

Staging for Spiral: What is the gain?

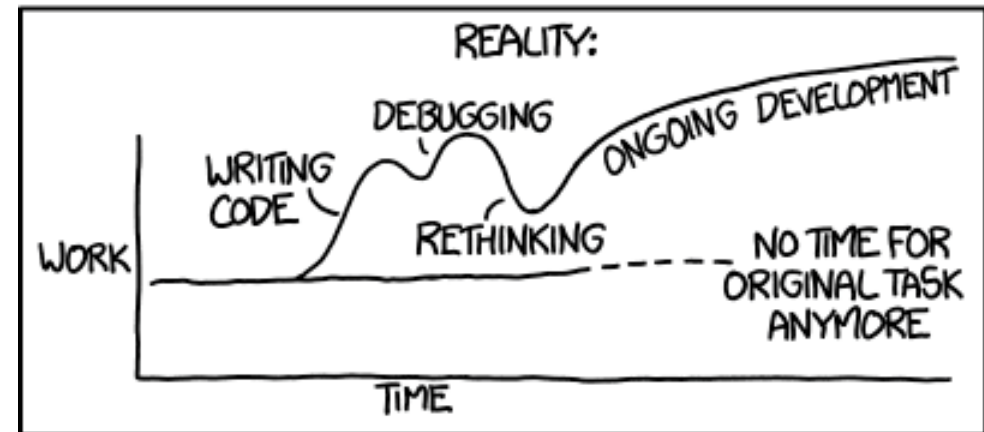
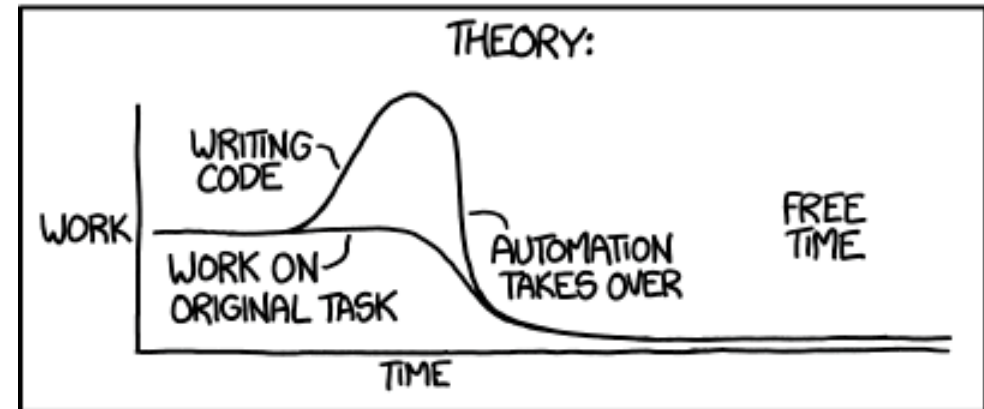
Georg Ofenbeck

*Department of Computer Science
ETH Zürich, Switzerland*

ETH zürich

SPIRAL
www.spiral.net

"I SPEND A LOT OF TIME ON THIS TASK.
I SHOULD WRITE A PROGRAM AUTOMATING IT!"



<http://xkcd.com/1319/>

Spiral

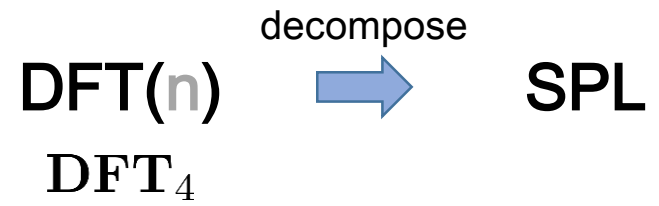
DFT(n)

Spiral

DFT(n)

DFT₄

Spiral

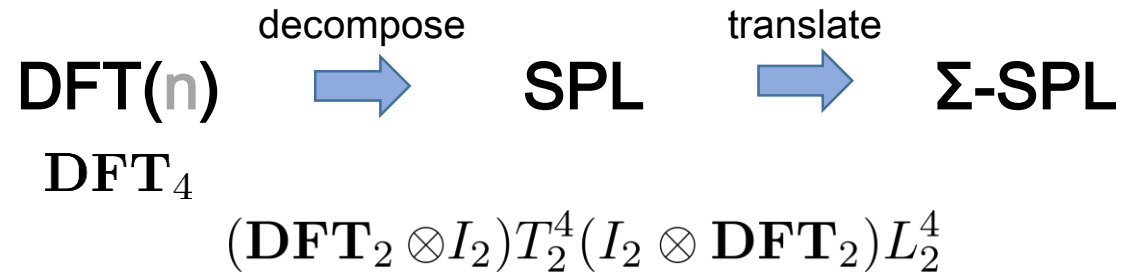


Spiral

DFT(n) $\xrightarrow{\text{decompose}}$ **SPL**

DFT₄
 $(\mathbf{DFT}_2 \otimes I_2) T_2^4 (I_2 \otimes \mathbf{DFT}_2) L_2^4$

Spiral



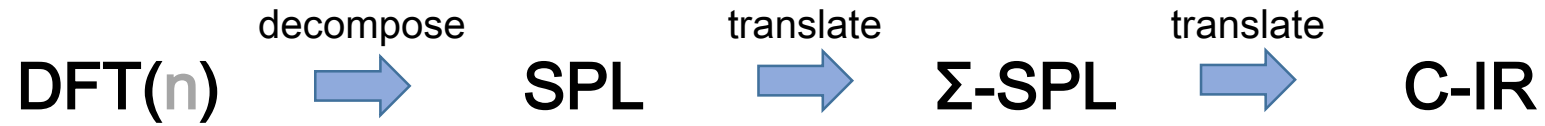
Spiral



DFT₄

$$(\mathbf{DFT}_2 \otimes I_2) T_2^4 (I_2 \otimes \mathbf{DFT}_2) L_2^4$$
$$\left(\sum_{j=0}^2 S(h_j) \mathbf{DFT}_2 G(h_j) \right) T_m^n \left(\sum_{j=0}^2 S(h_j) \mathbf{DFT}_2 G(h_j) \right) \text{perm}(\ell_2^4)$$

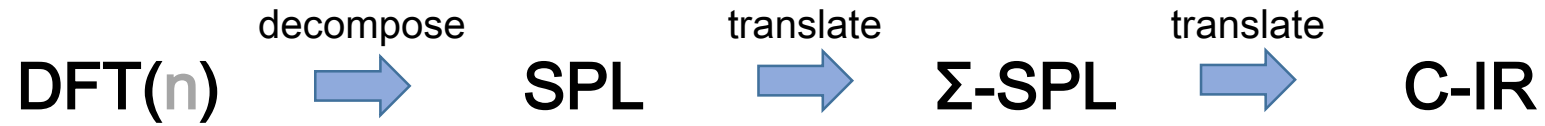
Spiral



\mathbf{DFT}_4

$$(\mathbf{DFT}_2 \otimes I_2) T_2^4 (I_2 \otimes \mathbf{DFT}_2) L_2^4 \\
 \left(\sum_{j=0}^2 S(h_j) \mathbf{DFT}_2 G(h_j) \right) T_m^n \left(\sum_{j=0}^2 S(h_j) \mathbf{DFT}_2 G(h_j) \right) \text{perm}(\ell_2^4)$$

Spiral



DFT_4

$$(\text{DFT}_2 \otimes I_2) T_2^4 (I_2 \otimes \text{DFT}_2) L_2^4$$

$$\left(\sum_{j=0}^2 S(h_j) \text{DFT}_2 G(h_j) \right) T_m^n \left(\sum_{j=0}^2 S(h_j) \text{DFT}_2 G(h_j) \right) \text{perm}(\ell_2^4)$$



Spiral



DFT_4

$$(\text{DFT}_2 \otimes I_2) T_2^4 (I_2 \otimes \text{DFT}_2) L_2^4$$

$$\left(\sum_{j=0}^2 S(h_j) \text{DFT}_2 G(h_j) \right) T_m^n \left(\sum_{j=0}^2 S(h_j) \text{DFT}_2 G(h_j) \right) \text{perm}(\ell_2^4)$$



Spiral



DFT_4

$$(\text{DFT}_2 \otimes I_2) T_2^4 (I_2 \otimes \text{DFT}_2) L_2^4$$

$$\left(\sum_{j=0}^2 S(h_j) \text{DFT}_2 G(h_j) \right) T_m^n \left(\sum_{j=0}^2 S(h_j) \text{DFT}_2 G(h_j) \right) \text{perm}(\ell_2^4)$$



```

...
for (int i=0;i < 2;i++) {
    y[i]      = x[i] + x[i*2+1];
    y[i*2+1] = x[i] - x[i*2+1];
}
for (int i=0;i < 4;i++) {
    y[i] = y[i] * sin(PI/4*(i+2));
}
...

```

Spiral



DFT₄

$$(\mathbf{DFT}_2 \otimes I_2) T_2^4 (I_2 \otimes \mathbf{DFT}_2) L_2^4 \\
 \left(\sum_{j=0}^2 S(h_j) \mathbf{DFT}_2 G(h_j) \right) T_m^n \left(\sum_{j=0}^2 S(h_j) \mathbf{DFT}_2 G(h_j) \right) \text{perm}(\ell_2^4)$$

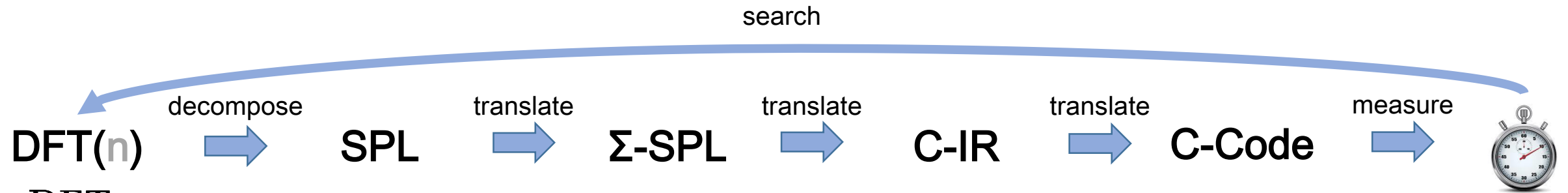


```

...
for (int i=0;i < 2;i++) {
    y[i]      = x[i] + x[i*2+1];
    y[i*2+1] = x[i] - x[i*2+1];
}
for (int i=0;i < 4;i++) {
    y[i] = y[i] * sin(PI/4*(i+2));
}
...

```

Spiral



$$\begin{aligned}
 & (\mathbf{DFT}_2 \otimes I_2) T_2^4 (I_2 \otimes \mathbf{DFT}_2) L_2^4 \\
 & \left(\sum_{j=0}^2 S(h_j) \mathbf{DFT}_2 G(h_j) \right) T_m^n \left(\sum_{j=0}^2 S(h_j) \mathbf{DFT}_2 G(h_j) \right) \text{perm}(\ell_2^4)
 \end{aligned}$$



```

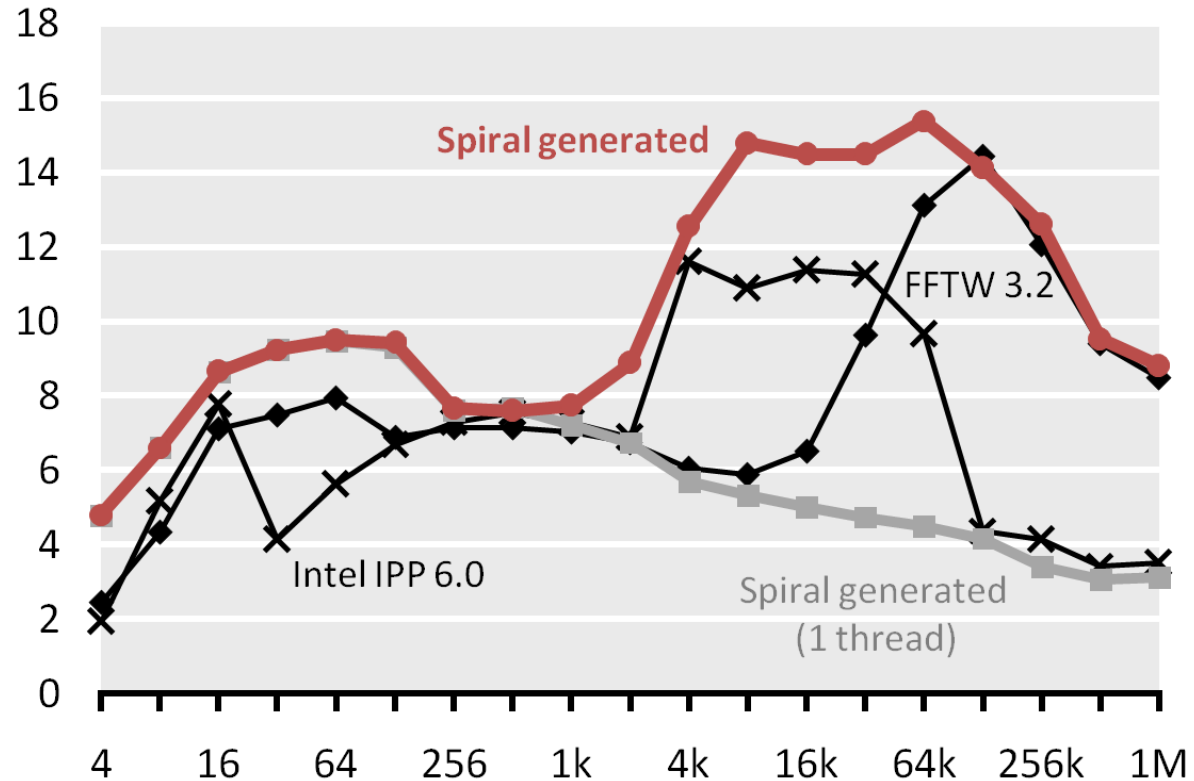
...
for (int i=0;i < 2;i++) {
    y[i]      = x[i] + x[i*2+1];
    y[i*2+1] = x[i] - x[i*2+1];
}
for (int i=0;i < 4;i++) {
    y[i] = y[i] * sin(PI/4*(i+2));
}
...

```

DFT on Intel Multicore

Complex DFT (Intel Core i7, 2.66 GHz, 4 cores)

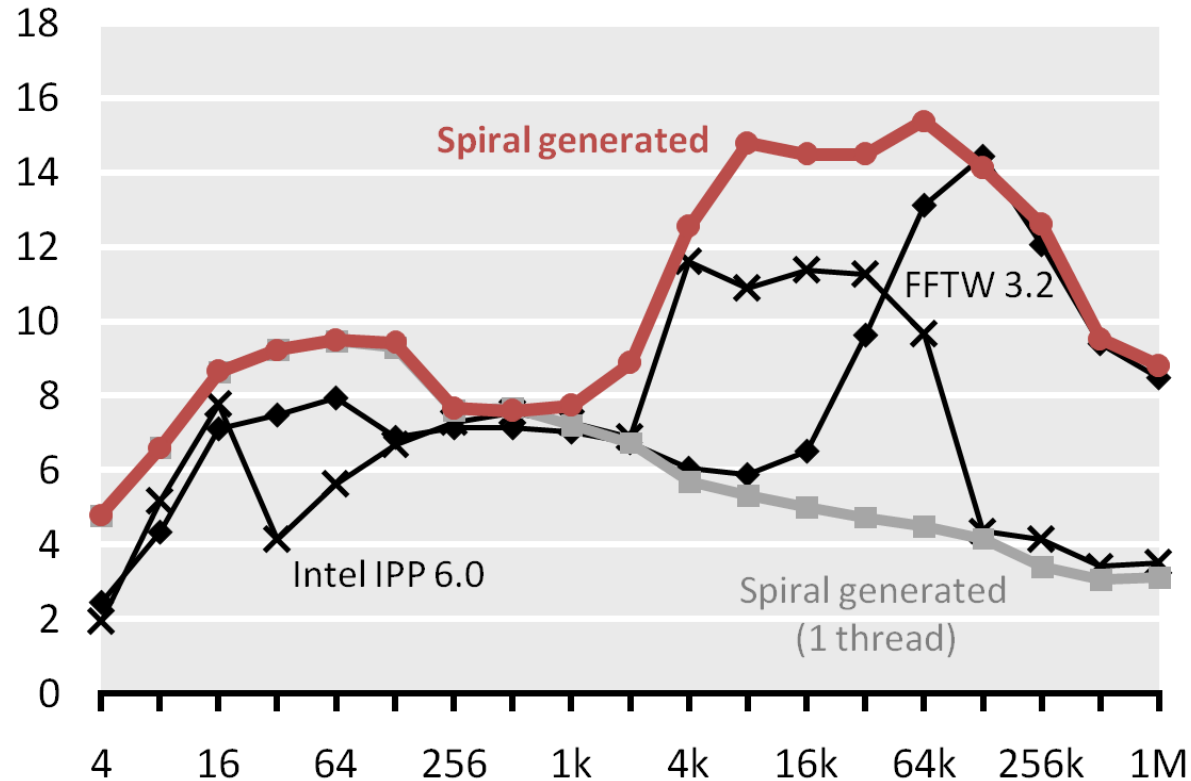
Performance [Gflop/s] vs. input size



DFT on Intel Multicore

Complex DFT (Intel Core i7, 2.66 GHz, 4 cores)

Performance [Gflop/s] vs. input size

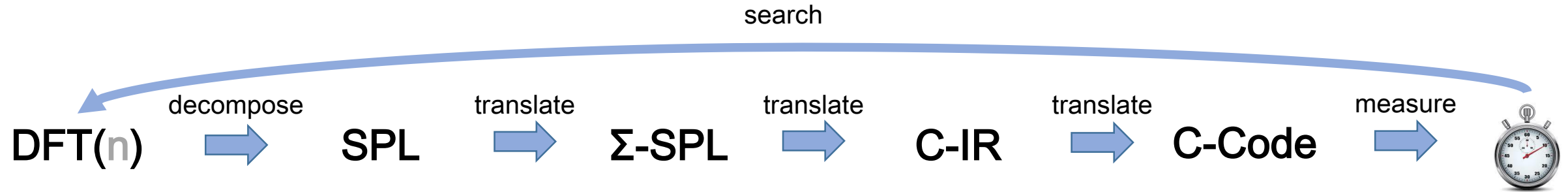


$$\begin{aligned}
 \text{DFT}_n &\rightarrow (\text{DFT}_k \otimes I_m) T_m^n (I_k \otimes \text{DFT}_m) L_k^n \\
 \text{DFT}_n &\rightarrow P_{k/2,2m}^\top (\text{DFT}_{2m} \oplus (I_{k/2-1} \otimes_i C_{2m} \text{rDFT}_{2m}(i/k))) (\text{RDFT}_k \otimes I_m) \\
 \text{RDFT}_n &\rightarrow (P_{k/2,m}^\top \otimes I_2) (\text{RDFT}_{2m} \oplus (I_{k/2-1} \otimes_i D_{2m} \text{rDFT}_{2m}(i/k))) (\text{RDFT}_k \otimes I_m) \\
 \text{rDFT}_{2n}(u) &\rightarrow L_m^{2n} (I_k \otimes_i \text{rDFT}_{2m}((i+u)/k)) (\text{rDFT}_{2k}(u) \otimes I_m)
 \end{aligned}$$



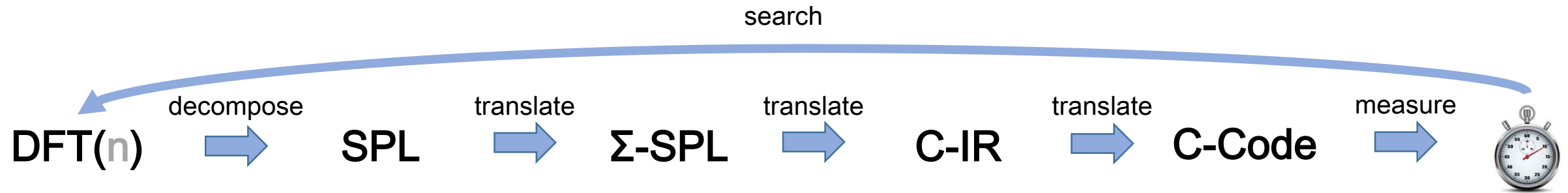
5MB vectorized, threaded,
general-size, adaptive library

My Research



Can we utilize modern program language features for:

My Research



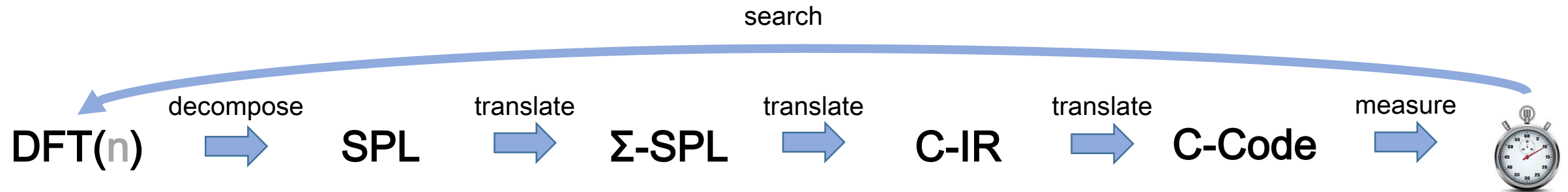
Can we utilize modern program language features for:

DSLs

Translation between DSLs

Rewrites on DSLs

My Research



Can we utilize modern program language features for:

DSLs

Translation between DSLs

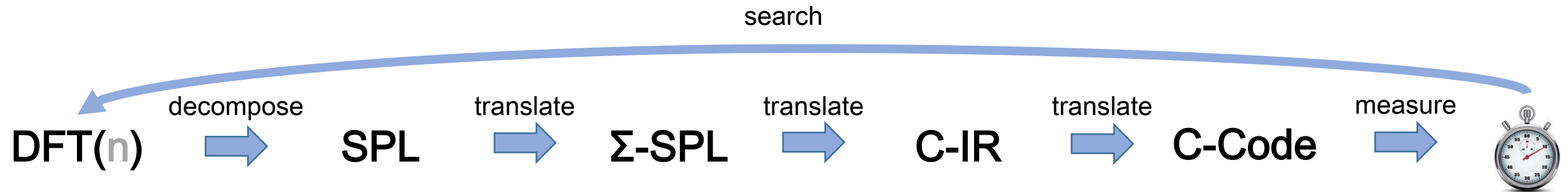
Rewrites on DSLs

Abstraction over

Low level transformations

Data layouts

My Research



Can we utilize modern program language features for:

DSLs

Translation between DSLs

Rewrites on DSLs

Abstraction over

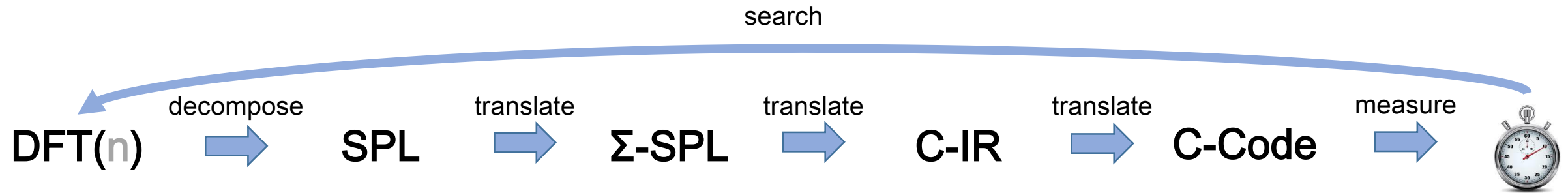
Low level transformations

Data layouts

Spiral in Scala: Towards the Systematic Construction of Generators for Performance Libraries

Georg Ofenbeck, Tiark Rompf, Alen Stojanov, Martin Odersky and Markus Püschel (Proc. International Conference on Generative Programming: Concepts & Experiences (GPCE), pp. 125-134, 2013)

My Research



Can we utilize modern program language features for:

DSLs

Translation between DSLs

Rewrites on DSLs

Abstraction over

Low level transformations

Data layouts

} This
talk

Spiral in Scala: Towards the Systematic Construction of Generators for Performance Libraries

Georg Ofenbeck, Tiark Rompf, Alen Stojanov, Martin Odersky and Markus Püschel (Proc. International Conference on Generative Programming: Concepts & Experiences (GPCE), pp. 125-134, 2013)

Code Style and Data Layouts

Σ-SPL

$$T_m^n \left(\sum_{j=0}^2 S(h_j) \mathbf{DFT}_2 G(h_j) \right)$$

Code Style and Data Layouts

Σ -SPL

Code Style

$$T_m^n \left(\sum_{j=0}^2 S(h_j) \mathbf{DFT}_2 G(h_j) \right) \rightarrow$$

Looped Code

Unrolled Code

Code with
Precomputation

Code Style and Data Layouts

Σ -SPL

Code Style

Data Layout

$$T_m^n \left(\sum_{j=0}^2 S(h_j) \mathbf{DFT}_2 G(h_j) \right)$$

Looped Code

Unrolled Code

Code with
Precomputation

C99 Complex
Interleaved Complex
Split Complex

C99 Complex
Interleaved Complex
Split Complex

C99 Complex
Interleaved Complex
Split Complex

Code Style and Data Layouts

Σ -SPL

Code Style

Data Layout

AST

$$T_m^n \left(\sum_{j=0}^2 S(h_j) \mathbf{DFT}_2 G(h_j) \right)$$



Looped Code

Unrolled Code

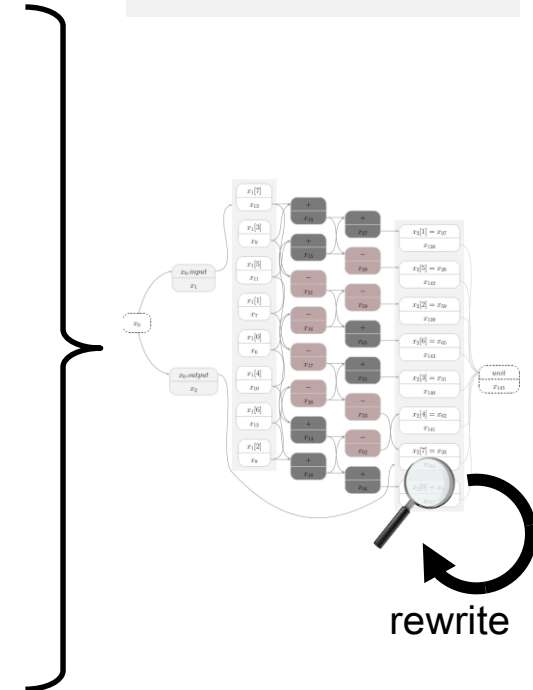
Code with
Precomputation



C99 Complex
Interleaved Complex
Split Complex

C99 Complex
Interleaved Complex
Split Complex

C99 Complex
Interleaved Complex
Split Complex



Code Style and Data Layouts

Σ -SPL

Code Style

Data Layout

AST

Code

$$T_m^n \left(\sum_{j=0}^2 S(h_j) \mathbf{DFT}_2 G(h_j) \right)$$



Looped Code



C99 Complex
Interleaved Complex
Split Complex

Unrolled Code

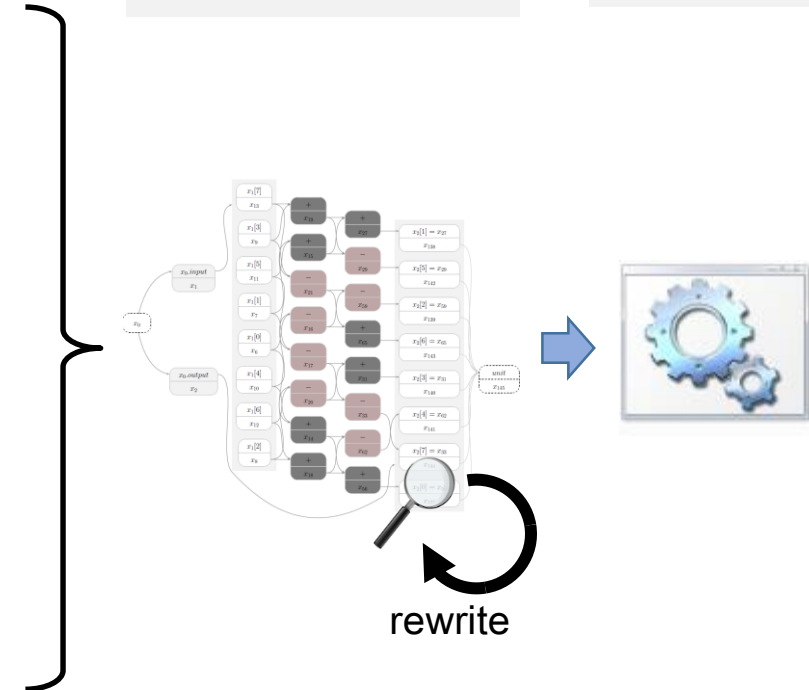


C99 Complex
Interleaved Complex
Split Complex

Code with
Precomputation

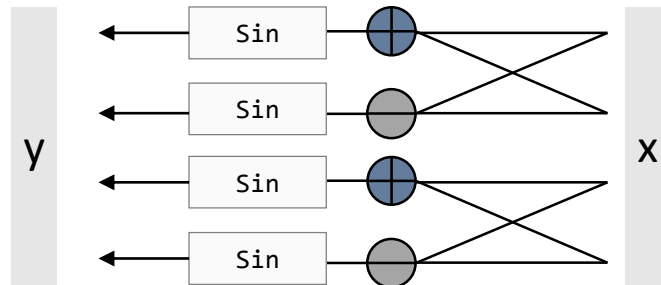


C99 Complex
Interleaved Complex
Split Complex



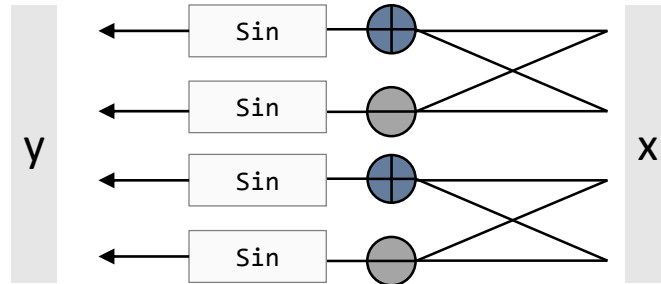
Code Style

$$T_m^n \left(\sum_{j=0}^2 S(h_j) \mathbf{DFT}_2 G(h_j) \right)$$



Code Style

$$T_m^n \left(\sum_{j=0}^2 S(h_j) \mathbf{DFT}_2 G(h_j) \right)$$

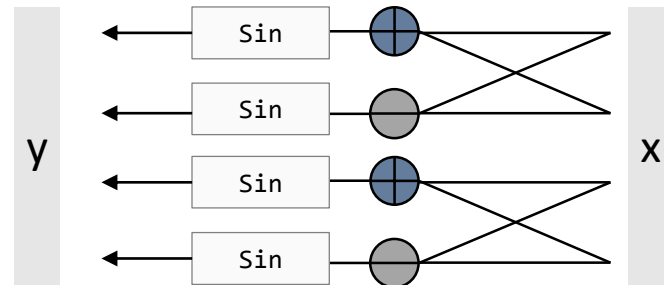


Looped

```
for (int i=0;i < 2;i++) {  
    y[i*2]    = x[i*2] + x[i*2+1];  
    y[i*2+1] = x[i*2] - x[i*2+1];  
}  
for (int i=0;i < 4;i++) {  
    y[i] = y[i] * sin(PI/4*(i+2));  
}
```

Code Style

$$T_m^n \left(\sum_{j=0}^2 S(h_j) \mathbf{DFT}_2 G(h_j) \right)$$



Looped

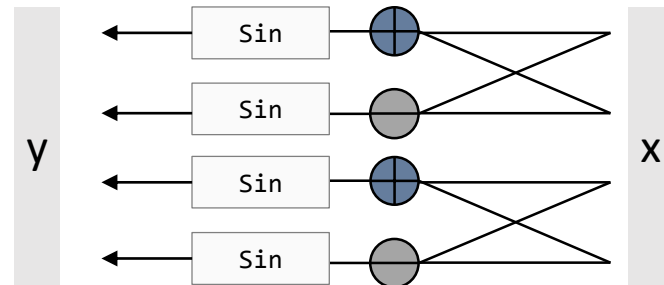
```
for (int i=0;i < 2;i++) {  
    y[i*2] = x[i*2] + x[i*2+1];  
    y[i*2+1] = x[i*2] - x[i*2+1];  
}  
for (int i=0;i < 4;i++) {  
    y[i] = y[i] * sin(PI/4*(i+2));  
}
```

Unrolled

```
y[0] = x[0] + x[1];  
y[1] = x[0] - x[1];  
y[2] = x[2] + x[3];  
y[3] = x[2] - x[3];  
  
y[0] = y[0] * sin(PI/4*(2));  
y[1] = y[1] * sin(PI/4*(3));  
y[2] = y[2] * sin(PI/4*(4));  
y[3] = y[3] * sin(PI/4*(5));
```

Code Style

$$T_m^n \left(\sum_{j=0}^2 S(h_j) \mathbf{DFT}_2 G(h_j) \right)$$



Looped

```
for (int i=0;i < 2;i++) {  
    y[i*2] = x[i*2] + x[i*2+1];  
    y[i*2+1] = x[i*2] - x[i*2+1];  
}  
for (int i=0;i < 4;i++) {  
    y[i] = y[i] * sin(PI/4*(i+2));  
}
```

Unrolled

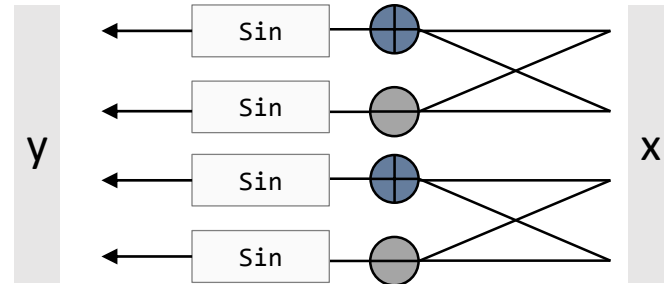
```
y[0] = x[0] + x[1];  
y[1] = x[0] - x[1];  
y[2] = x[2] + x[3];  
y[3] = x[2] - x[3];  
  
y[0] = y[0] * sin(PI/4*(2));  
y[1] = y[1] * sin(PI/4*(3));  
y[2] = y[2] * sin(PI/4*(4));  
y[3] = y[3] * sin(PI/4*(5));
```

Scalarized and Precomputed

```
t0 = x[0] + x[1];  
t1 = x[0] - x[1];  
t2 = x[2] + x[3];  
t3 = x[2] - x[3];  
  
y[0] = t0 * 1;  
y[1] = t1 * 0.707;  
y[2] = t2 * 0;  
y[3] = t3 * -0.707;
```

Data Layout

$$T_m^n \left(\sum_{j=0}^2 S(h_j) \mathbf{DFT}_2 G(h_j) \right)$$

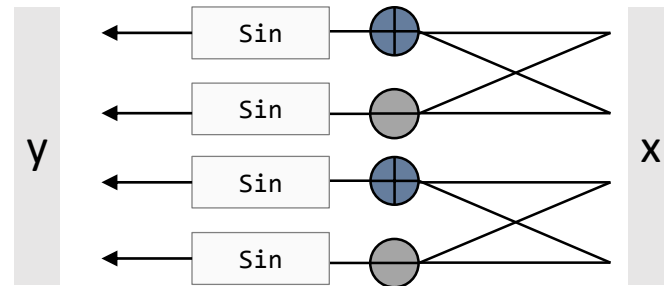


Array of Double

```
for (int i=0;i < 2;i++) {  
    y[i*2] = x[i*2] + x[i*2+1];  
    y[i*2+1] = x[i*2] - x[i*2+1];  
}  
...
```

Data Layout

$$T_m^n \left(\sum_{j=0}^2 S(h_j) \mathbf{DFT}_2 G(h_j) \right)$$



Array of Double

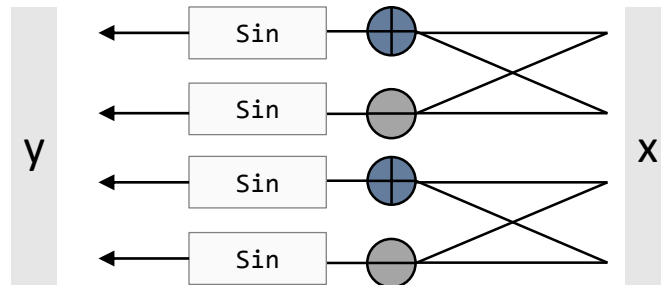
```
for (int i=0;i < 2;i++) {  
    y[i*2]    = x[i*2] + x[i*2+1];  
    y[i*2+1] = x[i*2] - x[i*2+1];  
}  
...
```

Interleaved Complex

```
for (int i=0;i < 2;i++) {  
    y[i*4]    = x[i*4]    + x[i*4+2]; //real  
    y[i*4+1] = x[i*4+1] + x[i*4+3]; //complex  
  
    y[i*4+2] = x[i*4]    - x[i*4+2]; //real  
    y[i*4+3] = x[i*4+1] - x[i*4+3]; //complex  
}  
...
```

Data Layout

$$T_m^n \left(\sum_{j=0}^2 S(h_j) \mathbf{DFT}_2 G(h_j) \right)$$



Array of Double

```
for (int i=0;i < 2;i++) {
    y[i*2]    = x[i*2] + x[i*2+1];
    y[i*2+1] = x[i*2] - x[i*2+1];
}
...
```

Interleaved Complex

```
for (int i=0;i < 2;i++) {
    y[i*4]    = x[i*4]    + x[i*4+2]; //real
    y[i*4+1]  = x[i*4+1] + x[i*4+3]; //complex

    y[i*4+2]  = x[i*4]    - x[i*4+2]; //real
    y[i*4+3]  = x[i*4+1] - x[i*4+3]; //complex
}
...
```

Split Complex

```
for (int i=0;i < 2;i++) {
    ry[i*2]    = rx[i*2] + rx[i*2+1];
    ry[i*2+1]  = rx[i*2] - rx[i*2+1];

    iy[i*2]    = ix[i*2] + ix[i*2+1];
    iy[i*2+1]  = ix[i*2] - ix[i*2+1];
}
...
```


Code Style and Data Layouts

Σ -SPL

Code Style

Data Layout

AST

Code

$$T_m^n \left(\sum_{j=0}^2 S(h_j) \mathbf{DFT}_2 G(h_j) \right)$$



Looped Code

Unrolled Code

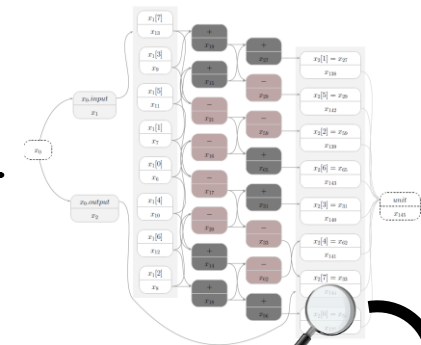
Code with
Precomputation



C99 Complex
Interleaved Complex
Split Complex

C99 Complex
Interleaved Complex
Split Complex

C99 Complex
Interleaved Complex
Split Complex



Code Style and Data Layouts

Σ -SPL

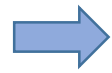
Code Style

Data Layout

AST

Code

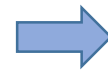
$$T_m^n \left(\sum_{j=0}^2 S(h_j) \mathbf{DFT}_2 G(h_j) \right)$$



Looped Code

Unrolled Code

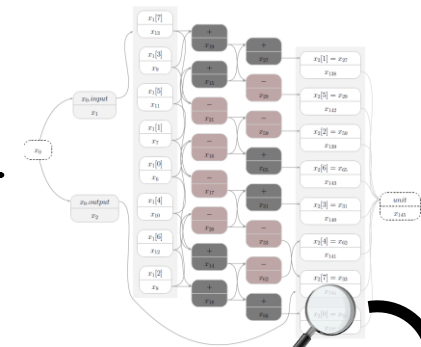
Code with
Precomputation



C99 Complex
Interleaved Complex
Split Complex

C99 Complex
Interleaved Complex
Split Complex

C99 Complex
Interleaved Complex
Split Complex



How to build this?

Option 1: "Do it yourself"

Option 2: C++ Meta programming

Option 3: Staging in Scala

...

Option 1: "Do it yourself"

Σ -SPL

Code Style

Data Layout

AST

Code

$$T_m^n \left(\sum_{j=0}^2 S(h_j) \mathbf{DFT}_2 G(h_j) \right)$$

rewrites

Looped Code

rewrites

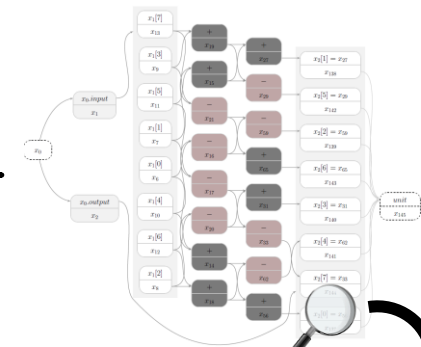
Unrolled Code

Code with
Precomputation

C99 Complex
Interleaved Complex
Split Complex

C99 Complex
Interleaved Complex
Split Complex

C99 Complex
Interleaved Complex
Split Complex



Option 1: “Do it yourself”

Σ-SPL

Code Style

Data Layout

AST

Code

$$T_m^n \left(\sum_{j=0}^2 S(h_j) \mathbf{DFT}_2 G(h_j) \right)$$

rewrites

Looped Code

rewrites

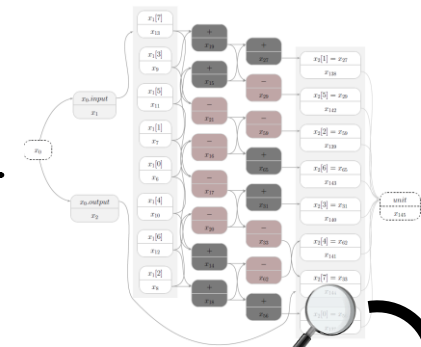
Unrolled Code

Code with
Precomputation

C99 Complex
Interleaved Complex
Split Complex

C99 Complex
Interleaved Complex
Split Complex

C99 Complex
Interleaved Complex
Split Complex



rewrite

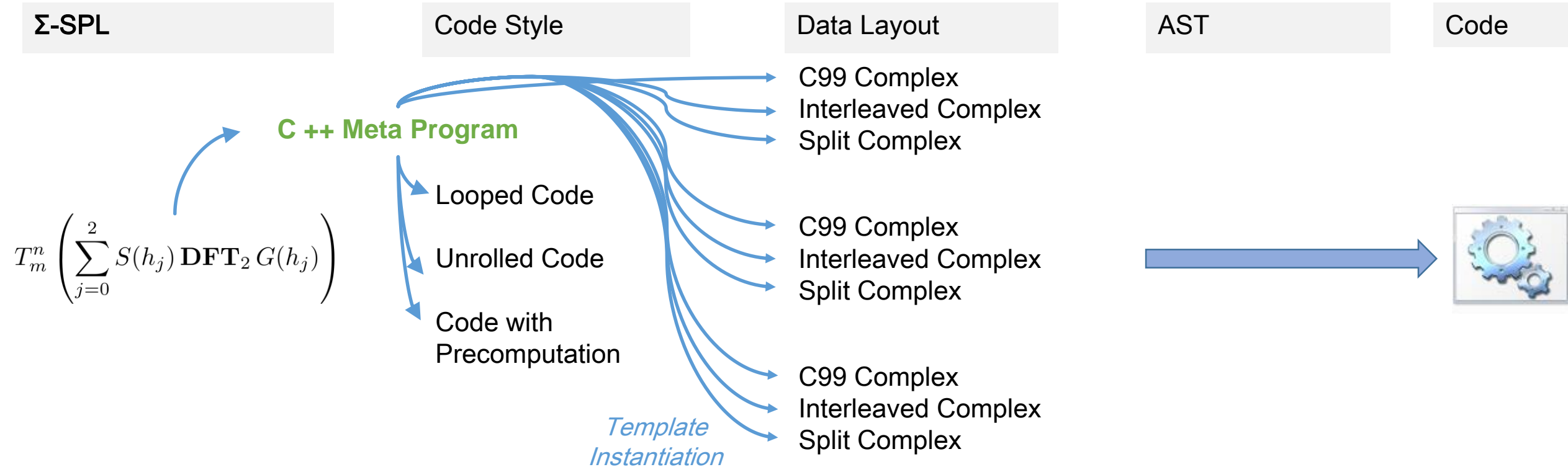


Rewrite Engine from Scratch

Large Implementation and Maintenance Effort

Little Reuse between “Backends”

Option 2: C++ Meta Programming



C++ Template Meta Programming

```
template <int T>  
struct Fibonacci{    enum { value = (Fibonacci<T - 1>::value +    Fibonacci<T - 2>::value) }; };
```

```
template <>  
struct Fibonacci<0>{    enum { value = 0 }; };
```

```
template <>  
struct Fibonacci<1>{    enum { value = 1 }; };
```

```
template <>  
struct Fibonacci<2>{    enum { value = 1 }; };
```

```
int main()  
{  
    int x = Fibonacci<45>::value;  
    cout << x << endl;  
}
```

Template Instantiation



```
int main()  
{  
    int x = 701408733;  
    cout << x << endl;  
}
```

C++ Template Meta Programming

$$T_m^n \left(\sum_{j=0}^2 S(h_j) \text{DFT}_2 G(h_j) \right)$$

C++ Meta Program

Looped Code

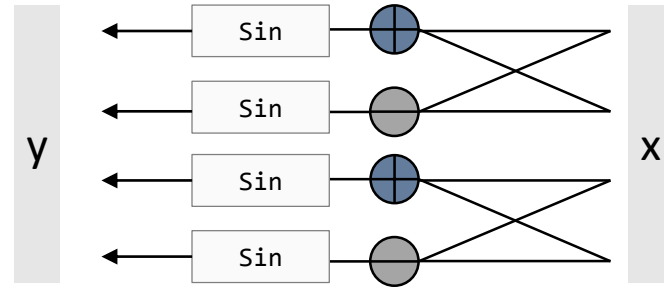
Unrolled Code

Code with Precomputation

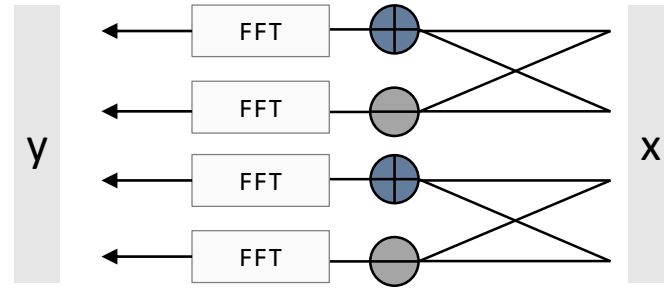
```
auto body1 = [&](const int& i) {
    y[i*2] = x[i*2] + x[i*2+1];
    y[i*2+1] = x[i*2] - x[i*2+1];
};
for(int i = 0; i < 2; i+= UnrollFact) {
    unroller( body1, i, uint_<UnrollFact - 1 >());
}
auto body2 = [&](const int& i) {
#ifdef PRECOMPUTE
    y[0] *= sin_ct(M_PI/4*(i+2));
#else
    y[0] *= sin(M_PI/4*(i+2));
#endif
};
for(int i = 0; i < 4; i+= UnrollFact) {
    unroller( body2, i, uint_<UnrollFact - 1>() );
}
```

<Templates not shown>

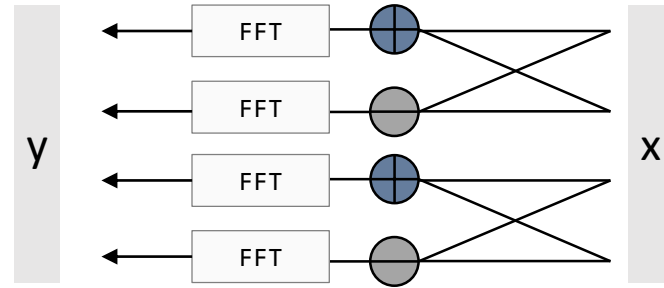
C++ Template Meta Programming



C++ Template Meta Programming

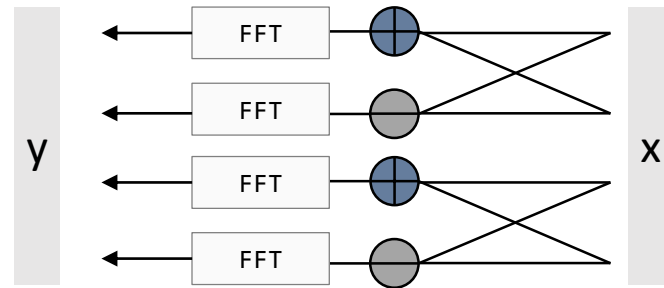


C++ Template Meta Programming



<https://github.com/pkeir/ctfft>

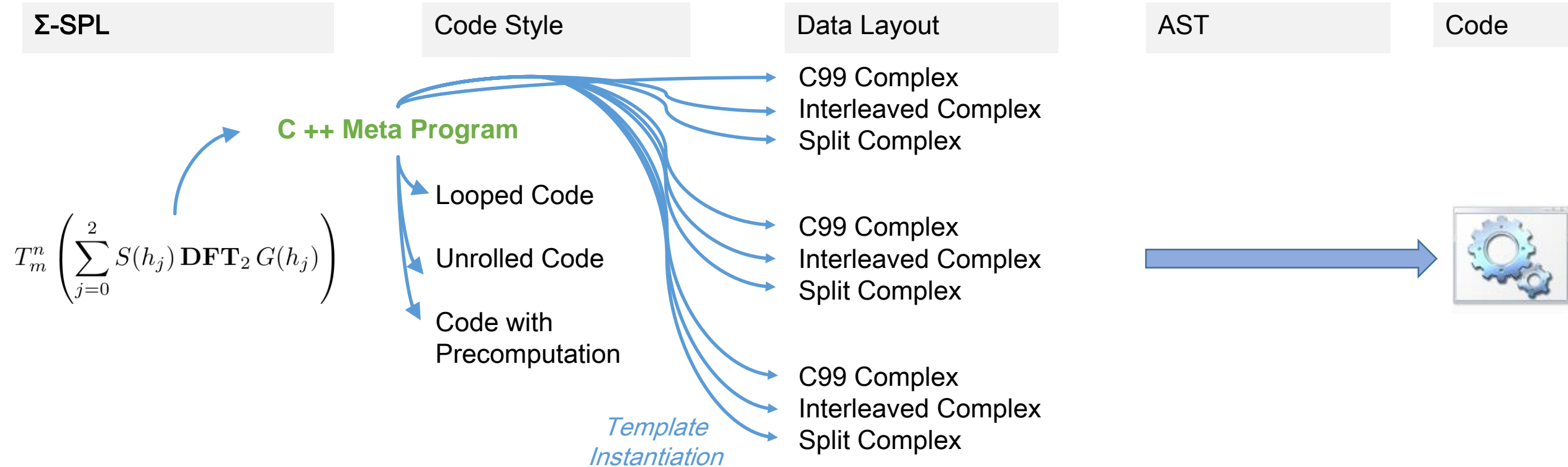
C++ Template Meta Programming



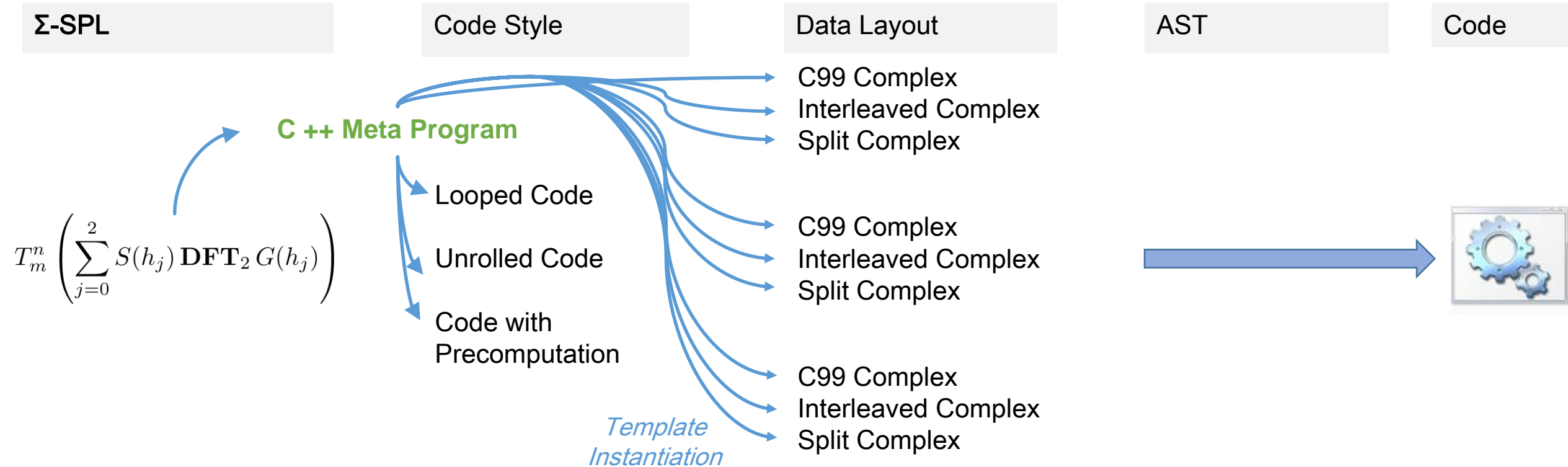
```
cffft.cpp:25:60:   in constexpr expansion of 'tupconst::map(F, std::tuple<_Elements ...>) [with F =
Comp<cxns::cx<double> (*) (const cxns::cx<double>&), Comp<Init<cxns::cx<double> >,
StaticCast<cxns::cx<double>, long unsigned int> > >; Ts = {long unsigned int, long unsigned int, long
unsigned int, long unsigned int, long unsigned int, long unsigned int, long unsigned int, long unsigned
int}; decltype (tupconst::map_helper(f, t, mk_index_range<0ul, (sizeof (Ts ...) - 1)>())) =
std::tuple<cxns::cx<double>, cxns::cx<double>, cxns::cx<double>, cxns::cx<double>, cxns::cx<double>,
cxns::cx<double>, cxns::cx<double>, cxns::cx<double> >; mk_index_range<0ul, (sizeof (Ts ...) - 1)> =
indicesT<long unsigned int, 0ul, 1ul, 2ul, 3ul, 4ul, 5ul, 6ul, 7ul>](tupconst::iota() [with long
unsigned int N = 8ul; decltype (tupconst::iota_helper(mk_index_range<0ul, (N - 1)>())) = std::tuple<long
unsigned int, long unsigned int, long unsigned int, long unsigned int, long unsigned int, long unsigned
int, long unsigned int, long unsigned int>; mk_index_range<0ul, (N - 1)> = indicesT<long unsigned int,
0ul, 1ul, 2ul, 3ul, 4ul, 5ul, 6ul, 7ul>]())'
```

<https://github.com/pkeir/ctfft>

Option 2: C++ Meta Programming



Option 2: C++ Meta Programming



C++ Meta Programming

One meta program for all targets

Debugging very difficult

Errors only appear when instantiated

By default no explicit AST

Option 3: Staging in Scala

Σ -SPL

Code Style

Data Layout

AST

Code

$$T_m^n \left(\sum_{j=0}^2 S(h_j) \mathbf{DFT}_2 G(h_j) \right)$$

Scala Staging Program

Looped Code

Unrolled Code

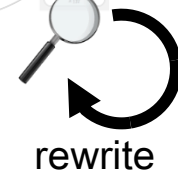
Code with
Precomputation

C99 Complex
Interleaved Complex
Split Complex

C99 Complex
Interleaved Complex
Split Complex

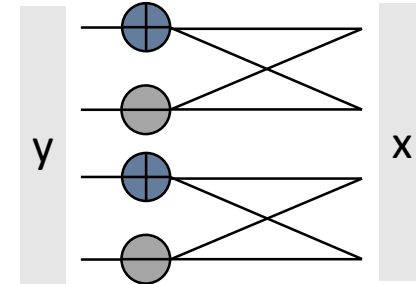
C99 Complex
Interleaved Complex
Split Complex

Staging decisions



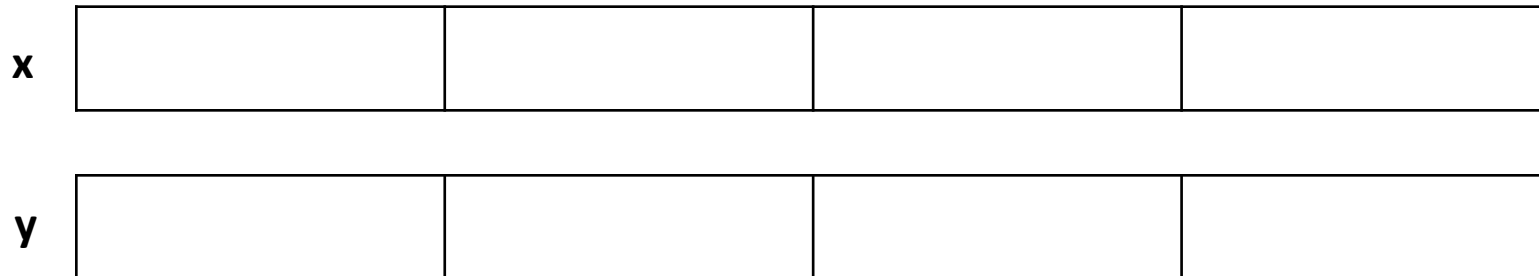
Staging Applied to the Running Example

```
def f(x: Array[Rep[Double]], y: Array[Rep[Double]]) = {  
  for (i <- 0 until 2) {  
    y(2*i) = x(i*2) + x(i*2+1)  
    y(2*i+1) = x(i*2) - x(i*2+1)  
  }  
}
```



Staging Applied to the Running Example

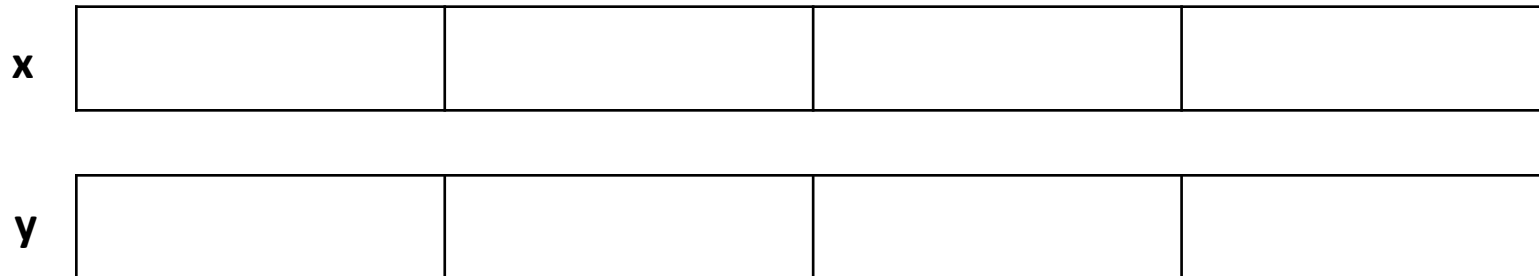
```
def f(x: Array[Rep[Double]], y: Array[Rep[Double]]) = {  
  for (i <- 0 until 2) {  
    y(2*i)    = x(i*2) + x(i*2+1)  
    y(2*i+1) = x(i*2) - x(i*2+1)  
  }  
}
```



Staging Applied to the Running Example

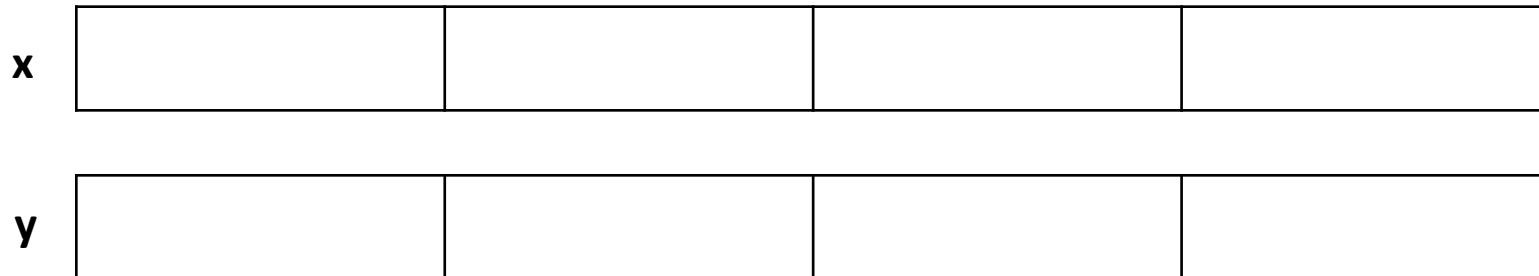
```
def f(x: Array[Rep[Double]], y: Array[Rep[Double]]) = {  
  for (i <- 0 until 2) {  
    y(2*i) = x(i*2) + x(i*2+1)  }  
}
```

i = 0



Staging Applied to the Running Example

```
def f(x: Array[Rep[Double]], y: Array[Rep[Double]]) = {  
  for (i <- 0 until 2) {  
    i = 0      y(2*i) = x(i*2) + x(i*2+1)  
  }  
}
```

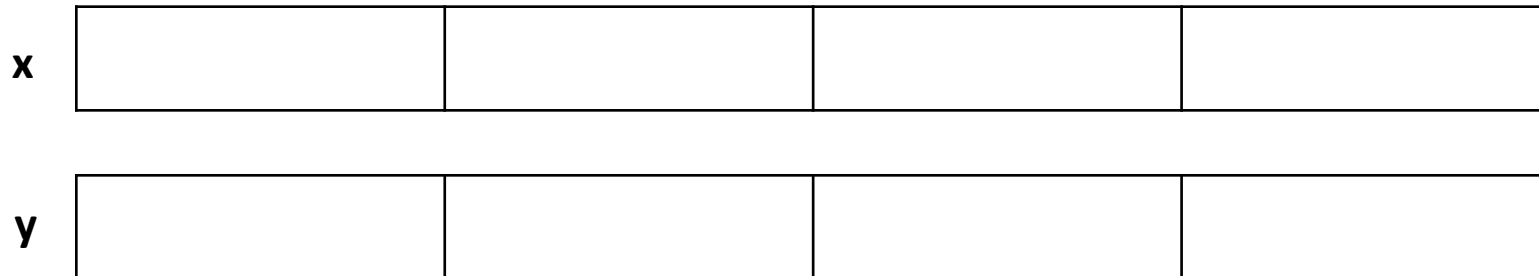


Staging Applied to the Running Example

```
def f(x: Array[Rep[Double]], y: Array[Rep[Double]]) = {  
  for (i <- 0 until 2) {  
    y(2*i) = x(i*2) + x(i*2+1)  
  }  
}
```

i = 0

$x(i*2)$
Array.apply(i: Int): T



Staging Applied to the Running Example

```
def f(x: Array[Rep[Double]], y: Array[Rep[Double]]) = {  
  for (i <- 0 until 2) {  
    y(2*i) = x(i*2) + x(i*2+1)  
  }  
}
```

Array.apply(i: Int): T

i = 0

Rep[Double]

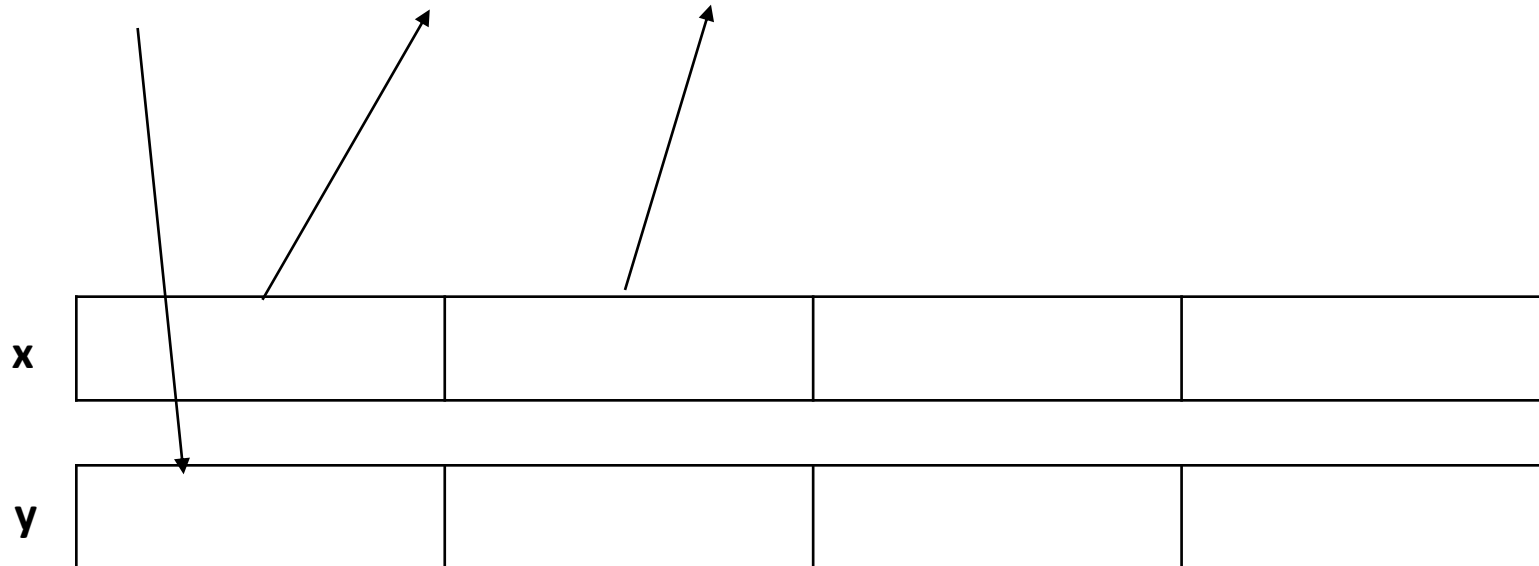


Staging Applied to the Running Example

```
def f(x: Array[Rep[Double]], y: Array[Rep[Double]]) = {  
  for (i <- 0 until 2) {  
    y(2*i) = x(i*2) + x(i*2+1)  }  
}
```

i = 0

Rep[Double] = Rep[Double] + Rep[Double]



Staging Applied to the Running Example

```
def f(x: Array[Rep[Double]], y: Array[Rep[Double]]) = {  
  for (i <- 0 until 2) {  
    i = 0      y(2*i) = x(i*2) + x(i*2+1)
```

$\text{Rep}[\text{Double}] = \underbrace{\text{Rep}[\text{Double}] + \text{Rep}[\text{Double}]}$



Staging Applied to the Running Example

```
def f(x: Array[Rep[Double]], y: Array[Rep[Double]]) = {  
  for (i <- 0 until 2) {  
    y(2*i) = x(i*2) + x(i*2+1)  }  
}
```

i = 0

Rep[Double] = Rep[Double] + Rep[Double]



How to add two “Reps”?



Staging Applied to the Running Example

```
def f(x: Array[Rep[Double]], y: Array[Rep[Double]]) = {  
  for (i <- 0 until 2) {  
    y(2*i) = x(i*2) + x(i*2+1)  
  }  
}
```

i = 0

Rep[Double] = Rep[Double] + Rep[Double]



How to add two “Reps”?

Implementation in **L**ightweight **M**odular **S**taging Library



Staging Applied to the Running Example

```
def f(x: Array[Rep[Double]], y: Array[Rep[Double]]) = {  
  for (i <- 0 until 2) {  
    y(2*i) = x(i*2) + x(i*2+1)  }  
}
```

i = 0

Rep[Double] = Rep[Double] + Rep[Double]



How to add two “Reps”?



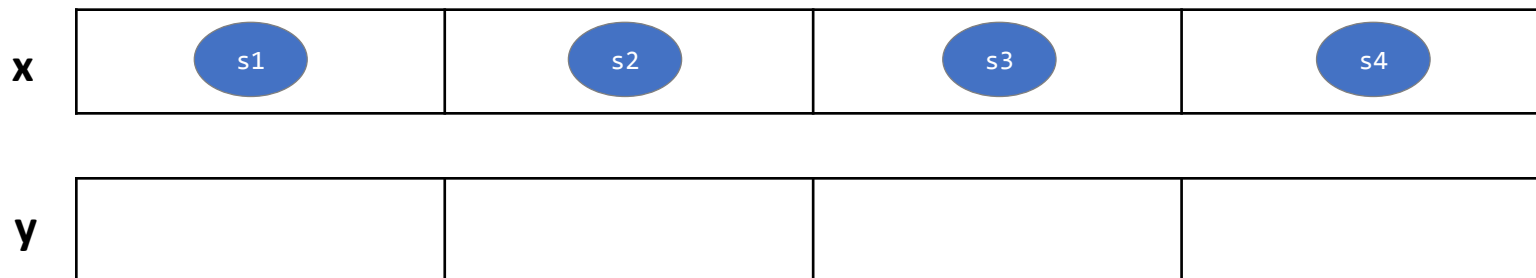
Staging Applied to the Running Example

```
def f(x: Array[Rep[Double]], y: Array[Rep[Double]]) = {  
  for (i <- 0 until 2) {  
    y(2*i) = x(i*2) + x(i*2+1)  }  
}
```

i = 0

Rep[Double] = Rep[Double] + Rep[Double]

How to add two “Reps”?



Staging Applied to the Running Example

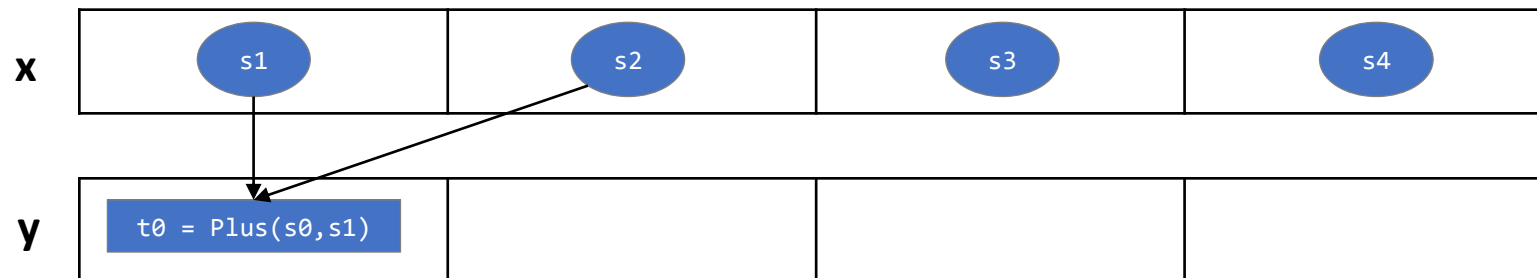
```
def f(x: Array[Rep[Double]], y: Array[Rep[Double]]) = {  
  for (i <- 0 until 2) {  
    y(2*i) = x(i*2) + x(i*2+1)  }  
}
```

i = 0

Rep[Double] = Rep[Double] + Rep[Double]



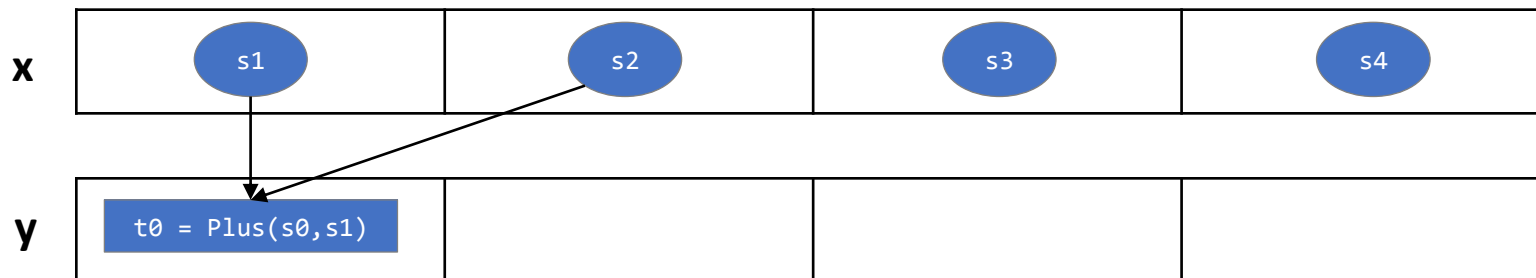
How to add two “Reps”?



Staging Applied to the Running Example

```
def f(x: Array[Rep[Double]], y: Array[Rep[Double]]) = {  
  for (i <- 0 until 2) {  
    y(2*i) = x(i*2) + x(i*2+1)  
    y(2*i+1) = x(i*2) - x(i*2+1)  
  }  
}
```

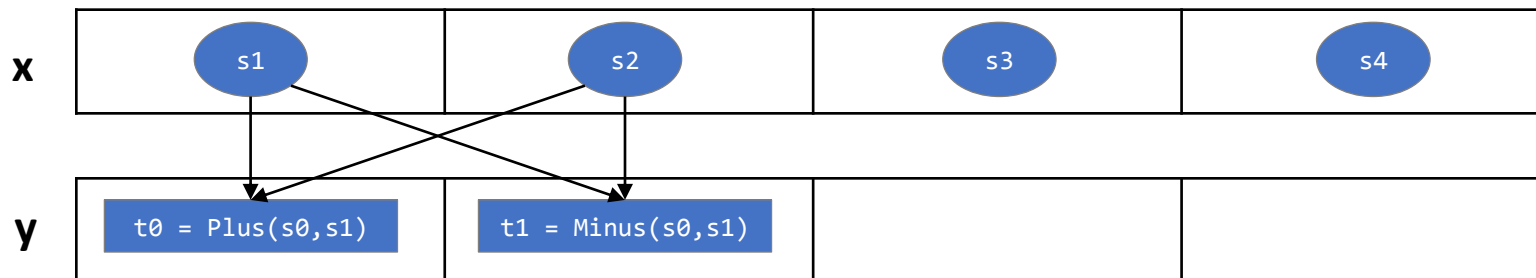
i = 0



Staging Applied to the Running Example

```
def f(x: Array[Rep[Double]], y: Array[Rep[Double]]) = {  
  for (i <- 0 until 2) {  
    y(2*i)   = x(i*2) + x(i*2+1)  
    y(2*i+1) = x(i*2) - x(i*2+1)  
  }  
}
```

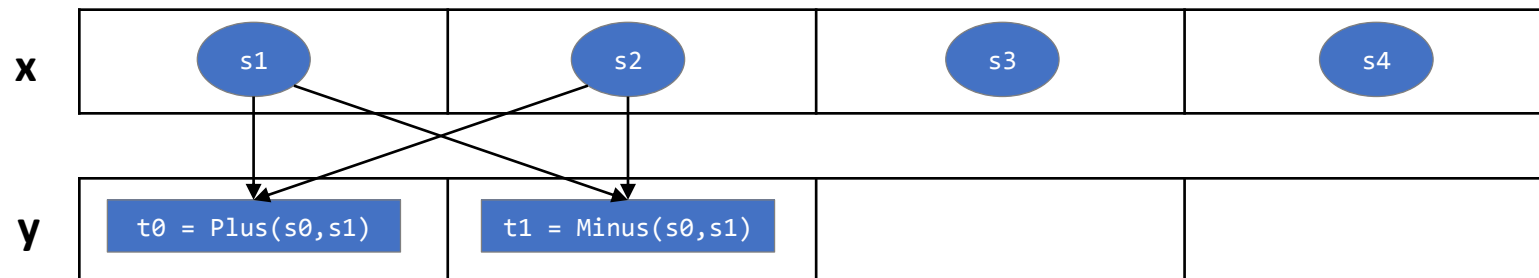
i = 0



Staging Applied to the Running Example

```
def f(x: Array[Rep[Double]], y: Array[Rep[Double]]) = {  
  for (i <- 0 until 2) {  
    y(2*i)   = x(i*2) + x(i*2+1)  
    y(2*i+1) = x(i*2) - x(i*2+1)  
  }  
}
```

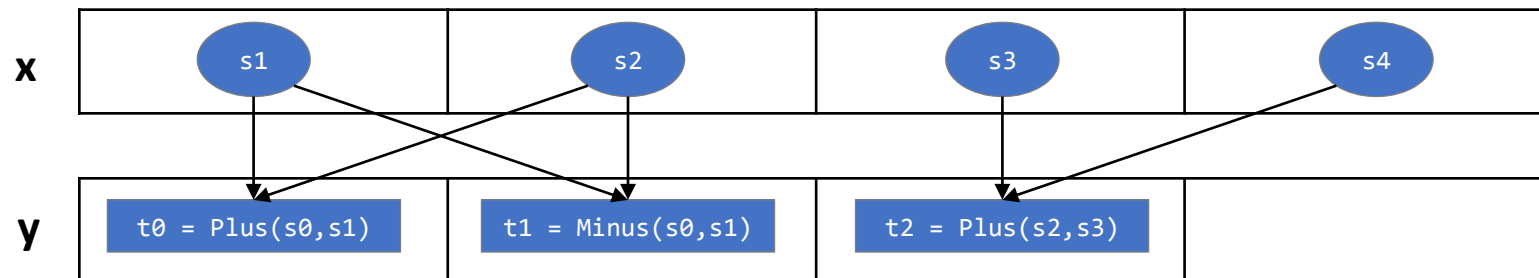
i = 1



Staging Applied to the Running Example

```
def f(x: Array[Rep[Double]], y: Array[Rep[Double]]) = {  
  for (i <- 0 until 2) {  
    y(2*i)   = x(i*2) + x(i*2+1)  
    y(2*i+1) = x(i*2) - x(i*2+1)  
  }  
}
```

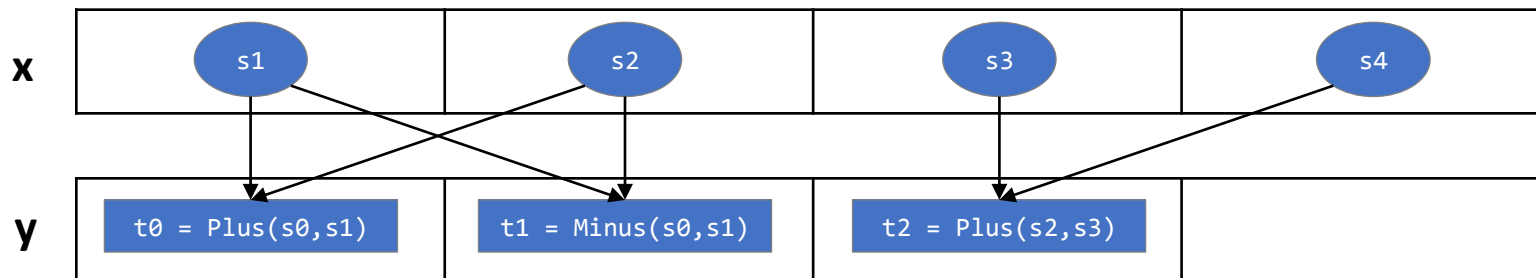
i = 1



Staging Applied to the Running Example

```
def f(x: Array[Rep[Double]], y: Array[Rep[Double]]) = {  
  for (i <- 0 until 2) {  
    y(2*i)   = x(i*2) + x(i*2+1)  
    y(2*i+1) = x(i*2) - x(i*2+1)  
  }  
}
```

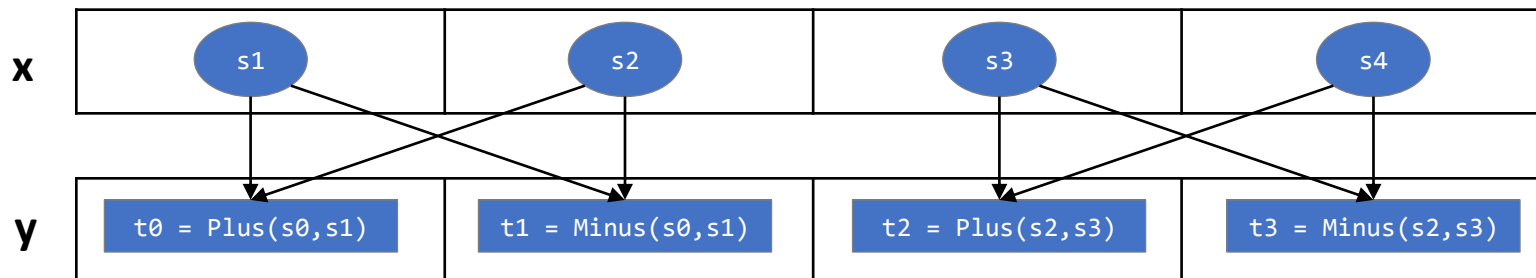
i = 1



Staging Applied to the Running Example

```
def f(x: Array[Rep[Double]], y: Array[Rep[Double]]) = {  
  for (i <- 0 until 2) {  
    y(2*i)   = x(i*2) + x(i*2+1)  
    y(2*i+1) = x(i*2) - x(i*2+1)  
  }  
}
```

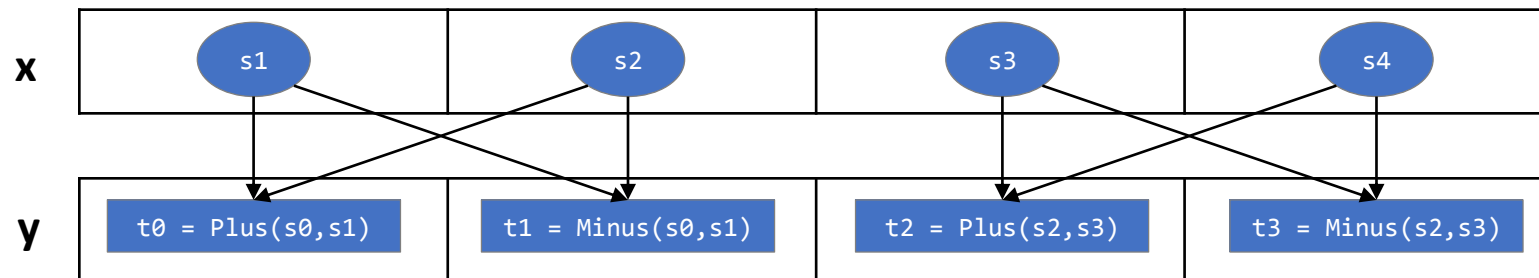
i = 1



Staging Applied to the Running Example

```
def f(x: Array[Rep[Double]], y: Array[Rep[Double]]) = {  
  for (i <- 0 until 2) {  
    y(2*i)   = x(i*2) + x(i*2+1)  
    y(2*i+1) = x(i*2) - x(i*2+1)  
  }  
}
```

i = 1

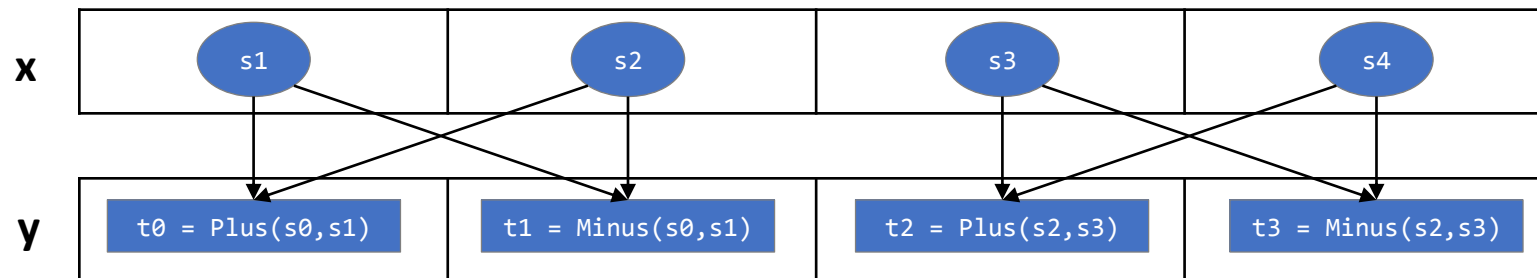


Staging Applied to the Running Example

```
def f(x: Array[Rep[Double]], y: Array[Rep[Double]]) = {  
  for (i <- 0 until 2) {  
    y(2*i) = x(i*2) + x(i*2+1)  
    y(2*i+1) = x(i*2) - x(i*2+1)  
  }  
}
```

i = 1

```
t0 = s0 + s1;  
t1 = s0 - s1;  
t2 = s2 + s3;  
t3 = s2 - s3;
```



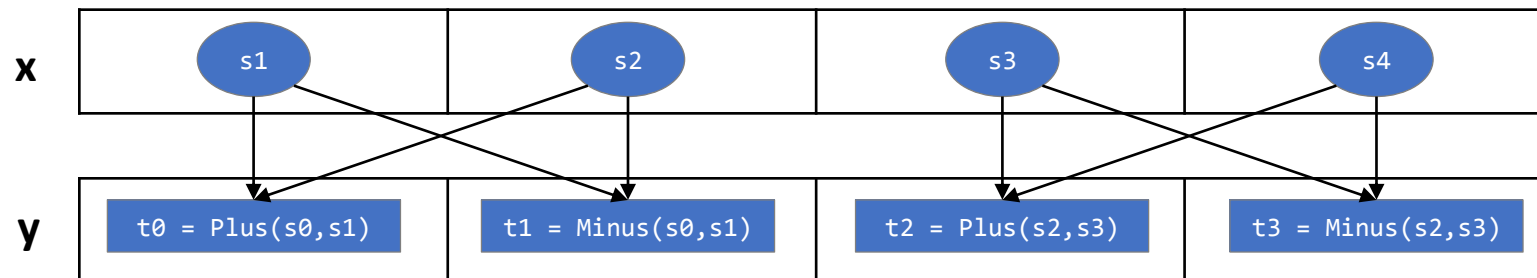
Staging Applied to the Running Example

```
def f(x: Array[Rep[Double]], y: Array[Rep[Double]]) = {  
  for (i <- 0 until 2) {  
    y(2*i)   = x(i*2) + x(i*2+1)  
    y(2*i+1) = x(i*2) - x(i*2+1)  
  }  
}
```

i = 1

Scalars only
Unrolled

```
t0 = s0 + s1;  
t1 = s0 - s1;  
t2 = s2 + s3;  
t3 = s2 - s3;
```



```

trait FFT { this: Arith with Trig =>
  def omega(k: Int, N: Int): Complex = {
    val kth = -2.0 * k * math.Pi / N Complex(cos(kth), sin(kth)) }
  case class Complex(re: Double, im: Double) {
    def +(that: Complex) = Complex(this.re + that.re, this.im + that.im)
    def -(that: Complex) = Complex(this.re - that.re, this.im - that.im)
    def *(that: Complex) = Complex(this.re * that.re - this.im * that.im,
      this.re * that.im + this.im * that.re)
  }
  def splitEvenOdd[T](xs: List[T]): (List[T], List[T]) = (xs: @unchecked)
  match {
    case e :: o :: xt =>
      val (es, os) = splitEvenOdd(xt) ((e :: es), (o :: os))
    case Nil => (Nil, Nil)
  }
  def mergeEvenOdd[T](even: List[T], odd: List[T]): List[T] =
    ((even, odd): @unchecked) match {
      case (Nil, Nil) => Nil
      case ((e :: es), (o :: os)) => e :: (o :: mergeEvenOdd(es, os)) }

  def fft(xs: List[Complex]): List[Complex] = xs match {
    case (x :: Nil) => xs case _ => val N = xs.length
    val (even0, odd0) = splitEvenOdd(xs)
    val (even1, odd1) = (fft(even0), fft(odd0))
    val (even2, odd2) = (even1 zip odd1 zipWithIndex) map {
      case ((x, y), k) => val z = omega(k, N) * y (x + z, x - z) } unzip; even2 ::: odd2 }17

```

```

trait FFT { this: Arith with Trig =>
  def omega(k: Int, N: Int): Complex = {
    val kth = -2.0 * k * math.Pi / N Complex(cos(kth), sin(kth)) }
  case class Complex(re: Rep[Double], im: Rep[Double]) {
    def +(that: Complex) = Complex(this.re + that.re, this.im + that.im)
    def -(that: Complex) = Complex(this.re - that.re, this.im - that.im)
    def *(that: Complex) = Complex(this.re * that.re - this.im * that.im,
      this.re * that.im + this.im * that.re)
  }
  def splitEvenOdd[T](xs: List[T]): (List[T], List[T]) = (xs: @unchecked)
  match {
    case e :: o :: xt =>
      val (es, os) = splitEvenOdd(xt) ((e :: es), (o :: os))
    case Nil => (Nil, Nil)
  }
  def mergeEvenOdd[T](even: List[T], odd: List[T]): List[T] =
    ((even, odd): @unchecked) match {
      case (Nil, Nil) => Nil
      case ((e :: es), (o :: os)) => e :: (o :: mergeEvenOdd(es, os)) }

  def fft(xs: List[Complex]): List[Complex] = xs match {
    case (x :: Nil) => xs case _ => val N = xs.length
    val (even0, odd0) = splitEvenOdd(xs)
    val (even1, odd1) = (fft(even0), fft(odd0))
    val (even2, odd2) = (even1 zip odd1 zipWithIndex) map {
      case ((x, y), k) => val z = omega(k, N) * y (x + z, x - z) } unzip; even2 ::: odd2 }

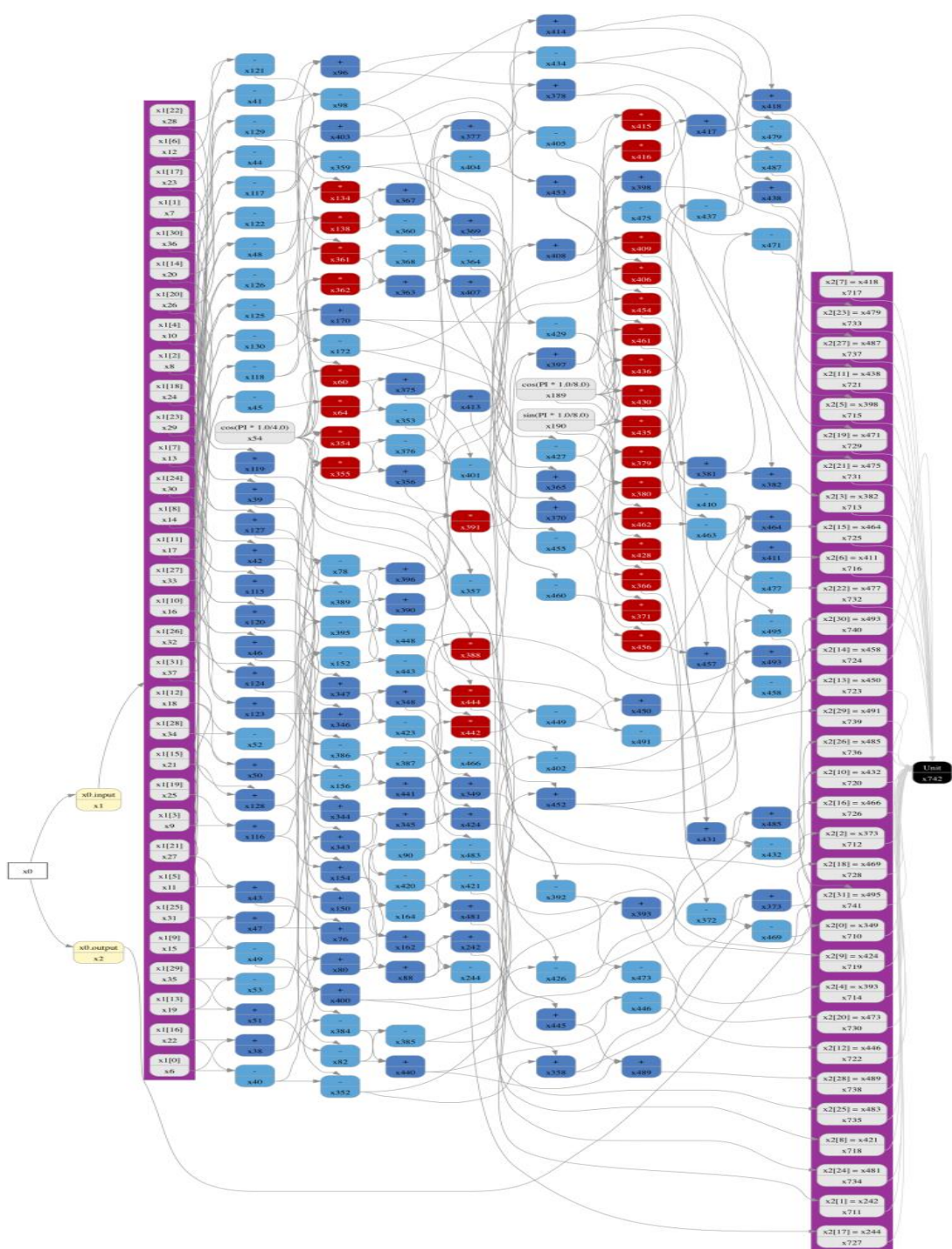
```

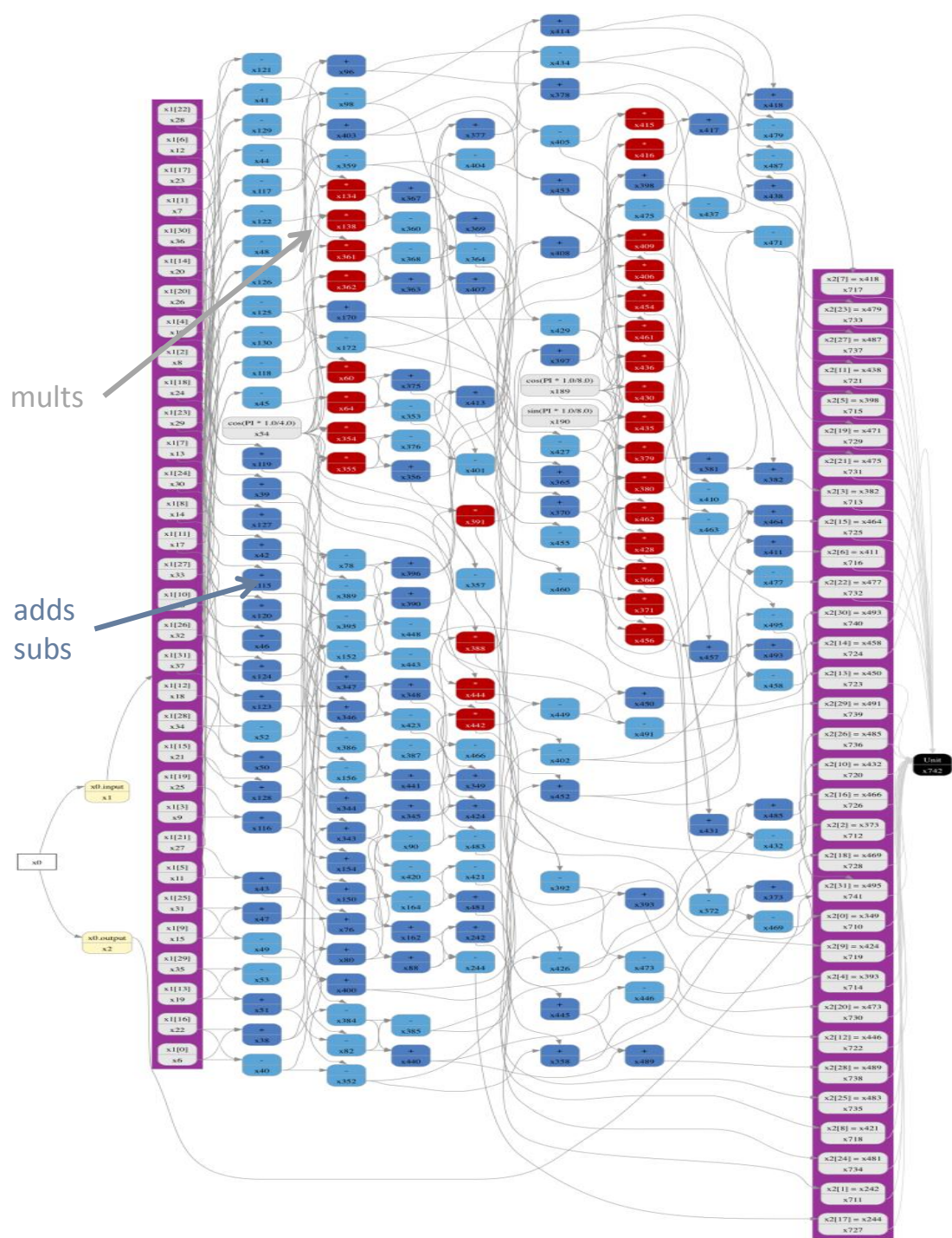
```

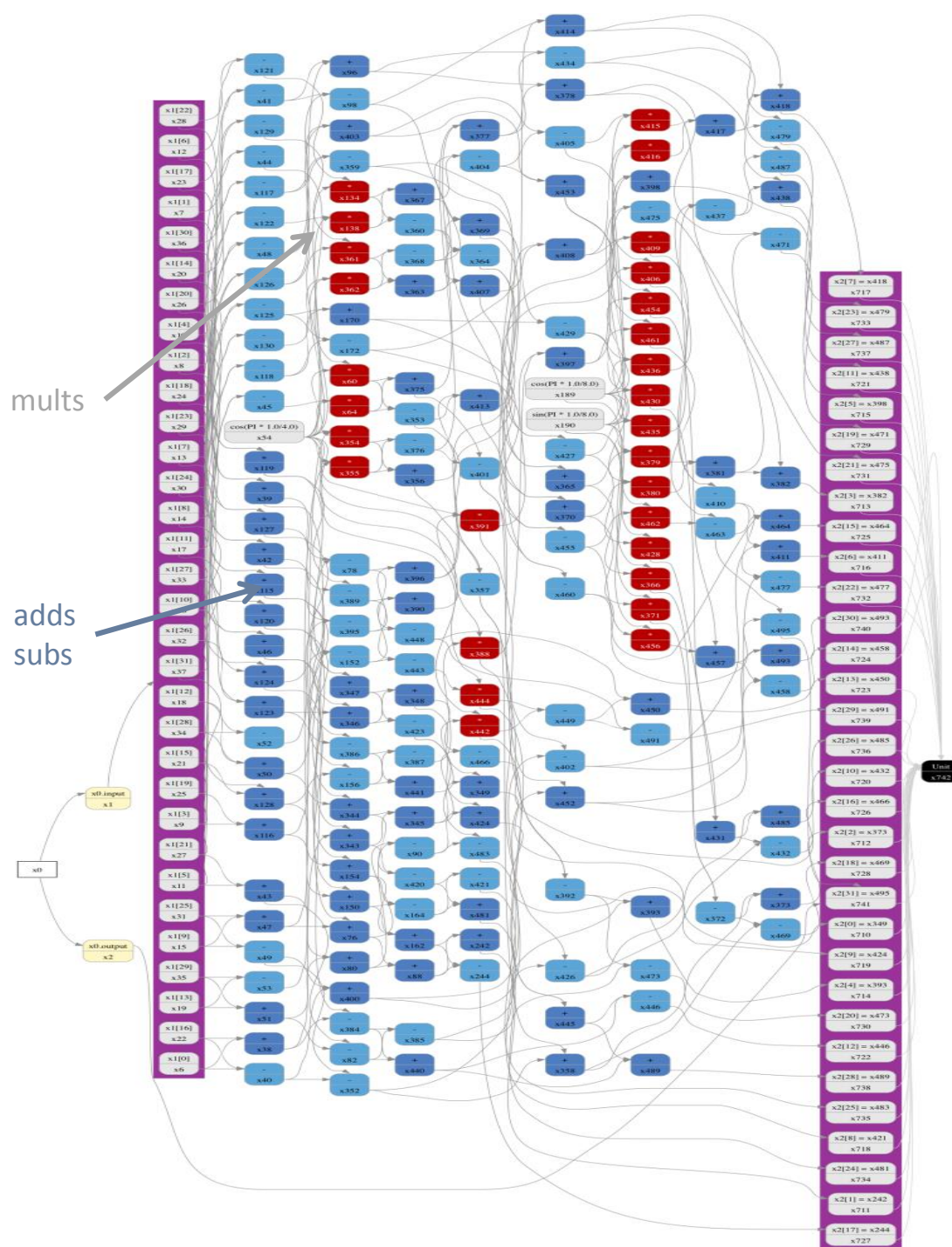
trait FFT { this: Arith with Trig =>
  def omega(k: Int, N: Int): Complex = {
    val kth = -2.0 * k * math.Pi / N Complex(cos(kth), sin(kth)) }
  case class Complex(re: Rep[Double], im: Rep[Double]) {
    def +(that: Complex) = Complex(this.re + that.re, this.im + that.im)
    def -(that: Complex) = Complex(this.re - that.re, this.im - that.im)
    def *(that: Complex) = Complex(this.re * that.re - this.im * that.im,
      this.re * that.im + this.im * that.re)
  }
  def splitEvenOdd[T](xs: List[T]): (List[T], List[T]) = (xs: @unchecked)
  match {
    case e :: o :: xt =>
      val (es, os) = splitEvenOdd(xt) ((e :: es), (o :: os))
    case Nil => (Nil, Nil)
  }
  def mergeEvenOdd[T](even: List[T], odd: List[T]): List[T] =
    ((even, odd): @unchecked) match {
      case (Nil, Nil) => Nil
      case ((e :: es), (o :: os)) => e :: (o :: mergeEvenOdd(es, os)) }

  def fft(xs: List[Complex]): List[Complex] = xs match {
    case (x :: Nil) => xs case _ => val N = xs.length
    val (even0, odd0) = splitEvenOdd(xs)
    val (even1, odd1) = (fft(even0), fft(odd0))
    val (even2, odd2) = (even1 zip odd1 zipWithIndex) map {
      case ((x, y), k) => val z = omega(k, N) * y (x + z, x - z) } unzip; even2 ::: odd2 }19

```







```

typedef struct {
    double* input;
    double* output;
} spiral_t;
const double x708[] = { 1.0, 0.9238795325112867, 0.7071067811865476, 0.3826834323650898, 0.0, 0.3826834323650898, 0.7071067811865476, 0.9238795325112867 };
const double x709[] = { -0.0, 0.3826834323650898, 0.7071067811865476, 0.9238795325112867, 1.0, 0.9238795325112867, 0.7071067811865476, 0.3826834323650898 };
void staged(spiral_t* x0) {
    double* x2 = x0->output;
    double* x1 = x0->input;
    double x6 = x1[0];
    double x22 = x1[16];
    double x38 = x6 + x22;
    double x14 = x1[8];
    double x30 = x1[24];
    double x46 = x14 + x30;
    double x343 = x38 + x46;
    double x10 = x1[4];
    double x26 = x1[20];
    double x42 = x10 + x26;
    double x18 = x1[12];
    double x34 = x1[28];
    double x50 = x18 + x34;
    double x344 = x42 + x50;
    double x345 = x343 + x344;
    double x8 = x1[2];
    double x24 = x1[18];
    double x115 = x8 + x24;
    double x16 = x1[10];
    double x32 = x1[26];
    double x123 = x16 + x32;
    double x346 = x115 + x123;
    double x12 = x1[6];
    double x28 = x1[22];
    double x119 = x12 + x28;
    double x20 = x1[14];
    double x36 = x1[30];
    double x127 = x20 + x36;
    double x347 = x119 + x127;
    double x348 = x346 + x347;
    double x349 = x345 + x348;
    x2[0] = x349;
    double x7 = x1[1];
    double x23 = x1[17];
    double x39 = x7 + x23;
    double x15 = x1[9];
    double x31 = x1[25];
    double x47 = x15 + x31;
    double x76 = x39 + x47;
    double x11 = x1[5];
    double x27 = x1[21];
    double x43 = x11 + x27;
    double x19 = x1[13];
    double x35 = x1[29];
    double x51 = x19 + x35;

```

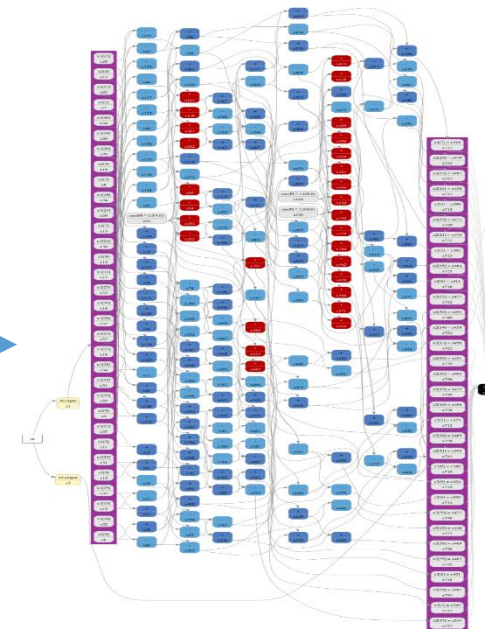
Selective Staging

```
trait FFT {...  
  case class Complex(re: Double, im: Double)  
  def fft(...)  
  ...
```



Scala FFT Function

```
trait FFT {...  
  case class Complex(re: Rep[Double], im: Rep[Double])  
  def fft(...)  
  ...
```

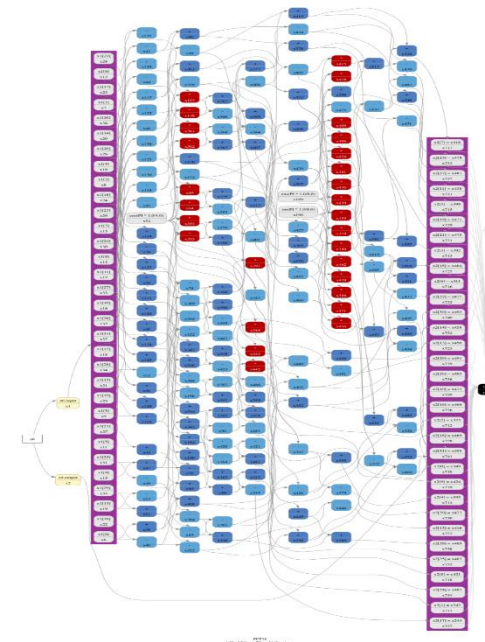


Selective Staging

```
trait FFT {...  
  case class Complex(re: Double, im: Double)  
  def fft(...)  
  ...  
}
```

```
trait FFT {...  
  case class Complex(re: Rep[Double], im: Rep[Double])  
  def fft(...)  
  ...  
}
```

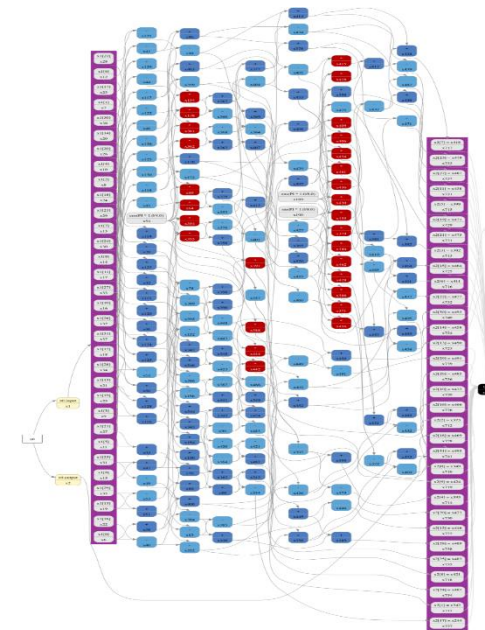
Scala FFT Function



Selective Staging

```
trait FFT {...  
  case class Complex(re: Double, im: Double)  
  def fft(...)  
  ...  
  
trait FFT[T] {...  
  case class Complex(re: T, im: T)  
  def fft(...)  
  ...  
  
trait FFT {...  
  case class Complex(re: Rep[Double], im: Rep[Double])  
  def fft(...)  
  ...
```

Scala FFT Function



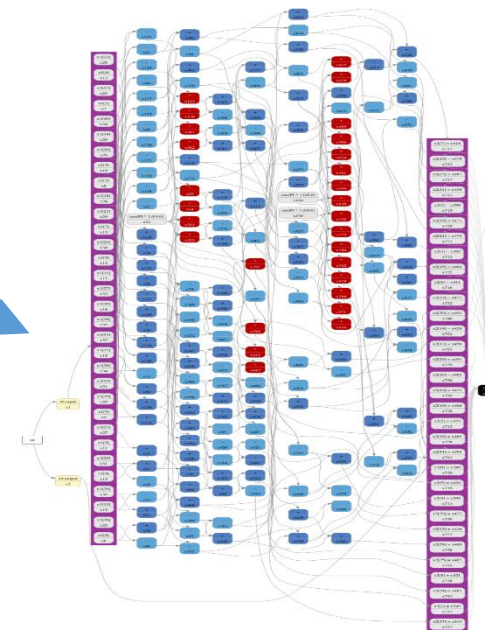
Selective Staging

```
trait FFT {...  
  case class Complex(re: Double, im: Double)  
  def fft(...)  
  ...
```

```
trait FFT[T] {...  
  case class Complex(re: T, im: T)  
  def fft(...)  
  ...
```

```
trait FFT {...  
  case class Complex(re: Rep[Double], im: Rep[Double])  
  def fft(...)  
  ...
```

Scala FFT Function



Option 3: Staging in Scala

Σ -SPL

Code Style

Data Layout

AST

Code

$$T_m^n \left(\sum_{j=0}^2 S(h_j) \mathbf{DFT}_2 G(h_j) \right)$$

Scala Staging Program

Looped Code

Unrolled Code

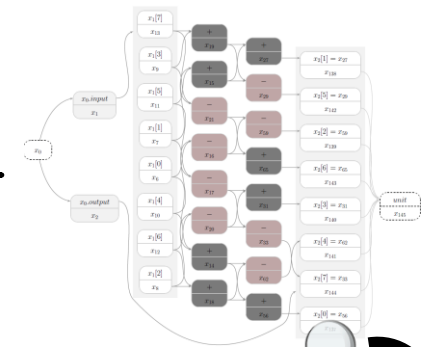
Code with
Precomputation

C99 Complex
Interleaved Complex
Split Complex

C99 Complex
Interleaved Complex
Split Complex

C99 Complex
Interleaved Complex
Split Complex

Staging decisions

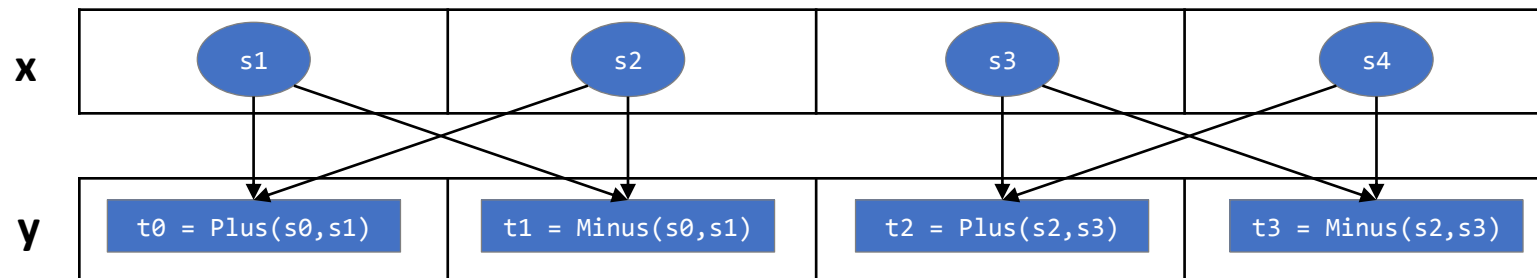


Code Style: Unrolled and Scalarized

```
def f(x: Array[Rep[Double]], y: Array[Rep[Double]]) = {  
  for (i <- 0 until 2) {  
    y(2*i) = x(i*2) + x(i*2+1)  
    y(2*i+1) = x(i*2) - x(i*2+1)  
  }  
}
```

Scalars only
Unrolled

```
t0 = s0 + s1;  
t1 = s0 - s1;  
t2 = s2 + s3;  
t2 = s2 - s3;
```



Code Style: Arrays

```
def f(x: Rep[Array[Double]], y: Rep[Array[Double]]) = {  
  for (i <- 0 until 2) {  
    y(2*i) = x(i*2) + x(i*2+1)  
    y(2*i+1) = x(i*2) - x(i*2+1)  
  }  
}
```

Code Style: Arrays

```
def f(x: Rep[Array[Double]], y: Rep[Array[Double]]) = {  
  for (i <- 0 until 2) {  
    i = 0  
    y(2*i) = x(i*2) + x(i*2+1)  
  }  
}
```

Code Style: Arrays

```
def f(x: Rep[Array[Double]], y: Rep[Array[Double]]) = {  
  for (i <- 0 until 2) {  
    y(2*i) = x(i*2) + x(i*2+1)  
  }  
}
```

i = 0

$\underbrace{x(i*2)}_{\text{Rep[...]}.apply(i: Int): T} + x(i*2+1)$

Code Style: Arrays

```
def f(x: Rep[Array[Double]], y: Rep[Array[Double]]) = {  
  for (i <- 0 until 2) {  
    y(2*i) = x(i*2) + x(i*2+1)  
  }  
}
```

i = 0

Rep[...].apply(i: Int): T

How to apply() on a Rep?

Code Style: Arrays

```
def f(x: Rep[Array[Double]], y: Rep[Array[Double]]) = {  
  for (i <- 0 until 2) {  
    i = 0  
    y(2*i) = x(i*2) + x(i*2+1)
```

Rep[...].apply(i: Int): T

How to apply() on a Rep?

Implementation in **L**ightweight **M**odular **S**taging Library

Code Style: Arrays

```
def f(x: Rep[Array[Double]], y: Rep[Array[Double]]) = {  
  for (i <- 0 until 2) {  
    y(2*i) = x(i*2) + x(i*2+1)  
  }  
}
```

i = 0

Rep[...].apply(i: Int): T

How to apply() on a Rep?

Code Style: Arrays

```
def f(x: Rep[Array[Double]], y: Rep[Array[Double]]) = {  
  for (i <- 0 until 2) {  
    y(2*i) = x(i*2) + x(i*2+1)  
  }  
}
```

i = 0

Rep[...].apply(i: Int): T

How to apply() on a Rep?

x

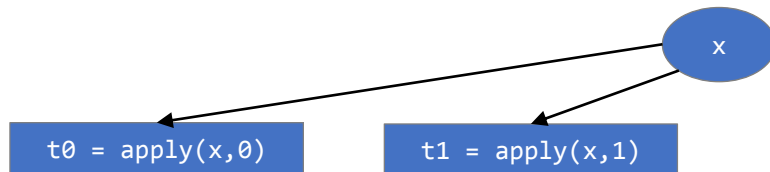
Code Style: Arrays

```
def f(x: Rep[Array[Double]], y: Rep[Array[Double]]) = {  
  for (i <- 0 until 2) {  
    y(2*i) = x(i*2) + x(i*2+1)  
  }  
}
```

i = 0

Rep[...].apply(i: Int): T

How to apply() on a Rep?



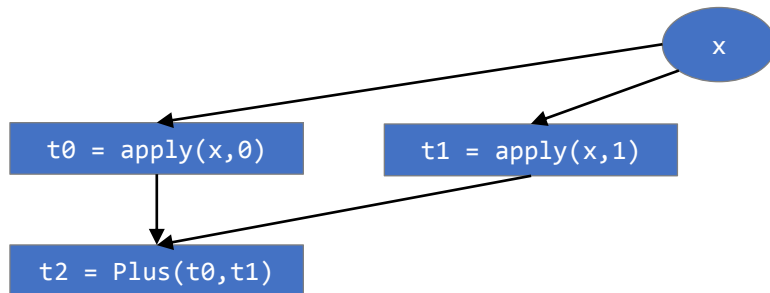
Code Style: Arrays

```
def f(x: Rep[Array[Double]], y: Rep[Array[Double]]) = {  
  for (i <- 0 until 2) {  
    y(2*i) = x(i*2) + x(i*2+1)  
  }  
}
```

i = 0

Rep[...].apply(i: Int): T

How to apply() on a Rep?

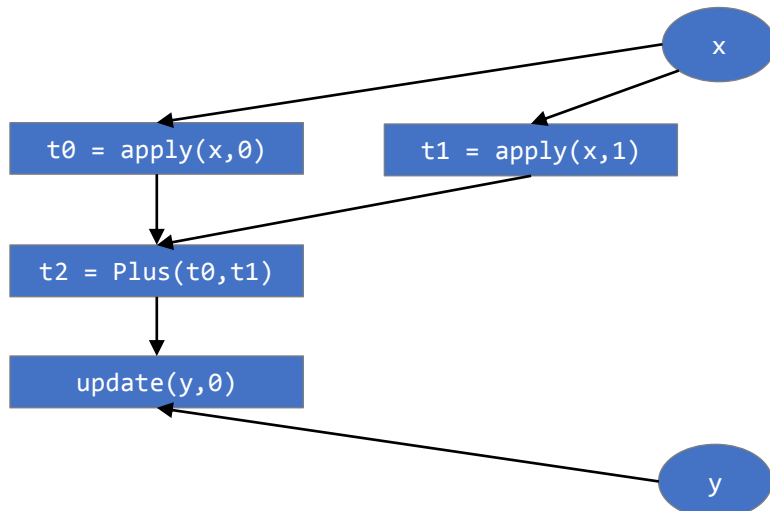


Code Style: Arrays

```
def f(x: Rep[Array[Double]], y: Rep[Array[Double]]) = {  
  for (i <- 0 until 2) {  
    i = 0  
    y(2*i) = x(i*2) + x(i*2+1)
```

Rep[...].apply(i: Int): T

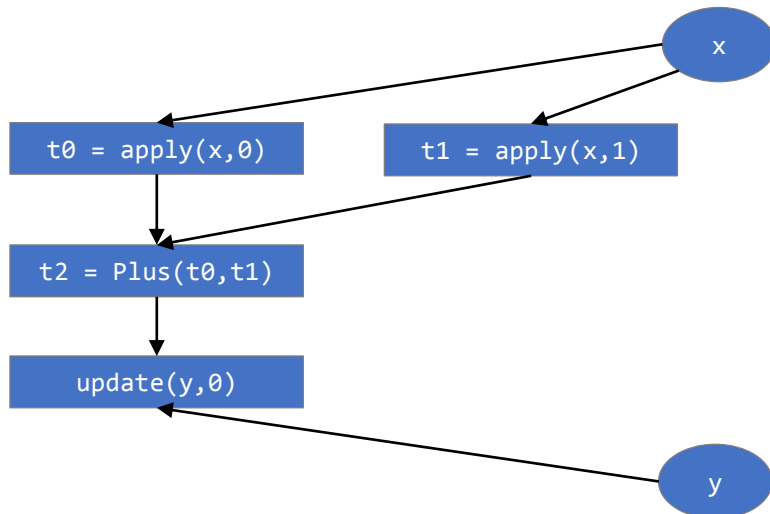
How to apply() on a Rep?



Code Style: Arrays

```
def f(x: Rep[Array[Double]], y: Rep[Array[Double]]) = {  
  for (i <- 0 until 2) {  
    y(2*i) = x(i*2) + x(i*2+1)  
    y(2*i+1) = x(i*2) - x(i*2+1)  
  }  
}
```

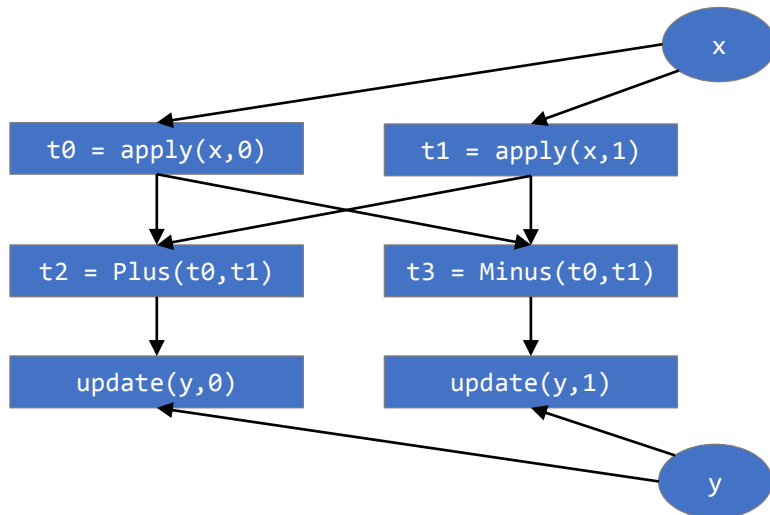
i = 0



Code Style: Arrays

```
def f(x: Rep[Array[Double]], y: Rep[Array[Double]]) = {  
  for (i <- 0 until 2) {  
    y(2*i) = x(i*2) + x(i*2+1)  
    y(2*i+1) = x(i*2) - x(i*2+1)  
  }  
}
```

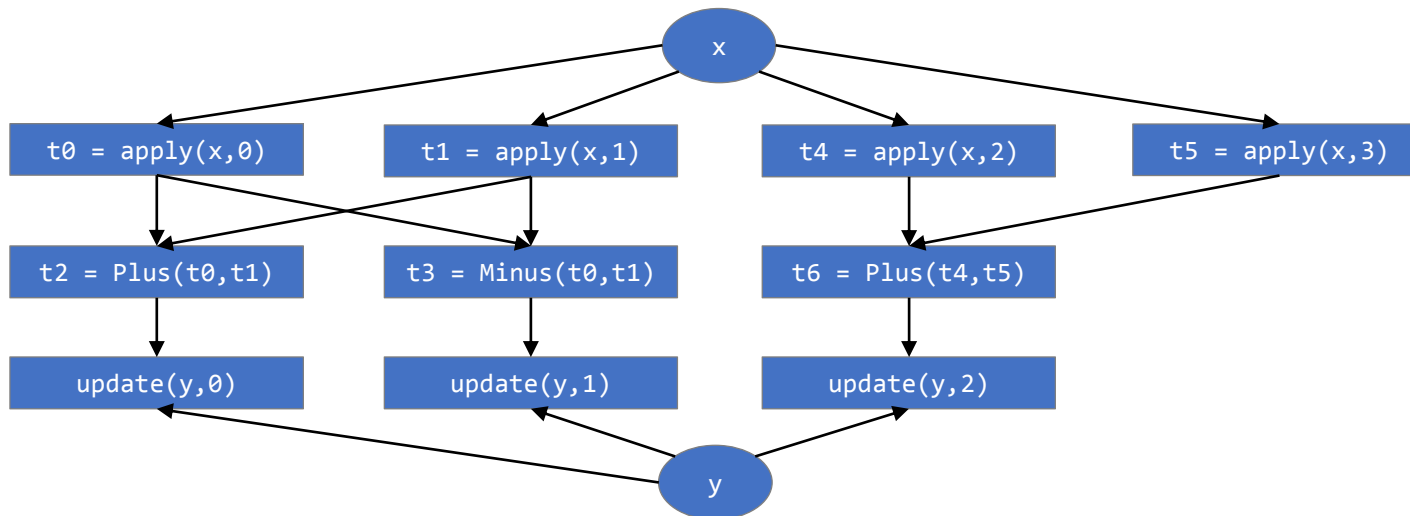
i = 0



Code Style: Arrays

```
def f(x: Rep[Array[Double]], y: Rep[Array[Double]]) = {  
  for (i <- 0 until 2) {  
    y(2*i) = x(i*2) + x(i*2+1)  
    y(2*i+1) = x(i*2) - x(i*2+1)  
  }  
}
```

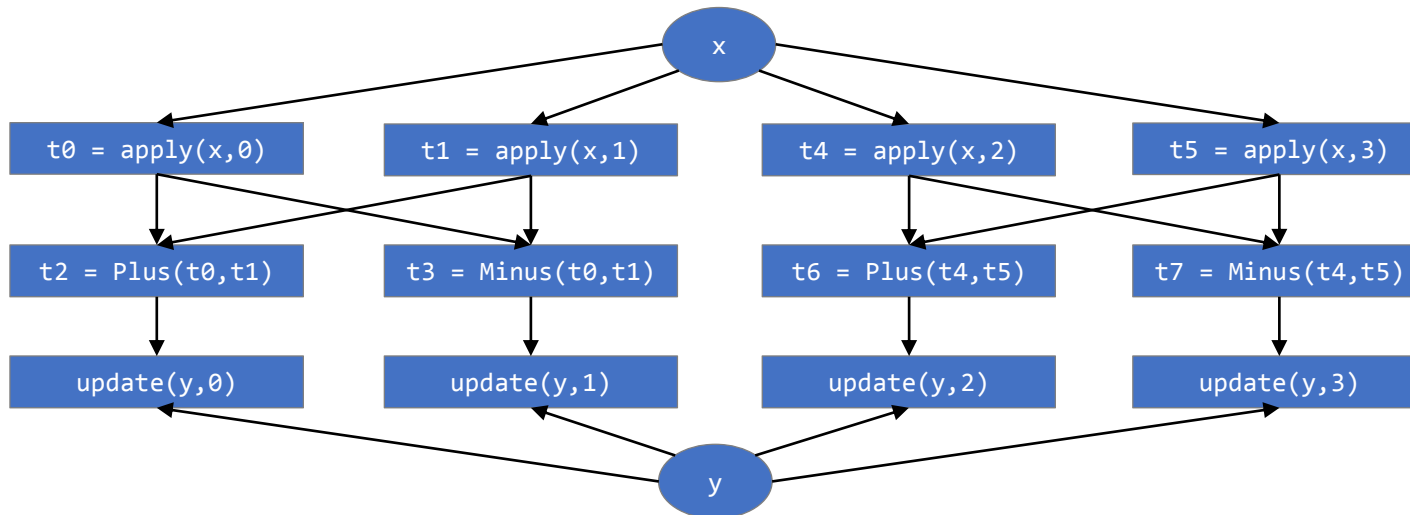
i = 1



Code Style: Arrays

```
def f(x: Rep[Array[Double]], y: Rep[Array[Double]]) = {  
  for (i <- 0 until 2) {  
    y(2*i) = x(i*2) + x(i*2+1)  
    y(2*i+1) = x(i*2) - x(i*2+1)  
  }  
}
```

i = 1

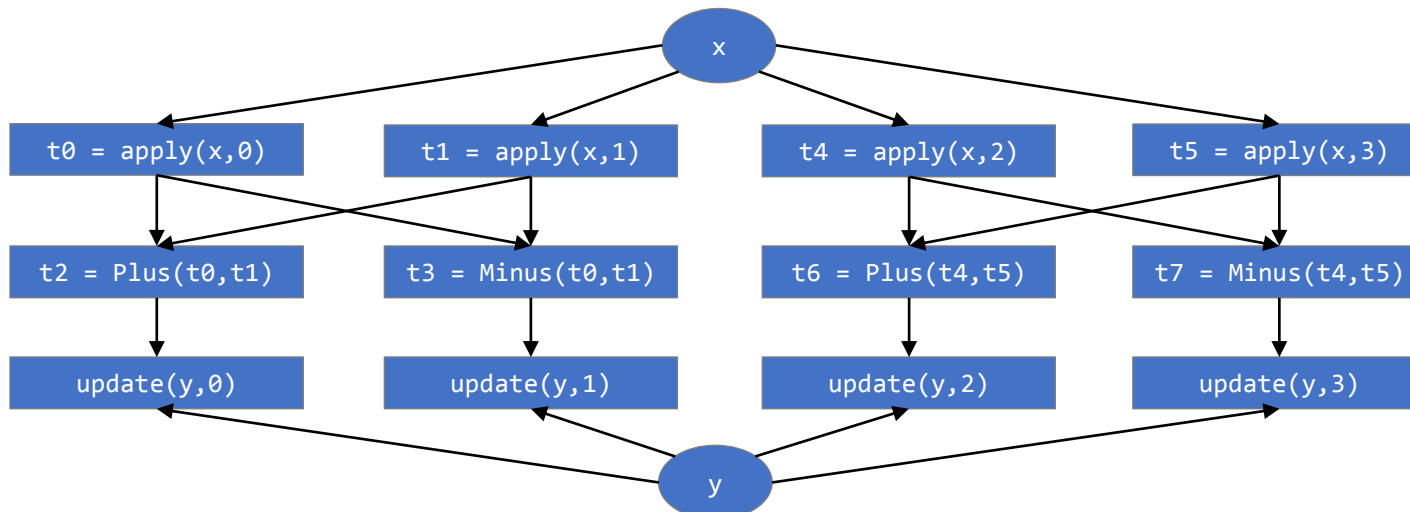


Code Style: Arrays

```
def f(x: Rep[Array[Double]], y: Rep[Array[Double]]) = {  
  for (i <- 0 until 2) {  
    y(2*i) = x(i*2) + x(i*2+1)  
    y(2*i+1) = x(i*2) - x(i*2+1)  
  }  
}
```

i = 1

```
t0 = x[0];  
t1 = x[1];  
t2 = t0 + t1;  
y[0] = t2;  
t3 = t0 - t1;  
y[1] = t3;  
t4 = x[0];  
t5 = x[1];  
t6 = t4 + x5;  
y[0] = t6;  
t7 = t4 - x5;  
y[3] = t7;
```



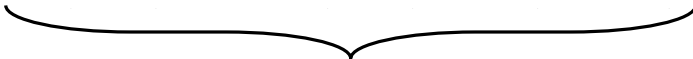
Code Style: Loops & Arrays

```
def f(x: Rep[Array[Double]], y: Rep[Array[Double]]) = {  
  for (i <- 0 until 2: Rep[Range]) {  
    y(2*i)    = x(i*2) + x(i*2+1)  
    y(2*i+1) = x(i*2) - x(i*2+1)  
  }  
}
```


Code Style: Loops & Arrays

```
def f(x: Rep[Array[Double]], y: Rep[Array[Double]]) = {  
  for (i <- 0 until 2: Rep[Range]) {
```

Code Style: Loops & Arrays

```
def f(x: Rep[Array[Double]], y: Rep[Array[Double]]) = {  
  for (i <- 0 until 2: Rep[Range]) {  
    
```

How to apply on a Rep?

Code Style: Loops & Arrays

```
def f(x: Rep[Array[Double]], y: Rep[Array[Double]]) = {  
  for (i <- 0 until 2: Rep[Range]) {
```

How to apply on a Rep?

```
Loop(i, 0, 2)
```

Code Style: Loops & Arrays

```
def f(x: Rep[Array[Double]], y: Rep[Array[Double]]) = {  
  for (i <- 0 until 2: Rep[Range]) {
```

How to apply on a Rep?

Loop(i, 0, 2)

i

Code Style: Loops & Arrays

```
def f(x: Rep[Array[Double]], y: Rep[Array[Double]]) = {  
  for (i <- 0 until 2: Rep[Range]) {  
    y(2*i)    = x(i*2) + x(i*2+1)  
    y(2*i+1) = x(i*2) - x(i*2+1)  
  }  
}
```

Loop(i, 0, 2)

i

Code Style: Loops & Arrays

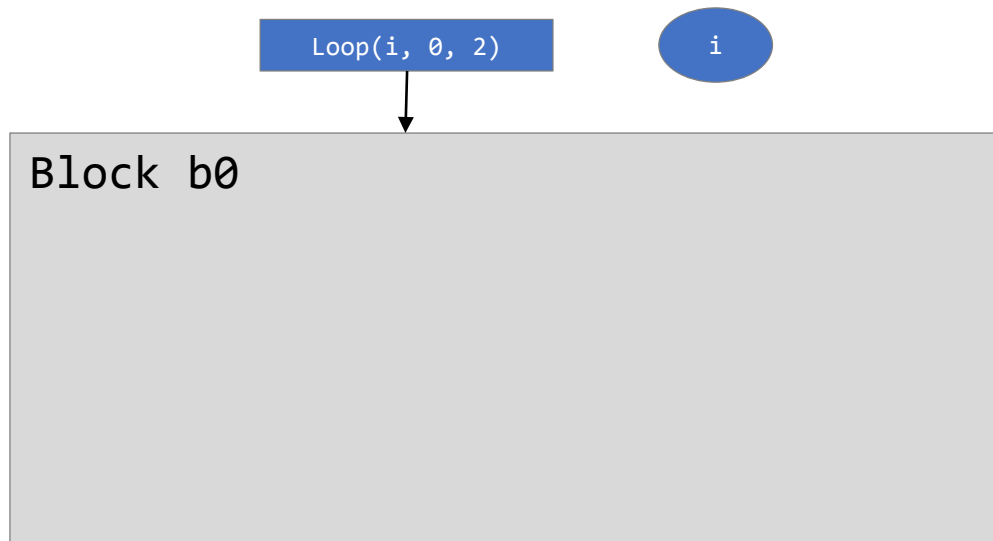
```
def f(x: Rep[Array[Double]], y: Rep[Array[Double]]) = {  
  for (i <- 0 until 2: Rep[Range]) {  
    y(2*i)    = x(i*2) + x(i*2+1)    } Loop  
    y(2*i+1) = x(i*2) - x(i*2+1)    } Body  
  }  
}
```

Loop(i, 0, 2)

i

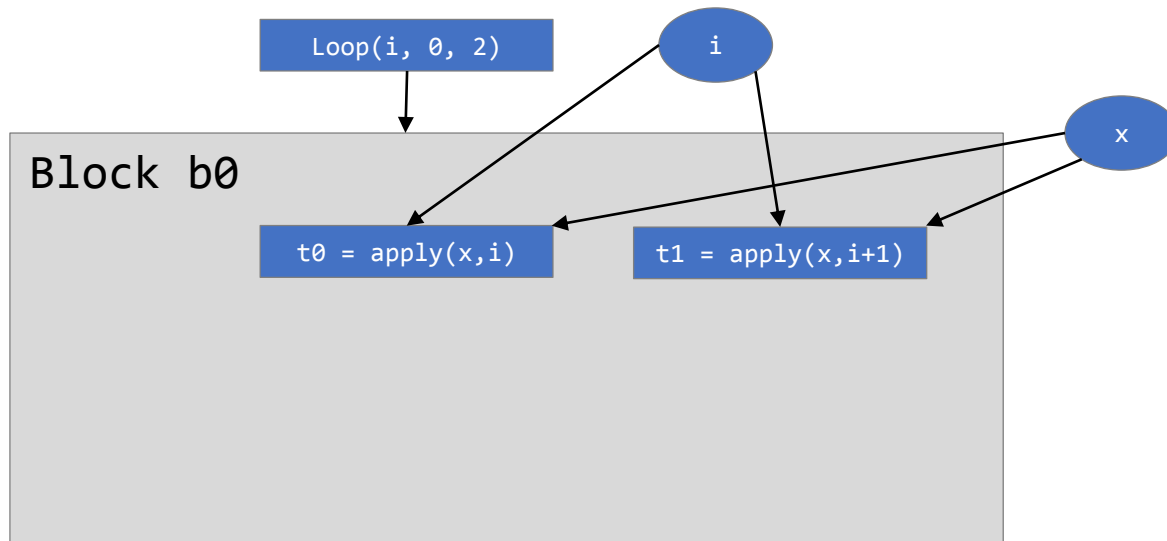
Code Style: Loops & Arrays

```
def f(x: Rep[Array[Double]], y: Rep[Array[Double]]) = {  
  for (i <- 0 until 2: Rep[Range]) {  
    y(2*i)   = x(i*2) + x(i*2+1)   } Loop  
    y(2*i+1) = x(i*2) - x(i*2+1)   } Body  
  }  
}
```



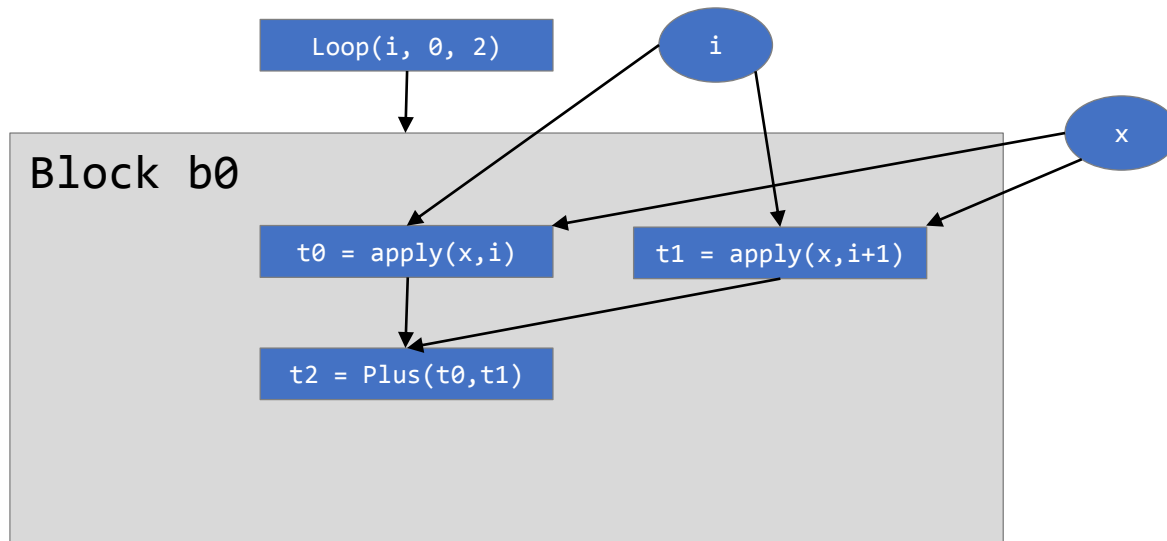
Code Style: Loops & Arrays

```
def f(x: Rep[Array[Double]], y: Rep[Array[Double]]) = {  
  for (i <- 0 until 2: Rep[Range]) {  
    y(2*i)   = x(i*2) + x(i*2+1)   } Loop  
    y(2*i+1) = x(i*2) - x(i*2+1)   } Body  
  }  
}
```



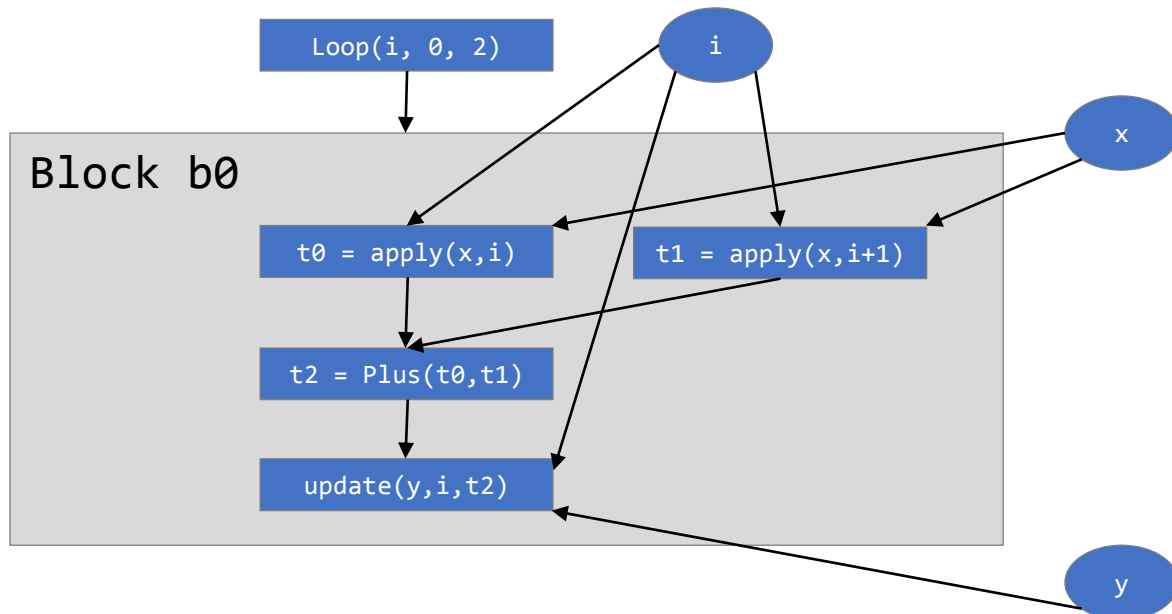
Code Style: Loops & Arrays

```
def f(x: Rep[Array[Double]], y: Rep[Array[Double]]) = {  
  for (i <- 0 until 2: Rep[Range]) {  
    y(2*i)   = x(i*2) + x(i*2+1)   } Loop  
    y(2*i+1) = x(i*2) - x(i*2+1)   } Body  
  }  
}
```



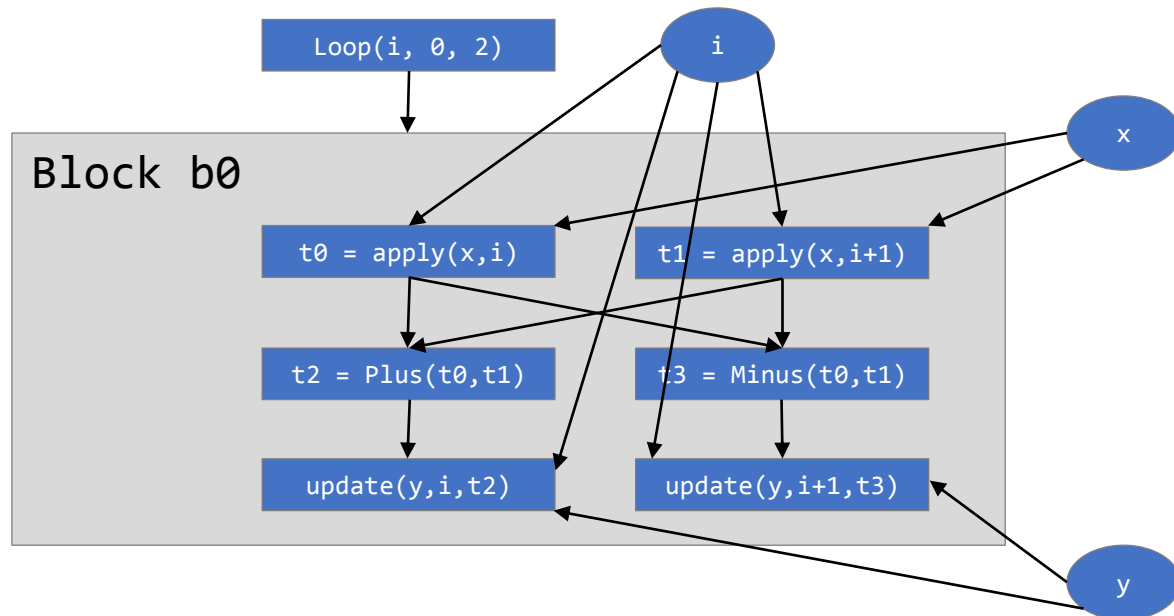
Code Style: Loops & Arrays

```
def f(x: Rep[Array[Double]], y: Rep[Array[Double]]) = {  
  for (i <- 0 until 2: Rep[Range]) {  
    y(2*i) = x(i*2) + x(i*2+1) } Loop  
    y(2*i+1) = x(i*2) - x(i*2+1) } Body  
  }  
}
```



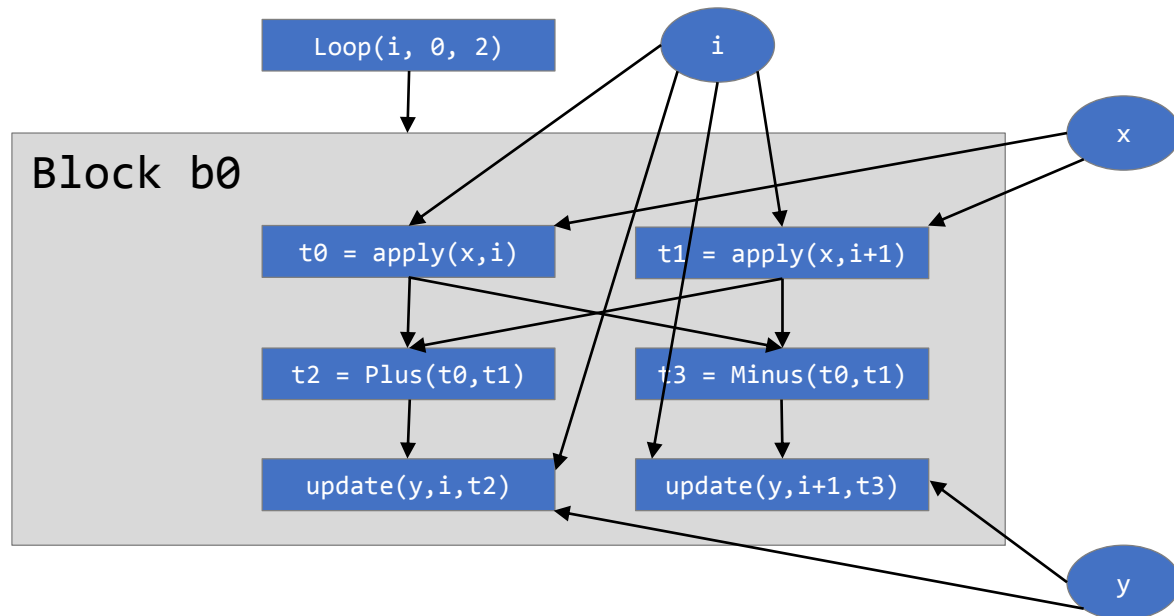
Code Style: Loops & Arrays

```
def f(x: Rep[Array[Double]], y: Rep[Array[Double]]) = {  
  for (i <- 0 until 2: Rep[Range]) {  
    y(2*i)   = x(i*2) + x(i*2+1)   } Loop  
    y(2*i+1) = x(i*2) - x(i*2+1)   } Body  
  }  
}
```



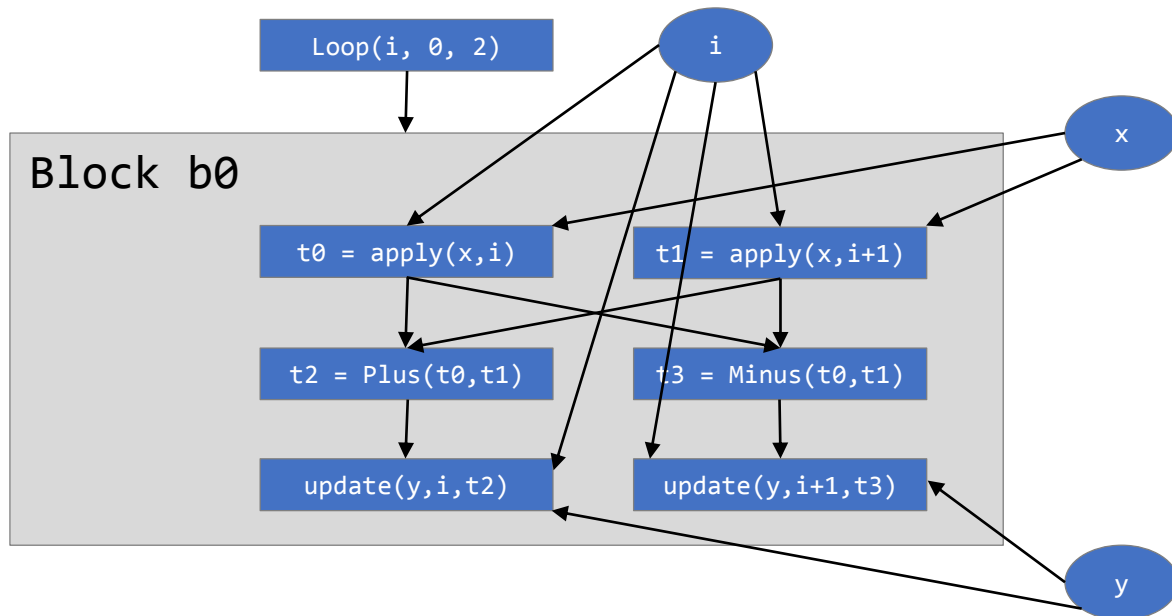
Code Style: Loops & Arrays

```
def f(x: Rep[Array[Double]], y: Rep[Array[Double]]) = {  
  for (i <- 0 until 2: Rep[Range]) {  
    y(2*i) = x(i*2) + x(i*2+1) } Loop  
    y(2*i+1) = x(i*2) - x(i*2+1) } Body  
  }  
}
```



Code Style: Loops & Arrays

```
def f(x: Rep[Array[Double]], y: Rep[Array[Double]]) = {  
  for (i <- 0 until 2: Rep[Range]) {  
    y(2*i) = x(i*2) + x(i*2+1) } Loop  
    y(2*i+1) = x(i*2) - x(i*2+1) } Body  
  }  
}
```



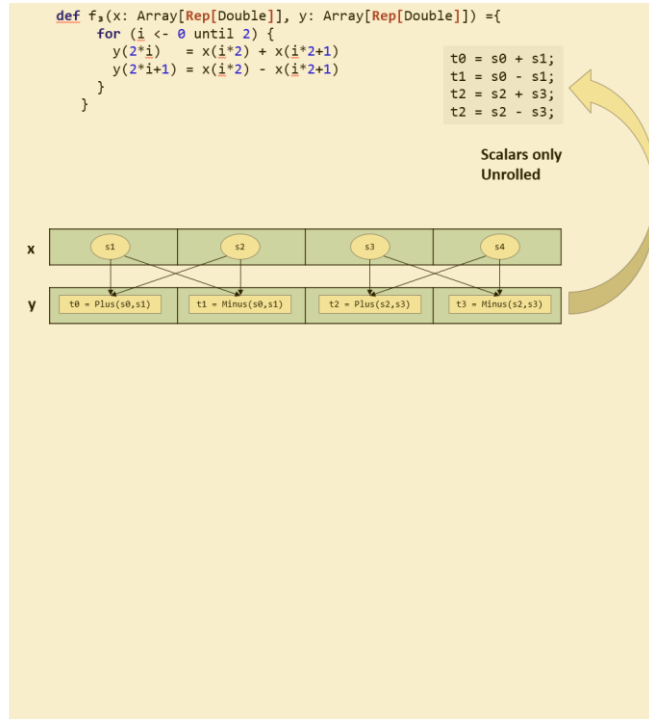
```
for (int i=0;i < 2;i++)  
{  
  t0 = x[i];  
  t1 = x[i+1];  
  t2 = t0 + t1;  
  y[i] = t2;  
  t3 = t0 - t1;  
  y[i+1] = t3;  
}
```



Abstraction over Code Types

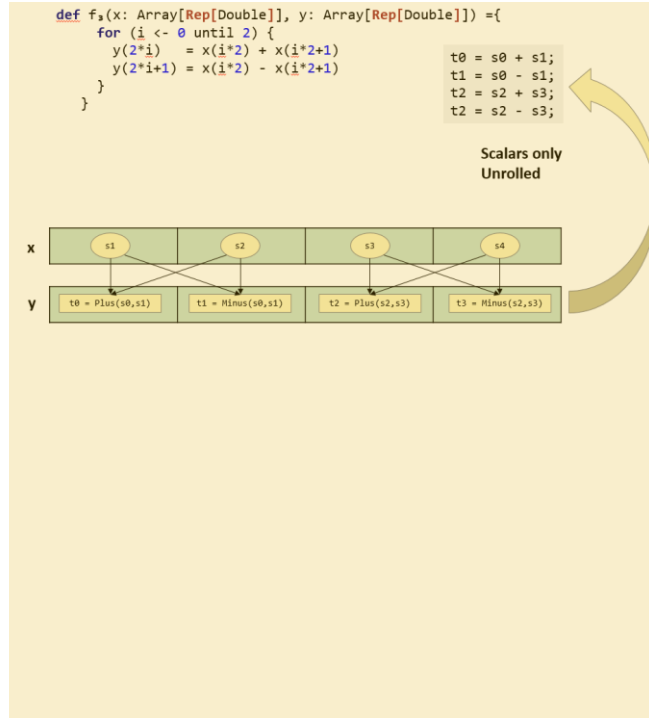
Abstraction over Code Types

Unrolled, scalarized

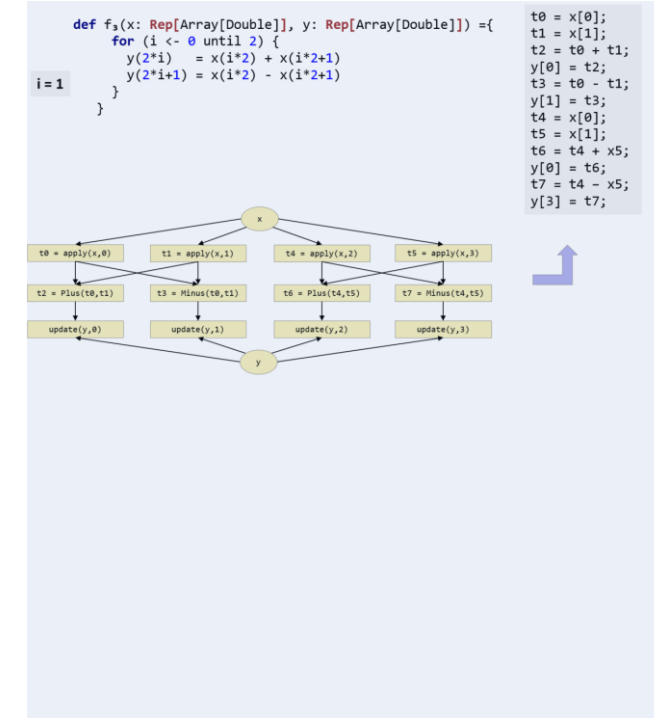


Abstraction over Code Types

Unrolled, scalarized

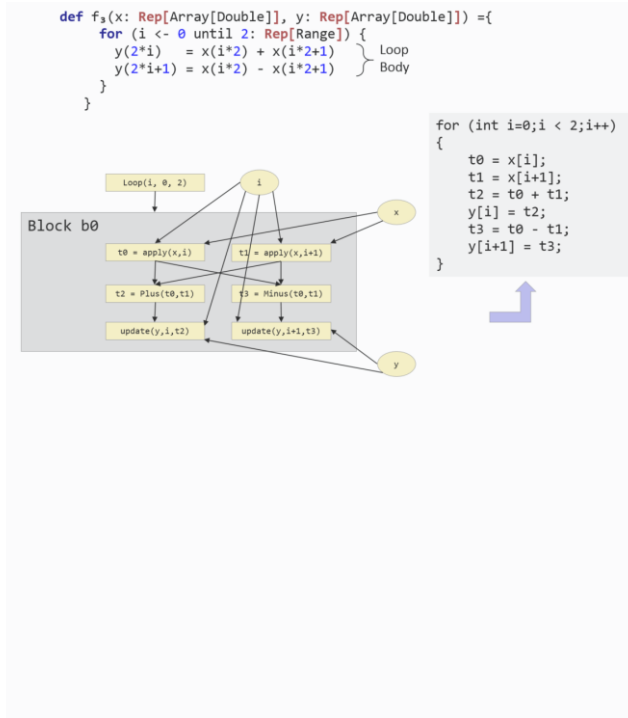


Unrolled, Arrays

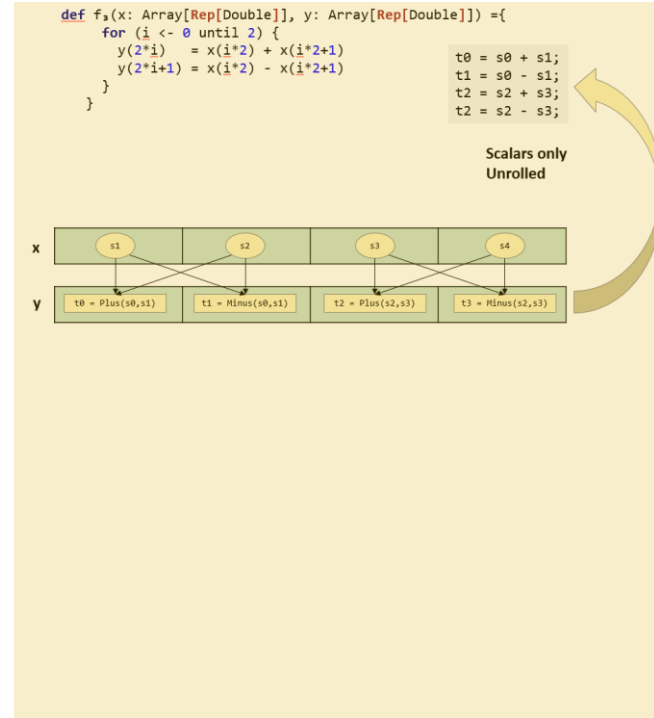


Abstraction over Code Types

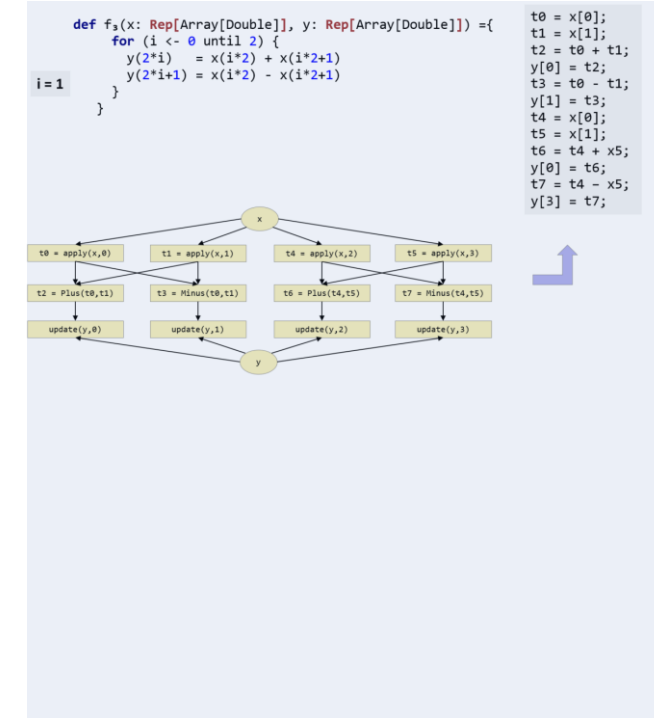
Looped, Arrays



Unrolled, scalarized

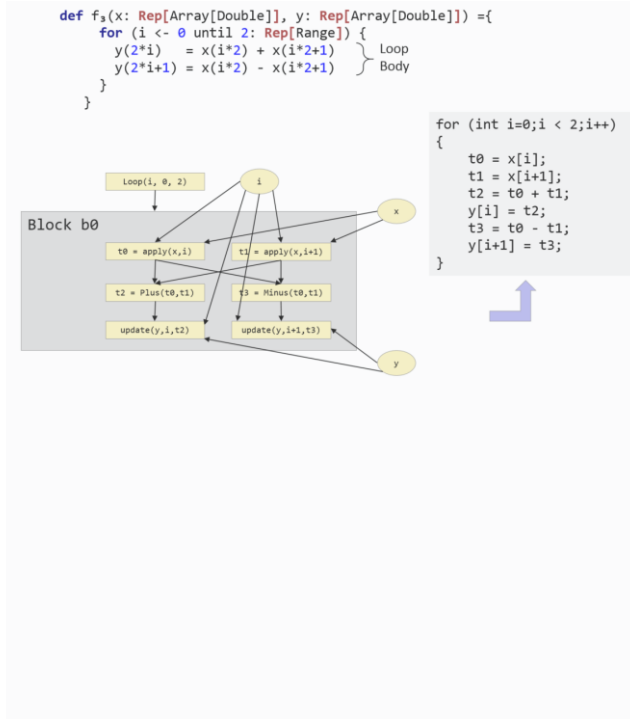


Unrolled, Arrays

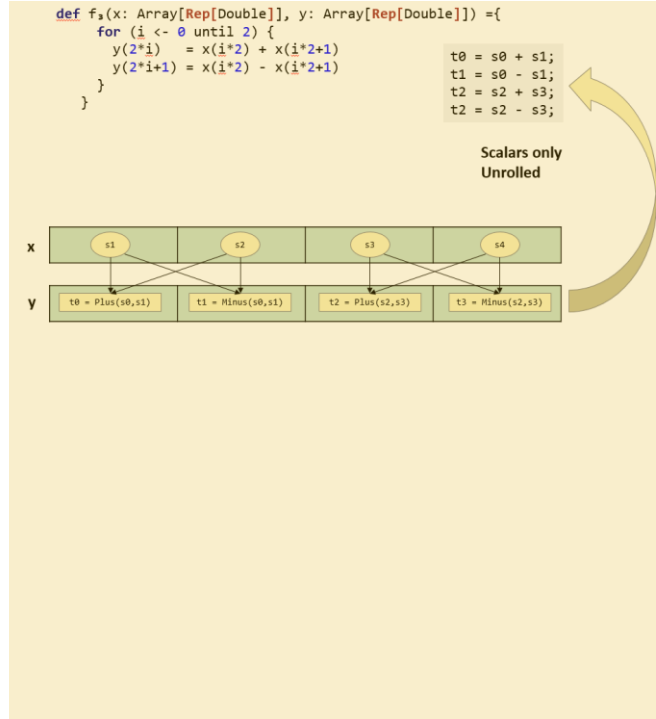


Abstraction over Code Types

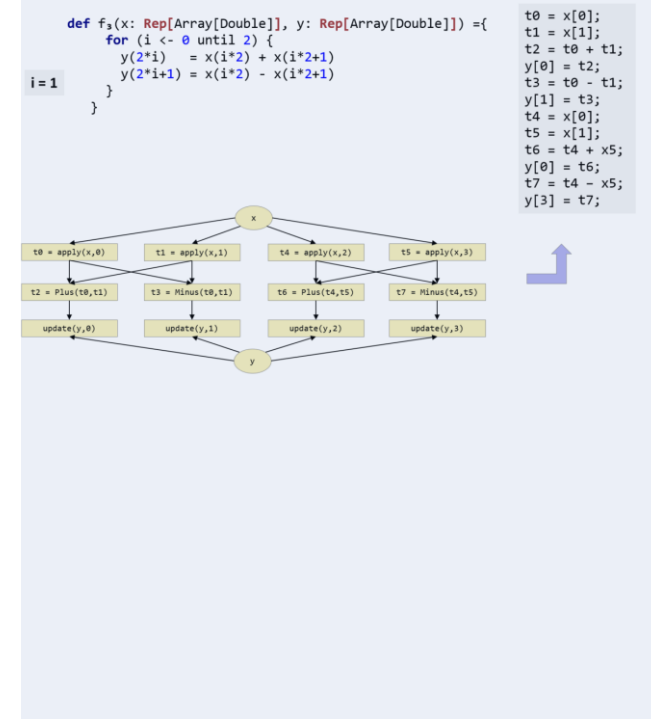
Looped, Arrays



Unrolled, scalarized



Unrolled, Arrays



```
def f[L[_],A[_],T](looptype: L, x: A[Array[T]], y: A[Array[T]]) = {
  for (i <- 0 until 2: L[Range]) {
    y(2*i) = x(i*2) + x(i*2+1)
    y(2*i+1) = x(i*2) - x(i*2+1)
  }
}
```

Abstraction over Code Types

Looped, Arrays

```
def f3(x: Rep[Array[Double]], y: Rep[Array[Double]]) = {
  for (i <- 0 until 2: Rep[Range]) {
    y(2*i) = x(i*2) + x(i*2+1) } Loop
    y(2*i+1) = x(i*2) - x(i*2+1) } Body
  }
}
```

```
for (int i=0; i < 2; i++)
{
  t0 = x[i];
  t1 = x[i+1];
  t2 = t0 + t1;
  y[i] = t2;
  t3 = t0 - t1;
  y[i+1] = t3;
}
```

f_3 (
Rep,
Rep[Array[Double]],
Rep[Array[Double]]
)

Unrolled, scalarized

```
def f3(x: Array[Rep[Double]], y: Array[Rep[Double]]) = {
  for (i <- 0 until 2) {
    y(2*i) = x(i*2) + x(i*2+1)
    y(2*i+1) = x(i*2) - x(i*2+1)
  }
}
```

Scalars only
Unrolled

Unrolled, Arrays

```
def f3(x: Rep[Array[Double]], y: Rep[Array[Double]]) = {
  i = 1
  for (i <- 0 until 2) {
    y(2*i) = x(i*2) + x(i*2+1)
    y(2*i+1) = x(i*2) - x(i*2+1)
  }
}
```

```
t0 = x[0];
t1 = x[1];
t2 = t0 + t1;
y[0] = t2;
t3 = t0 - t1;
y[1] = t3;
t4 = x[2];
t5 = x[3];
t6 = t4 + t5;
y[2] = t6;
t7 = t4 - t5;
y[3] = t7;
```

```
def f[L[_],A[_],T](looptype: L, x: A[Array[T]], y: A[Array[T]]) = {
  for (i <- 0 until 2: L[Range]) {
    y(2*i) = x(i*2) + x(i*2+1)
    y(2*i+1) = x(i*2) - x(i*2+1)
  }
}
```

Abstraction over Code Types

Looped, Arrays

```
def f3(x: Rep[Array[Double]], y: Rep[Array[Double]]) = {
  for (i <- 0 until 2: Rep[Range]) {
    y(2*i) = x(i*2) + x(i*2+1) } Loop
    y(2*i+1) = x(i*2) - x(i*2+1) } Body
  }
}
```

```
for (int i=0; i < 2; i++)
{
  t0 = x[i];
  t1 = x[i+1];
  t2 = t0 + t1;
  y[i] = t2;
  t3 = t0 - t1;
  y[i+1] = t3;
}
```

f_3 (
Rep,
Rep[Array[Double]],
Rep[Array[Double]]
)

Unrolled, scalarized

```
def f3(x: Array[Rep[Double]], y: Array[Rep[Double]]) = {
  for (i <- 0 until 2) {
    y(2*i) = x(i*2) + x(i*2+1)
    y(2*i+1) = x(i*2) - x(i*2+1)
  }
}
```

Scalars only
Unrolled

Unrolled, Arrays

```
def f3(x: Rep[Array[Double]], y: Rep[Array[Double]]) = {
  for (i <- 0 until 2) {
    y(2*i) = x(i*2) + x(i*2+1)
    y(2*i+1) = x(i*2) - x(i*2+1)
  }
}
```

```
t0 = x[0];
t1 = x[1];
t2 = t0 + t1;
y[0] = t2;
t3 = t0 - t1;
y[1] = t3;
t4 = x[0];
t5 = x[1];
t6 = t4 + t5;
y[0] = t6;
t7 = t4 - t5;
y[3] = t7;
```

```
def f[L[_],A[_],T](looptype: L, x: A[Array[T]], y: A[Array[T]]) = {
  for (i <- 0 until 2: L[Range]) {
    y(2*i) = x(i*2) + x(i*2+1)
    y(2*i+1) = x(i*2) - x(i*2+1)
  }
}
```

type NoRep[T] = T

Abstraction over Code Types

Looped, Arrays

```
def f3(x: Rep[Array[Double]], y: Rep[Array[Double]]) = {
  for (i <- 0 until 2: Rep[Range]) {
    y(2*i) = x(i*2) + x(i*2+1) } Loop
    y(2*i+1) = x(i*2) - x(i*2+1) } Body
  }
}
```

```
for (int i=0; i < 2; i++)
{
  t0 = x[i];
  t1 = x[i+1];
  t2 = t0 + t1;
  y[i] = t2;
  t3 = t0 - t1;
  y[i+1] = t3;
}
```

f_3 (
Rep,
Rep[Array[Double]],
Rep[Array[Double]]
)

Unrolled, scalarized

```
def f3(x: Array[Rep[Double]], y: Array[Rep[Double]]) = {
  for (i <- 0 until 2) {
    y(2*i) = x(i*2) + x(i*2+1)
    y(2*i+1) = x(i*2) - x(i*2+1)
  }
}
```

Scalars only
Unrolled

Unrolled, Arrays

```
def f3(x: Rep[Array[Double]], y: Rep[Array[Double]]) = {
  i = 1
  for (i <- 0 until 2) {
    y(2*i) = x(i*2) + x(i*2+1)
    y(2*i+1) = x(i*2) - x(i*2+1)
  }
}
```

```
t0 = x[0];
t1 = x[1];
t2 = t0 + t1;
y[0] = t2;
t3 = t0 - t1;
y[1] = t3;
t4 = x[2];
t5 = x[3];
t6 = t4 + x5;
y[2] = t6;
t7 = t4 - x5;
y[3] = t7;
```

f_3 (
NoRep,
Rep[Array[Double]],
Rep[Array[Double]]
)

```
def f[L[_],A[_],T](looptype: L, x: A[Array[T]], y: A[Array[T]]) = {
  for (i <- 0 until 2: L[Range]) {
    y(2*i) = x(i*2) + x(i*2+1)
    y(2*i+1) = x(i*2) - x(i*2+1)
  }
}
```

type NoRep[T] = T

Abstraction over Code Types

Looped, Arrays

```
def f3(x: Rep[Array[Double]], y: Rep[Array[Double]]) = {
  for (i <- 0 until 2: Rep[Range]) {
    y(2*i) = x(i*2) + x(i*2+1)
    y(2*i+1) = x(i*2) - x(i*2+1)
  }
}
```

```
for (int i=0; i < 2; i++)
{
  t0 = x[i];
  t1 = x[i+1];
  t2 = t0 + t1;
  y[i] = t2;
  t3 = t0 - t1;
  y[i+1] = t3;
}
```

f_3 (
Rep,
Rep[Array[Double]],
Rep[Array[Double]]
)

Unrolled, scalarized

```
def f3(x: Array[Rep[Double]], y: Array[Rep[Double]]) = {
  for (i <- 0 until 2) {
    y(2*i) = x(i*2) + x(i*2+1)
    y(2*i+1) = x(i*2) - x(i*2+1)
  }
}
```

Scalars only
Unrolled

f_3 (
NoRep,
NoRep[Array[Rep[Double]]],
NoRep[Array[Rep[Double]]]
)

Unrolled, Arrays

```
def f3(x: Rep[Array[Double]], y: Rep[Array[Double]]) = {
  i = 1
  for (i <- 0 until 2) {
    y(2*i) = x(i*2) + x(i*2+1)
    y(2*i+1) = x(i*2) - x(i*2+1)
  }
}
```

```
t0 = x[0];
t1 = x[1];
t2 = t0 + t1;
y[0] = t2;
t3 = t0 - t1;
y[1] = t3;
t4 = x[0];
t5 = x[1];
t6 = t4 + t5;
y[0] = t6;
t7 = t4 - t5;
y[3] = t7;
```

f_3 (
NoRep,
Rep[Array[Double]],
Rep[Array[Double]]
)

```
def f[L[_],A[_],T](looptype: L, x: A[Array[T]], y: A[Array[T]]) = {
  for (i <- 0 until 2: L[Range]) {
    y(2*i) = x(i*2) + x(i*2+1)
    y(2*i+1) = x(i*2) - x(i*2+1)
  }
}
```

type NoRep[T] = T

Abstraction over Code Types

Looped, Arrays

```
def f3(x: Rep[Array[Double]], y: Rep[Array[Double]]) = {
  for (i <- 0 until 2: Rep[Range]) {
    y(2*i) = x(i*2) + x(i*2+1)
    y(2*i+1) = x(i*2) - x(i*2+1)
  }
}
```

```
for (int i=0; i < 2; i++)
{
  t0 = x[i];
  t1 = x[i+1];
  t2 = t0 + t1;
  y[i] = t2;
  t3 = t0 - t1;
  y[i+1] = t3;
}
```

f_3 (
Rep,
Rep[Array[Double]],
Rep[Array[Double]]
)

Unrolled, scalarized

```
def f3(x: Array[Rep[Double]], y: Array[Rep[Double]]) = {
  for (i <- 0 until 2) {
    y(2*i) = x(i*2) + x(i*2+1)
    y(2*i+1) = x(i*2) - x(i*2+1)
  }
}
```

Scalars only
Unrolled

```
t0 = s0 + s1;
t1 = s0 - s1;
t2 = s2 + s3;
t3 = s2 - s3;
```

f_3 (
NoRep,
NoRep[Array[Rep[Double]]],
NoRep[Array[Rep[Double]]]
)

Unrolled, Arrays

```
def f3(x: Rep[Array[Double]], y: Rep[Array[Double]]) = {
  for (i <- 0 until 2) {
    y(2*i) = x(i*2) + x(i*2+1)
    y(2*i+1) = x(i*2) - x(i*2+1)
  }
}
```

```
t0 = x[0];
t1 = x[1];
t2 = t0 + t1;
y[0] = t2;
t3 = t0 - t1;
y[1] = t3;
t4 = x[0];
t5 = x[1];
t6 = t4 + t5;
y[0] = t6;
t7 = t4 - t5;
y[3] = t7;
```

f_3 (
NoRep,
Rep[Array[Double]],
Rep[Array[Double]]
)

```
def f[L[_],A[_],T](looptype: L, x: A[Array[T]], y: A[Array[T]]) = {
  for (i <- 0 until 2: L[Range]) {
    y(2*i) = x(i*2) + x(i*2+1)
    y(2*i+1) = x(i*2) - x(i*2+1)
  }
}
```

type NoRep[T] = T

*Typeclasses omitted for simplicity

Option 3: Staging in Scala

Σ -SPL

Code Style

Data Layout

AST

Code

Scala Staging Program

$$T_m^n \left(\sum_{j=0}^2 S(h_j) \mathbf{DFT}_2 G(h_j) \right)$$

Looped Code

Unrolled Code

Code with
Precomputation

C99 Complex
Interleaved Complex
Split Complex

C99 Complex
Interleaved Complex
Split Complex

C99 Complex
Interleaved Complex
Split Complex

Staging decisions



Abstraction over Data Layout

```
class DataAbstraction[T] {  
  def apply (i: Int): T  
  def update (i: Int, y: T)  
}
```

Abstraction over Data Layout

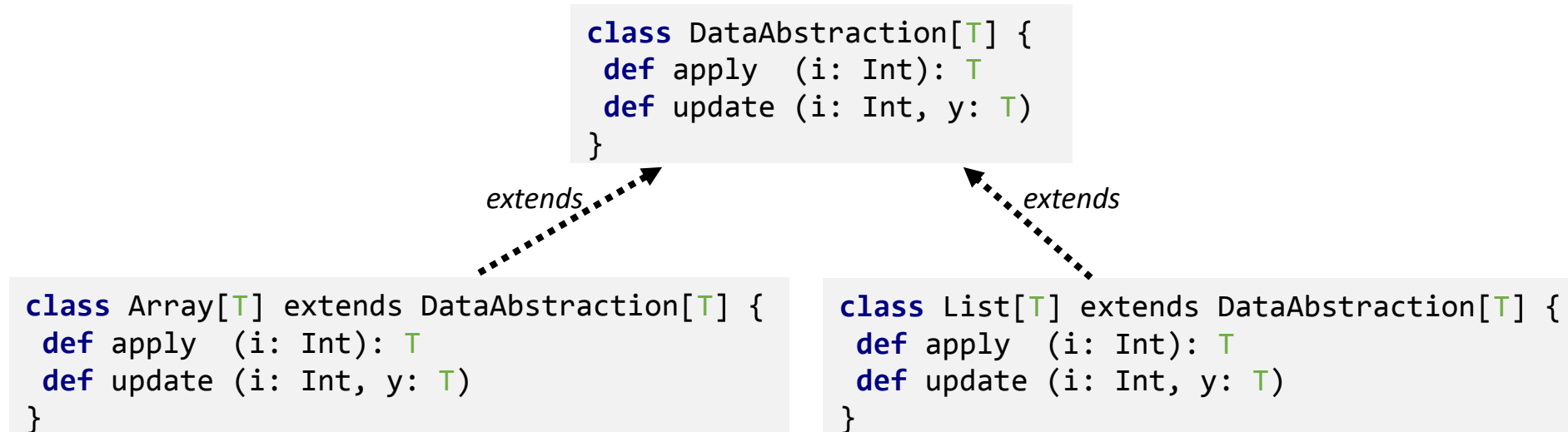
```
class DataAbstraction[T] {  
  def apply (i: Int): T  
  def update (i: Int, y: T)  
}
```

extends

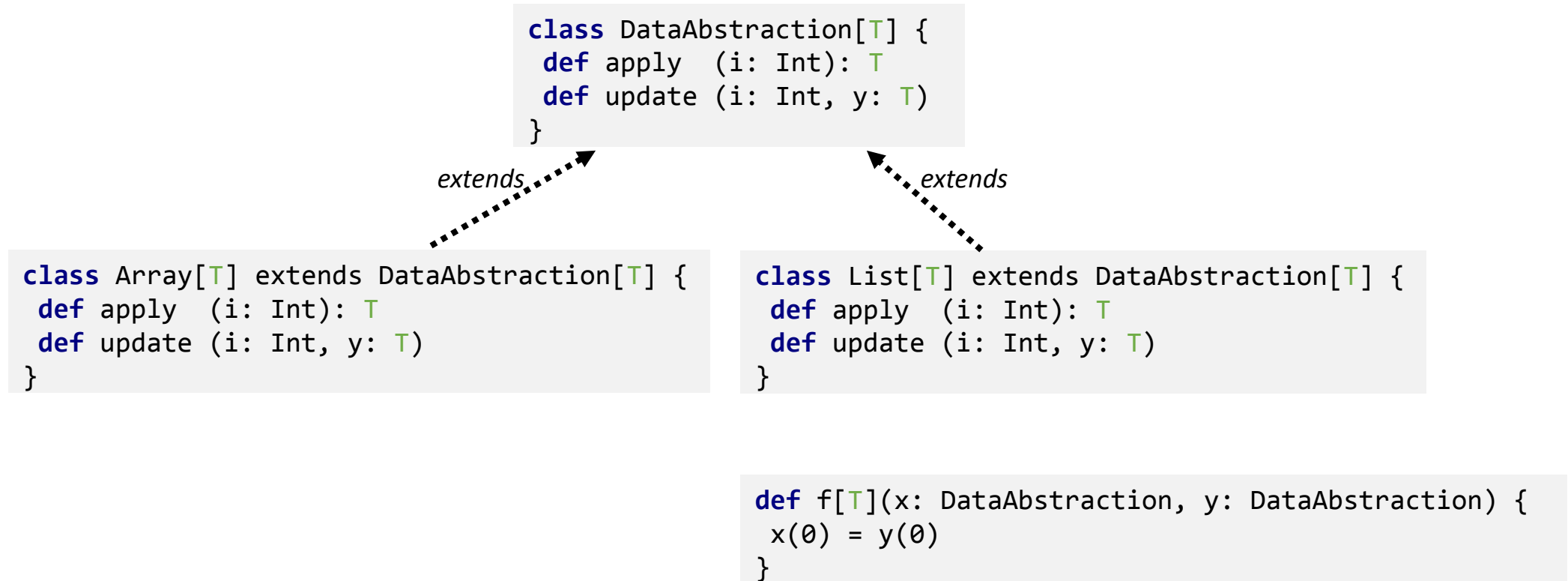


```
class Array[T] extends DataAbstraction[T] {  
  def apply (i: Int): T  
  def update (i: Int, y: T)  
}
```

Abstraction over Data Layout



Abstraction over Data Layout



Abstraction over Data Layout

```
class DataAbstraction[T] {  
  def apply (i: Int): T  
  def update (i: Int, y: T)  
}
```

extends

```
class Array[T] extends DataAbstraction[T] {  
  def apply (i: Int): T  
  def update (i: Int, y: T)  
}
```

extends

```
class List[T] extends DataAbstraction[T] {  
  def apply (i: Int): T  
  def update (i: Int, y: T)  
}
```

```
val a = Array[Double](3)  
val b = List[Double](3)  
  
f(a,b)
```

```
def f[T](x: DataAbstraction, y: DataAbstraction) {  
  x(0) = y(0)  
}
```

Abstraction over Data Layout

```
class Vector[A[_], T, L]{  
  type looptype = L  
  def apply (i: A[Int]): T  
  def update (i: A[Int], y: T)  
}
```

Abstraction over Data Layout

```
class Vector[A[_], T, L]{  
  type looptype = L  
  def apply (i: A[Int]): T  
  def update (i: A[Int], y: T)  
}
```

extends

```
class LoopedArray extends Vector[Rep,Rep[Double], Rep] {  
  val data: Rep[Array[Double]]  
  def apply (i: Rep[Int]): Rep[Double] = data(i)  
  def update (i: Rep[Int], y: Rep[Double]) = { data(i) = y }  
}
```

Abstraction over Data Layout

```
class Vector[A[_], T, L]{  
  type looptype = L  
  def apply (i: A[Int]): T  
  def update (i: A[Int], y: T)  
}
```

extends

extends

```
class LoopedArray extends Vector[Rep,Rep[Double], Rep] {  
  val data: Rep[Array[Double]]  
  def apply (i: Rep[Int]): Rep[Double] = data(i)  
  def update (i: Rep[Int], y: Rep[Double]) = { data(i) = y }  
}
```

```
class UnrolledScalar extends Vector [NoRep, Rep[Double], NoRep] {  
  val data: Array[Rep[Double]]  
  def apply (i: Int): Rep[Double]  
  def update (i: Int, y: Rep[Double])  
}
```


Abstraction over Data Layout

```
class Vector[A[_], T, L]{  
  type looptype = L  
  def apply (i: A[Int]): T  
  def update (i: A[Int], y: T)  
}
```

extends

```
class LoopedArray extends Vector[Rep,Rep[Double], Rep] {  
  val data: Rep[Array[Double]]  
  def apply (i: Rep[Int]): Rep[Double] = data(i)  
  def update (i: Rep[Int], y: Rep[Double]) = { data(i) = y }  
}
```

extends

```
class UnrolledScalar extends Vector [NoRep, Rep[Double], NoRep] {  
  val data: Array[Rep[Double]]  
  def apply (i: Int): Rep[Double]  
  def update (i: Int, y: Rep[Double])  
}
```

```
def f[A1[_], A2[_], L1, L2, T] (x: Vector[A1[_], T, L1], y: Vector[A1[_], T, L2]) = {  
  for (i <- 0 until 2: y.looptype) {  
    y(2*i)    = x(i*2) + x(i*2+1)  
    y(2*i+1) = x(i*2) - x(i*2+1)  
  }  
}
```

```
val x = new LoopedArray  
val y = new UnrolledScalar  
f(x,y)
```

Abstraction over Data Layout

```
class Vector[A[_], T, L]{  
  type looptype = L  
  def apply (i: A[Int]): T  
  def update (i: A[Int], y: T)  
}
```

extends

```
class LoopedArray extends Vector[Rep,Rep[Double], Rep] {  
  val data: Rep[Array[Double]]  
  def apply (i: Rep[Int]): Rep[Double] = data(i)  
  def update (i: Rep[Int], y: Rep[Double]) = { data(i) = y }  
}
```

extends

```
class UnrolledScalar extends Vector [NoRep, Rep[Double], NoRep] {  
  val data: Array[Rep[Double]]  
  def apply (i: Int): Rep[Double]  
  def update (i: Int, y: Rep[Double])  
}
```

```
def f[A1[_], A2[_], L1, L2, T] (x: Vector[A1[_], T, L1], y: Vector[A1[_], T, L2]) = {  
  for (i <- 0 until 2: y.looptype) {  
    y(2*i)    = x(i*2) + x(i*2+1)  
    y(2*i+1) = x(i*2) - x(i*2+1)  
  }  
}
```

x

```
val x = new LoopedArray  
val y = new UnrolledScalar  
f(x,y)
```

Abstraction over Data Layout

```
class Vector[A[_], T, L]{  
  type looptype = L  
  def apply (i: A[Int]): T  
  def update (i: A[Int], y: T)  
}
```

extends

```
class LoopedArray extends Vector[Rep,Rep[Double], Rep] {  
  val data: Rep[Array[Double]]  
  def apply (i: Rep[Int]): Rep[Double] = data(i)  
  def update (i: Rep[Int], y: Rep[Double]) = { data(i) = y }  
}
```

extends

```
class UnrolledScalar extends Vector [NoRep, Rep[Double], NoRep] {  
  val data: Array[Rep[Double]]  
  def apply (i: Int): Rep[Double]  
  def update (i: Int, y: Rep[Double])  
}
```

```
def f[A1[_], A2[_], L1, L2, T] (x: Vector[A1[_], T, L1], y: Vector[A1[_], T, L2]) = {  
  for (i <- 0 until 2: y.looptype) {  
    y(2*i)    = x(i*2) + x(i*2+1)  
    y(2*i+1) = x(i*2) - x(i*2+1)  
  }  
}
```



```
val x = new LoopedArray  
val y = new UnrolledScalar  
f(x,y)
```

y



Abstraction over Data Layout

```
class Vector[A[_], T, L]{  
  type looptype = L  
  def apply (i: A[Int]): T  
  def update (i: A[Int], y: T)  
}
```

extends

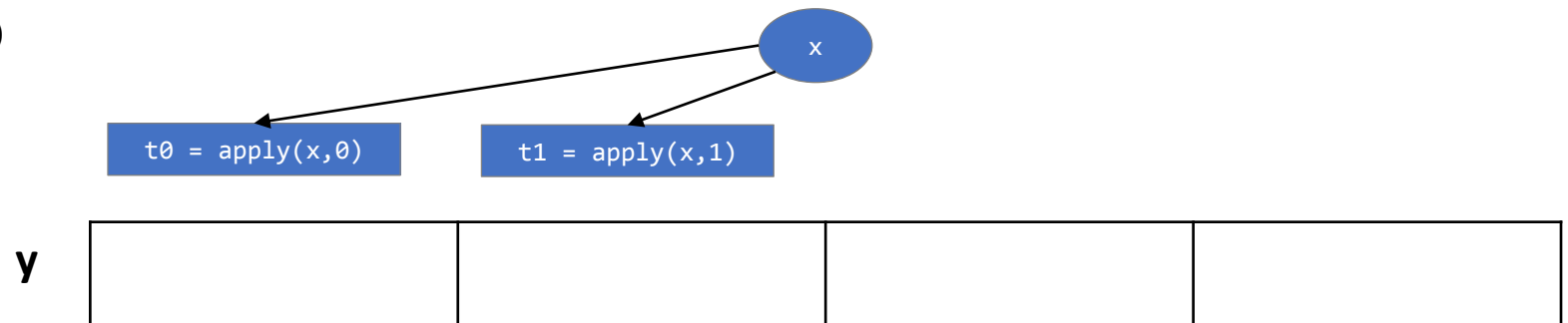
```
class LoopedArray extends Vector[Rep,Rep[Double], Rep] {  
  val data: Rep[Array[Double]]  
  def apply (i: Rep[Int]): Rep[Double] = data(i)  
  def update (i: Rep[Int], y: Rep[Double]) = { data(i) = y }  
}
```

extends

```
class UnrolledScalar extends Vector [NoRep, Rep[Double], NoRep] {  
  val data: Array[Rep[Double]]  
  def apply (i: Int): Rep[Double]  
  def update (i: Int, y: Rep[Double])  
}
```

```
def f[A1[_], A2[_], L1, L2, T] (x: Vector[A1[_], T, L1], y: Vector[A1[_], T, L2]) = {  
  for (i <- 0 until 2: y.looptype) {  
    y(2*i)    = x(i*2) + x(i*2+1)  
    y(2*i+1) = x(i*2) - x(i*2+1)  
  }  
}
```

```
val x = new LoopedArray  
val y = new UnrolledScalar  
f(x,y)
```



Abstraction over Data Layout

```
class Vector[A[_], T, L]{  
  type looptype = L  
  def apply (i: A[Int]): T  
  def update (i: A[Int], y: T)  
}
```

extends

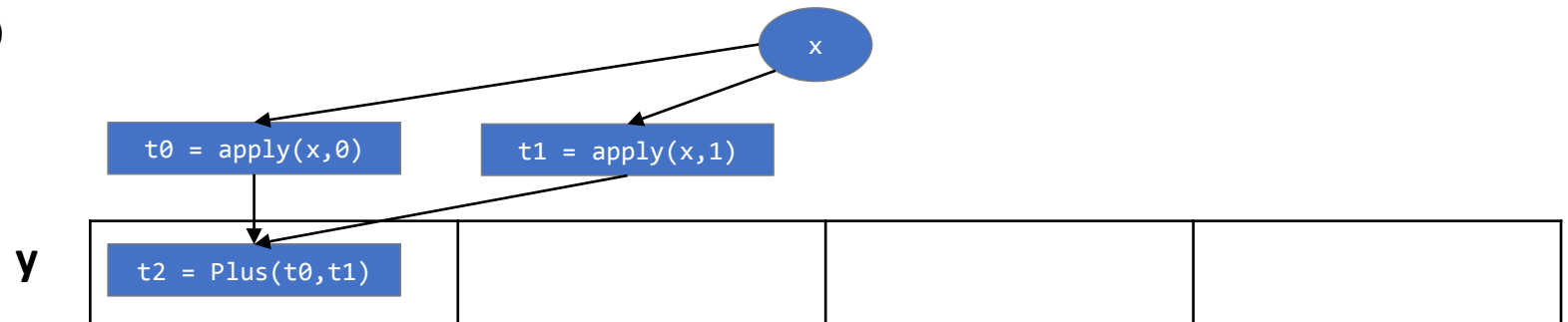
```
class LoopedArray extends Vector[Rep,Rep[Double], Rep] {  
  val data: Rep[Array[Double]]  
  def apply (i: Rep[Int]): Rep[Double] = data(i)  
  def update (i: Rep[Int], y: Rep[Double]) = { data(i) = y }  
}
```

extends

```
class UnrolledScalar extends Vector [NoRep, Rep[Double], NoRep] {  
  val data: Array[Rep[Double]]  
  def apply (i: Int): Rep[Double]  
  def update (i: Int, y: Rep[Double])  
}
```

```
def f[A1[_], A2[_], L1, L2, T] (x: Vector[A1[_], T, L1], y: Vector[A1[_], T, L2]) = {  
  for (i <- 0 until 2: y.looptype) {  
    y(2*i) = x(i*2) + x(i*2+1)  
    y(2*i+1) = x(i*2) - x(i*2+1)  
  }  
}
```

```
val x = new LoopedArray  
val y = new UnrolledScalar  
f(x,y)
```



Abstraction over Data Layout

```
class Vector[A[_], T, L]{
  type looptype = L
  def apply (i: A[Int]): T
  def update (i: A[Int], y: T)
}
```

extends

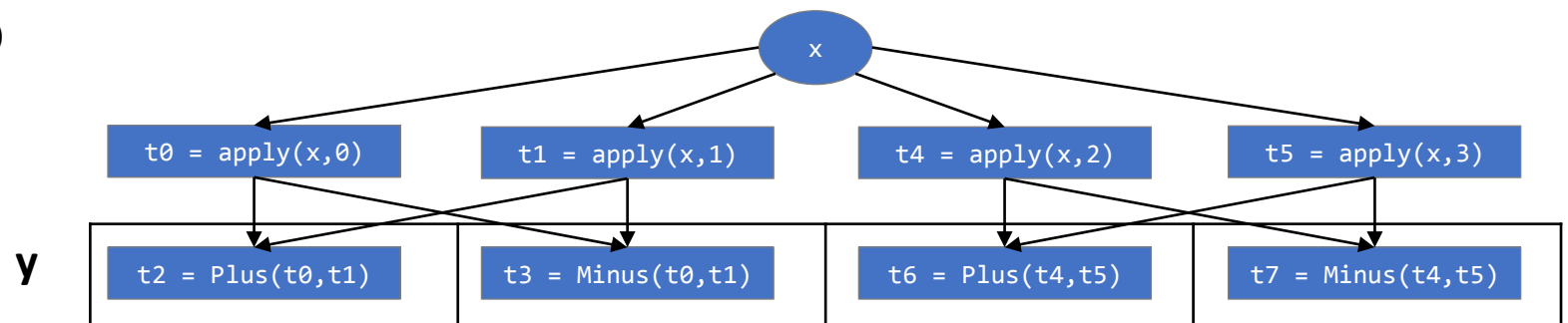
```
class LoopedArray extends Vector[Rep,Rep[Double], Rep] {
  val data: Rep[Array[Double]]
  def apply (i: Rep[Int]): Rep[Double] = data(i)
  def update (i: Rep[Int], y: Rep[Double]) = { data(i) = y }
}
```

extends

```
class UnrolledScalar extends Vector [NoRep, Rep[Double], NoRep] {
  val data: Array[Rep[Double]]
  def apply (i: Int): Rep[Double]
  def update (i: Int, y: Rep[Double])
}
```

```
def f[A1[_], A2[_], L1, L2, T] (x: Vector[A1[_], T, L1], y: Vector[A1[_], T, L2]) ={
  for (i <- 0 until 2: y.looptype) {
    y(2*i)    = x(i*2) + x(i*2+1)
    y(2*i+1) = x(i*2) - x(i*2+1)
  }
}
```

```
val x = new LoopedArray
val y = new UnrolledScalar
f(x,y)
```



Abstraction over Data Layout

```
class Vector[A[_], T, L]{  
  type looptype = L  
  def apply (i: A[Int]): T  
  def update (i: A[Int], y: T)  
}
```

extends

```
class LoopedArray extends Vector[Rep,Rep[Double], Rep] {  
  val data: Rep[Array[Double]]  
  def apply (i: Rep[Int]): Rep[Double] = data(i)  
  def update (i: Rep[Int], y: Rep[Double]) = { data(i) = y }  
}
```

extends

```
class UnrolledScalar extends Vector [NoRep, Rep[Double], NoRep] {  
  val data: Array[Rep[Double]]  
  def apply (i: Int): Rep[Double]  
  def update (i: Int, y: Rep[Double])  
}
```

extends

```
class StagedTree extends Vector [NoRep, Rep[Double], NoRep] {  
  val data: Rep[Tree[Double]]  
  def apply (i: Rep[Int]): Rep[Double] = data(i)  
  def update (i: Rep[Int], y: Rep[Double]) = { data(i) = y }  
}
```

Option 3: Staging in Scala

Σ-SPL

Code Style

Data Layout

AST

Code

$$T_m^n \left(\sum_{j=0}^2 S(h_j) \mathbf{DFT}_2 G(h_j) \right)$$

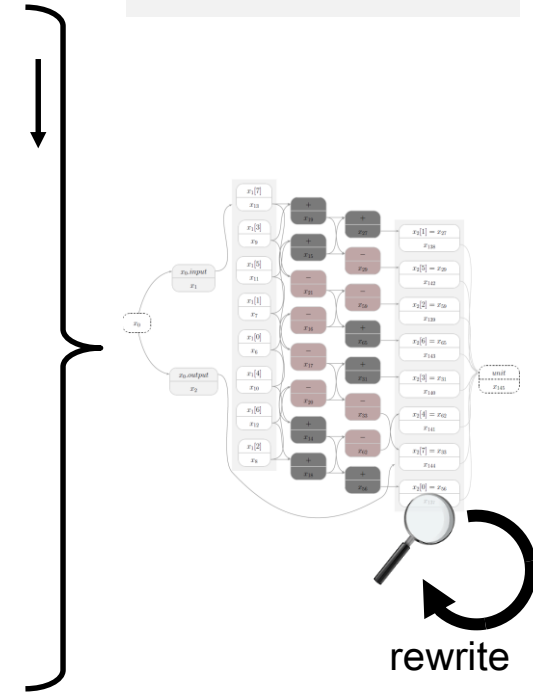
Looped Code
 Unrolled Code
 Code with Precomputation
Staging decisions

C99 Complex
 Interleaved Complex
 Split Complex

C99 Complex
 Interleaved Complex
 Split Complex

C99 Complex
 Interleaved Complex
 Split Complex

```
def f[A1[_], A2[_], L1, L2, T]
(x: Vector[A1[_], T, L1], y: Vector[A1[_], T, L2]) ...
```

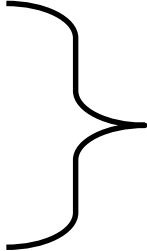


Benefits

Single Decision Point for Data Layout and Code Style

```
val sigmaspl = ...  
val f = sigmaspl.translate Creating the Meta Function
```

```
val input = new LoopedArrayInterleavedComplex  
val output = new LoopedArraySplitComplex  
val unrolled_code = new UnrolledScalarizedC99Complex  
val twiddles = TwiddleComputation[Rep]
```



Staging decisions

```
val ast = f(input,output,unrolled_code,twiddles) Instantiation
```

Benefits

Single Decision Point for Data Layout and Code Style

```
val sigmaspl = ...  
val f = sigmaspl.translate Creating the Meta Function
```

```
val input = new LoopedArrayInterleavedComplex  
val output = new LoopedArraySplitComplex  
val unrolled_code = new UnrolledScalarizedC99Complex  
val twiddles = TwiddleComputation[Rep]
```

} *Staging decisions*

```
val ast = f(input,output,unrolled_code,twiddles) Instantiation
```

Single Maintenance Point for Translations

```
def f[A1[_], A2[_], L1, L2, T] (x: Vector[A1[_], T, L1], y: Vector[A1[_], T, L2]) = {  
  for (i <- 0 until 2: y.looptype) {  
    y(2*i) = x(i*2) + x(i*2+1)  
    y(2*i+1) = x(i*2) - x(i*2+1)  
  }  
}
```

Benefits

Single Decision Point for Data Layout and Code Style

```
val sigmaspl = ...  
val f = sigmaspl.translate Creating the Meta Function
```

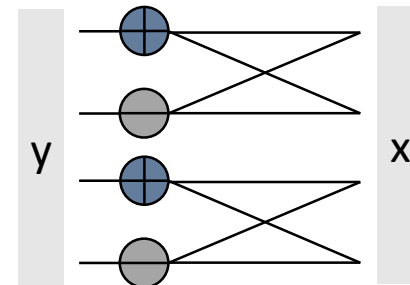
```
val input = new LoopedArrayInterleavedComplex  
val output = new LoopedArraySplitComplex  
val unrolled_code = new UnrolledScalarizedC99Complex  
val twiddles = TwiddleComputation[Rep]
```

} *Staging decisions*

```
val ast = f(input,output,unrolled_code,twiddles) Instantiation
```

Single Maintenance Point for Translations

```
def f[A1[_], A2[_], L1, L2, T] (x: Vector[A1[_], T, L1], y: Vector[A1[_], T, L2]) = {  
  for (i <- 0 until 2: y.looptype) {  
    y(2*i) = x(i*2) + x(i*2+1)  
    y(2*i+1) = x(i*2) - x(i*2+1)  
  }  
}
```



Benefits

Certain Errors Manifest at Compile Time

DFT(47) has ~16.000 Decomposition

Project

- spiral [Spirals] (C:\Users\Georg\spiral)
 - .idea
 - .idea_modules
 - doc
 - lib
 - project [Spirals-build] (sources root)
 - scripts
 - src
 - ch.ethz.spiral
 - datatypes
 - db
 - dsls
 - dir
 - intrinsic
 - CIR_DSL
 - FSigmaSPL_DSL.scala
 - Intrinsic_DSL.scala
 - SigmaSPL.scala
 - SigmaSPLBase.scala
 - SPL.scala

```

}
def ff2[Vx[_], Ex[_], Px[_], Rx[_], Tx](x: CVector[Vx, Ex, Rx, Px, Tx], y: CVector[Vx, Ex, Rx, Px, Tx], size: Int) {
  val vrepx = y.vrep
  val erepx = y.erep

  C.comment("start F2")
  val idx0 = vrepx(0)
  val idx1 = vrepx(1)
  val t1 = x(idx0)
  val t2 = x(idx1)
  val r1 = erepx.plus(t1, t2)
  val r2 = erepx.minus(t1, t2)
  y.update(idx0, r1)
  y.update(idx1, r2)
  C.comment("end F2")
}

```

Debugger Console

Frames

- "main" @ 1 in group "main": RUNNING
- ff2():188, SigmaSPL2CIR (ch.ethz.spiral.rewrites)
- apply\$mcV\$sp():641, SigmaSPL2CIR\$\$anonfun\$translate\$13\$\$anonfun\$apply\$mcV\$sp\$6 (ch.ethz.spiral.rewrites)
- apply():641, SigmaSPL2CIR\$\$anonfun\$translate\$13\$\$anonfun\$apply\$mcV\$sp\$6 (ch.ethz.spiral.rewrites)
- __ifThenElse():14, IfThenElse\$class (scala.virtualization.lms.common)
- __ifThenElse():224, SigmaSPLBase (ch.ethz.spiral.dsls)
- apply\$mcV\$sp():633, SigmaSPL2CIR\$\$anonfun\$translate\$13 (ch.ethz.spiral.rewrites)
- apply():633, SigmaSPL2CIR\$\$anonfun\$translate\$13 (ch.ethz.spiral.rewrites)
- __ifThenElse():14, IfThenElse\$class (scala.virtualization.lms.common)
- __ifThenElse():224, SigmaSPLBase (ch.ethz.spiral.dsls)
- translate():624, SigmaSPL2CIR (ch.ethz.spiral.rewrites)
- traverseStm():409, SigmaSPLBase2CIR (ch.ethz.spiral.rewrites)

Variables

- this = {ch.ethz.spiral.scoring.Scoring\$trans\$2\$@4190}
- x = {ch.ethz.spiral.datatypes.ComplexDataTypes\$Interleaved}
- y = {ch.ethz.spiral.datatypes.ComplexDataTypes\$Interleaved}
 - size = 2
- vrepx = {ch.ethz.spiral.datatypes.ImplicitDataTypes\$ImplicitOp}
- erepx = {ch.ethz.spiral.datatypes.ImplicitDataTypes\$ImplicitOp}
- idx0 = {java.lang.Integer@4206}*0
- idx1 = {java.lang.Integer@4207}*1
- t1 = {ch.ethz.spiral.datatypes.ImplicitDataTypes\$ComplexElem}

Watches

- x.data = {scala.virtualization.lms.internal.Expression}

SBT Console SBT Action

```

[warn] It would fail on the following in
[warn]     val pattern = subpatterns
[warn]
[warn] non-variable type argument scala.
[warn] non-variable type argument scala.
[warn] there were 9 deprecation warning(s)
[warn] there were 579 feature warning(s)
[warn] 15 warnings found
[info] Compiling 1 Scala source to C:\Us
[success] Total time: 158 s, completed 2
[info] Set current project to Spirals (i
> [info] Set current project to Spirals
[warn] Potentially incompatible versions
[warn]   org.scala-lang: 2.10.2-RC1, 2.
[success] Total time: 0 s, completed 20-
[info] Set current project to Spirals (i
>

```

References

<http://spiral.net/>

<http://spiral.net/software/spiral-scala.html>

- Georg Ofenbeck, Tiark Rompf, Alen Stojanov, Martin Odersky and Markus Püsichel. *Spiral in Scala: Towards the Systematic Construction of Generators for Performance Libraries, GPCE 2013*
- Tiark Rompf, Martin Odersky. *Lightweight Modular Staging: A Pragmatic Approach to Runtime Code Generation and Compiled DSLs*. Commun. ACM 55(6): 121-130 (2012)
- Markus Püsichel, Franz Franchetti and Yevgen Voronenko
[Spiral](#)
in Encyclopedia of Parallel Computing, Eds. David Padua, Springer 2011