A presentation by Samuel Häusler about the paper:

SPL

A Language and Compiler for DSP Algorithms

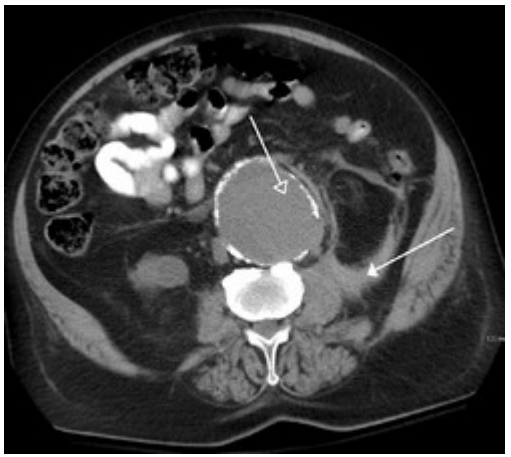
Jiangxing Jong, Jeremy Johnson, Robert Johnson and David Padua

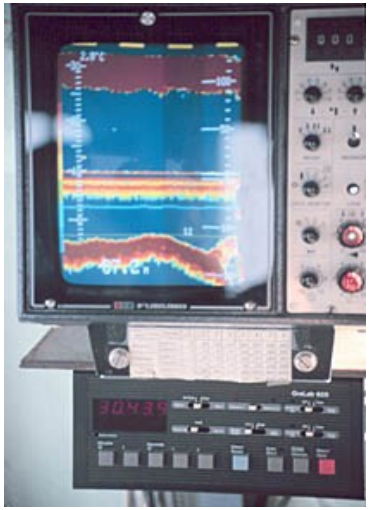
Overview

- **Motivation**
- **The language SPL**
- **The SPL compiler**
- **Experiments and results**
- **Conclusion**

Motivation

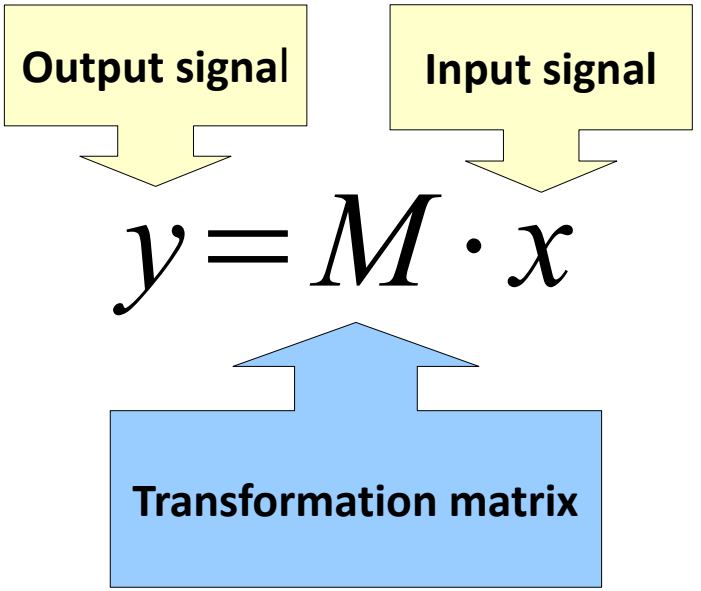
Digital Signal Processing is part of everyday life





Pictures: wikipedia.org

Signal transformations:



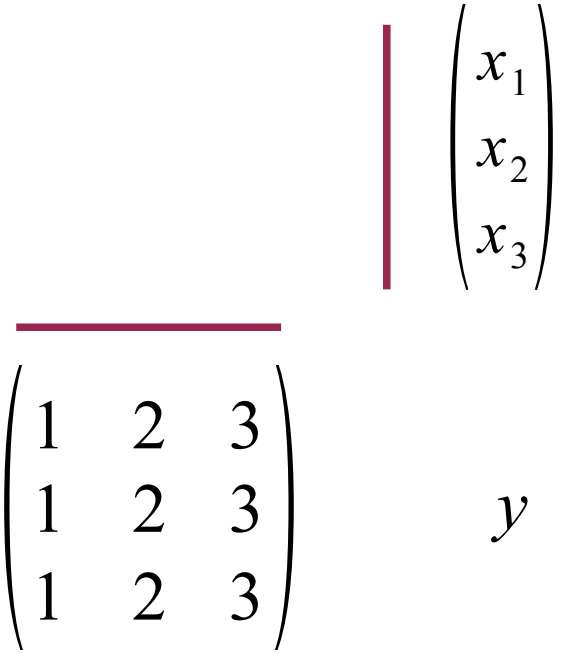
Example: Discrete Fourier transformation (1D)

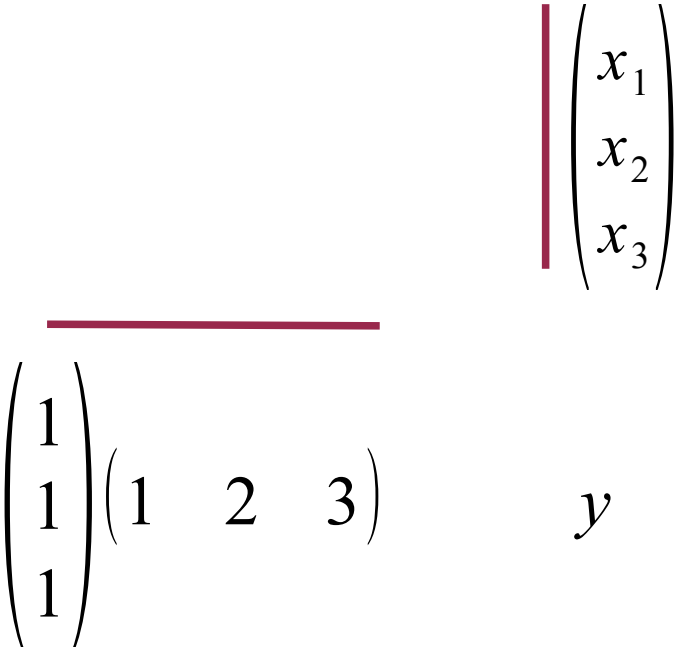
$$y = F_n \cdot x \quad \text{where } F_{pq} = \omega_n^{pq} \text{ with } \omega = e^{\frac{2\pi i}{n}}$$

$$F_4 = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -i & -1 & i \\ 1 & -1 & 1 & -1 \\ 1 & i & -1 & -i \end{pmatrix}$$

Fast signal transforms

$$\left(\begin{array}{ccc} 1 & 0 & 2 \\ 0 & 3 & 0 \\ 1 & 0 & 2 \end{array}\right) \left| \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \right. = \begin{pmatrix} x_1 + 2\,x_3 \\ 3\,x_2 \\ x_1 + 2\,x_3 \end{pmatrix}$$


$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$
$\begin{pmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \\ 1 & 2 & 3 \end{pmatrix}$
y


$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$
$\begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \begin{pmatrix} 1 & 2 & 3 \end{pmatrix}$
y

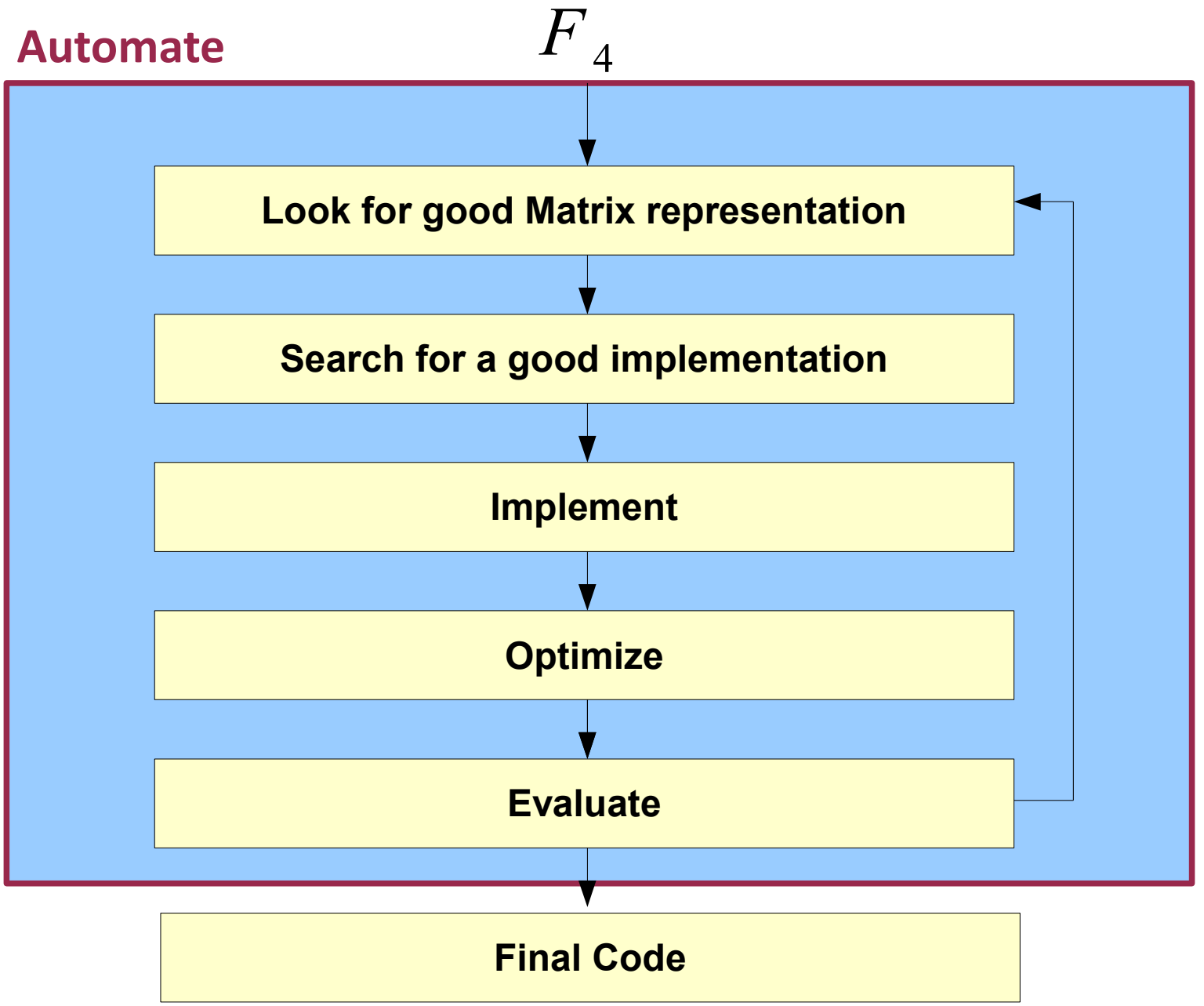
Fast Fourier Transform

$$F_4 = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -i & -1 & i \\ 1 & -1 & 1 & -1 \\ 1 & i & -1 & -i \end{pmatrix} = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -i \end{pmatrix} \cdot \begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & -1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$F_4 = (F_2 \otimes I_2)\, T_2^4\, (I_2 \otimes F_2)\, L_2^4$$

$$F_2 = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \qquad I_2 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

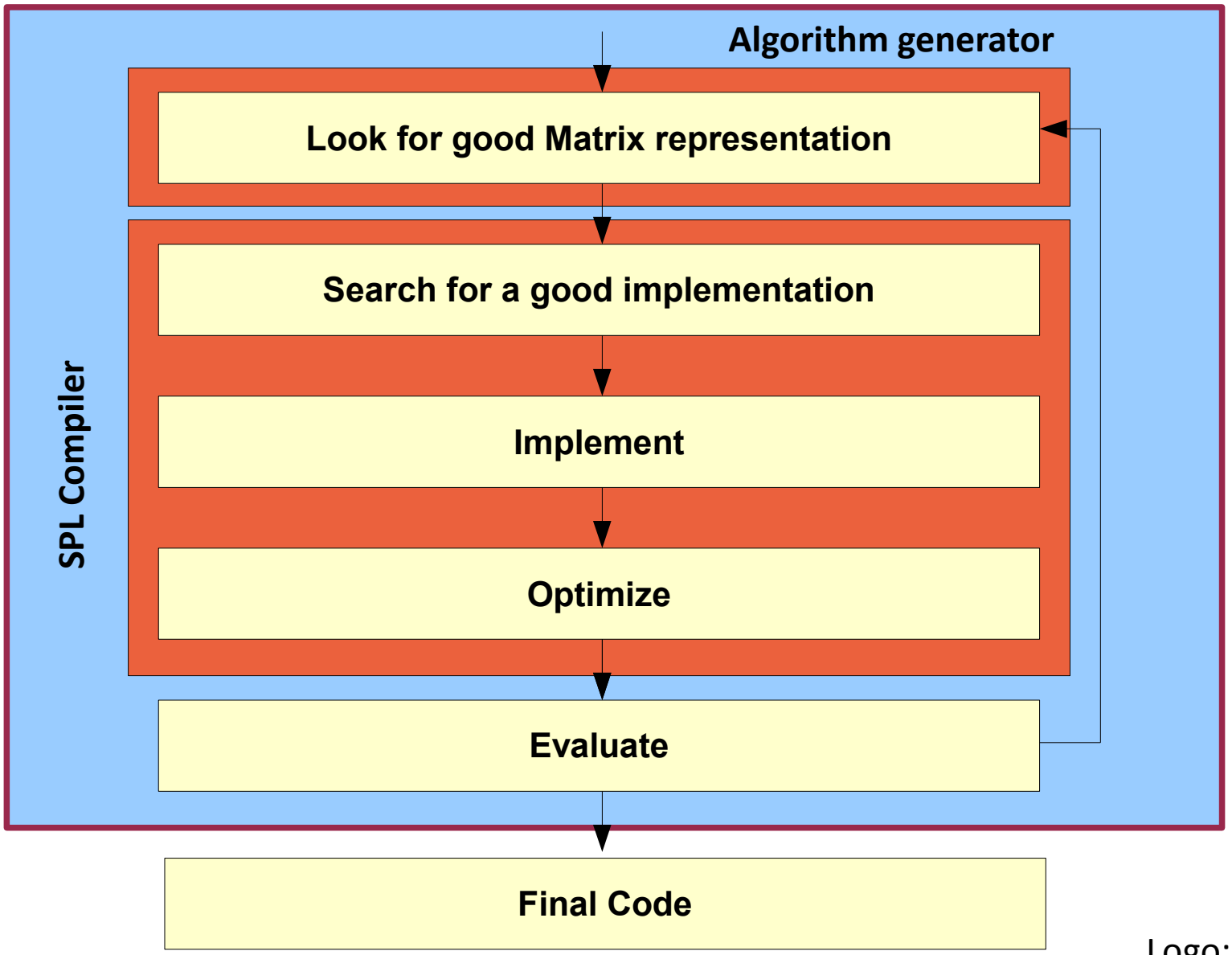
Implementation of signal transformations



Example:

SPIRAL
Software/Hardware Generation for DSP Algorithms

$$F_4$$



Logo:Spiral.net

Algorithm generator:

$$F_4 = (I_2 \otimes F_2)\, T_2^4 (I_2 \otimes F_2)\, L_2^4$$

```
(compose
     (tensor (F 2) (I 2))
(T 4 2)
     (tensor (I 2) (F 2))
(L 4 2))
```

Compiler

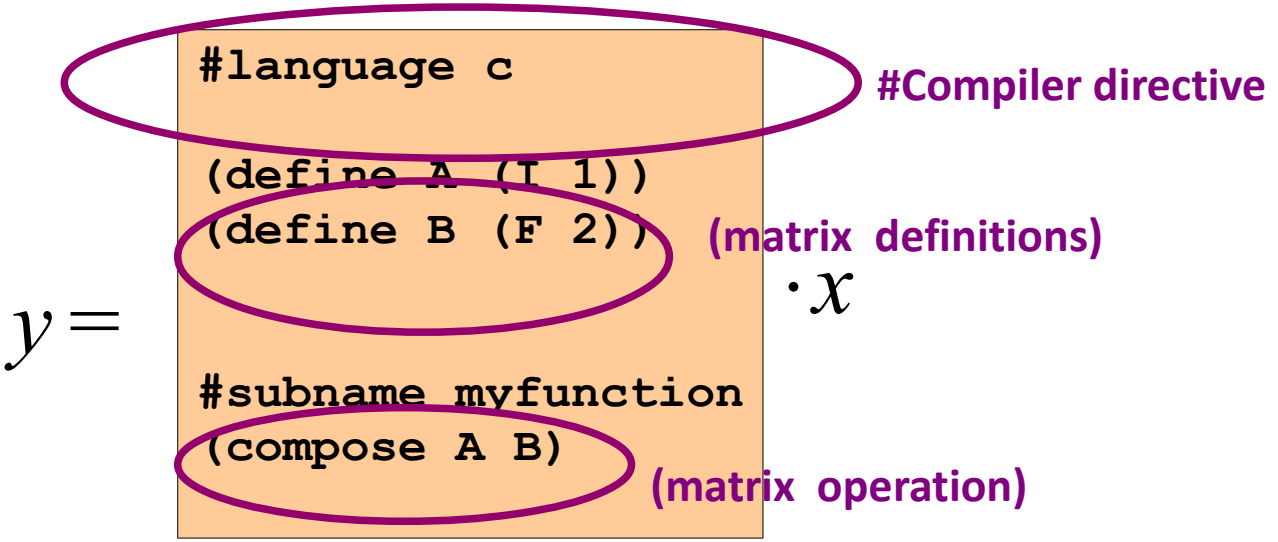
```
(template (compose A B))

   [A.in_size==B.out_size]

   (B_ ($in,$t0,0,0,1,1)
   A_($t0,$out,0,0,1,1))
```

Overview

- Motivation
- The language SPL
- The SPL compiler
- Experiments and results
- Conclusion

The language SPL

A SPL program is a transformation matrix



General matrix definitions

```
(matrix ((a11 a12 a13)(a21 a22 a23)(a31 a32 a33))
```

$$\begin{pmatrix} a11 & a12 & a13 \\ a21 & a22 & a23 \\ a31 & a32 & a33 \end{pmatrix}$$

```
(diagonal (a11 a22 a33))
```

$$\begin{pmatrix} a11 & 0 & 0 \\ 0 & a22 & 0 \\ 0 & 0 & a33 \end{pmatrix}$$

```
(permutation(3 1 2))
```

$$\begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}$$

Parametrized matrices

Identity matrix: `(I n)`

Fourier transformation: `(F n)`

Stride permutation matrix: `(L mn n)`

for index j: $F_r^n : j \Rightarrow j \cdot r \bmod n$

$$L_2^4 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Twiddle matrix: `(T mn n)`

$$T_m^{mn} = \begin{pmatrix} I & 0 & 0 \\ 0 & W_m & 0 \\ 0 & 0 & W_m^{(n-1)} \end{pmatrix}$$

Matrix operations

Matrix product: `(compose A B)`

Direct sum: `(direct-sum A B)`

Tensor product: `(tensor A B)`

$$\begin{pmatrix} 1 & 0 & 2 \\ 0 & 3 & 0 \\ 1 & 0 & 2 \end{pmatrix} \otimes A = \begin{pmatrix} A & 0 & 2A \\ 0 & 3A & 0 \\ A & 0 & 2\mathrm{A} \end{pmatrix}$$

Name assignment:

```
(define name formula)
```

```
e.g. (define A (I 1))
```

Program example

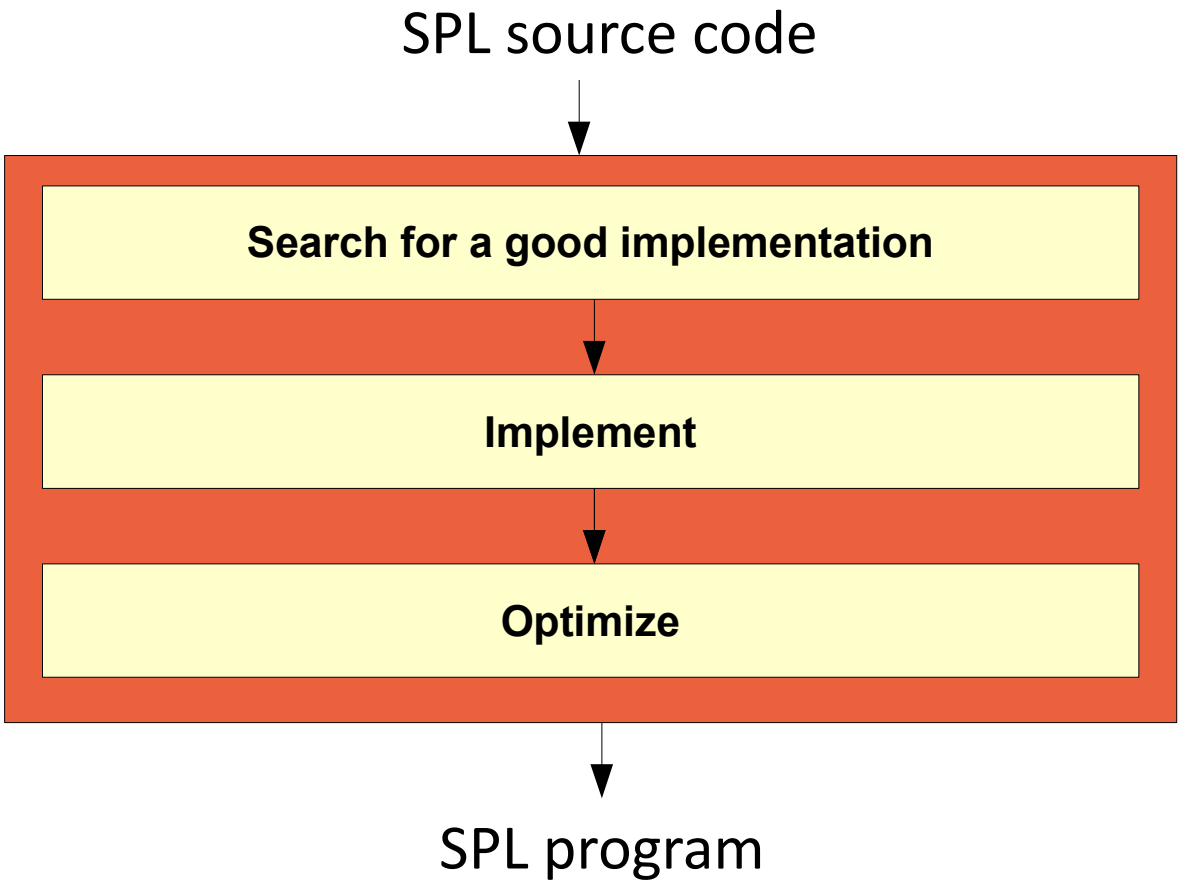
```
(define F4
   (compose
      (tensor (F 2) (I 2)) (T 4 2)
      (tensor (I 2) (F 2)) (L 4 2)))
#subname fft16
(compose
   (tensor F4 (I 4)) (T 16 4)
      (tensor (I 4) F4) (L 16 4))
```

$$F_{16} = (F_4 \otimes I_4)\, T_4^{16} (I_4 \otimes F_4)\, L_4^{16}, \text{ where } F_4 = (F_2 \otimes I_2)\, T_2^4 (I_2 \otimes F_2)\, L_2^4$$

Overview

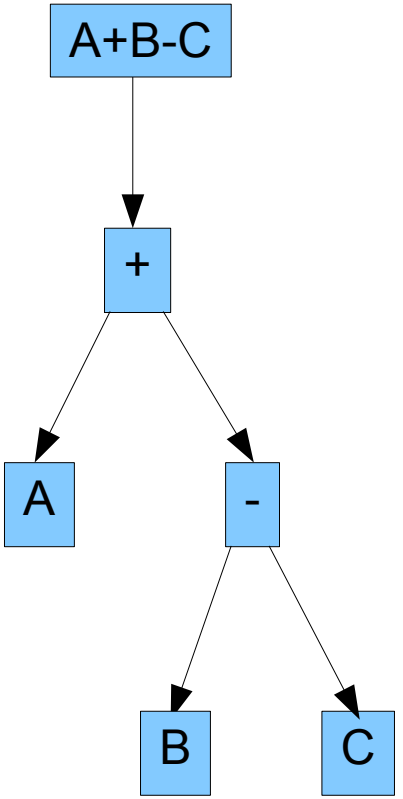
- Motivation
- The language SPL
- The SPL compiler
- Experiments and results
- Conclusion

The compiler

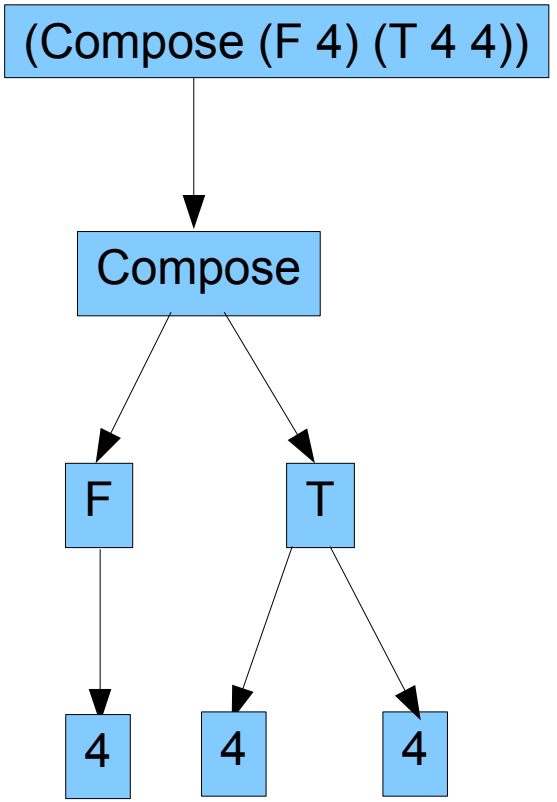


Parsing

Abstract syntax tree:



Example (compose (F 4) (T 4 4))



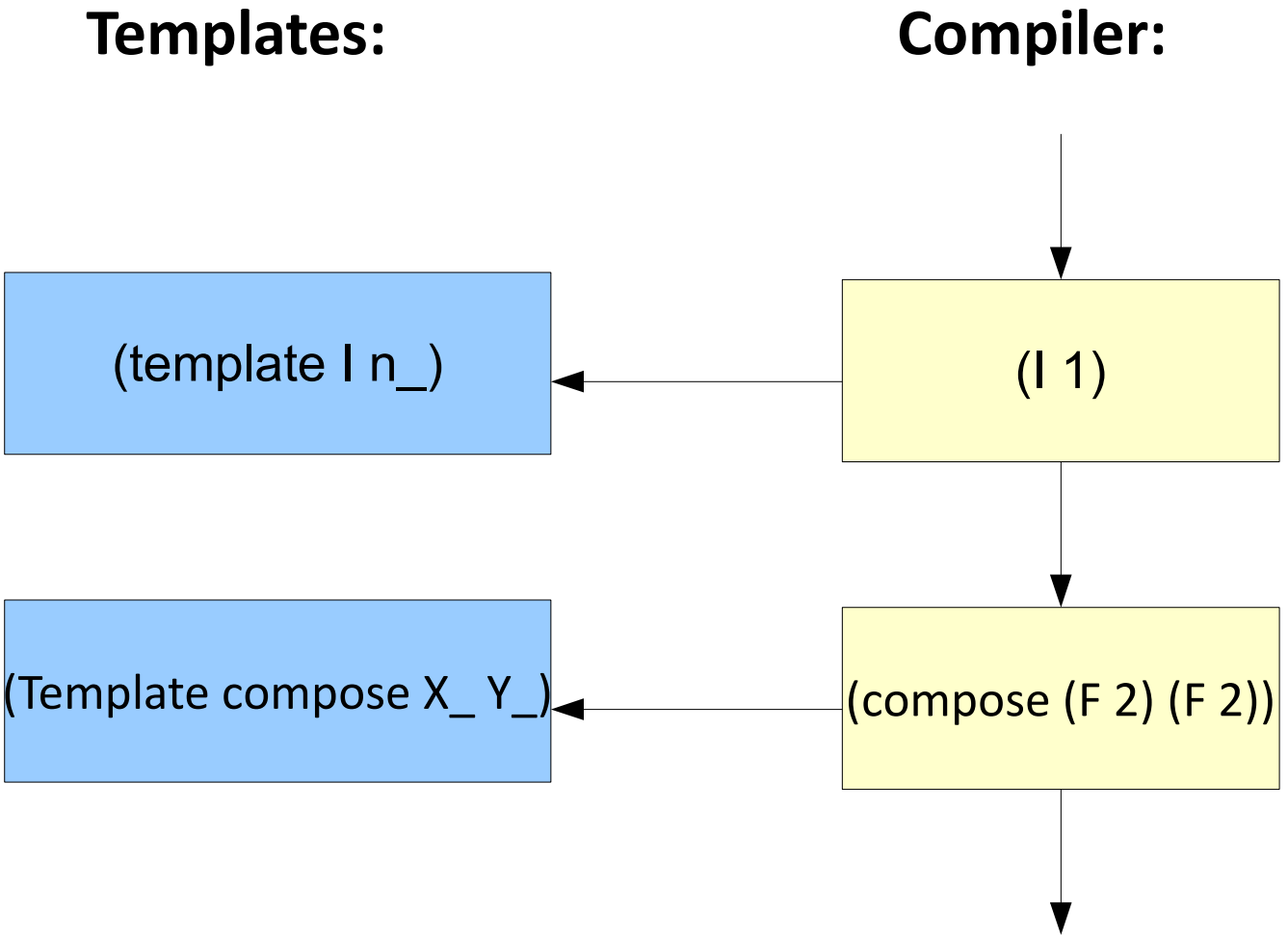
Templates

```
(template (F n_) [n_>0]
   (do $i=0, n_-1
          $out($i0)=0

           do $i1=0,n_-1
                $r0 = $i0 * $i1
                $f0 = W(n_ $r0)*$in($i1)
                $out($i0)=$out($i0)+$f0
           end
  end))
```

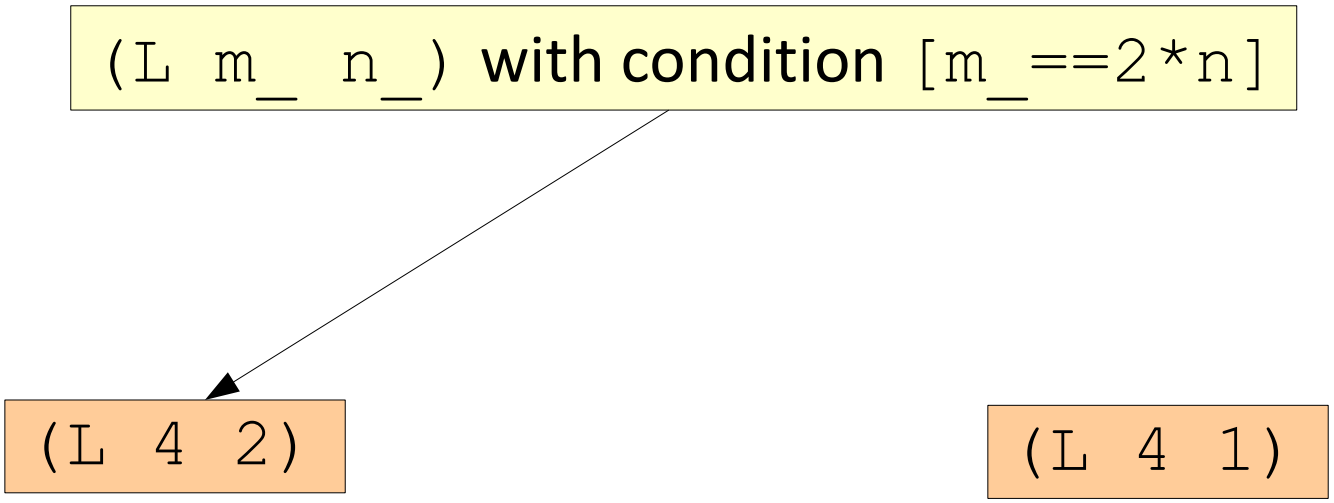
Templates

```
(template pattern condition i-code)
```

Templates


Templates:
Compiler:
(template I n_)
(I 1)
(Template compose X_ Y_)
(compose (F 2) (F 2))

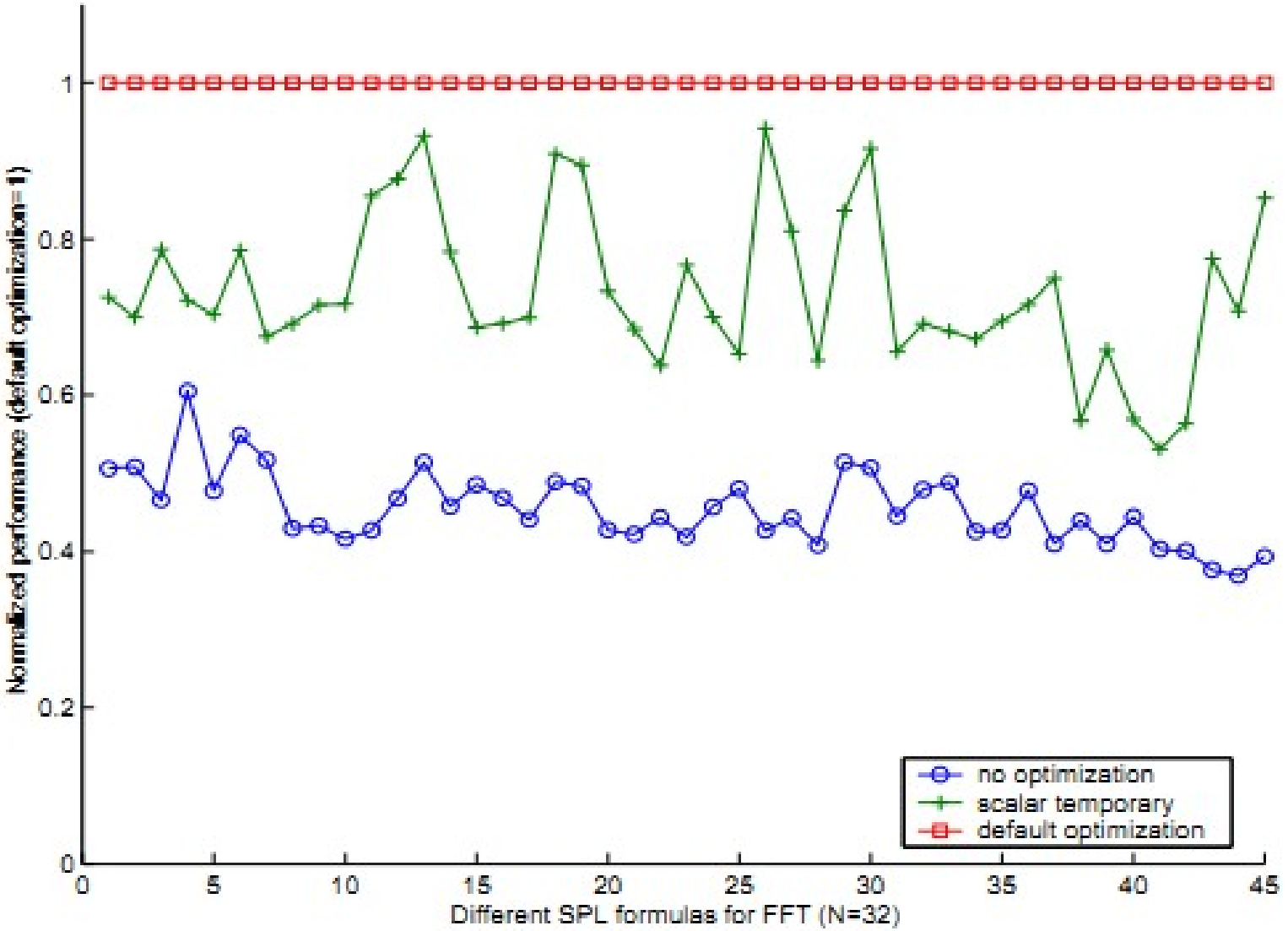
Templates



Templates

Example: A template for (F n)

```
(template (F n_) [n_>0]
   (do $i=0, n_-1
          $out($i0)=0

           do $i1=0,n_-1
                $r0 = $i0 * $i1
                $f0 = W(n_ $r0)*$in($i1)
                $out($i0)=$out($i0)+$f0
           end
  end))
```

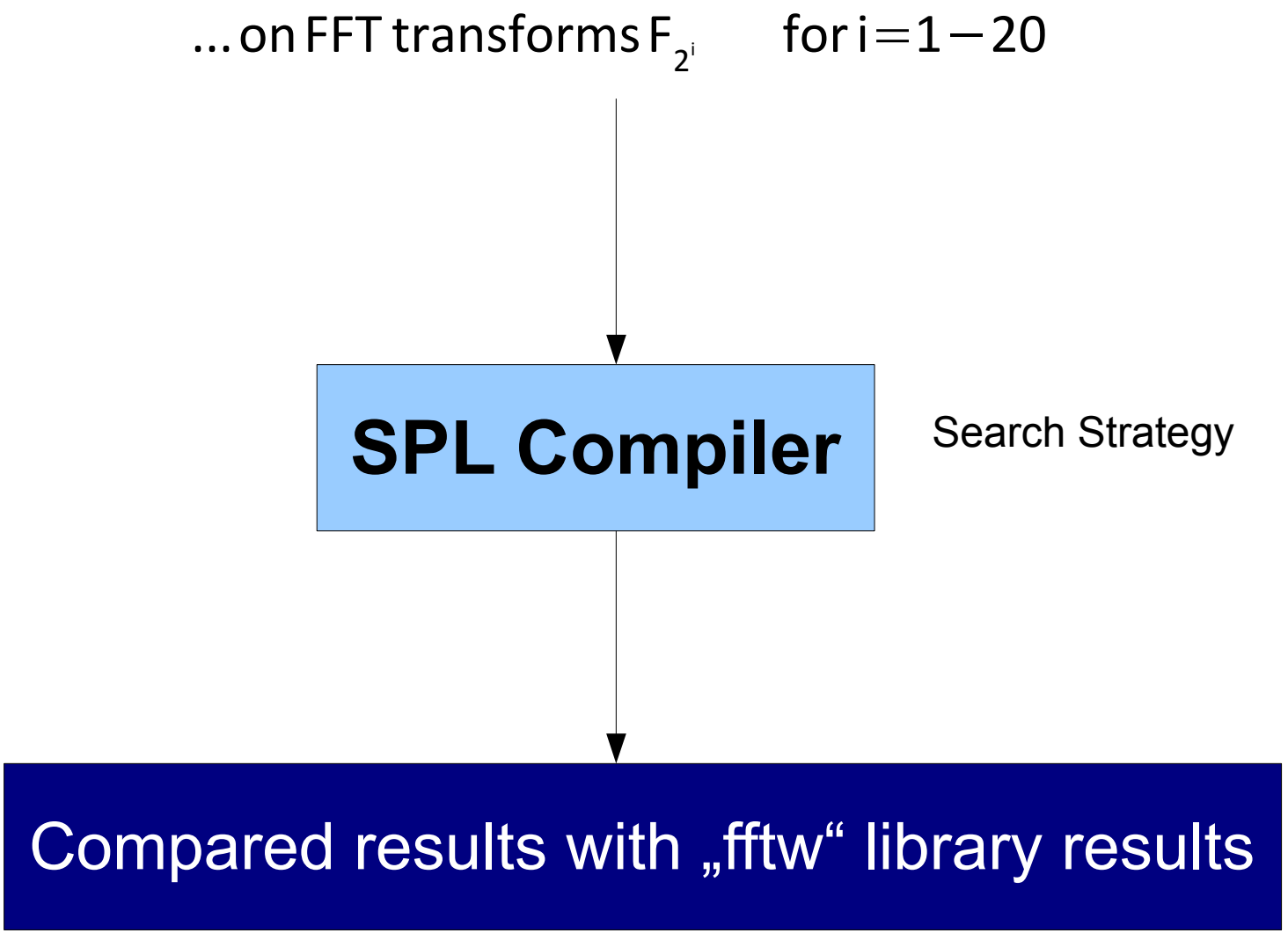


Picture:paper

Overview

- Motivation
- The language SPL
- The SPL compiler
- Experiments and results
- Conclusion

Experiments and results

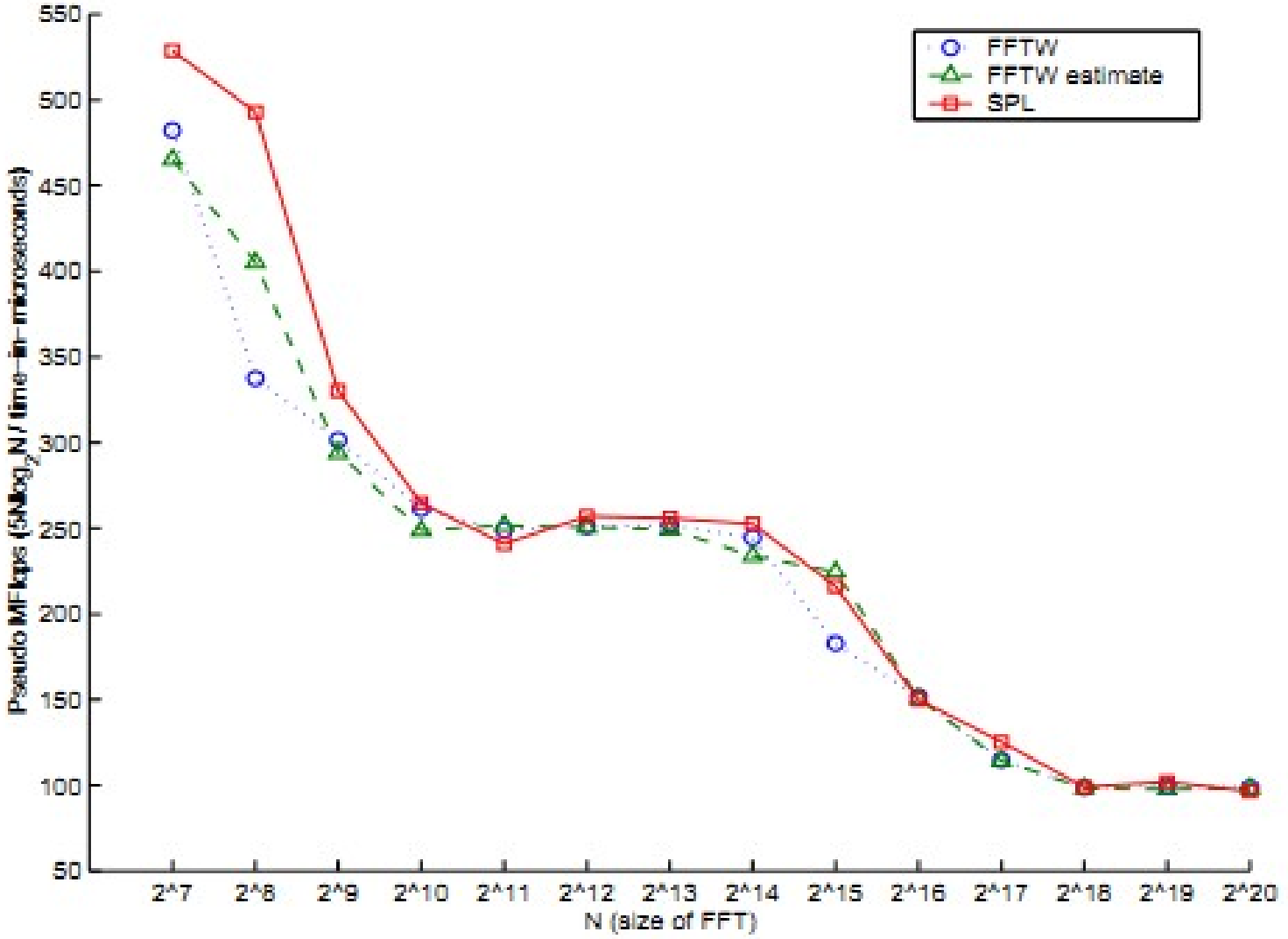


Search strategy

Dynamic programming over factorization by:

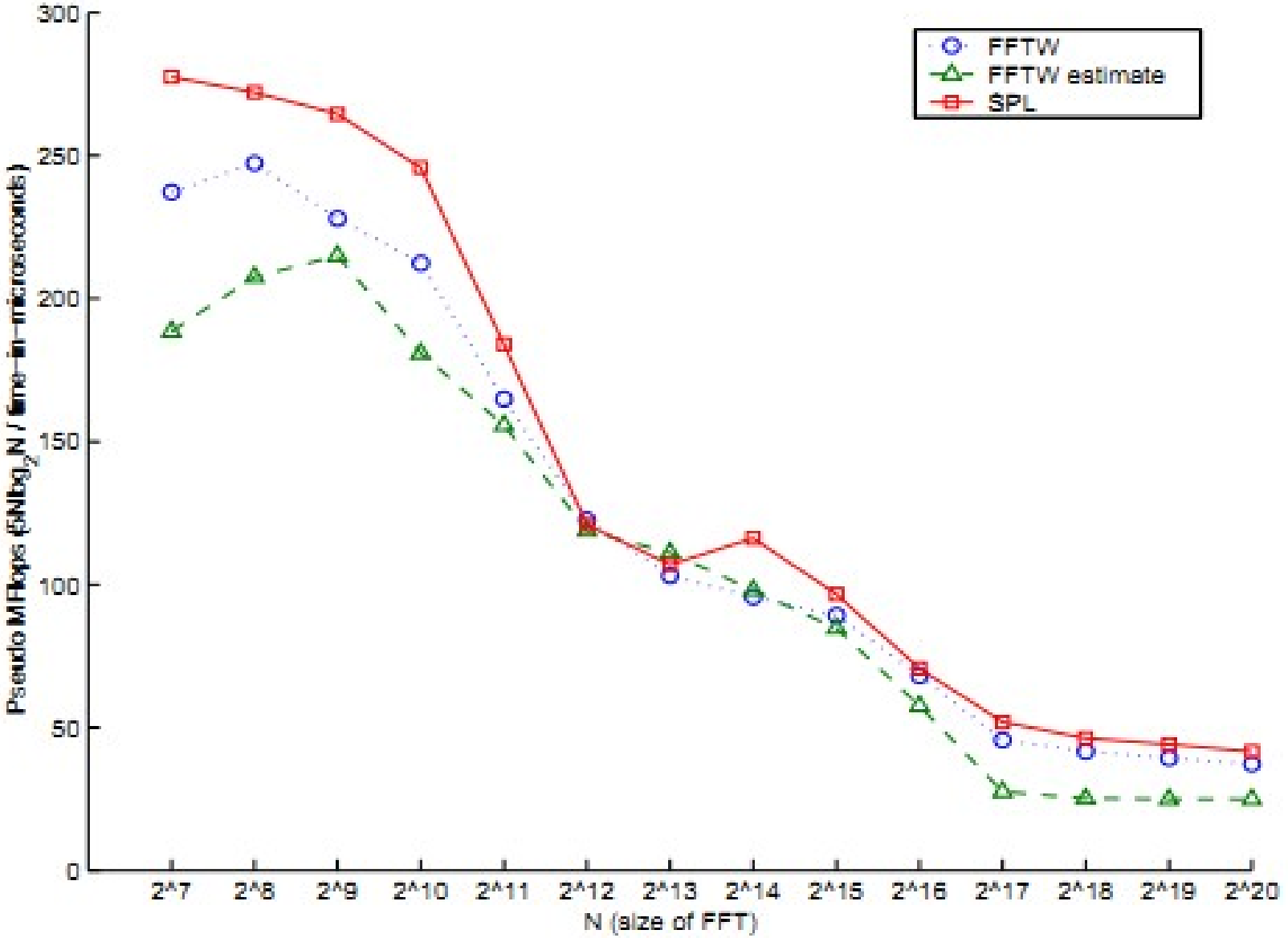
$$F_{rs} = \left(F_r \otimes I_s \right) T_s^{rs} \left(I_r \otimes F_s \right) L_r^{rs}$$

SPARC



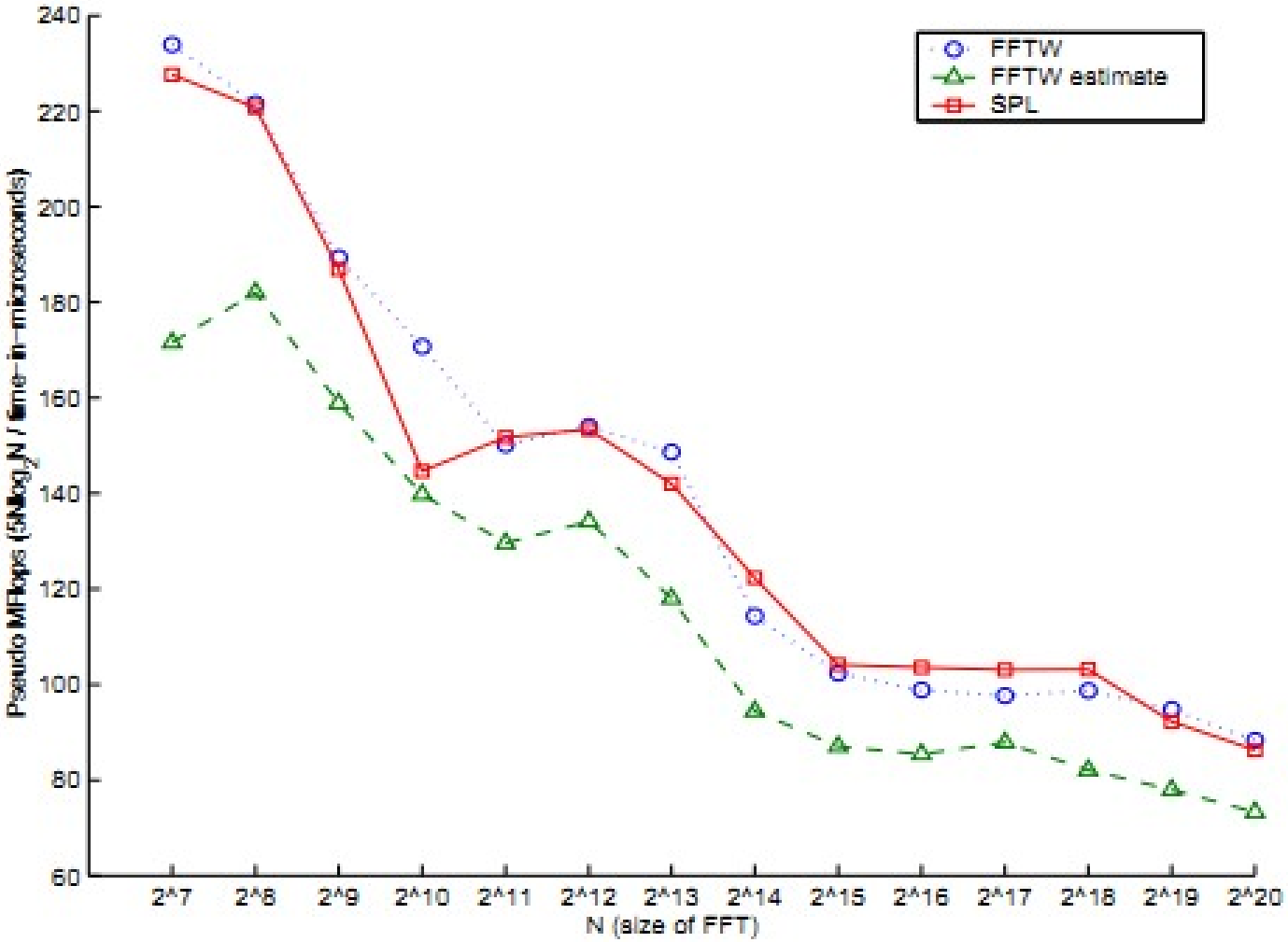
Picture:paper

MIPS



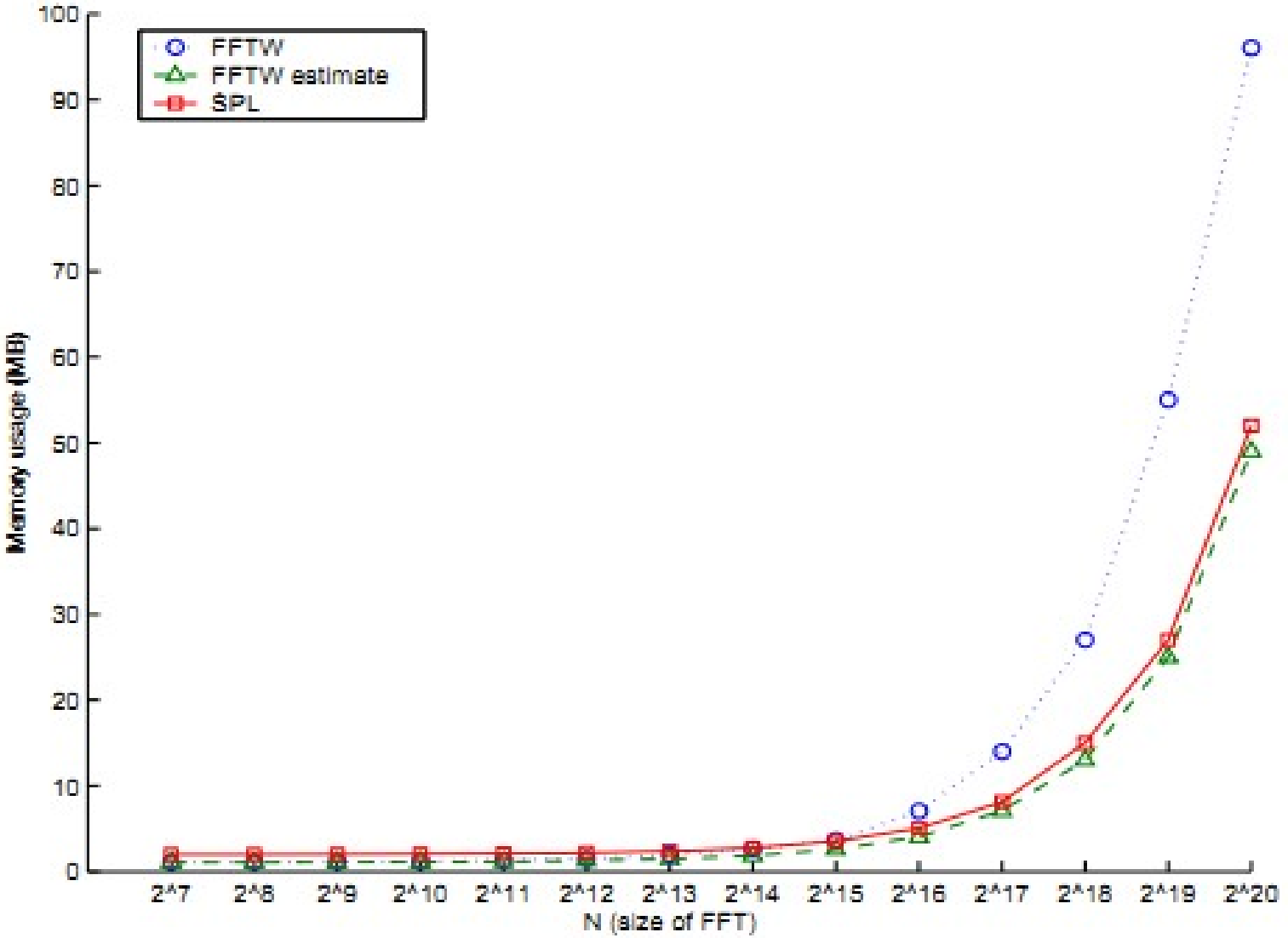
Picture:paper

Pentium II



Picture:paper

Memory usage



Conclusions

- Competitive with fftw if connected with search library
- More general, not only fft
- Results show power of domain-specific language and compilers
- Many more domain-specific projects in future