Beyond ASCII

Parsing programs with graphical presentations

Martijn Schrage
Doaitse Swierstra

Utrecht University

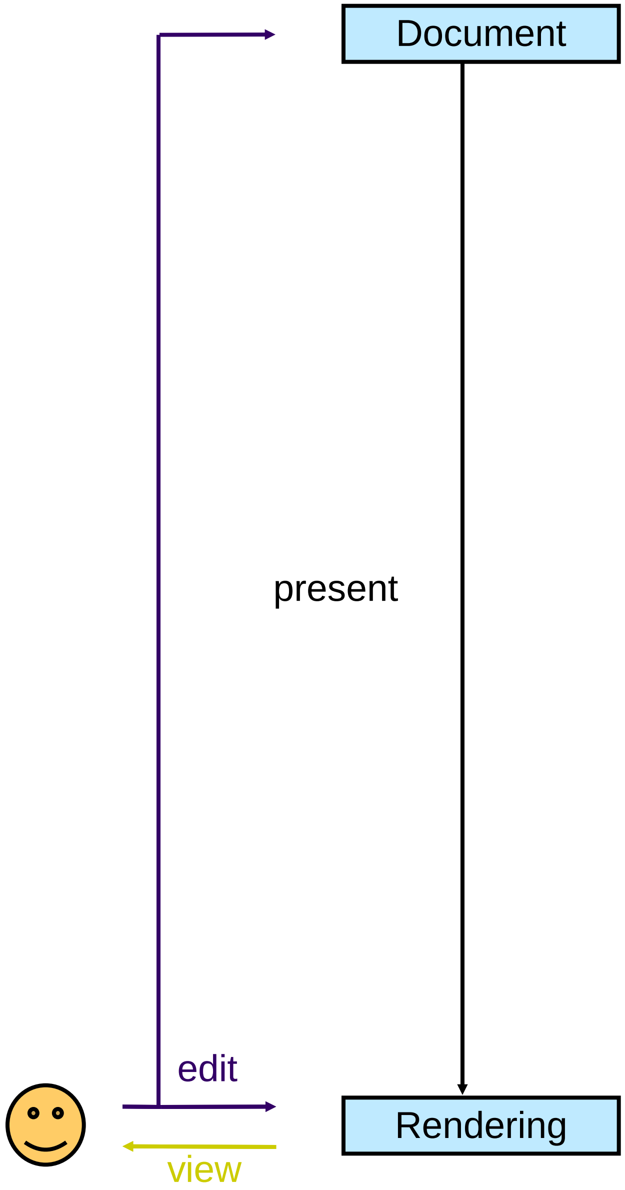
This talk

- Proxima overview
- Demo
- Document presentation
- Scanner and parser algorithms
- Conclusion

Proxima

- Generic presentation-oriented editor
- Graphical presentation with derived information
- Modeless mix of
  - Structural editing: e.g. change section to subsection
  - Free-text editing: e.g. delete [1+2, 5] → [15]
- Applications:
  - Source editor
  - Active documents
- ~15.000 lines of Haskell
- Web interface under development

Architecture


Document

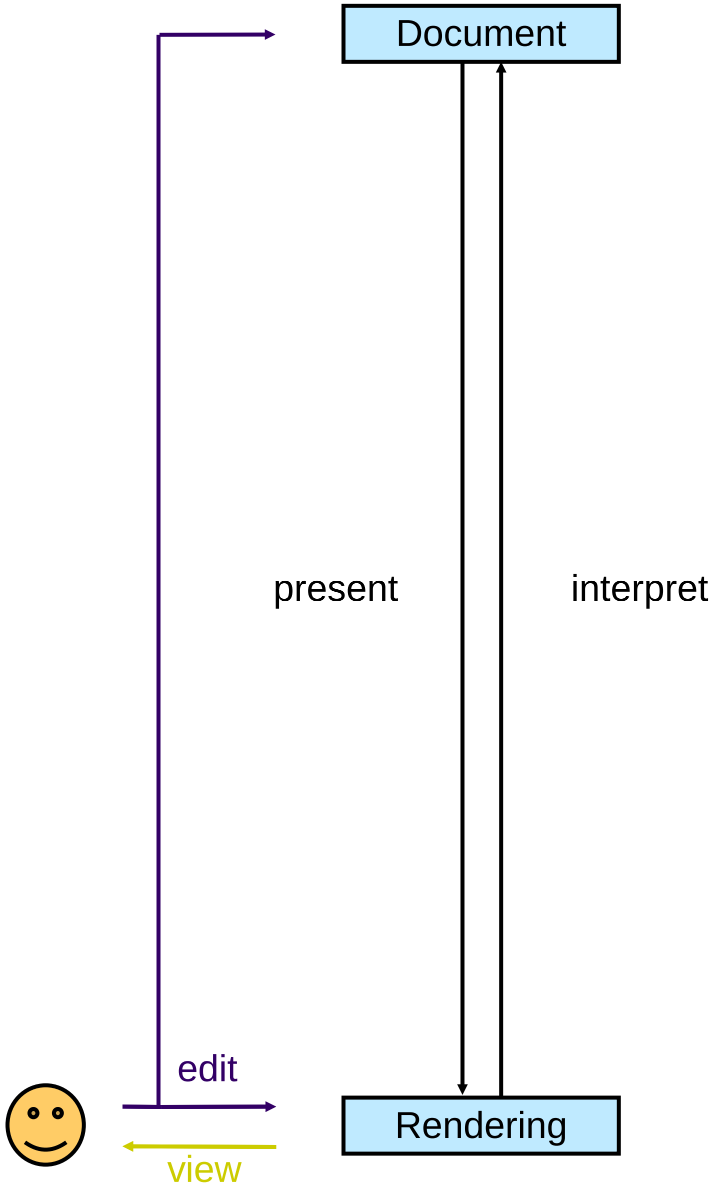
present

edit

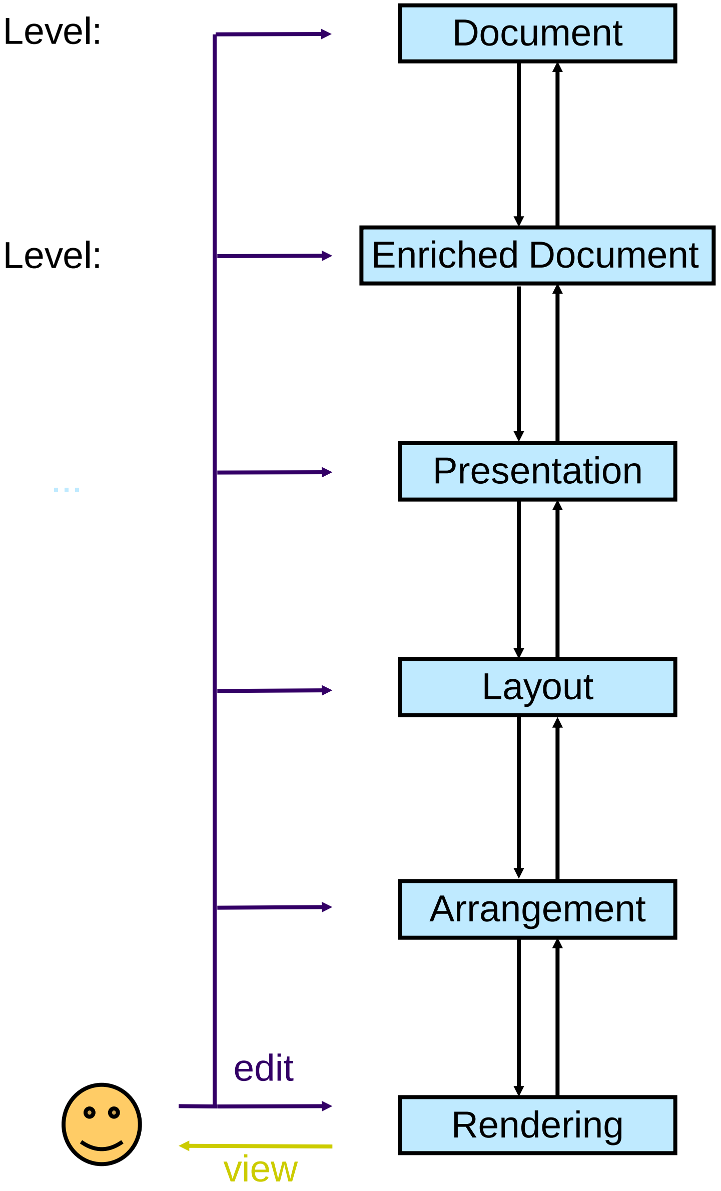
Rendering

view

Architecture



Architecture



Level: Document — Internal document tree

Level: Enriched Document — Document + derived values

... Presentation — Logical presentation: rows, columns, tokens, etc.

Layout — Presentation + explicit whitespace

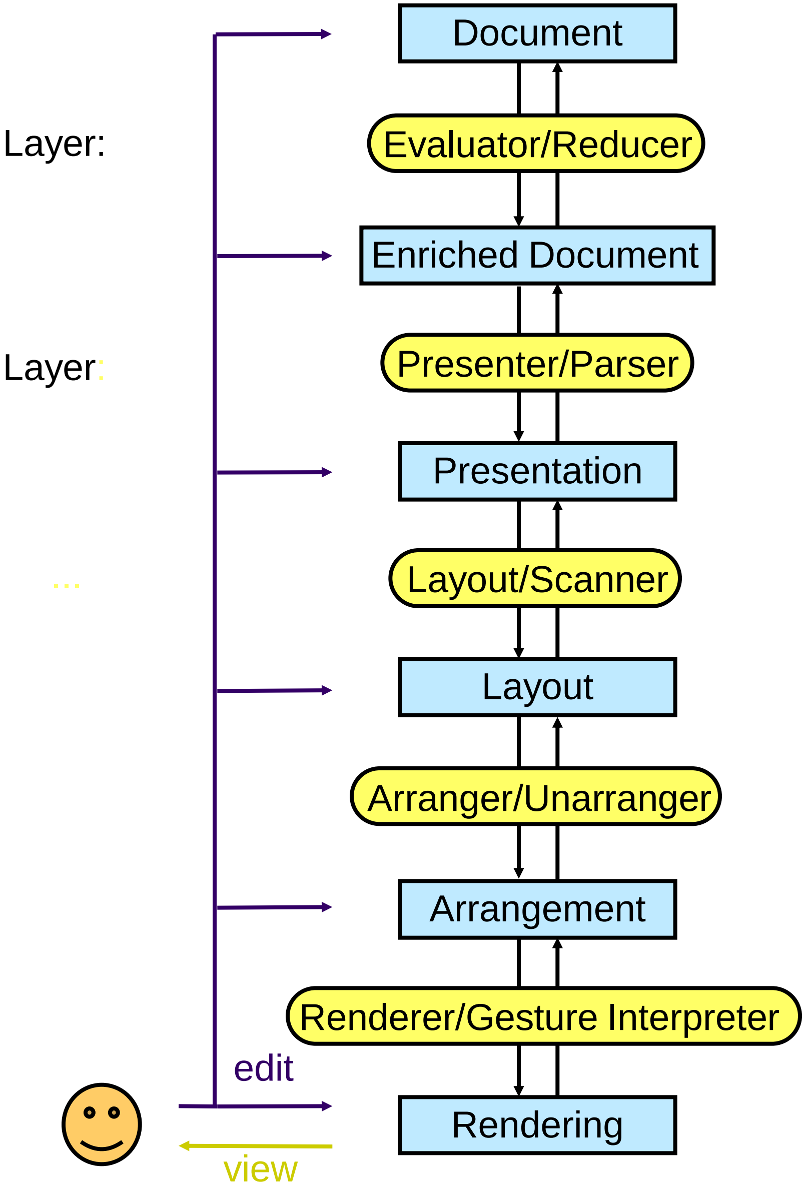
Arrangement — Presentation with exact positions & sizes

Rendering — Image

edit

view

Architecture



Layer:

Layer:

...

Document

Internal document tree

Evaluator/Reducer

Enriched Document

Document + derived values

Presenter/Parser

Presentation

Logical presentation: rows, columns,
tokens, etc.

Layout/Scanner

Layout

Presentation + explicit whitespace

Arranger/Unarranger

Arrangement

Presentation with exact positions & sizes

Renderer/Gesture Interpreter

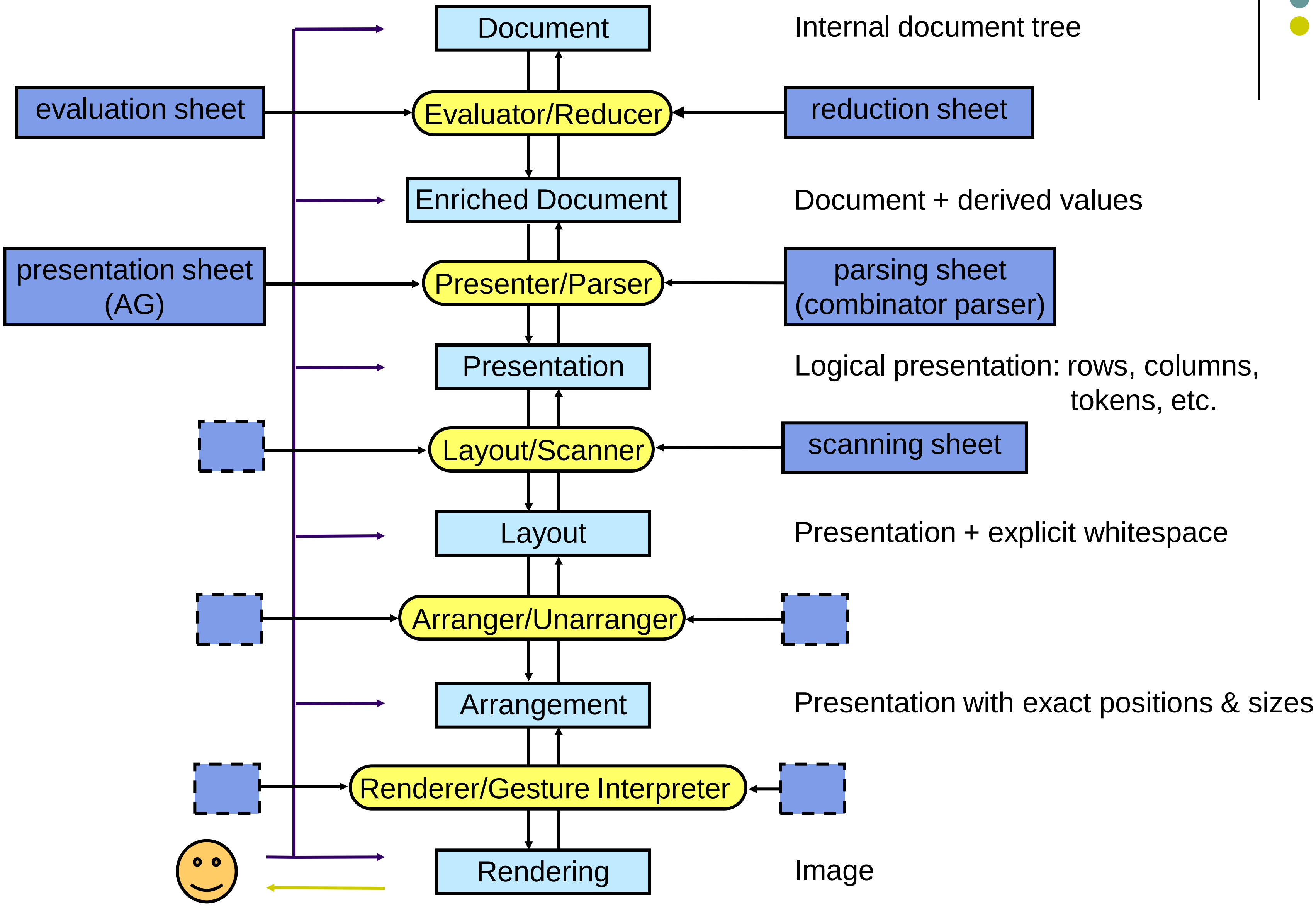
Rendering

Image

edit

view

Architecture



Document

Internal document tree

evaluation sheet

Evaluator/Reducer

reduction sheet

Enriched Document

Document + derived values

presentation sheet (AG)

Presenter/Parser

parsing sheet (combinator parser)

Presentation

Logical presentation: rows, columns, tokens, etc.

Layout/Scanner

scanning sheet

Layout

Presentation + explicit whitespace

Arranger/Unarranger

Arrangement

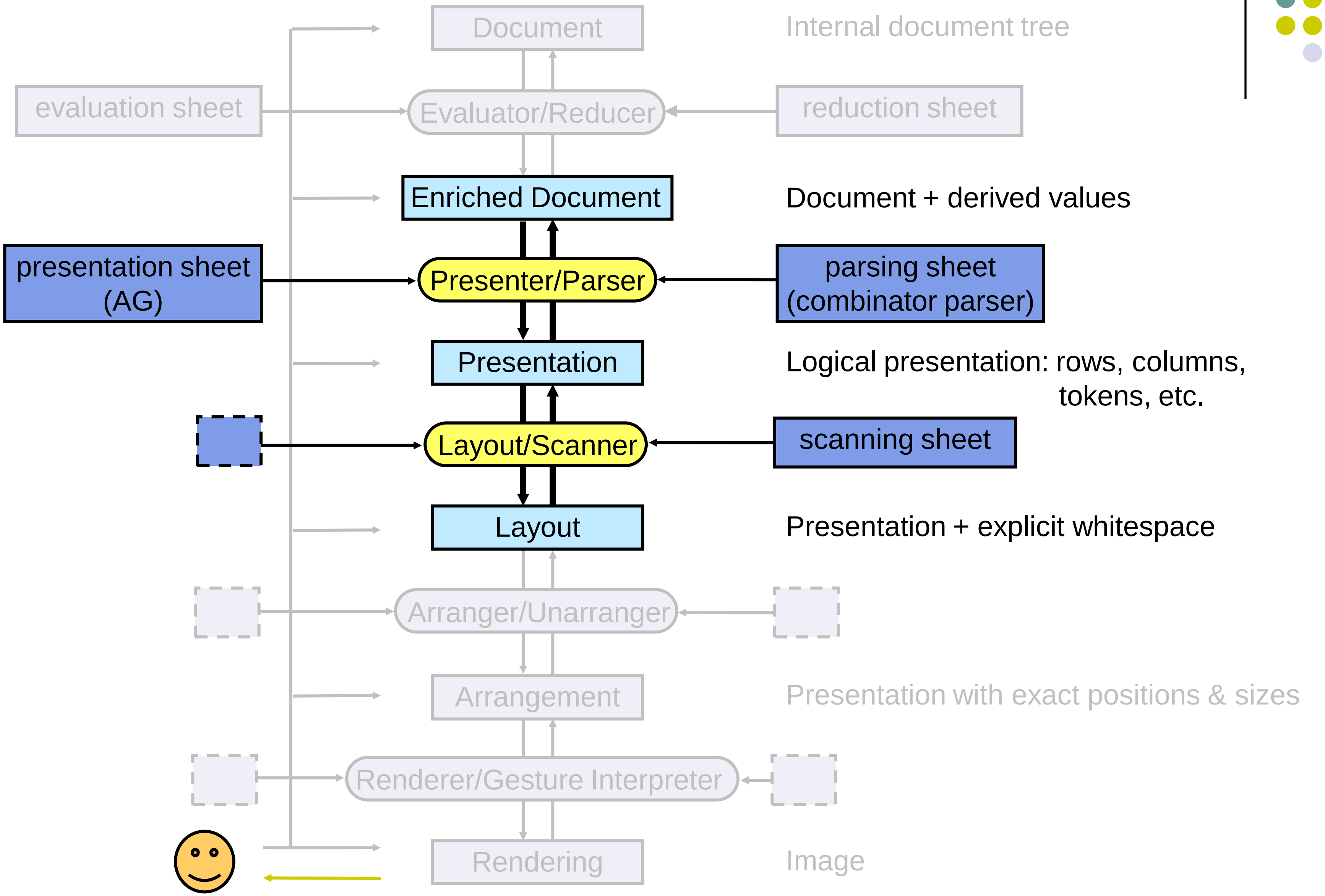
Presentation with exact positions & sizes

Renderer/Gesture Interpreter

Rendering

Image

This talk


Document
Internal document tree
evaluation sheet
Evaluator/Reducer
reduction sheet
Enriched Document
Document + derived values
presentation sheet
(AG)
Presenter/Parser
parsing sheet
(combinator parser)
Presentation
Logical presentation: rows, columns,
tokens, etc.
Layout/Scanner
scanning sheet
Layout
Presentation + explicit whitespace
Arranger/Unarranger
Arrangement
Presentation with exact positions & sizes
Renderer/Gesture Interpreter
Rendering
Image

Demo

- Helium editor
  - Functional language similar to Haskell
  - Graphical presentations
  - In-place parse and type errors
  - Derived values in source
  - 1200 lines of code
- Bayesian network documentation editor
  - Documentation for Bayesian Networks
  - Editable graphs with multiple views
  - Word-processor functionality
  - Derived tables
  - 800 lines of code

The problem

- How to parse this mix of textual and graphical structures?

$$x = \frac{1}{3^2+5} + 1;$$

Document

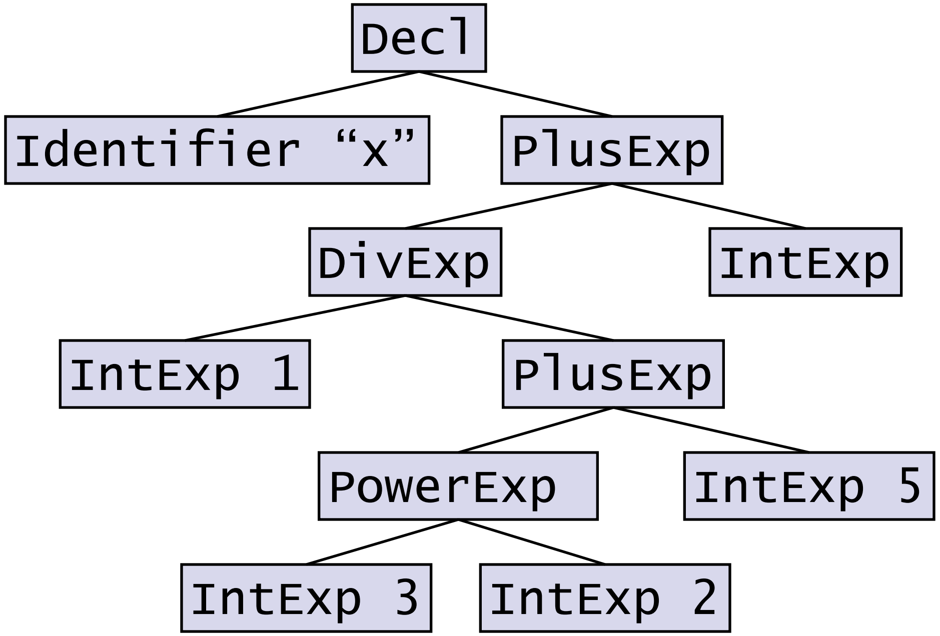
```
data Decl = Decl ident:Identifier exp:Exp
data Identifier = Ident str:String
data Exp = PlusExp  exp1:Exp exp2:Exp
         | DivExp   exp1:Exp exp2:Exp
         | PowerExp exp1:Exp exp2:Exp
         | IntExp   val:Int
```

$$x = \frac{1}{3^2+5} + 1;$$

Document

```
data Decl = Decl ident:Identifier exp:Exp
data Identifier = Ident str:String
data Exp = PlusExp  exp1:Exp exp2:Exp
         | DivExp   exp1:Exp exp2:Exp
         | PowerExp exp1:Exp exp2:Exp
         | IntExp   val:Int
```

$$x = \frac{1}{3^2+5} + 1;$$



Decl

Identifier “x”

PlusExp

DivExp

IntExp

IntExp 1

PlusExp

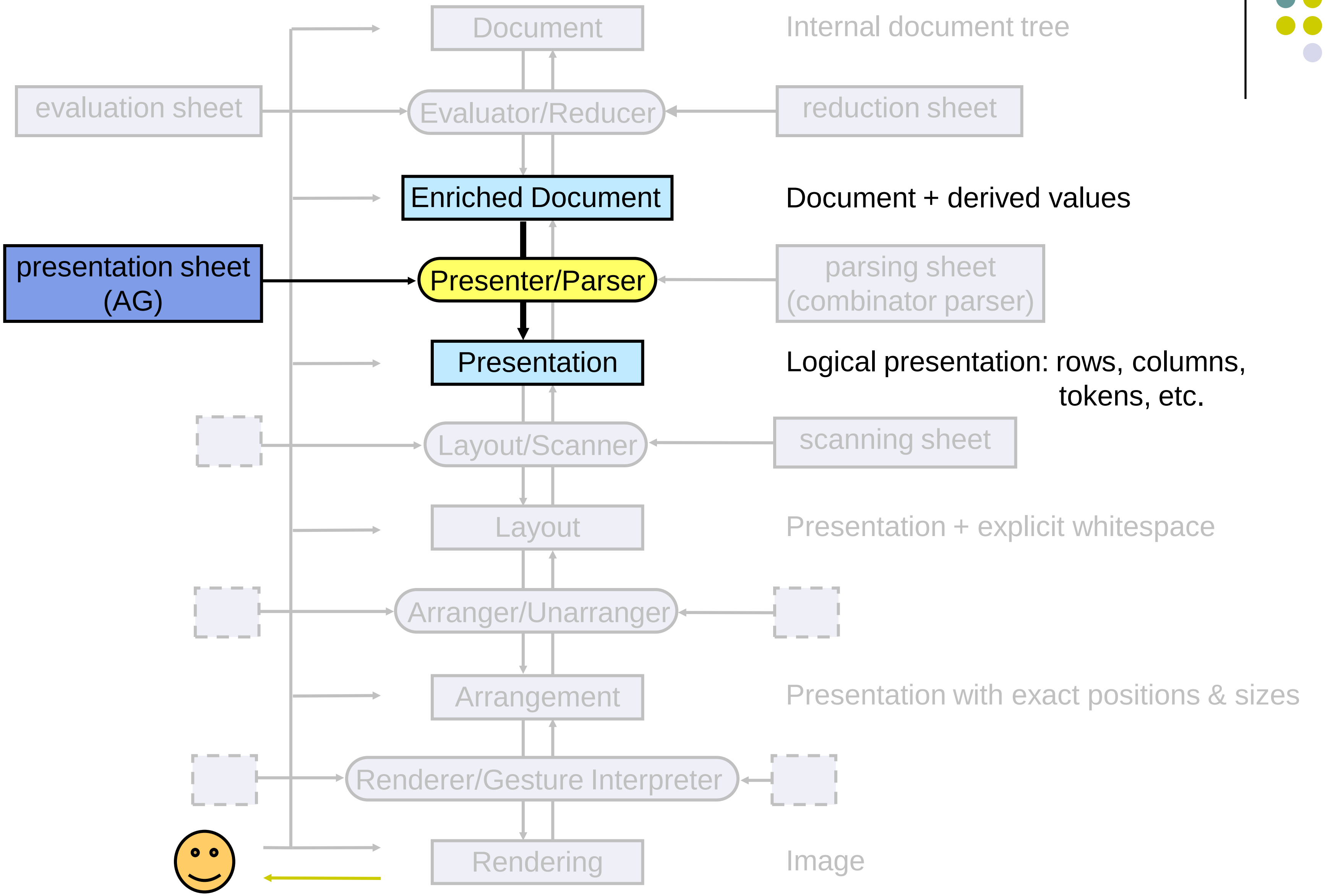
PowerExp

IntExp 5

IntExp 3

IntExp 2

Presentation



Presentation level: Xprez

- Presentation language of Proxima
- Box language: rows and columns
- Strings, polygons, circles, etc.
- Tokens, converted to strings by Layout layer
- Implemented in Haskell

Xprez

Example:

`frac (text “1”) (text “1+2”)` → $\frac{1}{1+2}$

```
frac :: Xprez -> Xprez -> Xprez
frac e1 e2 =
  let numerator   = hAlignCenter (pad (shrink e1) )
      denominator = hAlignCenter (pad (shrink e2) )
  in colR 2 [ numerator, vSpace 2, hLine
            , vSpace 2, denominator ] ‘withHStretch‘ False

pad xp = row [ hSpace 2, xp, hSpace 2 ]

shrink e = e ‘withFontSize_‘
           (\fs -> (70 ‘percent‘ fs) ‘max‘ 10)
```

The presentation sheet

- Presentation rule for each type constructor
  - sequence: parser + list of tokens
  - structural: any presentation

```
SEM Decl
  | Decl loc.pres = sequence parseDecl
             [ @ident.pres, key @idP1 "="
             , @exp.pres, sym @idP2 ";" ]
SEM Exp
  | PlusExp loc.pres = sequence parseExp
                        [ @exp1.pres
                        , operator @idP1 "+"
                        , @exp2.pres ]
  | DivExp loc.pres = sequence parseExp
                        [ structuralToken @idP1 $
                            frac @exp1.pres @exp2.pres ]
```

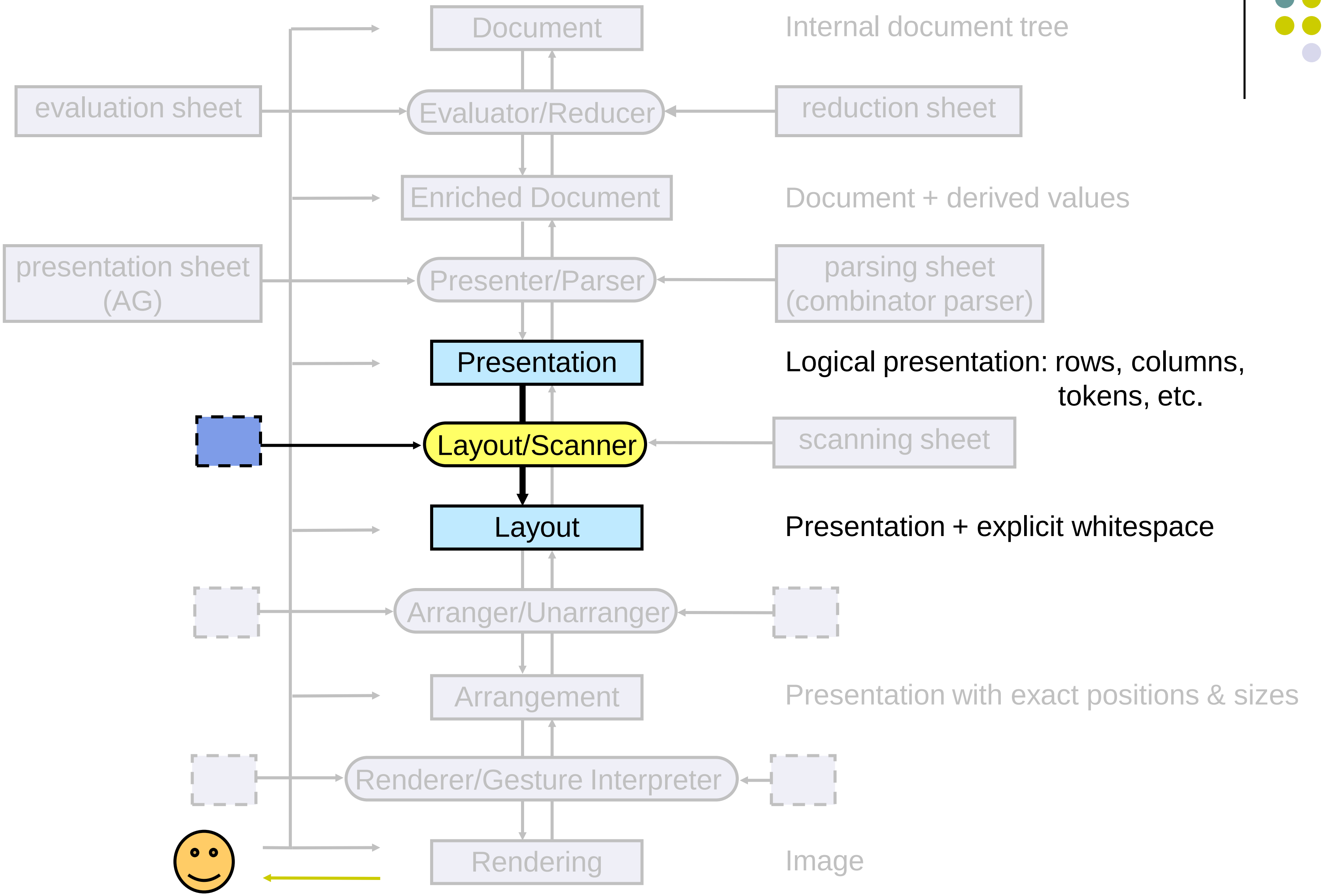
The presentation sheet

- Presentation rule for each type constructor
  - sequence: parser + list of tokens
  - structural: any presentation

key idp str = token idp str ‘withColor‘ blue
sym idp str = token idp str ‘withColor‘ orange
operator idp str = token idp str ‘withColor‘ green

```
SEM Decl
  | Decl loc.pres = sequence parseDecl
             [ @ident.pres, key @idP1 "="
             , @exp.pres, sym @idP2 ";" ]
SEM Exp
  | PlusExp loc.pres = sequence parseExp
                        [ @exp1.pres
                        , operator @idP1 "+"
                        , @exp2.pres ]
  | DivExp loc.pres = sequence parseExp
                        [ structuralToken @idP1 $
                            frac @exp1.pres @exp2.pres ]
```

Layout



Internal document tree

Document + derived values

Logical presentation: rows, columns, tokens, etc.

Presentation + explicit whitespace

Presentation with exact positions & sizes

Image

Layout: explicit whitespace

Presentation level:

```
sequence parseDecl
  [ token_0 “x”, token_1 “=” , token_2 “1”, token_3 “+”
  , token_4 “2”, token_5 “;”]

whitespace map: [ 0 → (0,1), 1 → (0,3), 2 → (1,4)
                , 3 → (0,1), 4 → (0,1), 5 → (1,0)
                ]
```

Rendering level:

```
x =    1
     + 2;
```

Layout: explicit whitespace

Presentation level:

```
sequence parseDecl
  [ token0 “x”, token1 “=” , token2 “1”, token3 “+”
  , token4 “2”, token5 “;”]

whitespace map: [ 0 → (0,1), 1 → (0,3), 2 → (1,4)
                , 3 → (0,1), 4 → (0,1), 5 → (1,0)
                ]
```

(breaks, spaces)

focus is stored here

Rendering level:

```
x =    1
     + 2;
```

Layout: explicit whitespace

Presentation level:

```
sequence parseDecl
  [ token_0 “x”, token_1 “=” , token_2 “1”, token_3 “+”
  , token_4 “2”, token_5 “;”]

whitespace map: [ 0 → (0,1), 1 → (0,3), 2 → (1,4)
                , 3 → (0,1), 4 → (0,1), 5 → (1,0)
                ]
```

(breaks, spaces)

focus is stored here

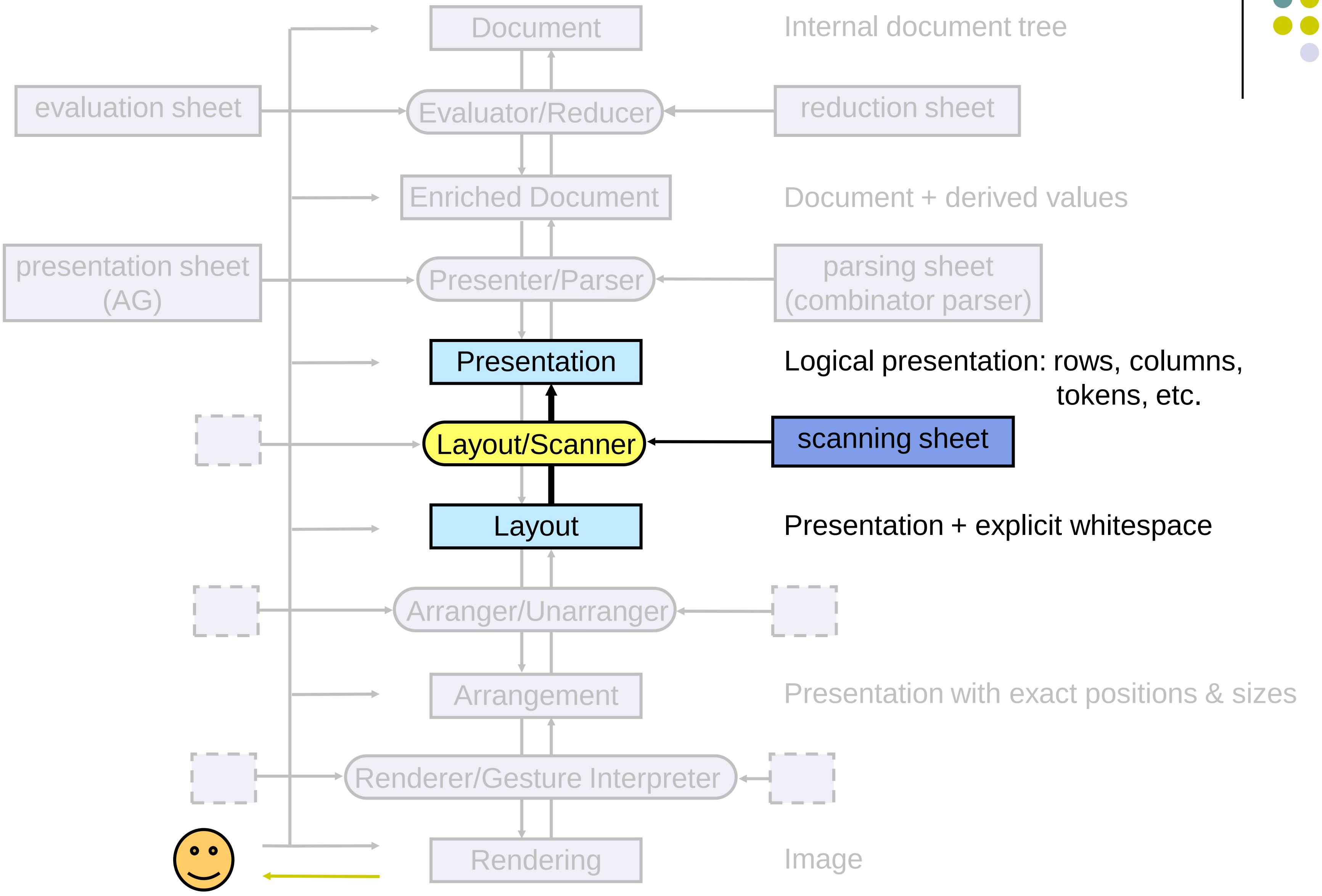
Layout level:

```
seq $ col [ row [ “x”, “ ”, “=”, “   ”, “1” ]
          , row [ “    ”, “+”, “ ”, “2”, “;”]
          , row [ “” ]
          ]
```

Rendering level:

```
x =   1
    + 2;
```

Scanning


Document
Internal document tree
evaluation sheet
Evaluator/Reducer
reduction sheet
Enriched Document
Document + derived values
presentation sheet (AG)
Presenter/Parser
parsing sheet (combinator parser)
Presentation
Logical presentation: rows, columns, tokens, etc.
Layout/Scanner
scanning sheet
Layout
Presentation + explicit whitespace
Arranger/Unarranger
Arrangement
Presentation with exact positions & sizes
Renderer/Gesture Interpreter
Rendering
Image

Sequential vs Structural

rendering:

$$x = \frac{1}{3^2+5} + 1;$$

layout level:

```
seq (row [ “x”, “ ”, “=“ , “ ”
         , struc
            (col [ seq (row [ “1”])
                 , hLine
                 , seq ( row [ struc (..)
                             , “+”, “5”])
                 ])
         , “ “, “+”, “1”, “;”])
```

sequential

structural

Sequential vs Structural

rendering:

$$x = \frac{1}{3^2+5} + 1;$$

layout level:

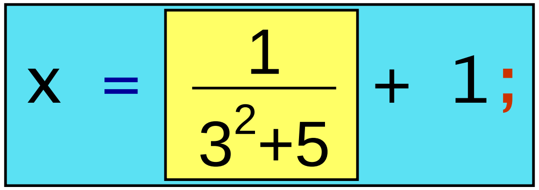
```
seq (row [ “x”, “ ”, “=“ , “ ”
         , struc
            (col [ seq (row [ “1”])
                 , hLine
                 , seq ( row [ struc (..)
                             , “+”, “5”])
                 ])
         , “ “, “+”, “1”, “;”])
```

sequential

structural

Sequential vs Structural

rendering:



$$x = \frac{1}{3^2+5} + 1;$$

layout level:

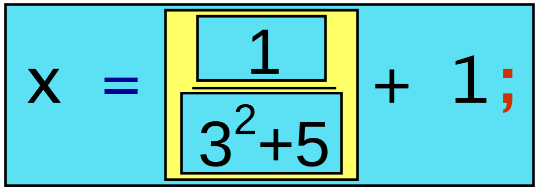
```
seq (row [ “x”, “ ”, “=“ , “ ”
         , struc
            (col [ seq (row [ “1”])
                 , hLine
                 , seq ( row [ struc (..)
                             , “+”, “5”])
                 ])
         , “ “, “+”, “1”, “;”])
```

🟦 sequential

🟨 structural

Sequential vs Structural

rendering:



layout level:

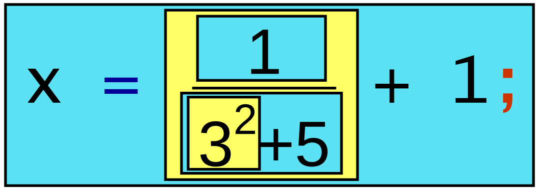
```
seq (row [ “x”, “ ”, “=“ , “ ”
         , struc
            (col [ seq (row [ “1”])
                 , hLine
                 , seq ( row [ struc (..)
                             , “+”, “5”])
                 ])
         , “ “, “+”, “1”, “;”])
```

sequential

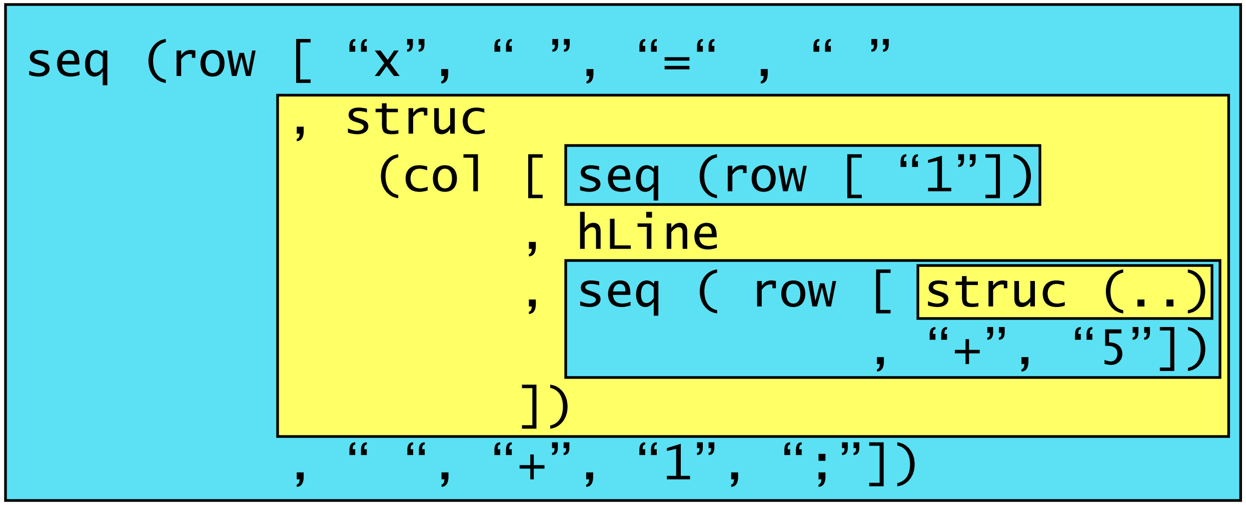
structural

Sequential vs Structural

rendering:



layout level:



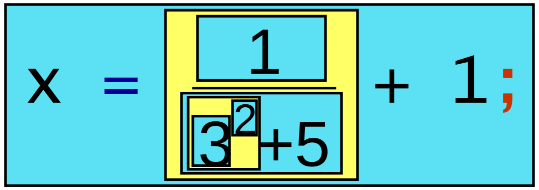
```
seq (row [ “x”, “ ”, “=“ , “ ”
         , struc
            (col [ seq (row [ “1”])
                 , hLine
                 , seq ( row [ struc (..)
                             , “+”, “5”])
                 ])
         , “ “, “+”, “1”, “;”])
```

🟦 sequential

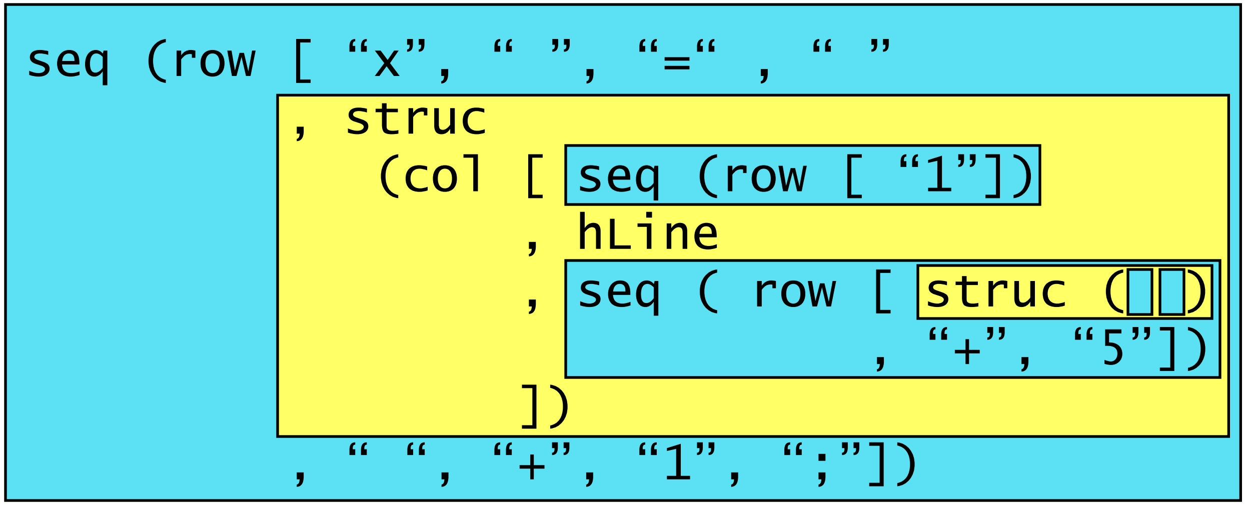
🟨 structural

Sequential vs Structural

rendering:



layout level:

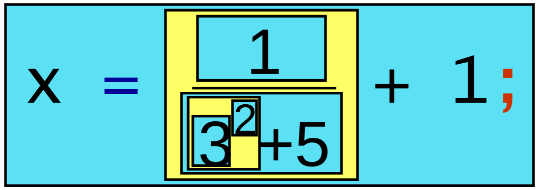


```
seq (row [ “x”, “ ”, “=“ , “ ”
         , struc
             (col [ seq (row [ “1”])
                  , hLine
                  , seq ( row [ struc (□□)
                                , “+”, “5”])
                  ])
         , “ “, “+”, “1”, “;”])
```

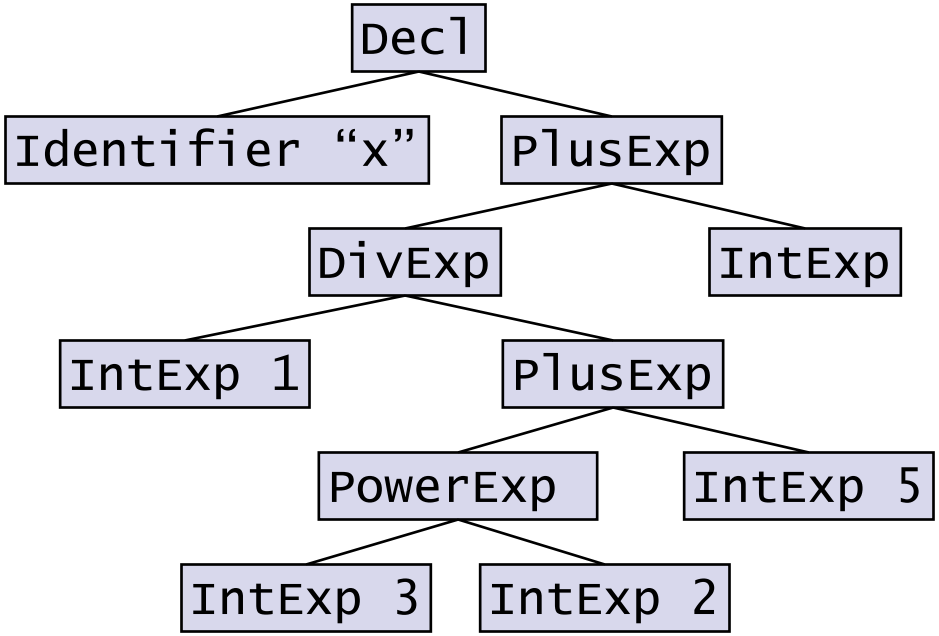
🟦 sequential

🟨 structural

Location in document tree

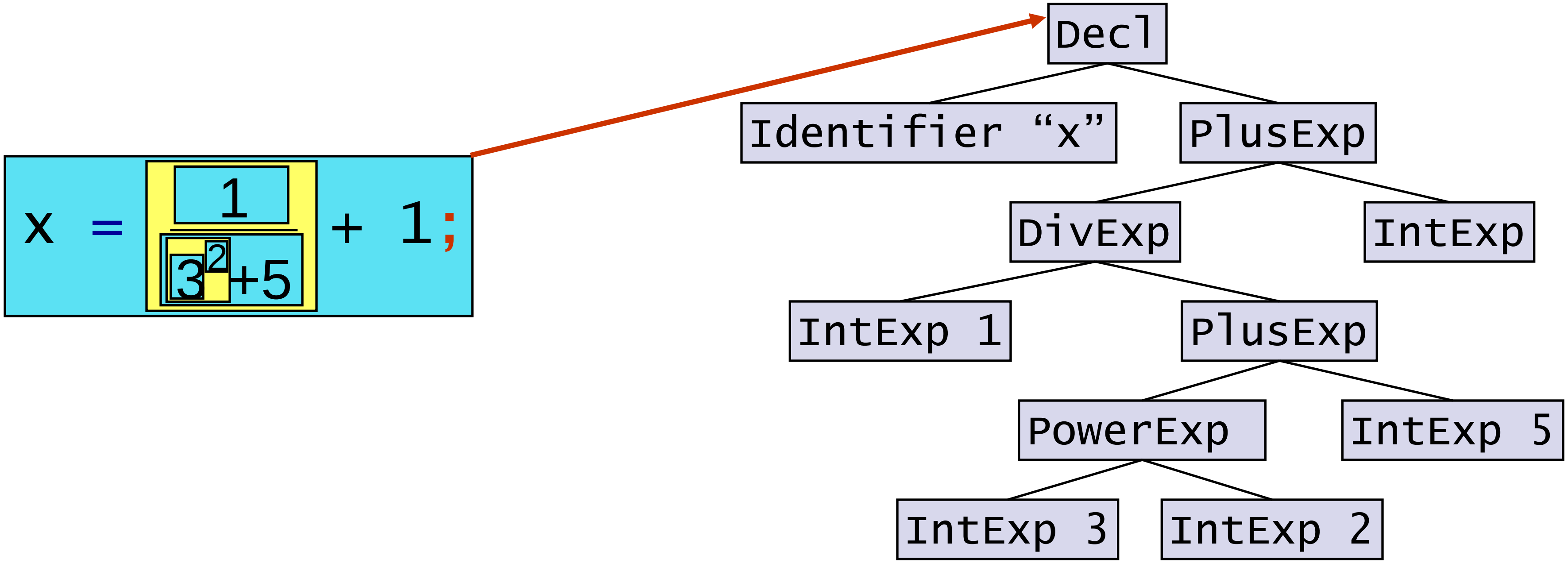


$x = \frac{1}{3^2+5} + 1;$

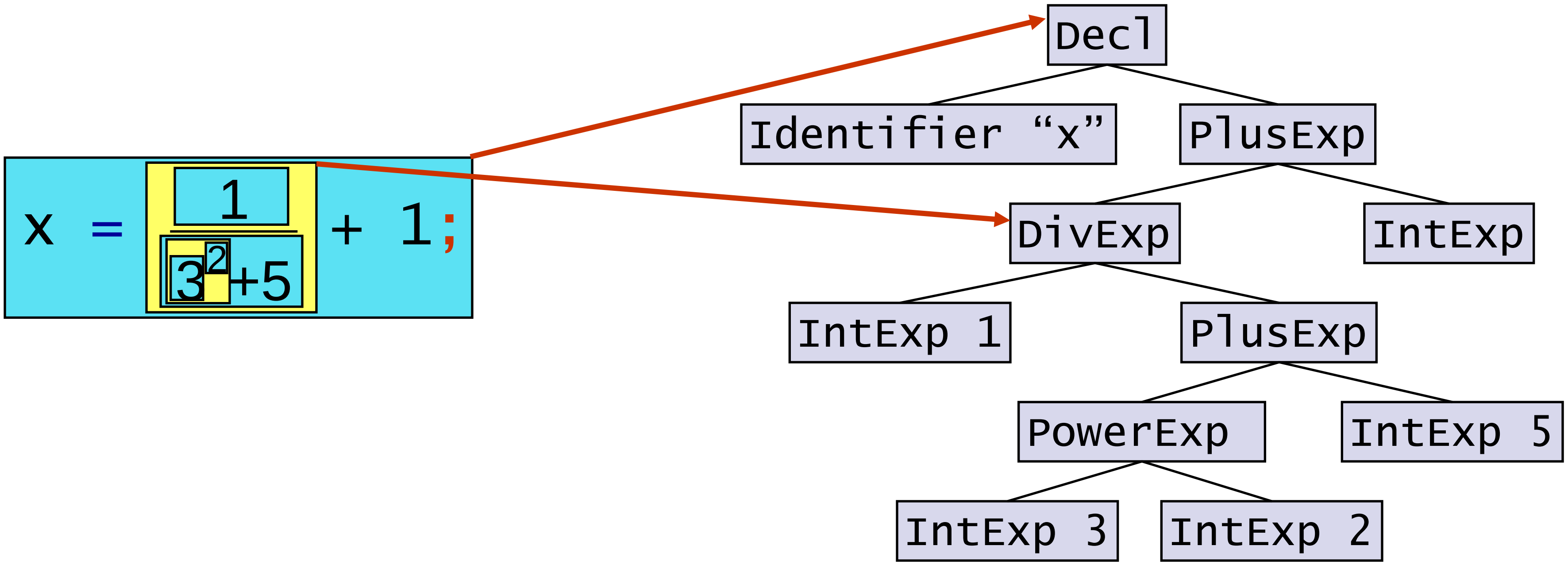


Decl
Identifier "x"
PlusExp
DivExp
IntExp
IntExp 1
PlusExp
PowerExp
IntExp 5
IntExp 3
IntExp 2

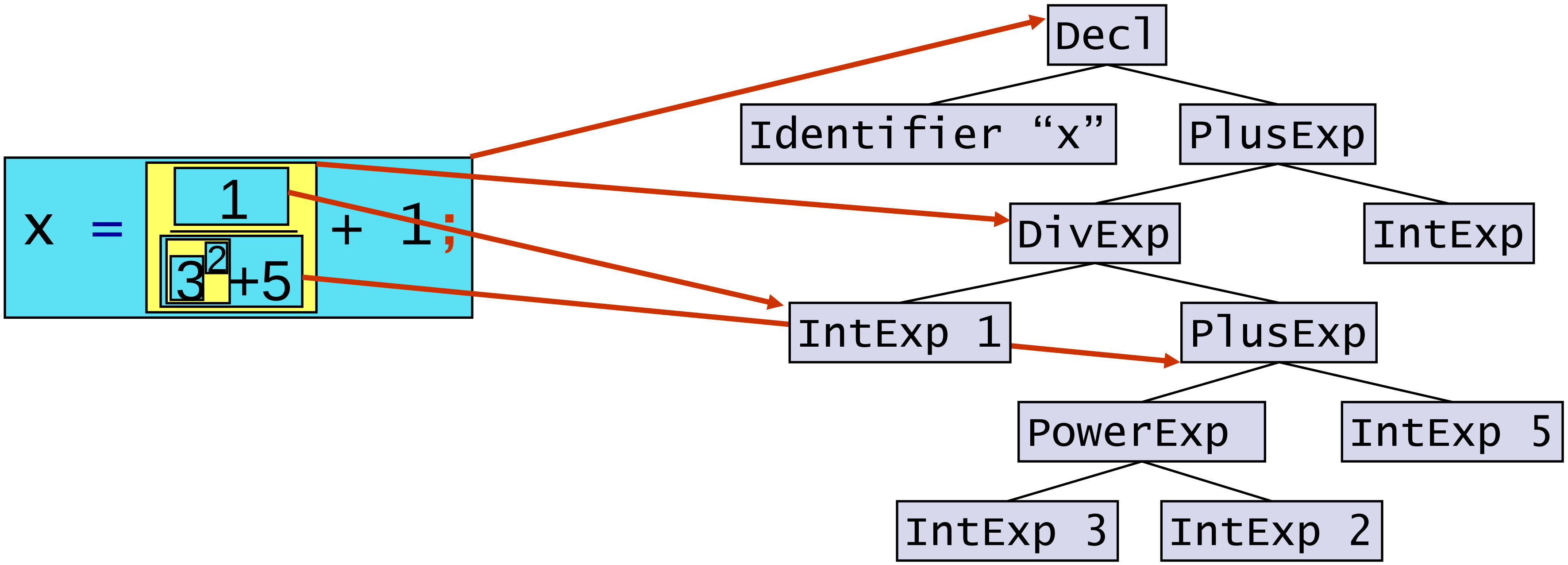
Location in document tree



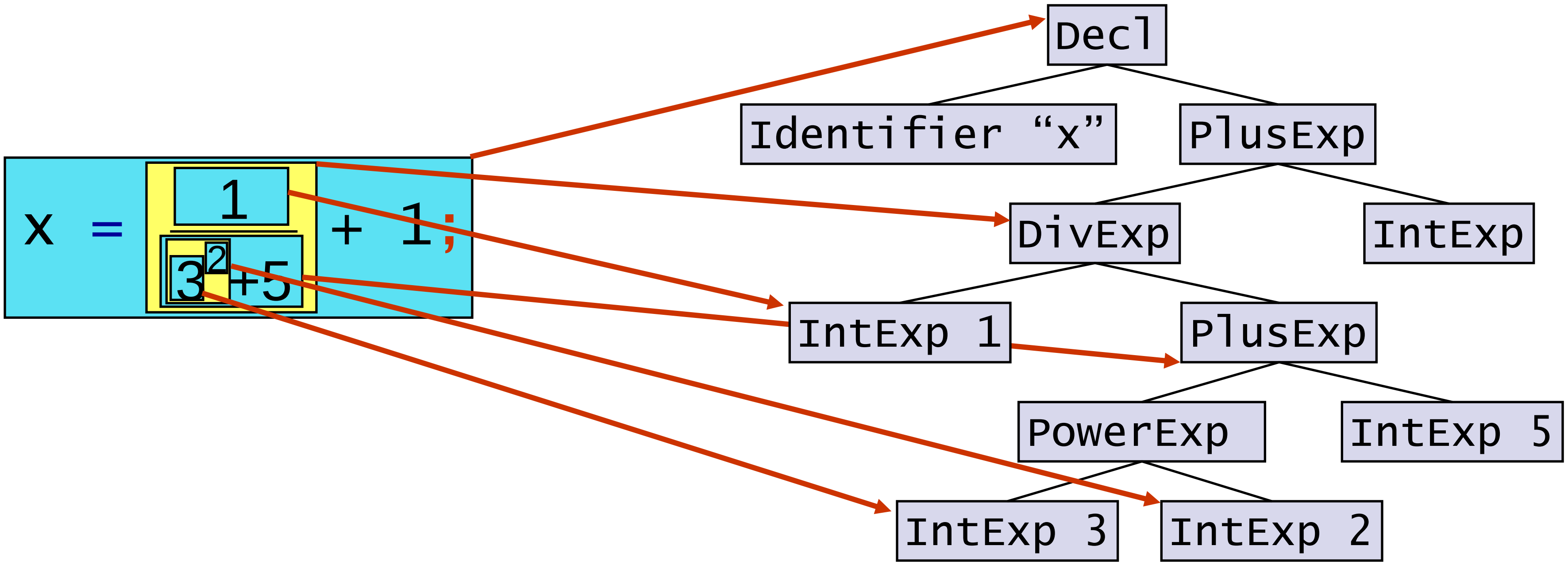
Location in document tree



Location in document tree



Location in document tree



Scanning

- Input: rows/columns, structural/sequential

```
data Token =
     StructuralTk IDP Location        [Token]
   | SequentialTk IDP Location Parser [Token]
   | UserTk       IDP String
   | ErrorTk      IDP String
```

- Result:
  - Tree of tokens, root: `StructuralTk [..]`
  - Whitespace map: IDP → whitespace (& focus)

Scanning

- Input: rows/columns, structural/sequential

unique id

```
data Token =
     StructuralTk IDP Location        [Token]
   | SequentialTk IDP Location Parser [Token]
   | UserTk       IDP String
   | ErrorTk      IDP String
```

path to originating document node: →

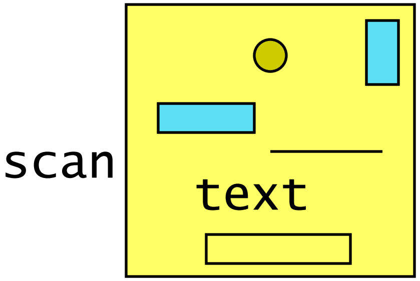
- Result:
  - Tree of tokens, root: `StructuralTk [..]`
  - Whitespace map: IDP → whitespace (& focus)

Scanning

Structural: - recursively scan children

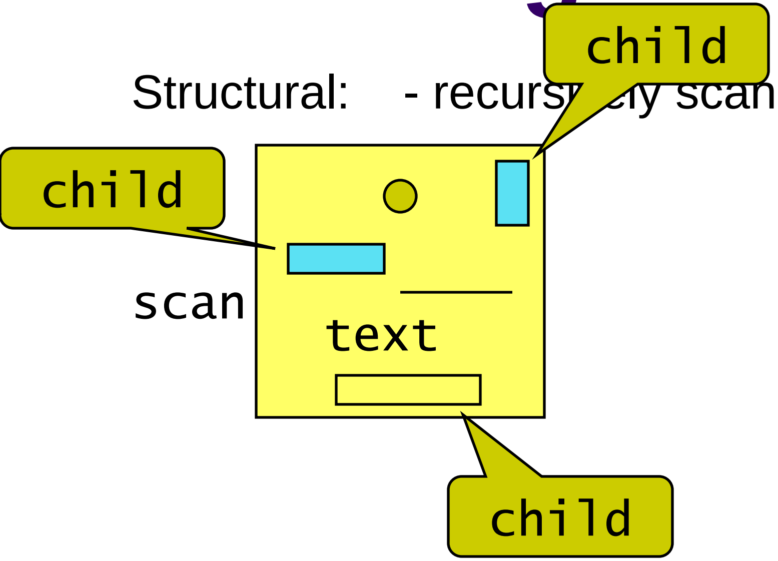
Scanning

Structural: - recursively scan children



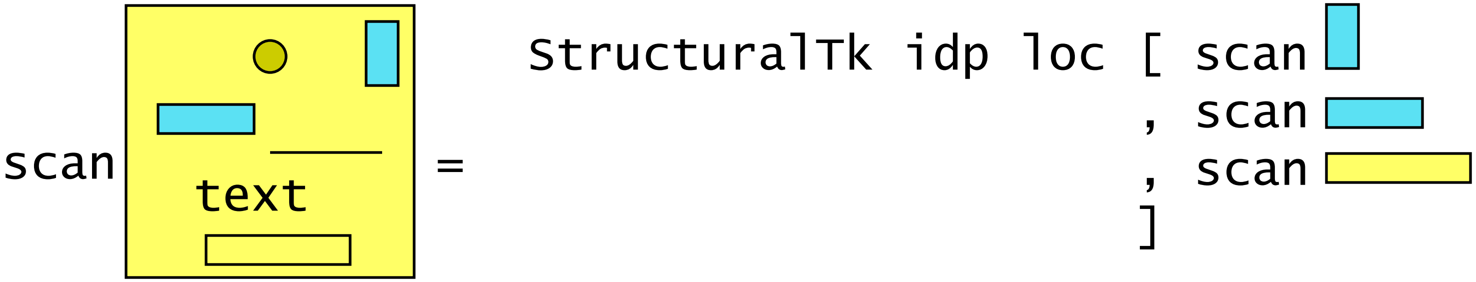
Scanning

Structural: - recursively scan children



Scanning

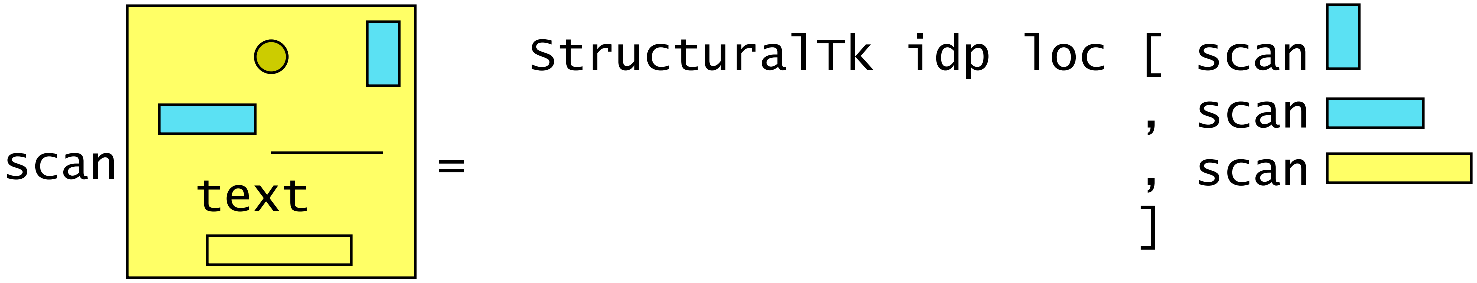
Structural: - recursively scan children



```
scan      =   StructuralTk idp loc [ scan
                                   , scan
                                   , scan
                                   ]
```

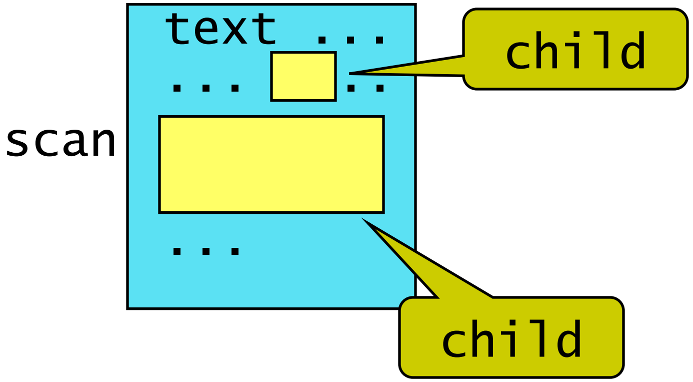
Scanning

Structural: - recursively scan children



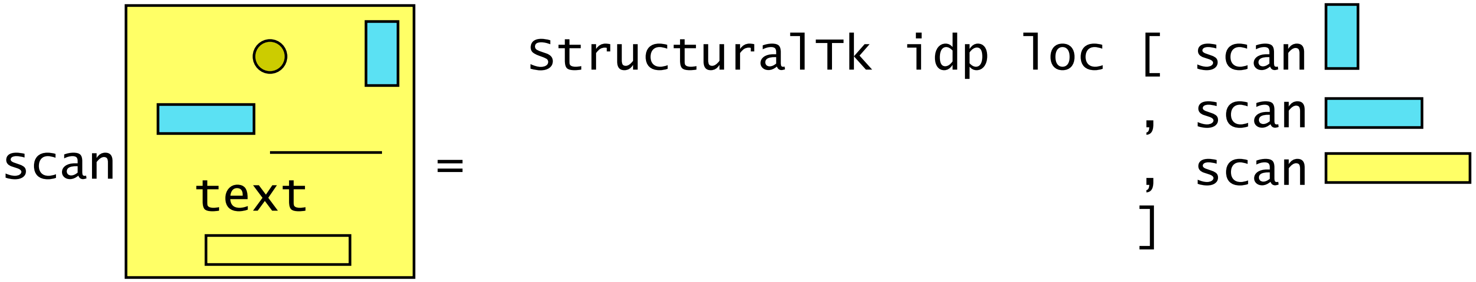
```
scan text =  StructuralTk idp loc [ scan
                                  , scan
                                  , scan
                                  ]
```

Sequential: - create tokens based on reg. exp. in scanning sheet
- recursively scan structural children
- store whitespace & focus in whitespace map



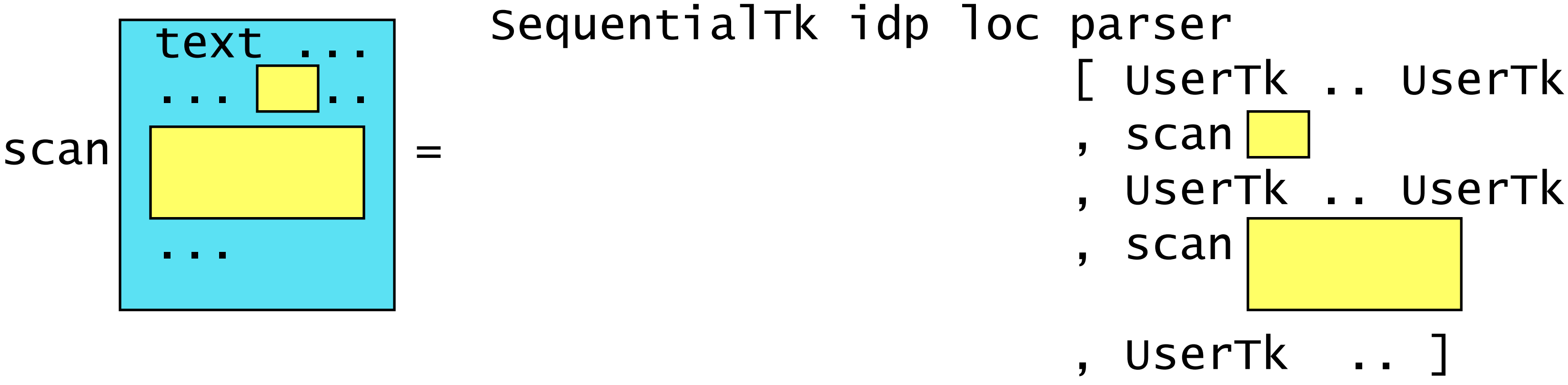
Scanning

Structural: - recursively scan children



```
scan [text] =   StructuralTk idp loc [ scan
                                     , scan
                                     , scan
                                     ]
```

Sequential:
- create tokens based on reg. exp. in scanning sheet
- recursively scan structural children
- store whitespace & focus in whitespace map

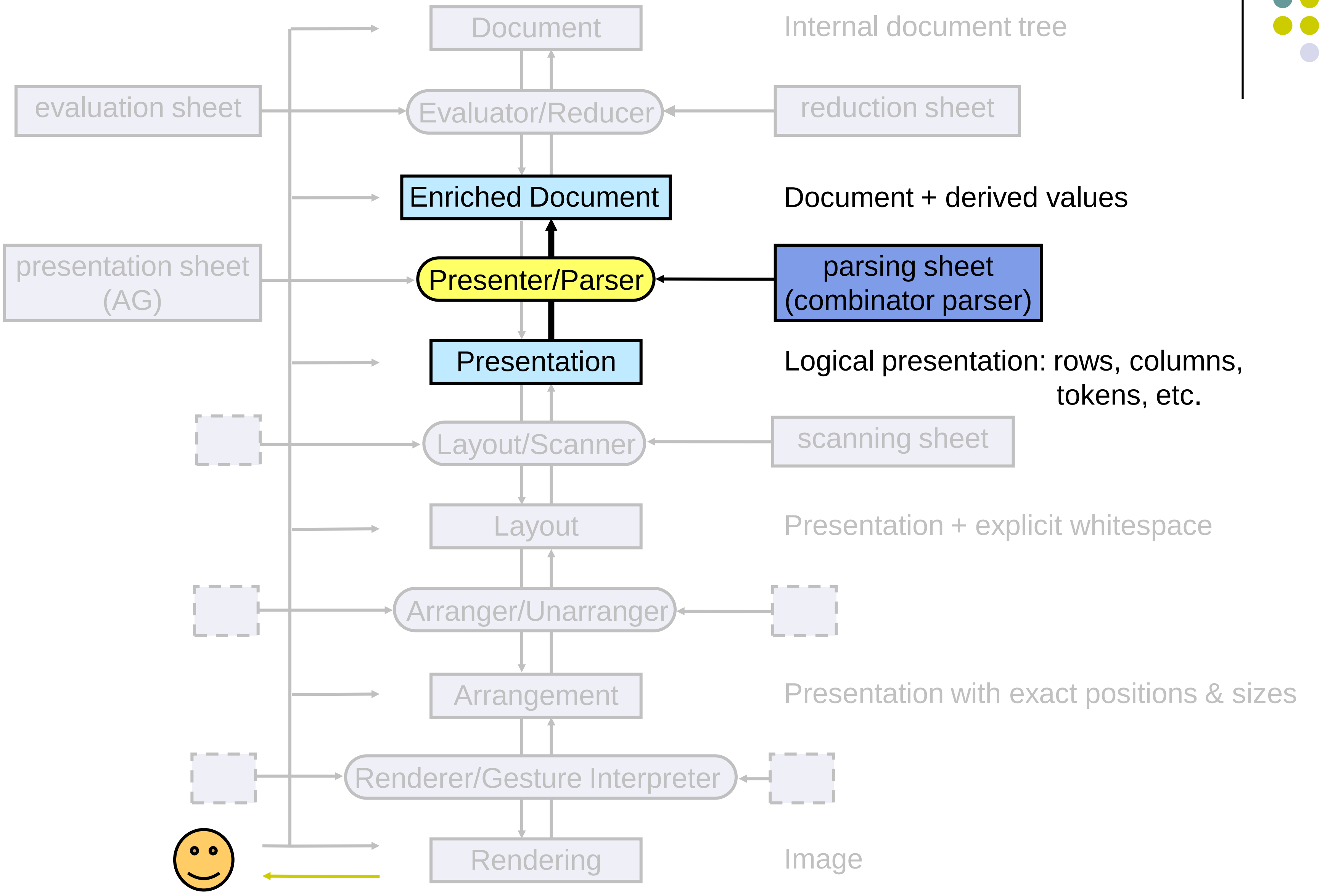


```
scan [text ...
      ...  ..
      ...]     =   SequentialTk idp loc parser
                                [ UserTk .. UserTk
                                , scan
                                , UserTk .. UserTk
                                , scan

                                , UserTk  .. ]
```

+ whitespace map

Parsing


Document
Internal document tree
evaluation sheet
Evaluator/Reducer
reduction sheet
Enriched Document
Document + derived values
presentation sheet (AG)
Presenter/Parser
parsing sheet (combinator parser)
Presentation
Logical presentation: rows, columns, tokens, etc.
Layout/Scanner
scanning sheet
Layout
Presentation + explicit whitespace
Arranger/Unarranger
Arrangement
Presentation with exact positions & sizes
Renderer/Gesture Interpreter
Rendering
Image

Parsing: Structural

StructuralTk _ (Node c_0 .. c_n) [$token_0$ .. $token_n$]

- Recursively parse $token_0$ .. $token_n$
- Yields values for children, but
  - not all children need to be in presentation
- Solution:
  - for absent child, use value from (Node c_0 .. c_n)

- Next version of Proxima: change management
  - only parse changed child, otherwise use c_i
  - child may appear more than once
  - take edited one (only one may be edited)

Parsing: Sequential

`SequentialTk _ _ parser  [Token`$_0$` .. Token`$_n$`]`

- use parser to parse list of tokens
- parser is a combinator parser
  - special primitive for structural presentations

```
pDecl :: Parser Decl
pDecl = Decl
        <$> pIdent <* pToken "="
        <*> pExp   <* pToken ";"

pExp :: Parser Exp
pExp = pStructural Node_Div
   <|> pStructural Node_Power
   <|> PlusExp <$> pExp <* pToken "+" <*> pExp
```

Conclusions

- Graphical presentations are benificial
- Easy to write parser
- Fast enough
  - Change management: lot faster
  - Incremental parsing possible

http://www.cs.uu.nl/wiki/Proxima

Questions?

http://www.cs.uu.nl/wiki/Proxima