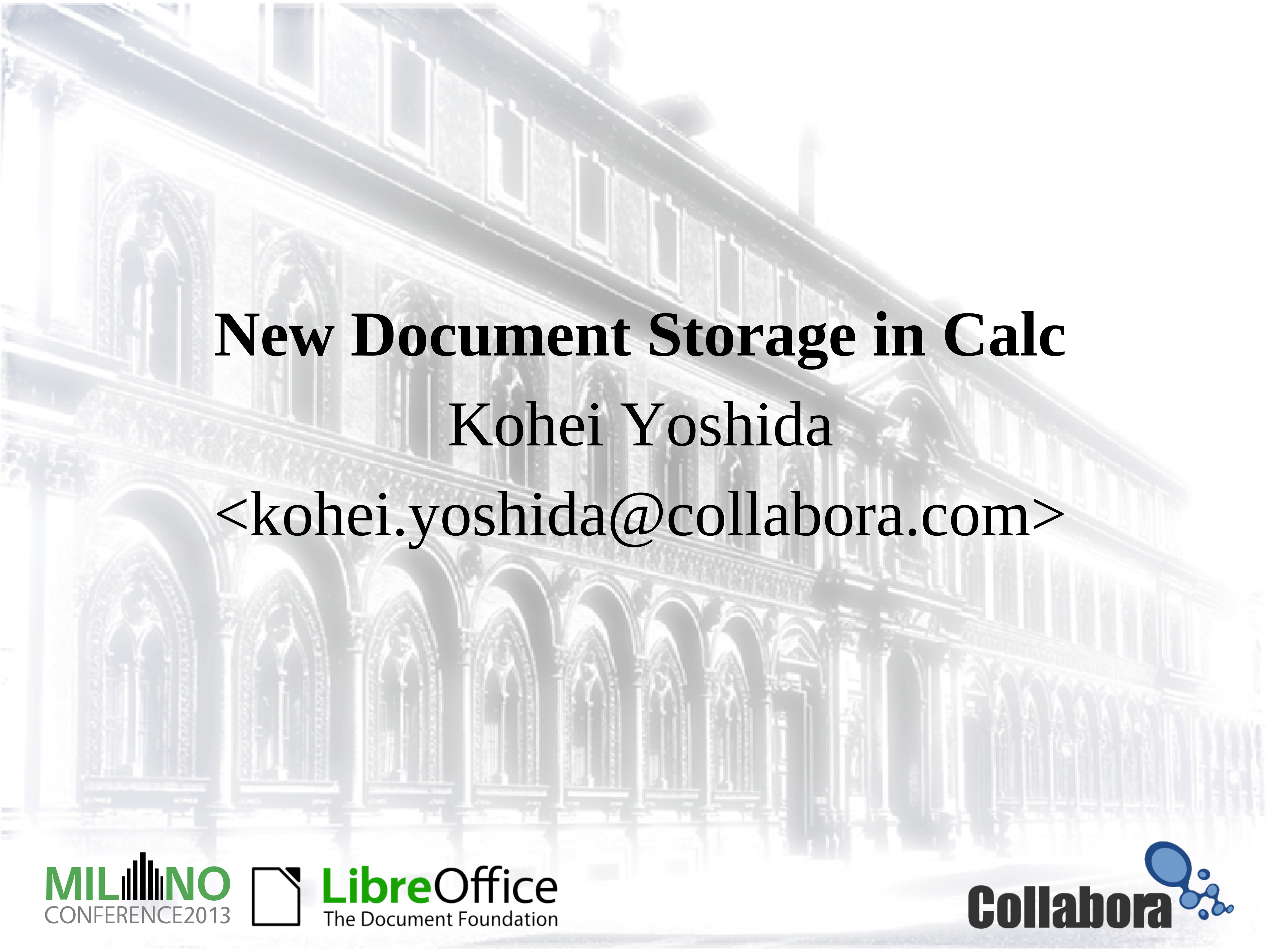


New Document Storage in Calc

Kohei Yoshida

<kohei.yoshida@collabora.com>

MILANO CONFERENCE2013

LibreOffice The Document Foundation

Collabora

Topics

- **New document storage**
  - Difference from old storage
  - mdds::multi_type_vector
- **Formula groups**
- **OpenCL interpreter**

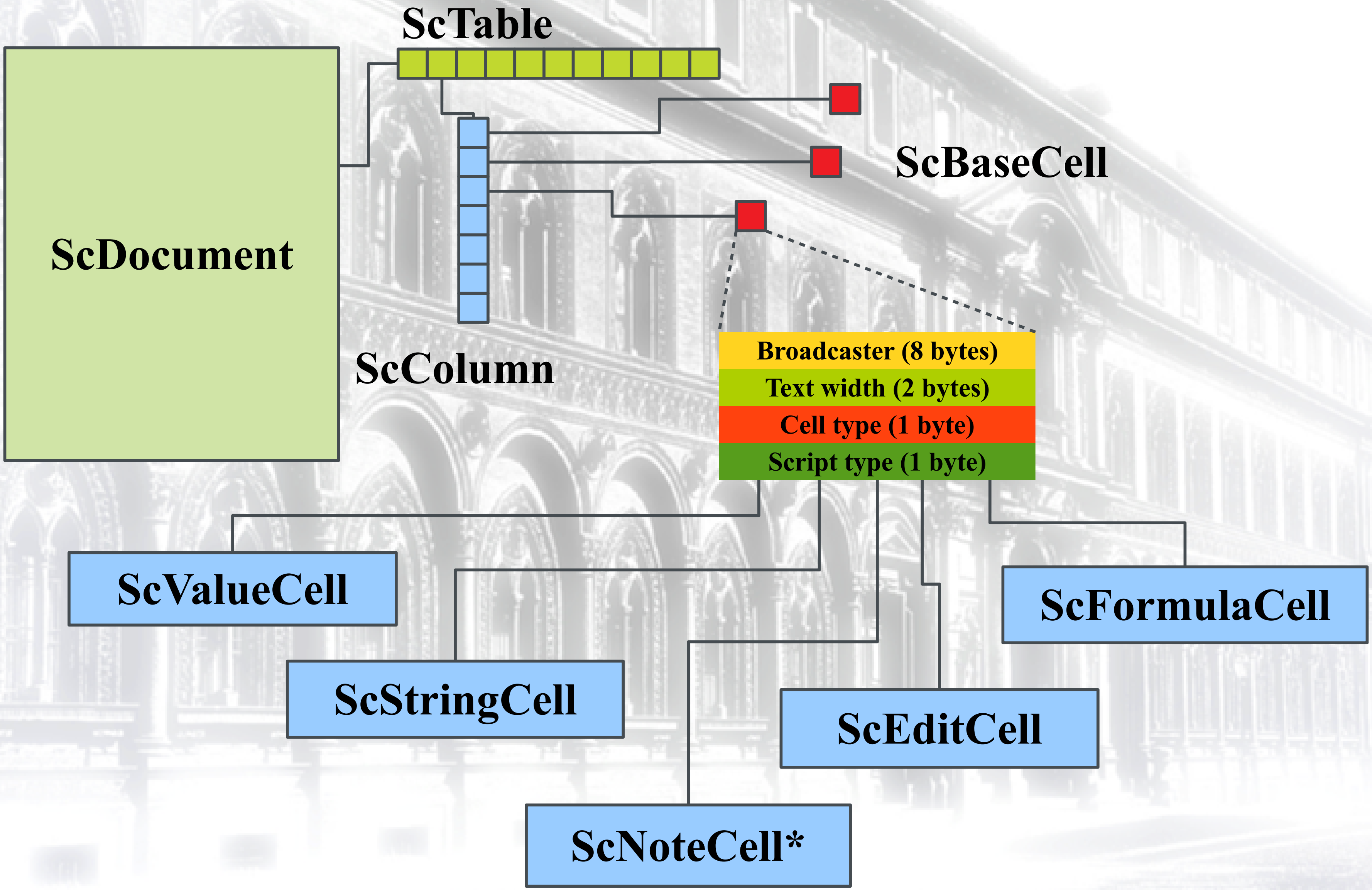
New Document Storage

MILANO CONFERENCE2013

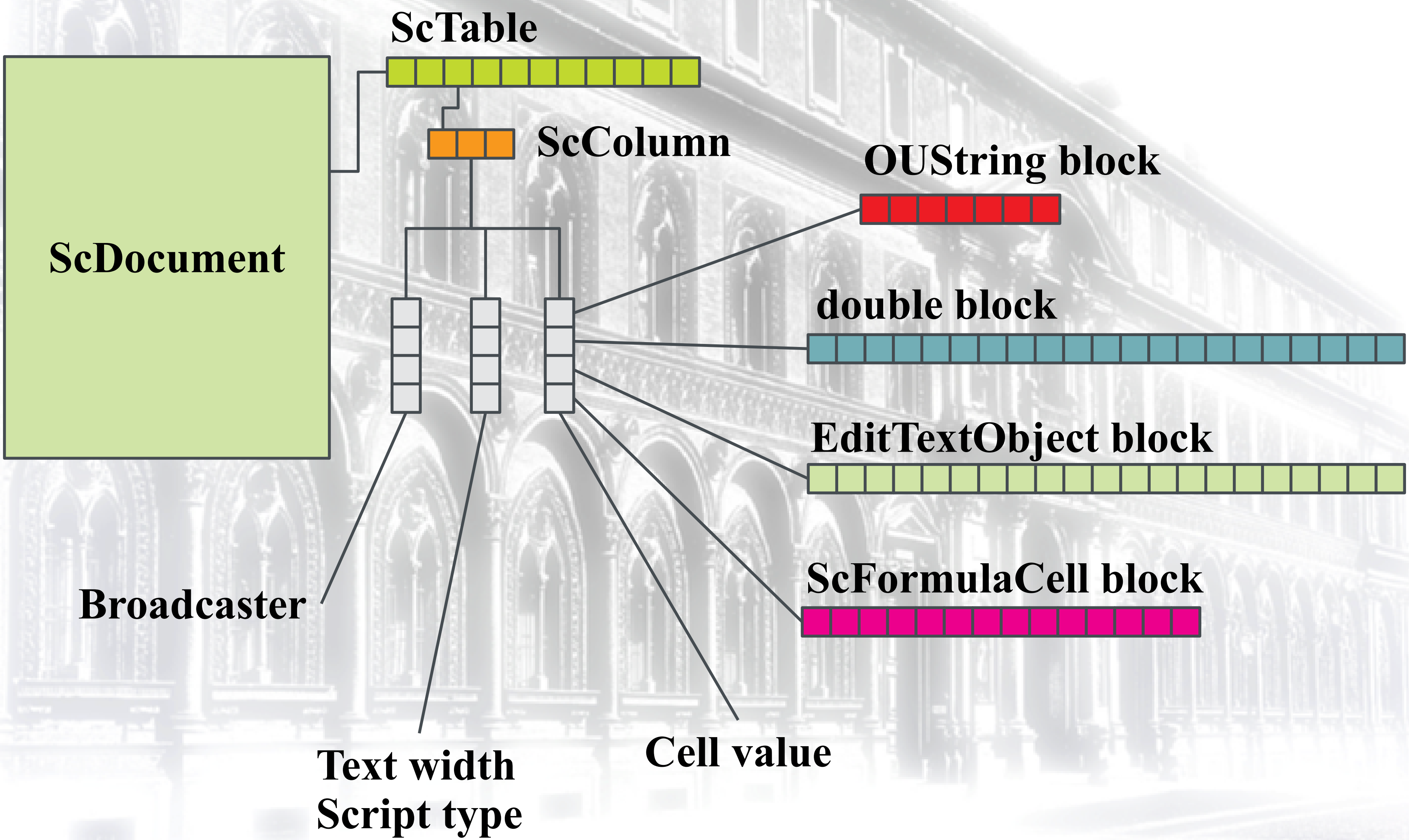
LibreOffice The Document Foundation

Collabora

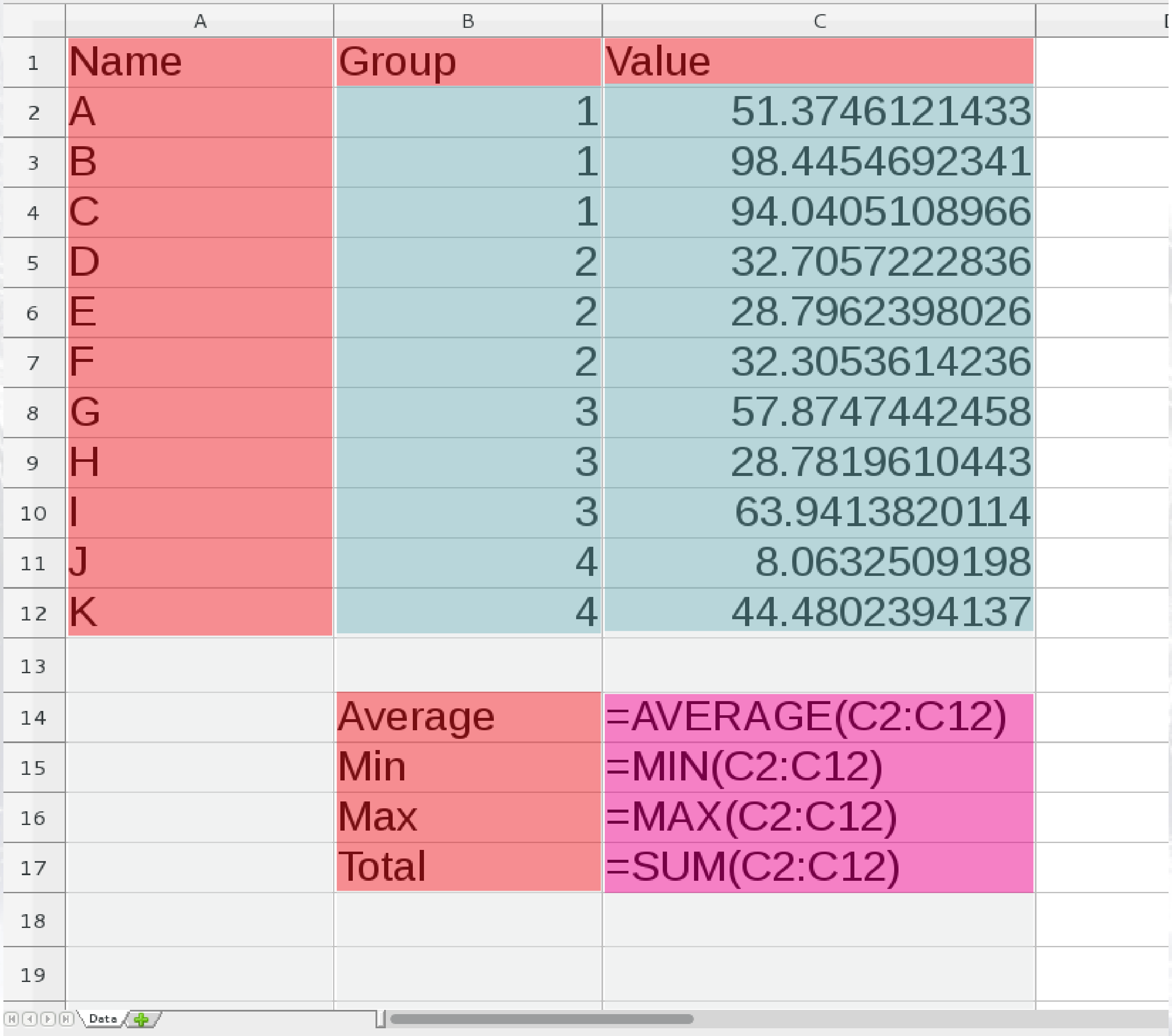
Old document model



New document model



New document model

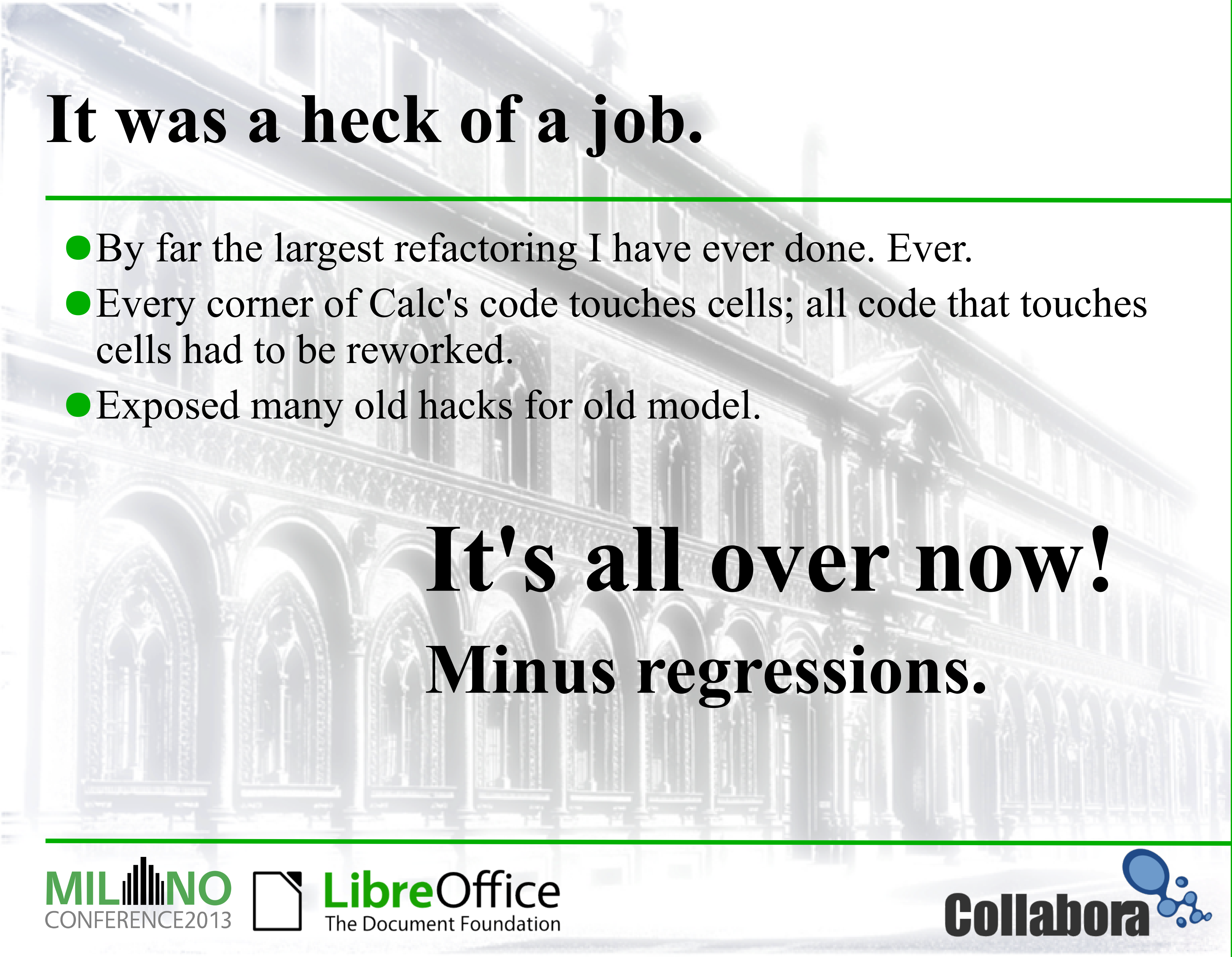


	A	B	C
1	Name	Group	Value
2	A	1	51.3746121433
3	B	1	98.4454692341
4	C	1	94.0405108966
5	D	2	32.7057222836
6	E	2	28.7962398026
7	F	2	32.3053614236
8	G	3	57.8747442458
9	H	3	28.7819610443
10	I	3	63.9413820114
11	J	4	8.0632509198
12	K	4	44.4802394137
13			
14		Average	=AVERAGE(C2:C12)
15		Min	=MIN(C2:C12)
16		Max	=MAX(C2:C12)
17		Total	=SUM(C2:C12)
18			
19			

Why new document model?

- Smaller memory footprint.
- Better locality of reference.
- Faster iteration of cells.
- Allow vectorized calculations via SIMD and/or GPU.

Having said that...



It was a heck of a job.

- By far the largest refactoring I have ever done. Ever.
- Every corner of Calc's code touches cells; all code that touches cells had to be reworked.
- Exposed many old hacks for old model.

It's all over now!

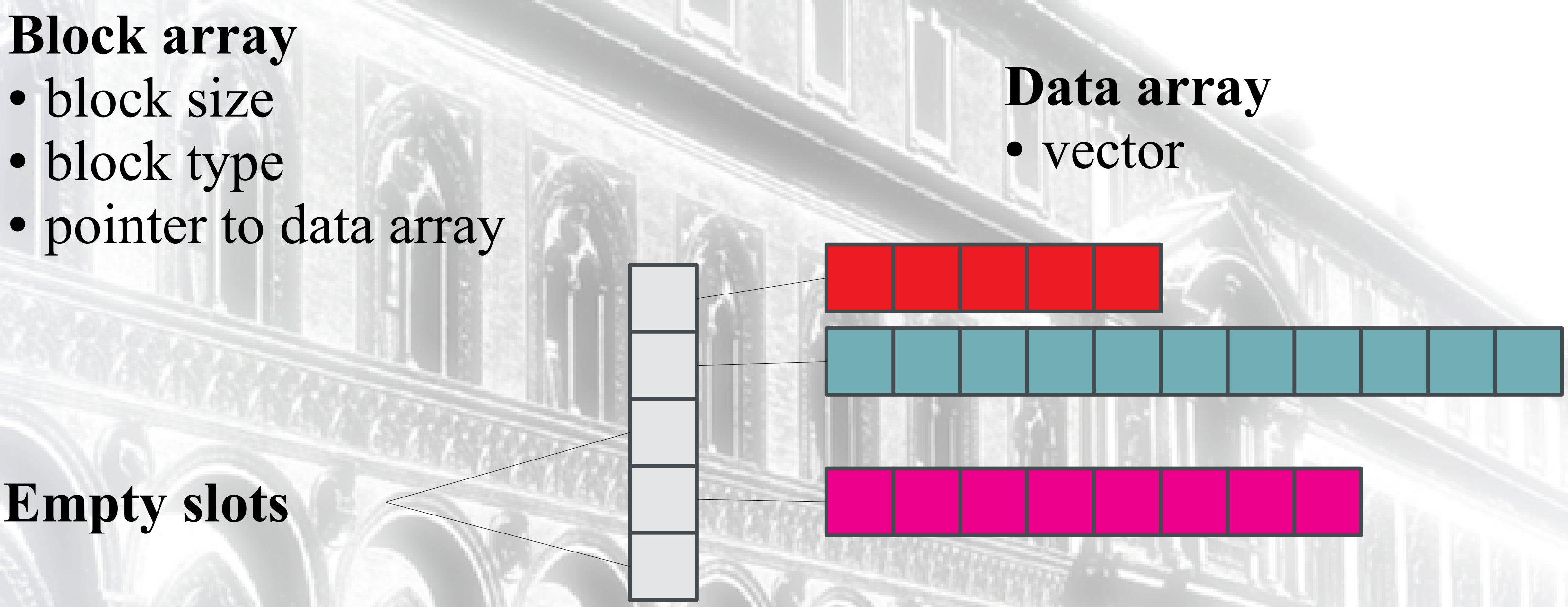
Minus regressions.

What Data Structure Is Used

mdds::multi_type_vector

- Used in new document storage.
  - Cells
  - Broadcasters
  - Text widths / script types
- C++ template from mdds library http://code.google.com/p/multidimalgorithm/
- One year for the initial version.
- Several iterations of improvement.

mdds::multi_type_vector



Block array

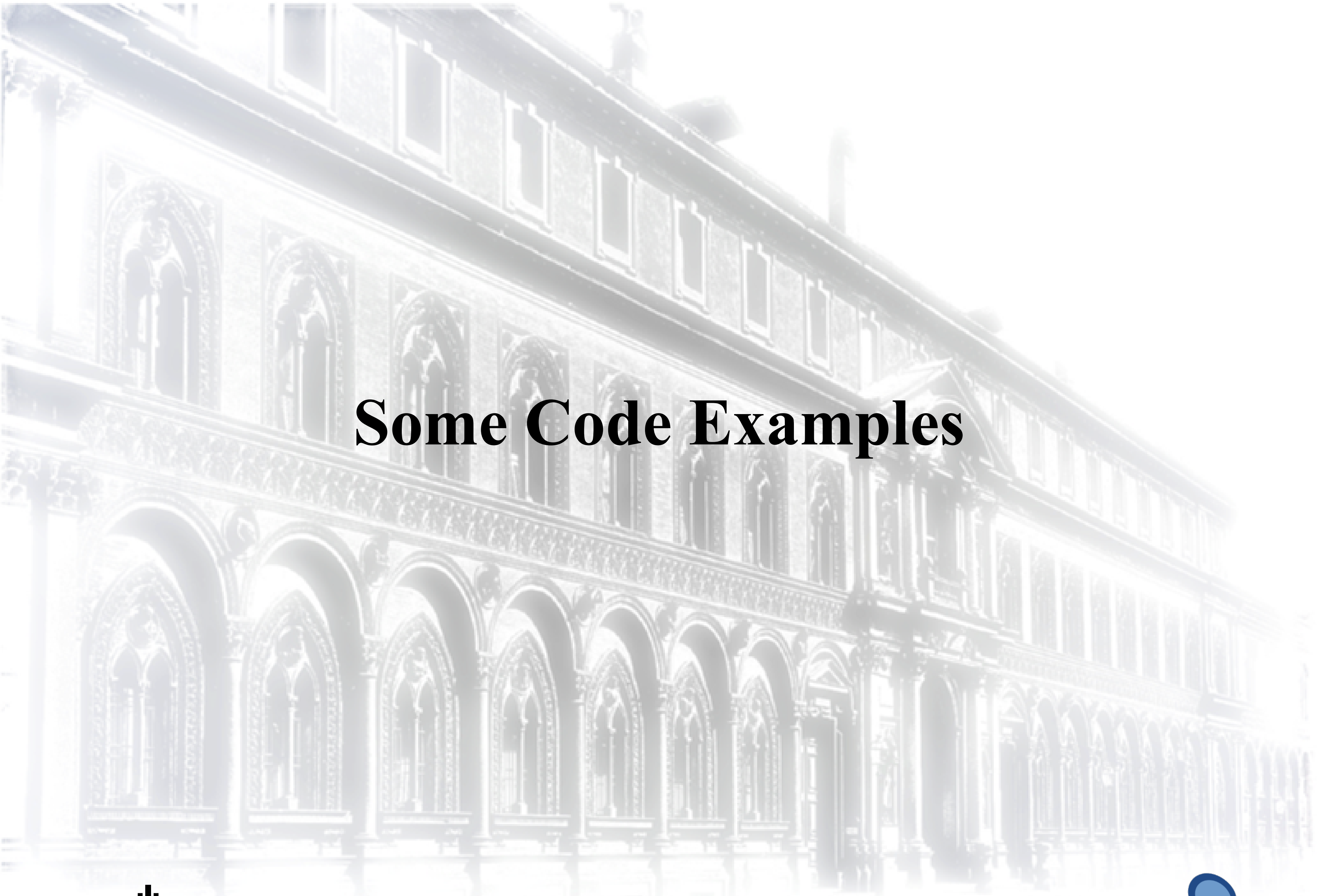
- block size
- block type
- pointer to data array

Data array

- vector

Empty slots

- Storage of unlimited number of types in single logical array.
- Contiguous elements of same type in contiguous memory space.



Some Code Examples

Putting Data In

Scenario

Insert a whole bunch of numeric values.
The values are stored contiguously.

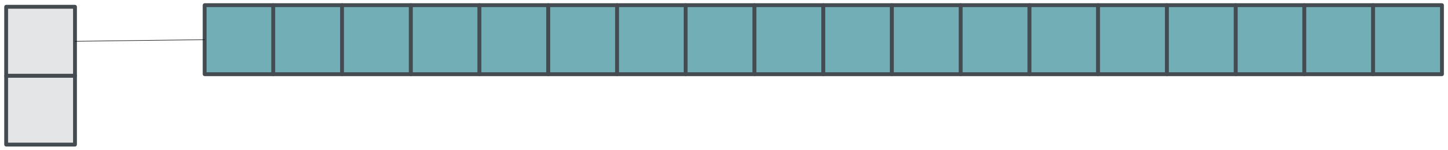
```
typedef multi_type_vector<mtv::element_block_func> mtv_type;
typedef vector<double> val_type;
const size_t test_size = 50000000;
val_type vals(test_size, 2.3);
```

1.51778 sec

1.42703 sec

0.111298 sec

Repeated single insertions



Finished!

Single array insertion

Finished!

Prefer array insertion over repeated single insertions.

Scenario

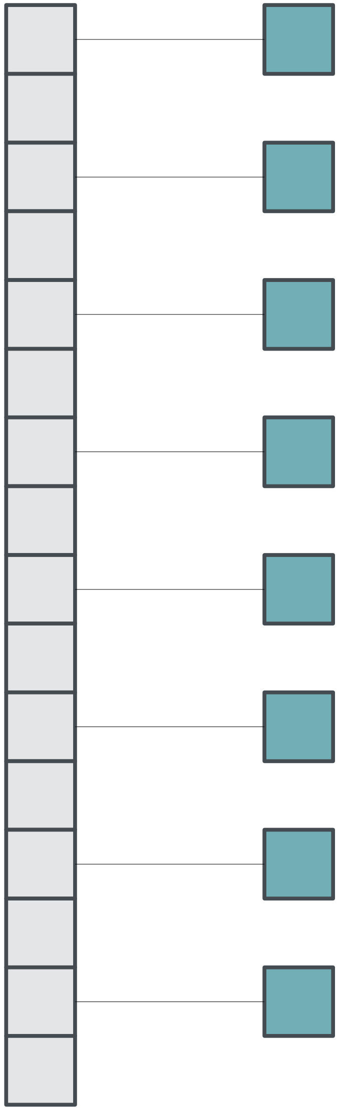
Insert a whole bunch of numeric values. But values are only to be set at logical even positions. Cells at the odd positions will be left empty.

```
typedef multi_type_vector<mtv::element_block_func> mtv_type;
typedef vector<double> val_type;
const size_t test_size = 1000000;
val_type vals(test_size/10, 2.3);
```

38.2751 sec

0.03113 sec

Repeated insertion of alternating empty and non-empty cells.



Each insertion creates two new blocks.

The more blocks the slower the block position lookup.

Using a position hint indicator helps avoid the cost of block position lookup.

Accessing Data

Scenario

Iterate through the entire container and add all numeric values. Containers contain numeric cells at odd row positions.

```
using namespace mdds::mtv;
typedef multi_type_vector<element_block_func> mtv_type;
typedef vector<double> val_type;
const size_t test_size = 100000;
val_type vals(test_size/2, 2.3);

mtv_type store(test_size);
mtv_type::iterator pos = store.begin();
for (size_t i = 0, n = vals.size(); i < n; ++i)
    pos = store.set(pos, i*2, vals[i]);

double sum = 0.0;
for (size_t i = 0, n = store.size(); i < n; ++i)
{
    if (store.get_type(i) == element_type_numeric)
        sum += store.get<double>(i);
}
cout << "sum = " << sum << endl;
```

18.9474 sec

```
using namespace mdds::mtv;
typedef multi_type_vector<element_block_func> mtv_type;
typedef vector<double> val_type;
const size_t test_size = 100000;
val_type vals(test_size/2, 2.3);

mtv_type store(test_size);
mtv_type::iterator pos = store.begin();
for (size_t i = 0, n = vals.size(); i < n; ++i)
    pos = store.set(pos, i*2, vals[i]);

double sum = 0;
mtv_type::const_iterator it = store.begin(), it_end = store.end();
for (; it != it_end; ++it)
{
    if (it->type != element_type_numeric)
        continue;

    numeric_element_block::const_iterator blk
        = numeric_element_block::begin(*it->data);
    numeric_element_block::const_iterator blk_end
        = numeric_element_block::end(*it->data);
    for (; blk != blk_end; ++blk)
        sum += *blk;
}
cout << "sum = " << sum << endl;
```

0.00056 sec

What's in block iterator node?

```
using namespace mdds::mtv;
typedef multi_type_vector<element_block_func> mtv_type;

mtv_type store(10);
mtv_type::iterator it = store.begin();
```

Scenario

Iterate through the container above the 100th element. Check every 3rd element, and if it's numeric, add it to the total.

```
using namespace mdds::mtv;
typedef multi_type_vector<element_block_func> mtv_type;
typedef vector<double> val_type;
const size_t test_size = 100000;
val_type vals(test_size/2, 2.3);

mtv_type store(test_size);
mtv_type::iterator pos = store.begin();
for (size_t i = 0, n = vals.size(); i < n; ++i)
    pos = store.set(pos, i*2, vals[i]);

double sum = 0.0;
for (size_t i = 100, n = store.size(); i < n; i += 3)
{
    if (store.get_type(i) == element_type_numeric)
        sum += store.get<double>(i);
}

cout << "sum = " << sum << endl;
```

6.49647 sec

No code example for iterating through blocks.

too much work just to keep track of logical element positions.

```
using namespace mdds::mtv;
typedef multi_type_vector<element_block_func> mtv_type;
typedef vector<double> val_type;
const size_t test_size = 100000;
val_type vals(test_size/2, 2.3);

mtv_type store(test_size);
mtv_type::iterator pos = store.begin();
for (size_t i = 0, n = vals.size(); i < n; ++i)
    pos = store.set(pos, i*2, vals[i]);

double sum = 0.0;
pos = store.begin();
for (size_t i = 100, n = store.size(); i < n; i += 3)
{
    mtv_type::position_type pos_obj = store.position(pos, i);
    pos = pos_obj.first;
    size_t offset = pos_obj.second;
    if (pos->type == element_type_numeric)
        sum += numeric_element_block::at(*pos->data, offset);
}

cout << "sum = " << sum << endl;
```

0.0008 sec

What's a position object?

```
using namespace mdds::mtv;
typedef multi_type_vector<element_block_func> mtv_type;

mtv_type store(100);
mtv_type::position_type pos_obj = store.position(4);
```

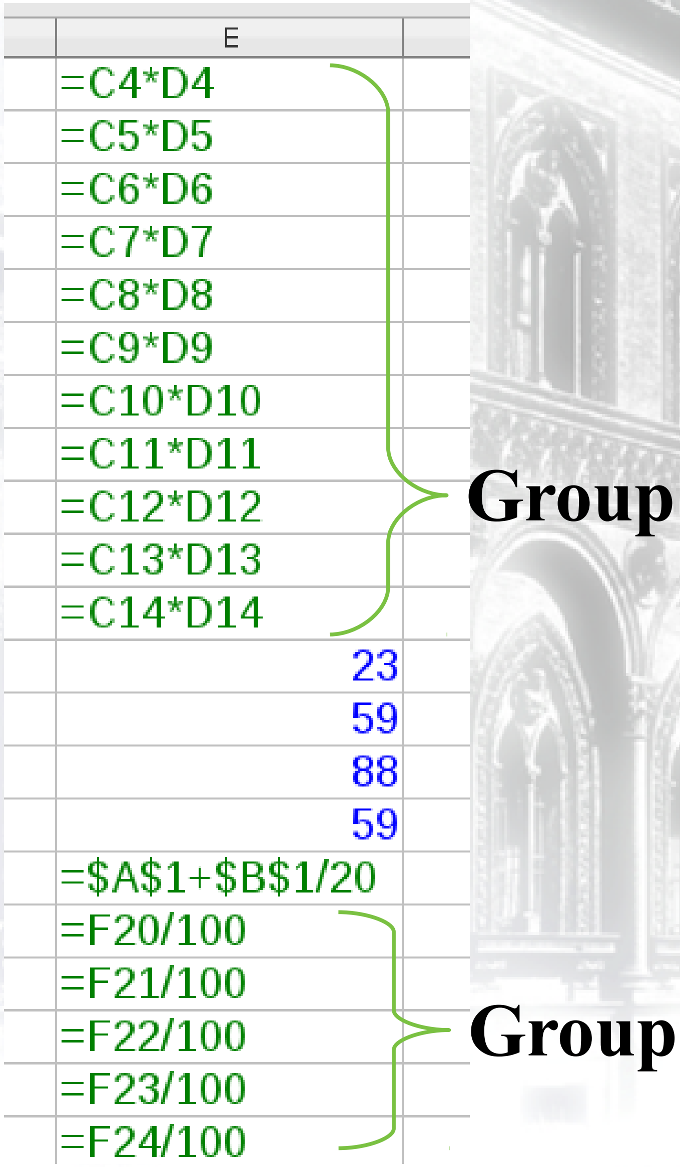
The takeaways

- Prefer one-step array insertion over repeated individual value insertions.
- Always use block iterators as position hints if you do individual value insertions in loop.
- Know what's in a block iterator: type, position, size, and data.
- Know what a position object is, and use it judiciously.

Enough with code...

Formula Groups

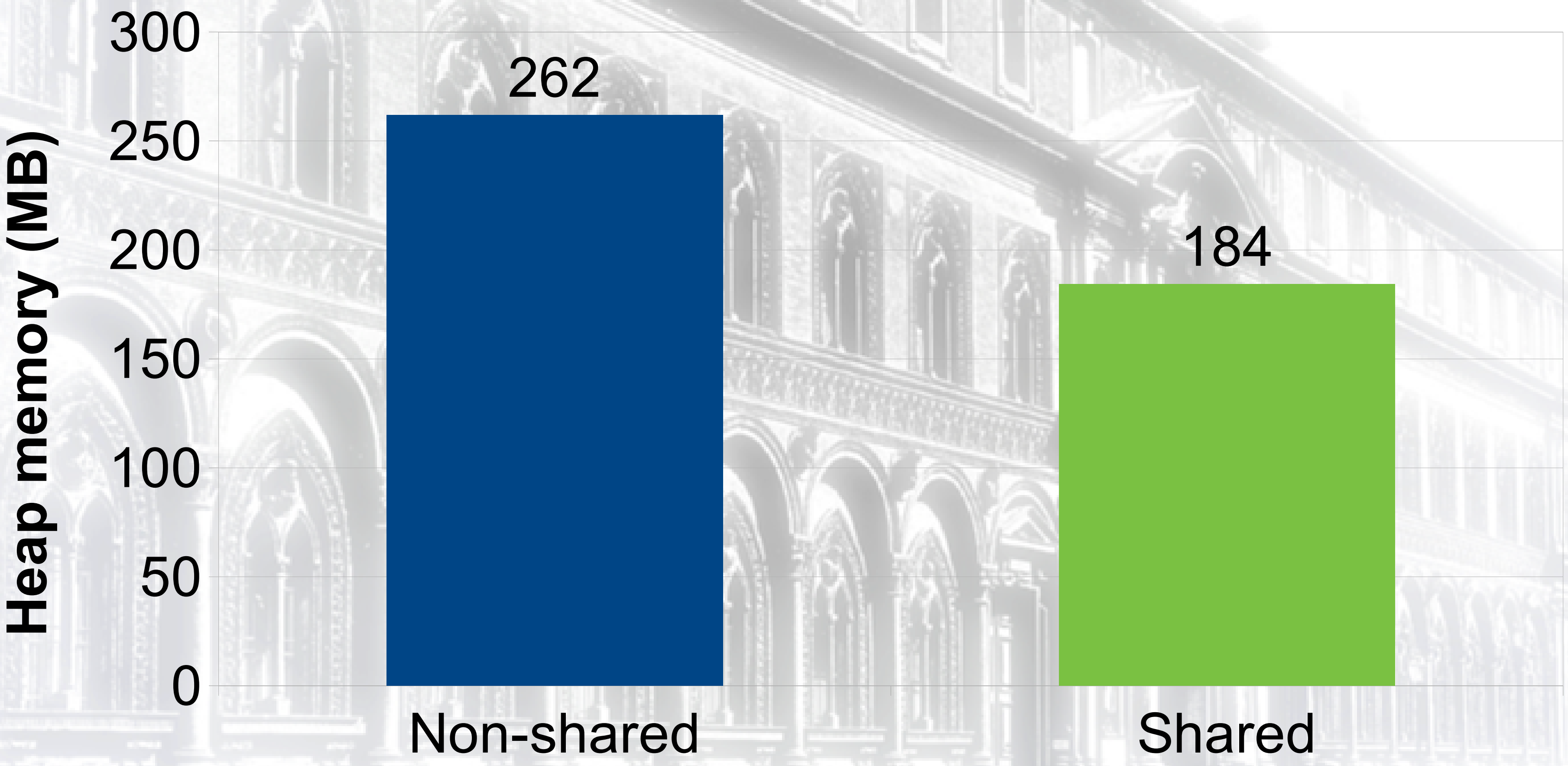
What's a formula group?



E
=C4*D4
=C5*D5
=C6*D6
=C7*D7
=C8*D8
=C9*D9
=C10*D10
=C11*D11
=C12*D12
=C13*D13
=C14*D14
23
59
88
59
=$A$1+$B$1/20
=F20/100
=F21/100
=F22/100
=F23/100
=F24/100

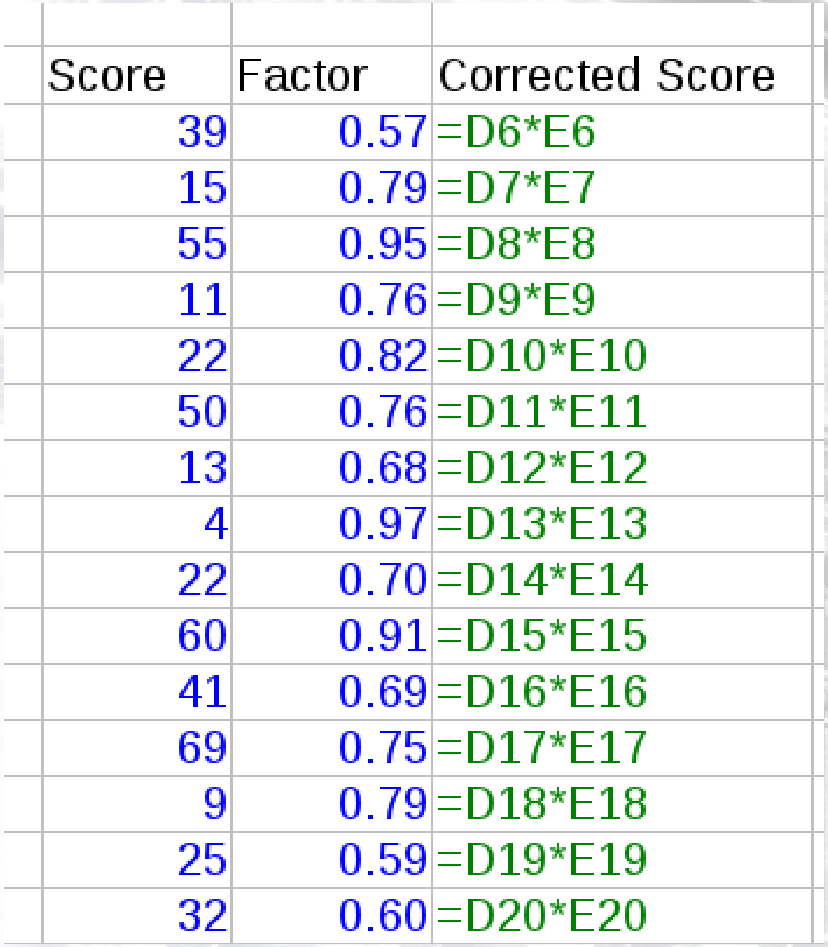
- Group of adjacent formula cells whose formula tokens are identical.
- In the vertical direction only.
- One token array for the whole group for reduced memory usage (a.k.a. shared formula).

Effect of shared formula

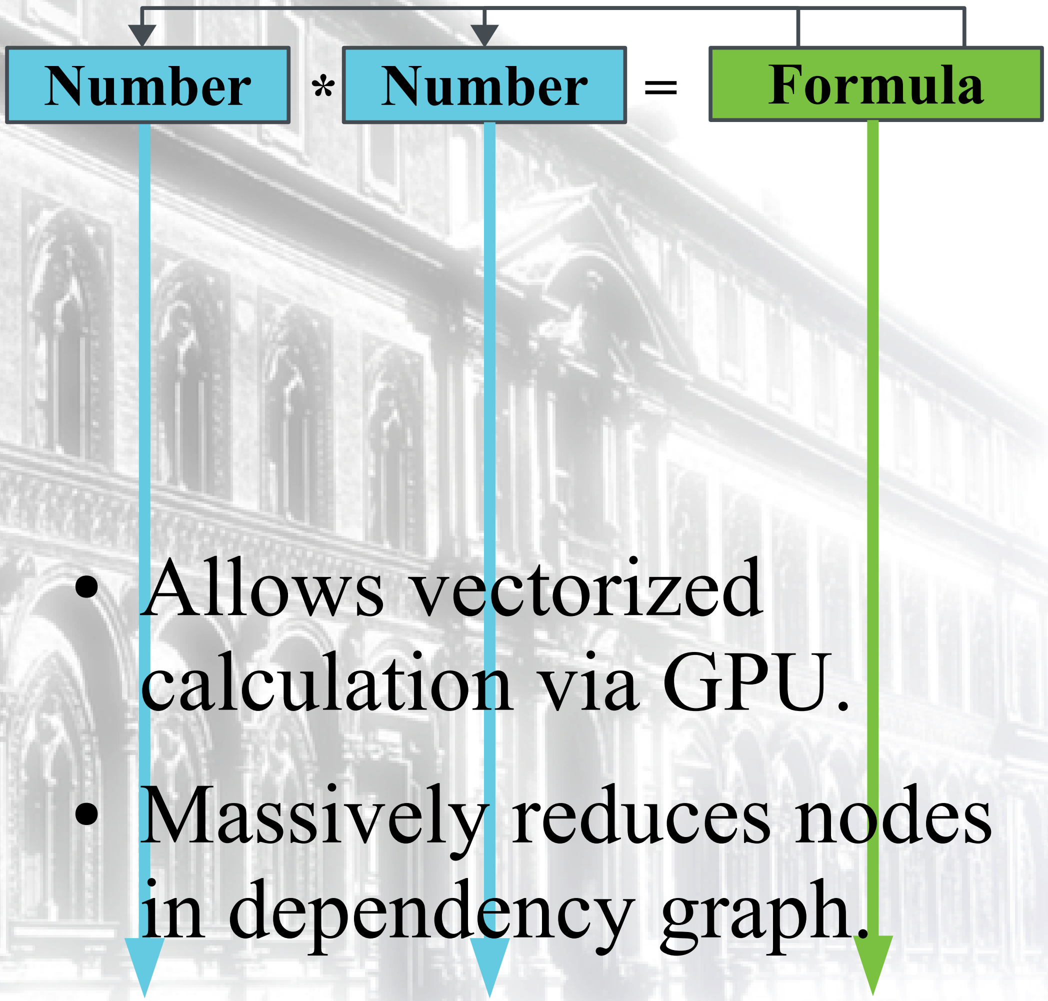


http://kohei.us/2013/08/15/shared-formula-to-reduce-memory-usage/

Why formula groups?



Score	Factor	Corrected Score
39	0.57	=D6*E6
15	0.79	=D7*E7
55	0.95	=D8*E8
11	0.76	=D9*E9
22	0.82	=D10*E10
50	0.76	=D11*E11
13	0.68	=D12*E12
4	0.97	=D13*E13
22	0.70	=D14*E14
60	0.91	=D15*E15
41	0.69	=D16*E16
69	0.75	=D17*E17
9	0.79	=D18*E18
25	0.59	=D19*E19
32	0.60	=D20*E20



- Allows vectorized calculation via GPU.
- Massively reduces nodes in dependency graph.

OpenCL Interpreter

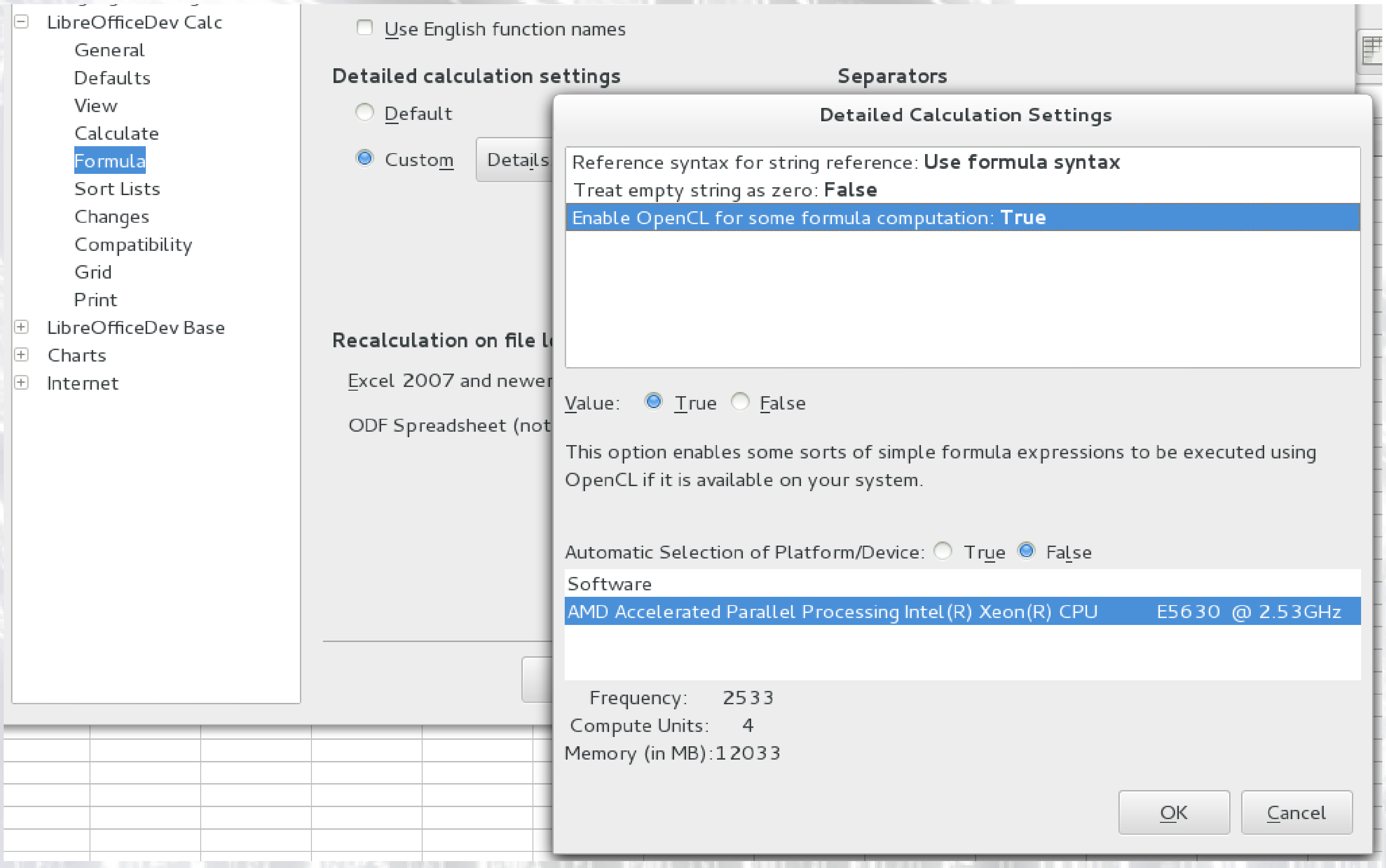
OpenCL Interpreter

- Vectorized group calculation.
- OpenCL API - public standard http://www.khronos.org/opencl/
- Supported by AMD, NVIDIA, and Intel GPU's.
- Parallel computation of formula groups.
- Code funded & co-developed by

AMD

MULTICORE WARE

Enable OpenCL Interpreter



□ Use English function names

Detailed calculation settings

Separators

○ Default

● Custom

Detailed Calculation Settings

Reference syntax for string reference: **Use formula syntax**
Treat empty string as zero: **False**
Enable OpenCL for some formula computation: **True**

Value: ● True ○ False

This option enables some sorts of simple formula expressions to be executed using OpenCL if it is available on your system.

Automatic Selection of Platform/Device: ○ True ● False

Software
AMD Accelerated Parallel Processing Intel(R) Xeon(R) CPU E5630 @ 2.53GHz

Frequency: 2533
Compute Units: 4
Memory (in MB):12033

UI and OpenCL device detection by Markus Mohrhard.

Current issues

- Still only effective on limited use cases.
- Stability improvement.
- Unit test ?
- More functions to cover.
- Very promising.

Thanks for listening!

MILANO CONFERENCE2013

LibreOffice The Document Foundation

Collabora