Collabora

LibreOffice Calc

Spreadsheets on the GPU

Michael Meeks <michael.meeks@collabora.com>

mmeeks, #libreoffice-dev, irc.freenode.net

"Stand at the crossroads and look; ask for the ancient paths, ask where the good way is, and walk in it, and you will find rest for your souls..." - Jeremiah 6:16

LibreOffice

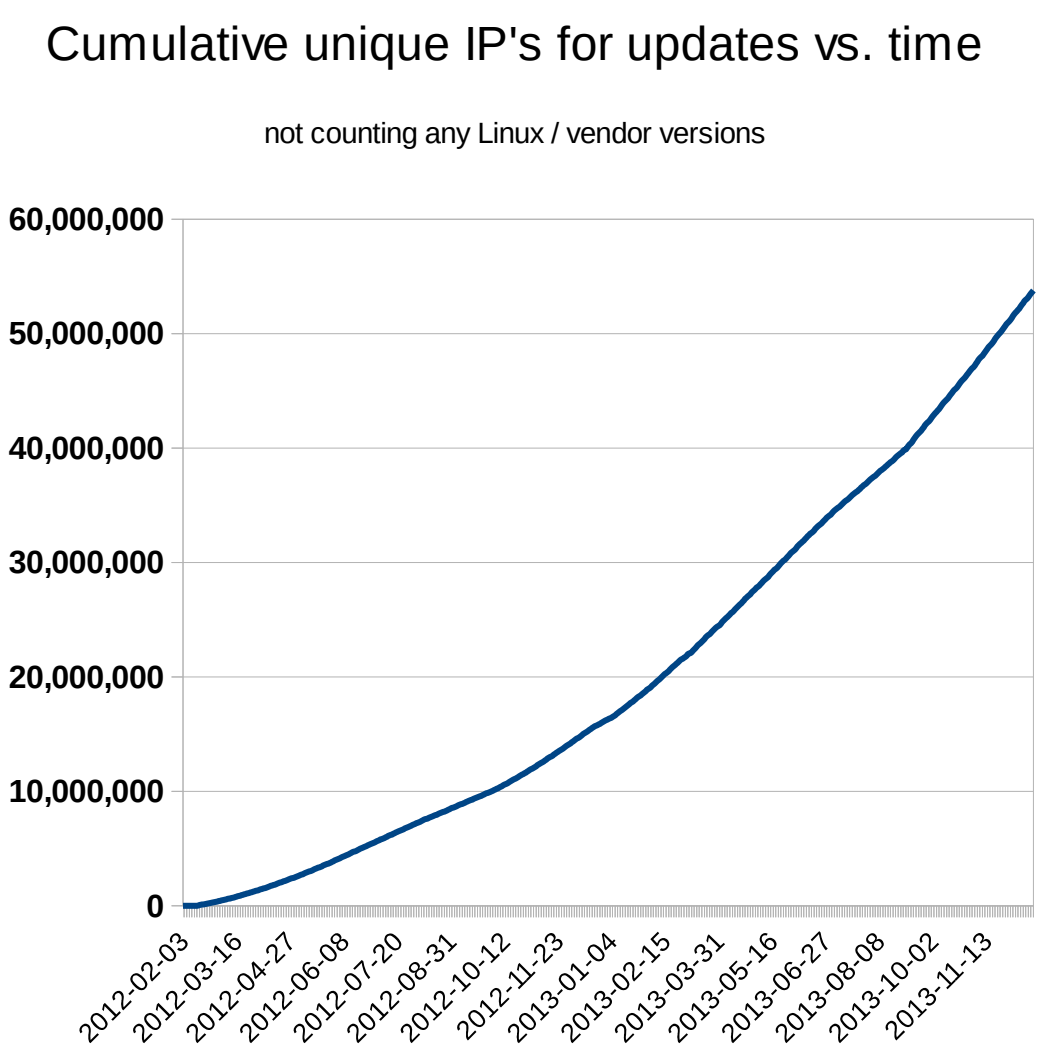
Overview

- LibreOffice ?
- A bit about:
  - GPUs …
  - Spreadsheets
- Internal re-factoring
  - OpenCL optimisation
  - new calc features
  - XML / load performance
- Calc / GPU questions ?
- Questions ?

LibreOffice Project & Software

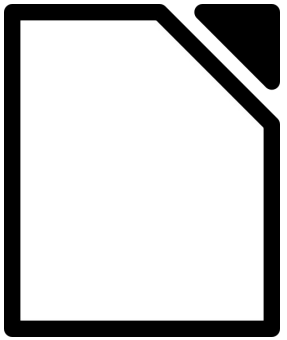
- Open Source / Free Software
- One million new unique IPs per week (that we can track)
  - Double the weekly growth one year ago.
- Tens of millions of users, and growing fast.
- Hundred+ contributing coders each month
- 2500+ commits last month
- Around a thousand developers (including QA, Translators, UX etc.

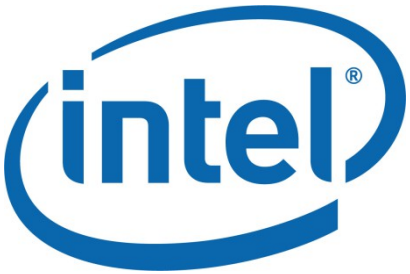
http://www.libreoffice.org/

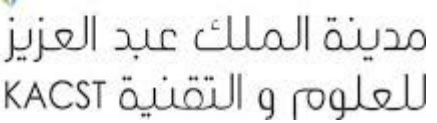


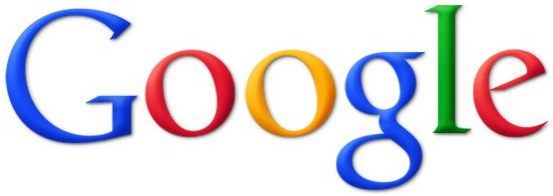
Advisory Board Members

This slide's layout is a victim of our success here ...









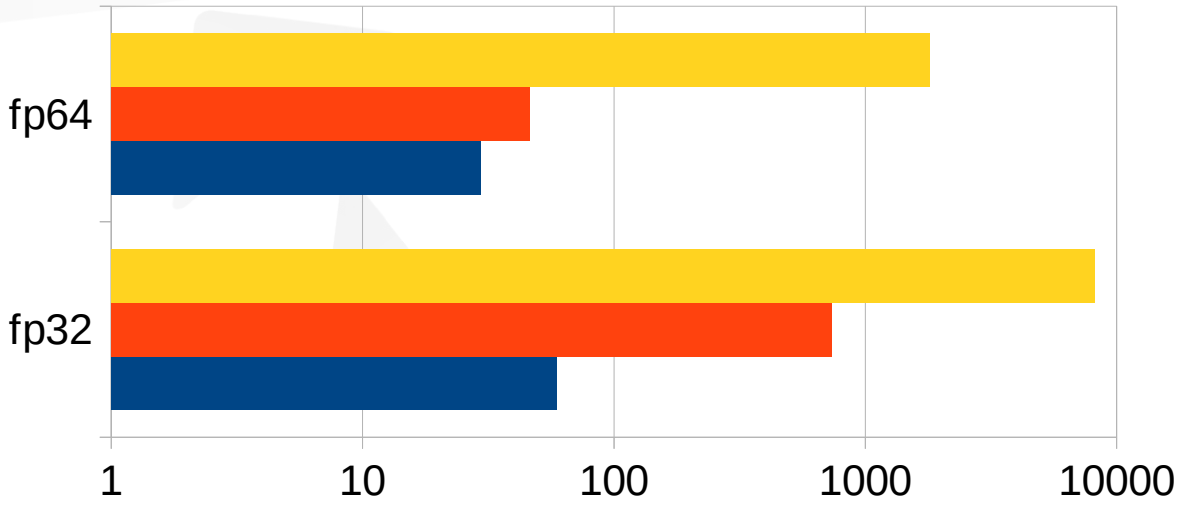
FREE SOFTWARE
FOUNDATION

Why use the GPU ?

APUs – GPU faster than CPU

- Tons of un-used Compute Units across your APU
- Double precision is un-reasonably slower
  - And **precision** is **non-negotiable** for spreadsheets IEE764 required.
- Better power usage per flop.

Numbers based on a Kaveri 7850K APU - & top-end discrete Graphics card.

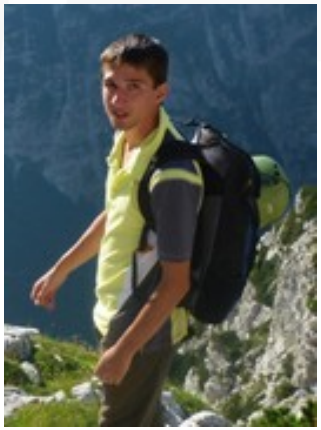


Flops : note the log scale ...

Developers behind the calc re-work:

Kohei Yoshida:

- MDDS maintainer
- Heroic calc core re-factorer
- Code Ninja etc.



Matus Kukan

Data Streamer,
G-builder,
Size optimizer ..

Markus Mohrhard

- Calc maintainer,
- Chart2 wrestler
- Unit tester par Excellence

etc.

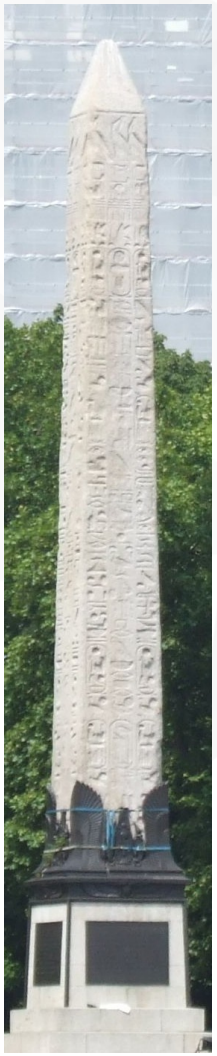
AMD

Jagan Lokanatha
Kismat Singh

MULTICORE WARE

A large OpenCL team, Particularly I-Jui (Ray) Sung

Spreadsheet Geometry



An early Spreadsheet C 3000 BC

Aspect ratio: 8:1

Contents:

Victory against every land … who giveth all life forever …

50% of spreadsheets used to make business decisions.

Excel 2003

64k x 256

Aspect: 256:1

Excel 2010

10^6 x 16k

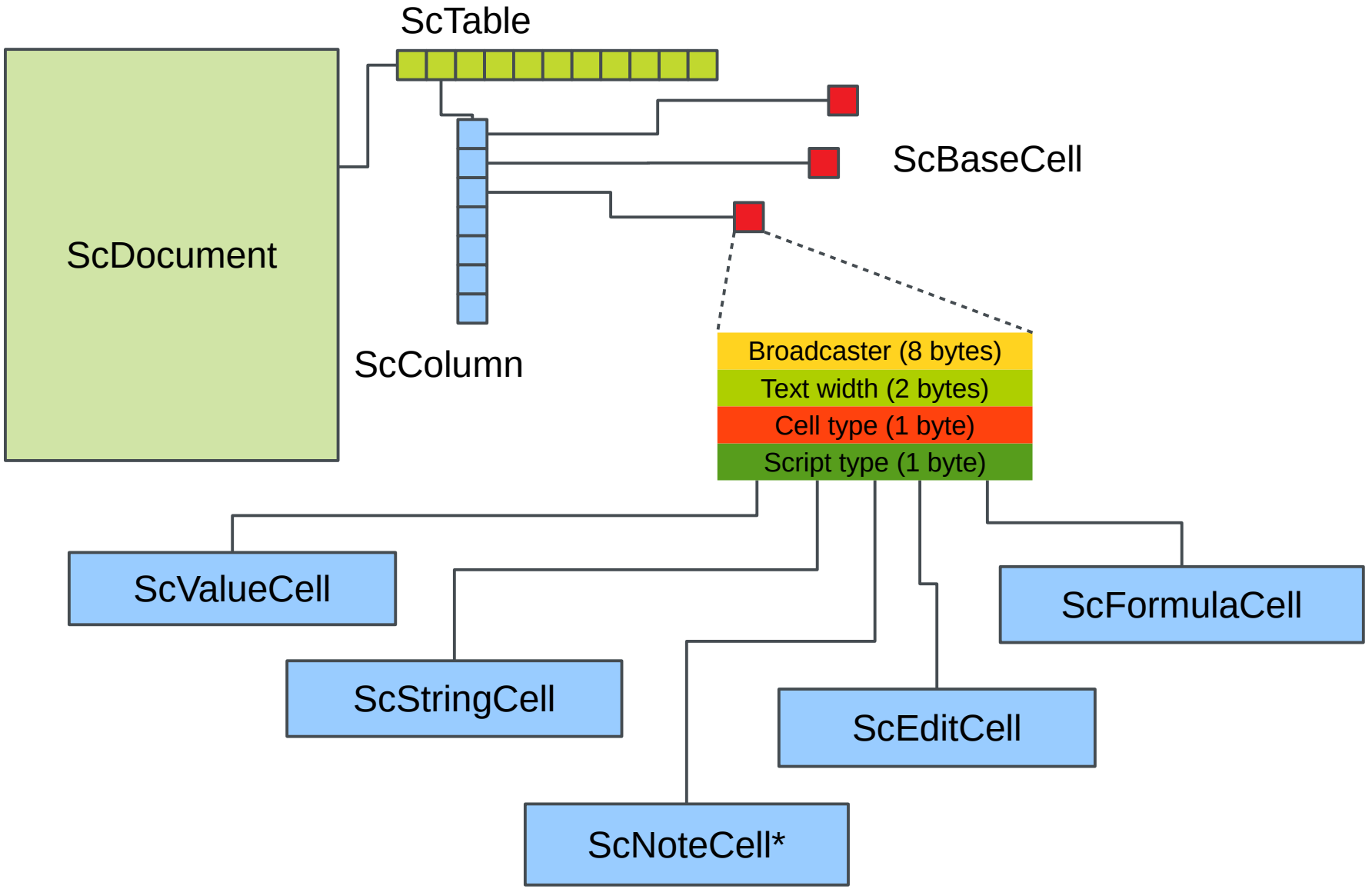
Aspect: 16:1

The *'Broom Handle'* aspect ratio.

Columnar data structures

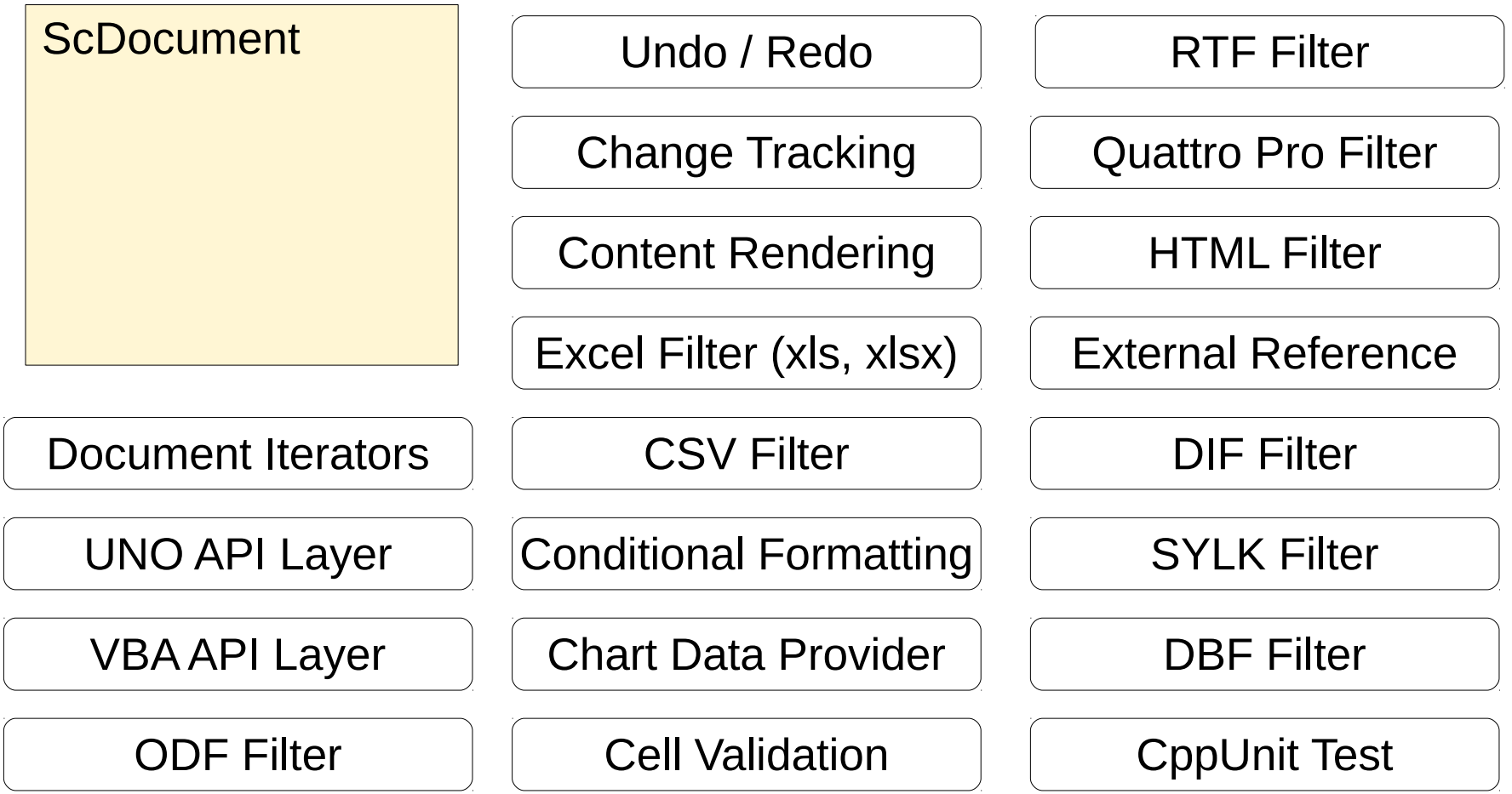
Spreadsheet Core Data Storage

The joy of Object Orientation



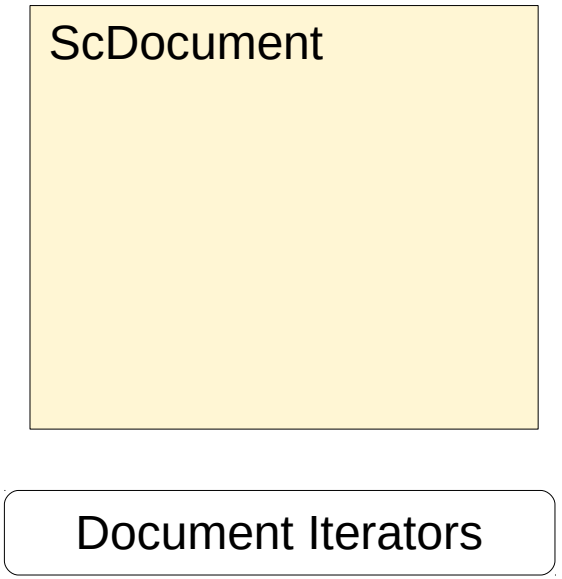
Abstraction of Cell Value Access

ScBaseCell Usage **(Before)**



Abstraction of Cell Value Access

ScBaseCell Usage **(After)**



Biggest calc core re-factor in a decade+

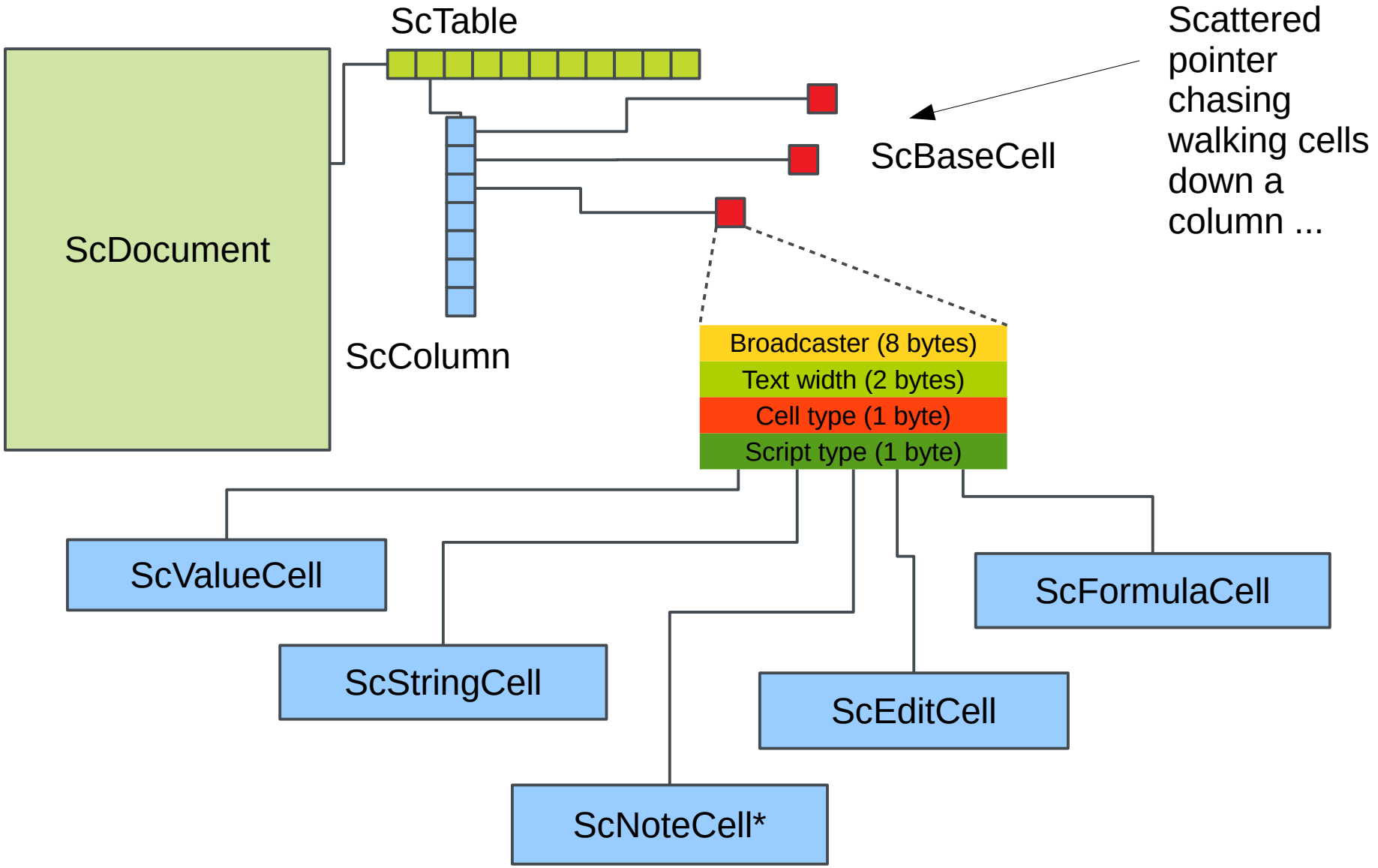
Dis-infecting the horrible, long-term, inherited structural problems of Calc.

Lots of new **unit tests** being created for the first time for the calc core.

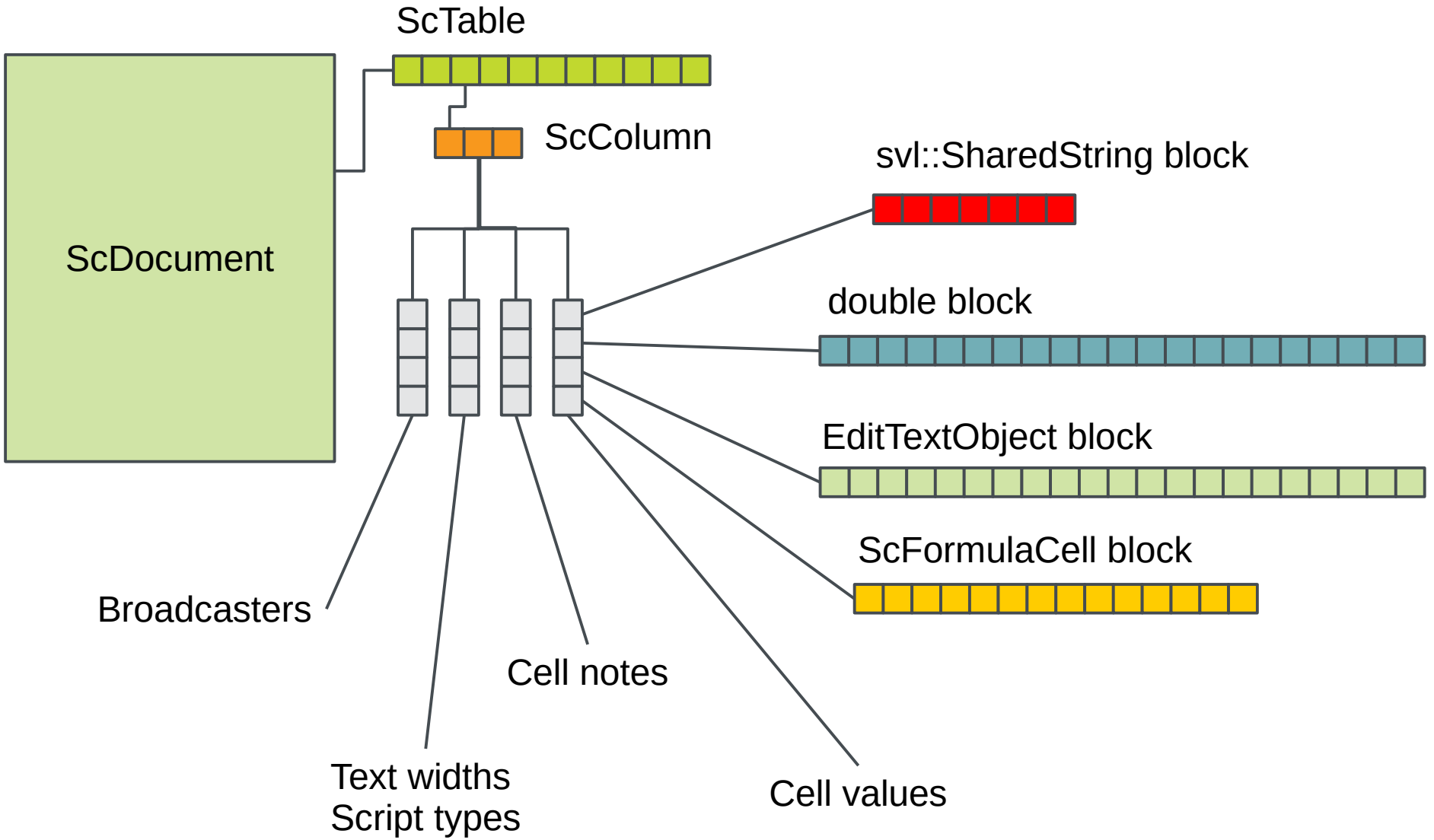
Moved to using new 'MDDS' data structures.

2x weeks with no compile ...

Before (ScBaseCell)



After (mdds::multi_type_vector)



Iterating over cells (old way)

… loop down a column … and the inner loop:

```
    double nSum = 0.0;
    ScBaseCell* pCell = pCol >maItems[nColRow].pCell;
    ++nColRow;
    switch (pCell->GetCellType())
    {
       case CELLTYPE_VALUE:
            nSum += ((ScValueCell*)pCell)->GetValue();
            break;
       case CELLTYPE_FORMULA:
            … something worse ...
       case CELLTYPE_STRING:
       case CELLTYPE_EDIT:
            …
       case CELLTYPE_NOTE:
            …
    }
```


Iterating over cells (new way)

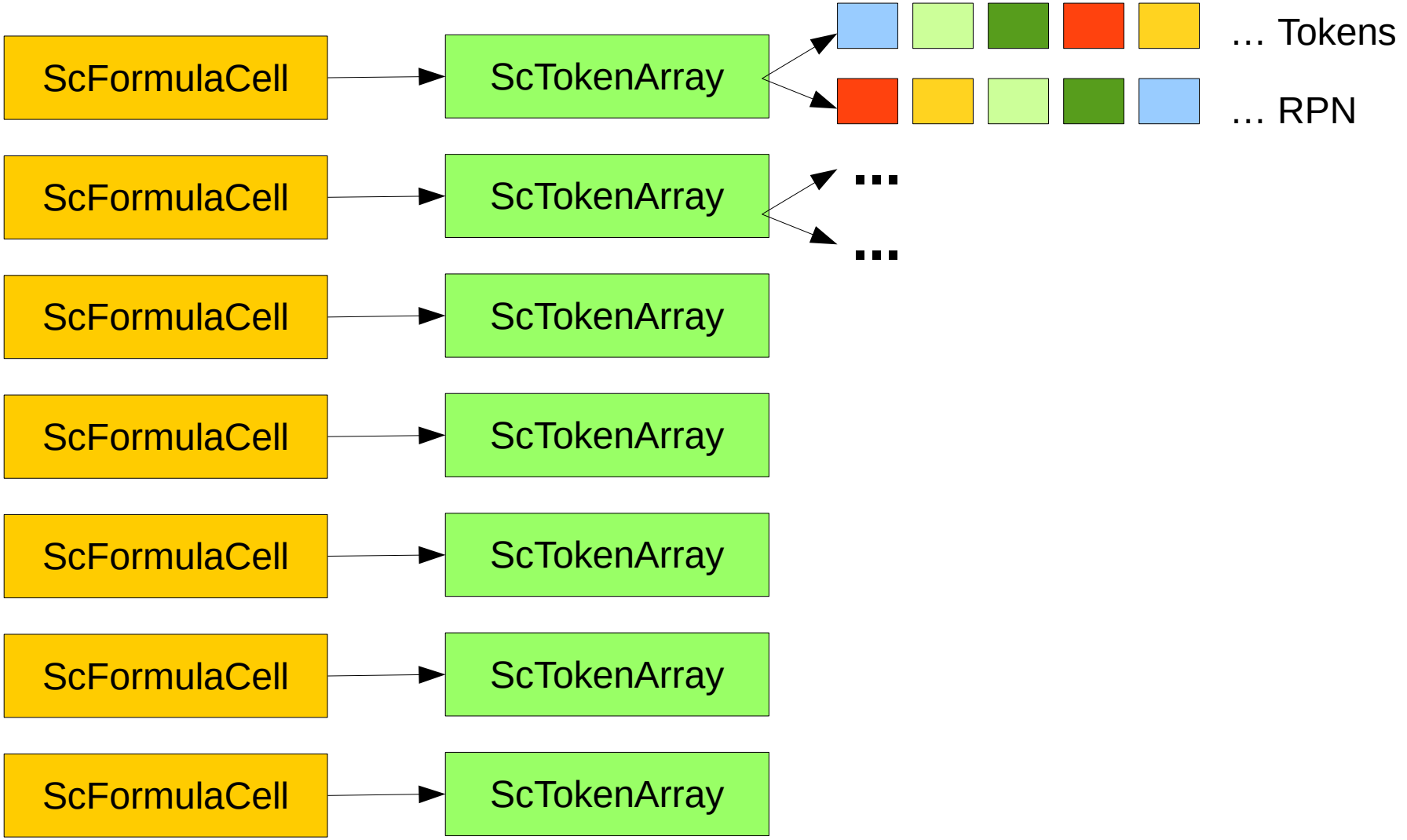
```
double nSum = 0.0;

for (size_t i = 0; i < nChunkLength; i++)
    nSum += pDoubleChunk[i];
```

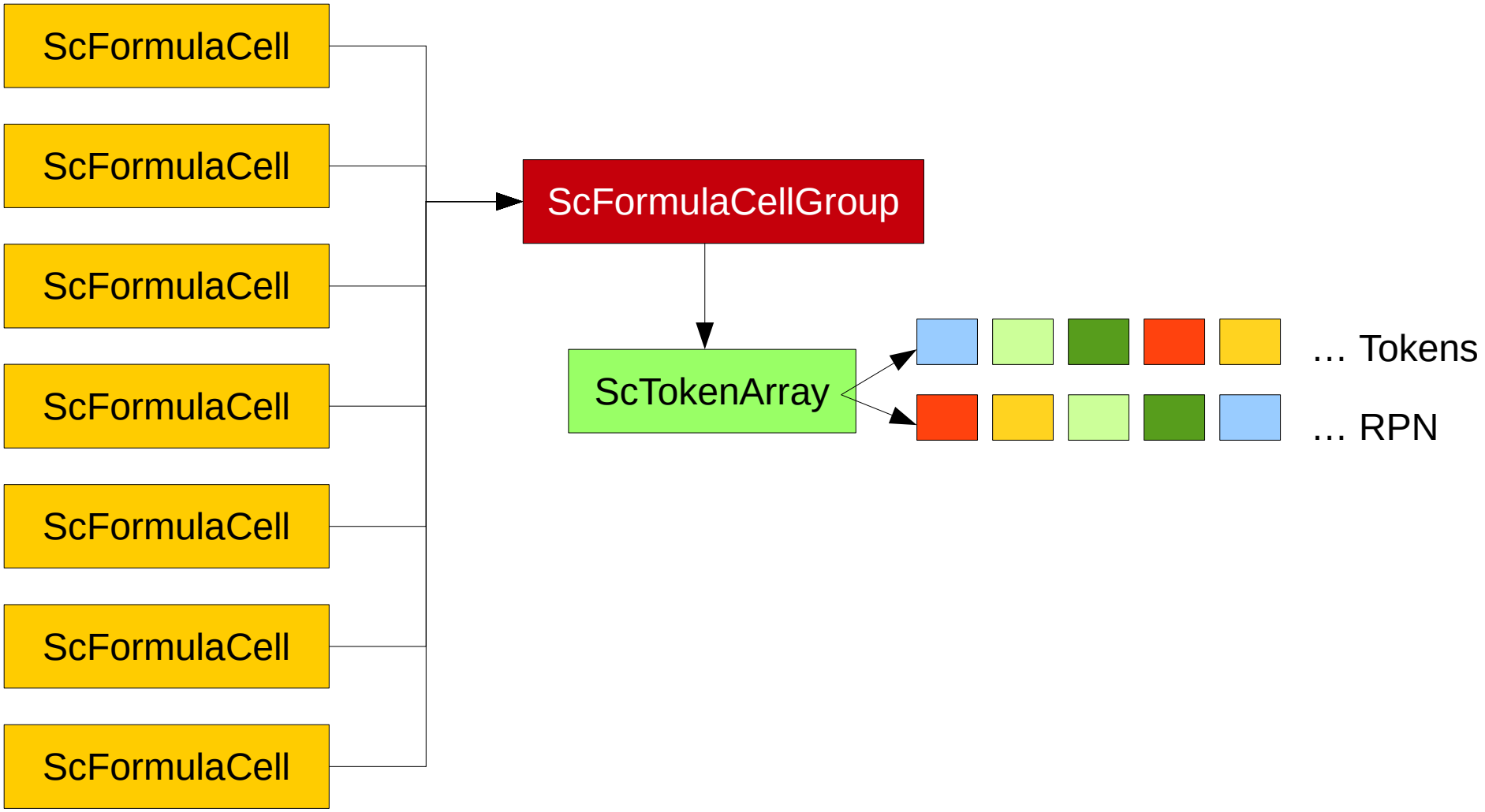
ONO. from a vectoriser ...

Shared Formula

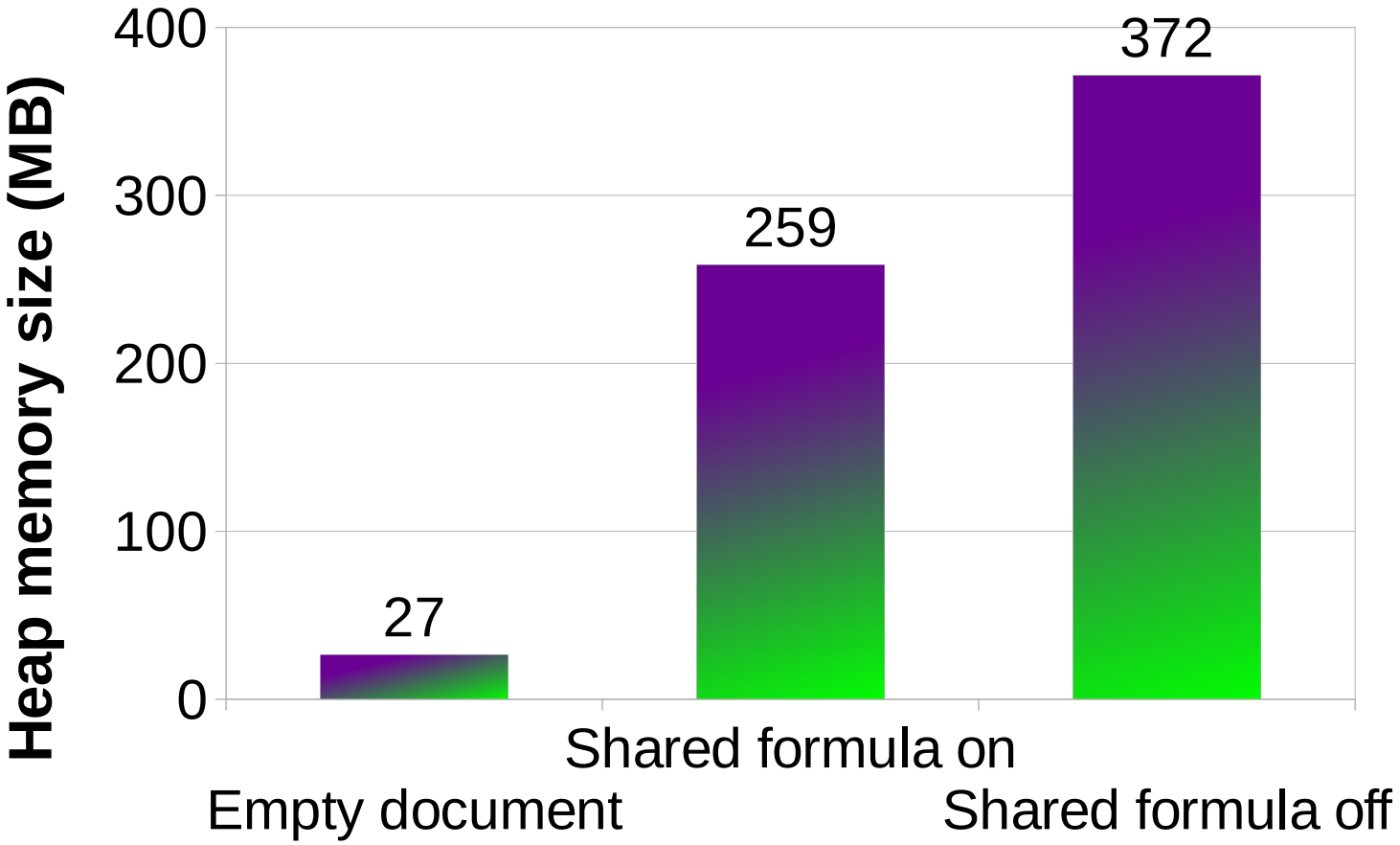
Before


ScFormulaCell
ScTokenArray
… Tokens
… RPN
ScFormulaCell
ScTokenArray
ScFormulaCell
ScTokenArray
ScFormulaCell
ScTokenArray
ScFormulaCell
ScTokenArray
ScFormulaCell
ScTokenArray
ScFormulaCell
ScTokenArray

After


ScFormulaCell
ScFormulaCell
ScFormulaCell
ScFormulaCell
ScFormulaCell
ScFormulaCell
ScFormulaCell
ScFormulaCellGroup
ScTokenArray
… Tokens
… RPN

Memory usage

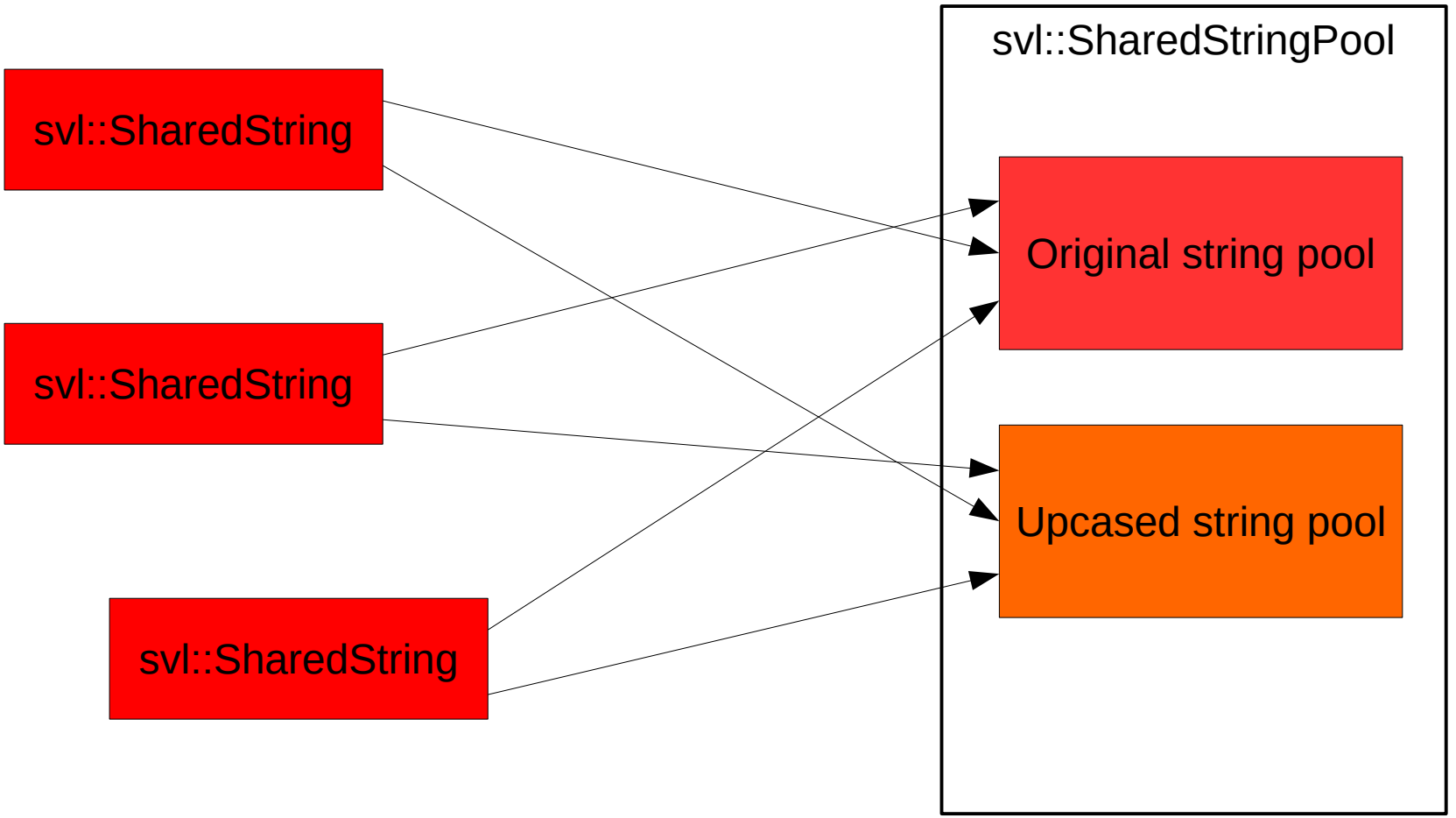


Test document used:
http://kohei.us/wp-content/uploads/2013/08/shared-formula-memory-test.ods

Shared string re-work

- String comparisons were slow
  - Also not tractable for a GPU
  - Case-insensitive equality is a **hard** problem – ICU & heavy lifting.
- String comparisons a lot in functions, and Pivot Tables.
- Shared string storage is useful.
- So fix it ...

Concept


svl::SharedStringPool
svl::SharedString
svl::SharedString
svl::SharedString
Original string pool
Upcased string pool

String comparison (old way)

```
utl::TransliterationWrapper* pTransliteration = NULL;
OUString aStr1, aStr2;

if (bCaseSensitive)
    // Case sensitive transliterator.
    pTransliteration = ScGlobal::GetCaseTransliteration();
else
    // Case insensitive transliterator.
    pTransliteration = ScGlobal::GetpTransliteration();

// Parse both strings to check equality.
bool bEqual = pTransliteration->isEqual(aStr1, aStr2);
```


String comparison (new way)

```
svl::SharedString aStr1, aStr2;

const rtl_uString* p1;
const rtl_uString* p2;

if (bCaseSensitive)
{
    // Get pointers to original strings in the pool.
    p1 = aStr1.getData();
    p2 = aStr2.getData();
}
else
{
    // Get pointers to upcased strings in the pool.
    p1 = aStr1.getDataIgnoreCase();
    p2 = aStr2.getDataIgnoreCase();
}

// Compare pointer values.
bool bEqual = p1 == p2;
```


OpenCL / calculation ...

Why OpenCL & HSA ...

- GPU and CPU optimisation …
  - Why write custom SSE2/SSE3 etc. assembly detect arch, and select backend cross platforms.
  - Instead get OpenCL (from APU vendor) to generate the best code ...
- Hetrogenous System Architecture rocks:
  - An AMD64 like innovation:
  - shared Virtual Memory Address space & pointers: GPU ↔ CPU.
  - Avoid wasteful copies, fast dispatch
  - Great OpenCL 2.0 support.
  - Use the right Compute Unit for the job.

HSA FOUNDATION

Auto-compile Formula → OpenCL

```
#pragma OPENCL EXTENSION cl_khr_fp64: enable
int isNan(double a) { return isnan(a); }
double legalize(double a, double b) { return isNan(a)?b:a;}
double tmp0_0_fsum(__global double *tmp0_0_0)
{
    double tmp = 0;
    {
    int i;
    i = 0;
    tmp = legalize(((tmp0_0_0[i])+(tmp)), tmp);
    i = 1;
    tmp = legalize(((tmp0_0_0[i])+(tmp)), tmp);
    i = 2;
    tmp = legalize(((tmp0_0_0[i])+(tmp)), tmp);
    } // to scope the int i declaration
    return tmp;
}
double tmp0_nop(__global double *tmp0_0_0)
{
    double tmp = 0;
    int gid0 = get_global_id(0);
    tmp = tmp0_0_fsum(tmp0_0_0);
    return tmp;
}
__kernel void DynamicKernel_nop_fsum(__global double *result, __global double
*tmp0_0_0)
{
    int gid0 = get_global_id(0);
    result[gid0] = tmp0_nop(tmp0_0_0);
}
```

	A	B	C
1	=SUM($B$1:$B$3)	1	3
2		2	2
3		3	1

Formulae compiled idly / on entry in a thread … to hide latency.

Kernel generation thanks to:

MULTICORE WARE

The same formula for a longer sum …

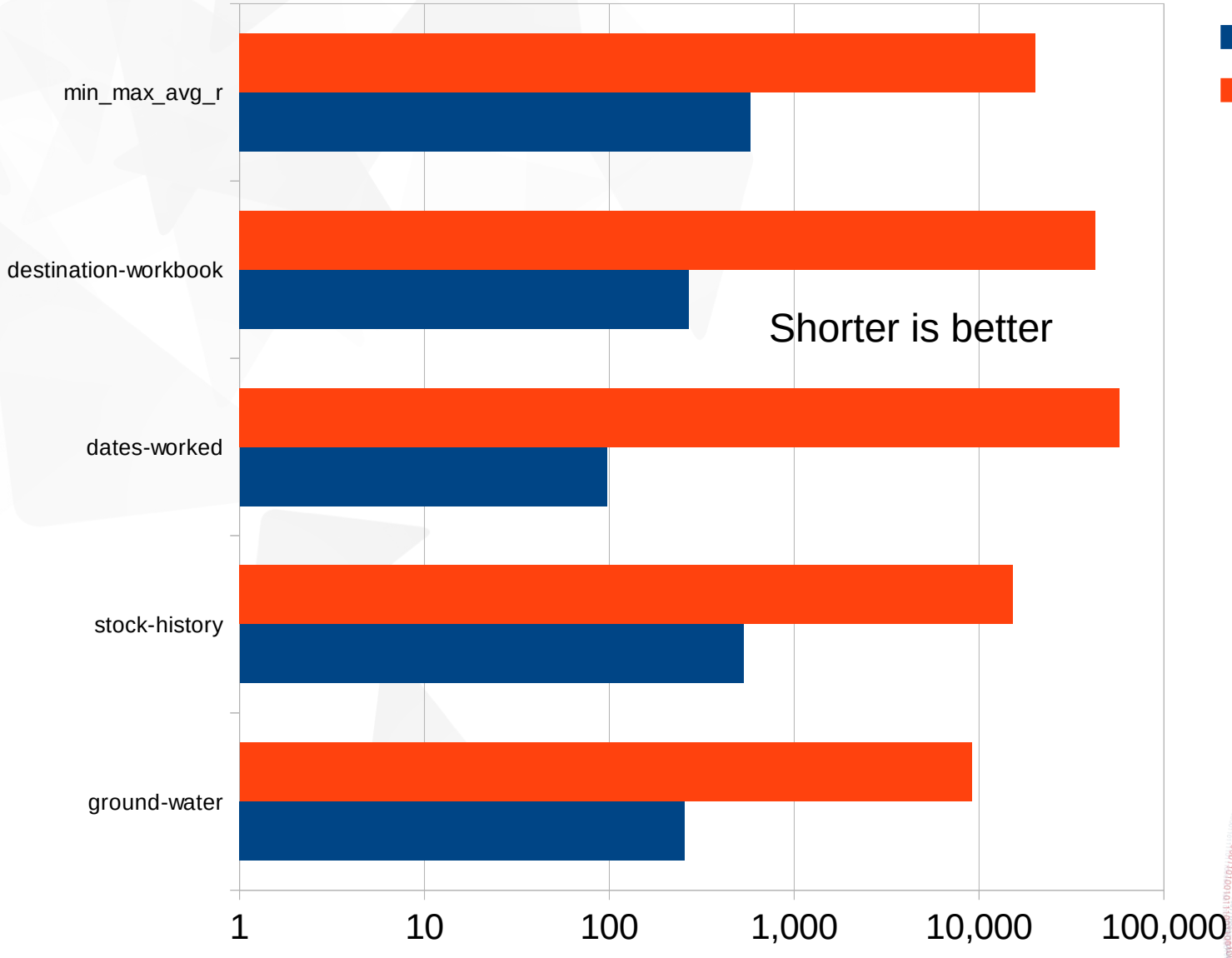
Compiled from standard formula syntax

```
__kernel void
tmp0_0_0_reduction(__global double* A,
                   __global double *result,
                   int arrayLength, int windowSize)
{
    double tmp, current_result =0;
    int writePos = get_group_id(1);
    int lidx = get_local_id(0);
    __local double shm_buf[256];
    int offset = 0;
    int end = windowSize;
    end = min(end, arrayLength);
    barrier(CLK_LOCAL_MEM_FENCE);
    int loop = arrayLength/512 + 1;
    for (int l=0; l<loop; l++) {
        tmp = 0;
        int loopOffset = l*512;
        if((loopOffset + lidx + offset + 256) < end) {
            tmp = legalize(((A[loopOffset + lidx + offset])+
(tmp)), tmp);
            tmp = legalize(((A[loopOffset + lidx + offset +
256])+(tmp)), tmp);
        } else if ((loopOffset + lidx + offset) < end)
            tmp = legalize(((A[loopOffset + lidx + offset])+
(tmp)), tmp);
        shm_buf[lidx] = tmp;
        barrier(CLK_LOCAL_MEM_FENCE);
        for (int i = 128; i >0; i/=2) {
            if (lidx < i)
                shm_buf[lidx] = ((shm_buf[lidx])+
(shm_buf[lidx + i]));
            barrier(CLK_LOCAL_MEM_FENCE);
        }
        if (lidx == 0)
            current_result =((current_result)+(shm_buf[0]));
        barrier(CLK_LOCAL_MEM_FENCE);
    }
    if (lidx == 0)
        result[writePos] = current_result;
}
```

```
double tmp0_0_fsum(__global double
*tmp0_0_0) {
    double tmp = 0;
    int gid0 = get_global_id(0);
    tmp = ((tmp0_0_0[gid0])+(tmp));
    return tmp;
}
double tmp0_nop(__global double
*tmp0_0_0) {
    double tmp = 0;
    int gid0 = get_global_id(0);
    tmp = tmp0_0_fsum(tmp0_0_0);
    return tmp;
}
__kernel void
DynamicKernel_nop_fsum(__global double
*result,

__global double *tmp0_0_0)
{
    int gid0 = get_global_id(0);
    result[gid0] = tmp0_nop(tmp0_0_0);
}
```

Performance numbers for sample sheets.



GPU / OpenCL

Software

30x → 500x faster for these samples vs. the legacy software calculation

on Kaveri.

Yet another log plot … milliseconds on the X axis ...

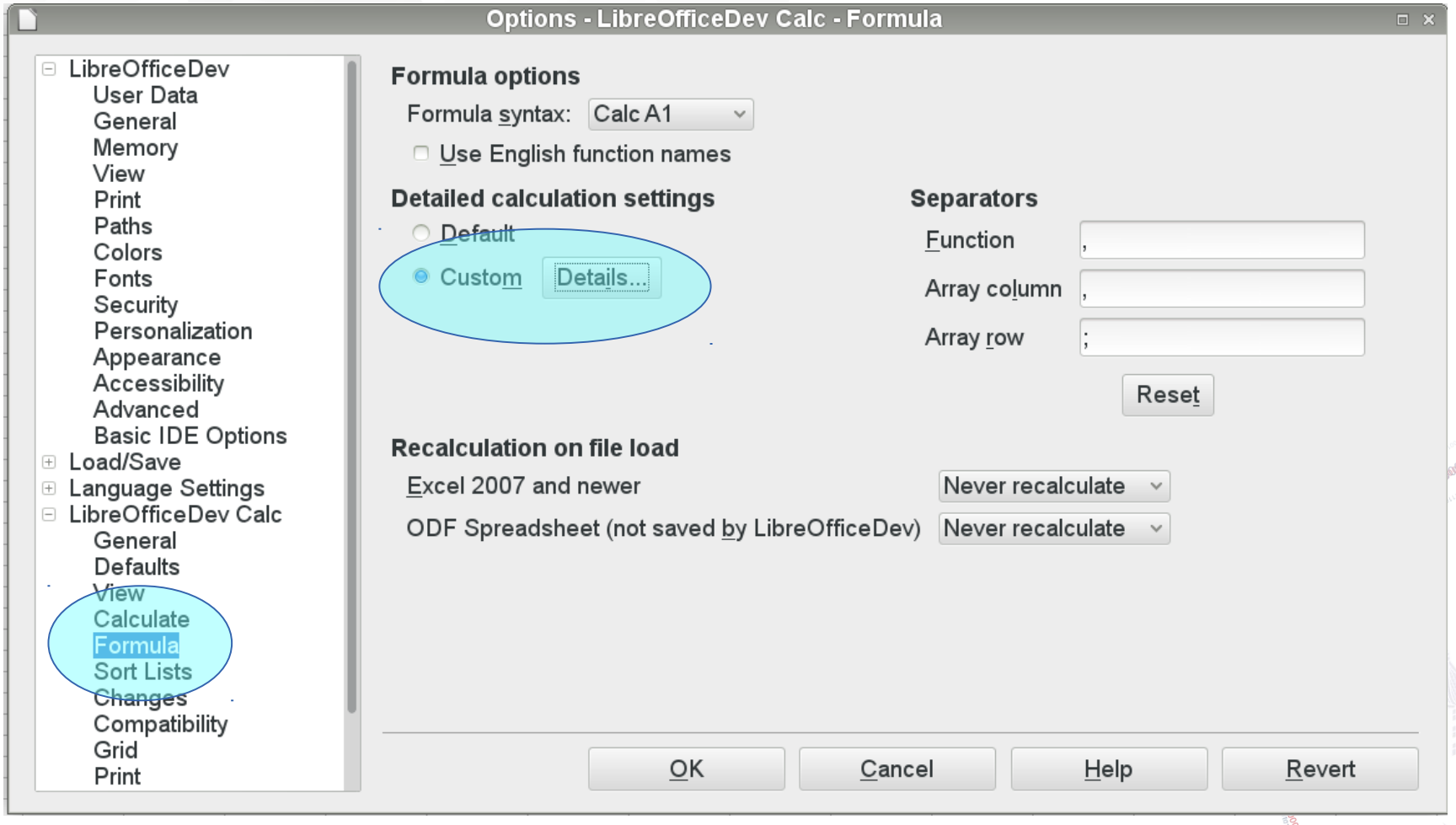
In more detail ...

- This is a spreadsheet
  - What do you mean what is the X factor ?
- Highly spreadsheet geometry dependent
  - Don't like your X factor – add more rows, or complexity.
- Representative sheets important – some based on real-world madness …
- Functions:
  - Research shows vast majority of distinct fomulae have very simple functions: SUM, AVERAGE, SUMIF, VLOOKUP, etc.
    - We optimise those
  - We don't do eg. Text functions like UPPER

How that works in practise:

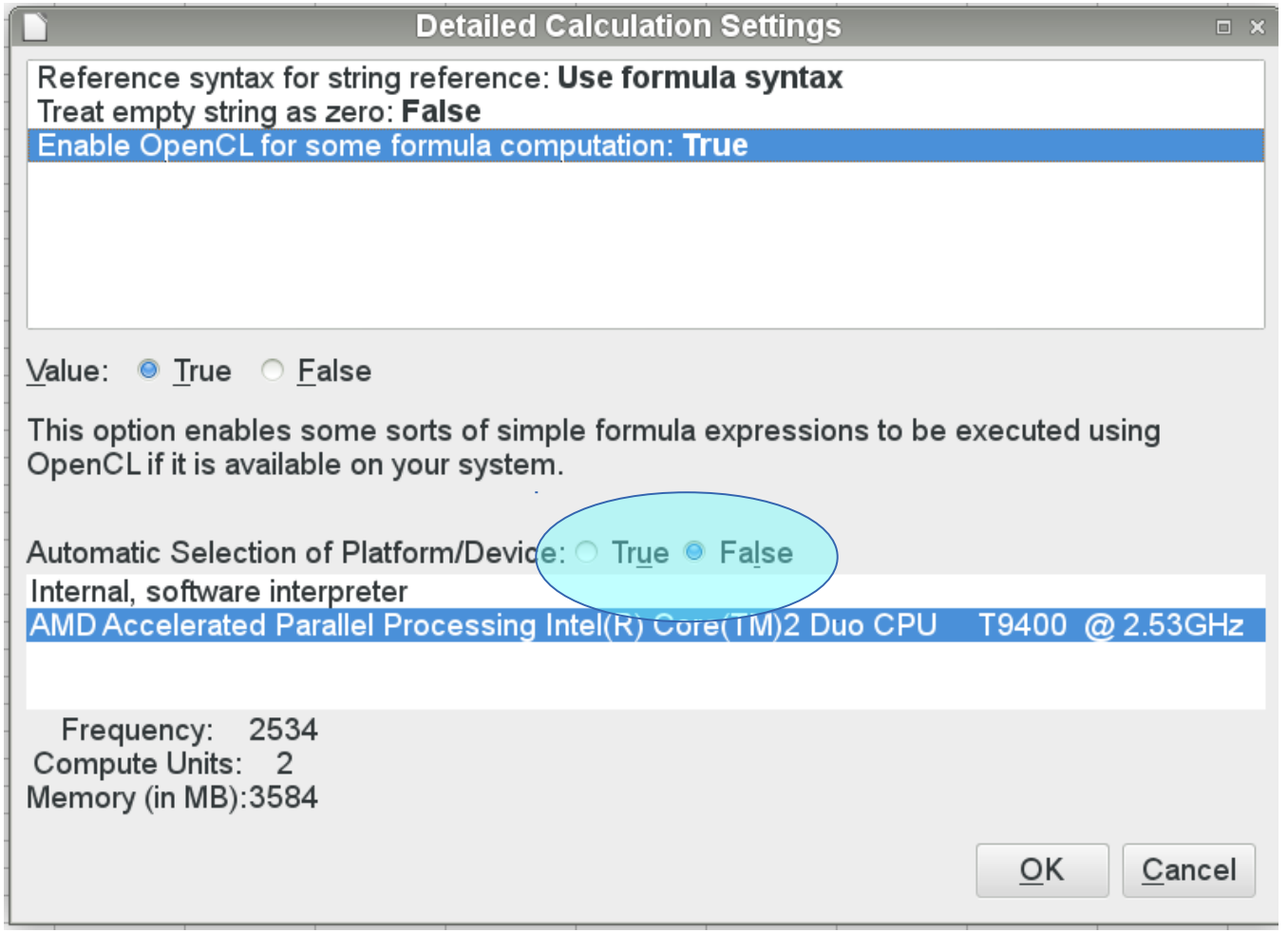
Enabling Custom Calculation

- Turn on OpenCL computation: **Tools → Options**



Enabling OpenCL goodness

- Auto-select the best OpenCL device via a micro-benchmark
  - Or disable that and explicitly select a device.



Big data needs Document Load optimization

Parallelized Loading ...

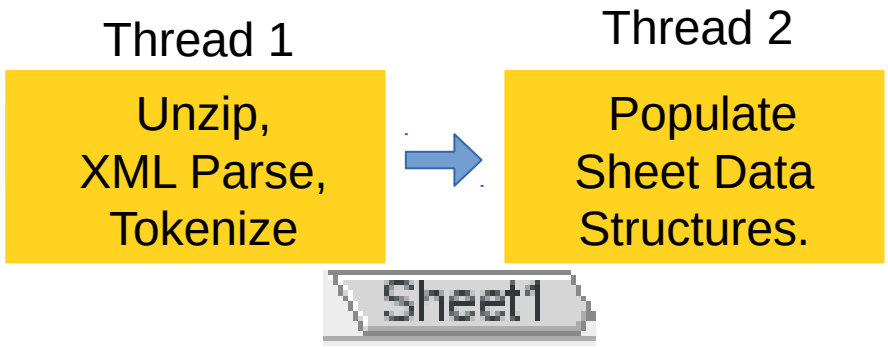
- Desktop CPU cores are often idle.
- XML parsing:
  - The ideal application of parallelism
  - SAX parsers:

    "**S**ucking ic**A**che e**X**perience" parsers

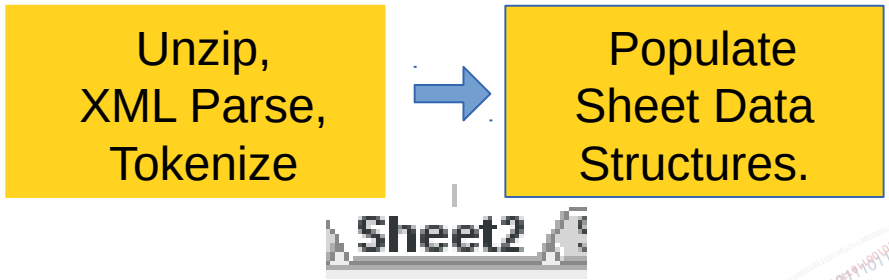
    - read, parse a tiny piece of XML & emit an event … punch that deep into the core of the APP logic, and return ..
    - Parse another tiny piece of XML.
  - Better APIs and impl's needed: Tokenizing, Namespace handling etc.
  - Luckily easy to retro-fit threading ...
  - Dozens of performance wins in XFastParser.

Utilising your 32 core CPU …
(boxes are threads).

- Split XML Parse & Sheet populate



- Parallelised Sheet Loading …

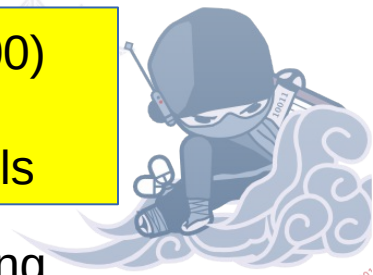


Progress bar thread

… etc.

- Parallel to GPU compilation

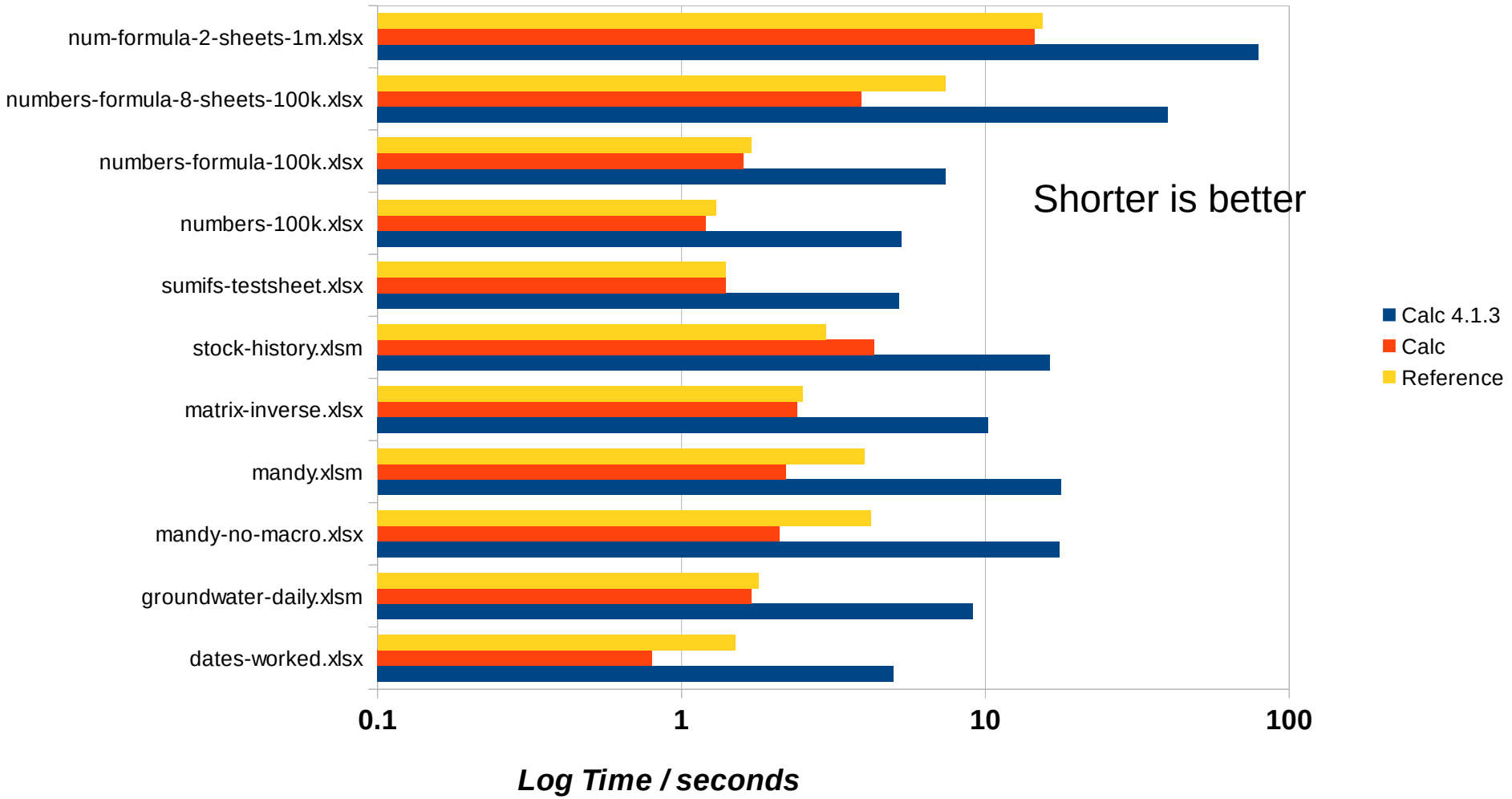
=COVAR(A1:A300,B1:B300)
→ OpenCL code
→ Ready to execute kernels



Tools->Options->Advanced->"Experimental Mode" required for parallel loading

Does it work ? with GPU enabled

Wall-clock time to load set of large XLSX spreadsheets: 8 thread Intel machine



Apologies for another log scale: **Average 5X vs. 4.1.3**

How does that pan out ?

Problems^W Opportunities …

- Picking a good OpenCL driver
  - White / Black / Any listing of known good / bad / mixed Hardware / Driver / OS …
- Which core to pick ?
  - fp64 perf etc. Time vs. Power
  - Currently micro-benchmark time.
- HSA rocks
  - CL_MEM_USE_HOST_PTR is a royal pain:
    - Alignment issues currently cause lots of copying in several cases.
  - OpenCL 2.0's Shared Virtual Memory is awesome
- Compiler Performance:
  - Excel RPN → C string → IR → GPU
  - SPIR sounds great – if it can be stable.

Future OpenCL work ...

- Volunteers / funders welcome
- Kill per-cell dependency graphing
  - Badly needs to be per-column:
    - Shrink memory usage, improve load time
    - Detect independent column calculations
      - Enabling parallel execution, wider CSE etc.
- SPIR integration
  - Avoid 'NaN' foo by adapting to data shape faster.
- Calc as a flow process, 'construct your pipeline in a sheet'
- Crazy awesome demos: Mobile vs. PC ...
- ZIP – LZ 77 / OpenCL acceleration … or similar

LibreOffice Conclusions

- **LibreOffice is innovating:**
  - Going interesting places no-one has gone before:
    - OpenCL in a generic spreadsheets a first
    - Why write 5x hand-coded assembler versions and select per platform.
      - there is already a tool for that.
  - Run your workload on the right Compute Unit to save time & battery.
- **Re-factoring for OpenCL improves performance for all**
  - Faster for CPU and GPU
  - PCMark 8.2 includes LibreOffice → benchmarking.
- **LibreOffice loves new contributor & features**
  - Talk to me about getting involved ...
- **Thanks for all of your help and support !**

Oh, that my words were recorded, that they were written on a scroll, that they were inscribed with an iron tool on lead, or engraved in rock for ever! I know that my Redeemer lives, and that in the end he will stand upon the earth. And though this body has been destroyed yet in my flesh I will see God, I myself will see him, with my own eyes - I and not another. How my heart yearns within me. - Job 19: 23-27