Reducing CPU & memory use

Go faster stripes everywhere

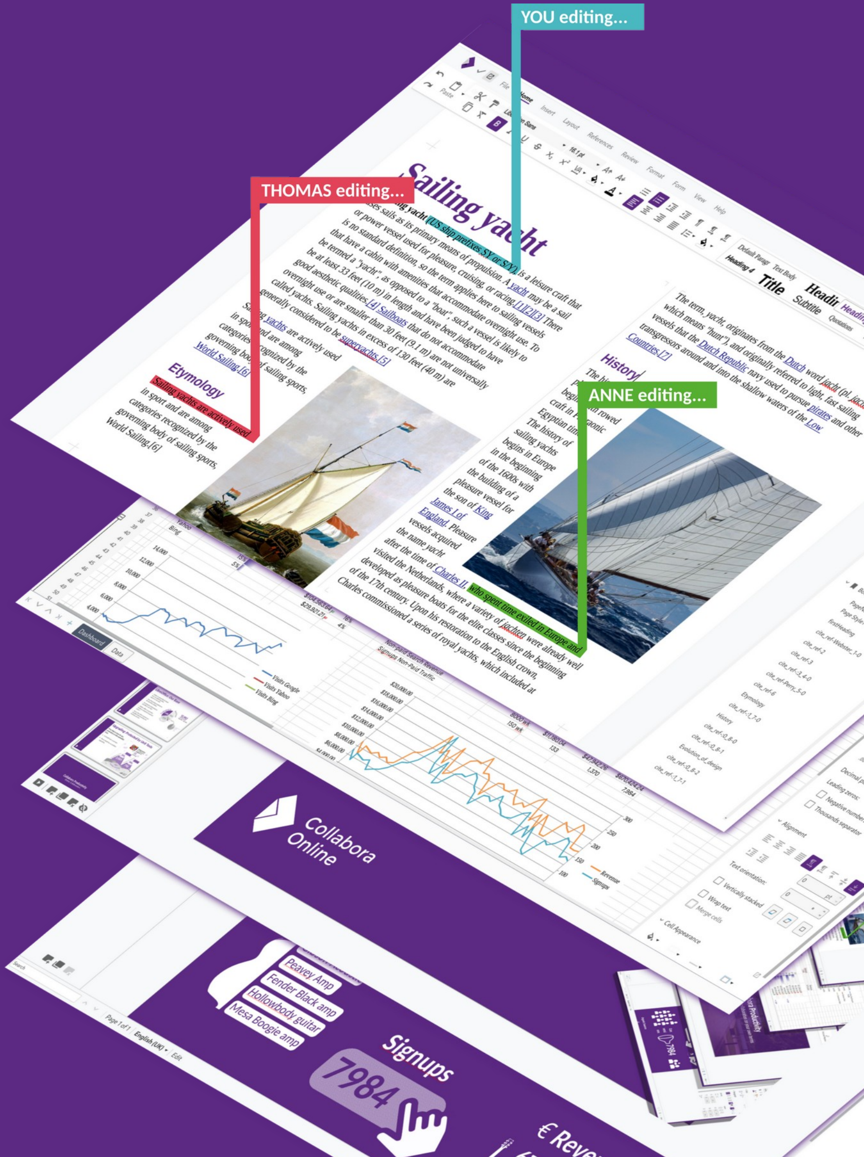
Michael Meeks
CEO
michael.meeks@collabora.com

Collabora Online

LibreOffice Technology

Technical Day

COOL days



Always profile
before optimizing !

Technical Day COOL days

Demo profiling: example flame-graph

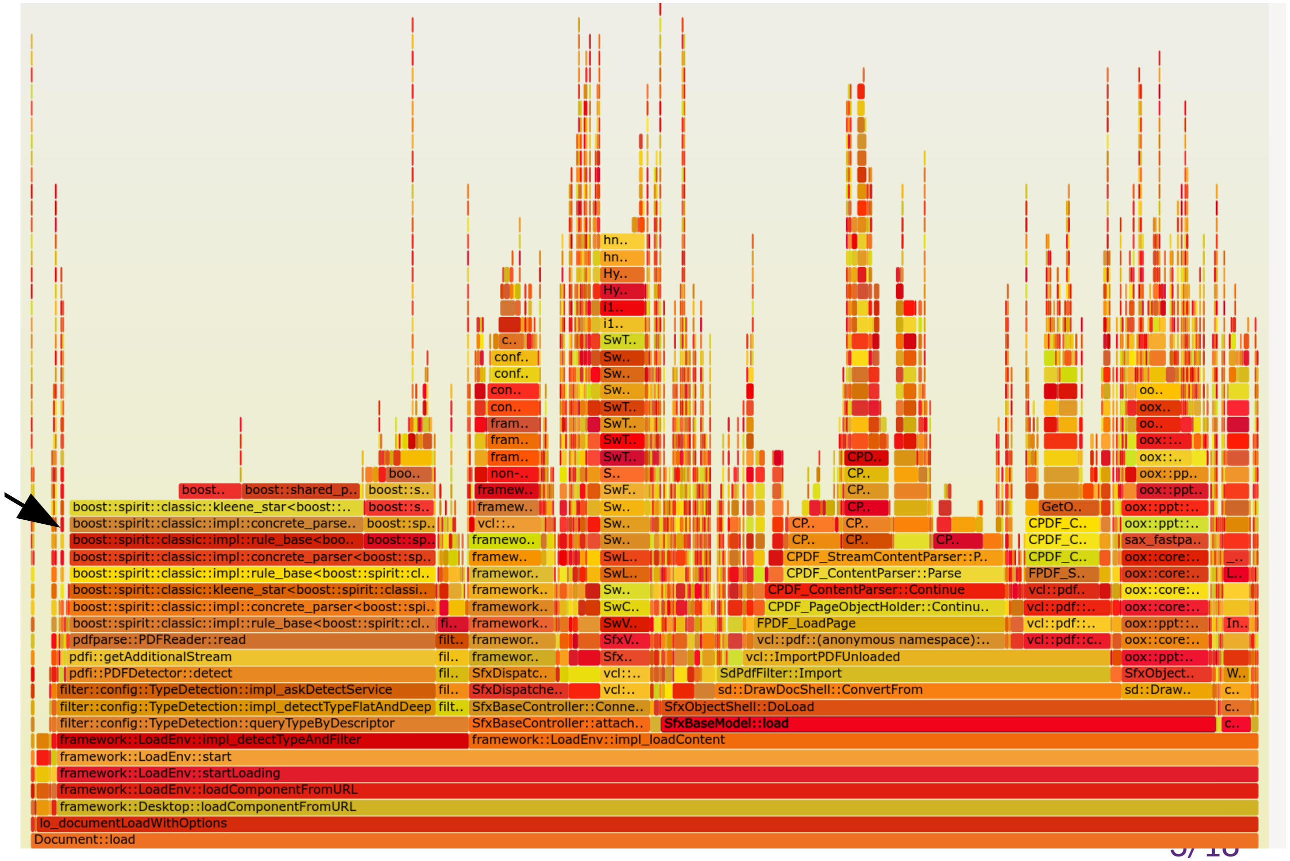
What does it mean ?

17%+ of a week of profiling:

- Detecting: "is it Hybrid PDF"?
- Unbelievably wasteful
- Scan last 4k block 'AdditionalStreams' +17%

Width is proportional to CPU time (not wall-clock)

Stack of function callers



Collabora Online

Run Length Encoding / RLE

Tile Deltas cache / optimization

Store previous tiles

- So we can delta them
- Previously – generated row CRC while copying & kept all pixels
- Now use RLE bitmask.
- Substantially compressed: 256x256x4 → 256 kB
- RLE compressed: < 26 Kb
  - **10x size win**
- **90** tile cache (per view) vs. **24** (per view)
  - Plus **~2Mb per view** size saving.

RLE DeltaBitmapRow:

```
uint64_t   _rleMask[4];
size_t     _rleSize;
uint32_t *_rleData;
```

- Split mask bits from Data
- _rleMask bit-set – ‘1’
  - copy previous pixel 0 default transparent
- Compare: **No need for a hash**: just compare _rleSize & _rleMask.

The magic of AVX2 – branch free RLE

prepend ‘last’ pixel of the last block

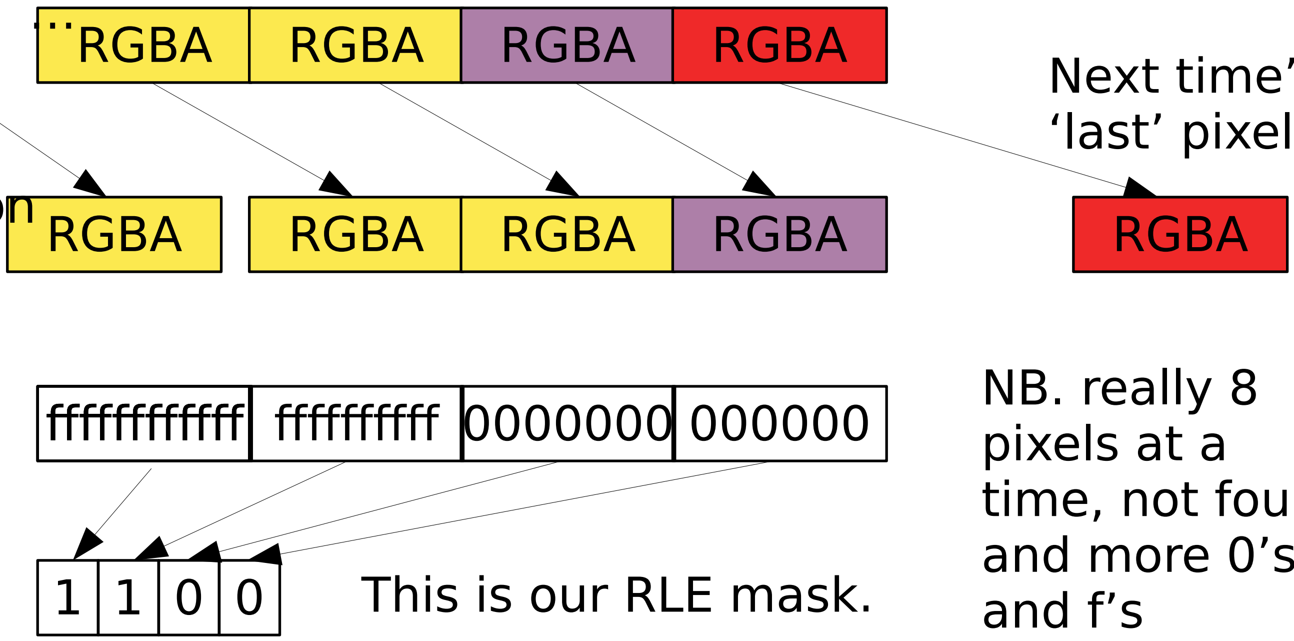
Block of next four pixels to RLE

Next time’s ‘last’ pixel:

Build comparison register:

Compare: cmpeq_epi32

Magic: floating point sign mask: movemask_ps



This is our RLE mask.

NB. really 8 pixels at a time, not four and more 0’s and f’s needed …

How many pixels to copy? popcount(RLE mask)
Which ones ? RLE Mask → LUT + AVX2 gather
permutevar8x32_epi32 … is your friend

Performance win – around 4.5x …

Benchmark repo with pretty documents & pre-rendered tiles:

- https://github.com/CollaboraOnline/benchmark

Simple, built in RLE benchmark tool:

```
./coolbench /opt/libreoffice/benchmark/*/*.png
Benchmark CPU - best hand-optimized CPU RLE code
took: 5780ms - time/rle: 115.616us
Benchmark SIMD - best (so far) AVX2 optimized code
took: 1206ms - time/rle: 24.1266us
```

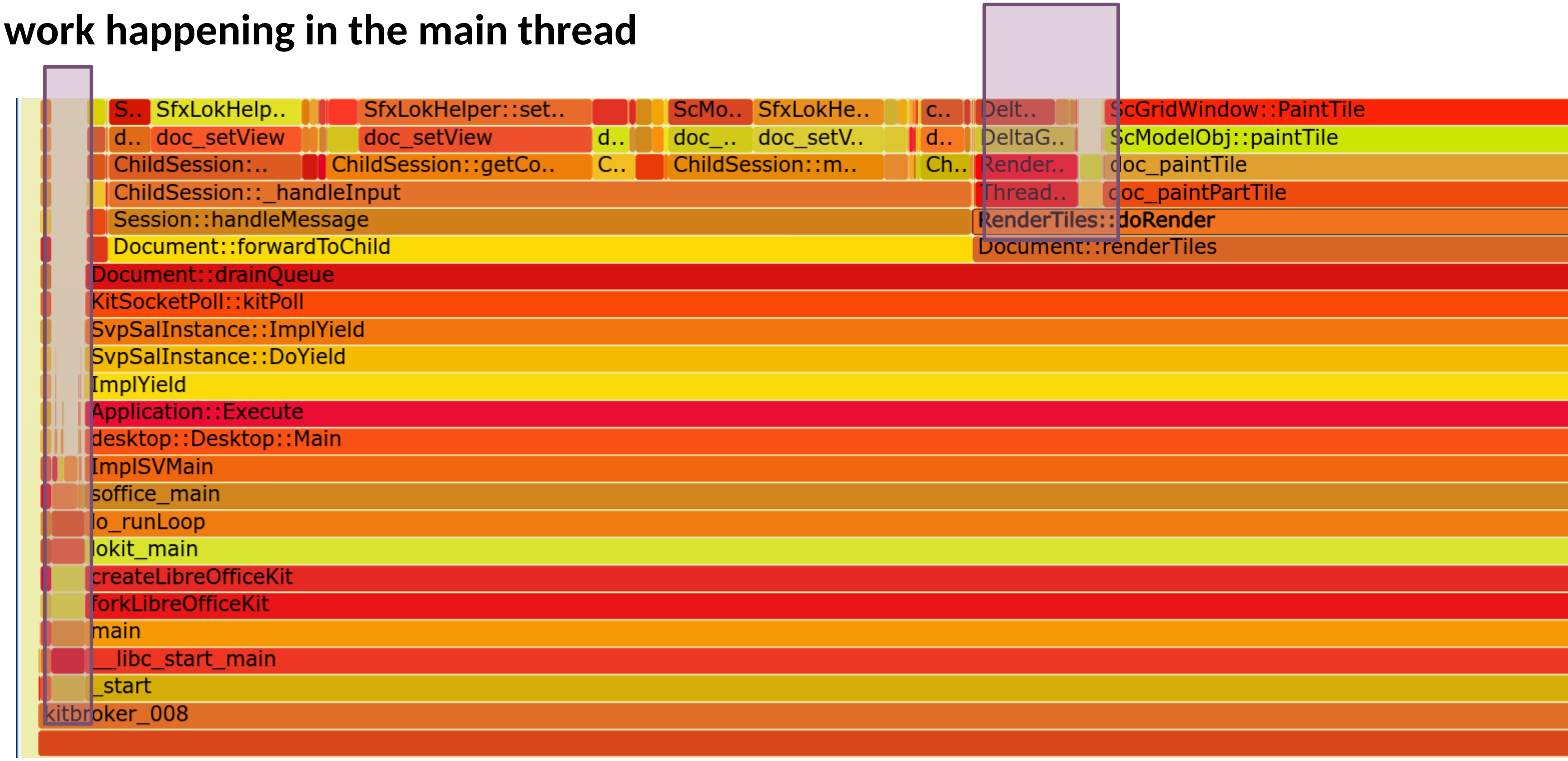
- **100 tiles → 2.5ms not long.**

Getting threading right:

```
- if (!_shutdown && !_work.empty())
+ while (!_shutdown && !_work.empty())
```

perf: surprisingly little delta threading

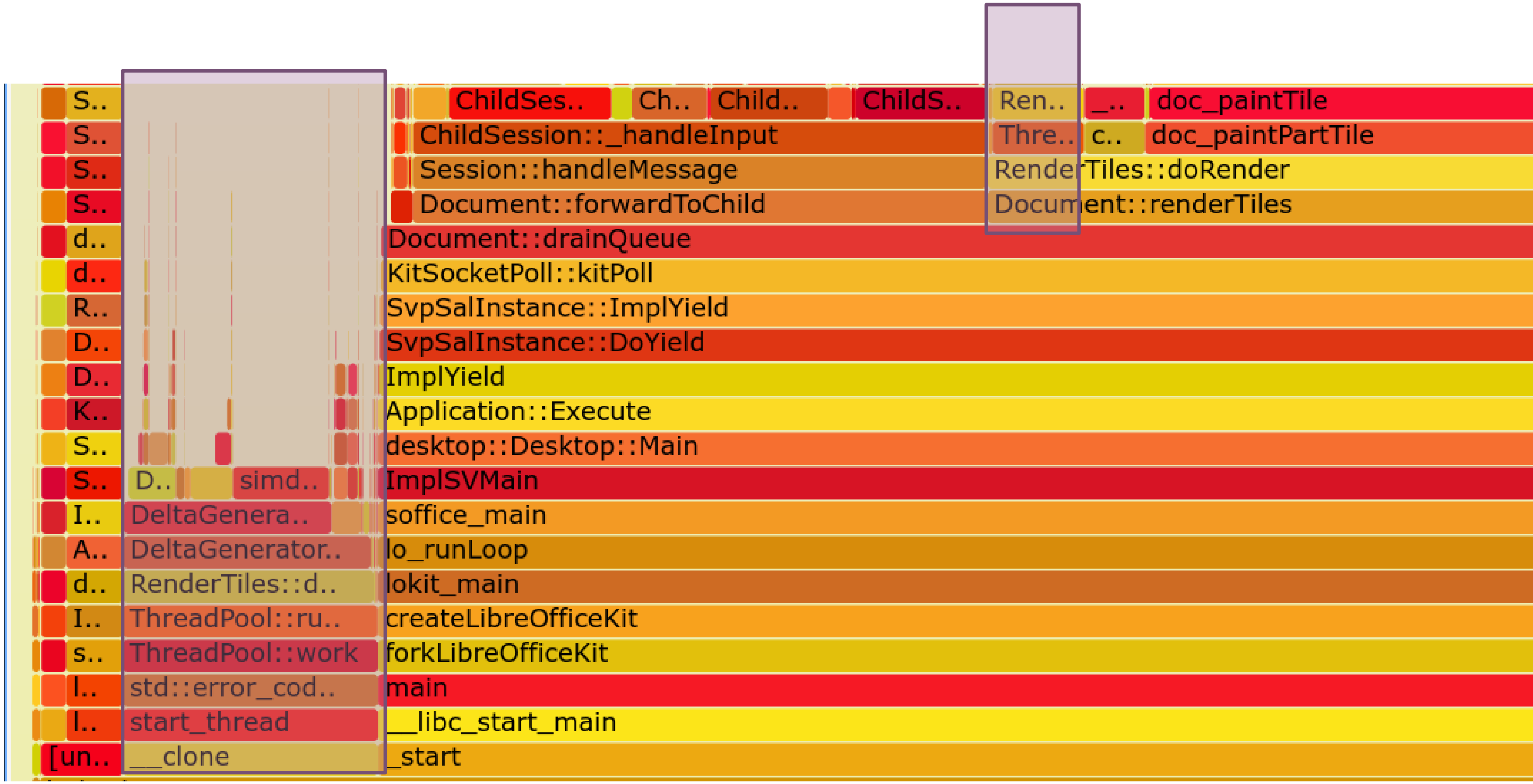
Thread default 4x wide on deltas ... - but all the work happening in the main thread



Technical Day COOL days

perf: surprisingly little delta threading

An ‘if (work)’ instead of ‘while (work)’ 1 line fix 4x latency redux in delta’ing.



Collabora Online

Convolve the two:

4x x 4.5x ~= 18x

RGBA & pre-multiplied alpha

Documents rendered to an alpha surface

- Pre-multiplied the sensible way to go so of course:
  - HTML5 canvas API – not pre-multiplied
  - HTML5 canvas implementation – pre-multiplied [!]
    - cf. complaints about not getting back RGBA you put into it ...

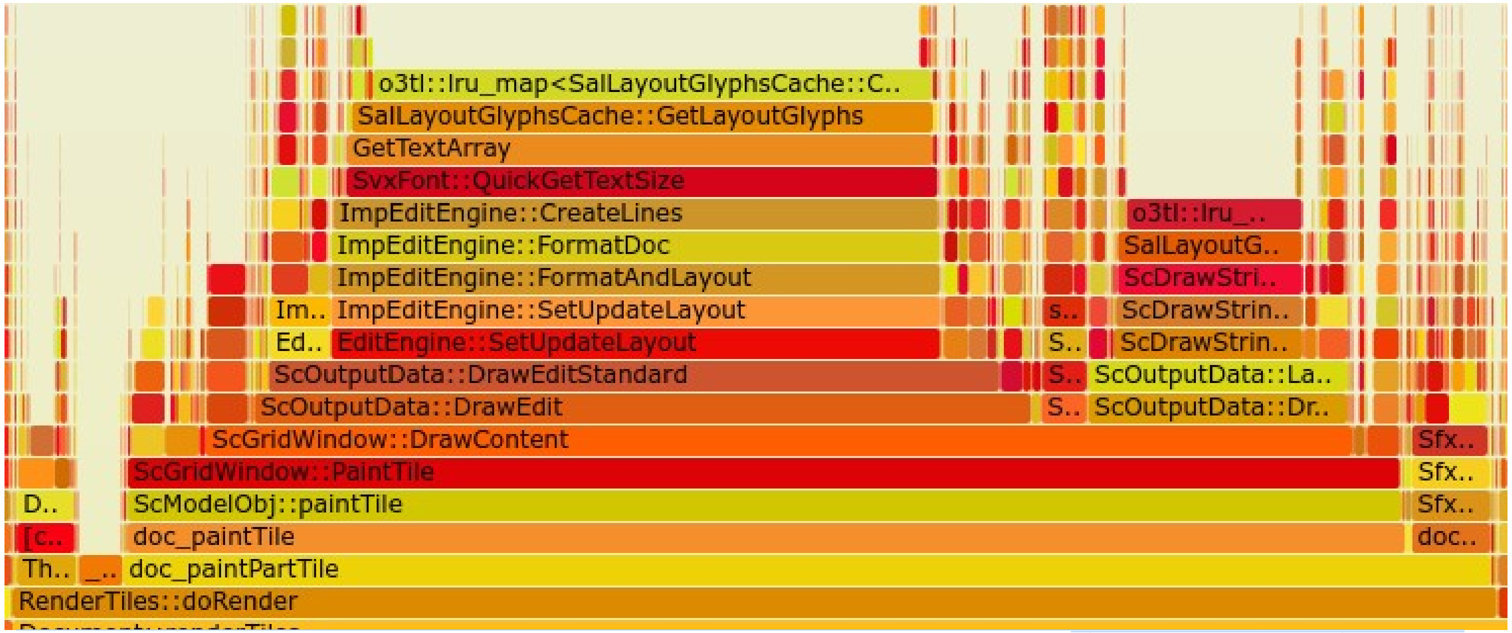
Change the approach and win

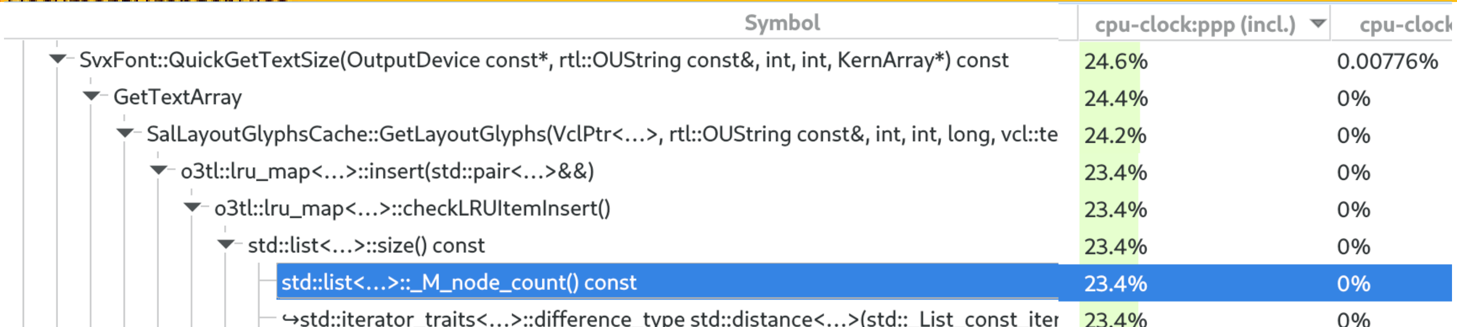
- COOL → un-premultiply → **net** → canvas API → re-pre-multiply → graphics
- COOL → **net** → un-premultiply → canvas API → re-pre-multiply → graphics
  - Leave the web's problems to the browser JIT.
- Also RGBA support to Cairo from libpixman, to avoid BGRA conversion.

Technical Day

COOL days

LRU: std::list::size as std::distance()





Symbol	cpu-clock:ppp (incl.)	cpu-clock
SvxFont::QuickGetTextSize(OutputDevice const*, rtl::OUString const&, int, int, KernArray*) const	24.6%	0.00776%
GetTextArray	24.4%	0%
SalLayoutGlyphsCache::GetLayoutGlyphs(VclPtr<...>, rtl::OUString const&, int, int, long, vcl::te	24.2%	0%
o3tl::lru_map<...>::insert(std::pair<...>&&)	23.4%	0%
o3tl::lru_map<...>::checkLRUItemInsert()	23.4%	0%
std::list<...>::size() const	23.4%	0%
std::list<...>::_M_node_count() const	23.4%	0%
↪std::iterator_traits<...>::difference_type std::distance<...>(std::_List_const_iter	23.4%	0%

Collabora Online

Technical Day COOL days

Collabora Online

Kill paint to a giant virtual device

Older Writer rendering path:

```
//Refresh with virtual device
  to avoid flickering.
VclPtrInstance<VirtualDevice>
pVout( *mpOut );
pVout->SetMapMode( mpOut-
>GetMapMode() );
Size aSize( VisArea().SSize() );
aSize.AdjustWidth(20);
aSize.AdjustHeight(20);
if( pVout->SetOutputSize( aSize ) )
```

Un-necessary PC 'flicker reduction' optimization

- We push tiles to JS for a flicker-free scroll/zoom anyway.

Giant / whole document area

- Plus a bit.

Back that with lots of memory & do lots of rendering into it

Memory use

Lots of space (& time) saving:

Discovered a lurking benchmark

- Allocate 64Mb of RAM, and performing a CPU rendering benchmark before loading each document …
- Good to get initial dirty-page count down to ~20Mb in one line.

Image caching

- Compressed images are small!
  - Not so TIFFs → swap them.
- Cache & Images & glibc allocator trim on idle → mobile-phone style.

Sparse documents:

- Calc – file save used to allocate all 16k columns – making many things slower.
- Calc – discourage users to leap to limits of document

$ make run-inproc

- Run under massif / valgrind as a single process in the build-tree …
- Avoiding real-CPU timing jitter:
  - flat profiles for no change … vital.

Future / Ongoing work:

Lots more …

- Even faster tile / RLE:
  - AVX512
  - Aligned memory for faster load/store
  - Pre-fetching
- Keep profiling …
  - and improving what we see.

Thank you!

By Michael Meeks

T S P D

@CollaboraOffice
hello@collaboraoffice.com
www.collaboraoffice.com

Collabora Online

LibreOffice Technology

Technical Day COOL days