Π Ε Ι Θ Ω

# Peitho

# なぜつくったか

スライドをMarkdownで書くのはずっとやってきた。Decksetはデザインの自由度がなく、k1LoW/deckは思想は最高だけど、デザイン調整はGoogle Slidesでの手作業になる。

デザイン成果物がHTML/CSSというただのコードなら、生成AIに書かせられるし、gitでdiffもレビューもできる。

# 設計の三本柱

- **コンテンツとデザインの分離** — コンテンツはMarkdown、デザインはレイアウトHTMLとCSS。両者を混ぜない
- **バージョン管理できるデザイン** — 成果物はただのHTML/CSS。diffが取れ、レビューできる
- **型検査されるスロット契約** — 内容がレイアウトの宣言に合わないときは、黙って捨てずにビルドエラー

# デッキはただのMarkdown

`---` で区切ると1枚のスライド。最も浅い見出しがタイトル、コードブロックはcodeスロット、残りはbodyスロットへ自動で割り当てられる。

```
# 1枚目のスライド

本文の段落。

- リストも
- 使える

---

# 2枚目のスライド
```

# レイアウト自身がスキーマ

デザインの単位はレイアウトで、ただのHTMLファイル。スロットの名前・受け入れる型・個数を宣言し、ビルド時にPeithoがここから契約を取り出す。

```
<section class="peitho-slide">
  <h1><slot name="title" accepts="inline" arity="1"></slot></h1>
  <div class="body">
    <slot name="body" accepts="blocks" arity="0..*"></slot>
  </div>
</section>
```

# 合わなければビルドが止まる

スロットの過不足も、型の不整合も、行番号とヒント付きのビルドエラーになる。「黙って何かが起きる」経路を残さない。

```
error: slide 2 ('code-slide'), line 7: slot 'code' got 2 item(s),
       but layout 'title-body-code' allows 0..1
  = help: use a layout with more code capacity or remove one code block
```

# キー付きper-slide調整

スライドに安定キーを振ると`data-slide-key`属性が付いて、CSSからそのスライドだけを狙える。デッキに存在しないキーを指定したCSSはビルドエラーになるので、参照切れが静かに残らない。

```
[data-slide-key="payoff"] .slot-code {
  background: #0d2822;
}
```

# 型駆動ディスパッチ

レイアウトは複数持てて、スライドの内容の形にスロット契約が一致するレイアウトへ自動で振り分けられる。

- ページ設定コメントでの明示指定が最優先
- 自動で決まるのは、一致がちょうど1枚のときだけ
- 複数一致も0枚一致もビルドエラー。黙って解決しない

# デモサイトpeitho.gosu.ke

リポジトリの`examples/`を`peitho build`してそのまま配信したもの。同じツール、同じMarkdown規約で、まったく違うデッキになる。

# Keynote — クリーム地とセリフ体

cover(タイトルのみ)とstatement(タイトル+本文)の2レイアウト構成。内容の形から型駆動ディスパッチで振り分けられる。

# Code Walkthrough — コード必須のレイアウト

codeスロットが`arity="1"`なので、毎スライドにコードが必須。シンタックスハイライトはsyntectでビルド時にかけていて、実行時JSはない。

```
// Phases live in the type
```

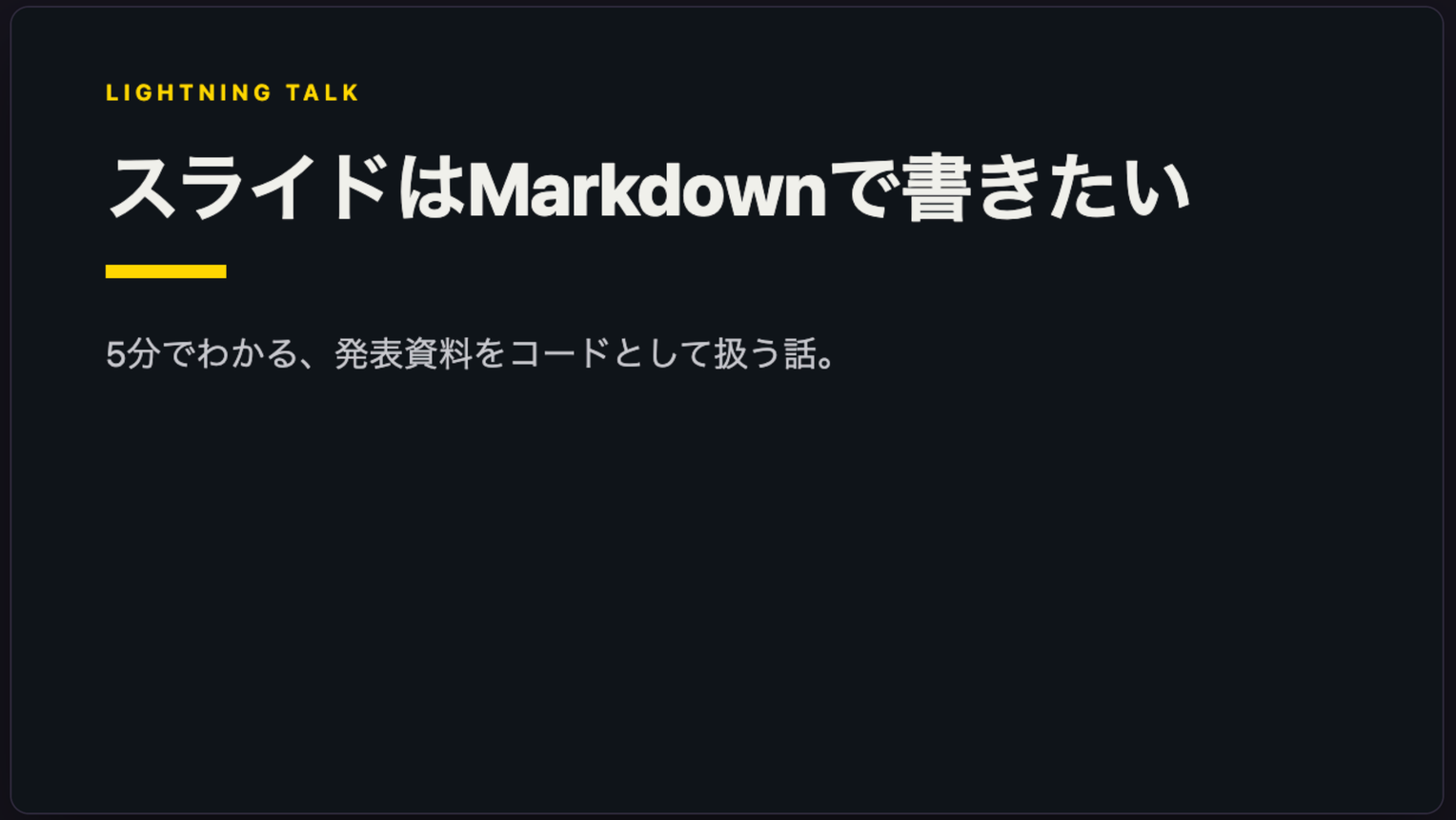
```
pub struct Deck<Phase> {
    slides: Vec<Slide>,
    _phase: PhantomData<Phase>,
}

pub enum Parsed {}
pub enum Checked {}
pub enum Rendered {}
```

> A deck carries its pipeline phase as a type parameter.

> The phase markers are empty enums: they exist only at compile time.

# Lightning Talk — コードを書けないデッキ

レイアウトにcodeスロットがないので、コードブロックを書くとビルドエラーになる。レイアウトが「このスライドに何を書けるか」を決めて、ビルドが守らせる。



# peitho presentで発表する

ローカルサーバが立ち上がり、外部ディスプレイにスライドをフルスクリーン、手元に発表者ツールを自動配置。Escで全部まとめて終了する。

# 残り時間で色が変わるタイマー

frontmatterの`time: 15m`が計画時間になる。経過・計画・超過が並び、残り3分でオレンジ、残り1分で赤、超過すると`+MM:SS`が追加で出る。

# かめ🐢とうさぎ🐰のトラッカー

かめが経過時間、うさぎがスライドの進捗。位置関係だけで「今のペースで間に合うか」が読み取れる。数字の暗算は発表中にやりたくない。

# 超過したセクションが赤くなるアジェンダ

セクションごとに計画・実績・差分が並び、発表中のセクションも計画を超えた瞬間に赤くなる。セクション合計とデッキ全体の `time` がズレていたらビルドエラー。

| AGENDA | ACTUAL / PLANNED | |
|---|---|---|
| Setup 01 | 0:57 / 1:00 | -0:03 |
| Problem 02 | 0:55 / 1:00 | -0:05 |
| **Approach** 03–04 | 2:20 / 2:00 | · |
| Wrap-up 05 | — / 1:00 | · |

# 検査を通らないとレンダリングできない

ビルドコアはRust。デッキの型がパイプラインの段階ごとに分かれていて、レンダラは検査済みの型しか受け取れない。検査をすっ飛ばすコードはコンパイルが通らない。

```
// Parsed → Mapped → Checked → Rendered
fn render_deck(deck: Deck<Checked>) -> Result<Deck<Rendered>>
```

ΠΕΙΘΩ

# 言葉で、人の心を動かす