org-special-block-extras

Musa Al-hassy

April 17, 2020

Contents

1 Example Use

User type something along the lines of the following.

```
#+begin_red org
/This/
      *text*
            _is_
                 red!
#+end_red
```

Which generates *red* text when exported to HTML and LaTeX, **while supporting Org markup**.

This article may be read as a PDF or as HTML or as pure Org!

(Since we're only considering HTML & LaTeX, the Github rendition of this article may not render things as desired.)

The remaining sections are implementation matter.

Figure 1: Write Org-markup once, generate for many backends ^_^

2 Core Utility

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Core utility

(defun org-special-block-extras--advice (backend blk contents _)
  "Invoke the appropriate custom block handler, if any.

A given custom block BLK has a TYPE extracted from it, then we
send the block CONTENTS along with the current export BACKEND to
the formatting function ORG-SPECIAL-BLOCK-EXTRAS/TYPE if it is
defined, otherwise, we leave the CONTENTS of the block as is.

We also support the seemingly useless blocks that have no
contents at all, not even an empty new line."
  (let* ((type    (nth 1 (nth 1 blk)))
         (handler (intern (format "org-special-block-extras--%s" type))))
    (ignore-errors (apply handler backend (or contents "") nil))))

(advice-add #'org-html-special-block :before-until
            (-partial #'org-special-block-extras--advice 'html))

(advice-add #'org-latex-special-block :before-until
            (-partial #'org-special-block-extras--advice 'latex))
```

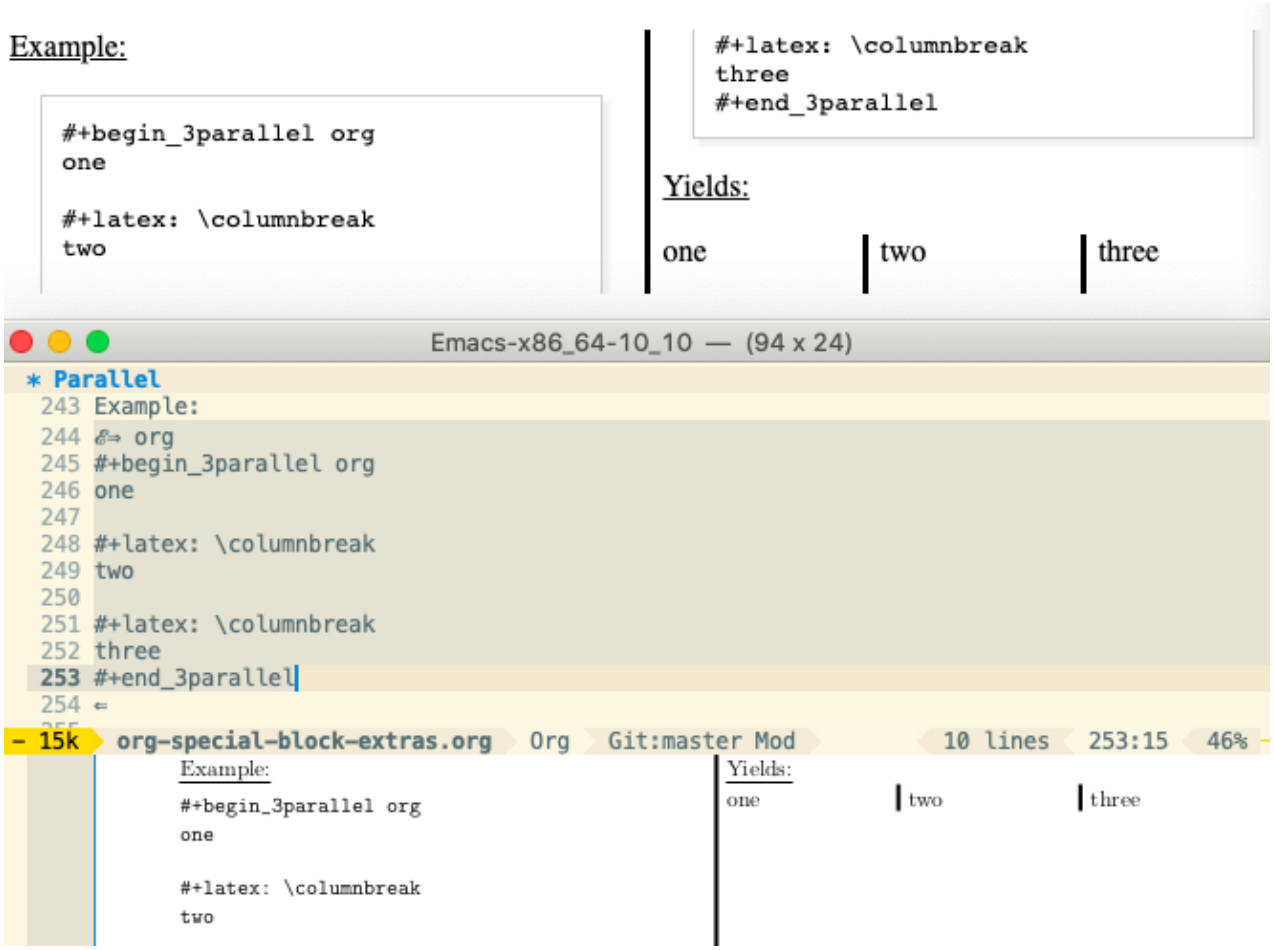

Figure 2: Displaying thoughts side-by-side ^_^

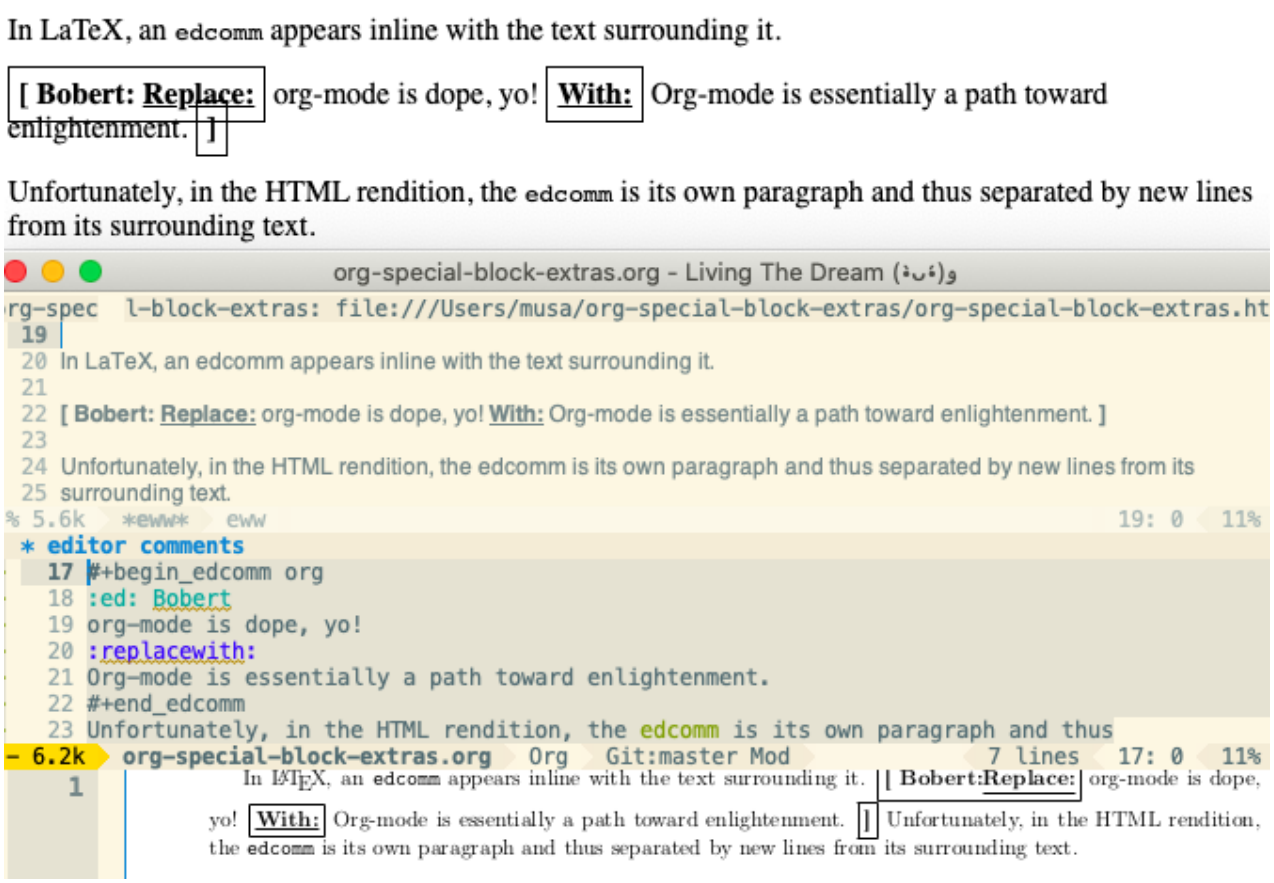

Figure 3: "First-class editor comments" In order: Chrome, Emacs Web Wowser, Org source, PDF

3 Colours

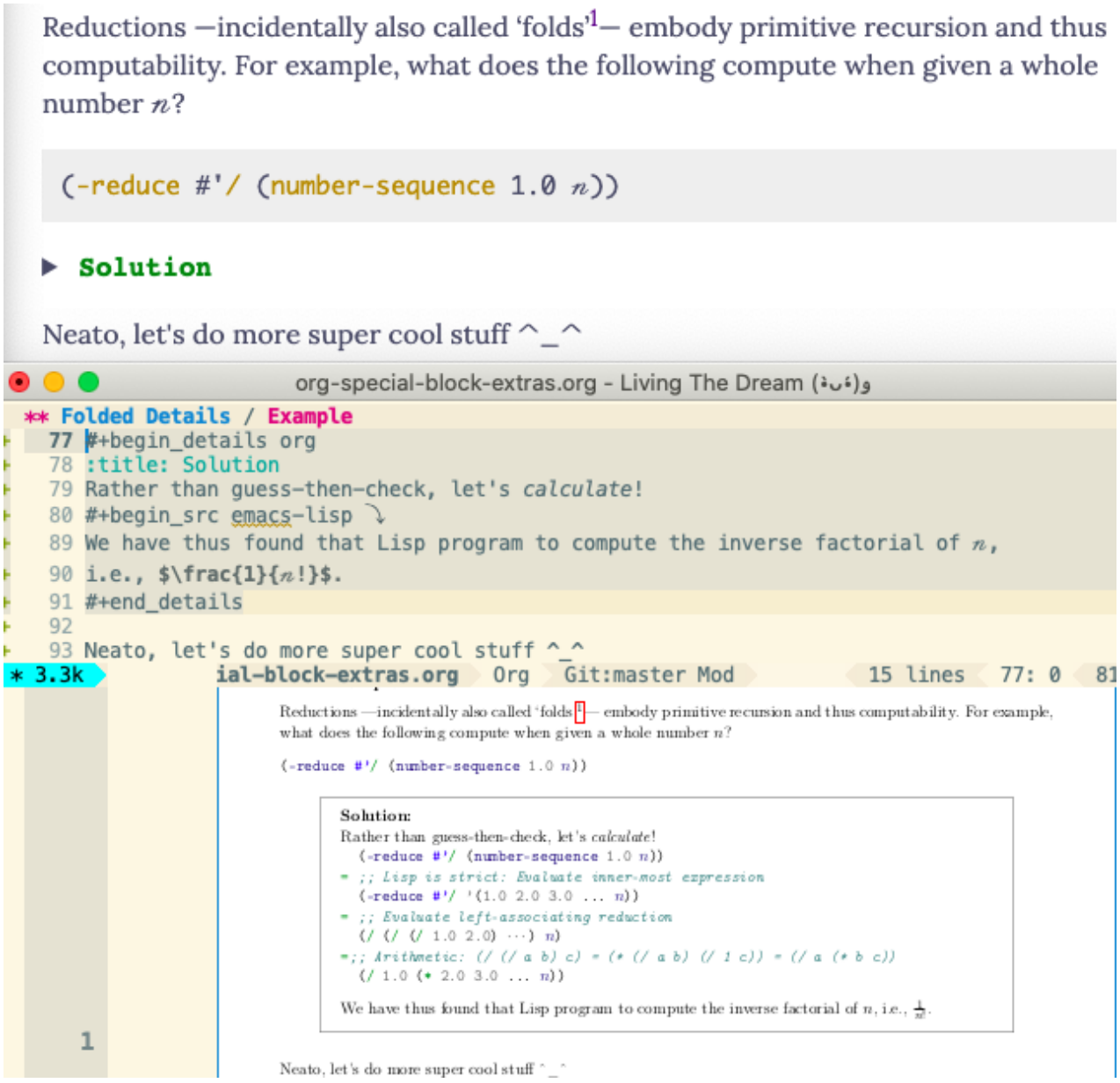

Figure 4: Visually hiding, folding away, details

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;; Load support for 19 colour custom blocks

(defvar org-special-block-extras--colors
  '(black blue brown cyan darkgray gray green lightgray lime
          magenta olive orange pink purple red teal violet white
          yellow)
  "Colours that should be available on all systems.")

(loop for colour in org-special-block-extras--colors
      do (eval (read (format
                      "(defun org-special-block-extras--%s (backend contents)
                     (format (pcase backend
                     (`latex \"\\\\begingroup\\\\color{%s}%%s\\\\endgroup\")
                     (`html  \"<div style=\\\"color:%s;\\\">%%s</div>\")
                     (t      \"org-special-block-extras: Unsupported backend\"))
                     contents))"
                      colour colour colour))))
```

3.1 Examples

This text is black!
This text is blue!
This text is brown!
This text is cyan!
This text is darkgray!
This text is gray!
This text is green!
This text is lightgray!
This text is lime!
This text is magenta!
This text is olive!
This text is orange!
This text is pink!
This text is purple!
This text is red!
This text is teal!
This text is violet!

This text is yellow!

4 Parallel

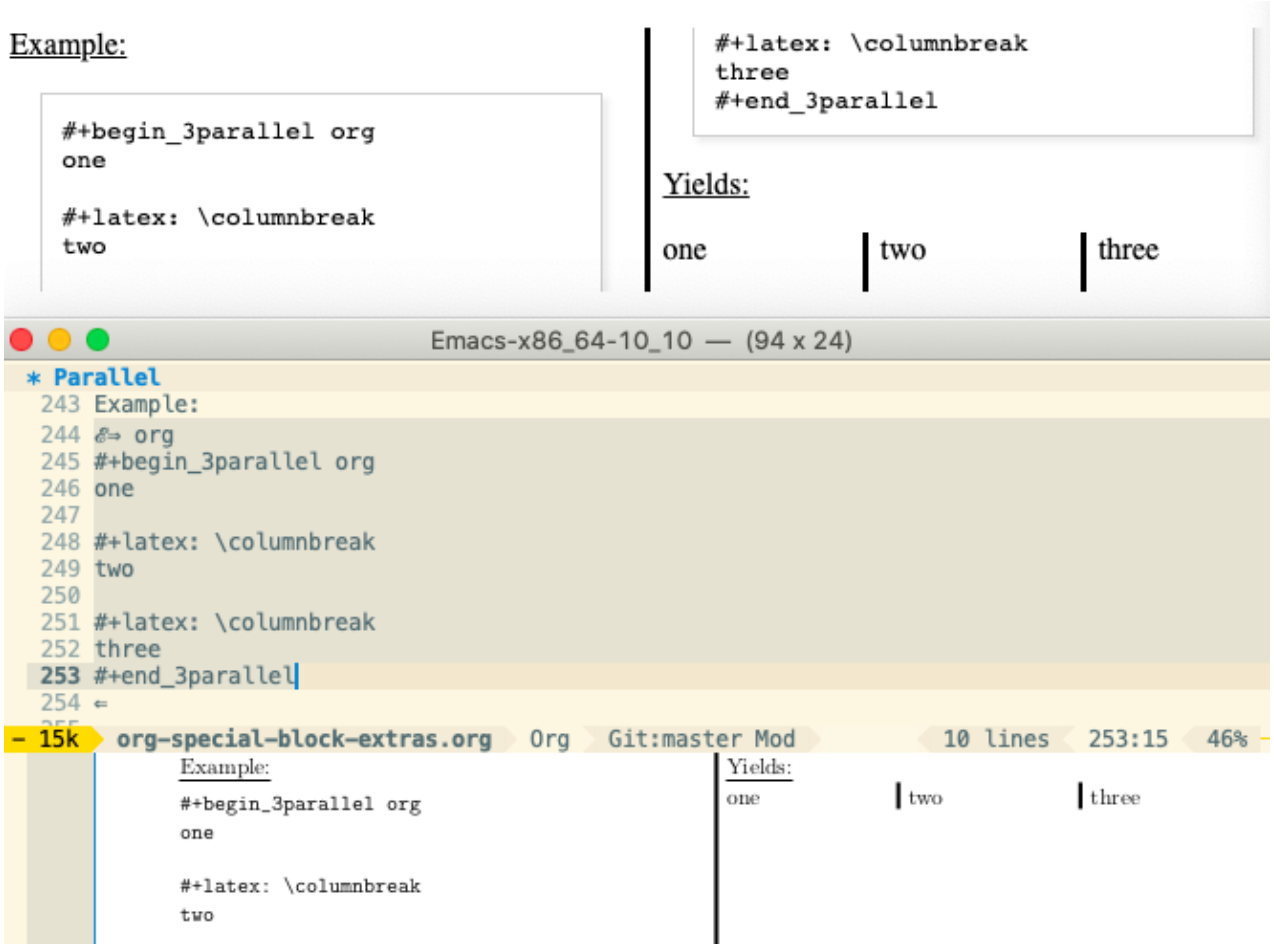

Figure 5: Displaying thoughts side-by-side ^_^

Example:

```
#+begin_3parallel org
one

#+latex: \columnbreak
two

#+latex: \columnbreak
three
#+end_3parallel
```

Yields:

one | two | three

```
#+LATEX_HEADER: \usepackage{multicol}
```

I initially used the names `paralell<n>` but names ending with a number did not inherit highlighting, so I shifted the number to being a prefix instead.

- For LaTeX, new lines are used to suggest opportunities for column breaks and are needed even if explicit columnbreaks are declared.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;; Parallel blocks: parallel<n>[NB] for n:2..5, optionally with 'N'o 'b'ar
;; in-between the columns.
;;
;; Common case is to have three columns, and we want to avoid invoking the
;; attribute via org, so making this.

(loop for cols in '("1" "2" "3" "4" "5")
      do (loop for rule in '("solid" "none")
      do (eval (read (concat
"(defun org-special-block-extras--" cols "parallel"
(if (equal rule "solid") "" "NB")
"(backend contents)"
"(format (pcase backend"
"('html \"<div style=\\\"column-rule-style:" rule ";column-count:" cols ";\\\"%s</div>\")"
"('latex \"\\\\par \\\\setlength{\\\\columnseprule}{" (if (equal rule "solid") "2" "0") "pt}"
"          \\\\begin{minipage}[t]{\\\\linewidth}"
"          \\\\begin{multicols}{" cols "}"
"          %s"
"          \\\\end{multicols}\\\\end{minipage}\")) contents))")))))

(defalias #'org-special-block-extras--parallel   #'org-special-block-extras--2parallel)
(defalias #'org-special-block-extras--parallelNB #'org-special-block-extras--2parallelNB)
```

(The Emacs Web Wowser, `M-x eww`, does not display `parallel` environments as desired.)

5 :argument: Extraction

```
(defun org-special-block-extras--extract-arguments (contents &rest args)
"Get list of CONTENTS string with ARGS lines stripped out and values of ARGS.

Example usage:
```

```
    (-let [(contents' . (&alist 'k_0 ... 'k_n))
           (...extract-arguments contents 'k_0 ... 'k_n)]
          body)

Within `body', each `k_i' refers to the `value' of argument
`:k_i:' in the CONTENTS text and `contents'' is CONTENTS
with all `:k_i:' lines stripped out.

+ If `:k:' is not an argument in CONTENTS, then it is assigned value NIL.
+ If `:k:' is an argument in CONTENTS but is not given a value in CONTENTS,
  then it has value the empty string."
  (let ((ctnts contents)
        (values (loop for a in args
                      for regex = (format ":%s:\\(.*\\)" a)
                      for v = (cadr (s-match regex contents))
                      collect (cons a v))))
    (loop for a in args
          for regex = (format ":%s:\\(.*\\)" a)
          do (setq ctnts (s-replace-regexp regex "" ctnts)))
    (cons ctnts values)))
```

6 Editor Comments

"Editor Comments" are intended to be top-level first-class comments in an article that are inline with the surrounding text and are delimited in such a way that they are visible but drawing attention. I first learned about this idea from Wolfram Kahl —who introduced me to Emacs many years ago.

In LaTeX, an `edcomm` appears inline with the text surrounding it. [**Bobert:Replace:** org-mode is dope, yo! **With:** Org-mode is essentially a path toward enlightenment.] Unfortunately, in the HTML rendition, the `edcomm` is its own paragraph and thus separated by new lines from its surrounding text.

Any new —possibly empty— inner lines in the `edcomm` *are desirably preserved*

```
(defvar org-special-block-extras-hide-editor-comments nil
  "Should editor comments be shown in the output or not.")

(defun org-special-block-extras--edcomm (backend contents)
"Format CONTENTS as an first-class editor comment according to BACKEND.

The CONTENTS string has two optional argument switches:
1. :ed: ⇒ To declare an editor of the comment.
2. :replacewith: ⇒ [Nullary] The text preceding this clause
   should be replaced by the text after it."
  (-let* (
          ;; Get arguments
          ((contents_1 . (&alist 'ed))
           (org-special-block-extras--extract-arguments contents 'ed))

          ;; Strip out any <p> tags
          (_ (setq contents_1 (s-replace-regexp "<p>" "" contents_1)))
          (_ (setq contents_1 (s-replace-regexp "</p>" "" contents_1)))

          ;; Are we in the html backend?
```

In LaTeX, an edcomm appears inline with the text surrounding it.

[**Bobert: Replace:** org-mode is dope, yo! **With:** Org-mode is essentially a path toward enlightenment.]

Unfortunately, in the HTML rendition, the edcomm is its own paragraph and thus separated by new lines from its surrounding text.

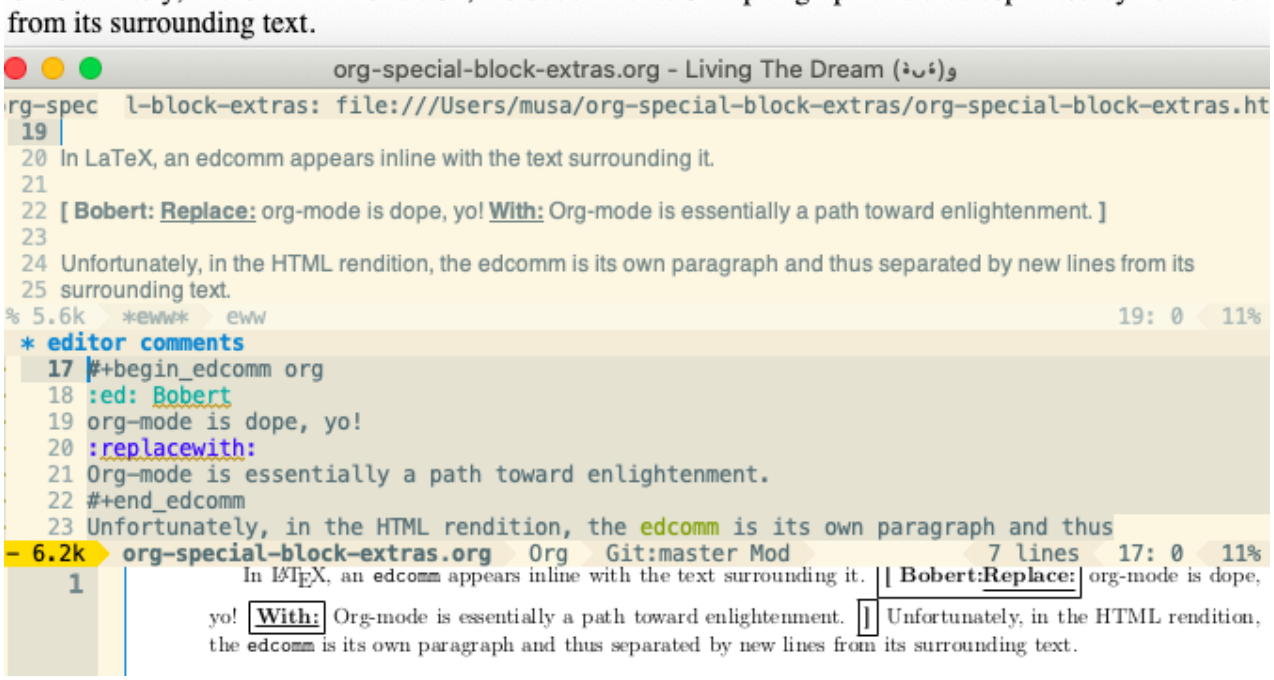

Figure 6: In order: Chrome, Emacs Web Wowser, Org source, PDF

```
        (html? (equal backend 'html))

        ;; fancy display style
        (boxed (lambda (x)
                 (if html?
                     (concat "<span style=\"border-width:1px"
                             ";border-style:solid;padding:5px\">"
                             "<strong>" x "</strong></span>")
                 (concat "\\fbox{\\bf " x "}"))))

        ;; Is this a replacement clause?
        ((this that) (s-split ":replacewith:" contents_1))
        (replacement-clause? that) ;; There is a 'that'
        (replace-keyword (if html? " <u>Replace:</u>"
                           "\\underline{Replace:}"))
        (with-keyword    (if html? "<u>With:</u>"
                           "\\underline{With:}"))
        (editor (format "[%s:%s"
                        (if (s-blank? ed) "Editor Comment" ed)
                        (if replacement-clause?
                            replace-keyword
                          "")))
        (contents_2 (if replacement-clause?
                        (format "%s %s %s" this
```

```
45                                    (funcall boxed with-keyword)
46                                    that)
47                            contents1))
48
49              ;; "[Editor Comment:"
50              (edcomm-begin (funcall boxed editor))
51              ;; "]"
52              (edcomm-end (funcall boxed "]")))
53
54      (setq org-export-allow-bind-keywords t) ;; So users can use "#+bind" immediately
55      (if org-special-block-extras-hide-editor-comments
56          ""
57        (format (pcase backend
58                  ('html "<p> %s %s %s</p>")
59                  ('latex "%s %s %s"))
60                edcomm-begin contents2 edcomm-end))))
```

In the HTML export, the `edcomm` special block is *not* in-line with the text surrounding it —ideally, it would be inline so that existing paragraphs are not split into multiple paragraphs but instead have an editor's comment indicating suggested alterations; see Line 16 above.

6.1 Examples

Org-markup is supported, as expected.

All editor comments are disabled by declaring, in your Org file:

```
#+bind: org-special-block-extras-hide-editor-comments t
```

The `#+bind:` keyword makes Emacs variables buffer-local during export —it is evaluated *after* any `src` blocks. To use it, one must declare in their Emacs init file the following line, which our `edcomm` utility ensures is true.

```
(setq org-export-allow-bind-keywords t)
```

(Remember to `C-c C-c` the `#+bind` to activate it, the first time it is written.)

6.1.1 No optional arguments

[Editor Comment: *Please* **change** this section to be more, ya know, professional. **]**

6.1.2 Only declaring an `:ed:` —editor

[Bobert: *Please* **change** this section to be more, ya know, professional. **]**

Possibly with no contents: **[Bobert:** **]**

6.1.3 Empty contents, no editor, nothing

[Editor Comment: **]**

Possibly with an empty new line: **[Editor Comment:** **]**

6.1.4 With a `:replacewith:` clause

[Editor Comment:Replace: The two-dimensional notation; e.g., $\sum_{i=0}^{n} i^2$ **With:** A linear one-dimensional notation; e.g., $(\Sigma i : 0..n \bullet i^2)$ **]**

Possibly "malformed" replacement clauses.

1. Forget the thing to be replaced. **[Editor Comment:Replace:** **With:** A linear one-dimensional notation; e.g., $(\Sigma i : 0..n \bullet i^2)$ **]**
2. Forget the new replacement thing. **[Editor Comment:Replace:** The two-dimensional notation; e.g., $\sum_{i=0}^{n} i^2$ **With:** **]**
3. Completely lost one's train of thought. **[Editor Comment:Replace:** **With:** **]**

7 Folded Details

- 'Conversation-style' articles, where the author asks the reader questions whose answers are "folded away" so the reader can think about the exercise before seeing the answer.
- Hiding boring but important code snippets, such as a list of import declarations or a tedious implementation.

```
#+LATEX_HEADER: \usepackage{tcolorbox}
```

```
(defun org-special-block-extras--details (backend contents)
"Format CONTENTS as a ‘folded region’ according to BACKEND.

CONTENTS may have a ‘:title’ argument specifying a title for
the folded region."
(-let* (;; Get arguments
        ((contents' . (&alist 'title))
         (org-special-block-extras--extract-arguments contents 'title)))
  (when (s-blank? title) (setq title "Details"))
  (setq title (s-trim title))
  (format
   (s-collapse-whitespace ;; Remove the whitespace only in the nicely presented
                          ;; strings below
    (pcase backend
      (‘html "<details class=\"code-details\">
                <summary>
                  <strong>
                    <font face=\"Courier\" size=\"3\" color=\"green\"> %s
                    </font>
                  </strong>
                </summary>
                %s
              </details>")
      (‘latex "\\begin{quote}
                \\begin{tcolorbox}[colback=white,sharp corners,boxrule=0.4pt]
                  \\textbf{%s:}
```

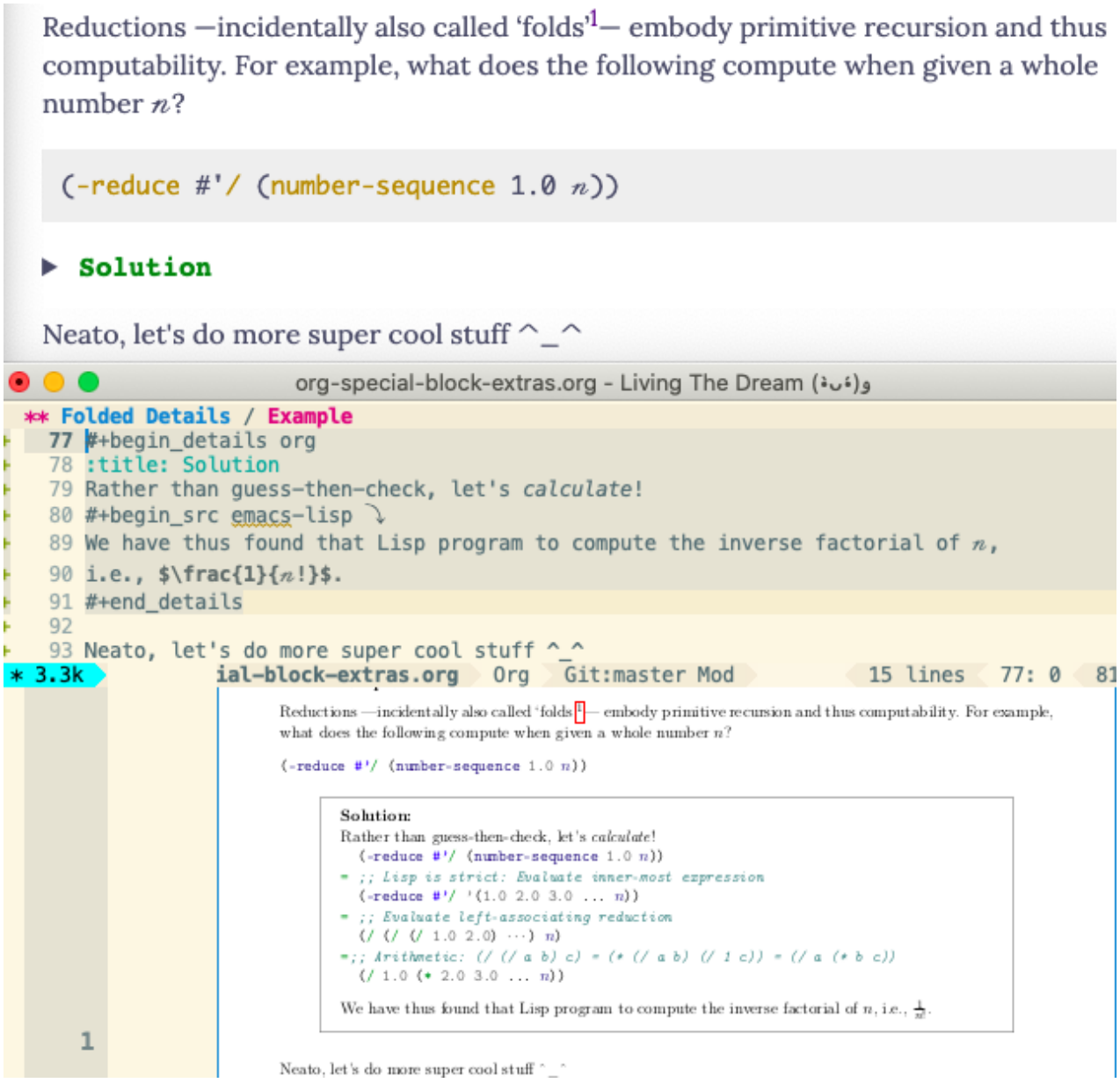

Figure 7: Visually hiding, folding away, details

```
                    %s
                  \\end{tcolorbox}
                \\end{quote}")))
    title contents')))
```

We could use `\begin{quote}\fbox{\parbox{\linewidth}{\textbf{Details:} ...}}\end{quote}`; however, this does not work well with `minted`, for coloured source blocks. Instead, we use `tcolorbox`.

7.1 Example

Reductions —incidentally also called 'folds'[1]— embody primitive recursion and thus computability. For example, what does the following compute when given a whole number n?

```
(-reduce #'/ (number-sequence 1.0 n))
```

> **Solution:** Rather than guess-then-check, let's *calculate*!
>
> ```
>   (-reduce #'/ (number-sequence 1.0 n))
> = ;; Lisp is strict: Evaluate inner-most expression
>   (-reduce #'/ '(1.0 2.0 3.0 ... n))
> = ;; Evaluate left-associating reduction
>   (/ (/ (/ 1.0 2.0) ···) n)
> =;; Arithmetic: (/ (/ a b) c) = (* (/ a b) (/ 1 c)) = (/ a (* b c))
>   (/ 1.0 (* 2.0 3.0 ... n))
> ```
>
> We have thus found that Lisp program to compute the inverse factorial of n, i.e., $\frac{1}{n!}$.

Neato, let's do more super cool stuff ^_^

(In the Emacs Web Wowser, folded regions are displayed unfolded similar to LaTeX.)

[1] See *A tutorial on the universality and expressiveness of fold* and *Unifying Structured Recursion Schemes*