FORM and ParFORM: projects and outlooks

Mikhail Tentyukov

TTP Karlsruhe, BLTP JINR Dubna

J.A.M. Vermaseren

NIKHEF, Amsterdam

CALC-2006, 15-25 July, Dubna.

FORM and ParFORM

FORM

is a program by J. Vermaseren for symbolic manipulation of algebraic expressions specialized to handle very large expressions of millions of terms in an efficient and reliable way.

J.A.M. Vermaseren, arXiv:math-ph/0010025;

M. Tentyukov and J.A.M. Vermaseren, arXiv:cs.SC/0604052

http://www.nikhef.nl/˜form.

ParFORM

is a parallel version of FORM developed in Karlsruhe.

D. Fliegner *et al* arXiv:hep-ph/9906426;

D. Fliegner *et al* arXiv:hep-ph/0007221;

M. Tentyukov *et al* arXiv:cs.sc/0407066

ParFORM history

Profs.:
J. Kühn, H-M. Staudenmaier, M. Steinhauser.

D. Fliegner, A. Rétey, A. Onischenko, M. Frank,
M. Tentyukov.

D. Fliegner, A. Rétey: Working prototype 1998 – 2000.

A. Onischenko: FORM 3.1 features support, Large file (more than 2 Gbytes on 32-bit systems) support.

Recent development SFB/TR 9 Project A2:
J. Kühn, M. Steinhauser, M. Tentyukov

FORM setup

Module by module. Each module:

1. Compilation. 2. Generating. 3. Sorting:

Very long expressions!

Non-interactive interpreter.

User provides a program, Interpreter runs the program.

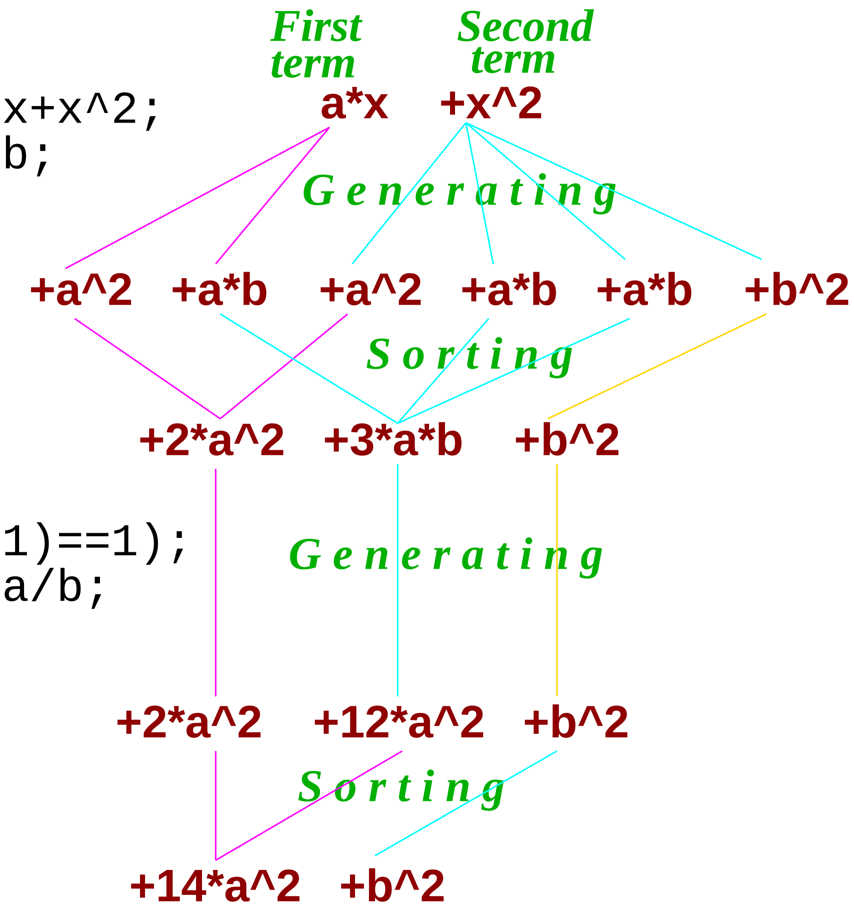
Modules are terminated by “dot” instructions.

First module

```
l expr = a*x+x^2;
id x = a + b;

.sort
```

Second module

```
if(count(b,1)==1);
multiply 4*a/b;
endif;

print;
.end
```



First term *Second term*

a*x +x^2

Generating

+a^2 +a*b +a^2 +a*b +a*b +b^2

Sorting

+2*a^2 +3*a*b +b^2

Generating

+2*a^2 +12*a^2 +b^2

Sorting

+14*a^2 +b^2

FORM setup

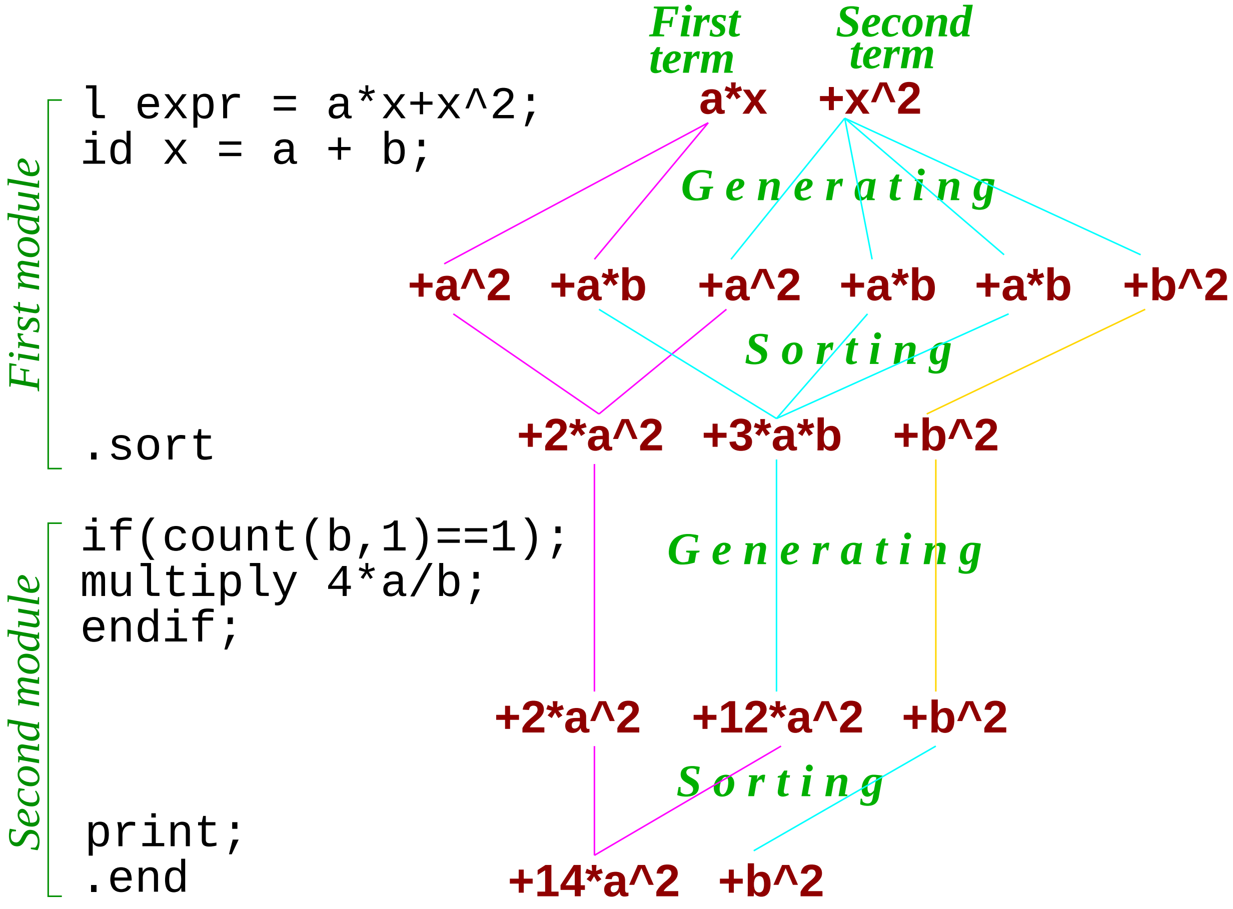
Module by module. Each module:

1. Compilation. 2. Generating. 3. Sorting:

Each module:

Definition of new expressions, algebraic instructions, output instructions.

Expressions remain active through many modules.



First module

```
l expr = a*x+x^2;
id x = a + b;




.sort
```

Second module

```
if(count(b,1)==1);
multiply 4*a/b;
endif;



print;
.end
```

First term *Second term*

a*x **+x^2**

Generating

+a^2 +a*b +a^2 +a*b +a*b +b^2

Sorting

+2*a^2 +3*a*b +b^2

Generating

+2*a^2 +12*a^2 +b^2

Sorting

+14*a^2 +b^2

FORM setup

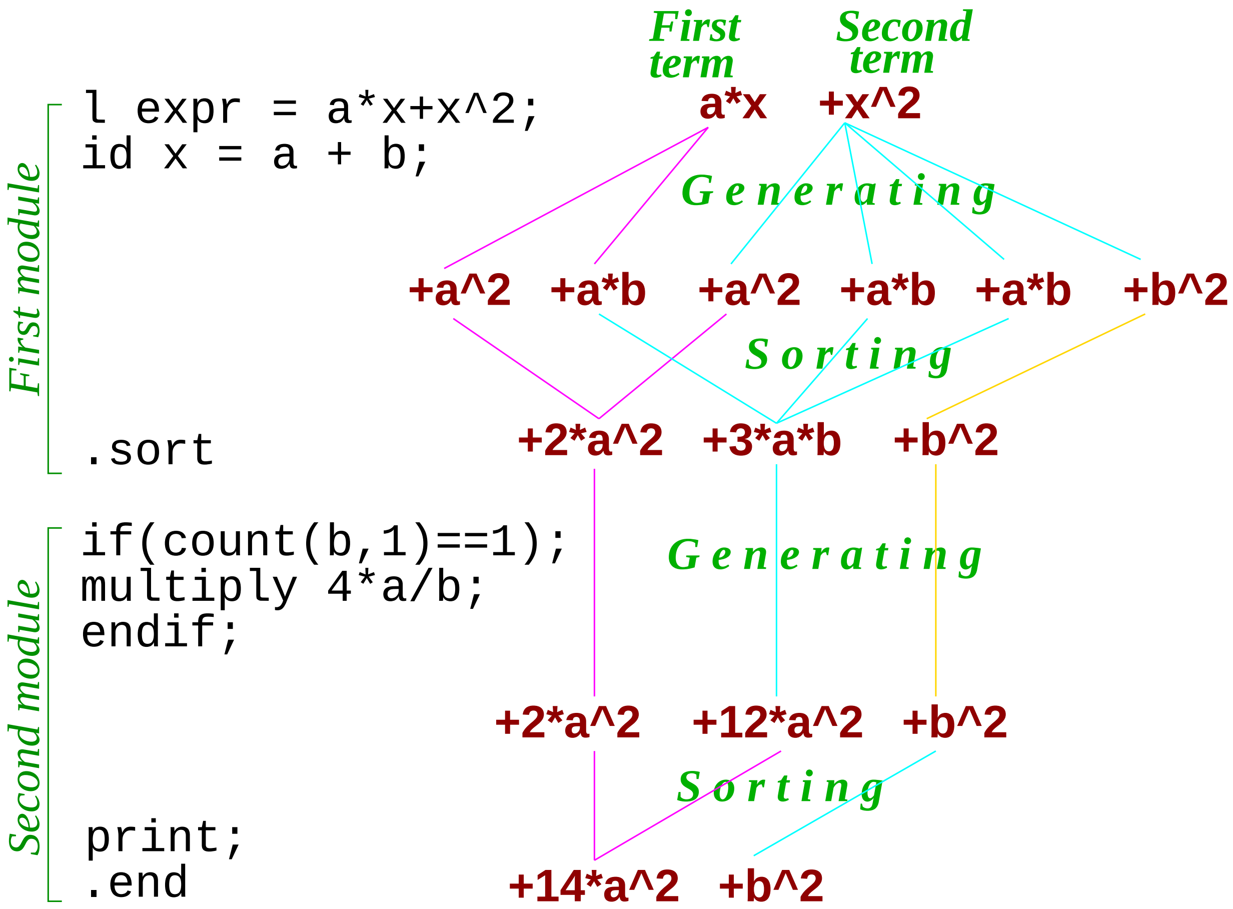
Module by module. Each module:

1. Compilation. 2. Generating. 3. Sorting:

Compilation: Input translated into internal representation.

Generating: Algebraic instructions executed for each term.

Sorting: Generated terms sorted, equivalent terms are summed up.



First module

```
l expr = a*x+x^2;
id x = a + b;
.sort
```

Second module

```
if(count(b,1)==1);
multiply 4*a/b;
endif;
print;
.end
```

First term a*x

Second term +x^2

Generating

+a^2 +a*b +a^2 +a*b +a*b +b^2

Sorting

+2*a^2 +3*a*b +b^2

Generating

+2*a^2 +12*a^2 +b^2

Sorting

+14*a^2 +b^2

FORM setup

Module by module. Each module:

1. Compilation. 2. Generating. 3. Sorting:

specific feature:

only local operations on single term:

id x = a + b;

Non-local operations are not allowed:

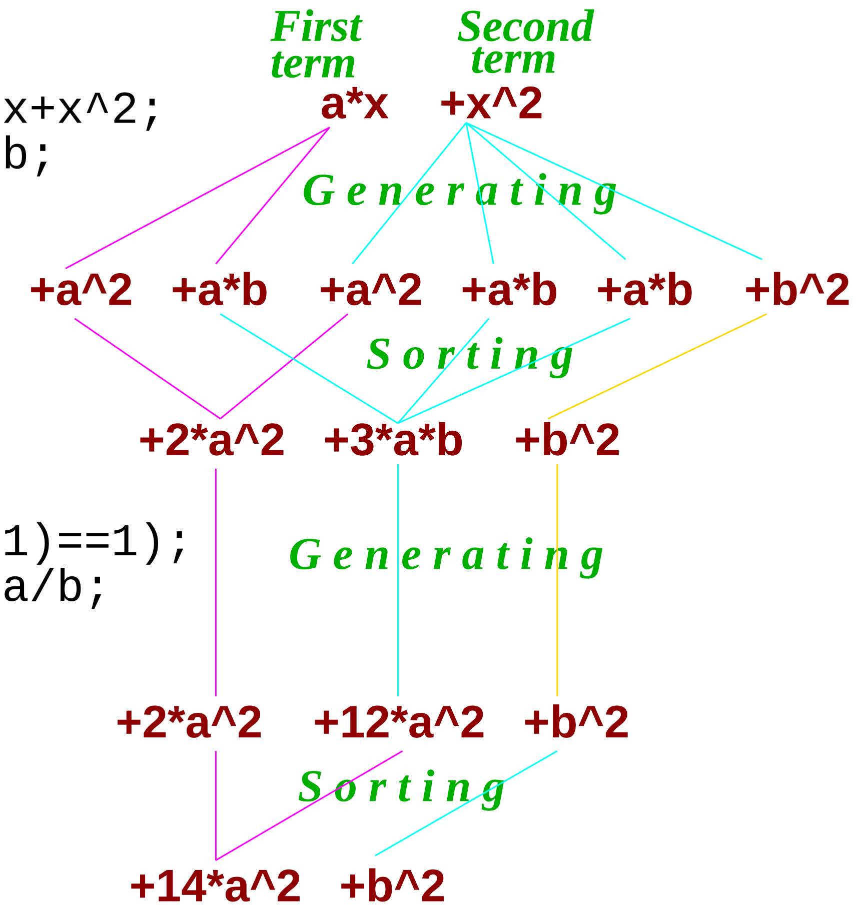
~~id a + b = x~~;

First module

```
l expr = a*x+x^2;
id x = a + b;




.sort
```

Second module

```
if(count(b,1)==1);
multiply 4*a/b;
endif;




print;
.end
```



First term

Second term

a*x +x^2

Generating

+a^2 +a*b +a^2 +a*b +a*b +b^2

Sorting

+2*a^2 +3*a*b +b^2

Generating

+2*a^2 +12*a^2 +b^2

Sorting

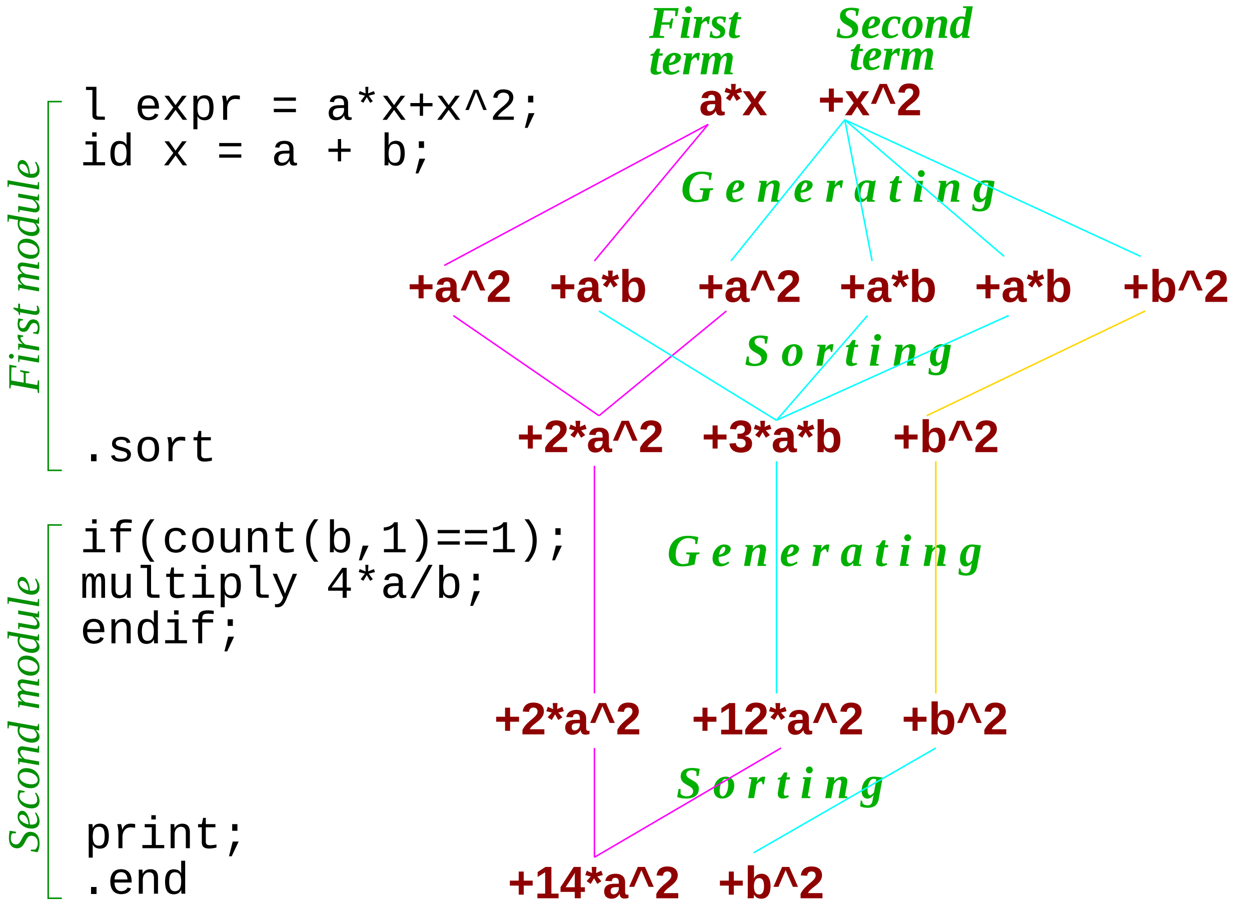
+14*a^2 +b^2

FORM setup

Module by module. Each module:

1. Compilation. 2. Generating. 3. Sorting:

Locality Principle: All *explicit* algebraic operations are local. Non-local operations are allowed only *implicitly* in the sorting procedure at the end of the modules and in some other special cases.



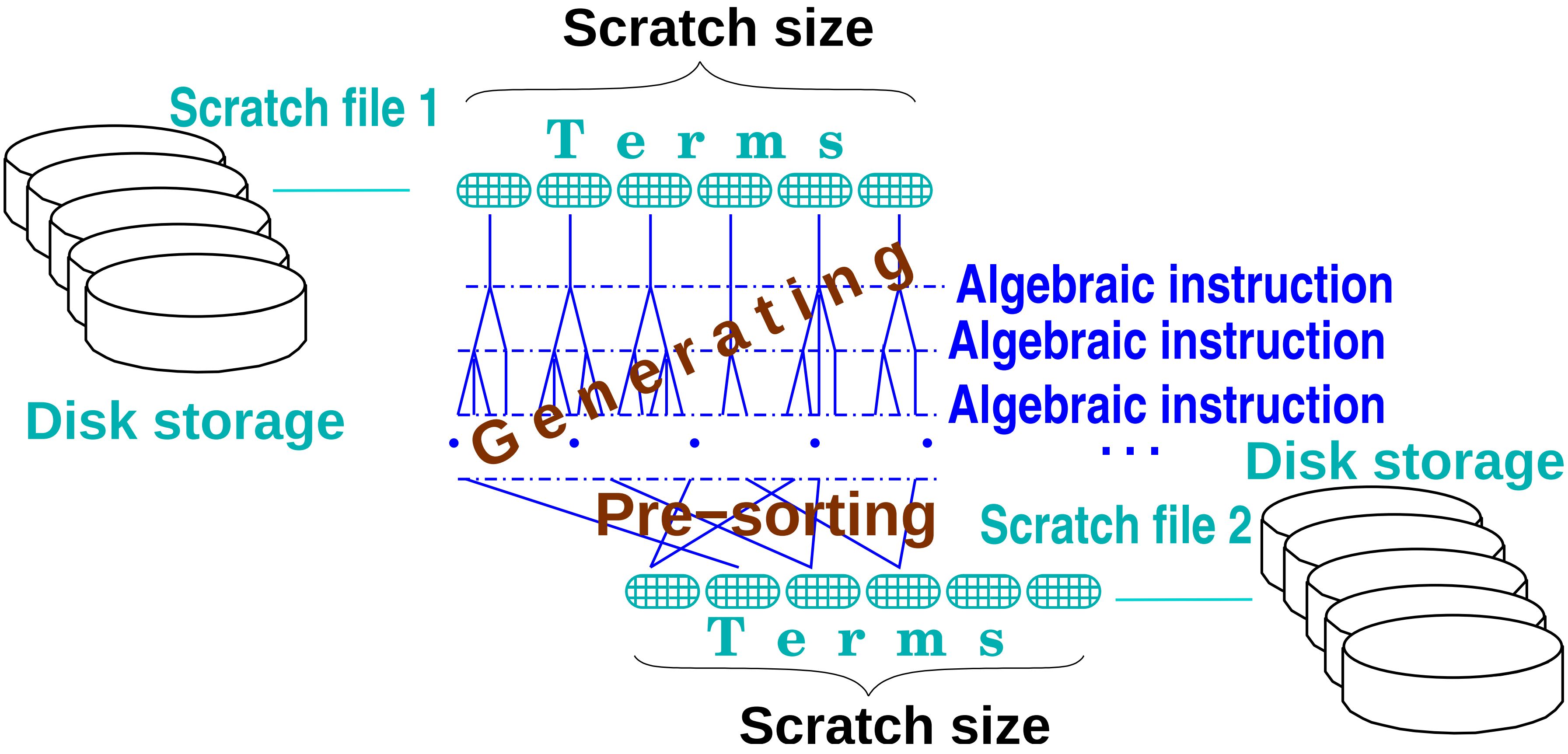
```
l expr = a*x+x^2;
id x = a + b;

.sort
```

```
if(count(b,1)==1);
multiply 4*a/b;
endif;

print;
.end
```

Main FORM feature

Locality Principle → Expressions as “streams” of terms

Expressions bigger than the memory (RAM) available!



Main FORM feature

Locality Principle → Expressions as “streams” of terms

Expressions bigger than the memory (RAM) available!

⇓

“Space complexity” is not a problem anymore!

Disk storage is cheap, and we are always on the top!

The only restriction is the time.

Space Complexity ⟹ Time Complexity

Main FORM feature

Locality Principle → Expressions as “streams” of terms

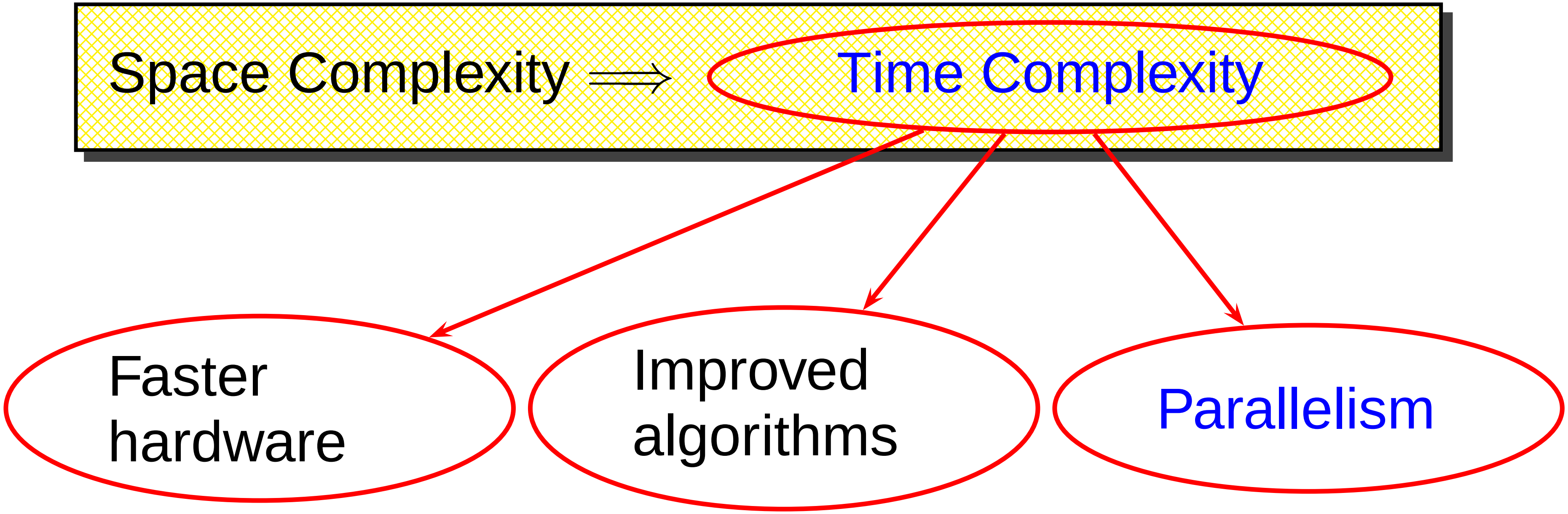
Expressions bigger than the memory (RAM) available!

⇓

“Space complexity” is not a problem anymore!

Disk storage is cheap, and we are always on the top!

The only restriction is the time.



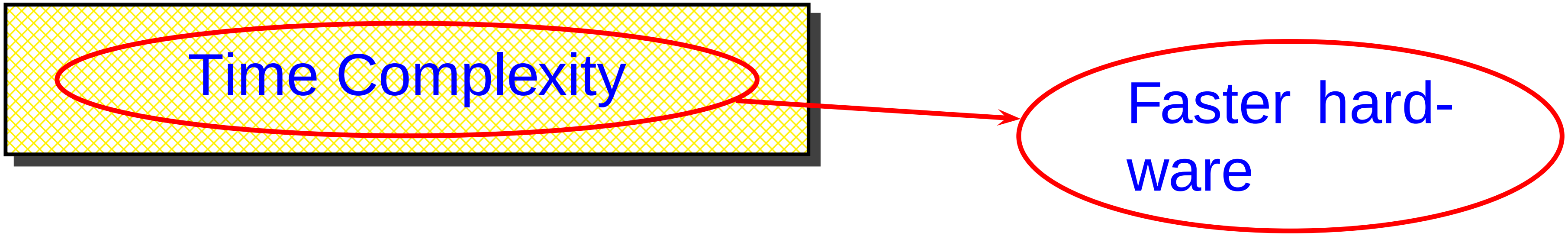
Space Complexity $\Longrightarrow$ Time Complexity

Faster hardware

Improved algorithms

Parallelism

The Time Complexity



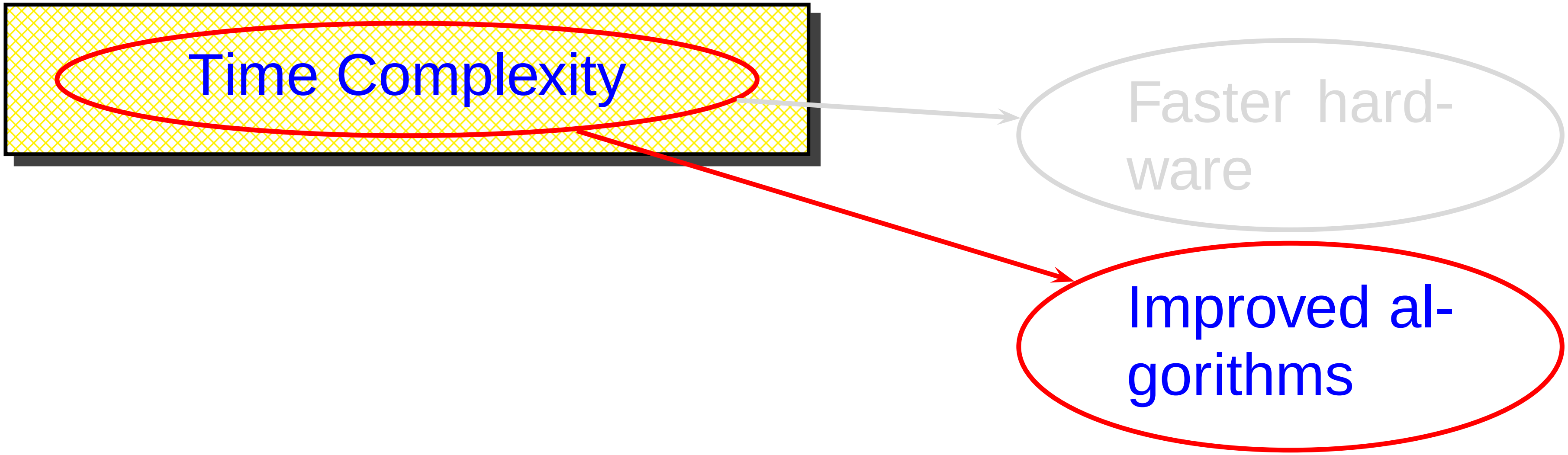
Time Complexity

Faster hardware

Neither Gigaflops nor Top500!
This means GigaHertz and RPMs!

The specifics of the moment:
The fastest computers are the cheapest computers!

The Time Complexity



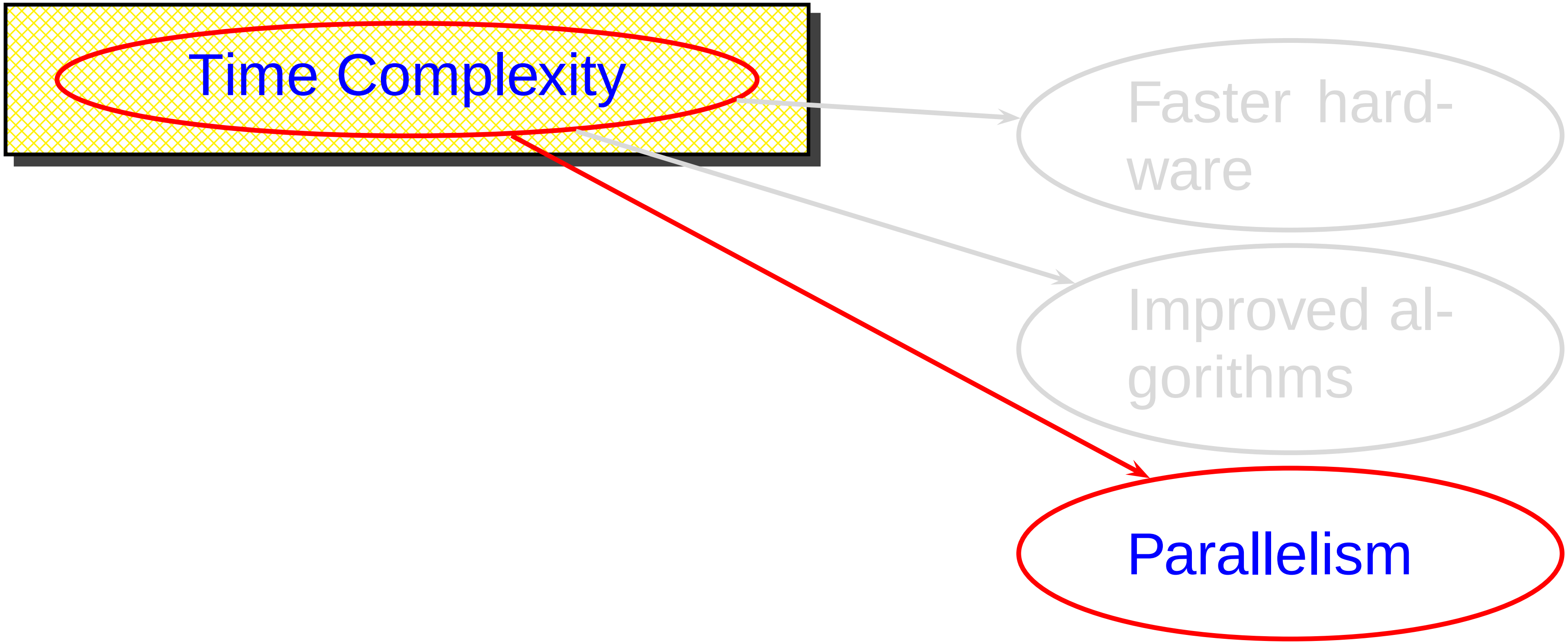
Time Complexity

Faster hardware

Improved algorithms

Sometimes better algorithm is impossible!
The algorithm elaboration requires the researcher time!
Specific of the researchers programming:
The most important is the *Wall-Clock* time!

The Time Complexity



In practice, the only way to improve the performance. But parallelism does not come for free!

Parallelism

Not every problem is parallelizable.
Parallelizable problem: multiplication of two matrices.
Non-parallelizable problem: calculation of the Fibonacci series (1,1,2,3,5,. . .) by means of the recurrence formula $F(k+2) = F(k+1) + F(k)$.

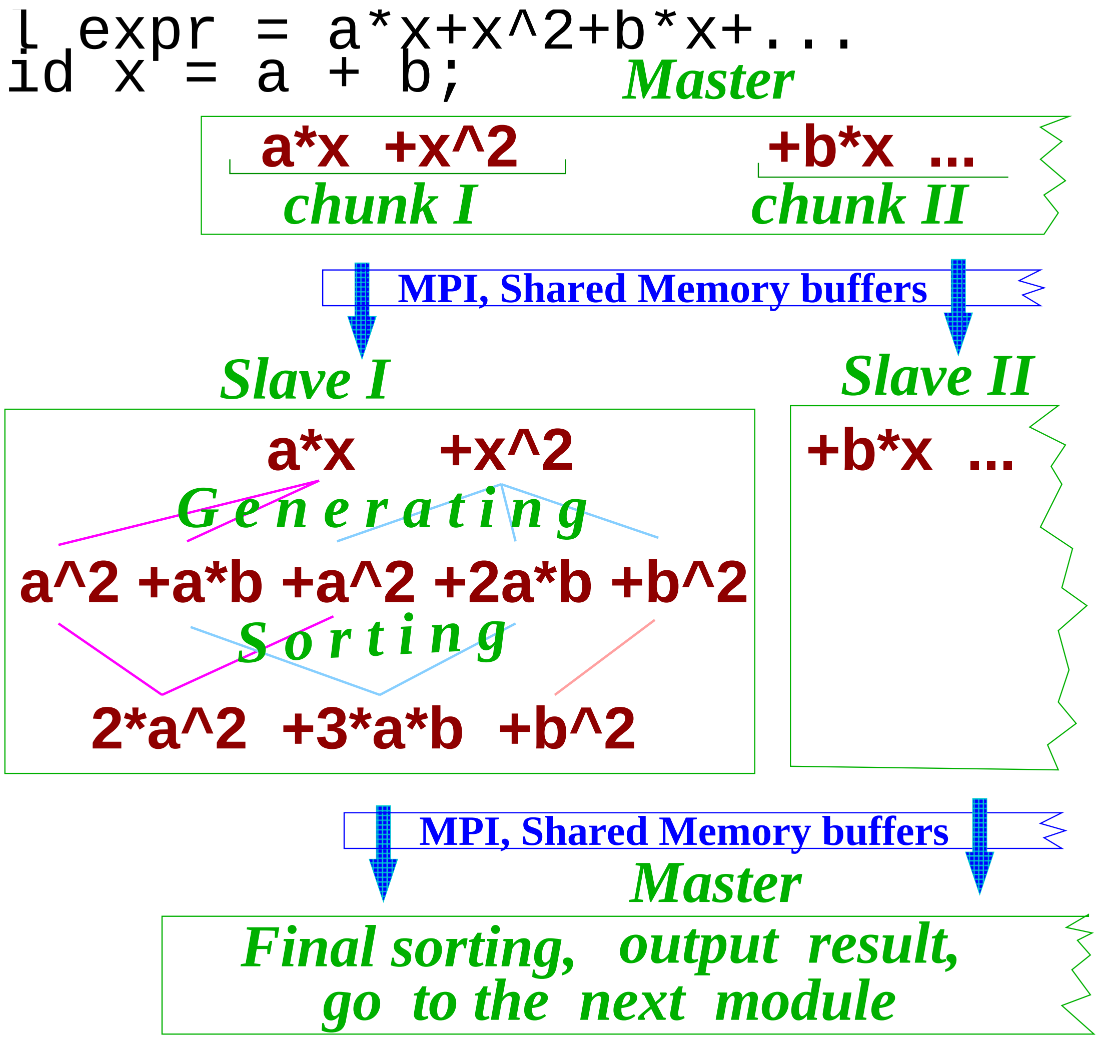
- Reducing the problem to several independent subproblems evaluatable in parallel – *requires the reseacher time*.
  - Programming problems.
  - Managing problems.

  Remember, the most important is the *Wall-Clock* time!
- Completely automatic parallelization. *Let the computer do the job!*

The concept of FORM parallelization

The **Locality Principle**, again!

The master splits the input expression in chunks. Each chunk is sent to one of the slaves. Slaves perform generating/sorting and send the result back.



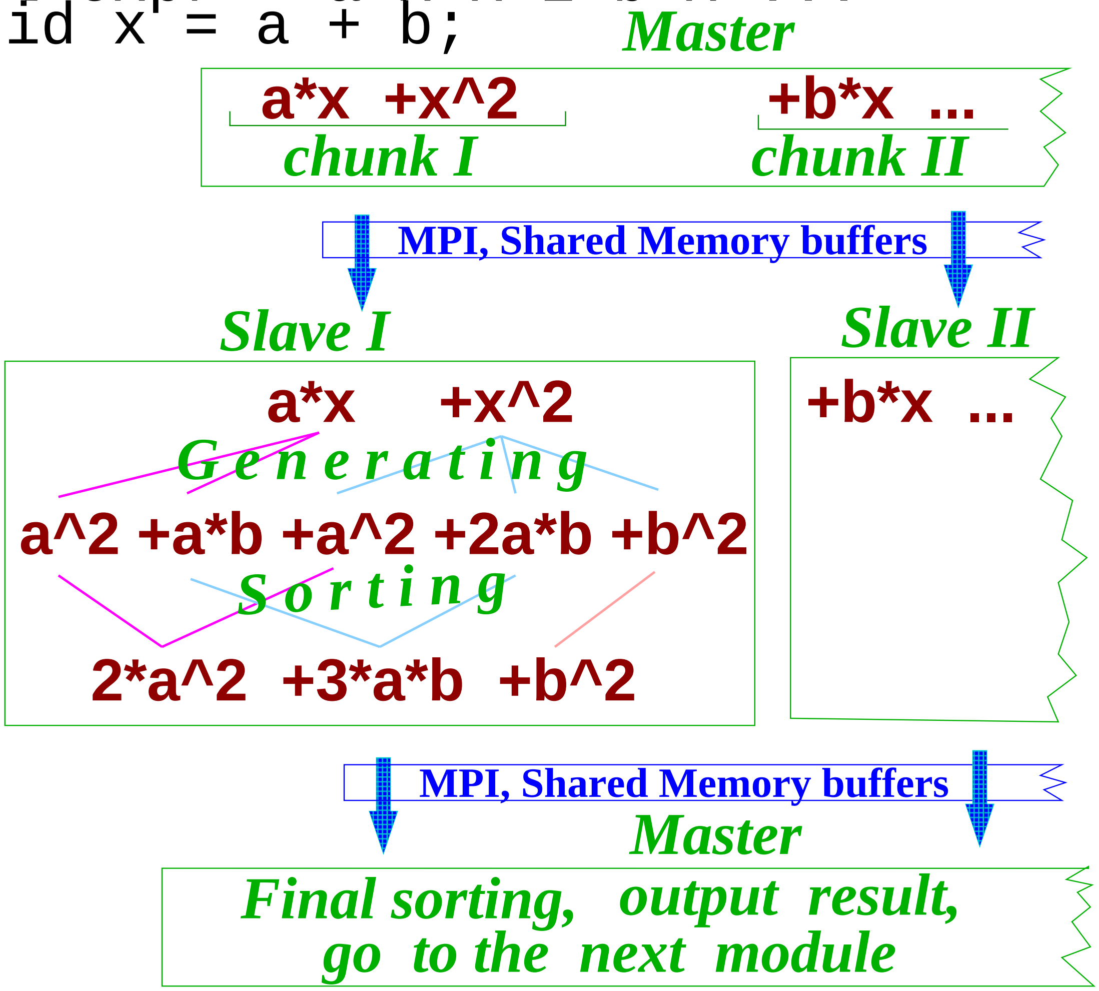
```
L expr = a*x+x^2+b*x+...
id x = a + b;
```

The concept of FORM parallelization

Transparently for the user! The same FORM program.

```
L expr = a*x+x^2+b*x+...
id x = a + b;
```

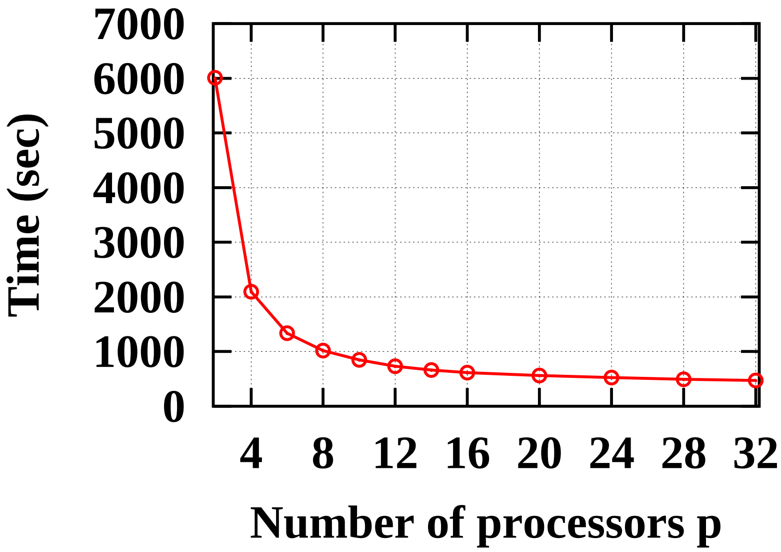
The Master, and each slave are INDEPENDENT processes. Communication with MPI or (recently) by means of the explicitely shared memory buffers. Formerly was also PVM – cancelled!

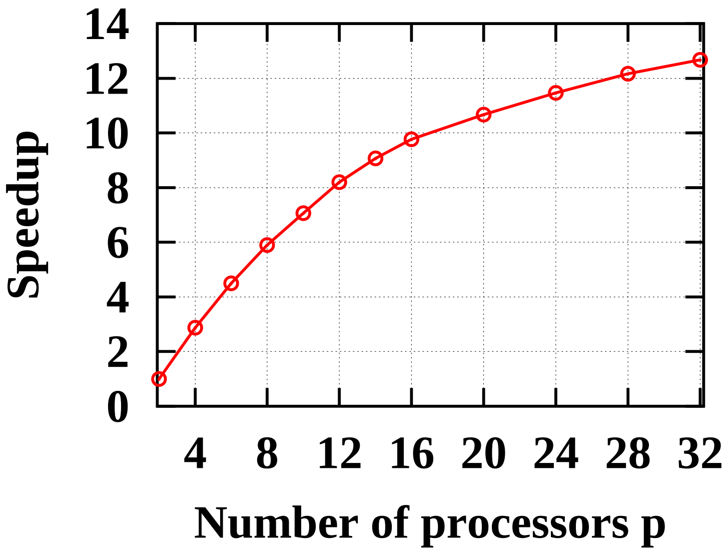


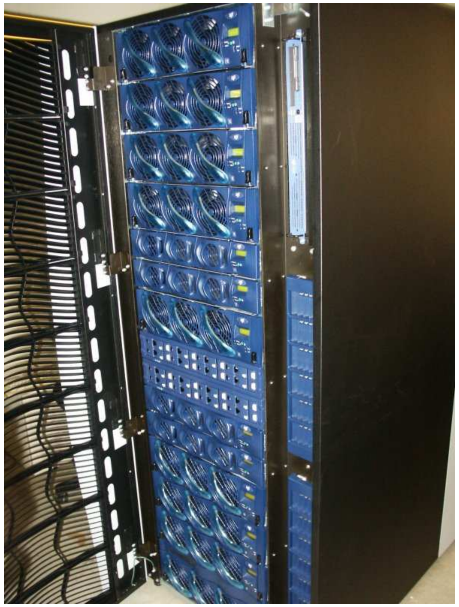
MPI

- Advantages of MPI
  - Communication via MPI is UNIVERSAL: Programmer- transparent w.r.t. networking and SMP – Symmetric Multi Processing.
  - This is the Industrial Standard upported many vendors.
- Disadvantages of MPI
  - The Message Passing concept itself may lead to overhead on SMP;
  - No rigorous standard for the MPI libraries $\longrightarrow$ problems with installation;
  - Most efficient MPI implementations are commercial, no sources, problem for debugging.

SMP computer (SM)







32 Itanium II 1300 MGz SGI Altix 3700:

Reduction in run time

m_s	24 months	$\longrightarrow$	4 months
$\Gamma(H \to b\bar{b})$	90 months	$\longrightarrow$	15 months
β^{qQED} (5 loops)	56 months	$\longrightarrow$	7 months

Clusters vs. SMP

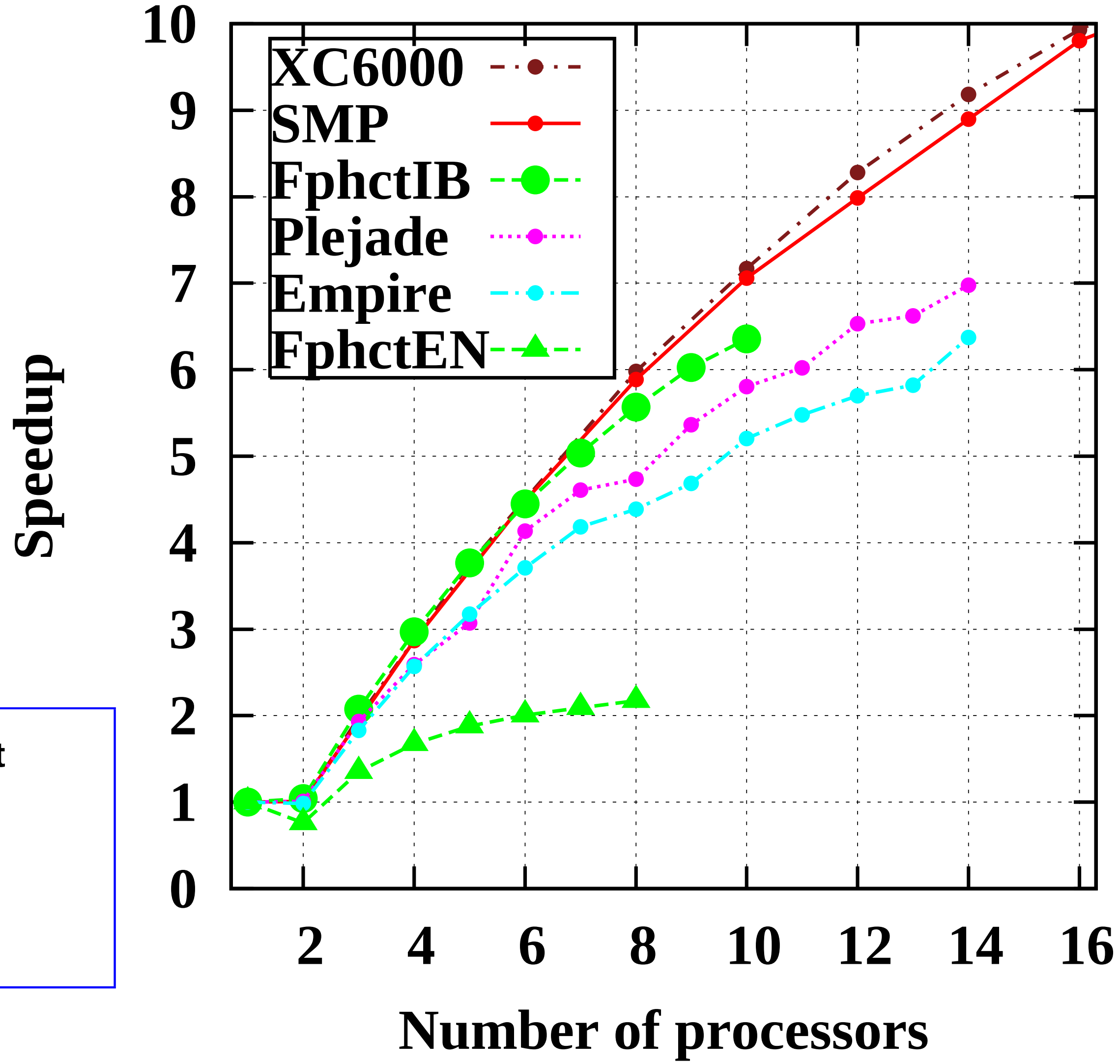
Interconnections:

- ● RZ-KA; QsNet
- ● TTP; SMP (NUMAlink)
- ● TTP; Infiniband
- ● Zeuthen; Infiniband
- ● TTP; Gigabit Ethernet
- ▲ TTP; Fast Ethernet

ParFORM is available on request

Current installations:

Karlsruhe, DESY, Zeuthen,
Bielefeld, Edmonton



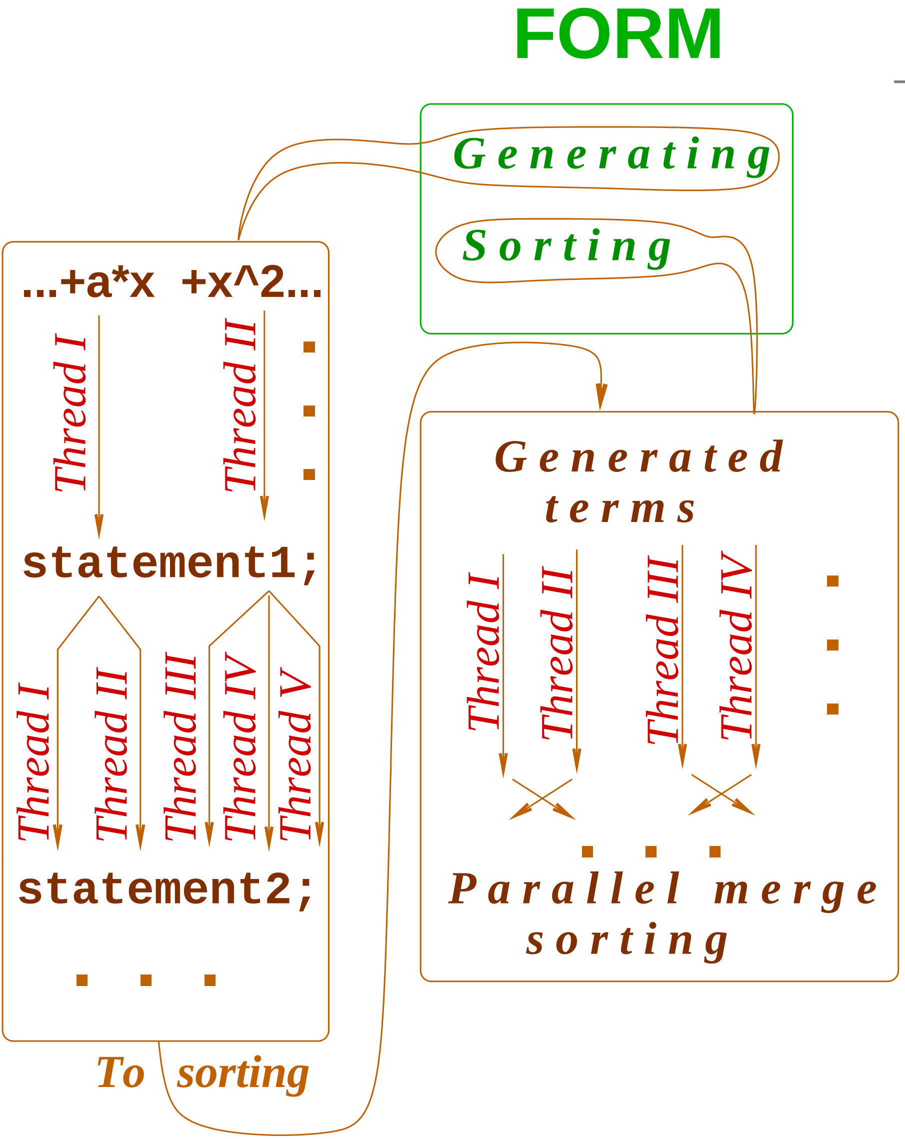
Fine grained model

Advantages:

- perfect load balancing;
- minimal overhead.

Disadvantage:

- complicated implementation;
- problems with memory affininy and cache coherency.

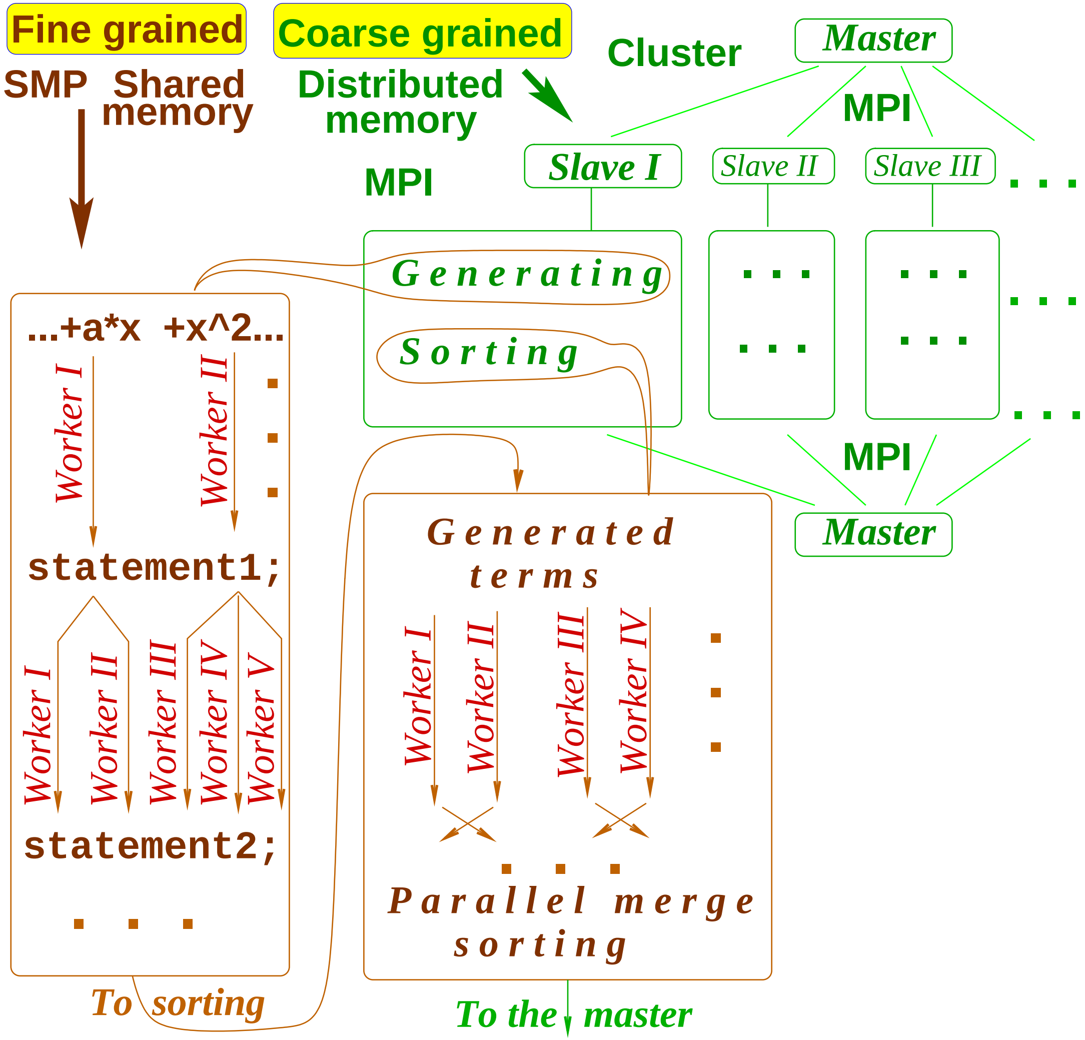


POSIX threads-based model (TFORM)

(The same structure as ParFORM!)

Combined model

Suppose, we have several SMP nodes connected by high-speed network. Use message passing library for inter-node communications and “fine grained” model for SMP parallelization.



FORM restrictions and extensions

FORM is less user-friendly than popular CAS like Mathematica.

- Algebraic imperatives acts on individual terms, not complete expressions.
- All the control flow instructions are executed for each term. No “real” control flow at the algebraic processor level! Only the preprocessor allows to control the flow!
- That is why FORM is often used in combination with other software systems which provide a more fluent control flow.

We need:

Effective mechanism
for communication with external programs.

FORM restrictions and extensions

The Locality Principle restricts possible algorithms!

- Impossible to avoid non-local operations completely, e.g. FORM supports implicitly the sort operation, bracketing and some others.
- Sometimes non-local operations are crucial, e.g. GCD in Laporta, or Gröbner basis. Shold we extend FORM?
- Important to keep the Locality Princilple as strong as possible!
- The idea: Delegate non-local operatrions to other CAS (less efficient but less restricted).

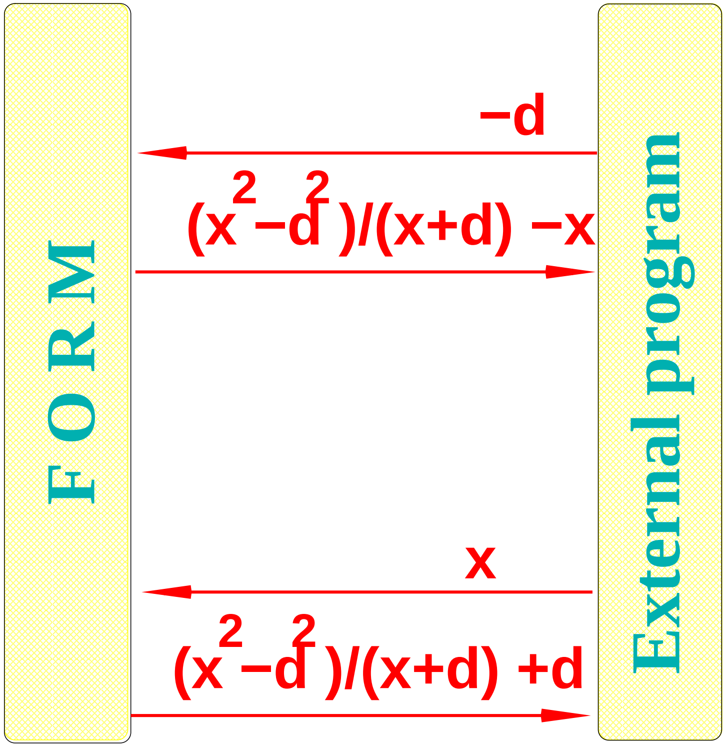
We need:

Effective mechanism
for communication with external programs.

Communication with external program

External program can be used as a “black box”, in the spirit of a *component model*:

```
s a,b;
#external cat
#toexternal "(a+b)\n\n"
l e=
#fromexternal
;
print;
.end
.  .  .
   e =
      b + a;
```



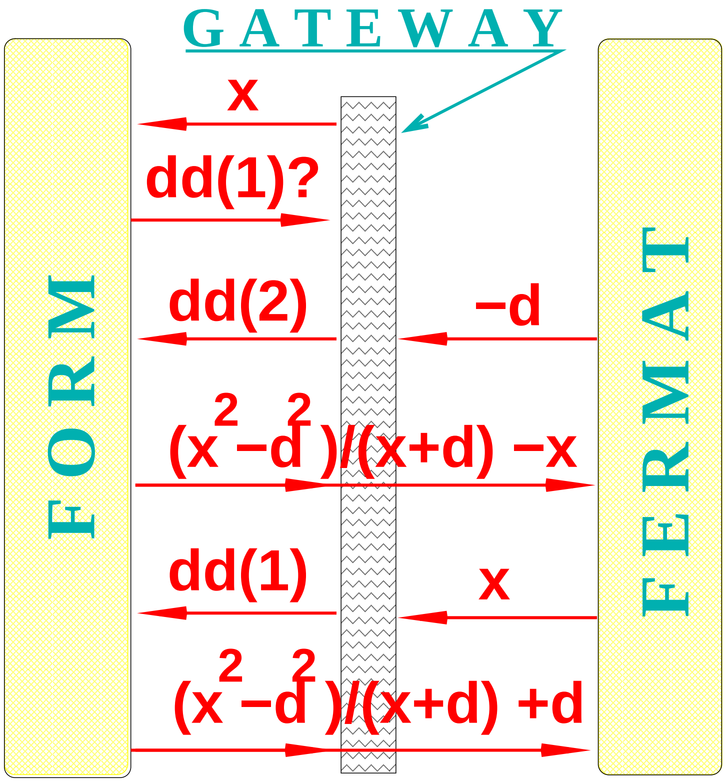
STURMAN by Ch. Sturm

Laporta implementation in FORM3, ratio polynomials treated by Fermat (by R.Lews, http://www.bway.net/~lewis)

Fermat:

- Approx. 20 times faster than the Mathematica “Together[]”
- Able to handle very large expressions!

Gateway program is used also to hide large expressions from FORM.



Upcoming FORM 3.2

Sources of FORM, ParFORM and TFORM share the same CVS repository.

Current version is FORM 3.1. Upcoming FORM 3.2, see J.A.M. Vermaseren and M. Tentyukov, “What is new in FORM”, Proceedings LandL2006.

- Various commands for output reduction.
- GZip compression.
- External channels and supporting libraries and templates, see arXiv:cs.SC/0604052
- ParFORM
- TFORM