mubarsq);
double clq3(int i1, int i2, int i3, int i4, double mubarsq);
double cee(int i1, int i2, int i3, int i4, double mubarsq);
double cuu(int i1, int i2, int i3, int i4, double mubarsq);
double cdd(int i1, int i2, int i3, int i4, double mubarsq);
double ceu(int i1, int i2, int i3, int i4, double mubarsq);
double ced(int i1, int i2, int i3, int i4, double mubarsq);
double cud1(int i1, int i2, int i3, int i4, double mubarsq);
double cud8(int i1, int i2, int i3, int i4, double mubarsq);
double cle(int i1, int i2, int i3, int i4, double mubarsq);
double clu(int i1, int i2, int i3, int i4, double mubarsq);
double cld(int i1, int i2, int i3, int i4, double mubarsq);
double cqe(int i1, int i2, int i3, int i4, double mubarsq);
double cqu1(int i1, int i2, int i3, int i4, double mubarsq);
double cqu8(int i1, int i2, int i3, int i4, double mubarsq);
double cqd1(int i1, int i2, int i3, int i4, double mubarsq);
double cqd8(int i1, int i2, int i3, int i4, double mubarsq);
double cledq(int i1, int i2, int i3, int i4, double mubarsq);
double cquqd1(int i1, int i2, int i3, int i4, double mubarsq);
double cquqd8(int i1, int i2, int i3, int i4, double mubarsq);
double clequ1(int i1, int i2, int i3, int i4, double mubarsq);
double clequ3(int i1, int i2, int i3, int i4, double mubarsq);
double cduq(int i1, int i2, int i3, int i4, double mubarsq);
double cqqu(int i1, int i2, int i3, int i4, double mubarsq);
double cqqq(int i1, int i2, int i3, int i4, double mubarsq);
double cduu(int i1, int i2, int i3, int i4, double mubarsq);
```

Motivations for development

```
Coupling[cH, {}, 0] → Coupling[cB2,{},0]^4 Coupling[s4γ,{},0]^2 / (64 Coupling[mΦ,{},0]^2) - 19 hbar Coupling[cB2,{},0]^6 Coupling[s4γ,{},0]^2 / (192 Coupling[mΦ,{},0]^2) +
  hbar Coupling[c2γ,{},0]^2 Coupling[cB2,{},0]^6 Coupling[s4γ,{},0]^2 / (128 Coupling[mΦ,{},0]^2) + Coupling[cB2,{},0]^2 Coupling[cW2,{},0]^2 Coupling[s4γ,{},0]^2 / (32 Coupling[mΦ,{},0]^2) -
  19 hbar Coupling[cB2,{},0]^4 Coupling[cW2,{},0]^2 Coupling[s4γ,{},0]^2 / (192 Coupling[mΦ,{},0]^2) + 3 hbar Coupling[c2γ,{},0]^2 Coupling[cB2,{},0]^4 Coupling[cW2,{},0]^2 Coupling[s4γ,{},0]^2 / (128 Coupling[mΦ,{},0]^2) + ··· 4982 ··· +
  1/Coupling[sγ,{},0]^2 3 hbar Bar[Coupling[au, {Index[d$$1, Flavor], Index[d$$2, Flavor]}, 0]] × Bar[Coupling[cHqu, {Index[d$$3, Flavor], Index[d$$4, Flavor]}, 0]] ×
   Bar[Coupling[cHqu, {Index[d$$5, Flavor], Index[d$$6, Flavor]}, 0]] × Coupling[au, {Index[d$$1, Flavor], Index[d$$4, Flavor]}, 0] ×
   Coupling[cHqu, {Index[d$$3, Flavor], Index[d$$6, Flavor]}, 0] × Coupling[cHqu, {Index[d$$5, Flavor], Index[d$$2, Flavor]}, 0] Coupling[cγ, {}, 0]^4
   Coupling[μt, {}, 0]^4 LF[{Coupling[mqt, {Index[d$$1, Flavor]}, 0], Coupling[mqt, {Index[d$$3, Flavor]}, 0], Coupling[mqt, {Index[d$$5, Flavor]}, 0],
     Coupling[mut, {Index[d$$2, Flavor]}, 0], Coupling[mut, {Index[d$$4, Flavor]}, 0], Coupling[mut, {Index[d$$6, Flavor]}, 0]}, {1, 1, 1, 1, 1, 1, 0}] +
  1/Coupling[sγ,{},0]^2 3 hbar Bar[Coupling[au, {Index[d$$1, Flavor], Index[d$$2, Flavor]}, 0]] × Bar[Coupling[au, {Index[d$$3, Flavor], Index[d$$4, Flavor]}, 0]] ×
   Bar[Coupling[cHqu, {Index[d$$5, Flavor], Index[d$$6, Flavor]}, 0]] × Coupling[cHqu, {Index[d$$1, Flavor], Index[d$$4, Flavor]}, 0] ×
   Coupling[cHqu, {Index[d$$3, Flavor], Index[d$$6, Flavor]}, 0] × Coupling[cHqu, {Index[d$$5, Flavor], Index[d$$2, Flavor]}, 0] Coupling[cγ, {}, 0]^4
   Coupling[μt, {}, 0]^4 LF[{Coupling[mqt, {Index[d$$1, Flavor]}, 0], Coupling[mqt, {Index[d$$3, Flavor]}, 0], Coupling[mqt, {Index[d$$5, Flavor]}, 0],
     Coupling[mut, {Index[d$$2, Flavor]}, 0], Coupling[mut, {Index[d$$4, Flavor]}, 0], Coupling[mut, {Index[d$$6, Flavor]}, 0]}, {1, 1, 1, 1, 1, 1, 0}] +
  1/Coupling[sγ,{},0]^2 3 hbar Bar[Coupling[au, {Index[d$$1, Flavor], Index[d$$2, Flavor]}, 0]] × Bar[Coupling[cHqu, {Index[d$$3, Flavor], Index[d$$4, Flavor]}, 0]] ×
   Bar[Coupling[cHqu, {Index[d$$5, Flavor], Index[d$$6, Flavor]}, 0]] × Coupling[au, {Index[d$$3, Flavor], Index[d$$2, Flavor]}, 0] ×
   Coupling[cHqu, {Index[d$$1, Flavor], Index[d$$6, Flavor]}, 0] × Coupling[cHqu, {Index[d$$5, Flavor], Index[d$$4, Flavor]}, 0] Coupling[cγ, {}, 0]^4
   Coupling[μt, {}, 0]^4 LF[{Coupling[mqt, {Index[d$$1, Flavor]}, 0], Coupling[mqt, {Index[d$$3, Flavor]}, 0], Coupling[mqt, {Index[d$$5, Flavor]}, 0],
     Coupling[mut, {Index[d$$2, Flavor]}, 0], Coupling[mut, {Index[d$$4, Flavor]}, 0], Coupling[mut, {Index[d$$6, Flavor]}, 0]}, {1, 1, 1, 1, 1, 1, 0}] -
  1/Coupling[sγ,{},0]^4 3 hbar Bar[Coupling[cHqu, {Index[d$$1, Flavor], Index[d$$2, Flavor]}, 0]] × Bar[Coupling[cHqu, {Index[d$$3, Flavor], Index[d$$4, Flavor]}, 0]] ×
   Bar[Coupling[cHqu, {Index[d$$5, Flavor], Index[d$$6, Flavor]}, 0]] × Coupling[au, {Index[d$$1, Flavor], Index[d$$4, Flavor]}, 0] ×
   Coupling[cHqu, {Index[d$$3, Flavor], Index[d$$6, Flavor]}, 0] × Coupling[cHqu, {Index[d$$5, Flavor], Index[d$$2, Flavor]}, 0] Coupling[cγ, {}, 0]^5
   Coupling[μt, {}, 0]^5 LF[{Coupling[mqt, {Index[d$$1, Flavor]}, 0], Coupling[mqt, {Index[d$$3, Flavor]}, 0], Coupling[mqt, {Index[d$$5, Flavor]}, 0],
     Coupling[mut, {Index[d$$2, Flavor]}, 0], Coupling[mut, {Index[d$$4, Flavor]}, 0], Coupling[mut, {Index[d$$6, Flavor]}, 0]}, {1, 1, 1, 1, 1, 1, 0}] -
  1/Coupling[sγ,{},0]^4 3 hbar Bar[Coupling[au, {Index[d$$1, Flavor], Index[d$$2, Flavor]}, 0]] × Bar[Coupling[cHqu, {Index[d$$3, Flavor], Index[d$$4, Flavor]}, 0]] ×
   Bar[Coupling[cHqu, {Index[d$$5, Flavor], Index[d$$6, Flavor]}, 0]] × Coupling[cHqu, {Index[d$$1, Flavor], Index[d$$4, Flavor]}, 0] ×
   Coupling[cHqu, {Index[d$$3, Flavor], Index[d$$6, Flavor]}, 0] × Coupling[cHqu, {Index[d$$5, Flavor], Index[d$$2, Flavor]}, 0] Coupling[cγ, {}, 0]^5
   Coupling[μt, {}, 0]^5 LF[{Coupling[mqt, {Index[d$$1, Flavor]}, 0], Coupling[mqt, {Index[d$$3, Flavor]}, 0], Coupling[mqt, {Index[d$$5, Flavor]}, 0],
     Coupling[mut, {Index[d$$2, Flavor]}, 0], Coupling[mut, {Index[d$$4, Flavor]}, 0], Coupling[mut, {Index[d$$6, Flavor]}, 0]}, {1, 1, 1, 1, 1, 1, 0}] +
  1/Coupling[sγ,{},0]^6 hbar Bar[Coupling[cHqu, {Index[d$$1, Flavor], Index[d$$2, Flavor]}, 0]] × Bar[Coupling[cHqu, {Index[d$$3, Flavor], Index[d$$4, Flavor]}, 0]] ×
   Bar[Coupling[cHqu, {Index[d$$5, Flavor], Index[d$$6, Flavor]}, 0]] × Coupling[cHqu, {Index[d$$1, Flavor], Index[d$$4, Flavor]}, 0] ×
   Coupling[cHqu, {Index[d$$3, Flavor], Index[d$$6, Flavor]}, 0] × Coupling[cHqu, {Index[d$$5, Flavor], Index[d$$2, Flavor]}, 0] Coupling[cγ, {}, 0]^6
   Coupling[μt, {}, 0]^6 LF[{Coupling[mqt, {Index[d$$1, Flavor]}, 0], Coupling[mqt, {Index[d$$3, Flavor]}, 0], Coupling[mqt, {Index[d$$5, Flavor]}, 0],
     Coupling[mut, {Index[d$$2, Flavor]}, 0], Coupling[mut, {Index[d$$4, Flavor]}, 0], Coupling[mut, {Index[d$$6, Flavor]}, 0]}, {1, 1, 1, 1, 1, 1, 0}]
```

Challenges and Motivations for development

Challenges and Motivations for development

(1) Large number of fields in the BSM $\Rightarrow$ very intricate matching expressions.

Challenges and Motivations for development

(1) Large number of fields in the BSM ⇒ very intricate matching expressions.

Large number of terms (100s - 1000s)

Challenges and Motivations for development

(1) Large number of fields in the BSM ⇒ very intricate matching expressions.

Large number of terms (100s - 1000s)

Wide variety of loop-functions in the results

Challenges and Motivations for development

(1) Large number of fields in the BSM ⇒ very intricate matching expressions.

Large number of terms (100s - 1000s)

Wide variety of loop-functions in the results

Variable patterns of repeated index contractions

Challenges and Motivations for development

(1) Large number of fields in the BSM $\Rightarrow$ very intricate matching expressions.

Large number of terms (100s - 1000s)

Wide variety of loop-functions in the results

Variable patterns of repeated index contractions

(2) Speed of execution when doing parameter scans or numerical analyses.

Challenges and Motivations for development

(1) Large number of fields in the BSM $\Rightarrow$ very intricate matching expressions.

Large number of terms (100s - 1000s)

Wide variety of loop-functions in the results

Variable patterns of repeated index contractions

(2) Speed of execution when doing parameter scans or numerical analyses.

(3) Extensibility w.r.t. the incorporation of higher mass dimensions, higher loop order and different operator bases.

Matchete syntax → C++ syntax

Matchete syntax → C++ syntax

`Coupling[g1, { }, 0]` ⟶ `g1`

Matchete syntax → C++ syntax

`Coupling[g1, { }, 0]` ⟶ `g1`

`Coupling[g1, { }, 0]`^{2} ⟶ `pow(g1,2)`

Matchete syntax → C++ syntax

`Coupling[g1, { }, 0]` ⟶ `g1`

`Coupling[g1, { }, 0]`^{2} ⟶ `pow(g1,2)`

`Coupling[Yu, {Index[d$$1, Flavor], Index[d$$2, Flavor]}, 0]` ⟶ `Yu[r1-1][r2-1]`

Matchete syntax → C++ syntax

`Coupling[g1, { }, 0]` ⟶ `g1`

`Coupling[g1, { }, 0]`^{2} ⟶ `pow(g1,2)`

`Coupling[Yu, {Index[d$$1, Flavor], Index[d$$2, Flavor]}, 0]` ⟶ `Yu[r1-1][r2-1]`

Repeated indices `{d$$1, d$$2}` ⟶ `{r1, r2}`

Matchete syntax → C++ syntax

`Coupling[g1, { }, 0]` ⟶ `g1`

$\texttt{Coupling[g1, \{ \}, 0]}^2$ ⟶ `pow(g1,2)`

`Coupling[Yu, {Index[d$$1, Flavor], Index[d$$2, Flavor]}, 0]` ⟶ `Yu[r1-1][r2-1]`

Repeated indices `{d$$1, d$$2}` ⟶ `{r1, r2}`

`LF[{Coupling[m1, {}, 0], Coupling[m1, {}, 0]}, {1,1,0}]` ⟶ `LF({m1, m2}, uid, mubarsq)`

Matchete syntax → C++ syntax

`Coupling[g1, { }, 0]` ⟶ `g1`

`Coupling[g1, { }, 0]`^{2} ⟶ `pow(g1,2)`

`Coupling[Yu, {Index[d$$1, Flavor], Index[d$$2, Flavor]}, 0]` ⟶ `Yu[r1-1][r2-1]`

Repeated indices `{d$$1, d$$2}` ⟶ `{r1, r2}`

`LF[{Coupling[m1, {}, 0], Coupling[m1, {}, 0]}, {1,1,0}]` ⟶ `LF({m1, m2}, uid, mubarsq)`

integer double

Matchete syntax → C++ syntax

```
Coupling[g1, { }, 0]    ⟶    g1

Coupling[g1, { }, 0]²   ⟶    pow(g1,2)

Coupling[Yu, {Index[d$$1, Flavor], Index[d$$2, Flavor]}, 0] ⟶  Yu[r1-1][r2-1]
```

Repeated indices `{d$$1, d$$2}` ⟶ `{r1, r2}`

```
LF[{Coupling[m1, {}, 0], Coupling[m1, {}, 0]}, {1,1,0}] ⟶   LF({m1, m2}, uid, mubarsq)
```

integer double

```
LF[{Coupling[mqt, {Index[d$$2, Flavor]}, 0], Coupling[mut, {Index[d$$1, Flavor]}, 0]},
    {1,1,0}] x Coupling[Yu, {Index[d$$1, Flavor], Index[d$$2, Flavor]}, 0]
    x Bar[Coupling[Yu, {Index[d$$1, Flavor], Index[d$$2, Flavor]}, 0]]
```

Matchete syntax → C++ syntax

`Coupling[g1, { }, 0]` ⟶ `g1`

`Coupling[g1, { }, 0]`^{2} ⟶ `pow(g1,2)`

`Coupling[Yu, {Index[d$$1, Flavor], Index[d$$2, Flavor]}, 0]` ⟶ `Yu[r1-1][r2-1]`

Repeated indices `{d$$1, d$$2}` ⟶ `{r1, r2}`

`LF[{Coupling[m1, {}, 0], Coupling[m1, {}, 0]}, {1,1,0}]` ⟶ `LF({m1, m2}, uid, mubarsq)`

integer double

```
LF[{Coupling[mqt, {Index[d$$2, Flavor]}, 0], Coupling[mut, {Index[d$$1, Flavor]}, 0]},
    {1,1,0}] x Coupling[Yu, {Index[d$$1, Flavor], Index[d$$2, Flavor]}, 0]
    x Bar[Coupling[Yu, {Index[d$$1, Flavor], Index[d$$2, Flavor]}, 0]]
```

⟶ `LF({mqt[r2-1], mut[r1-1]}, uid, mubarsq) * Yu[r1-1][r2-1] * Yu[r1-1][r2-1]`

(Assuming all couplings are real)

Matchete syntax → C++ syntax

```
Coupling[g1, { }, 0]  ⟶  g1

Coupling[g1, { }, 0]²  ⟶  pow(g1,2)

Coupling[Yu, {Index[d$$1, Flavor], Index[d$$2, Flavor]}, 0] ⟶ Yu[r1-1][r2-1]
```

Repeated indices `{d$$1, d$$2} ⟶ {r1, r2}`

```
LF[{Coupling[m1, {}, 0], Coupling[m1, {}, 0]}, {1,1,0}] ⟶ LF({m1, m2}, uid, mubarsq)
```

integer double

```
LF[{Coupling[mqt, {Index[d$$2, Flavor]}, 0], Coupling[mut, {Index[d$$1, Flavor]}, 0]},
    {1,1,0}] x Coupling[Yu, {Index[d$$1, Flavor], Index[d$$2, Flavor]}, 0]
    x Bar[Coupling[Yu, {Index[d$$1, Flavor], Index[d$$2, Flavor]}, 0]]

    ⟶ LF({mqt[r2-1], mut[r1-1]}, uid, mubarsq) * Yu[r1-1][r2-1] * Yu[r1-1][r2-1]
```

(Assuming all couplings are real)

How to accurately sum over repeated indices while keeping track of the order in which they appear?

Matchete syntax → C++ syntax

```
Coupling[g1, { }, 0]  ⟶  g1

Coupling[g1, { }, 0]²  ⟶  pow(g1,2)

Coupling[Yu, {Index[d$$1, Flavor], Index[d$$2, Flavor]}, 0] ⟶ Yu[r1-1][r2-1]
```

Repeated indices `{d$$1, d$$2} ⟶ {r1, r2}`

```
LF[{Coupling[m1, {}, 0], Coupling[m1, {}, 0]}, {1,1,0}] ⟶  LF({m1, m2}, uid, mubarsq)
```

integer double

```
LF[{Coupling[mqt, {Index[d$$2, Flavor]}, 0], Coupling[mut, {Index[d$$1, Flavor]}, 0]},
    {1,1,0}] x Coupling[Yu, {Index[d$$1, Flavor], Index[d$$2, Flavor]}, 0]
    x Bar[Coupling[Yu, {Index[d$$1, Flavor], Index[d$$2, Flavor]}, 0]]

    ⟶ LF({mqt[r2-1], mut[r1-1]}, uid, mubarsq) * Yu[r1-1][r2-1] * Yu[r1-1][r2-1]
```

(Assuming all couplings are real)

How to accurately sum over repeated indices while keeping track of the order in which they appear?

How to differentiate between masses with and without indices?

Matchete syntax → C++ syntax

```
Coupling[g1, { }, 0]  ⟶  g1
```

Coupling[g1, { }, 0]^2 ⟶ pow(g1,2)

```
Coupling[Yu, {Index[d$$1, Flavor], Index[d$$2, Flavor]}, 0] ⟶ Yu[r1-1][r2-1]
```

Repeated indices `{d$$1, d$$2} ⟶ {r1, r2}`

```
LF[{Coupling[m1, {}, 0], Coupling[m1, {}, 0]}, {1,1,0}] ⟶  LF({m1, m2}, uid, mubarsq)
```

integer double

```
LF[{Coupling[mqt, {Index[d$$2, Flavor]}, 0], Coupling[mut, {Index[d$$1, Flavor]}, 0]},
   {1,1,0}] x Coupling[Yu, {Index[d$$1, Flavor], Index[d$$2, Flavor]}, 0]
   x Bar[Coupling[Yu, {Index[d$$1, Flavor], Index[d$$2, Flavor]}, 0]]

  ⟶ LF({mqt[r2-1], mut[r1-1]}, uid, mubarsq) * Yu[r1-1][r2-1] * Yu[r1-1][r2-1]
```

(Assuming all couplings are real)

How to accurately sum over repeated indices while keeping track of the order in which they appear?

How to differentiate between masses with and without indices?

How to treat different quantities with indices on a similar footing?

A case study in repeated index summation

A case study in repeated index summation

Brute force approach:

A case study in repeated index summation

Brute force approach:

(1) In Mathematica - Expand everything and sum

A case study in repeated index summation

Brute force approach:

(1) In Mathematica - Expand everything and sum

```
LF[{mqt[[r2]], mut[[r1]]}, {1,1,0}] * Yu[[r1,r2]] * Yu[[r1,r2]]
= LF[{mqt[[1]], mut[[1]]}, {1,1,0}] * Yu[[1,1]] * Yu[[1,1]] +
  LF[{mqt[[1]], mut[[2]]}, {1,1,0}] * Yu[[2,1]] * Yu[[2,1]] +
  . . . +
  LF[{mqt[[3]], mut[[3]]}, {1,1,0}] * Yu[[3,3]] * Yu[[3,3]]
```

A case study in repeated index summation

Brute force approach:

(1) In Mathematica - Expand everything and sum

```
LF[{mqt[[r2]], mut[[r1]]}, {1,1,0}] * Yu[[r1,r2]] * Yu[[r1,r2]]
= LF[{mqt[[1]], mut[[1]]}, {1,1,0}] * Yu[[1,1]] * Yu[[1,1]] +
  LF[{mqt[[1]], mut[[2]]}, {1,1,0}] * Yu[[2,1]] * Yu[[2,1]] +
  . . . +
  LF[{mqt[[3]], mut[[3]]}, {1,1,0}] * Yu[[3,3]] * Yu[[3,3]]
```

$\longrightarrow$ Expand the loop-function in each term?

A case study in repeated index summation

Brute force approach:

(1) In Mathematica - Expand everything and sum

```
LF[{mqt[[r2]], mut[[r1]]}, {1,1,0}] * Yu[[r1,r2]] * Yu[[r1,r2]]
= LF[{mqt[[1]], mut[[1]]}, {1,1,0}] * Yu[[1,1]] * Yu[[1,1]] +
  LF[{mqt[[1]], mut[[2]]}, {1,1,0}] * Yu[[2,1]] * Yu[[2,1]] +
  . . . +
  LF[{mqt[[3]], mut[[3]]}, {1,1,0}] * Yu[[3,3]] * Yu[[3,3]]
```

$\longrightarrow$ Expand the loop-function in each term?

This is suboptimal

1. We end up with many more terms than before.
2. Expanding the loop-function at this stage can lead to NaNs and Infinities, if degenerate values of masses are encountered later.

A case study in repeated index summation

Brute force approach:

A case study in repeated index summation

Brute force approach:

(2) In C++: Nested "for" loops inside a wrapper function

A case study in repeated index summation

Brute force approach:

(2) In C++: Nested "for" loops inside a wrapper function

```
double EinsSum(LF({mqt, mut}, uid, mubarsq), Yu, Yu, --info about index order--) {
    double sum = 0;
    for(int r1 = 0; r1 < 3; r1++) {
        for(int r2 = 0; r2 < 3; r2++) {
             sum += LF({mqt[r2], mut[r1]}, uid, mubarsq) * Yu[r1][r2] * Yu[r1][r2];
        }
    }
    return sum;
}
```

A case study in repeated index summation

Brute force approach:

(2) In C++: Nested "for" loops inside a wrapper function

```
double EinsSum(LF({mqt, mut}, uid, mubarsq), Yu, Yu, --info about index order--) {
    double sum = 0;
    for(int r1 = 0; r1 < 3; r1++) {
        for(int r2 = 0; r2 < 3; r2++) {
            sum += LF({mqt[r2], mut[r1]}, uid, mubarsq) * Yu[r1][r2] * Yu[r1][r2];
        }
    }
    return sum;
}
```

This lacks extensibility

1. The level of nesting of the "for" loops changes on a case by case basis based on the number of repeated indices.
2. Depending on the matched expression, "EinsSum" needs to be able to work with a wide variety of input

A case study in repeated index summation

Brute force approach:

(2) In C++: Nested "for" loops inside a wrapper function

```
double EinsSum(LF({mqt, mut}, uid, mubarsq), Yu, Yu, --info about index order--) {
    double sum = 0;
    for(int r1 = 0; r1 < 3; r1++) {
        for(int r2 = 0; r2 < 3; r2++) {
            sum += LF({mqt[r2], mut[r1]}, uid, mubarsq) * Yu[r1][r2] * Yu[r1][r2];
        }
    }
    return sum;
}
```

This lacks extensibility

1. The level of nesting of the "for" loops changes on a case by case basis based on the number of repeated indices.
2. Depending on the matched expression, "EinsSum" needs to be able to work with a wide variety of input

3. To track the index order, one needs pointer-magic.

A case study in repeated index summation

A case study in repeated index summation

Our solution:

A case study in repeated index summation

Our solution:

```
double EinsSum(
v1 ⟶ { - - a vector of objects with indices - - },
v2 ⟶ { - - { }, { }, index info for each object - - },
v3 ⟶ { - - info about free indices - - }
       );
```

A case study in repeated index summation

Our solution:

```
double EinsSum(
v1 ⟶ { - - a vector of objects with indices - - },
v2 ⟶ { - - { }, { }, index info for each object - - },
v3 ⟶ { - - info about free indices - - }
     );
```

can contain 1d, 2d vectors as well as loop-function “functors”, if declared as a vector of `(std::variant)s`

A case study in repeated index summation

Our solution:

```
double EinsSum(
v1 ⟶ { - - a vector of objects with indices - - },
v2 ⟶ { - - { }, { }, index info for each object - - },
v3 ⟶ { - - info about free indices - - }
       );
```

can contain 1d, 2d vectors as well as loop-function "functors", if declared as a vector of `(std::variant)s`

We only need to create wrapper function on the Mathematica end, no expansions necessary!

A case study in repeated index summation

Our solution:

```
  double EinsSum(
v1 ⟶ { - - a vector of objects with indices - - },
v2 ⟶ { - - { }, { }, index info for each object - - },
v3 ⟶ { - - info about free indices - - }
  );
```

can contain 1d, 2d vectors as well as loop-function "functors", if declared as a vector of `(std::variant)s`

We only need to create wrapper function on the Mathematica end, no expansions necessary!

Inside the function body

```
Eval( v1[i], v4[i] )
```

A case study in repeated index summation

Our solution:

```
double EinsSum(
v1 ⟶ { - - a vector of objects with indices - - },
v2 ⟶ { - - { }, { }, index info for each object - - },
v3 ⟶ { - - info about free indices - - }
     );
```

can contain 1d, 2d vectors as well as loop-function "functors", if declared as a vector of `(std::variant)s`

We only need to create wrapper function on the Mathematica end, no expansions necessary!

Inside the function body

```
Eval( v1[i], v4[i] )

0 ⩽ i < v2.size()
```

A case study in repeated index summation

Our solution:

```
double EinsSum(
v1 ⟶ { - - a vector of objects with indices - - },⟵
v2 ⟶ { - - { }, { }, index info for each object - - },
v3 ⟶ { - - info about free indices - - }
      );
```

can contain 1d, 2d vectors as well as loop-function "functors", if declared as a vector of `(std::variant)s`

We only need to create wrapper function on the Mathematica end, no expansions necessary!

Inside the function body

```
Eval( v1[i], v4[i] )

0 ⩽ i < v2.size()
```

All combinations of values of the repeated indices

```
N = pow(num_flavours, num_indices)
↓
{0, 0, 0, 0}
{0, 0, 0, 1}
{0, 0, 0, 2}
.
.
.
{2, 2, 2, 2}
```

A case study in repeated index summation

Our solution:

```
double EinsSum(
v1 ⟶ { - - a vector of objects with indices - - },
v2 ⟶ { - - { }, { }, index info for each object - - },
v3 ⟶ { - - info about free indices - - }
     );
```

can contain 1d, 2d vectors as well as loop-function "functors", if declared as a vector of (std::variant)s

We only need to create wrapper function on the Mathematica end, no expansions necessary!

Inside the function body

```
Eval( v1[i], v4[i] )

0 ⩽ i < v2.size()
```

All combinations of values of the repeated indices

```
N = pow(num_flavours, num_indices)
```

```
{0, 0, 0, 0}
{0, 0, 0, 1}
{0, 0, 0, 2}
.
.
.
{2, 2, 2, 2}
```

+ v2
+ v3

```
{
 { {..}, {..}, },
 { {..}, {..}, },
 .
 .
 .
 { {..}, {..}, }
}
```

A case study in repeated index summation

Our solution:

```
double EinsSum(
v1 → { - - a vector of objects with indices - - },
v2 → { - - { }, { }, index info for each object - - },
v3 → { - - info about free indices - - }
     );
```

can contain 1d, 2d vectors as well as loop-function "functors", if declared as a vector of (std::variant)s

We only need to create wrapper function on the Mathematica end, no expansions necessary!

Inside the function body

```
Eval( v1[i], v4[i] )

0 ⩽ i < v2.size()
```

All combinations of values of the repeated indices

```
N = pow(num_flavours, num_indices)
↓
{0, 0, 0, 0}                 {
{0, 0, 0, 1}     + v2         { {..}, {..}, },  ← v4
{0, 0, 0, 2}     + v3         { {..}, {..}, },
.                ──→          .
.                             .
.                             .
{2, 2, 2, 2}                  { {..}, {..}, }
                             }
```

A case study in repeated index summation

Our solution:

```
double EinsSum(
v1 → { - - a vector of objects with indices - - },
v2 → { - - { }, { }, index info for each object - - },
v3 → { - - info about free indices - - }
     );
```

can contain 1d, 2d vectors as well as loop-function “functors”, if declared as a vector of (std::variant)s

We only need to create wrapper function on the Mathematica end, no expansions necessary!

Inside the function body

```
Eval( v1[i], v4[i] )

0 ⩽ i < v2.size()
```

Multiply all such `Eval()` calls for a given v4, using `std::accumulate` or `std::reduce`

All combinations of values of the repeated indices

```
N = pow(num_flavours, num_indices)
```

```
{0, 0, 0, 0}                {
{0, 0, 0, 1}     + v2         { {..}, {..}, },  ← v4
{0, 0, 0, 2}     + v3         { {..}, {..}, },
.                ⟶            .
.                             .
.                             .
{2, 2, 2, 2}                  { {..}, {..}, }
                            }
```

A case study in repeated index summation

Our solution:

```
double EinsSum(
v1 ⟶ { - - a vector of objects with indices - - },
v2 ⟶ { - - { }, { }, index info for each object - - },
v3 ⟶ { - - info about free indices - - }
      );
```

can contain 1d, 2d vectors as well as loop-function "functors", if declared as a vector of `(std::variant)s`

We only need to create wrapper function on the Mathematica end, no expansions necessary!

Inside the function body

```
Eval( v1[i], v4[i] )

0 ⩽ i < v2.size()
```

Multiply all such `Eval()` calls for a given v4, using `std::accumulate` or `std::reduce`

Only a single loop needed to iterate over all N v4's

All combinations of values of the repeated indices

```
N = pow(num_flavours, num_indices)
```

```
{0, 0, 0, 0}                {
{0, 0, 0, 1}   + v2          { {..}, {..}, },  ⟵ v4
{0, 0, 0, 2}   + v3          { {..}, {..}, },
.              ⟶             .
.                            .
.                            .
{2, 2, 2, 2}                 { {..}, {..}, }
                            }
```

A case study in repeated index summation

Our solution:

```
double EinsSum(
v1 ⟶ { - - a vector of objects with indices - - },
v2 ⟶ { - - { }, { }, index info for each object - - },
v3 ⟶ { - - info about free indices - - }
     );
```

can contain 1d, 2d vectors as well as loop-function "functors", if declared as a vector of (std::variant)s

We only need to create wrapper function on the Mathematica end, no expansions necessary!

Inside the function body

```
Eval( v1[i], v4[i] )

0 ⩽ i < v2.size()
```

Multiply all such `Eval()` calls for a given v4, using `std::accumulate` or `std::reduce`

Only a single loop needed to iterate over all N v4's
The entire construct is easily parallelizable and generalizable.

All combinations of values of the repeated indices

```
N = pow(num_flavours, num_indices)
        ↓
{0, 0, 0, 0}                    {
{0, 0, 0, 1}    + v2              { {..}, {..}, }, ⟵ v4
{0, 0, 0, 2}    + v3              { {..}, {..}, },
.               ⟶                 .
.                                 .
.                                 .
{2, 2, 2, 2}                      { {..}, {..}, }
                                }
```

Next steps

Next steps

- ***Setting up a robust IO. Interface with the .slha file format, for instance.***

Next steps

- *Setting up a robust IO. Interface with the .slha file format, for instance.*
- *Optimisations, for instance through caching in C++*

Next steps

- *Setting up a robust IO. Interface with the .slha file format, for instance.*
- *Optimisations, for instance through caching in C++*
- *Implementing non-Warsaw SMEFT operator bases*

Next steps

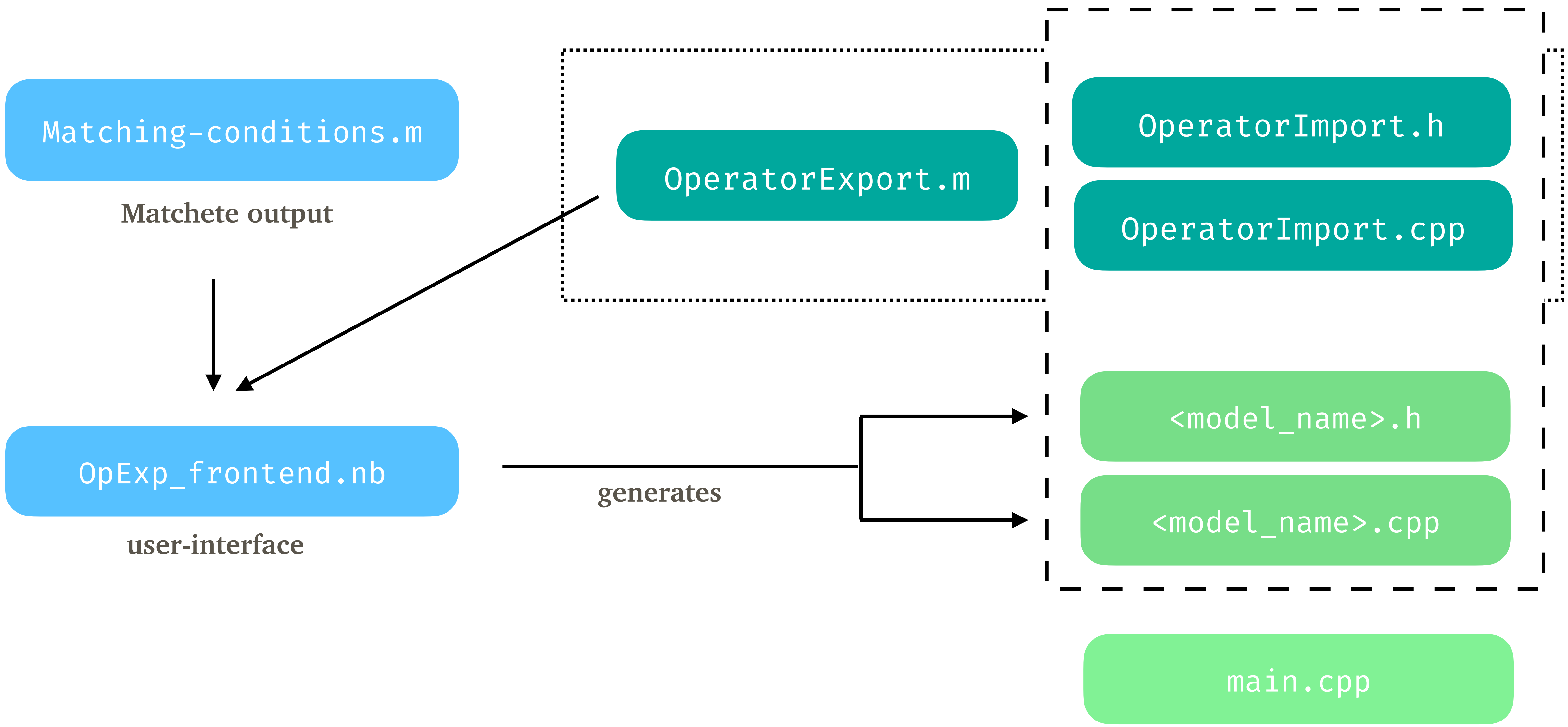
- *Setting up a robust IO. Interface with the .slha file format, for instance.*
- *Optimisations, for instance through caching in C++*
- *Implementing non-Warsaw SMEFT operator bases*
- *Going beyond dimension 6*

Next steps

- *Setting up a robust IO. Interface with the .slha file format, for instance.*
- *Optimisations, for instance through caching in C++*
- *Implementing non-Warsaw SMEFT operator bases*
- *Going beyond dimension 6*
- *Going beyond one-loop*

Thank You!

Summary



https://github.com/BSM-EFT/OperatorToCpp

platform for further numerical calculations