Space on Paper :

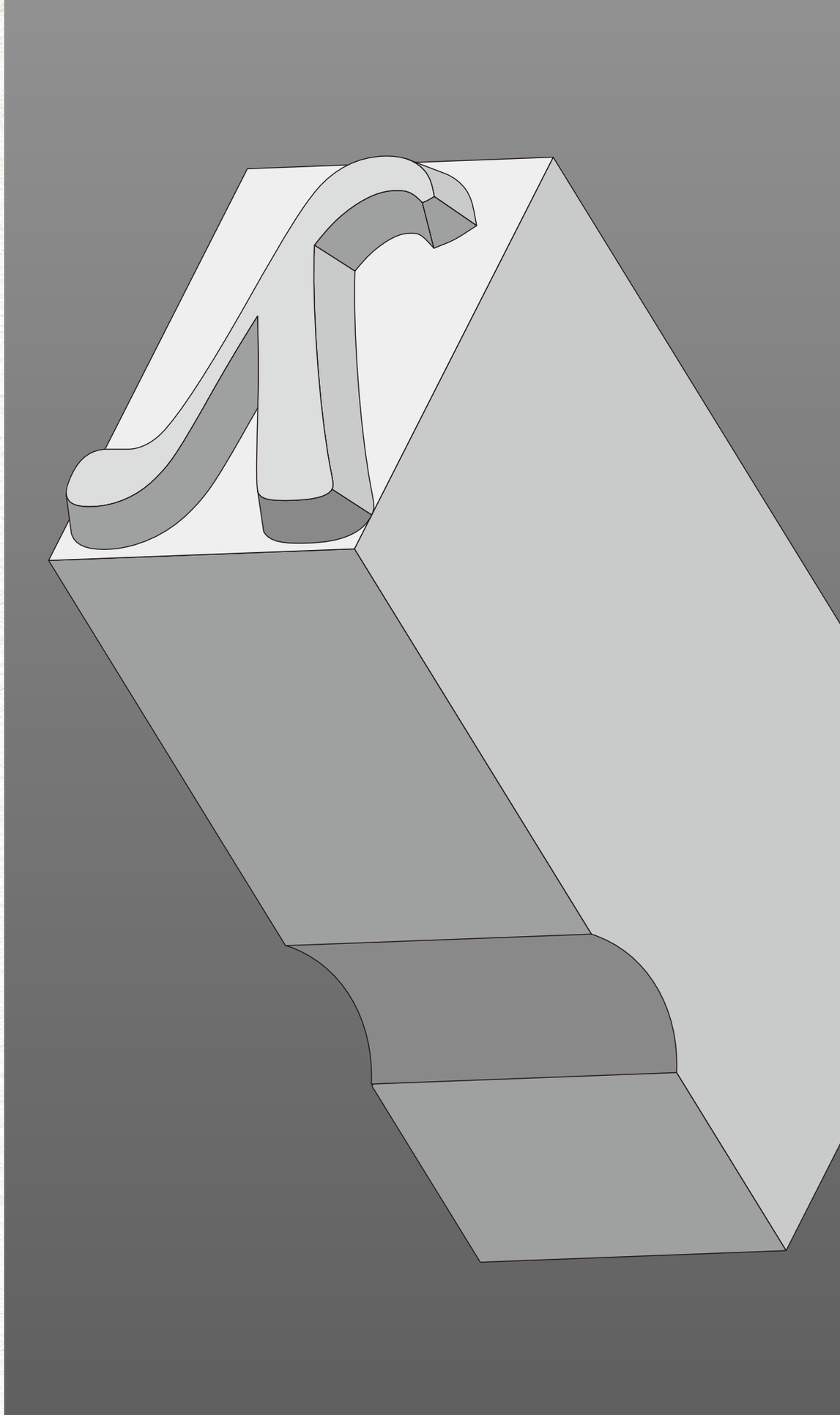
文字組版アルゴリズムの基礎

2024 年 9 月 21 日

夏のプログラミング・シンポジウム 2024

Takashi Suwa（gfngfn）

京都大学大学院・国立情報学研究所



提示テーマの例

以下の提示テーマに限らず，スペースに関するものであればどんな分野

- 宇宙
  - 宇宙の重力波の解析
  - 人工衛星の制御プログラム
  - 地球観測衛星の画像処理
- 空間
  - VR/AR，デジタルツイン
  - こだわりの研究スペース
  - 都市計画，建築
- 字間
  - 組版の文字詰め計算
  - 自然言語処理の形態素解析
  - 検索のための分かち書き処理
- 空白
  - 図形詰込みアルゴリズム
  - 最密充填問題

提示テーマの例

以下の提示テーマに限らず，スペースに関するものであればどんな分野

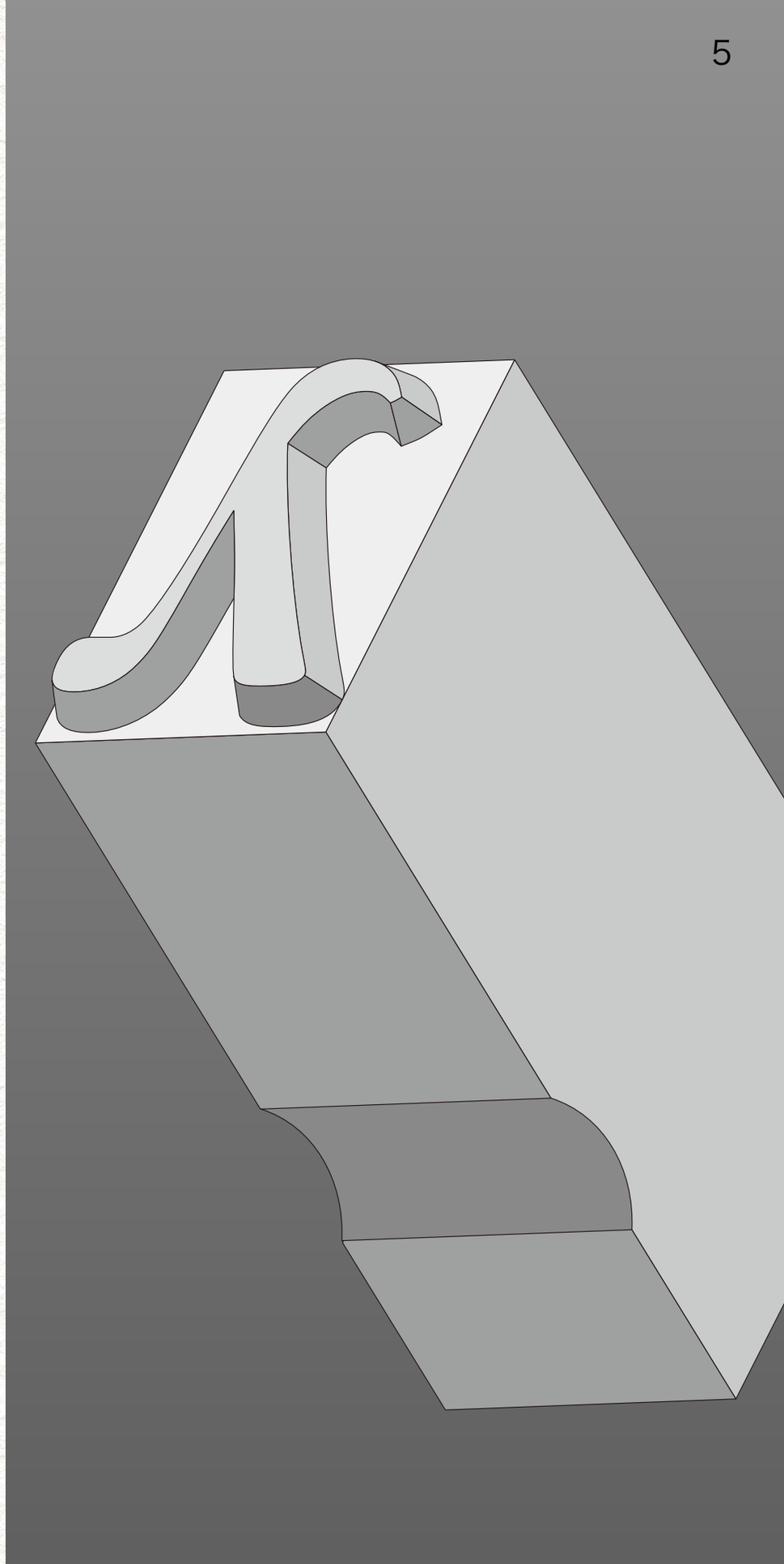
- 宇宙
  - 宇宙の重力波の解析
  - 人工衛星の制御プログラム
  - 地球観測衛星の画像処理
- 空間
  - VR/AR，デジタルツイン
  - こだわりの研究スペース
  - 都市計画，建築
- 字間
  - 組版の文字詰め計算
  - 自然言語処理の形態素解析
  - 検索のための分かち書き処理
- 空白
  - 図形詰込みアルゴリズム
  - 最密充填問題

本発表の内容： 組版処理アルゴリズム

- 最初に少しだけ拙作の組版処理システム **SATySFi** を紹介
- 組版処理上の種々の定式化を概説
  - 図形としての文字（＝**グリフ**，*glyph*）の取扱い
  - **行分割処理**（＝行を切り分けて段落にする処理）が
    どのように定式化されているか
  - **OpenType** フォントにどんなデータが格納されているか
  - **Unicode Database** や **JLreq** を利用した原稿処理
  - etc.
- **とりわけアルゴリズム的にも面白い行分割処理に焦点を当てる**

目次

- **最初に寄り道： SATγSFI の紹介**
- 文字の基本的な取扱い： ボックス列
- 行分割処理の定式化
- 原稿からボックス列への変換方法の概要
- 難しい拡張（SATγSFI の Future Work）
- まとめ



SATySFi（サティスファイ）とは

- 発表者が 2017 年度 IPA 未踏事業の 1 プロジェクトとして開発し，
  現在も牛歩で開発を続けている**組版処理システム**
  - 来月くらいには初めての非互換なリリースがようやくできる予定です
- **静的型つき**のいわゆる**函数型**のプログラミング言語で
  パッケージや文書本体が書ける
  - 一般的なプログラムと同じ要領で読み書きできるので
    カスタマイズ性が高い
    - cf. TEX/LATEX： 熟練者でも他人の書いた実装は読んで改変するのが難しい
  - 誤った記述をしたときも，型検査によって**迅速**（典型的には 0.1 秒ほど）
    かつ原因の解りやすいエラー報告を出してくれる傾向にある
    - cf. TEX/LATEX： 些細なミスでも十数秒待ってから解りにくいエラーが出たりする

動作デモ（今回は正常系だけ）

paper.saty

paper.saty | test51-pp.saty | pp-amidakuji-1.satyh | test50-pp.saty

```
@import: settings
@import: introduction
@import: example-of-error-report
@import: example-of-line-breaking
@import: example-of-math
@import: formal-definition
@import: bib

document (|

  title = {静的型つき組版処理システム\SATySFi;};

  authors = [
    (|
      name  = {諏訪 敬之};
      email = `bd.gfngfn@gmail.com`;
    |);
  ];

  abstract = {
    本稿では，2017年度未踏事業の1プロジェクトとして著者が開発し，現在も発展を続けている
    新しい組版処理システム\SATySFi;（サティスファイ）について紹介する．
    このシステムは（\TeX;/\LaTeX;と同様に）或る種のマークアップ言語とその処理系として実装されており，
    ユーザやパッケージ開発者が独自にコマンドを定義して使用することも可能である．
    従来システムとの顕著な違いのひとつは，OCamlによく似たいわゆる函数型
```

U:--- paper.saty Top (1,0) Git-master (Satysfi +2 FlyC- Undo-Tree Helm)

Async Shell Command | *shell*

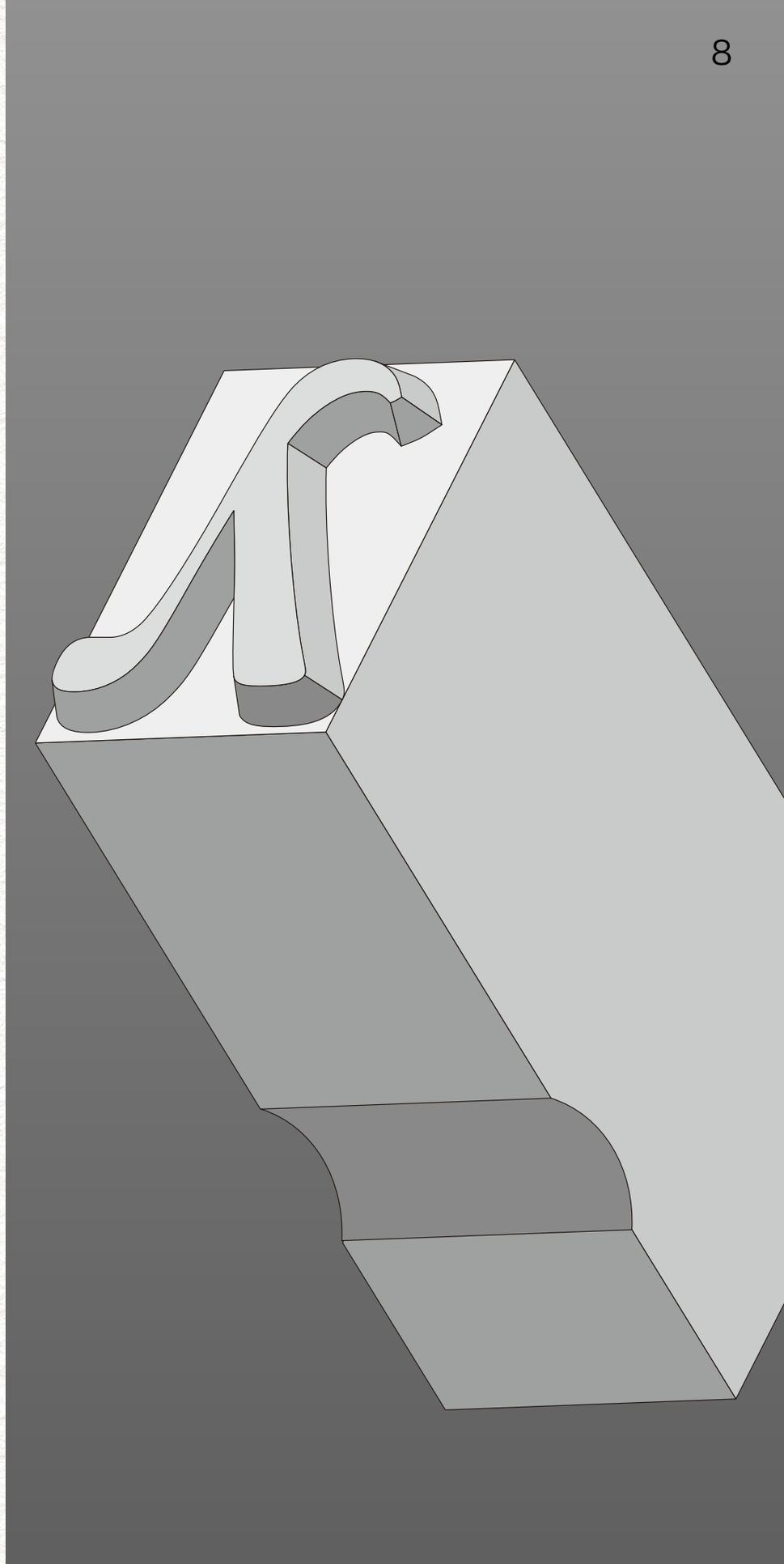
```
gfnMBP% satysfi paper.saty
```

U:**- *shell* Bot (1541,26) (Shell:run +3 Undo-Tree Helm)

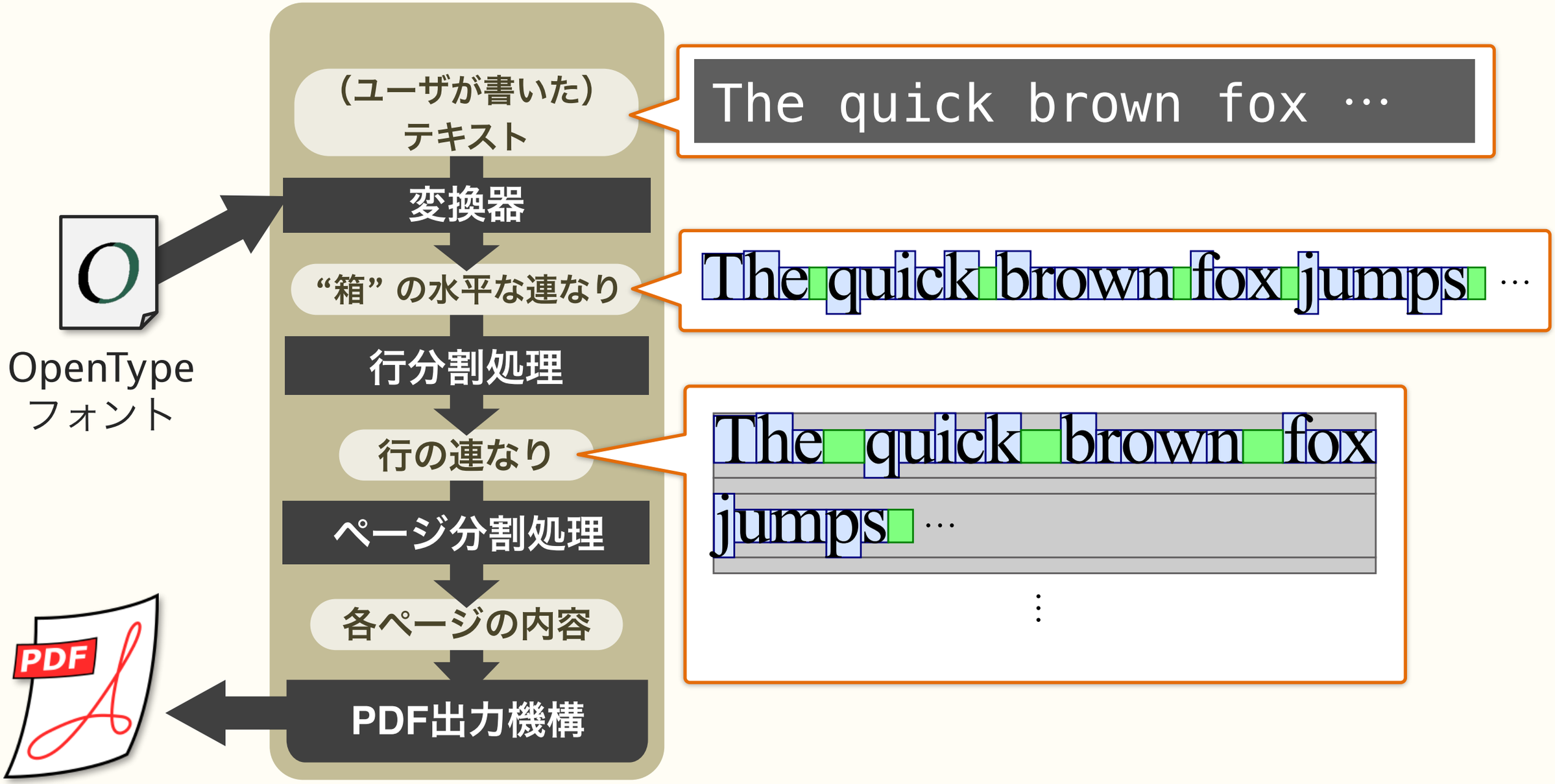
Mark set

目次

- 最初に寄り道： SATγSF| の紹介
- **文字の基本的な取扱い： ボックス列**
- 行分割処理の定式化
- 原稿からボックス列への変換方法の概要
- 難しい拡張（SATγSF| の Future Work）
- まとめ

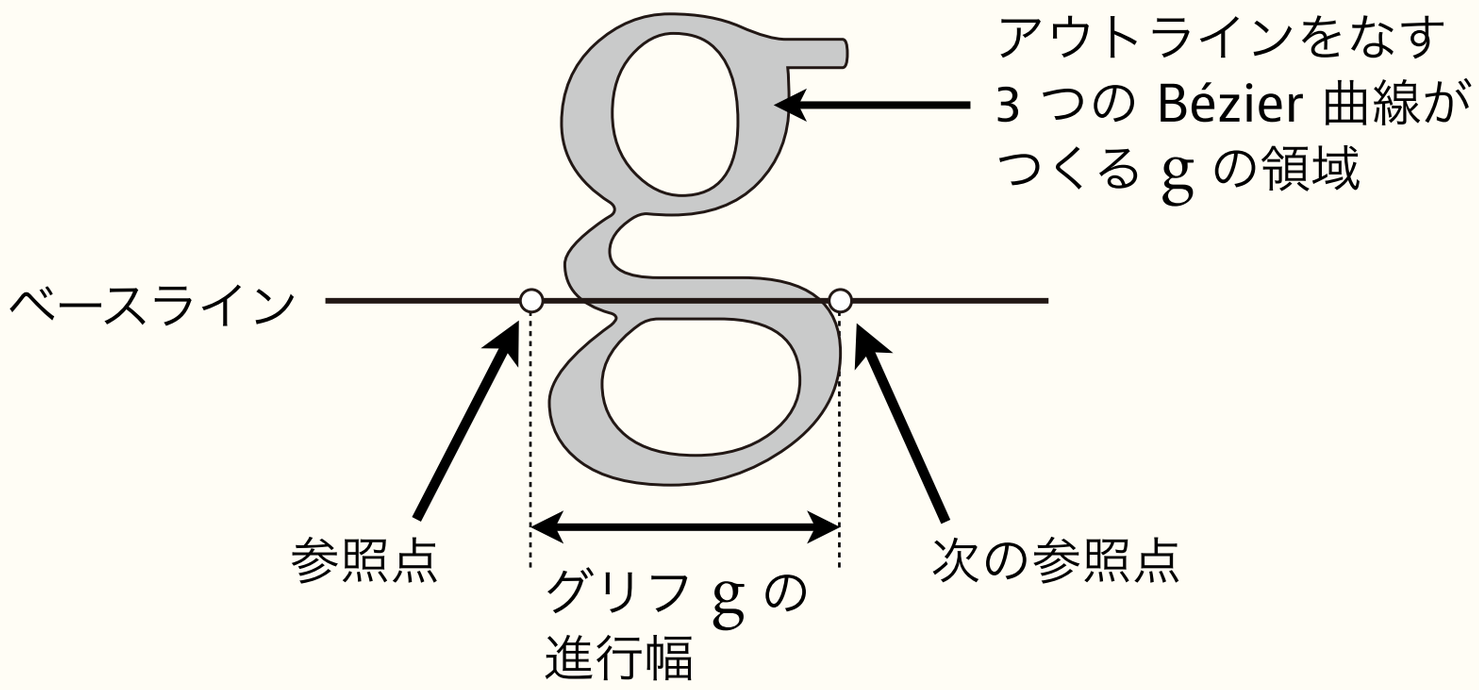


組版処理の模式図


OpenType
フォント
（ユーザが書いた）
テキスト
変換器
“箱”の水平な連なり
行分割処理
行の連なり
ページ分割処理
各ページの内容
PDF出力機構
PDF
The quick brown fox …
The quick brown fox jumps …
The quick brown fox
jumps …

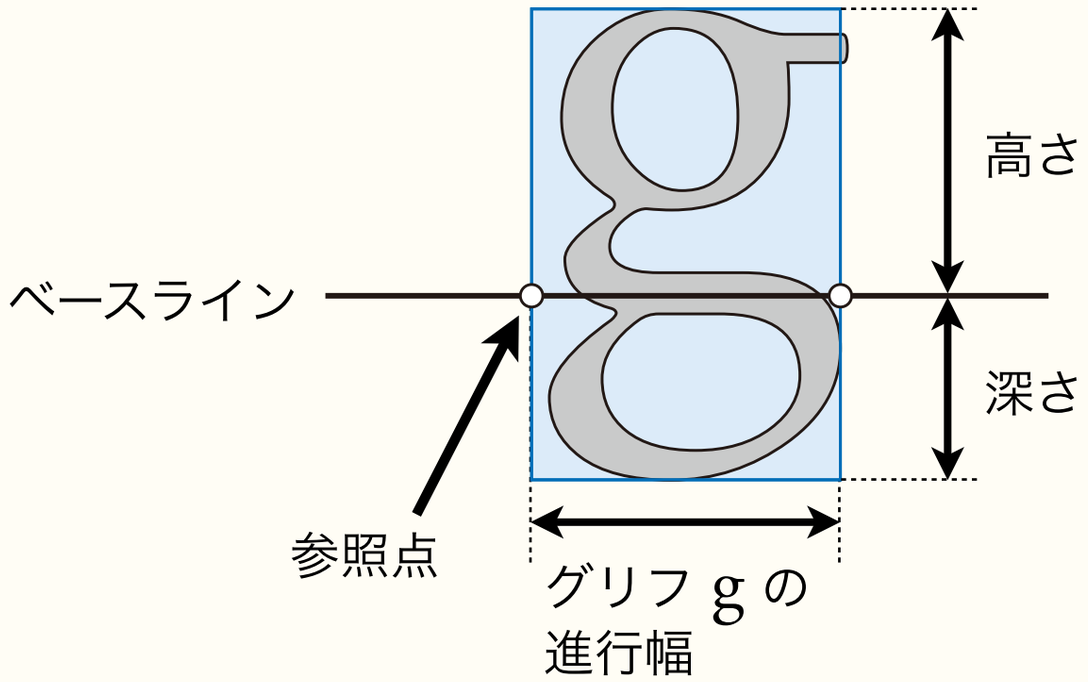
一般の組版処理に於ける文字の扱い方

- **グリフ** (*glyph*)： 図形としての文字を扱う単位
- グリフは**ベースライン** (*baseline*) を基準にして並ぶ
- 各グリフは以下のデータをもつ：
  - **アウトライン**： **参照点** (*reference point*) からの相対座標で記録された閉曲線
  - **進行幅** (*advance width*)： その文字の配置によって参照点をどれだけ進めるか



一般の組版処理に於ける文字の扱い方

- **グリフ** (*glyph*)： 図形としての文字を扱う単位
- グリフは**ベースライン** (*baseline*) を基準にして並ぶ
- 各グリフは以下のデータをもつ：
  - **アウトライン**： **参照点** (*reference point*) からの相対座標で記録された閉曲線
  - **進行幅** (*advance width*)： その文字の配置によって参照点をどれだけ進めるか



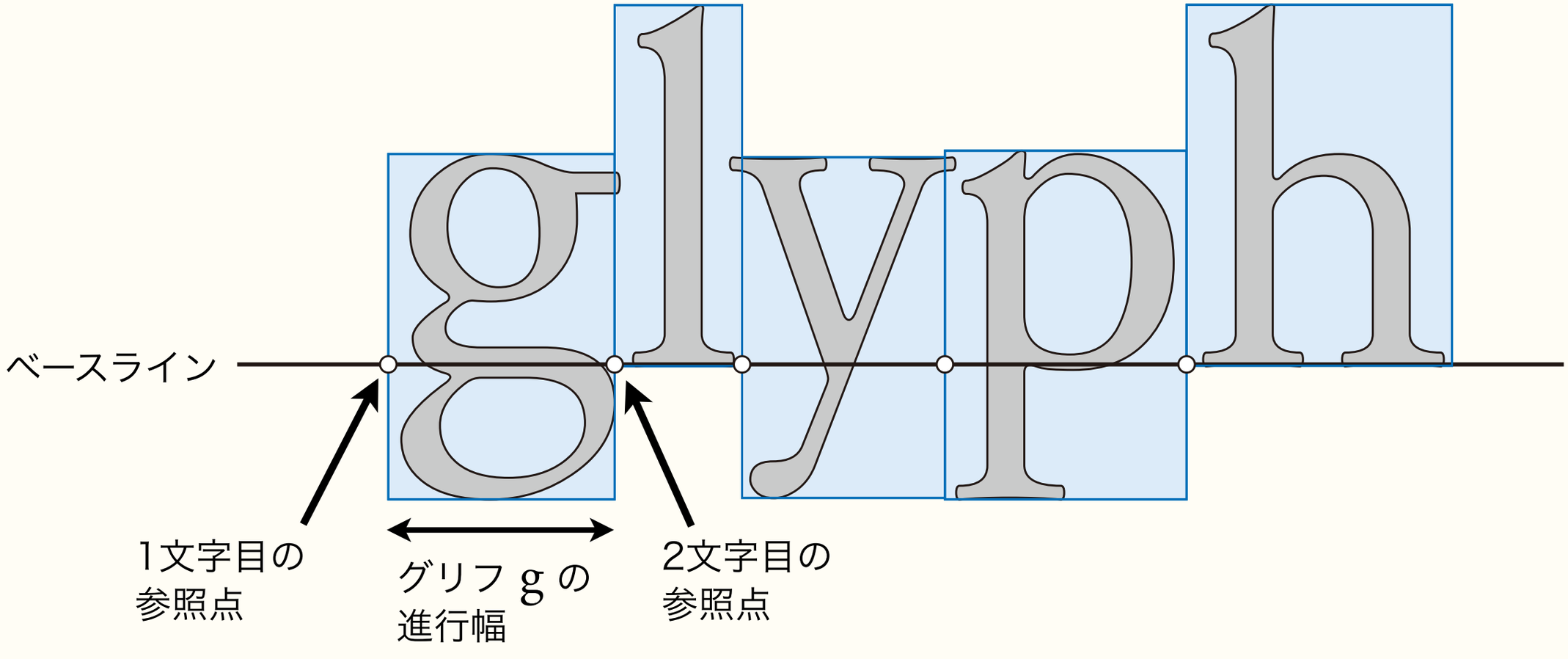
さらに以下で高さと深さを定義し，
グリフは仮想的な “箱” をもつとみなす

- **高さ**： max(0, 閉曲線の最高到達座標)
- **深さ**： max(0, −(閉曲線の最低到達座標))

※ 普遍的な定式化とまでは言えないが，
少なくとも TEX の組版処理 [Knuth 1978] では
この方法を採っており， SATYSFI でも踏襲

一般の組版処理に於ける文字の扱い方

- **グリフ** (*glyph*)： 図形としての文字を扱う単位
- グリフは**ベースライン** (*baseline*) を基準にして並ぶ
- 各グリフは以下のデータをもつ：
  - **アウトライン**： **参照点** (*reference point*) からの相対座標で記録された閉曲線
  - **進行幅** (*advance width*)： その文字の配置によって参照点をどれだけ進めるか



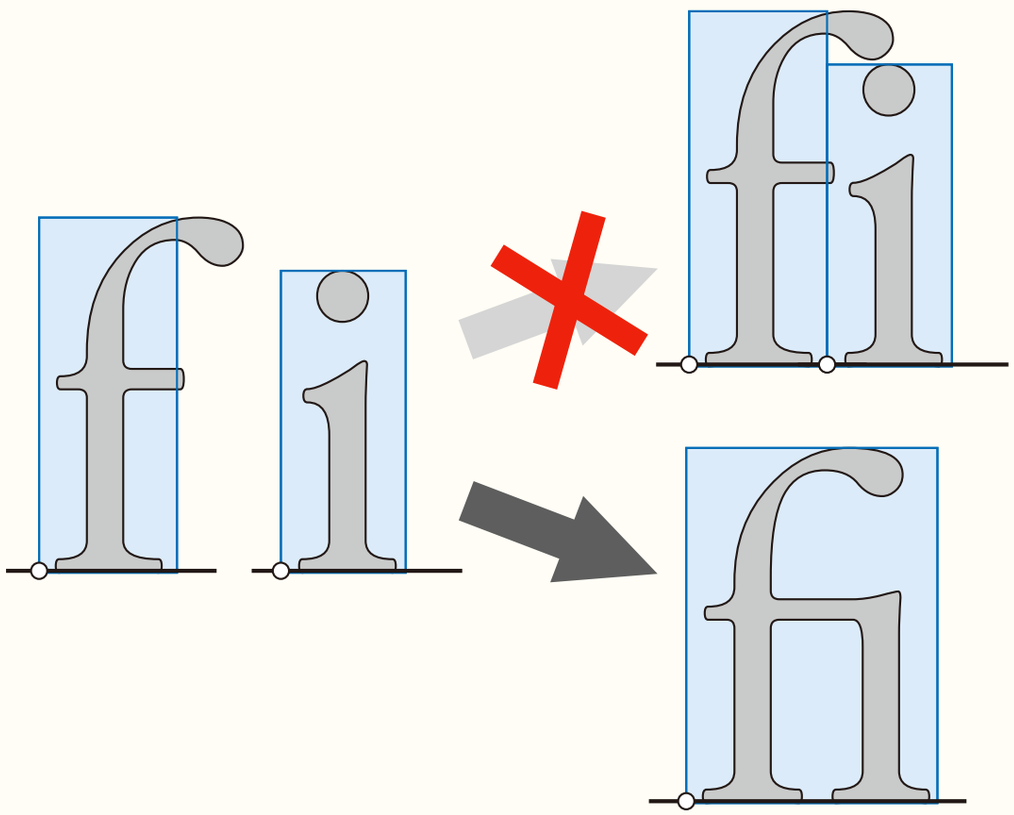
組版処理に於ける文字の定式化

アウトラインと進行幅だけでは支障をきたす場合もあるので,
特別な組み合わせに対してのみ適用される処理がある

- フォントファイル内にそのためのデータが格納されている

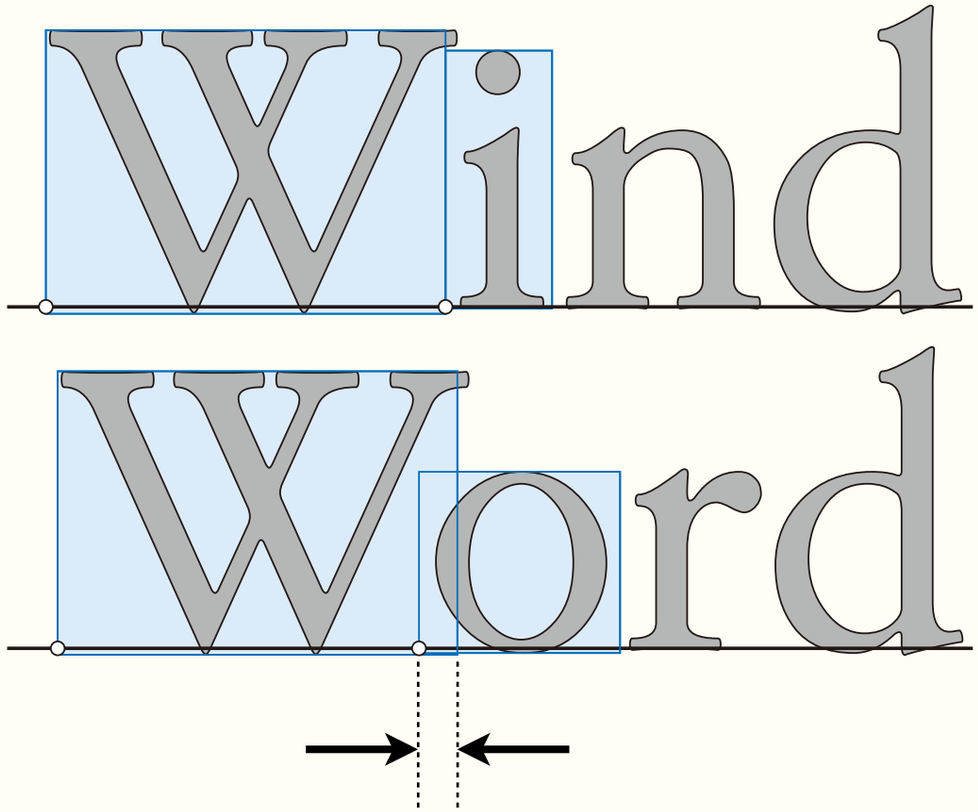
合字 (*ligature*)：

相性が悪い形状のグリフが並ぶときは
合体して 1 グリフにする



カーニング (*kerning*)：

進行幅に従うと字間が不自然になる
組み合わせでは調整を行なう



実際に組まれた合字・カーニングの例（デモ再掲）

プポーションなど，欧文組版に必要な高度な調整機能がいくつも備わっている．実際に以下の $\mathrm{SATYSFI}$ によって組まれた欧文を見てみよう：

The quick brown fox jumps over the lazy dog.

¿But aren't Kafka's Schloß and Æsop's Œuvres often naïve vis-à-vis the dæmonic phœnix's official rôle in fluffy soufflés?

The MIT License: Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the

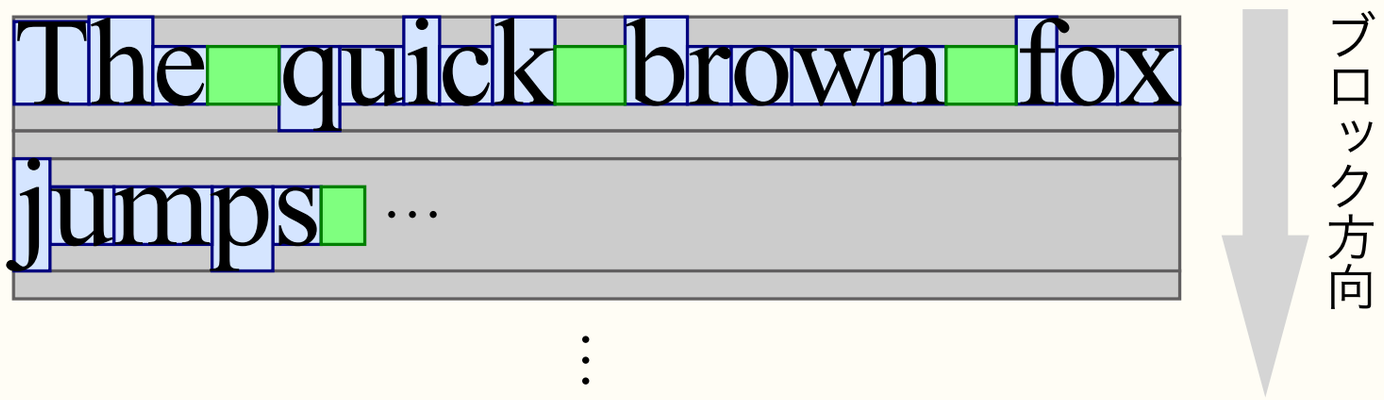
“箱”の連なりとしての行や段落

グリフの “箱” は（横書きでは）水平方向に並んで “行” をなす

- この “行” を**インラインボックス列**と呼ぶ

“行” は**行分割処理** (*line breaking*) によって切り分けられて “段落” になる

- 空白などを適切に伸縮し，両端が揃うように整形



操作対象としてのボックス列の定式化

インライン
ボックス列

$$ib ::= [ib]^*$$

0 個以上の
有限個の列

$$ib ::=$$

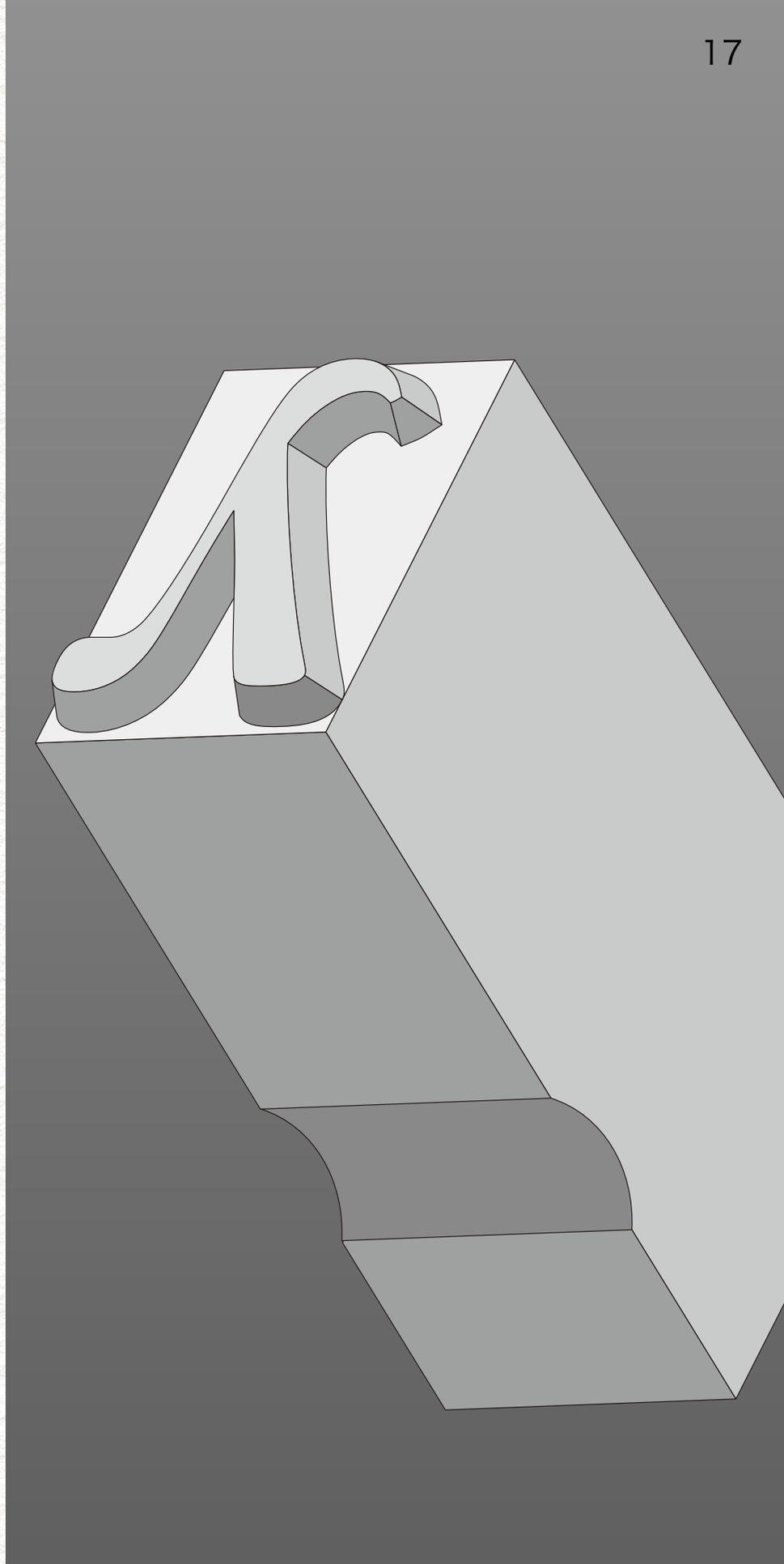
| e | … | あ | …　グリフ

|　空白

⋮

目次

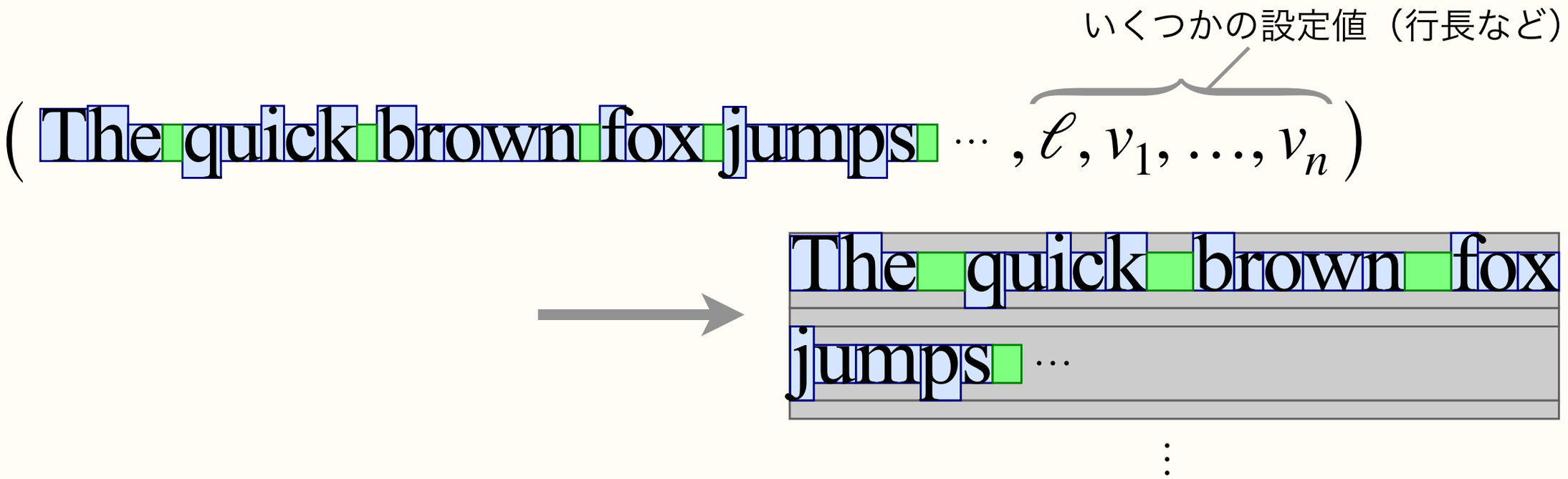
- 最初に寄り道： SATγSF| の紹介
- 文字の基本的な取扱い： ボックス列
- **行分割処理の定式化**
- 原稿からボックス列への変換方法の概要
- 難しい拡張（SATγSF| の Future Work）
- まとめ



行分割処理の定式化

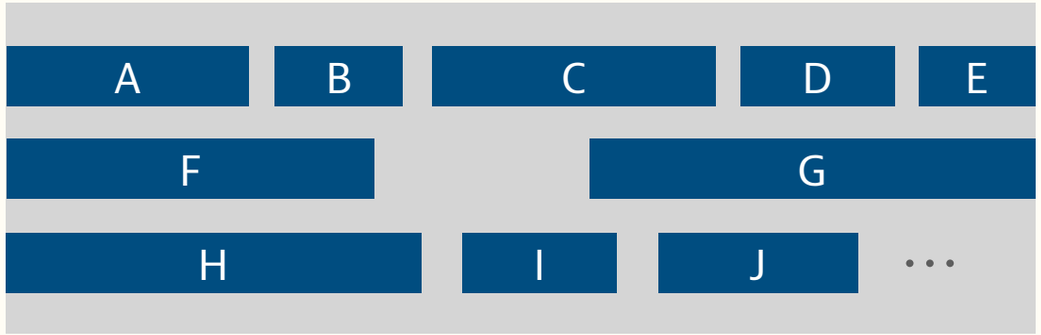
行分割処理問題

- **入力**： インラインボックス列 $\boldsymbol{ib}$, 所望の行長 ℓ
  - 他のパラメータは瑣末なので今回は捨象
- **出力**： 空白を適切に伸縮するなどして $\boldsymbol{ib}$ を各行の長さが ℓ になるように切り分けてできるブロックボックス列のうち**最も見映えの良いもの**
  - 見映えの良さの定量評価方法も人間の感覚を反映するように適切に定義

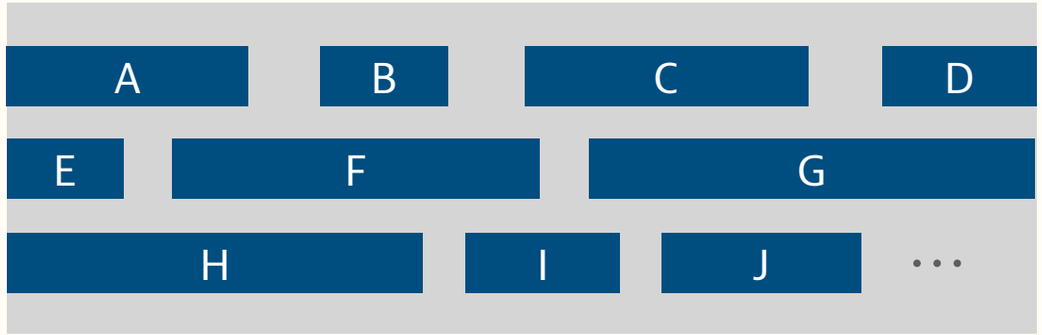


行分割処理の非自明さ

- 総当たりだと切り分け可能箇所の個数に対して指数時間かかる
- “前から順に詰めていき，溢れたらそこで行分割”という貪欲法では
  一般には最良の結果にならない



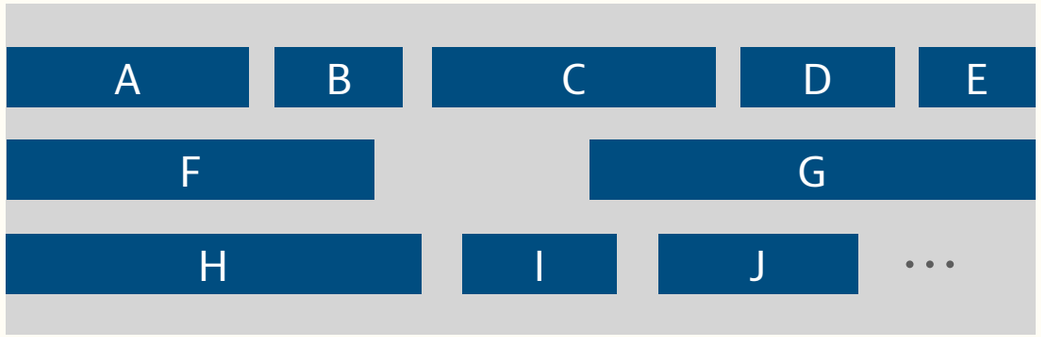
E まで 1 行目に収まるが，収めると
次の行がスカスカに



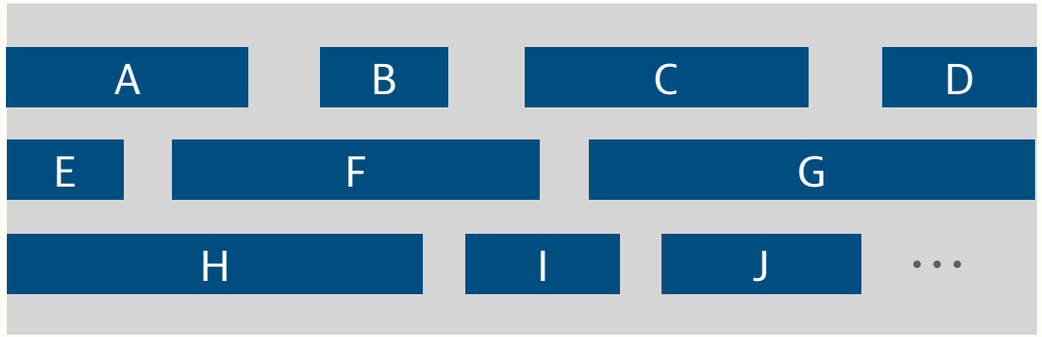
実は E を次の行へ送った方が
より自然な空白になり見映えが良かった

行分割処理の非自明さ

- 総当たりだと切り分け可能箇所の個数に対して指数時間かかる
- “前から順に詰めていき，溢れたらそこで行分割”という貪欲法では
  一般には最良の結果にならない



E まで 1 行目に収まるが，収めると
次の行がスカスカに



実は E を次の行へ送った方が
より自然な空白になり見映えが良かった

Knuth–Plass アルゴリズム [Knuth & Plass 1981] でそこそこ高速に求解可能

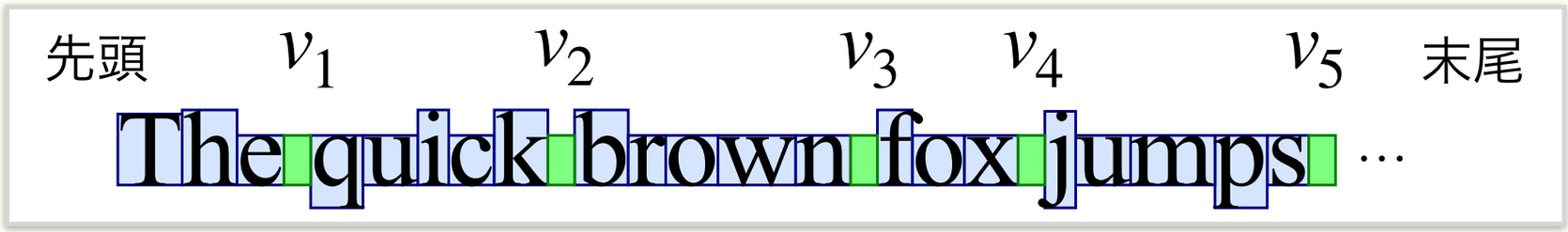
- TEX の行分割処理のために提案された
- SATYSFI もこれを踏襲（多少の独自拡張あり）

Knuth–Plass アルゴリズム [Knuth & Plass 1981]

行分割処理問題を**重みつき有向非巡回グラフ上の最短経路問題に帰着する**

内容 $\boldsymbol{ib}$ と行長指定 ℓ からグラフ $G = (V, E; d)$ を構成

- 頂点集合 $V :=$ ($\boldsymbol{ib}$ 中の行分割可能な箇所全体) $\uplus$ {先頭, 末尾}
  - 模式図：

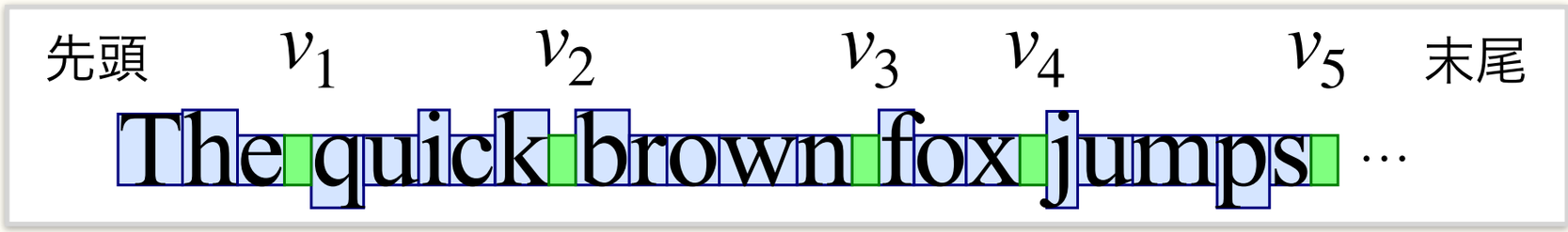


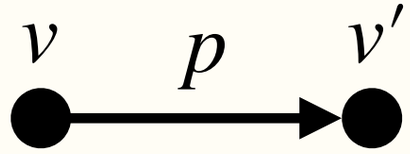
Knuth–Plass アルゴリズム [Knuth & Plass 1981]

行分割処理問題を**重みつき有向非巡回グラフ上の最短経路問題に帰着する**

内容 ***ib*** と行長指定 ℓ からグラフ $G = (V, E; d)$ を構成

- 頂点集合 $V :=$ (***ib*** 中の行分割可能な箇所全体) $\uplus$ {先頭, 末尾}
  - 模式図：



- 辺 

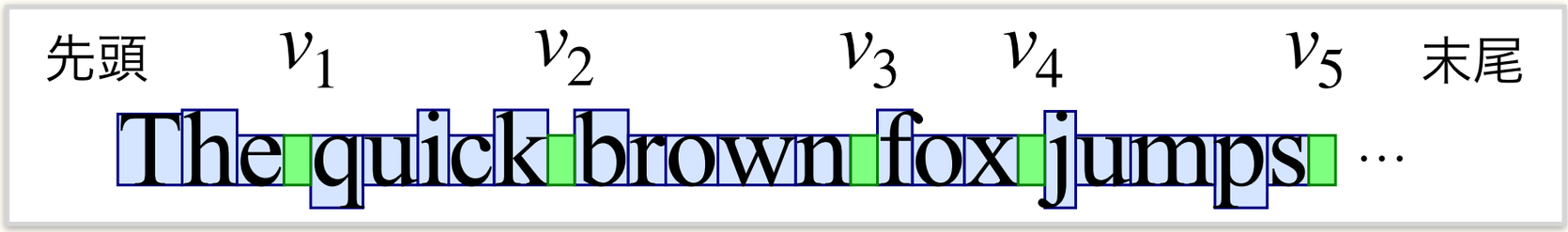
（***ib*** 内で v は v' よりも手前の点）
  - “**v で行分割してその次に v' で行分割したとき，v と v' の間の内容を長さが ℓ になるように 1 行として組むと，その見映えの評価は p**”
    - p は大きいほど見映えが悪いことを表すペナルティ値（決定方法は次頁）

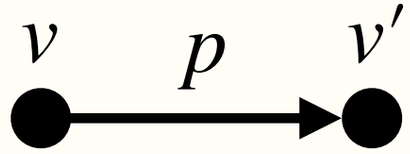
Knuth–Plass アルゴリズム [Knuth & Plass 1981]

行分割処理問題を**重みつき有向非巡回グラフ上の最短経路問題に帰着する**

内容 ***ib*** と行長指定 ℓ からグラフ $G = (V, E; d)$ を構成

- 頂点集合 $V :=$ (***ib*** 中の行分割可能な箇所全体) $\uplus$ {先頭, 末尾}
  - 模式図：



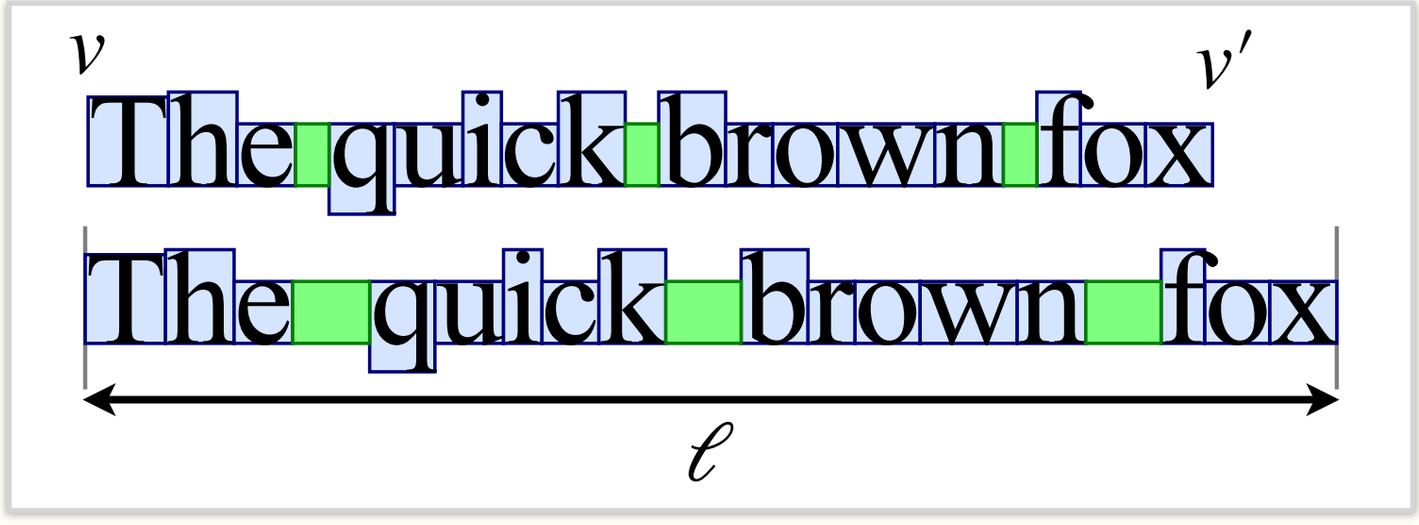
- 辺 

（***ib*** 内で v は v' よりも手前の点）
  - “**v で行分割してその次に v' で行分割したとき，v と v' の間の内容を長さが ℓ になるように 1 行として組むと，その見映えの評価は p**”
    - p は大きいほど見映えが悪いことを表すペナルティ値（決定方法は次頁）

G の「先頭」から「末尾」へ至る**最短経路を求めると，それが通る各頂点がペナルティ総和を最小化する（＝最も見映えの良くなる）切り分け箇所**

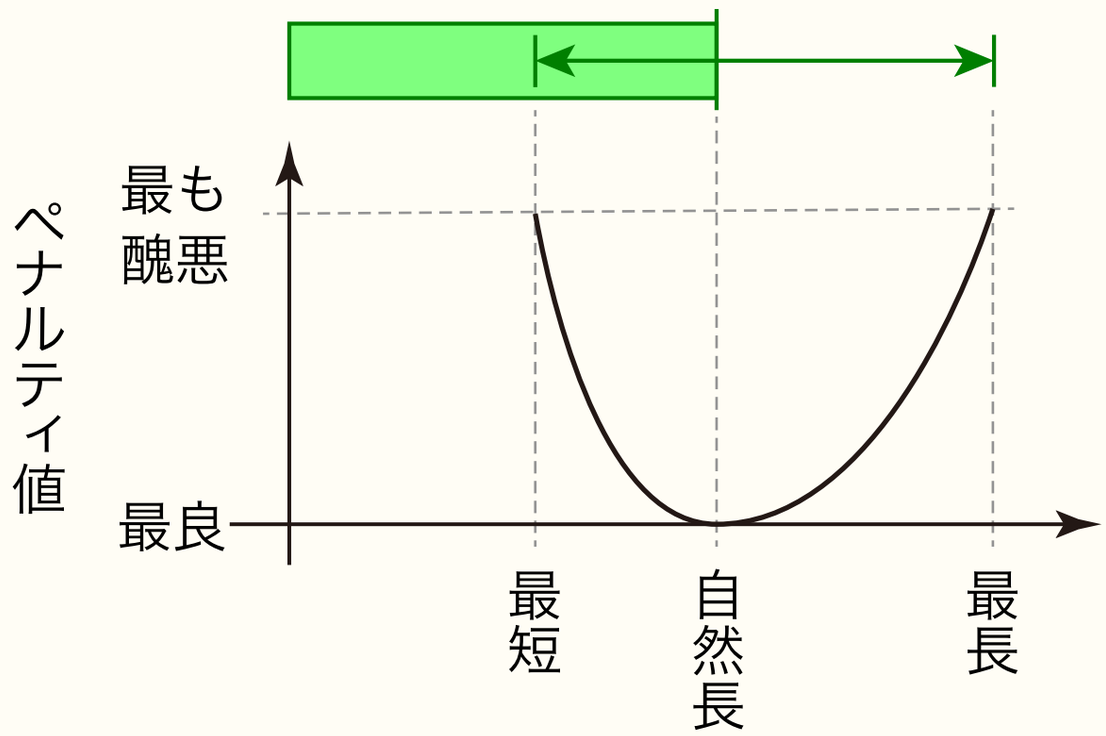
1 行の見映えの評価方法 [Knuth & Plass 1981]

行分割候補箇所 v とその次の候補箇所 v' を決めたら，空白を伸縮して
v と v' の間の内容が長さ ℓ になるようにし，その見映えを評価

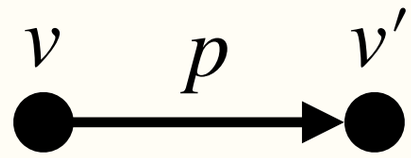


見映え評価のために，空白は
(自然長, 伸びる許容量, 縮む許容量)
という 3 つのパラメータを持つ

- バネのようなイメージ
- 自然長に近いほど見映えが良い
  ことを表す，下に凸な評価函数をもつ



時間計算量

- ひとまず普通に Dijkstra 法が適用でき，計算量は $O(|E| + |V| \log |V|)$
- 辺 

の数 $|E|$ は，愚直に v が v' より手前になるような全通りで張ると $O(|V|^2)$ になるが，実際には
明らかに見映えの悪い行になる組み合わせは事前に枝刈りでき，典型的にはそこまで大きくならない
  - 各頂点の出次数はせいぜい 10 程度なので “おおよそ線型”
  - ※ ペナルティ p が負（つまり積極的に改行してほしい場合）になることも一応ありうるので本当は Dijkstra 法はダメ

- **最初から頂点がトポロジカルソートされていることを利用し，動的計画法としても解ける**
  - これも $O(|V|^2)$ だが，典型的にはもっと高速
  - 解説は [黒木 2015] 参照

一般化された行分割可能箇所の指定方法

実際の行分割可能箇所は単語間の空白だけとは限らず,
もう少し一般化された仕組み

$$ib ::= [ib]^* \qquad ib ::= \text{e} \mid \cdots \mid \text{あ} \mid \cdots$$ グリフ

$$\mid \ \square$$ 伸縮する空白

$$\mid \left\{ \begin{matrix} ib \\ ib \ / \ ib \end{matrix} \right\}_p$$ **discretionary break**

$$\vdots$$

$$\left\{ \begin{matrix} ib_0 \\ ib_1 \ / \ ib_2 \end{matrix} \right\}_p \quad : \quad$$ “ここで行分割するかしないか選べる.
行分割しないなら ib_0 と等価で,
行分割するなら手前の行末に ib_1 を,
後方の行頭に ib_2 をそれぞれ結合し,
かつ分割自体によりペナルティ p が生じる”

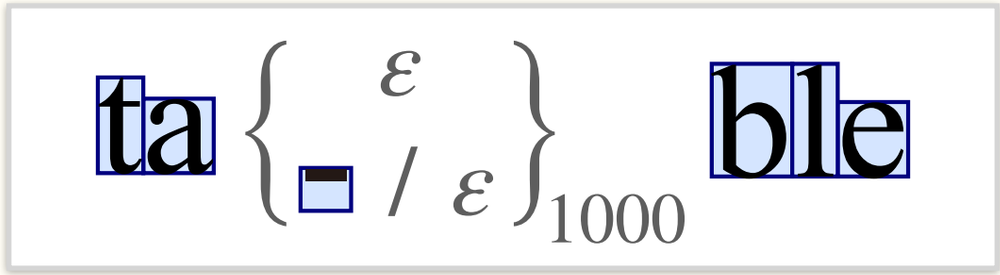
単語間の空白は $\left\{ \begin{matrix} \square \\ \varepsilon \ / \ \varepsilon \end{matrix} \right\}_{100}$ と表現され, 単独の □ は伸縮機能だけ

ε : 空列

Discretionary break による種々の行分割の表現

discretionary break の仕組みにより様々な行分割方法が表現できる：

- **ハイフネーション**による単語中での行分割：



  - **単語中の一部箇所は行分割してよく，分割時は手前の行末にハイフンが入る**
    - 例：“table” は “ta-ble” と分割してよいので上図のように扱う
  - **ハイフネーションは単語間での分割よりも好ましくないということを反映し，ペナルティを高めの 1000 に設定**
  - 実際は単語中で分割するかしないかで合字にするかやカーニングするかが変わりうるのでもう少し複雑な仕組み

Discretionary break による種々の行分割の表現

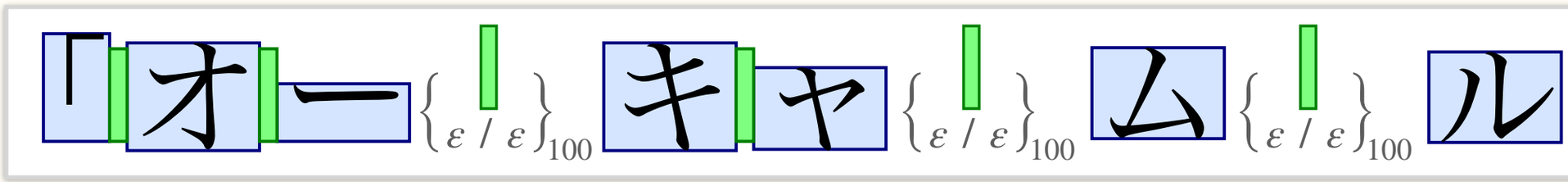
- 和文組版
  - 大抵の文字間は行分割してよい

    ➡ 原則として**各文字間に自然長 0 の空白をもつ discretionary break が入る**

  - ただし，行頭/行末禁則はある

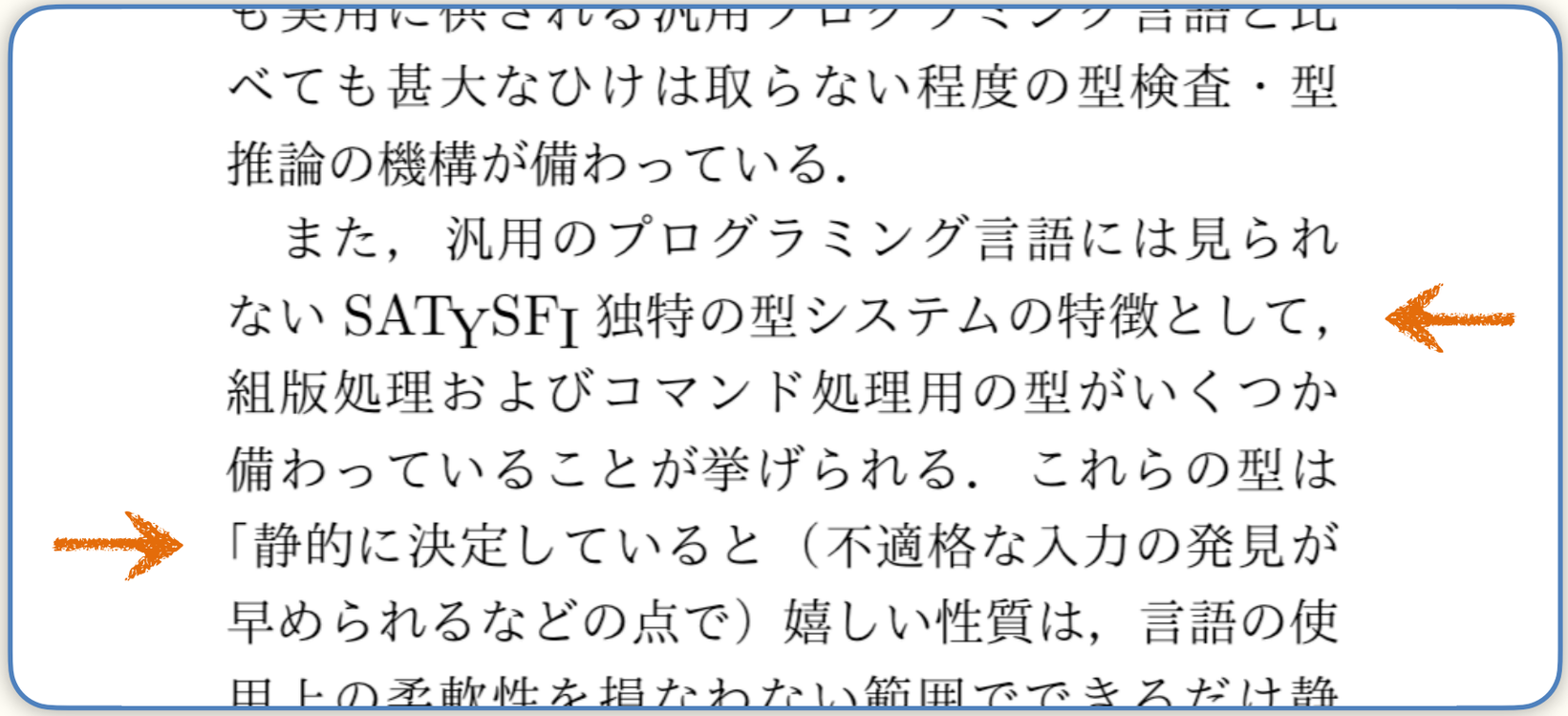
・行頭不可の文字： 閉じ括弧，句読点，小書き仮名「ゃ」，音引「ー」など

・行末不可の文字： 開き括弧など

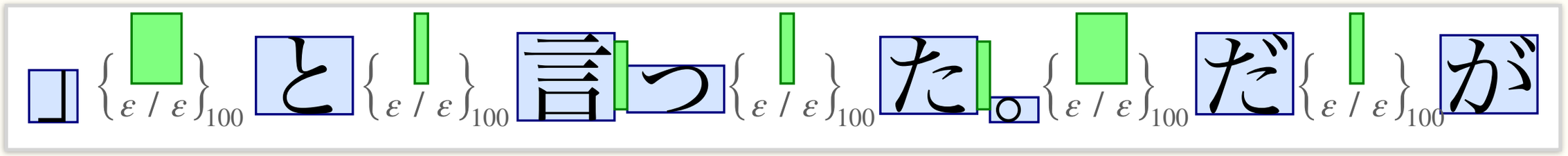


Discretionary break による種々の行分割の表現

- 行頭の開き括弧の直前や，行末の閉じ括弧や句読点の直後は，
  行中とは異なり二分アキ（＝いわゆる半角幅の空白）を入れない

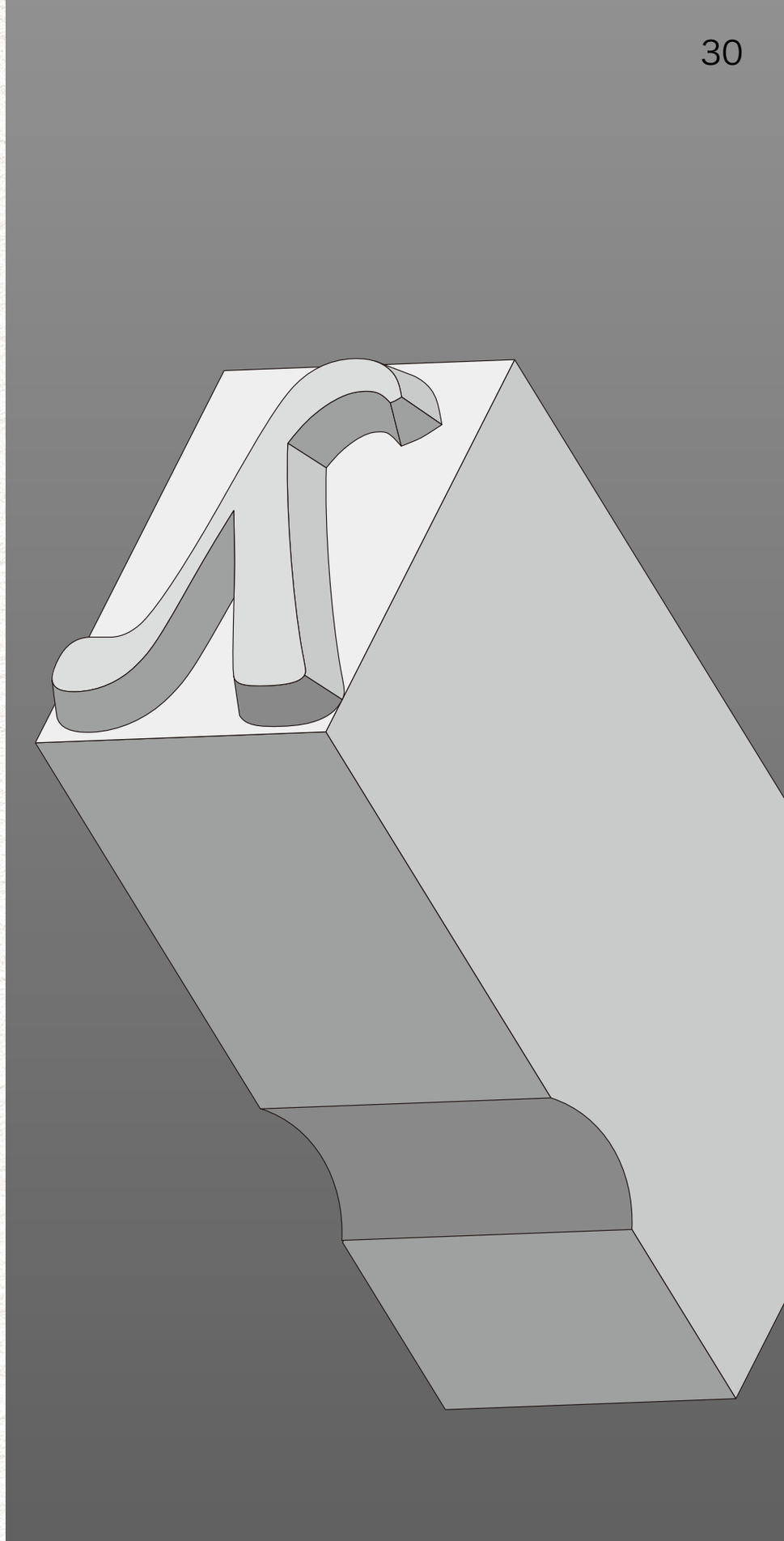


➡ 該当箇所は自然長が二分アキの空白をもつ discretionary break になる



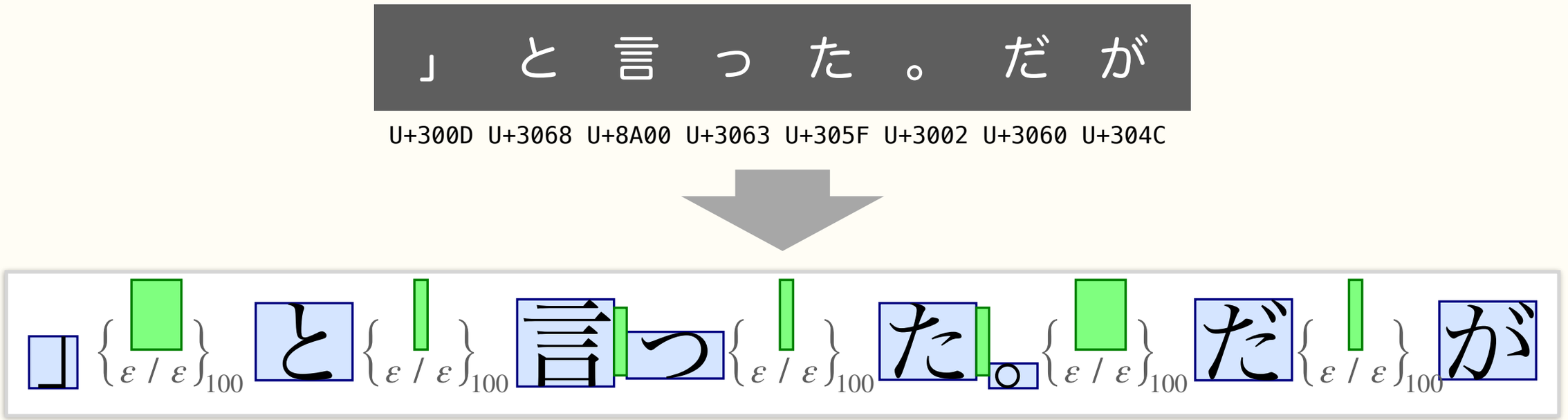
目次

- 最初に寄り道： SATγSFI の紹介
- 基本的な定式化： ボックス列
- 行分割処理の仕組み
- **原稿からボックス列への変換方法の概要**
- 難しい拡張（SATγSFI の Future Work）
- まとめ



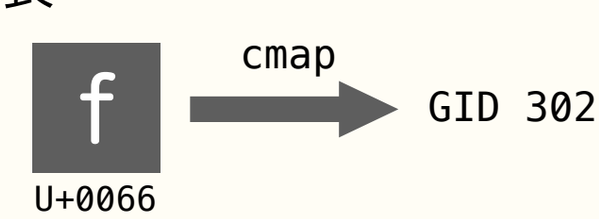
原稿からボックス列への変換

- ユーザは直接ボックス列を与えるわけではなく，テキストで原稿を書く
- 原稿 ＋ 各箇所で使うフォントなどのデータ からボックス列へと変換する何かしらの機構が必要
  - フォントファイルをデコードし，Unicode コードポイントから字形を取り出す
  - 合字やカーニングも組み合わせによって適切に処理
  - 欧文： **ハイフネーションをしてよい箇所を判定**
  - 和文： **行頭行末禁則や二分アキ・四分アキ挿入箇所を判定**



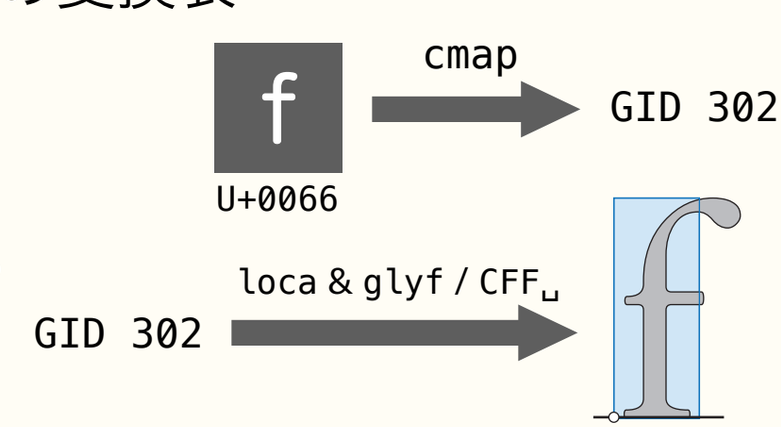
OpenType フォントのデコード

- 仕様書 *ISO/IEC 14496-22 Open Font Format* などに基づいてやるだけだが，歴史的経緯もあって仕様がわりと重厚
  - TrueType 系と CFF 系の 2 系統を半ば強引に 1 つの仕様につなぎ合わせた様相
- フォントファイルは 4 文字の名前がついたテーブルの集まり
  - **`cmap`**： コードポイントから，それに対応する最も標準的なグリフの **GID**（＝*glyph ID*，フォント内でのグリフの識別番号）への変換表



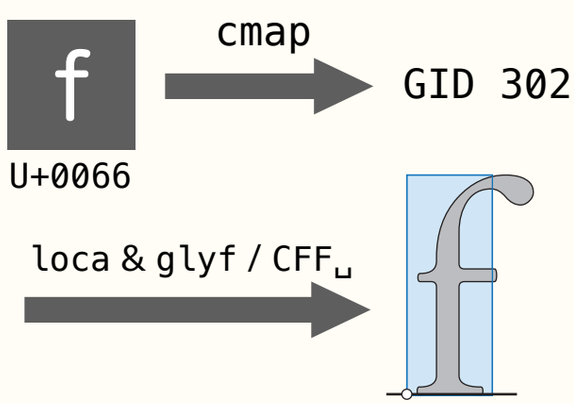
OpenType フォントのデコード

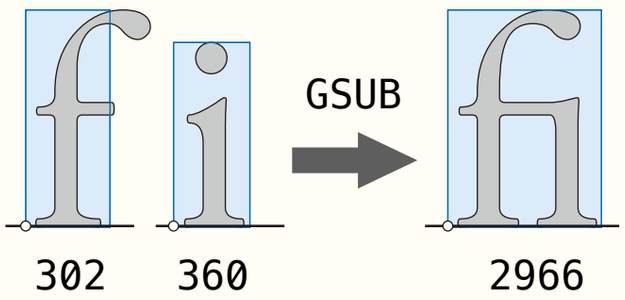
- 仕様書 *ISO/IEC 14496-22 Open Font Format* などに基づいてやるだけだが，歴史的経緯もあって仕様がわりと重厚
  - TrueType 系と CFF 系の 2 系統を半ば強引に 1 つの仕様につなぎ合わせた様相
- フォントファイルは 4 文字の名前がついたテーブルの集まり
  - **`cmap`**： コードポイントから，それに対応する最も標準的なグリフの **GID**（＝*glyph ID*，フォント内でのグリフの識別番号）への変換表
  - **`loca`** & **`glyf`**（TrueType 系にのみ存在）
    - `glyf` 内に各グリフのアウトラインが格納されており，GID で `loca` を表引きして `glyf` 内のオフセットを得る
  - **`CFF␣`**（CFF 系にのみ存在）
    - `loca`・`glyf` と同様の役割だが，複雑な形式なのでデコードが大変



OpenType フォントのデコード

- 仕様書 *ISO/IEC 14496-22 Open Font Format* などに基づいてやるだけだが，歴史的経緯もあって仕様がわりと重厚
  - TrueType 系と CFF 系の 2 系統を半ば強引に 1 つの仕様につなぎ合わせた様相
- フォントファイルは 4 文字の名前がついたテーブルの集まり
  - **cmap**： コードポイントから，それに対応する最も標準的なグリフの **GID**（＝*glyph ID*，フォント内でのグリフの識別番号）への変換表
  - **loca** & **glyf**（TrueType 系にのみ存在）
    - glyf 内に各グリフのアウトラインが格納されており，GID で loca を表引きして glyf 内のオフセットを得る
  - **CFF␣**（CFF 系にのみ存在）
    - loca・glyf と同様の役割だが，複雑な形式なのでデコードが大変
  - **GSUB**： 特定条件で GID 列を別の GID に置き換える指定
    - 合字に関する指定もこのテーブルに含まれている
  - **GPOS**： 特定条件でグリフの位置を調整するための指定
    - カーニングに関する指定もここにある



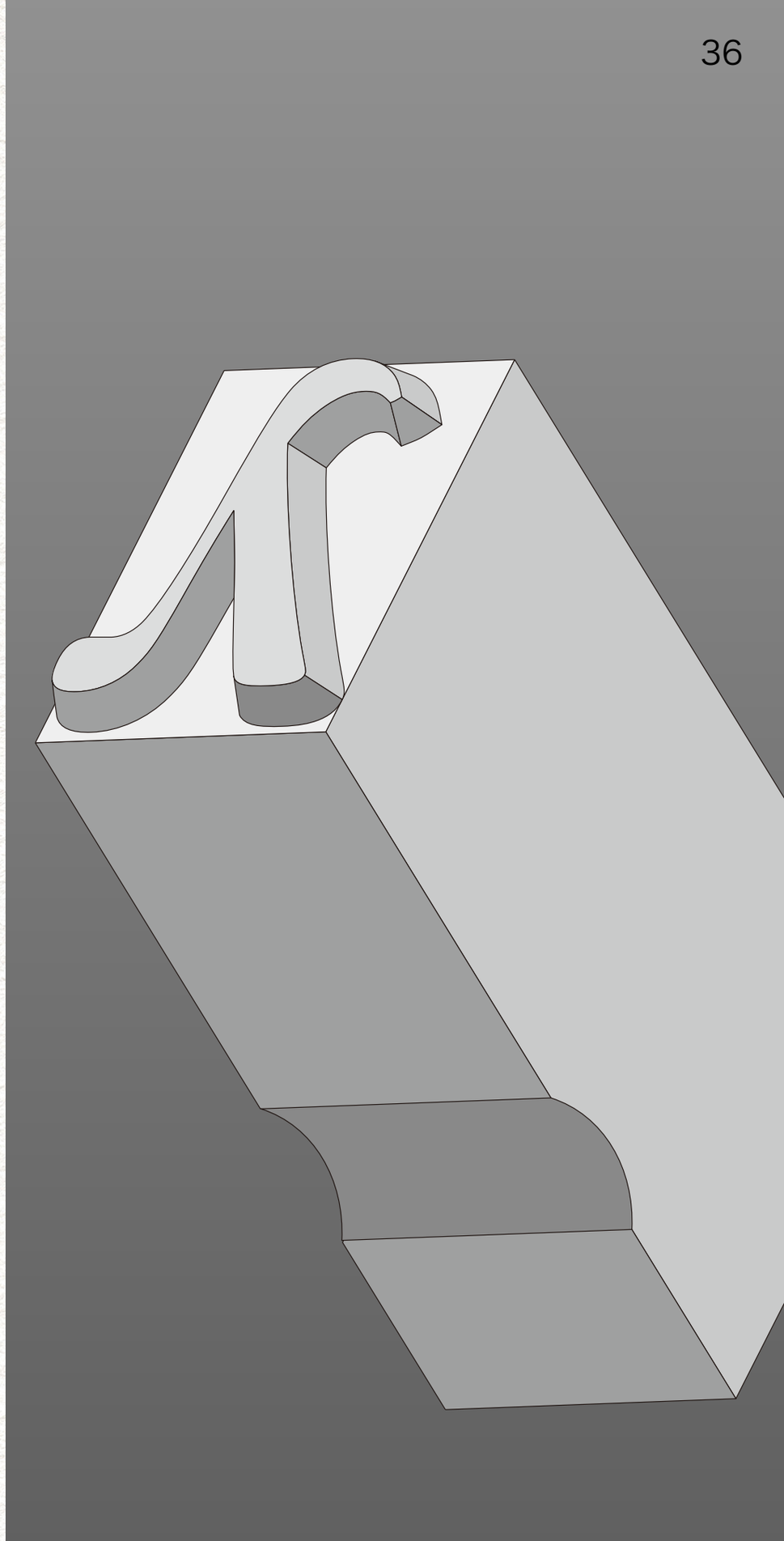


禁則とハイフネーション

- 和文： 行頭行末禁則など
  - 以下のような規則に準拠するとよい
    - 『**日本語組版処理の要件**』，通称 **JLreq** [W3C 日本語組版タスクフォース 2012]
    - **Unicode Line Breaking Algorithm** [Unicode Consortium 2017]
    - 分割可能箇所を決定する具体的アルゴリズムが与えられている
    - 前者は和文特化，後者はあらゆる文字体系を含めた最大公約数という様相
- 欧文： ハイフネーション
  - 英語だと，分割してよい箇所は単語ごとに決まっている
    - 辞書の各語の見出しに書いてある
  - とはいえ，愚直な表引きや大がかりな機械学習をせずとも **Liang–Knuth アルゴリズム** [Liang 1983] という推定アルゴリズムが使える
    - 最低限の実装はリストの線型走査だけでよい
    - Trie 木により対数時間まで下げられる

目次

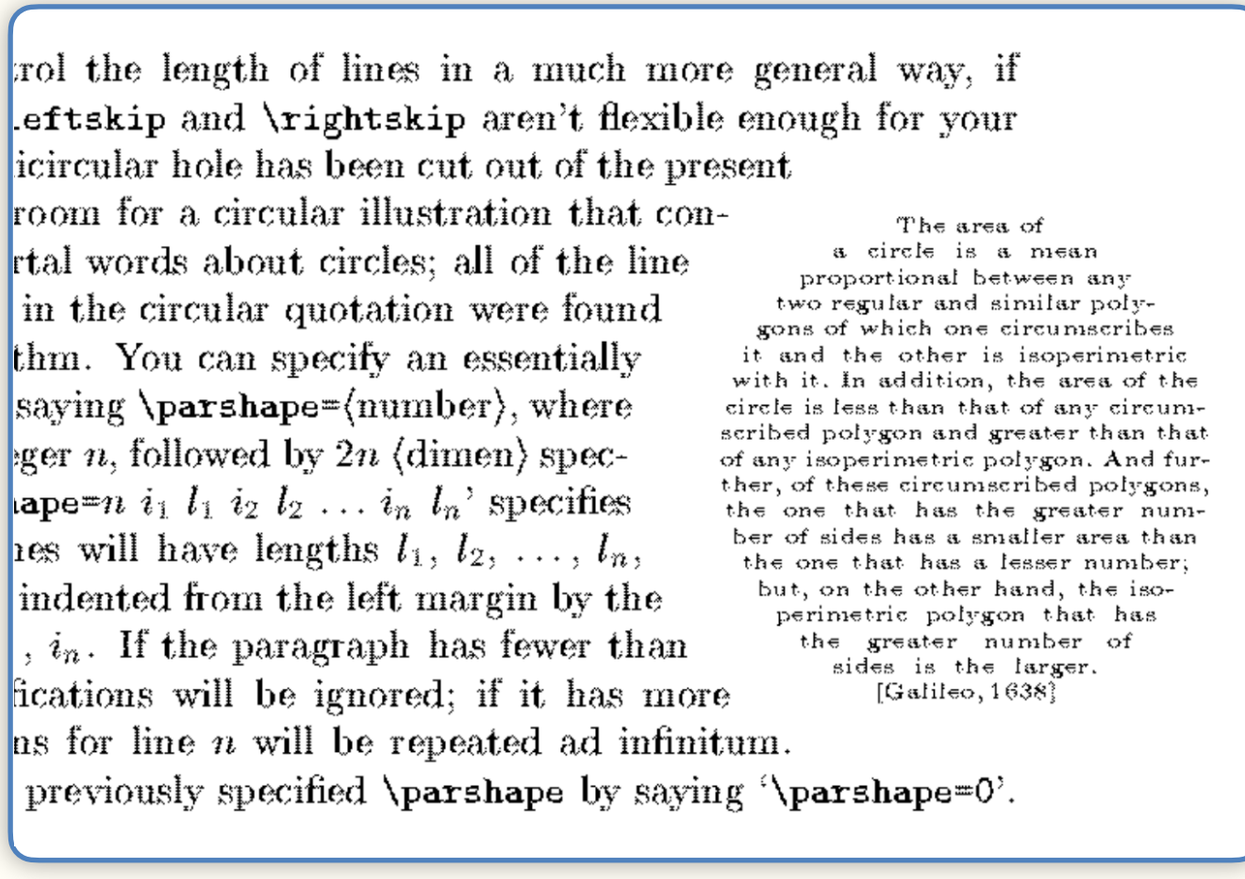
- 最初に寄り道： SATγSF$_I$ の紹介
- 基本的な定式化： ボックス列
- 行分割処理の仕組み
- 原稿からボックス列への変換方法の概要
- **難しい拡張（SATγSF$_I$ の Future Work）**
- まとめ



まずは簡単な拡張から

- **上下に隣接する 2 つの行の疎密の差を抑える**
  - 実は TEX が実際にこれをやっている
    - 伸縮の程度で $t ::=$ 密 | 普通 | 疎 | 超疎 の 4 段階に分類し，2 段階以上差のある行が並ぶのを防いでいる
  - 基本的にはグラフの頂点を v から (v, t) に変更するだけ
    - “v で行分割したとき，手前にきつさが t の行ができる”
    - 頂点数が 4 倍になるだけなので，時間計算量は定数倍

- **i 行目の行長を ℓ_i にして組む**
  - ただし $i \in \{1, \ldots, k\}$ の k は定数で，k 行目以降はずっと同じ行長
  - これも v を (v, i) に変更すればよく，グラフが k 倍になるだけ
    - まあ実質 k は $O(|V|)$ なのだが，それでも多項式時間ではある



rol the length of lines in a much more general way, if
.eftskip and \rightskip aren't flexible enough for your
icircular hole has been cut out of the present
room for a circular illustration that con-
rtal words about circles; all of the line
in the circular quotation were found
thm. You can specify an essentially
saying \parshape=⟨number⟩, where
ger n, followed by $2n$ ⟨dimen⟩ spec-
ape=$n\ i_1\ l_1\ i_2\ l_2\ \ldots\ i_n\ l_n$' specifies
es will have lengths $l_1, l_2, \ldots, l_n$,
indented from the left margin by the
, i_n. If the paragraph has fewer than
fications will be ignored; if it has more
ns for line n will be repeated ad infinitum.
previously specified \parshape by saying '\parshape=0'.

The area of
a circle is a mean
proportional between any
two regular and similar poly-
gons of which one circumscribes
it and the other is isoperimetric
with it. In addition, the area of the
circle is less than that of any circum-
scribed polygon and greater than that
of any isoperimetric polygon. And fur-
ther, of these circumscribed polygons,
the one that has the greater num-
ber of sides has a smaller area than
the one that has a lesser number;
but, on the other hand, the iso-
perimetric polygon that has
the greater number of
sides is the larger.
[Galileo, 1638]

難しい拡張の例

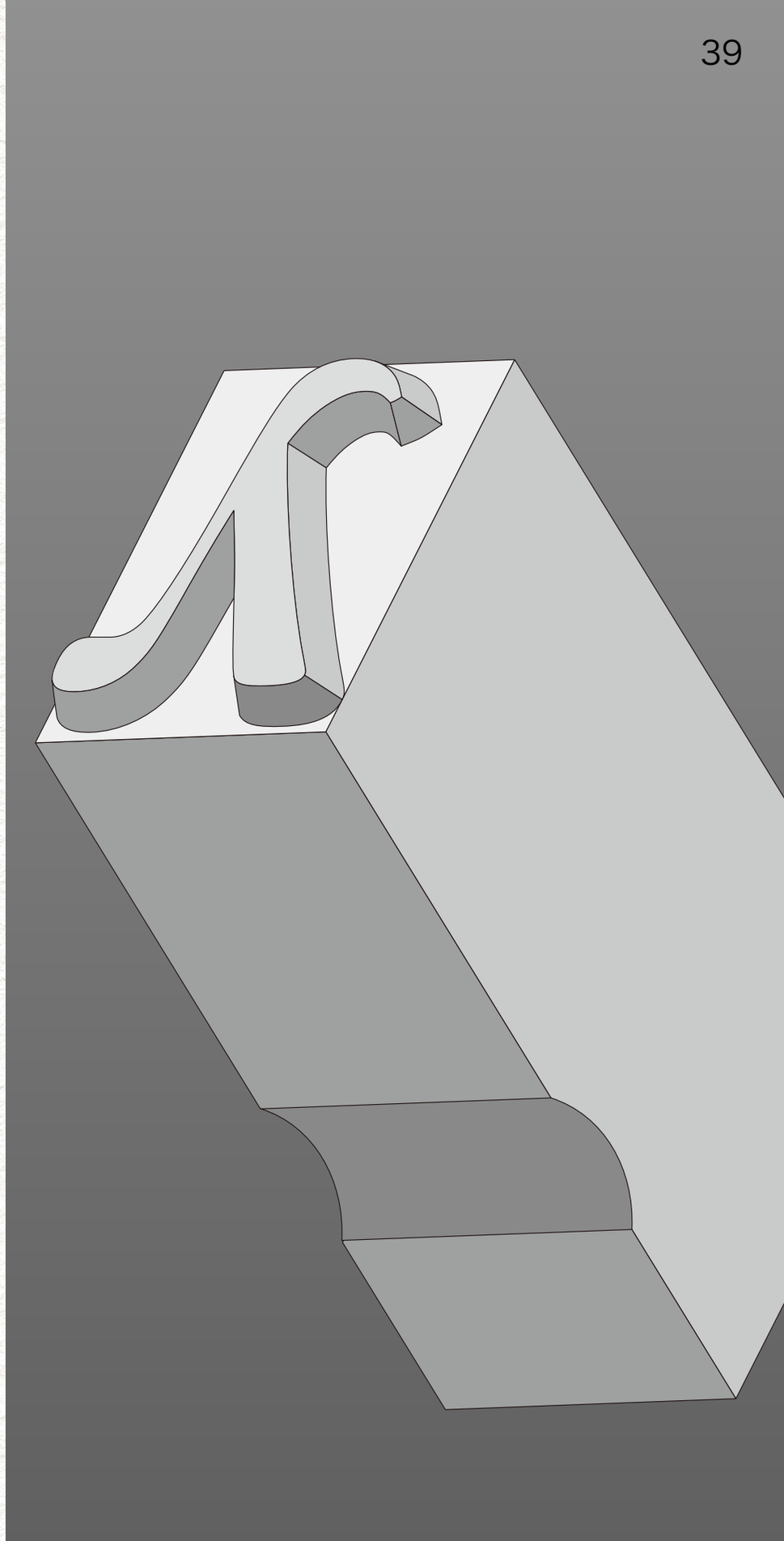
- **y 座標に応じて行長を $\ell(y)$ にして組む**
  - 先ほどの指定と似ているが，**一般には行ごとに高さが違うため劇的に難しくなる**
  - $\ell(-)$ が任意の曲線の場合，厳密には示していないが，どうやら NP 困難
  - $\ell(-)$ を非自明な特定のクラスに限定すれば多項式時間で解けるだろうか？
  - もし最適が無理だとしても，近似解法はないか？

- 和文の**割註**
  - 不確定要素がたくさんある
    - 均等に小さい 2 行に分ける場所の決定
    - 割註全体の長さはどうなるか
    - 割註自体が次の行にまでまたがる場合の考慮
  - ほぼ和文でしか使わないので，無理して Knuth–Plass アルゴリズムと混ぜる恩恵はないかも

和文には割註（このように2行に分けて組む註）があります

目次

- 最初に寄り道： SATγSF| の紹介
- 基本的な定式化： ボックス列
- 行分割処理の仕組み
- 原稿からボックス列への変換方法の概要
- 難しい拡張（SATγSF| の Future Work）
- **まとめ**



まとめ

組版処理のうち，特にアルゴリズム的に面白い部分である
行分割処理の定式化とその解法について簡単に紹介しました

- 定式化上の主な特徴：
  - ボックス列によるグリフと空白などの取り扱い
  - discretionary break による種々の行分割可能位置の表現
- 解法：
  - 最も見映えの良い切り分け方を得る処理は
    有向非巡回グラフの**最短経路問題**に帰着できる！
- 難しい拡張：
  - y 座標に応じて行長が変わるような段落を組む
  - 割註（本文中に小さい文字で 2 行で組まれる註釈）

参考文献

- Adobe Systems. *The Type 2 Charstring Format*. Technical Note #5177, 2000.
- Adobe Systems. *The Compact Font Format Specification*. Technical Note #5176 Version 1.0, 2003.
- ISO/IEC 14496-22:2019 *Information technology — Coding of audio-visual objects Part 22: Open Font Format*. 2019.
- Donald Knuth. *The TEXbook*. American Mathematical Society and Addison–Wesley, 1984.
- Donald Knuth and Michael Plass. Breaking paragraphs into lines. *Software–Practice and Experience*, 11, pages 1119–1184, 1981.
- Franklin Liang. *Word Hy-phen-a-tion by Com-put-er*. Ph. D. thesis, Stanford University, 1983.
- John Plaice and Yannis Haralanbous. Draft documentation for the Ω system. http://www.bakoma-tex.com/doc/omega/base/doc-18.pdf, 1999.
- Unicode Consortium. *Unicode Standard Annex #14: Unicode Line Breaking Algorithm (Unicode 10.0.0)*. http://unicode.org/reports/tr14/, 2017.
- W3C 日本語組版タスクフォース. W3C 技術ノート 日本語組版処理の要件. 東京電機大学出版局, 2012.
- 黒木 裕介. 組版におけるオペレーションズ・リサーチ. オペレーションズ・リサー チ 60(9), pages 536–542, 2015.