Alice goes floating
Frank Mittelbach
TUG 2016, Toronto, Canada, July 2016

*/Alice goes floating*

This morning I like to take you on a journey to Alice in Wonderland to see how she is floating among all her pictures.

So sit back, relax and enjoy!

*/Alice goes floating/Typesetting Alice*

Like the rabbit we need to be concerned about time passed so that shows up on the slides as well.

*/Alice goes floating/Typesetting Alice/Download Alice in Wonderland f...*

In preparation I downloaded the original text from the Gutenberg Project,

- did some minimal adjustments so that we a few headings,
- changed „underscores“ indicating  emphasis
- and made sure that „poems“ and similar items are treated as unbreakable blocks

I also hunted up the original drawings and placed them in their appropriate places in the source

*/Alice goes floating/Typesetting Alice/General settings*

For typesetting I chose fairly standard settings with some slightly more rigid values, for example,

- widows and orphans are totally forbidden
- and there is no extra flexibility in vertical spacing between paragraphs

Another characteristic is that heading at the top of a columns are encouraged

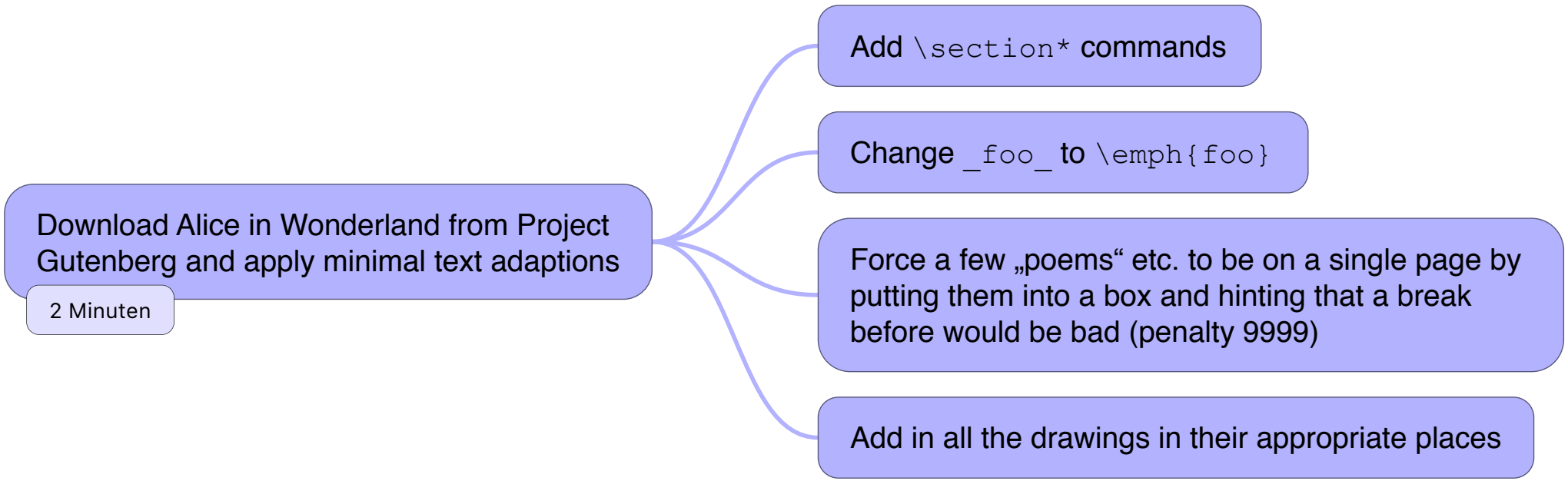
```
\textheight = 550.0pt (46 lines a 12pt)
\textwidth  = 229.5pt (approx 50-55 characters per line)

\clubpenalty      = 10000    % no orphans
\widowpenalty     = 10000    % no widows
\parskip          = 0pt      % no paragraph separation flexibility
\@beginparpenalty = 9999     % strongly discourage breaks in front of
                             % „verse“ and similar environments
\@secpenalty      = -9000    % strongly encourage section breaks

\tolerance        = 4000     % allow fairly loose paragraphs
```

*/Alice goes floating/Typesetting Alice/Run this through standard LaTe...*

…

# Typesetting Alice

Rollup: 11 Minuten


Download Alice in Wonderland from Project Gutenberg and apply minimal text adaptions
2 Minuten
Add \section* commands
Change _foo_ to \emph{foo}
Force a few „poems“ etc. to be on a single page by putting them into a box and hinting that a break before would be bad (penalty 9999)
Add in all the drawings in their appropriate places

General settings

1 Minute

- Two columns (46 lines) with `\flushbottom`
-  no widows or orphans no \parskip flexibility
-  favor headings on top of column
-  encourage „pre-text“ + display env. kept together
-  reasonably flexible `\tolerance` to allow for narrow columns

Run this through standard LaTeX we obtain …

3 Minuten

*/Alice goes floating/Typesetting Alice/Run this through standard LaTe.../… a document with a bunch of i...*

Running this through standard LaTeX (with the above settings) we obtain a document with a bunch of issues:

check out phase0-stdlatex-with-floats.pdf

*/Alice goes floating/Typesetting Alice/Can we do better?*

Can we do better?

*/Alice goes floating/Typesetting Alice/Can we do better?/Yes, but …*

The answer is „yes we can“ but there is a lot of manual labor involved —
and I speak from experience having done that kind of work for a number of books and up with up to 30% manual pagination + rewriting

*/Alice goes floating/Typesetting Alice/Can we do better?/Demo*

Show life demo paginating Alice with strict settings (no \parskip flexibility,
no widows and orphans) but using global optimization.

*/Alice goes floating/Typesetting Alice/Can we do better?/Demo/… an adjusted document*

The result is

phase4-strict-texflex-firstpagedrop.pdf

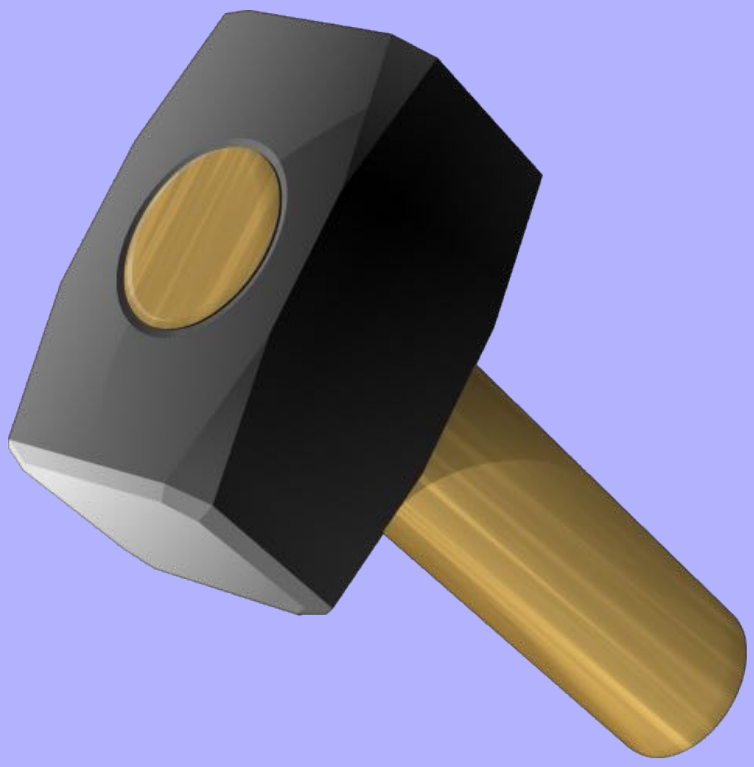
*/Alice goes floating/How?/First … some standard LaTeX ex...*

First the results from some more sample documents this time without any floats.

All documents have been set in two columns with a width of 8 cm.
Each column could hold 46 lines of text and the paragraph requirements have been fairly strict: no widows or orphans and only a small amount of flexibility (+1pt) for the paragraph separation.
This means that in each column one could gain a flexibility of up to 2 lines
(but only when there are 8 or more paragraphs in the column and we accept a stretch of up to 3 times the nominal value which corresponds to a badness of 2700).

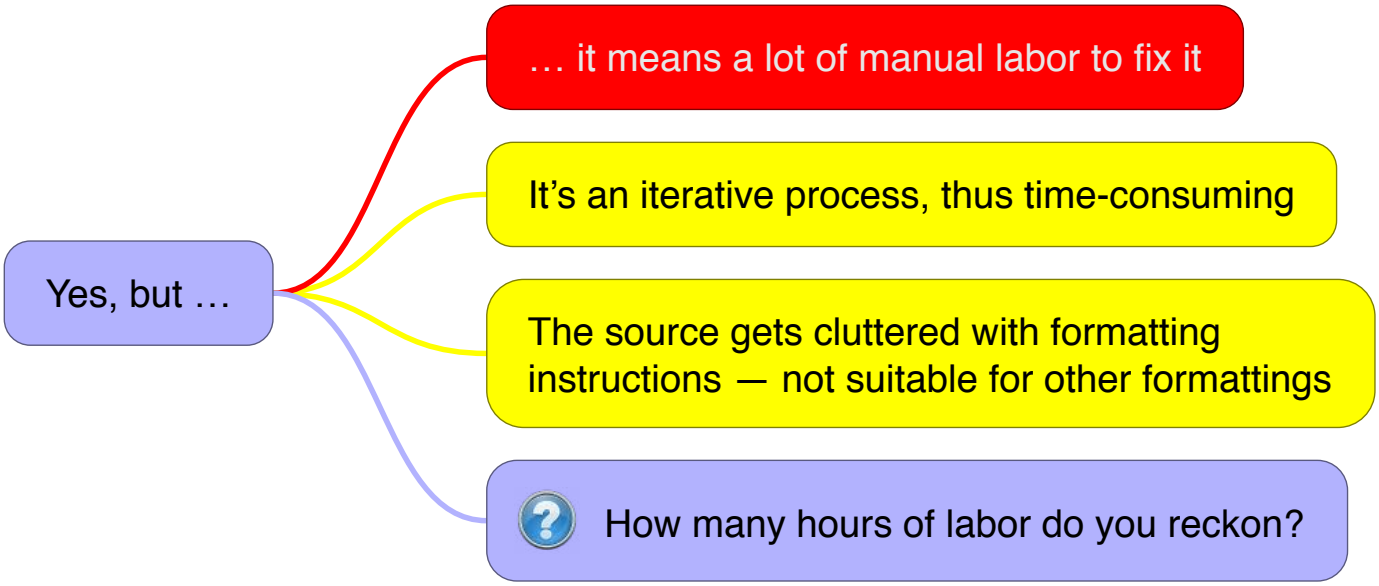
As it can be immediately seen, all documents show problematic page/column breaks (in the range of 4%-16%). If we remove the \parskip flexibility we will see up to 30% bad breaks.

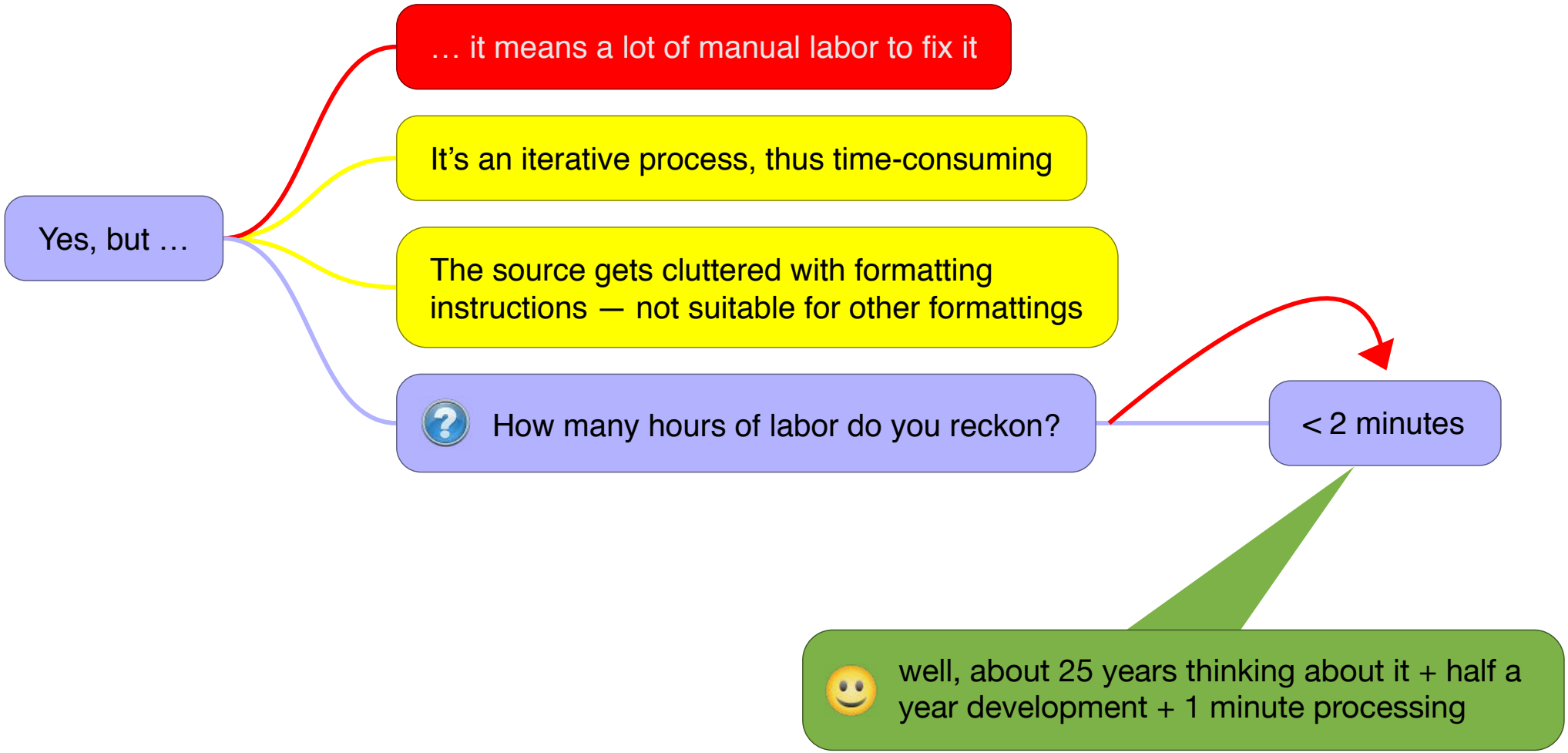

… a document with a bunch of issues

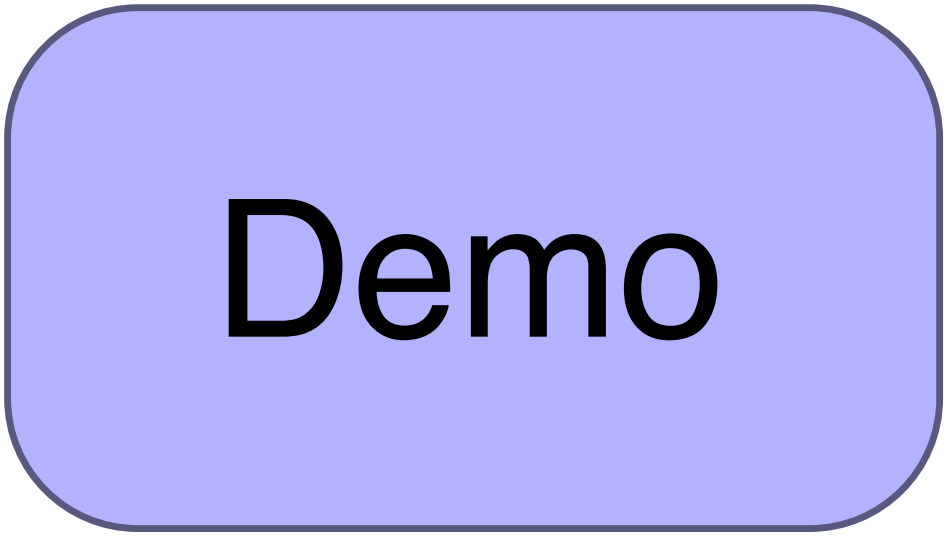


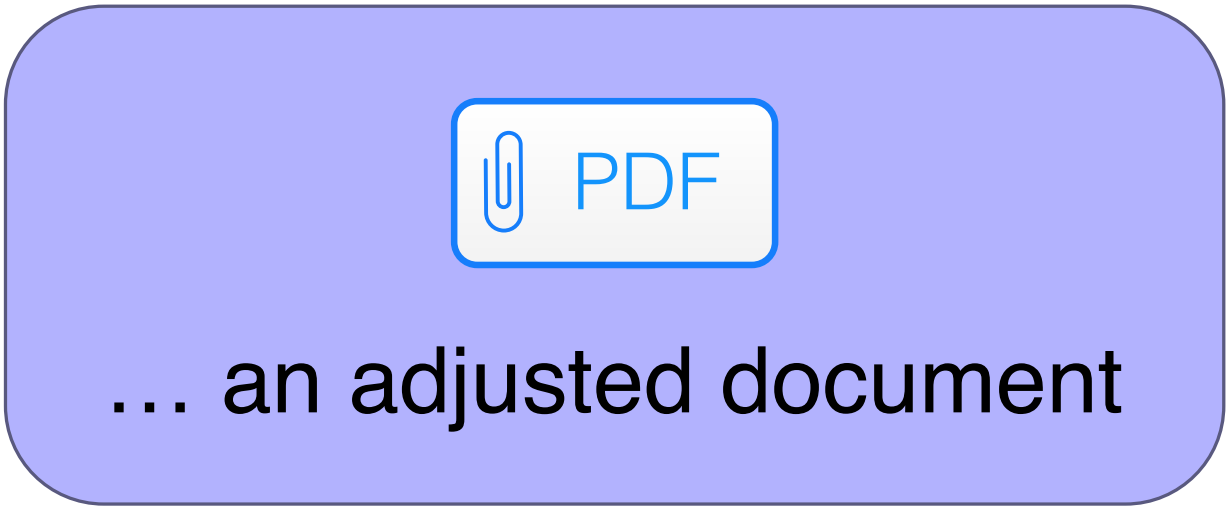
Can we do better?

5 Minuten


Yes, but …
… it means a lot of manual labor to fix it
It’s an iterative process, thus time-consuming
The source gets cluttered with formatting instructions — not suitable for other formattings
How many hours of labor do you reckon?


Yes, but …
… it means a lot of manual labor to fix it
It’s an iterative process, thus time-consuming
The source gets cluttered with formatting instructions — not suitable for other formattings
How many hours of labor do you reckon?
< 2 minutes
well, about 25 years thinking about it + half a year development + 1 minute processing


Demo


PDF
… an adjusted document

# How?

Rollup: 16 Minuten

First … some standard LaTeX examples

1 Minute

All examples are straight text without floats

Standard LaTeX here means the „greedy“ algorithm with small flexibility between paragraphs (`\parskip`) and no widows and orphans

| | document columns | paragraphs total | vertical badness good | bad | ugly/infinite | |
|---|---|---|---|---|---|---|
| **Alice in Wonderland** | 72 | 833 | 69 | 0 | **2+1** | (4.1%) |
| **Call of the Wild** | 78 | 340 | 64 | 1 | **9+4** | (16.6%) |
| **Grimm's Fairy Tales** | 236 | 1041 | 212 | 6 | **6+12** | (7.6%) |
| **Pride and Prejudice** | 316 | 2127 | 292 | 8 | **7+9** | (5.1%) |

*/Alice goes floating/How?/Idea*

The idea is the following: paragraph breaking and page breaking are fairly similar in that

- we have a similar about of breakpoints per line compared to breakpoints in a columns
- and the number of lines in a typical paragraph are not so much different to the number of columns in a chapter

So let's try to apply a suitably adapted version of the Knuth/Plass algorithm to pagination?

(Do we need a recap how Knuth/Plass works?)

*/Alice goes floating/How?/Idea/A quick recap: how does the Kn.../Dynamic programming approach*

Dynamic programming only works with certain type of problems that have the following characteristics:

- an optimal solution to the whole problem consists of optimal partial solutions

that is if we have a sub-optimal solution for, say the first 4 pages then it is not possible that this is part of the overall optimal solution

- subproblems overlap, that is if we try to find the optimal solution we would resolve the same subproblem many times

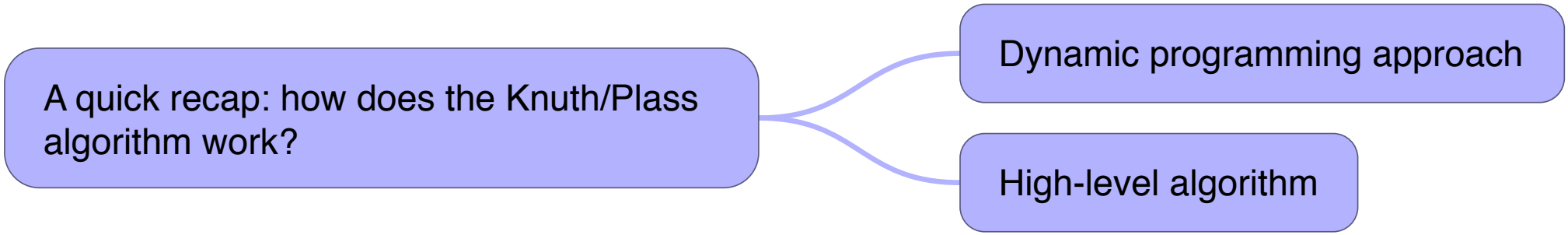
Idea

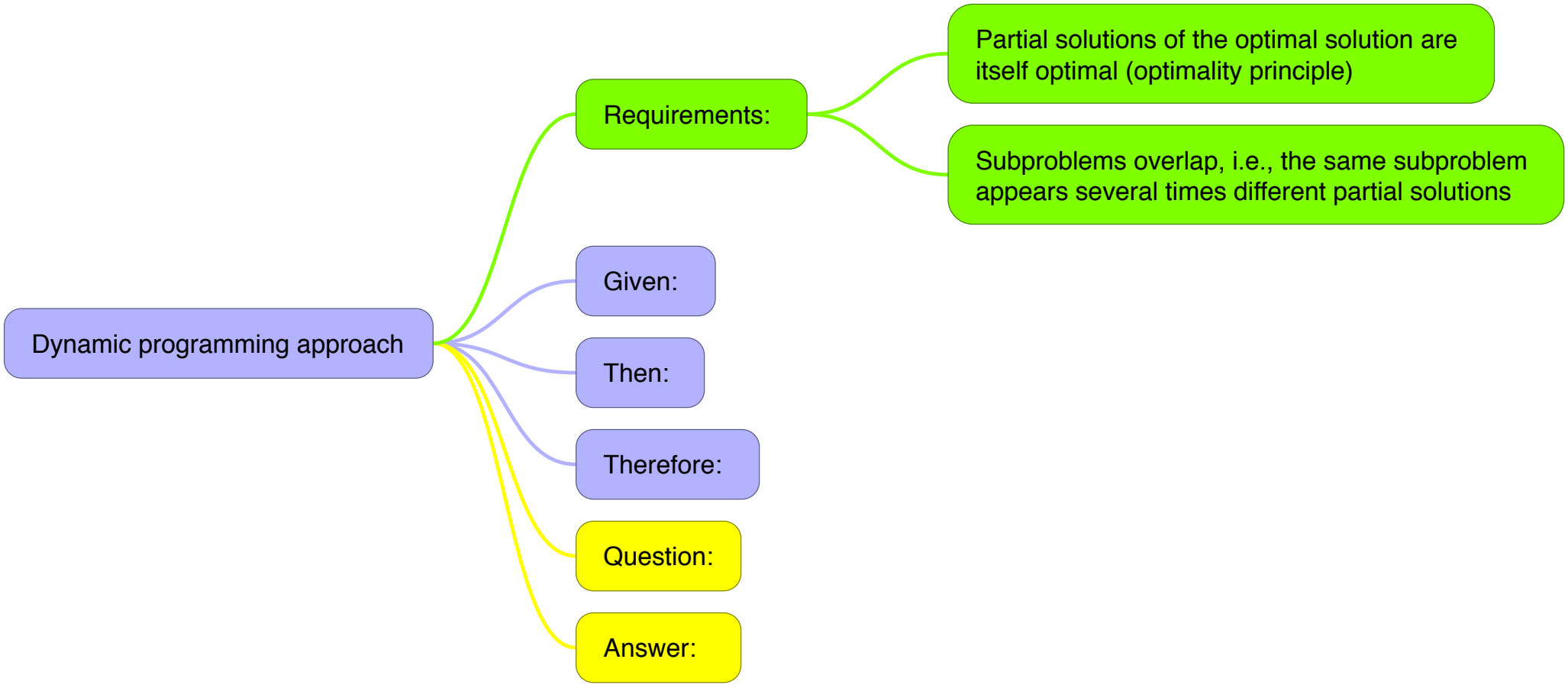
6 Minuten

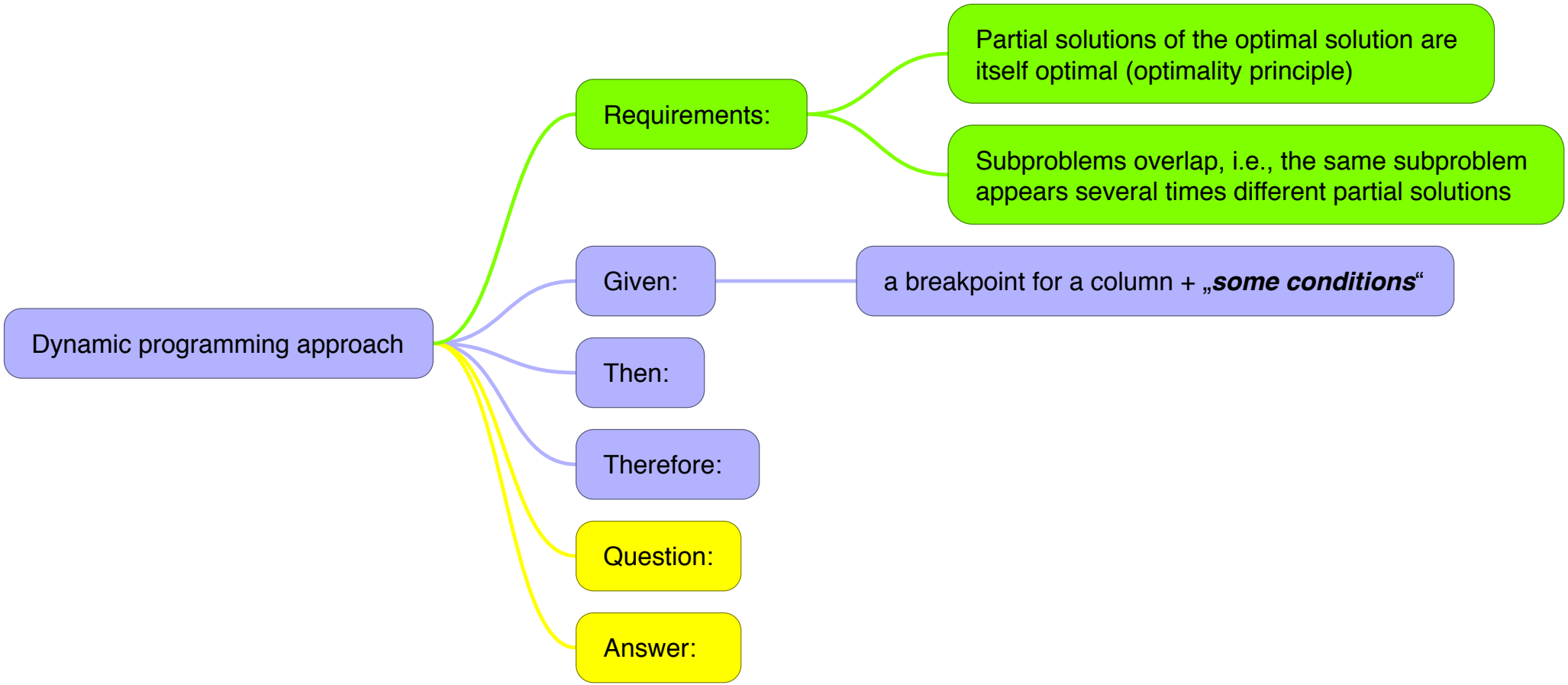
A typical column has a similar amount of breakpoints as a typical line with hyphenation (roughly 45-55 compared  30) and a typical chapter has not that many more pages than a typical paragraph has lines

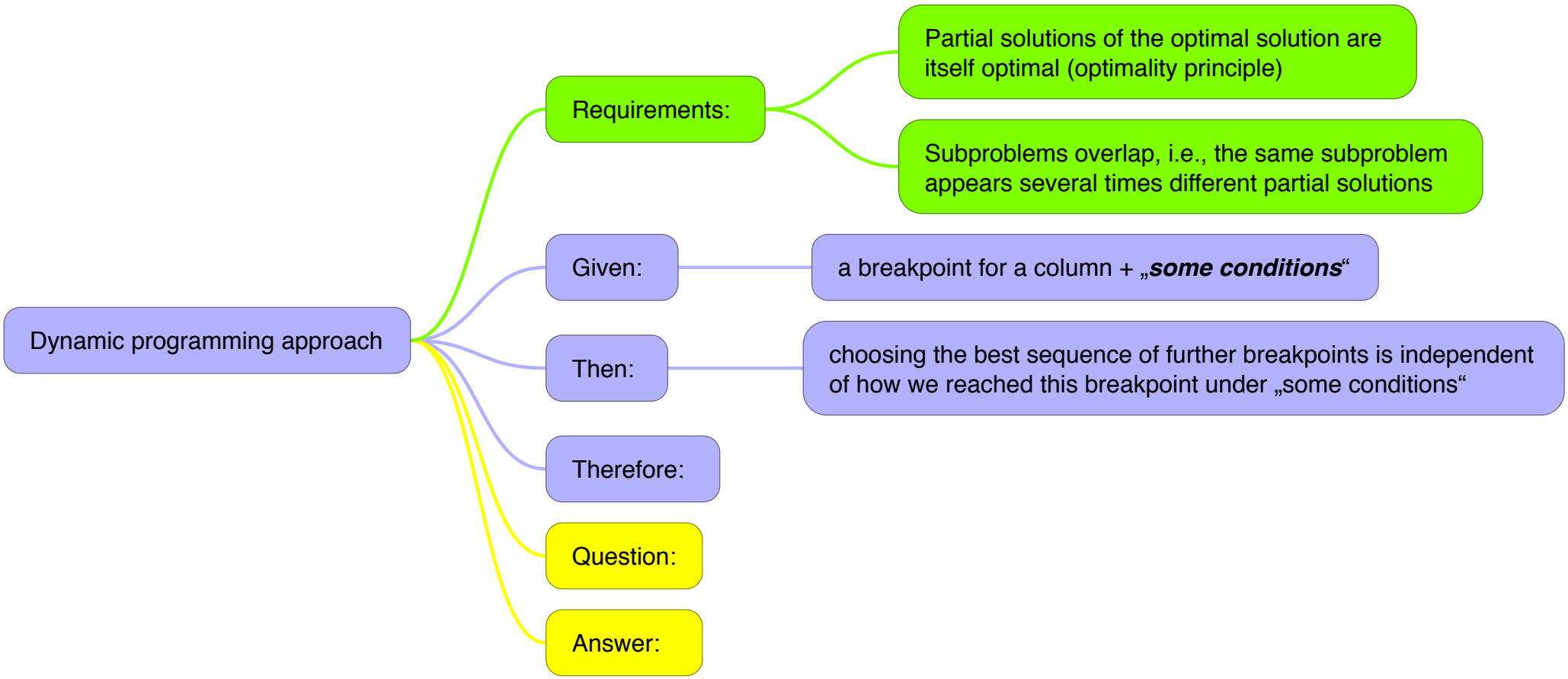
So applying Knuth/Plass (suitably changed) to pagination to achieve a globally optimized document should be possible

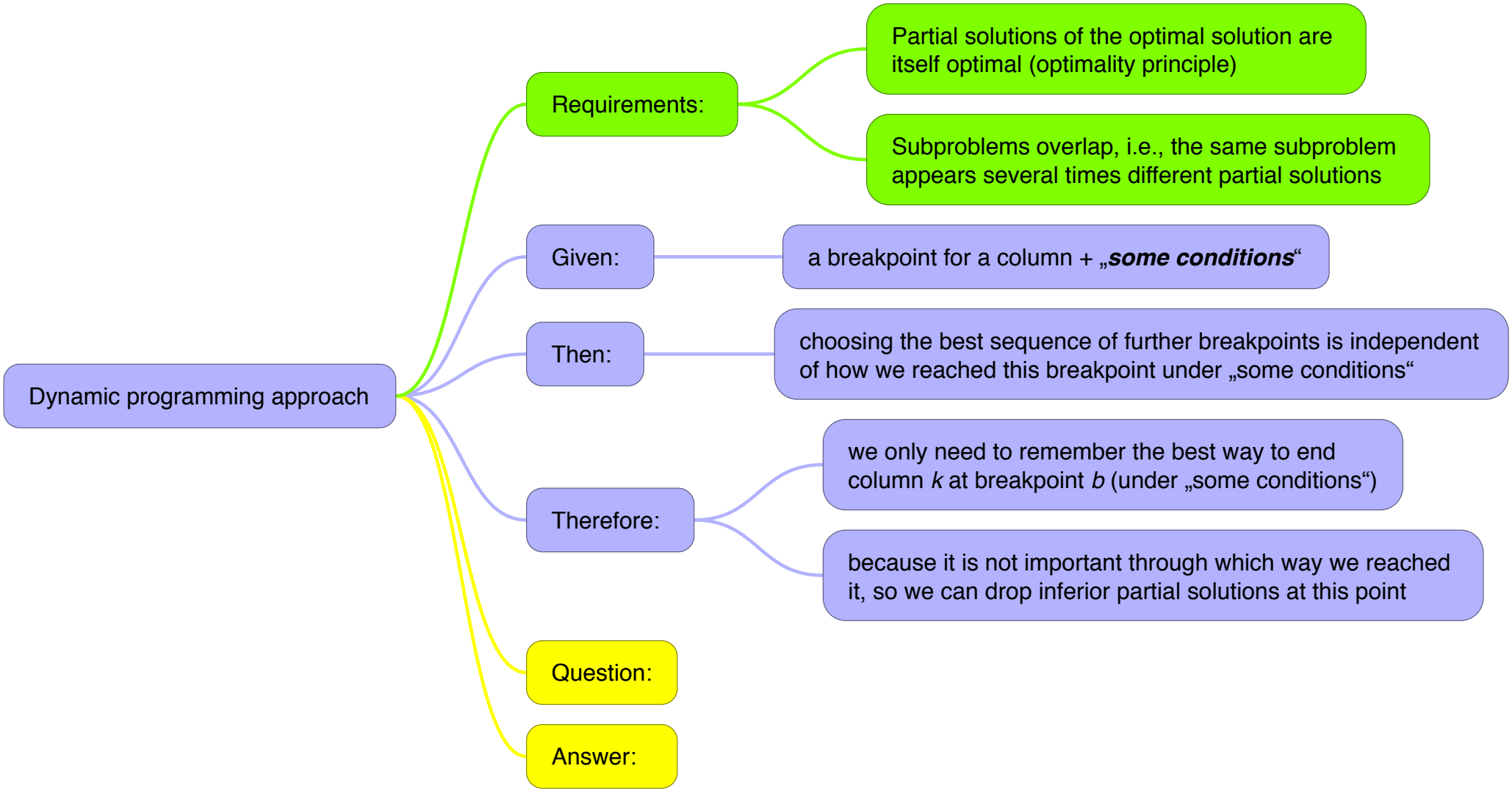
A quick recap: how does the Knuth/Plass algorithm work?

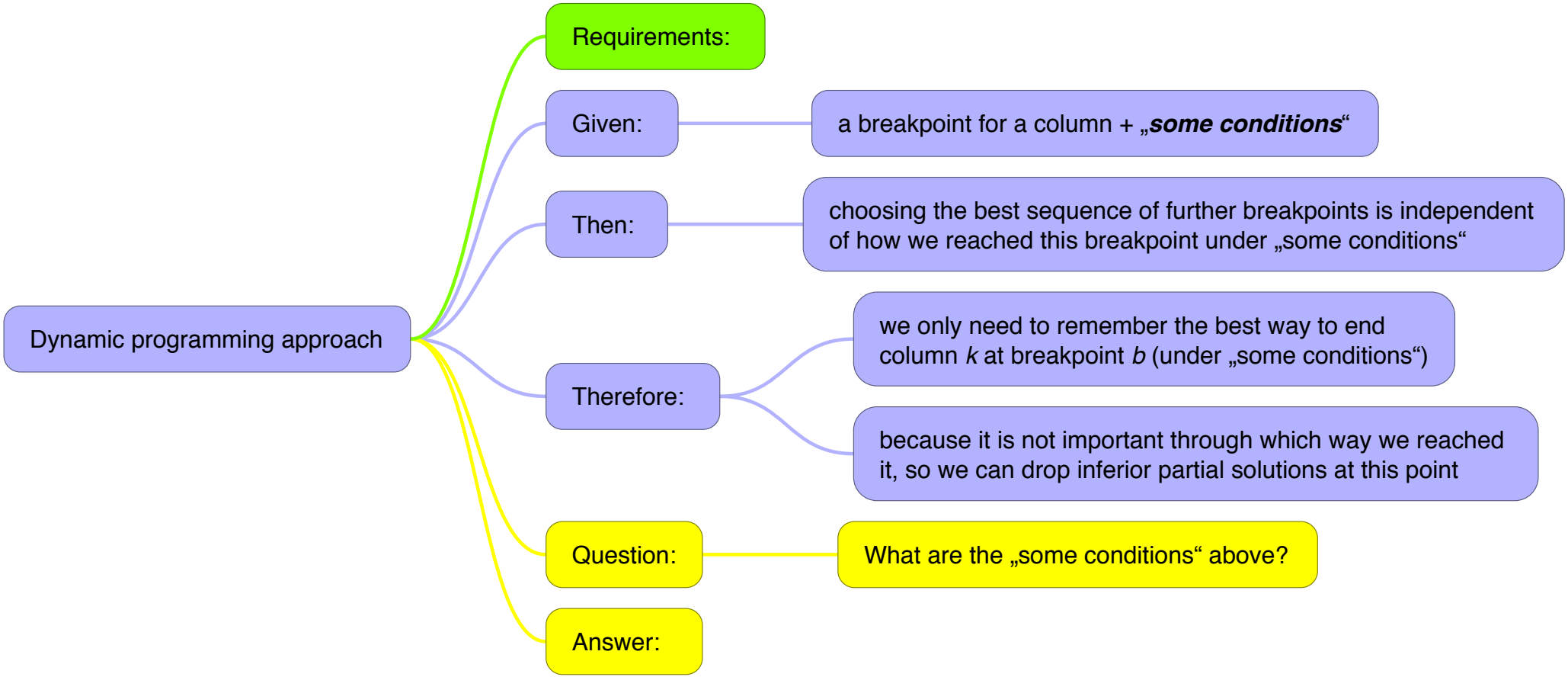

A quick recap: how does the Knuth/Plass algorithm work?
Dynamic programming approach
High-level algorithm

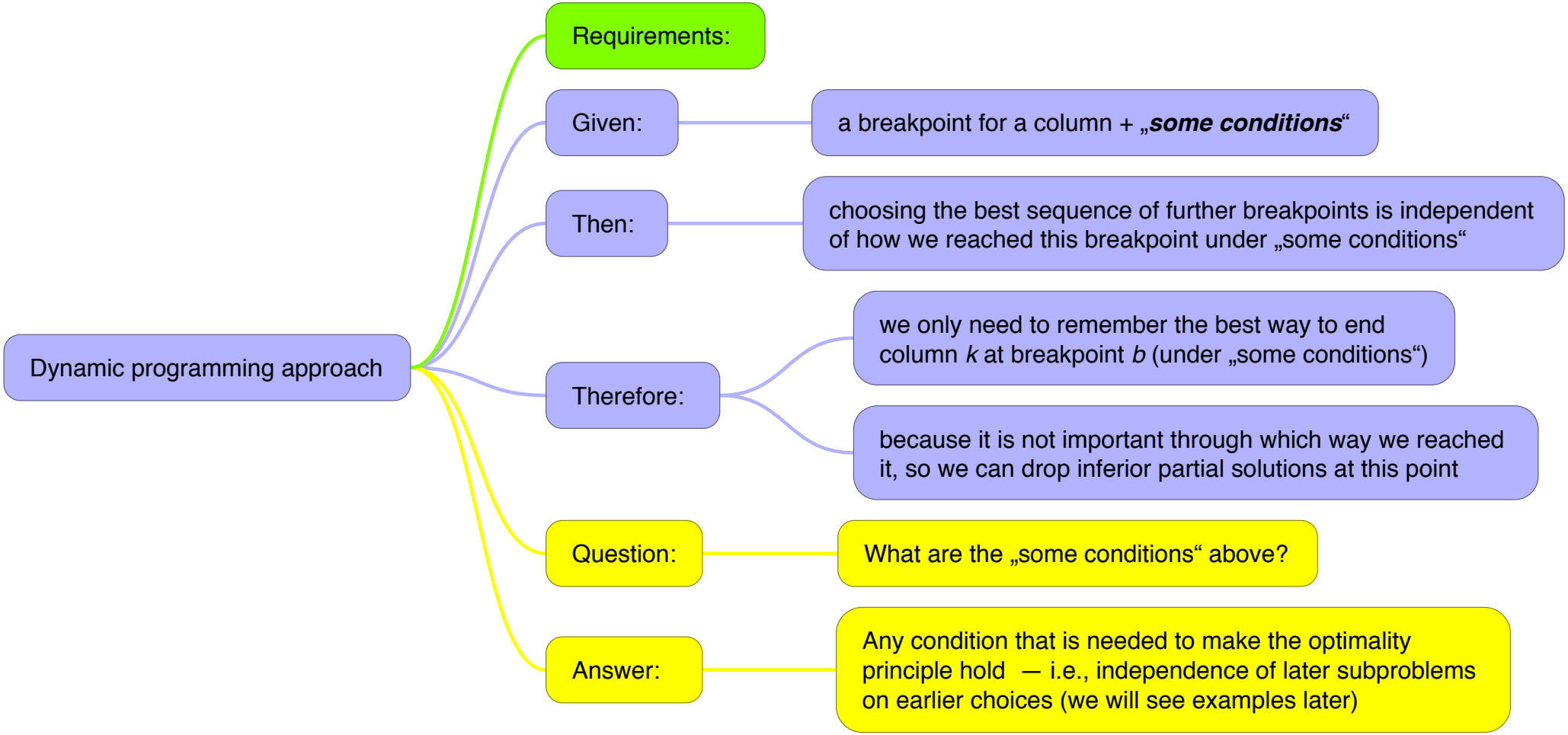

Dynamic programming approach
Requirements:
Partial solutions of the optimal solution are itself optimal (optimality principle)
Subproblems overlap, i.e., the same subproblem appears several times different partial solutions
Given:
Then:
Therefore:
Question:
Answer:


Dynamic programming approach
Requirements:
Partial solutions of the optimal solution are itself optimal (optimality principle)
Subproblems overlap, i.e., the same subproblem appears several times different partial solutions
Given:
a breakpoint for a column + „some conditions“
Then:
Therefore:
Question:
Answer:


Dynamic programming approach
Requirements:
Partial solutions of the optimal solution are itself optimal (optimality principle)
Subproblems overlap, i.e., the same subproblem appears several times different partial solutions
Given:
a breakpoint for a column + „some conditions“
Then:
choosing the best sequence of further breakpoints is independent of how we reached this breakpoint under „some conditions“
Therefore:
Question:
Answer:


Dynamic programming approach
Requirements:
Partial solutions of the optimal solution are itself optimal (optimality principle)
Subproblems overlap, i.e., the same subproblem appears several times different partial solutions
Given:
a breakpoint for a column + „some conditions“
Then:
choosing the best sequence of further breakpoints is independent of how we reached this breakpoint under „some conditions“
Therefore:
we only need to remember the best way to end column k at breakpoint b (under „some conditions“)
because it is not important through which way we reached it, so we can drop inferior partial solutions at this point
Question:
Answer:


Dynamic programming approach
Requirements:
Given:
a breakpoint for a column + „some conditions“
Then:
choosing the best sequence of further breakpoints is independent of how we reached this breakpoint under „some conditions“
Therefore:
we only need to remember the best way to end column k at breakpoint b (under „some conditions“)
because it is not important through which way we reached it, so we can drop inferior partial solutions at this point
Question:
What are the „some conditions“ above?
Answer:


Dynamic programming approach
Requirements:
Given:
a breakpoint for a column + „some conditions“
Then:
choosing the best sequence of further breakpoints is independent of how we reached this breakpoint under „some conditions“
Therefore:
we only need to remember the best way to end column k at breakpoint b (under „some conditions“)
because it is not important through which way we reached it, so we can drop inferior partial solutions at this point
Question:
What are the „some conditions“ above?
Answer:
Any condition that is needed to make the optimality principle hold — i.e., independence of later subproblems on earlier choices (we will see examples later)

*/Alice goes floating/How?/Idea/A quick recap: how does the Kn.../High-level algorithm*

So let's give a very high-level overview of the algorithm applied to pagination …

We loop through all possible breakpoints in the document …

… and maintain a list of „active" breakpoints representing the best way of ending some column under some condition.

Initially this list will only contain a single entry representing the start of the document.

So: one active element initially … now … **If**

- we can form a column from any element in the active list to the current breakpoint with an acceptable quality then this becomes a candidate solution for the next column
- out of the candidates we choose the best and add it to the active list
- if we have different conditions, then we have to choose the best among all with the same condition (so we may have to add several new elements to the active list)

Then we move to the next breakpoint.

(Two points here: we apply the optimality principle by only adding the „best" candidate and this is the part where the active list grows.)

**Once** an element of the active list is too far out to be able to form a column with the current breakpoint then we remove that element from the active list.

(For example, if the we are looking at more and more breakpoints, there will come a point when it is impossible to squeeze all the material from the start of the document to this breakpoint into a single column. So then re remove the element representing the start of the document. For all later breakpoints the situation would even be worse.)

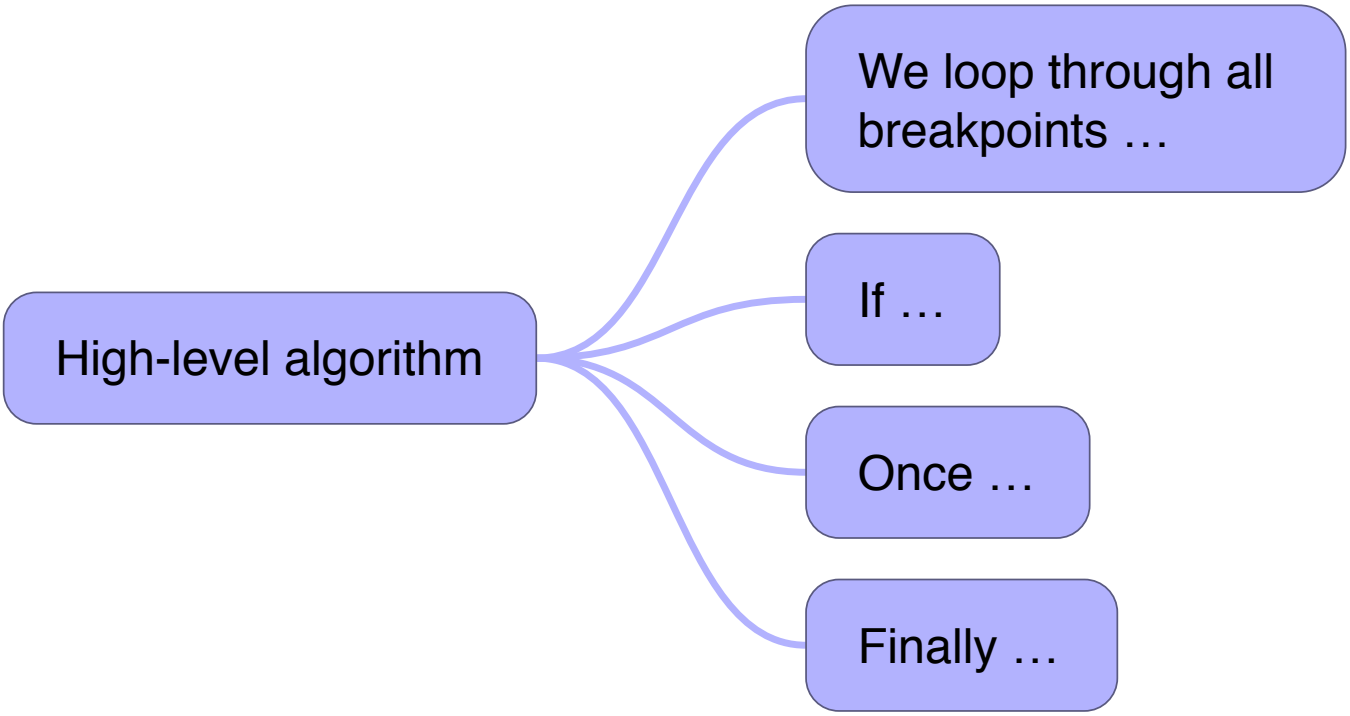
**Finally** when we reach the end of the document we can construct the optimal solution by simply moving backwards through the selections we made early to reach the best solution for the last column. (Requires some housekeeping, but otherwise is straight forward).

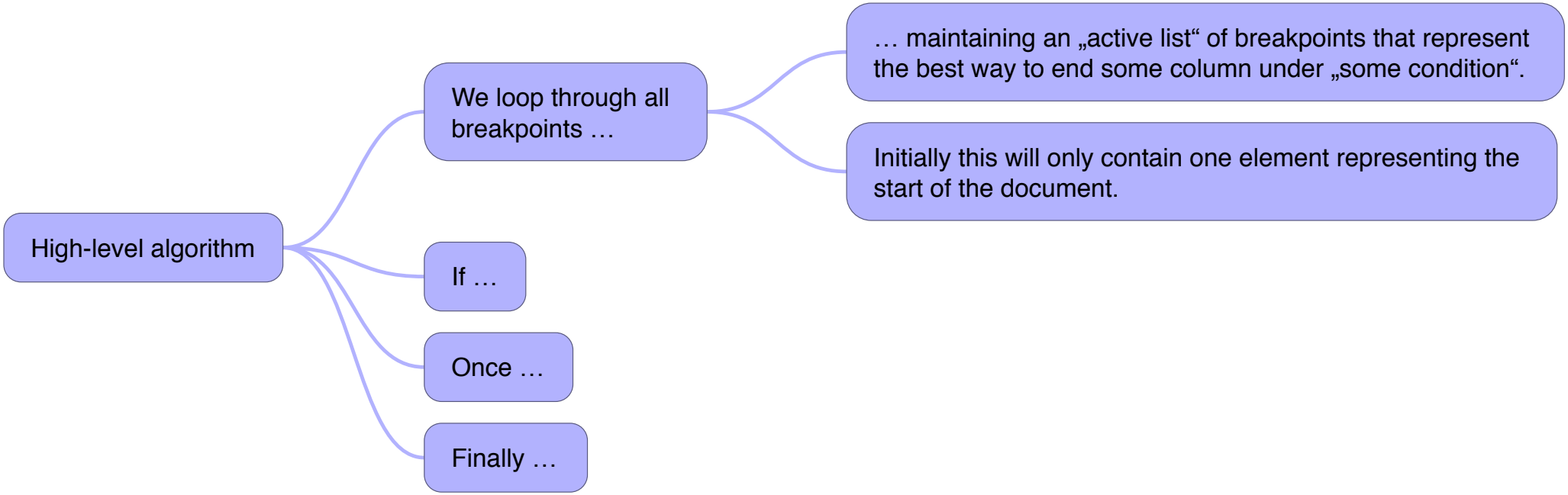
The interesting point here is that the algorithm runs in linear time if the active list is bounded by a constant, otherwise it runs in quadratic time (in the number of breakpoints)

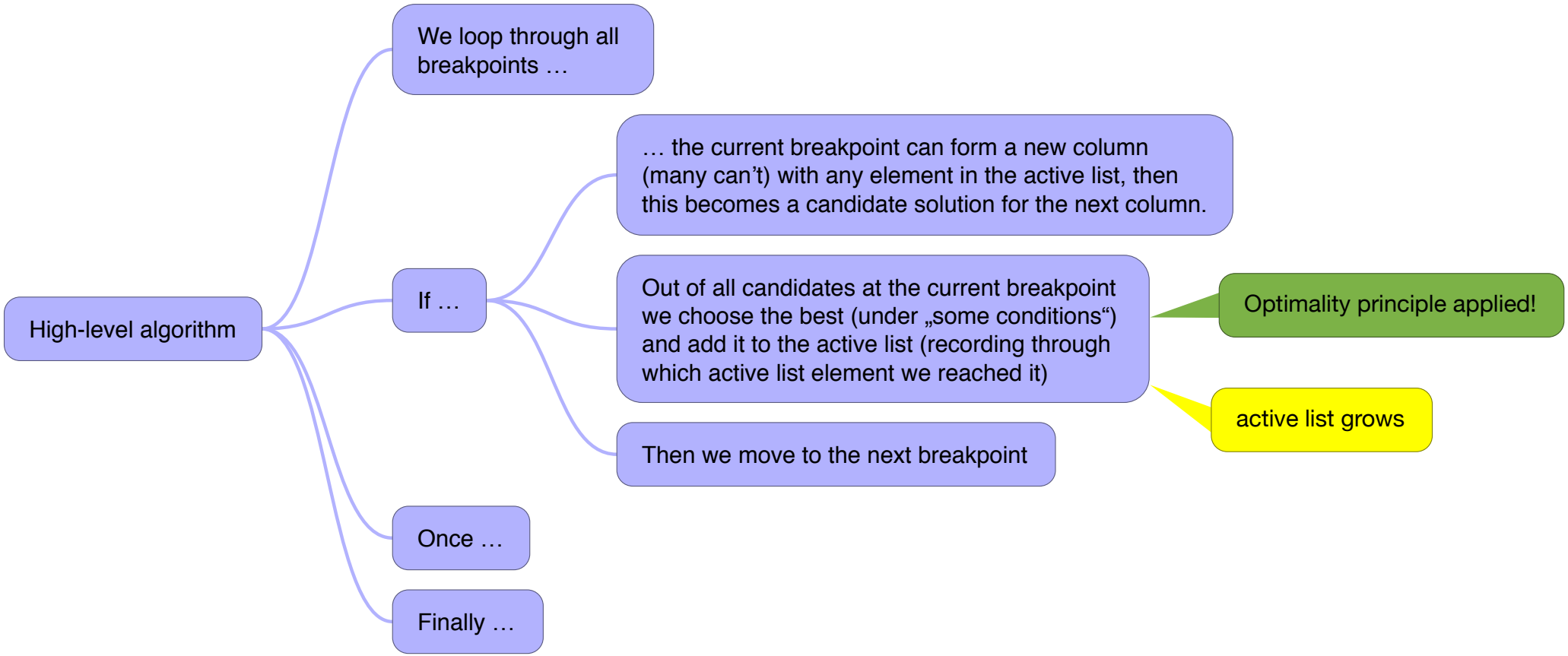
*/Alice goes floating/How?/So we should be able to apply .../Standard LaTeX examples optimi...*

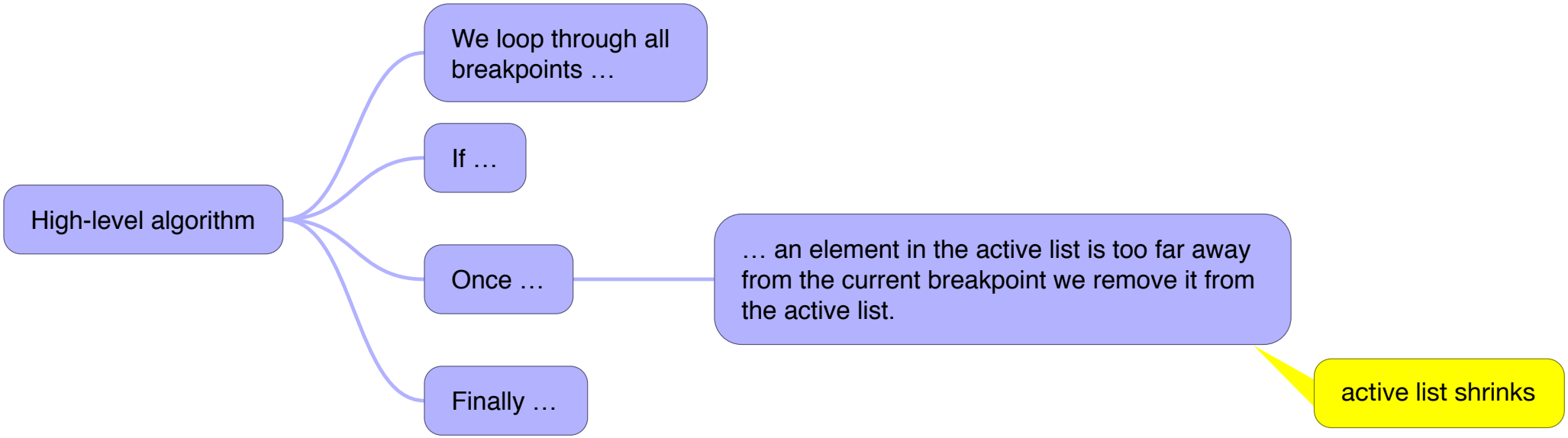
If we apply globally optimized pagination we fail in nearly all cases because we run out of alternatives (i.e., the active list gets empty along the way).

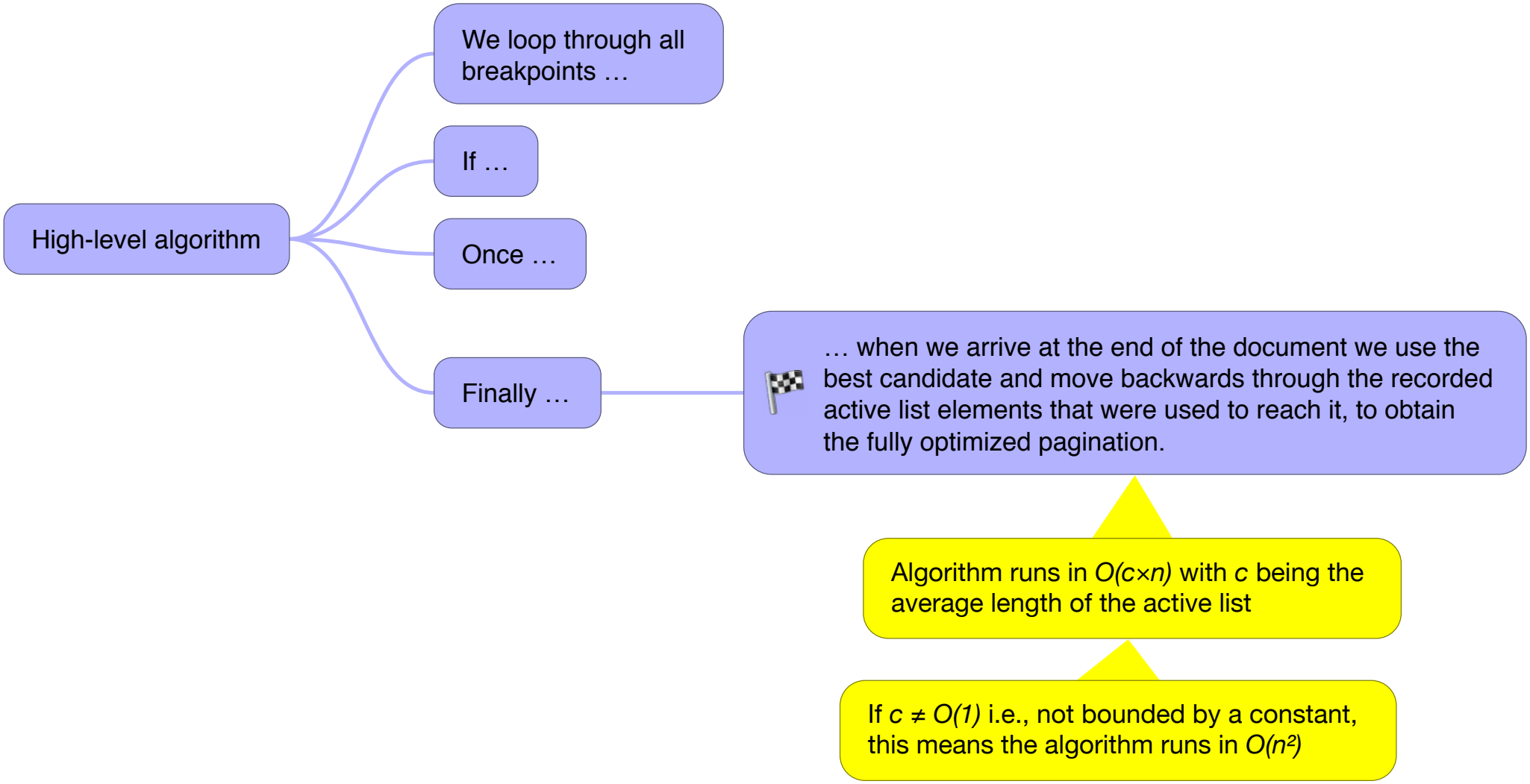
TUG-table-std-LaTeX-examples-optimized.png


High-level algorithm
We loop through all breakpoints …
If …
Once …
Finally …


High-level algorithm
We loop through all breakpoints …
… maintaining an „active list“ of breakpoints that represent the best way to end some column under „some condition“.
Initially this will only contain one element representing the start of the document.
If …
Once …
Finally …


High-level algorithm
We loop through all breakpoints …
If …
… the current breakpoint can form a new column (many can't) with any element in the active list, then this becomes a candidate solution for the next column.
Out of all candidates at the current breakpoint we choose the best (under „some conditions“) and add it to the active list (recording through which active list element we reached it)
Optimality principle applied!
active list grows
Then we move to the next breakpoint
Once …
Finally …


High-level algorithm
We loop through all breakpoints …
If …
Once …
… an element in the active list is too far away from the current breakpoint we remove it from the active list.
active list shrinks
Finally …


High-level algorithm
We loop through all breakpoints …
If …
Once …
Finally …
… when we arrive at the end of the document we use the best candidate and move backwards through the recorded active list elements that were used to reach it, to obtain the fully optimized pagination.
Algorithm runs in O(c×n) with c being the average length of the active list
If c ≠ O(1) i.e., not bounded by a constant, this means the algorithm runs in O(n²)

So we should be able to apply Knuth/Plass

**Or not?**

2 Minuten

There is a big difference between paragraph and page breaking as in contrast to paragraphs pages have little to no flexibility

So if you optimize a simple text document (such as Alice without floats) you are most likely running out of options and get the a equivalent of „overfull lines“

Standard LaTeX examples optimized

| | document | | nested list | | paragraphs | verbatim | | colour-- |
|---|---|---|---|---|---|---|---|---|
| | columns | blocks | max | average | total | good | bad | ugly/infinite |
| **Alice in Wonderland** | 72 | | | | 853 | 69 | 0 | 2+1 |
| base | | 4047 | 37 | 9 | | | *no solution* | |
| **Call of the Wild** | 78 | | | | 340 | 64 | 1 | 9+4 |
| base | – | 9118 | 9 | 3 | | | *no solution* | |
| **Grimm's Fairy Tales** | 236 | | | | 1041 | 212 | 6 | 6–12 |
| base | – | 27908 | 22 | 4 | | | *no solution* | |
| **Pride and Prejudice** | 216 | | | | 2127 | 292 | 8 | 7 \| 9 |
| base | 219 | 34845 | 39 | 4 | | 318 | – | – |

Standard LaTeX examples optimized

| | document | | active list | | paragraphs | vertical badness | | |
|---|---|---|---|---|---|---|---|---|
| | columns | blocks | max | average | total | good | bad | ugly/infinite |
| **Alice in Wonderland** | 72 | | | | 833 | 69 | 0 | **2+1** |
| base | – | 6947 | 37 | 12 | | | *no solution* | |
| **Call of the Wild** | 78 | | | | 340 | 64 | 1 | **9+4** |
| base | – | 9148 | 9 | 2 | | | *no solution* | |
| **Grimm's Fairy Tales** | 236 | | | | 1041 | 212 | 6 | **6+12** |
| base | – | 27908 | 22 | 4 | | | *no solution* | |
| **Pride and Prejudice** | 316 | | | | 2127 | 292 | 8 | **7+9** |
| base | 318 | 34645 | 39 | 14 | | 318 | – | – |

*/Alice goes floating/How?/So what now?*

So what now?

Basically we have to find ways to introduce more flexibility on the page.

*/Alice goes floating/How?/So what now?/Options*

For this we have basically 4 options:

**- allow non-sequential ordering of textual elements**

For most document types that is not an option as the order of presentation is essential for the readers understanding. However, with journals or newsletters and similar types reordering of independent „stories“ will introduce some extra flexibility.

**- allow variations in column heights**

A typical trick of the craft is running all columns of a double spread a line short or long.

**- allow variations in the height of textual elements**

It may be possible to format paragraphs to different numbers of lines (without sacrificing the quality) or to format tables and figures to different heights

(A variation of this is to change the content of textual elements —you have that option if you are not only the typesetter but also the author)

**- include float placement in optimization**

Placing floats onto different columns/pages will change the column height that remains for textual material and thus provides additional flexibility for pagination.

So what now?

7 Minuten

So what now?

7 Minuten

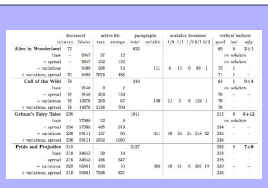
Introduce additional flexibility on the page

Options

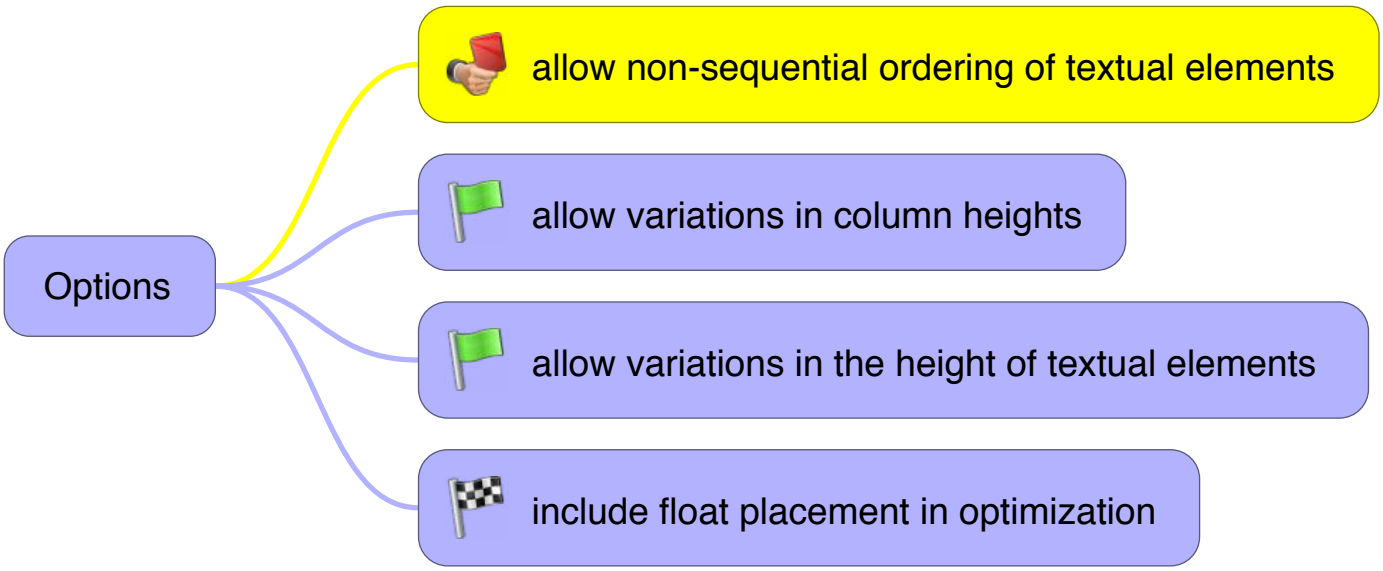
Running spreads short or long (briefly)

Variation in textual element height (briefly)

Optimality principle conditions:



Optimized examples


Options
allow non-sequential ordering of textual elements
allow variations in column heights
allow variations in the height of textual elements
include float placement in optimization

Running spreads short or long (briefly)

Provide additional flexibility by running double spreads one line long or short

A standard trick of the craft

Variation in textual element height (briefly)

Provide additional flexibility by providing different paragraph formattings if possible

TeX's \looseness is naive: value >0 will result in a last line with one (partial) word

Resolution: massage the hlist and add higher penalties near the end so that TeX will not like breaking there, then try \looseness

*/Alice goes floating/How?/So what now?/Optimality principle condition...*

So let's see what this means for the extra conditions needed to make the optimality principle work …

When **all columns are the same** (or all columns after a certain point) then we have no extra conditions and the algorithm runs in linear time.

However, **when the vary** in general, then breakpoints must end the same column when we choose among the candidates and this is the worst scenario that gives us quadratic run-time.

When we **run spreads short or long** then the breakpoint must end the same column on a spread and all columns have to use the same variation (long or short) unless we have just started a new spread. In that case the algorithm still runs in linear time but much slower as the active list will be 3 times the number of columns larger.

With **variation in textual element height** we do not have extra conditions for the optimality principle but the number of breaks in material of roughly a column height will be much higher so again the active list can get much larger (typical factor is between 10 and 50 without going into details here).

Finally, if **floats are involved**, the breakpoint ending a column must have exactly the same floats placed  up to this point. The complexity is not easy to determine, so we are not going to cover this here.

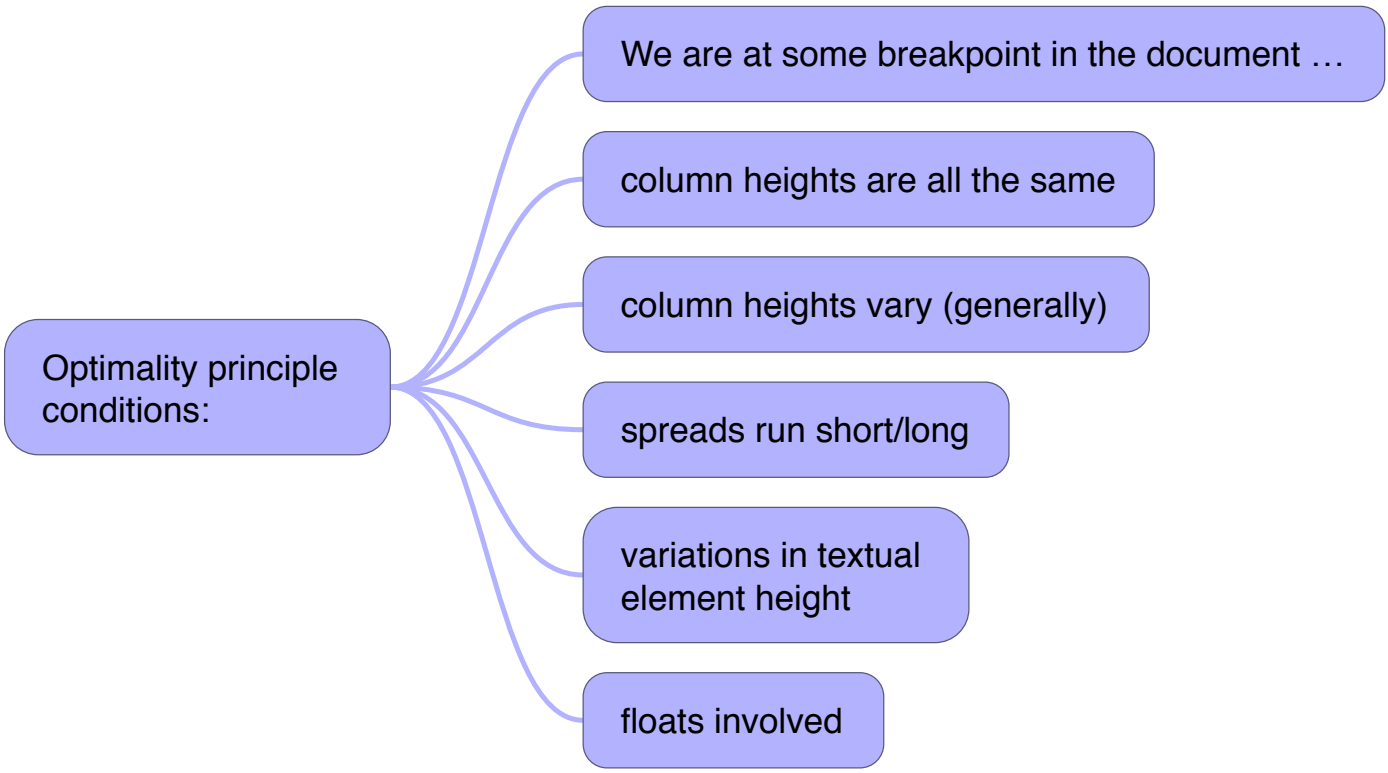
*/Alice goes floating/How?/So what now?/Optimized examples*

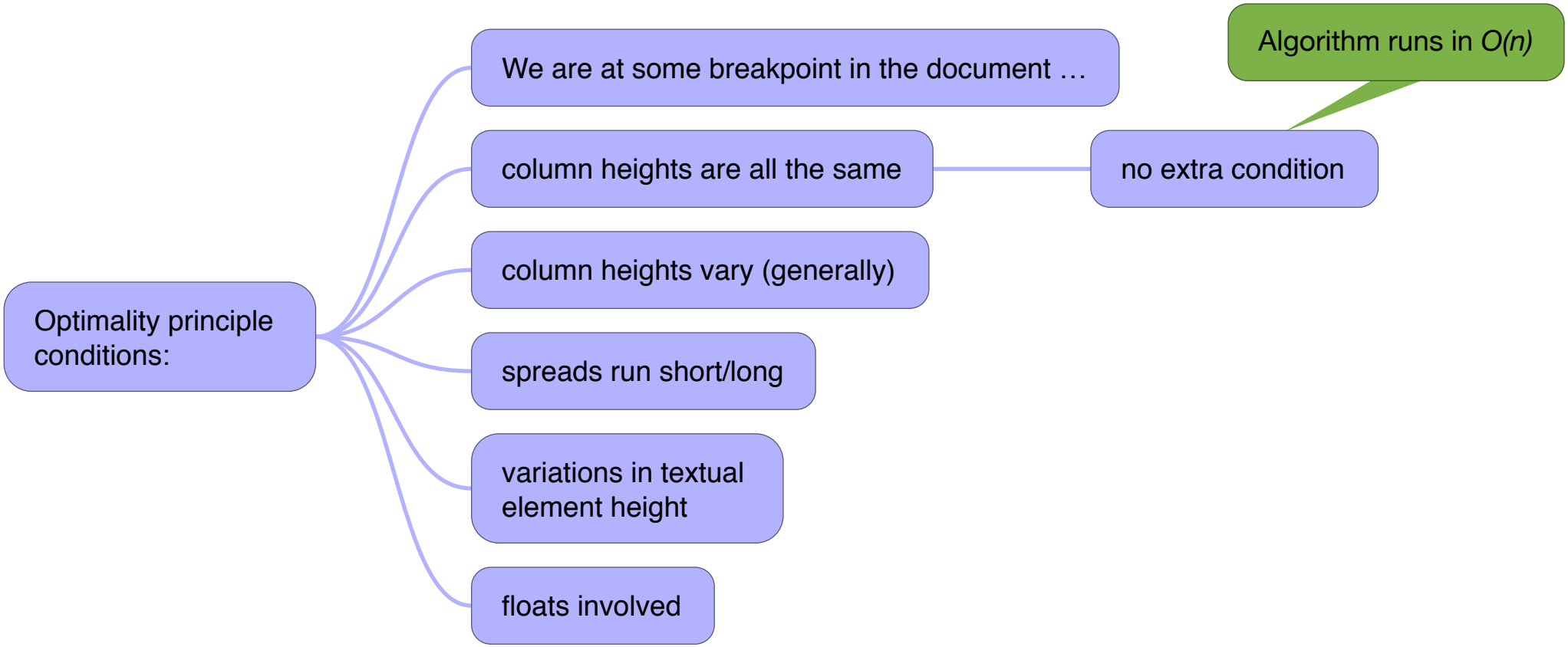
If we now take another look at our sample documents (without float) and apply the additional flexibility options (spread, paragraph variations, and both combined) we'll see that global optimizing becomes possible for all documents (with the exception of Carroll it is even enough to apply only one method).

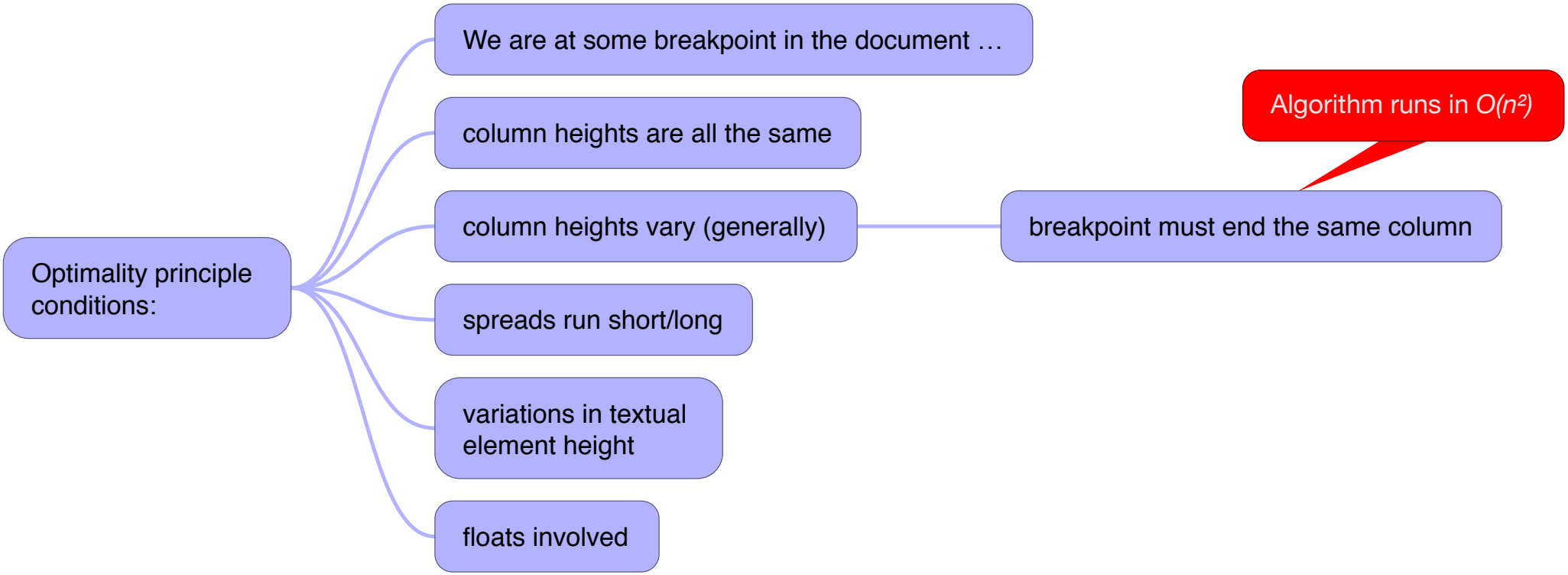
TUG-table-optimized-extended.png

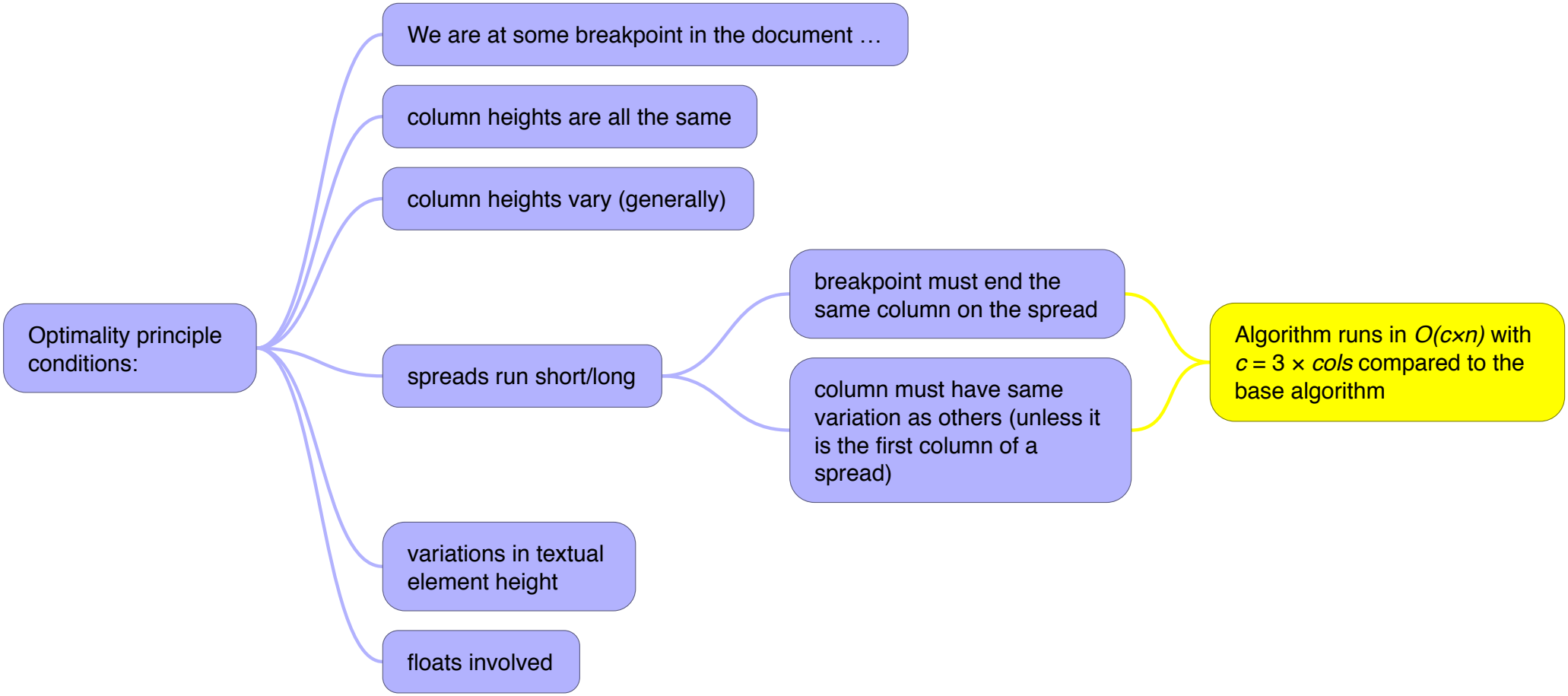
*/Alice goes floating/Adding floats*

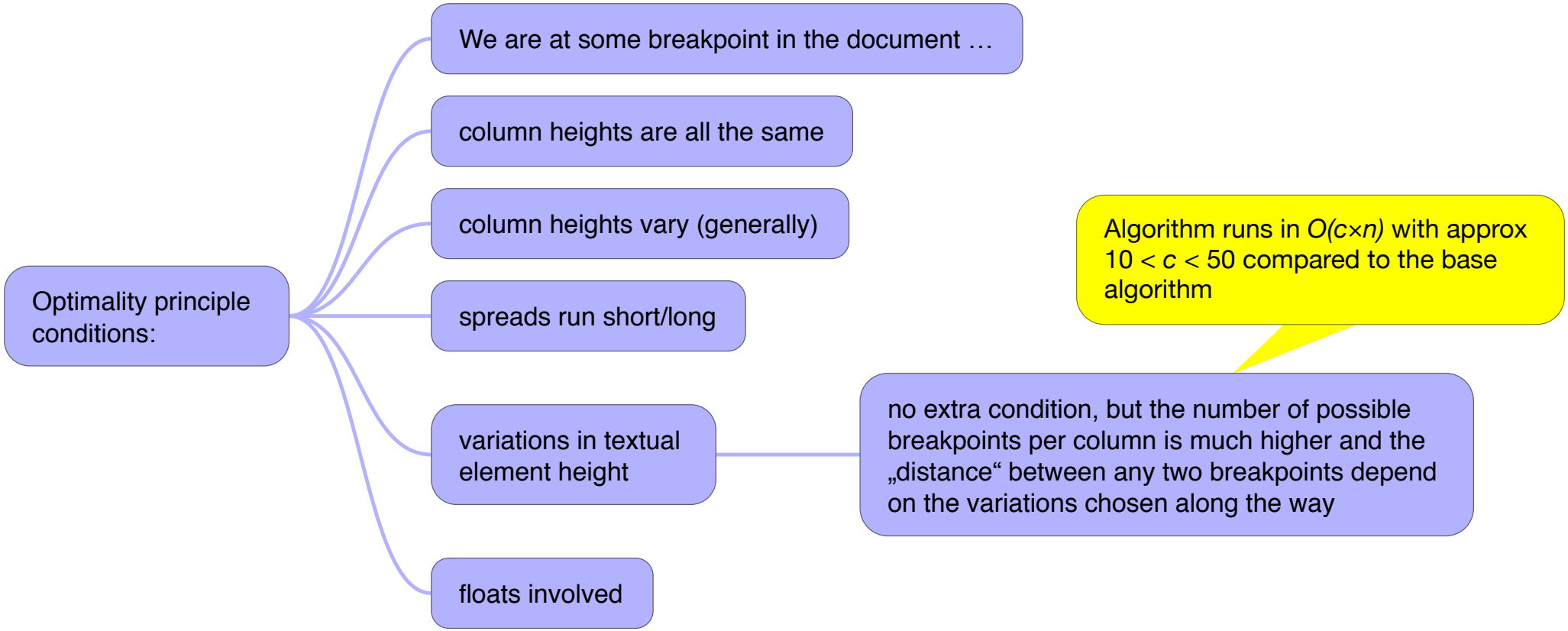
So let's add floats (and hopefully the Mad Hatter will help us) …

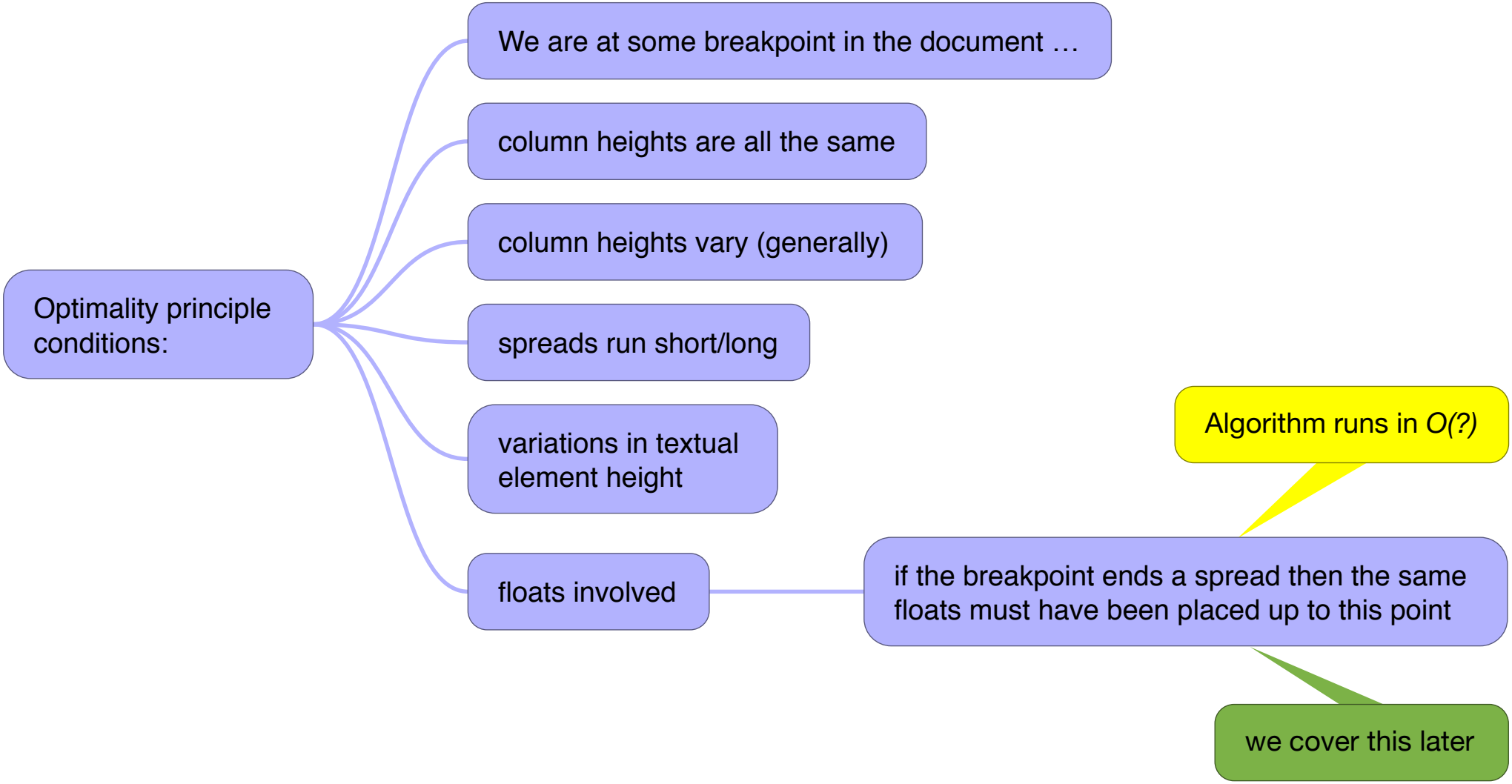

Optimality principle conditions:
We are at some breakpoint in the document …
column heights are all the same
column heights vary (generally)
spreads run short/long
variations in textual element height
floats involved


Optimality principle conditions:
We are at some breakpoint in the document …
column heights are all the same
column heights vary (generally)
spreads run short/long
variations in textual element height
floats involved
no extra condition
Algorithm runs in O(n)


Optimality principle conditions:
We are at some breakpoint in the document …
column heights are all the same
column heights vary (generally)
spreads run short/long
variations in textual element height
floats involved
breakpoint must end the same column
Algorithm runs in $O(n^2)$


Optimality principle conditions:
We are at some breakpoint in the document …
column heights are all the same
column heights vary (generally)
spreads run short/long
breakpoint must end the same column on the spread
column must have same variation as others (unless it is the first column of a spread)
Algorithm runs in O(c×n) with c = 3 × cols compared to the base algorithm
variations in textual element height
floats involved


Optimality principle conditions:
We are at some breakpoint in the document …
column heights are all the same
column heights vary (generally)
spreads run short/long
variations in textual element height
floats involved
no extra condition, but the number of possible breakpoints per column is much higher and the „distance“ between any two breakpoints depend on the variations chosen along the way
Algorithm runs in $O(c \times n)$ with approx $10 < c < 50$ compared to the base algorithm


Optimality principle conditions:
We are at some breakpoint in the document …
column heights are all the same
column heights vary (generally)
spreads run short/long
variations in textual element height
floats involved
if the breakpoint ends a spread then the same floats must have been placed up to this point
Algorithm runs in O(?)
we cover this later

| | document | | active list | | paragraphs | | available looseness | | | | | vertical badness | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | columns | blocks | max | average | total | variable | -1/0 | -1/1 | -1/2 | 0/1 | 0/2 | good | bad | ugly |
| **Alice in Wonderland** | 72 | | | | 833 | | | | | | | 69 | 0 | **2+1** |
| base | – | 6947 | 37 | 12 | | | | | | | | *no solution* | | |
| − spread | – | 6947 | 432 | 122 | | | | | | | | *no solution* | | |
| + variations | – | 9498 | 598 | 54 | | 111 | 6 | 15 | 0 | 89 | 1 | 73 | 1 | – |
| + variations, spread | 70 | 9498 | 7076 | 488 | | | | | | | | 71 | 1 | – |
| **Call of the Wild** | 78 | | | | 340 | | | | | | | 64 | 1 | **9+4** |
| base | – | 9148 | 9 | 2 | | | | | | | | *no solution* | | |
| − spread | 78 | 9148 | 263 | 134 | | | | | | | | 78 | – | – |
| + variations | 78 | 14970 | 263 | 67 | | 139 | 11 | 3 | 0 | 124 | 1 | 78 | – | – |
| + variations, spread | 78 | 14970 | 3156 | 704 | | | | | | | | 78 | – | – |
| **Grimm's Fairy Tales** | 236 | | | | 1041 | | | | | | | 212 | 6 | **6+12** |
| base | – | 27908 | 22 | 4 | | | | | | | | *no solution* | | |
| − spread | 234 | 27908 | 485 | 319 | | | | | | | | 234 | – | – |
| + variations | 238 | 59111 | 437 | 99 | | 441 | 10 | 50 | 21 | 318 | 42 | 238 | – | – |
| + variations, spread | 236 | 59111 | 5532 | 1030 | | | | | | | | 236 | – | – |
| **Pride and Prejudice** | 316 | | | | 2127 | | | | | | | 292 | 8 | **7+9** |
| base | 318 | 34645 | 39 | 14 | | | | | | | | 318 | – | – |
| − spread | 316 | 34645 | 486 | 347 | | | | | | | | 318 | – | – |
| + variations | 320 | 56861 | 633 | 70 | | 483 | 10 | 51 | 6 | 397 | 19 | 320 | – | – |
| + variations, spread | 316 | 56861 | 7596 | 837 | | | | | | | | 316 | – | – |

# Optimized examples

| | document | | active list | | paragraphs | | available looseness | | | | | vertical badness | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | columns | blocks | max | average | total | variable | -1/0 | -1/1 | -1/2 | 0/1 | 0/2 | good | bad | ugly |
| **Alice in Wonderland** | 72 | | | | 833 | | | | | | | 69 | 0 | **2+1** |
| base | – | 6947 | 37 | 12 | | | | | | | | | *no solution* | |
| + spread | – | 6947 | 432 | 122 | | | | | | | | | *no solution* | |
| + variations | – | 9498 | 598 | 54 | | 111 | 6 | 15 | 0 | 89 | 1 | 73 | 1 | – |
| + variations, spread | 70 | 9498 | 7076 | 488 | | | | | | | | 71 | 1 | – |
| **Call of the Wild** | 78 | | | | 340 | | | | | | | 64 | 1 | **9+4** |
| base | – | 9148 | 9 | 2 | | | | | | | | | *no solution* | |
| + spread | 78 | 9148 | 263 | 134 | | | | | | | | 78 | – | – |
| + variations | 78 | 14970 | 263 | 67 | | 139 | 11 | 3 | 0 | 124 | 1 | 78 | – | – |
| + variations, spread | 78 | 14970 | 3156 | 704 | | | | | | | | 78 | – | – |
| **Grimm's Fairy Tales** | 236 | | | | 1041 | | | | | | | 212 | 6 | **6+12** |
| base | – | 27908 | 22 | 4 | | | | | | | | | *no solution* | |
| + spread | 234 | 27908 | 485 | 319 | | | | | | | | 234 | – | – |
| + variations | 238 | 59111 | 437 | 90 | | 441 | 10 | 50 | 21 | 318 | 42 | 238 | – | – |
| + variations, spread | 236 | 59111 | 5532 | 1030 | | | | | | | | 236 | – | – |
| **Pride and Prejudice** | 316 | | | | 2127 | | | | | | | 292 | 8 | **7+9** |
| base | 318 | 34645 | 39 | 14 | | | | | | | | 318 | – | – |
| + spread | 316 | 34645 | 486 | 347 | | | | | | | | 318 | – | – |
| + variations | 320 | 56861 | 633 | 70 | | 483 | 10 | 51 | 6 | 397 | 19 | 320 | – | – |
| + variations, spread | 316 | 56861 | 7596 | 837 | | | | | | | | 316 | – | – |

# Adding floats

Rollup: 16 Minuten

*/Alice goes floating/Adding floats/Basic requirements*

In most documents float placement needs to obey certain rules, e.g.,

- sequential order of floats
- placement after (or at least visible from) the main call-out

While the above rules usually have to be enforced, the requirement that floats should be visible from their call-out is more a „wish“, as technically this is often simply impossible to guarantee for all floats.

With such rules in force there are still many possible placements that need to be judged according to some quality measurement. So one important question to ask is: what are good quality measures that distinguish different placements?

*/Alice goes floating/Adding floats/Issues*

Different float placements add flexibility to the pagination process as they change the column heights available for text.

However, this also means that (nearly) all placements need to be evaluated separately and that candidate solutions can only be collapsed if the same set of floats has been typeset at a particular breakpoint.

*/Alice goes floating/Adding floats/Issues/With different placement areas...*

4 columns top + bottom (no span) = 9 areas

#trials = (n+ m)! / ( n! m! )

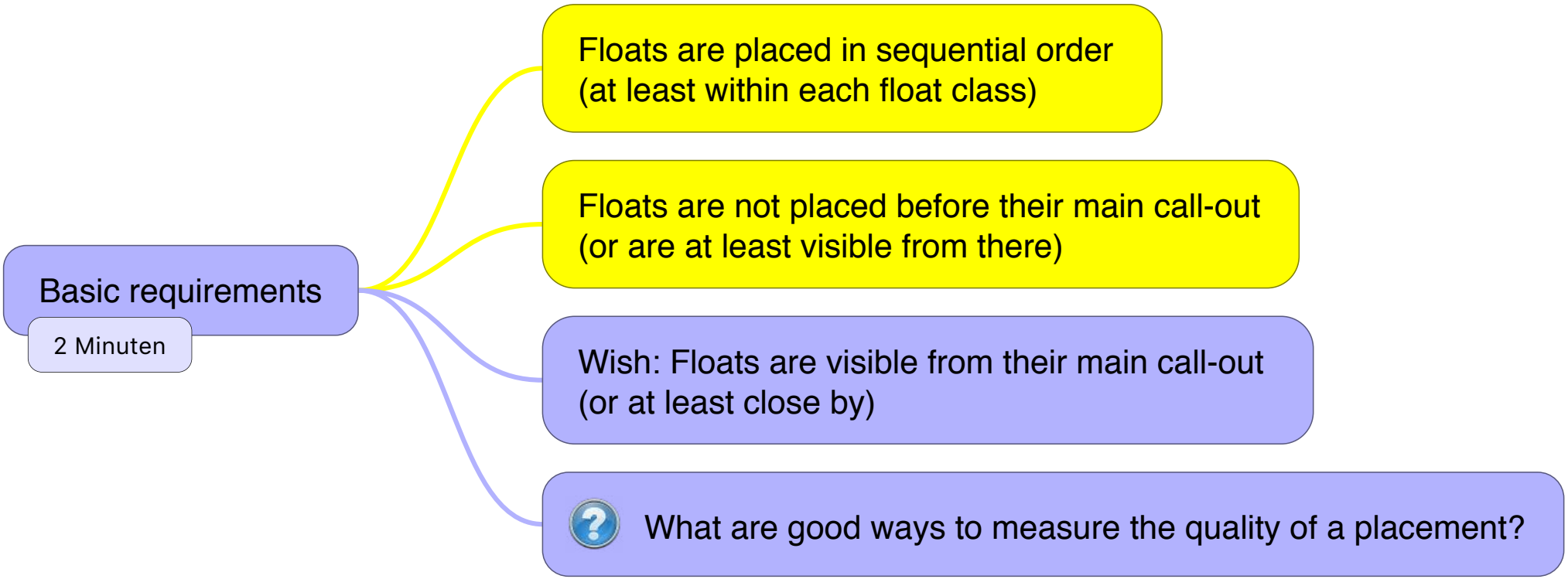
So if we have one additional float we increase by a factor of (n+1+m) / (n+1)

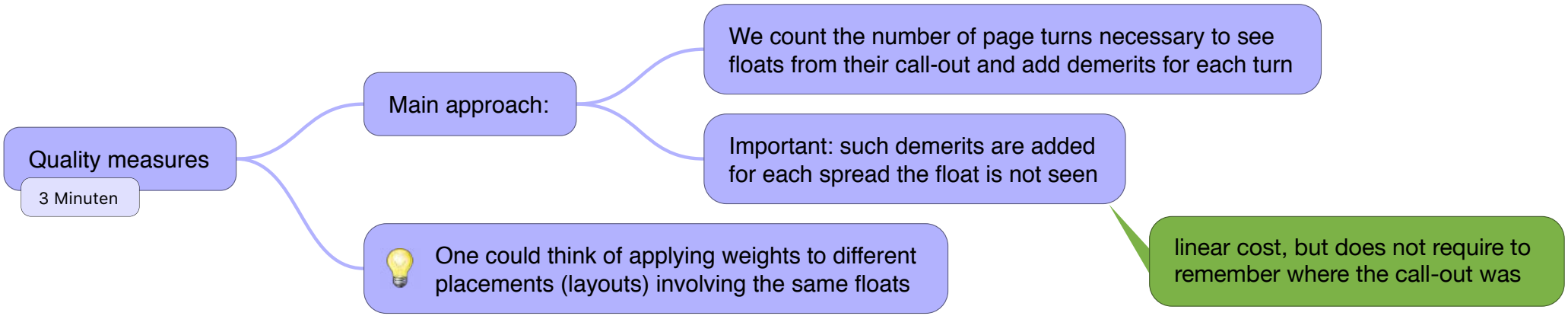
n=3 m=9 -> trials = 220
n=4 m=9 -> trials = 715
n= 5 -> trials = 2002
n=6 -> trials = 5005
n=7 -> trials = 11404
n=8 -> trials = 24310

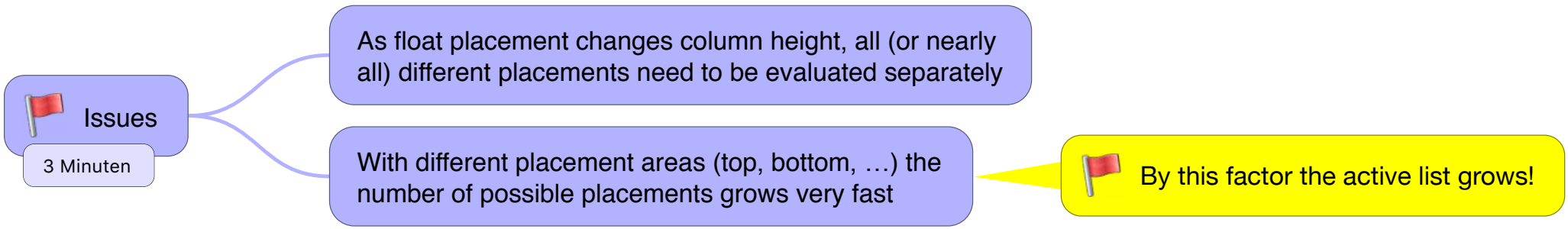
*/Alice goes floating/Adding floats/Pruning approaches*

It is therefore important to identify inferior placements early on to ensure that the algorithm performs in acceptable time.

At the same time it is necessary to keep enough candidate placements to ensure that the algorithm does not run out of options.


Basic requirements
2 Minuten
Floats are placed in sequential order
(at least within each float class)
Floats are not placed before their main call-out
(or are at least visible from there)
Wish: Floats are visible from their main call-out
(or at least close by)
What are good ways to measure the quality of a placement?


Quality measures
3 Minuten
Main approach:
We count the number of page turns necessary to see floats from their call-out and add demerits for each turn
Important: such demerits are added for each spread the float is not seen
linear cost, but does not require to remember where the call-out was
One could think of applying weights to different placements (layouts) involving the same floats


Issues
3 Minuten
As float placement changes column height, all (or nearly all) different placements need to be evaluated separately
With different placement areas (top, bottom, …) the number of possible placements grows very fast
By this factor the active list grows!

Pruning approaches

4 Minuten

- If we do not drop infeasible placements fast, then the running time of the algorithm will be very slow
- If, on the other hand, we drop too many, we may not find any solution at all
- Different situations will require different approaches
  - Perhaps start with rigorous settings and relax if we run out of placement options
- Examples:
- Conclusion

*/Alice goes floating/Adding floats/Pruning approaches/Examples:*

One of the problems in this area is that for all pruning techniques one can find counter-examples where they should not be applied.

*/Alice goes floating/Adding floats/Pruning approaches/Conclusion*

Conclusion: this is definitely an area that needs further research!

*/Alice goes floating/Adding floats/Interfaces*

Interfaces implemented in the current algorithm:

**- spread(s) setup**

Defines where on the spread floats can be placed, how many floats are allowed in total, and the number/order and initial height of individual columns.

**- float area(s) setup**

Number of floats in an area, effect on columns if float is placed and effect on other areas if float a float is added (e.g., other areas may be forbidden to receive floats or floats of a certain type).

**- call-out constraints**

Restricting floats to come after the call-out, not on an earlier column, not on an earlier page or not on an earlier spread.

**- manual placement options**

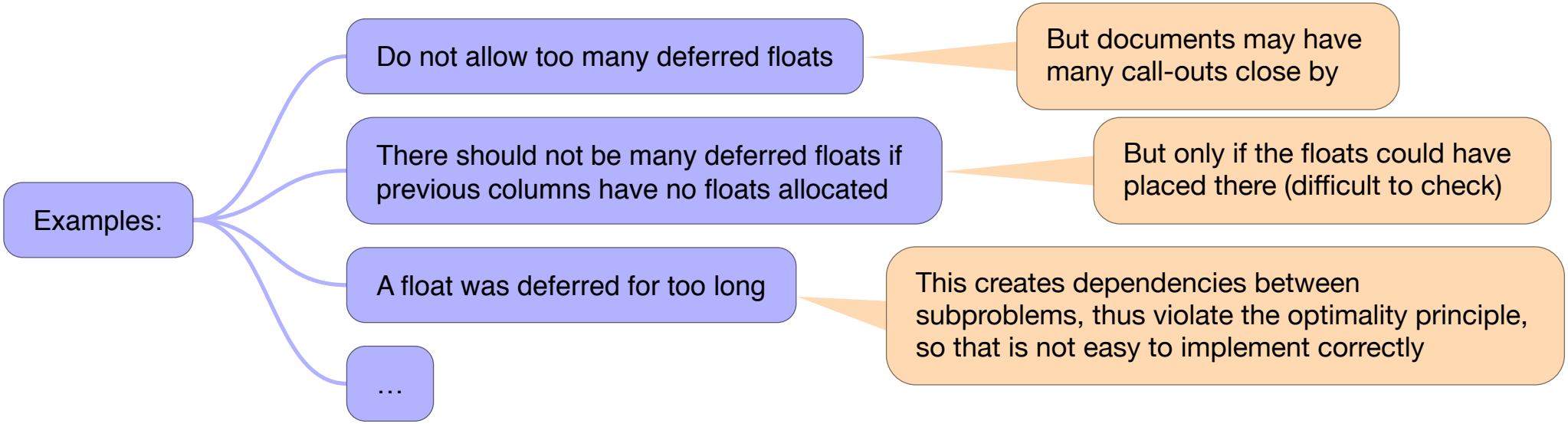
Possibility to require a float to appear on a certain spread or in a certain area or both. Possible, but not implemented extension would be to allow several such options in parallel.

*/Alice goes floating/The pagination framework/General approach*

The pagination framework implemented by the author is based on the LuaTeX engine.

It can be directly used with any TeX distribution that provides this engine. It is largely transparent to LaTeX so that all/most packages and extensions can be used with it, without adjustment. It does however, require the use of LuaTeX to interface with inner processing of the TeX engine.

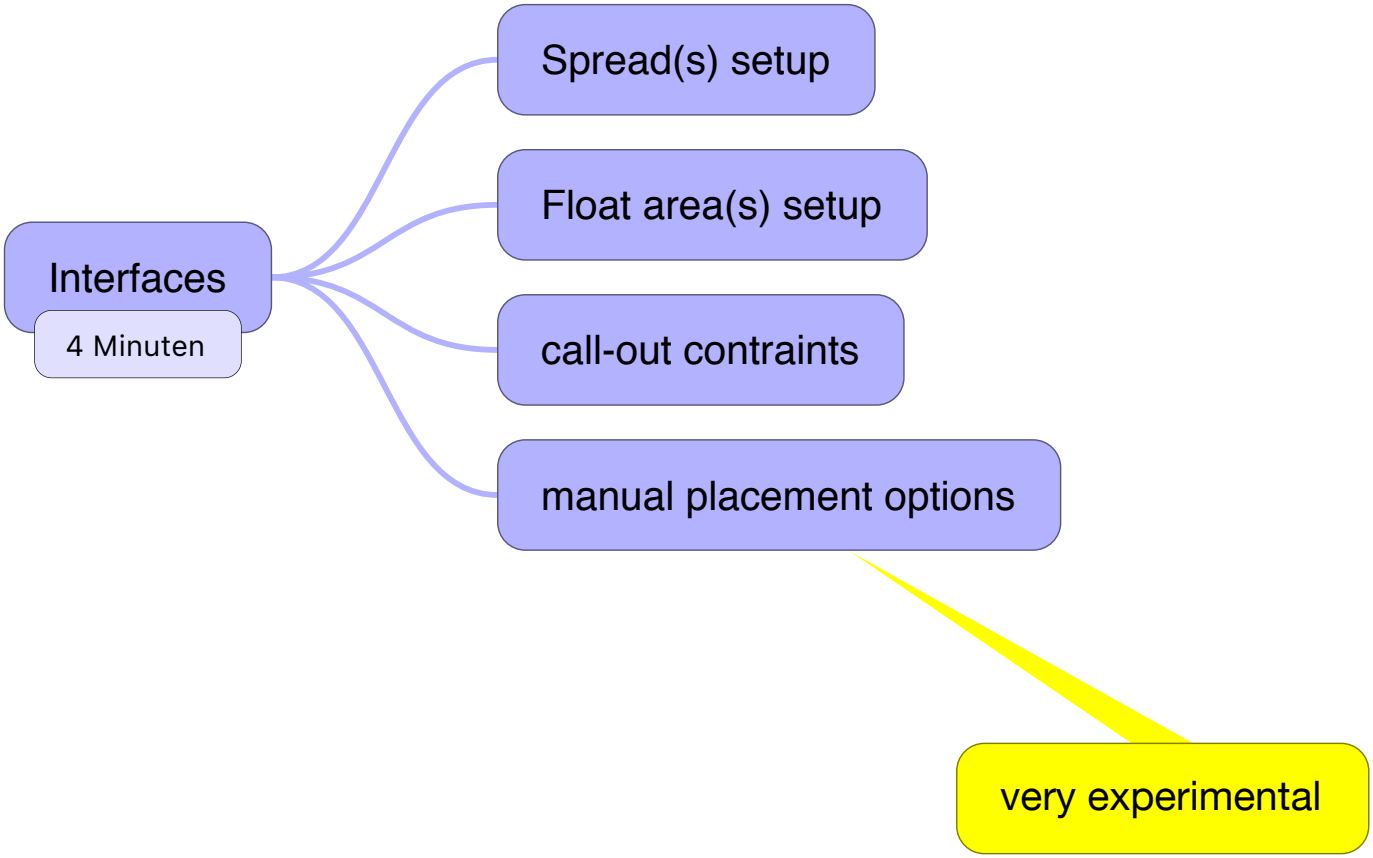
It is a framework as adjustments and extensions to the algorithm can be easily integrated. It consists of four major phases.

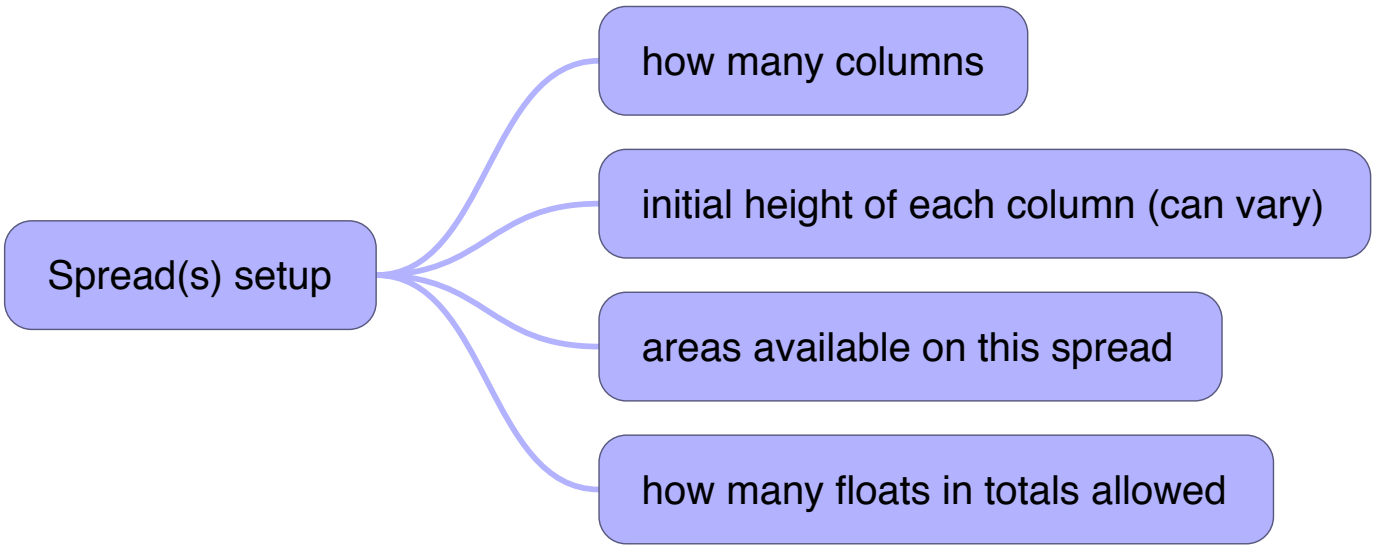

Examples:
Do not allow too many deferred floats
But documents may have many call-outs close by
There should not be many deferred floats if previous columns have no floats allocated
But only if the floats could have placed there (difficult to check)
A float was deferred for too long
This creates dependencies between subproblems, thus violate the optimality principle, so that is not easy to implement correctly
…

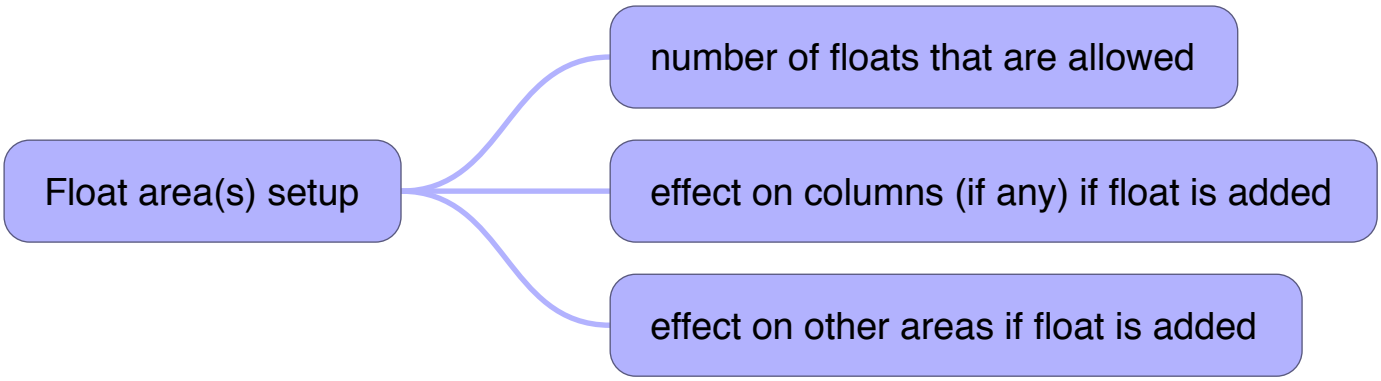
Pruning approaches

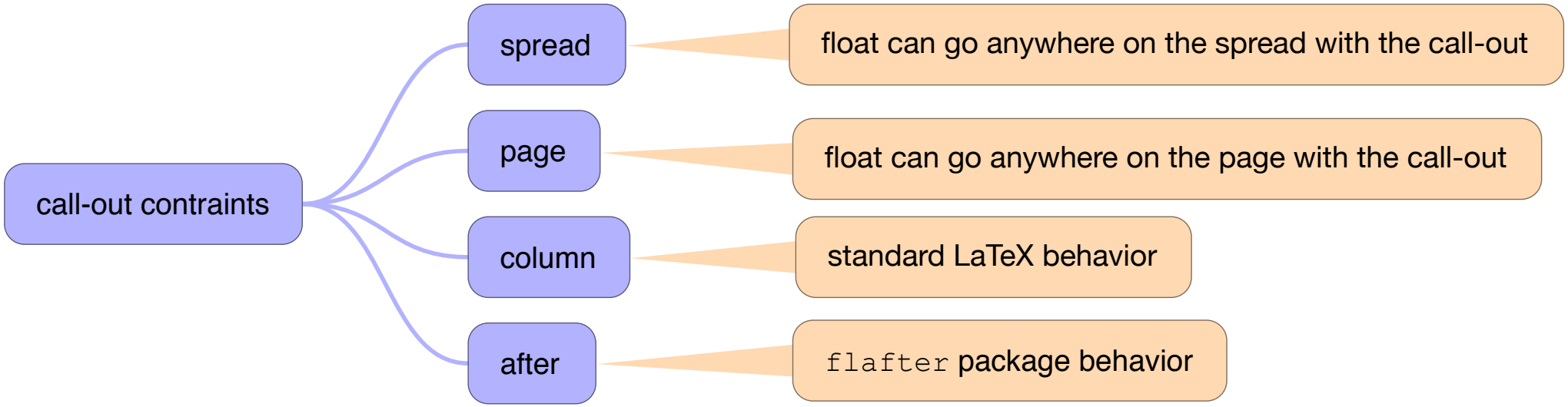
4 Minuten

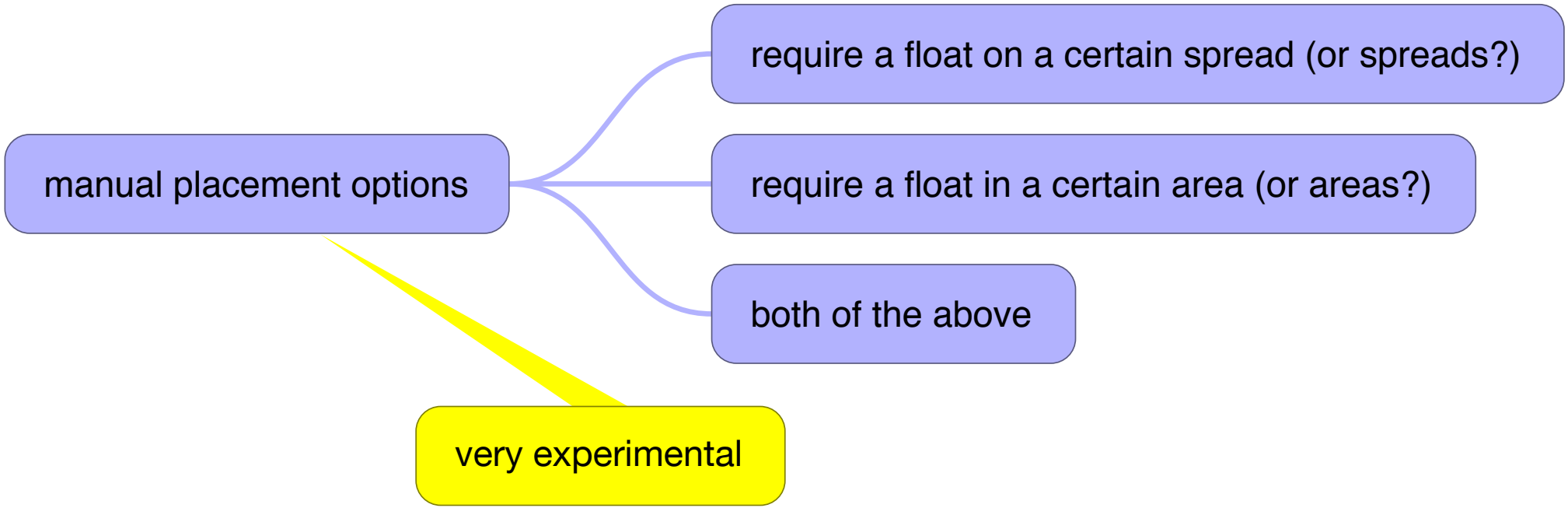
- If we do not drop infeasible placements fast, then the running time of the algorithm will be very slow
- If, on the other hand, we drop too many, we may not find any solution at all
- Different situations will require different approaches
  - Perhaps start with rigorous settings and relax if we run out of placement options
- Examples:
- Conclusion
  - Further research needed!


Interfaces
4 Minuten
Spread(s) setup
Float area(s) setup
call-out contraints
manual placement options
very experimental


Spread(s) setup
how many columns
initial height of each column (can vary)
areas available on this spread
how many floats in totals allowed

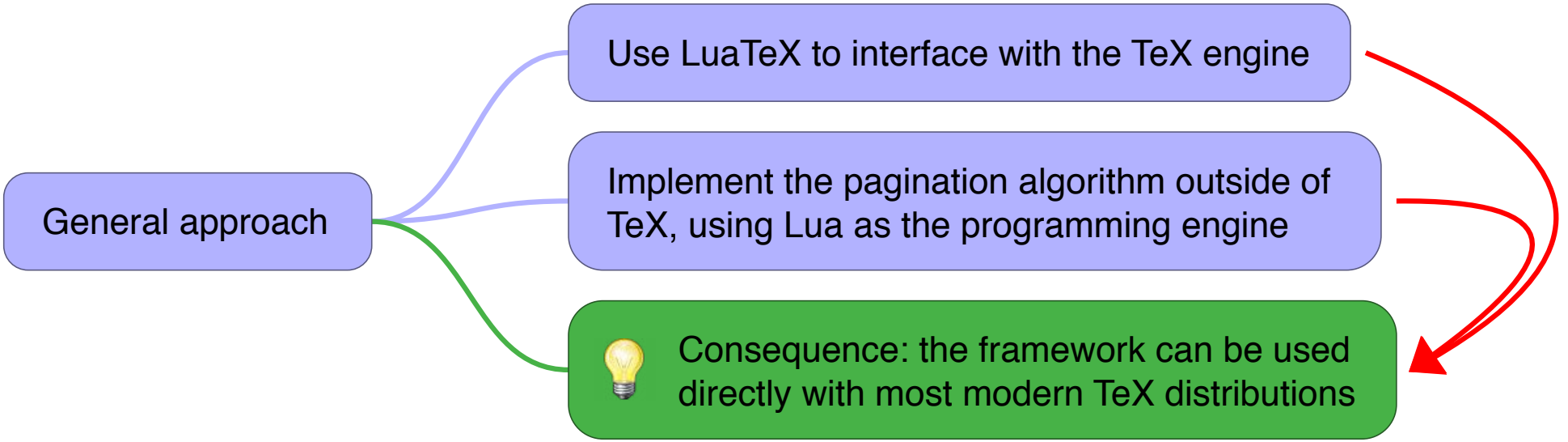

Float area(s) setup
number of floats that are allowed
effect on columns (if any) if float is added
effect on other areas if float is added


call-out contraints
spread
float can go anywhere on the spread with the call-out
page
float can go anywhere on the page with the call-out
column
standard LaTeX behavior
after
flafter package behavior


manual placement options
require a float on a certain spread (or spreads?)
require a float in a certain area (or areas?)
both of the above
very experimental

# The pagination framework

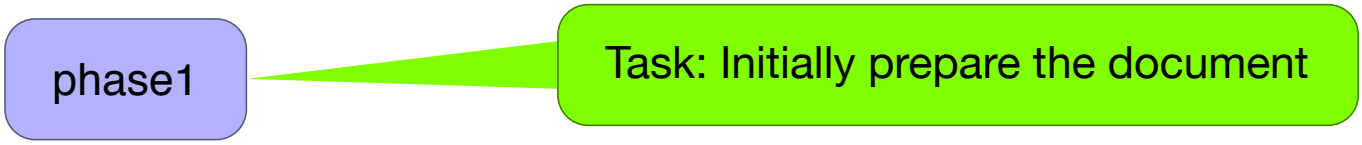
3 Minuten


General approach
Use LuaTeX to interface with the TeX engine
Implement the pagination algorithm outside of TeX, using Lua as the programming engine
Consequence: the framework can be used directly with most modern TeX distributions

The document, which consists of standard TEX files, is processed by a TEX engine without any modification
until all implicit content (e.g., table of content, bibliography, etc.) is generated and all cross-references are resolved.

The cross-references are not necessarily final (as the final pagination will be determined later) but this way they have hopefully the same space characteristics.

If that assumption does not hold, it is likely that you end up with an „impossible document“ that can not be processed with a globally optimizing pagination approach!


phase1
Task: Initially prepare the document

*/Alice goes floating/The pagination framework/phase2*

The engine is modified to interact with TEX's way of filling the main vertical list (from which, in an asynchronous way, TEX later cuts column material for pagination).
In particular, whenever TEX is ready to move new vertical material to the main vertical list this material is intercepted and analyzed. Information about each block (vertical size, depth, stretchability if any and penalty of a breakpoint) is then gathered and written out to an external file. If possible, data is accumulated, e.g., several objects in a row without any possibility for breaking them up are written out as a single data point to reduce later processing.
The modification is also able to interpret special flags (implemented as new types of "whatsit nodes" in TEX engine lingo) that can signal the start/end or switch of an explicit variation in the input source. This information is then used to structure the corresponding data in the output file for later processing.
The second modification to the engine is to intercept the generation of paragraphs targeted for the main galley prior to TEX applying line breaking:

- For each horizontal list that is passed to the line-breaking algorithm the framework algorithm then determines the number of acceptable variations in "looseness" within the specified parameter settings.

- For each possible variation it then does a paragraph breaking trial to determine the exact sequence of lines, vertical spaces and associated penalties under a specific "looseness" value.

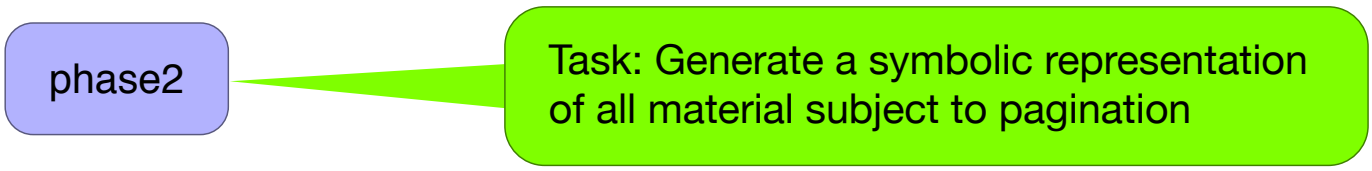
- The results of each trial is externally recorded together with the associated "looseness" value of the variation.

- Finally, instead of adding a vertical list representing the paragraph to the main vertical list, a single special node is passed so that the paragraph material is not collected again by the first modification described above.

As the result of this phase the external file will hold an abstraction of the document galley material including marked up variations for each paragraph.

*/Alice goes floating/The pagination framework/phase 2b*

This phase is a sub-phase of phase 2 (could be done in one go) and provides the call-out positions within the symbolic galley representation as a separate list for faster processing during gloabl optimization.


phase2
Task: Generate a symbolic representation of all material subject to pagination


phase 2b
Task: Produce a float callout list

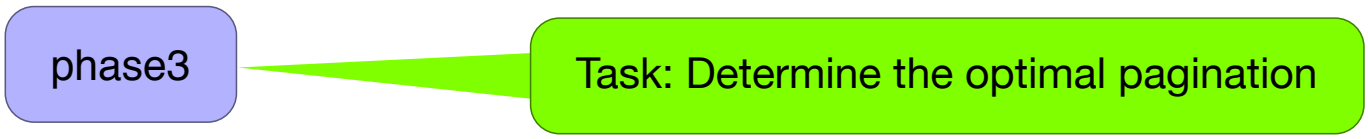
The result of phase 2 and 2b is used as input to a global optimizing algorithm modeled after the Knuth/Plass algorithm for line breaking that uses dynamic programming to determine an optimal sequence of page/ column breaks throughout the whole document. Com- pared to the line-breaking algorithm this page-breaking algorithm provides the following additional features:

- Support for variations within the input: This is used to automatically manage variant break sequences resulting from different paragraph breakings calculated in phase 2, but could also be used to support, for example, variations of figures in different size or similar applications.

- Support for shortening or lengthening the vertical size of double spreads to enable better columns/ page breaks across the whole document.

- Global optimization is guided by parameters that allow a document designer to balance the importance of individual aspects (e.g., avoiding widows against changing the page length or using sub- optimal paragraphs) against each other.

The result of this phase will be a sequence of optimal page break positions within the input together with length information for all pages/columns for which it applies. Also recorded is which of the variants have been chosen when selecting the optimal sequence.


phase3
Task: Determine the optimal pagination

This phase again uses a modified TEX engine that is capable of interpreting and using the results of the previous phases. For this it hooks into the same places as the modifications in phase 2, but this time applying different actions:

- To begin with, the vertical target size for gathering a complete column will be artificially set to the largest legal dimension so that by itself the TEX algorithm will not mistakenly break up the galley at an unwanted place due to some unusual combination of data.

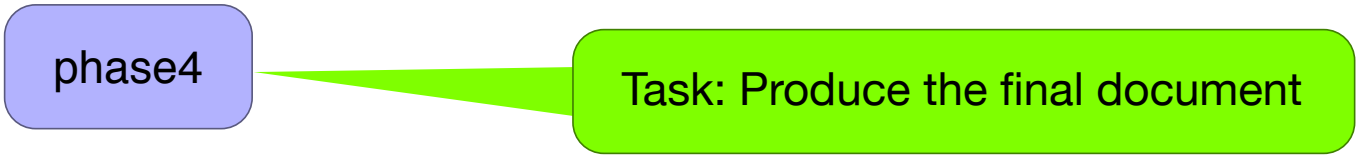
- Whenever TEX gets ready to apply line breaking to paragraph material for the main vertical list the modification looks up with which "looseness" this paragraph should be typeset and adjusts the necessary parameters so that TEX generates the lines corresponding to the variation selected in the optimal break sequence for the whole document determined in pagination phase 3.

- While TEX is moving objects to the main vertical list the algorithm keeps track of the galley blocks seen so far and when it is time for a column break according to the optimal solution it will explicitly place a suitable forcing penalty onto the main vertical list so that TEX is guaranteed to use this place to end the current column or page. Again as a safety measure other penalties seen at this point that should not result in a column break will be either dropped or otherwise rendered harmless so that TEX's internal (greedy) page-breaking algorithm is not misinterpreting them as a "best break" by mistake.

- Finally, whenever TEX has finished a column (due to the fact that we have added an explicit penalty in the previous step) we will arrange for the correct target dimensions for the current column according to the data from pagination phase 3. This is done immediately after TEX has decided what part of the galley it will pack up for use in its "output routine" (which is a set of TEX macros) but before this routine is actually called.

The result is a paginated document with optimized column breaks across the whole document.

It is however not necessarily a correctly formatted document (in case generated text depends on the final pagination) as explained earlier.


phase4
Task: Produce the final document

*/Alice goes floating/The „emergency stretch“ idea*

The line-breaking algorithm of TeX implements the idea of „emergency stretch“ if the algorithm runs out of alternatives to optimize.

A similar approach can be used when globally optimizing the pagination.

As a result more pagination options are bing considered and those that originally had „infinite“ badness are now becoming measurable and comparable.

However in contrast to line breaking, pagination often has to deal with columns with little or no flexibility whatsoever. If there is no flexibility then the approach is invalid and would result in solutions that would look horrendous.

It is therefore important to only apply this method with columns that do have at least some initial flexibility. In that case, all experiments so far have shown good results.

*/Alice goes floating/Examples*

On this slide we show the performance and results produced by the algorithm when paginating Alice in Wonderland using different parameter settings and more or less flexibility added through different options provided by the algorithm.

*/Alice goes floating/Open issues*

Current state of affairs …

*/Alice goes floating/Open issues/code*

There are a number of issues with the current code, e.g., it is still based on LuaTeX 0.8 and will not run without adjustments on the current version of the engine. (Basically, during the implementation a number of problems have been identified in the engine and those have since then been corrected — however, the code still implements workarounds based on the earlier interfaces)

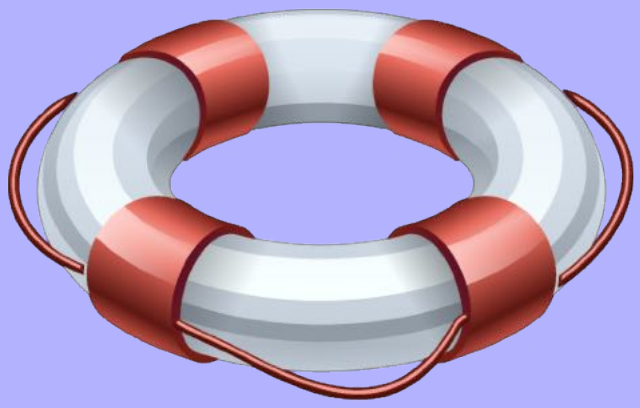
Footnotes are somewhat similar to floats and are currently not fully supported (in particular the ability to split footnotes across columns).

The pruning logic for float placements is still in its infancy and needs further thoughts.

And of course there are most likely other bugs (some of which are known, others probably not).

*/Alice goes floating/Open issues/clumsy/bad interfaces*

All customization interfaces so far are really just prove of concept implementations and need to be provided in a different ways for end users.



# The „emergency stretch“ idea

3 Minuten

… restart assuming an additional amount of (non-existent) flexibility available per column

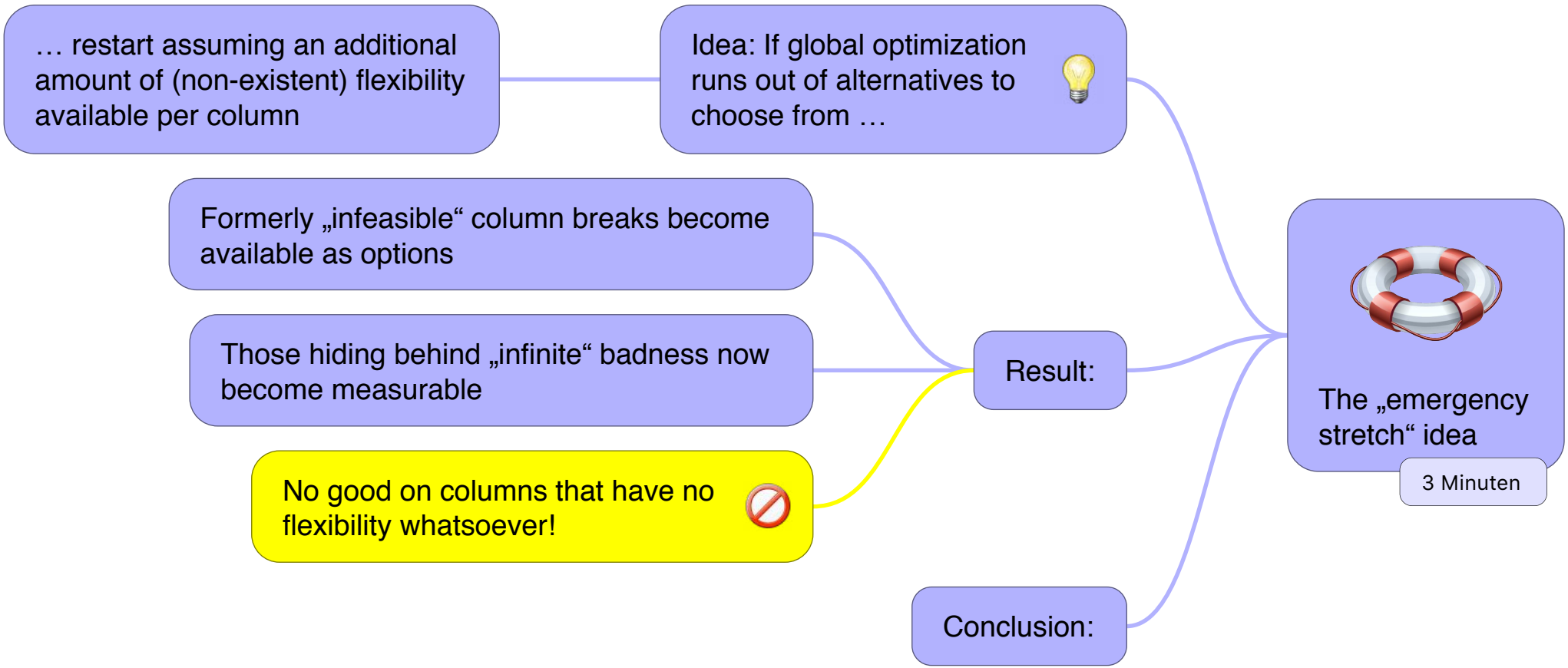
Idea: If global optimization runs out of alternatives to choose from …

Result:

Conclusion:

The „emergency stretch“ idea

3 Minuten


… restart assuming an additional amount of (non-existent) flexibility available per column
Idea: If global optimization runs out of alternatives to choose from …
Formerly „infeasible“ column breaks become available as options
Those hiding behind „infinite“ badness now become measurable
No good on columns that have no flexibility whatsoever!
Result:
Conclusion:
The „emergency stretch“ idea
3 Minuten

The „emergency stretch“ idea

3 Minuten

Idea: If global optimization runs out of alternatives to choose from …

… restart assuming an additional amount of (non-existent) flexibility available per column

Result:

Conclusion:

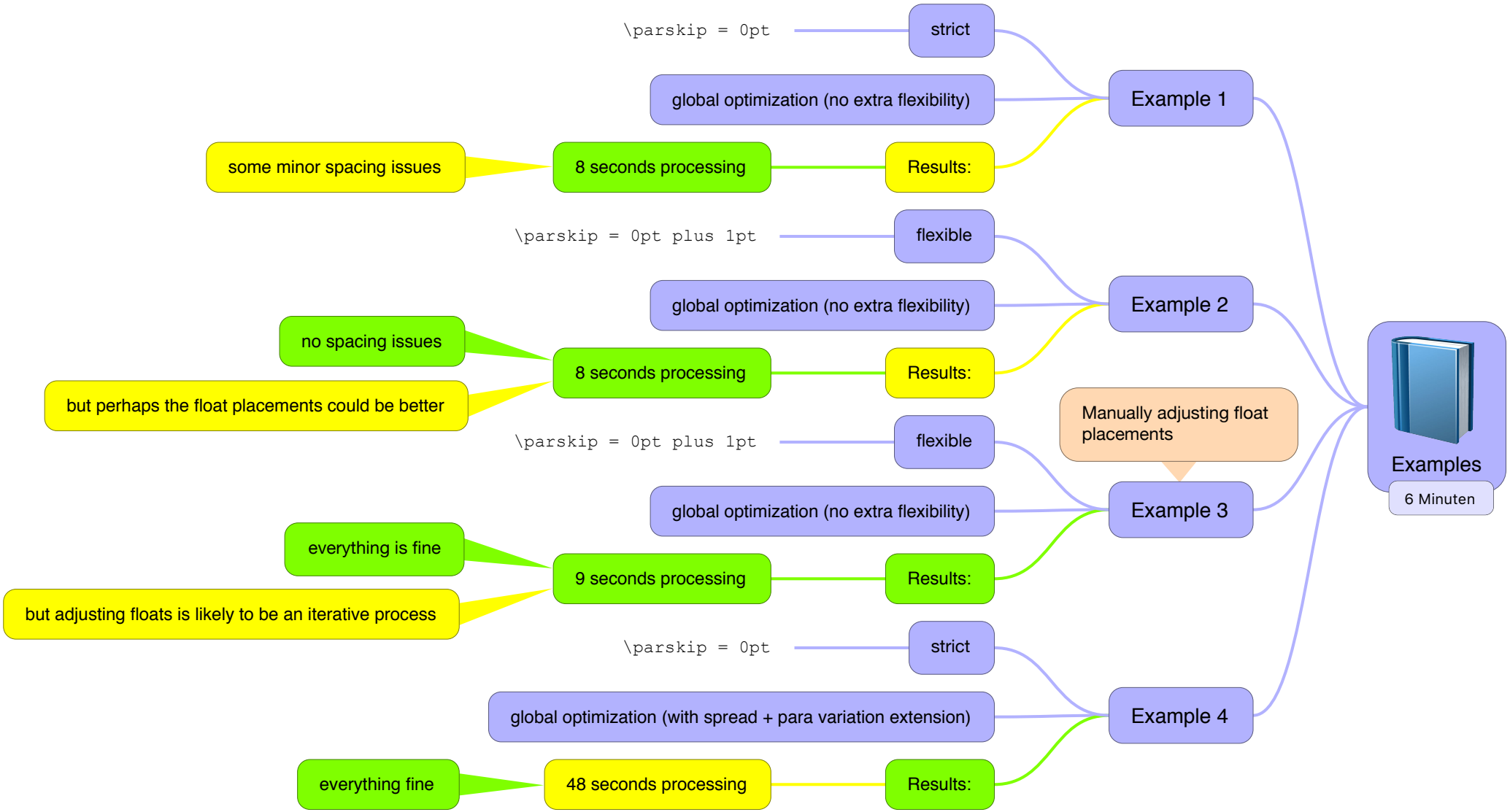
Don’t apply this on columns without flexibility

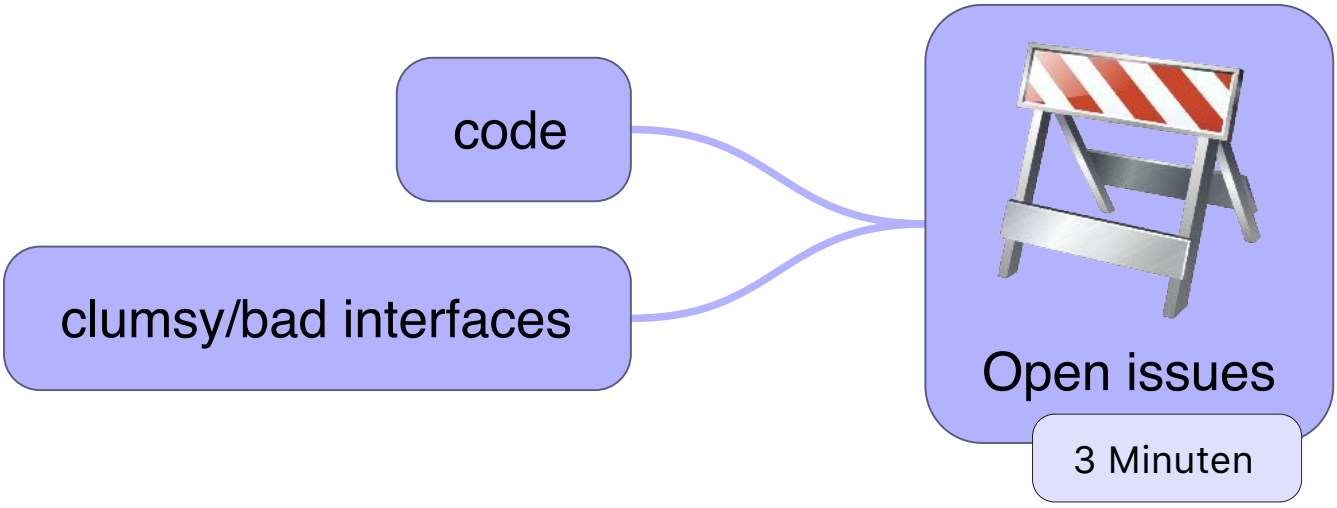
On all others proceed with fingers crossed

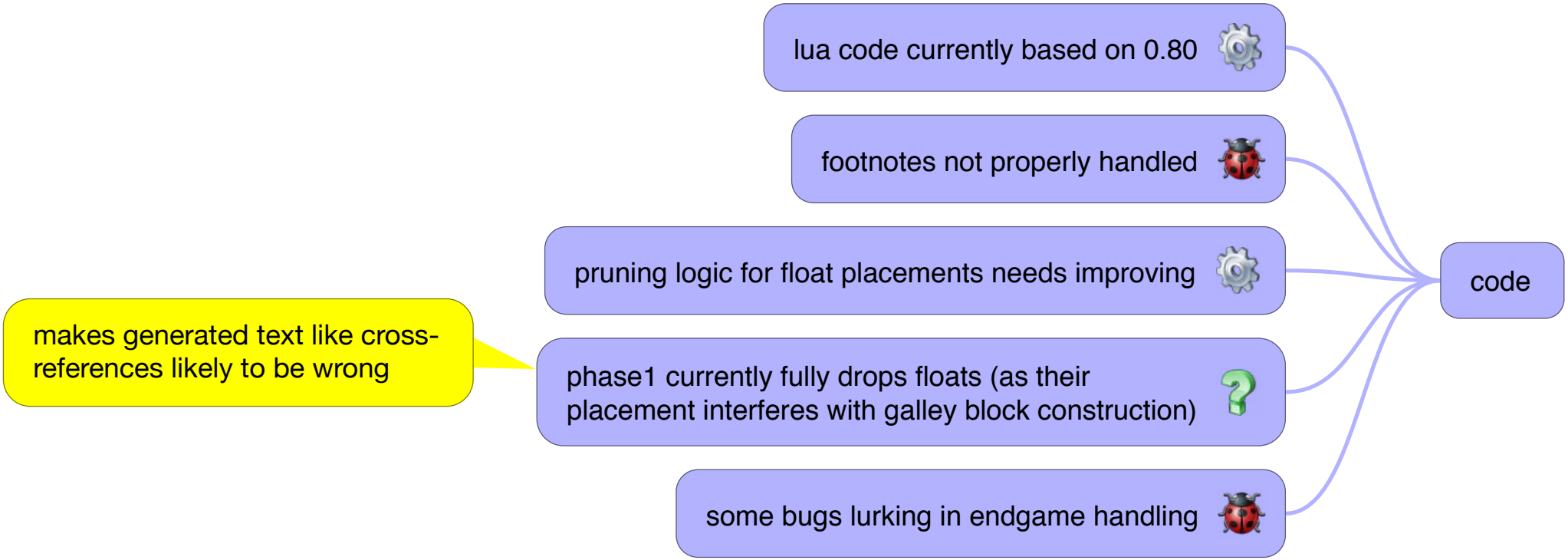
In most cases this will give you reasonable results!

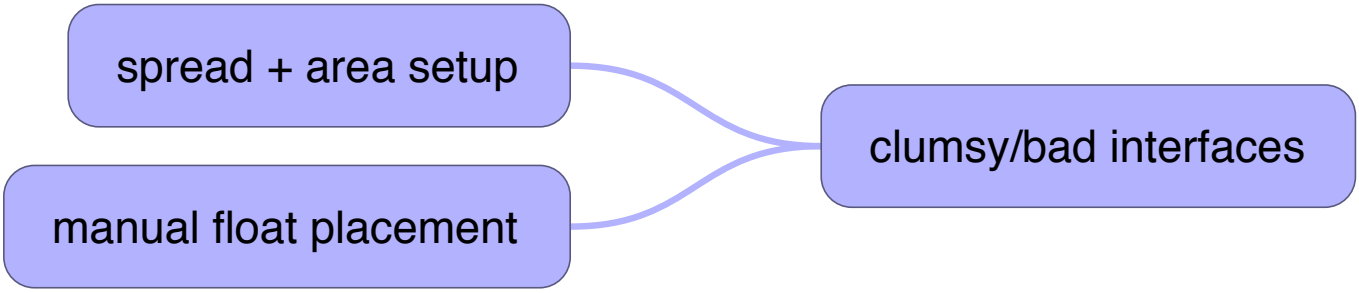
# Examples

6 Minuten


Examples
6 Minuten
Example 1
\parskip = 0pt
strict
global optimization (no extra flexibility)
Results:
8 seconds processing
some minor spacing issues
Example 2
\parskip = 0pt plus 1pt
flexible
global optimization (no extra flexibility)
Results:
8 seconds processing
no spacing issues
but perhaps the float placements could be better
Example 3
Manually adjusting float placements
\parskip = 0pt plus 1pt
flexible
global optimization (no extra flexibility)
Results:
9 seconds processing
everything is fine
but adjusting floats is likely to be an iterative process
Example 4
\parskip = 0pt
strict
global optimization (with spread + para variation extension)
Results:
48 seconds processing
everything fine


code
clumsy/bad interfaces
Open issues
3 Minuten

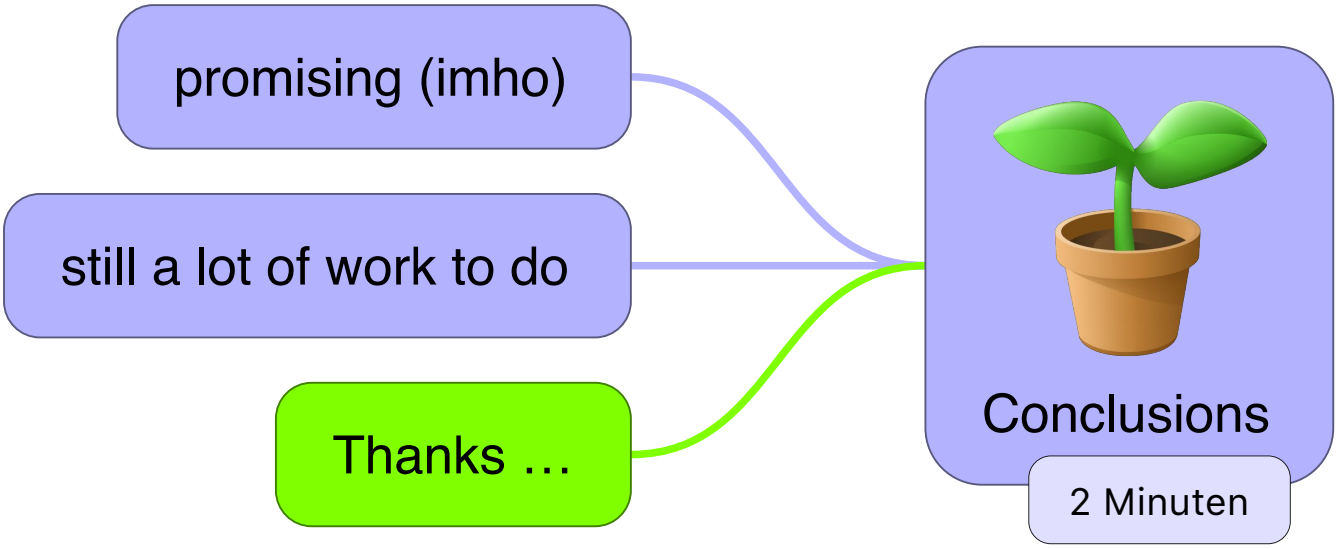

code
lua code currently based on 0.80
footnotes not properly handled
pruning logic for float placements needs improving
phase1 currently fully drops floats (as their placement interferes with galley block construction)
some bugs lurking in endgame handling
makes generated text like cross-references likely to be wrong


spread + area setup
manual float placement
clumsy/bad interfaces

In conclusion, the work so far looks fairly promising, but to turn this into a generally usable product a lot of work is still necessary.

# Conclusions

2 Minuten


promising (imho)
still a lot of work to do
Thanks …
Conclusions
2 Minuten


… to the LuaTeX team for providing the methods that made this possible

Thanks …

… and to you for listening for so long

Alice goes floating
Frank Mittelbach
TUG 2016, Toronto, Canada, July 2016