BACKSLASH DESIGN

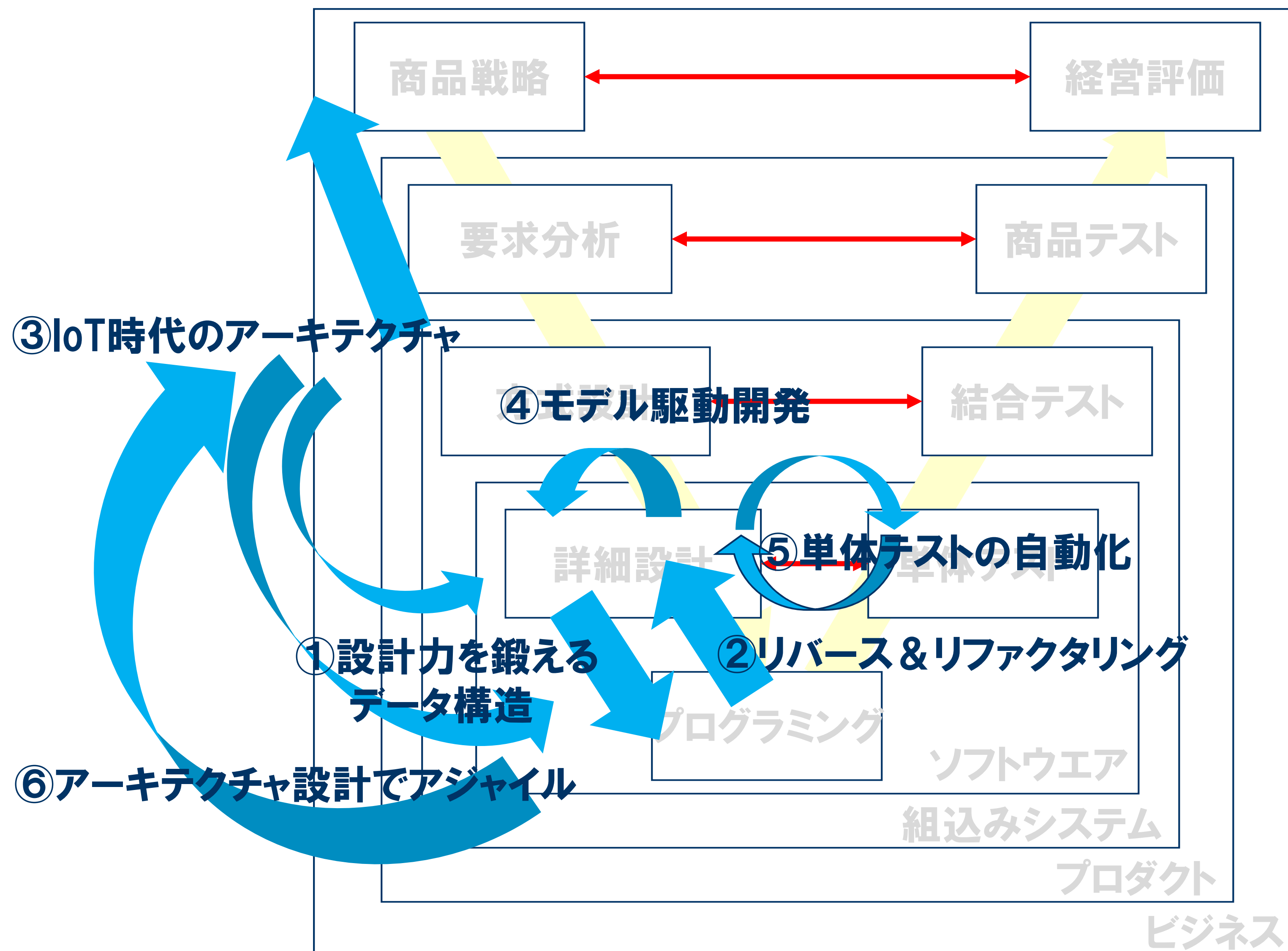
設計力を鍛える
～データ構造を設計していますか？～

2017年7月6日

主催：ビースラッシュ株式会社

Copyright BACKSLASH DESIGN Co., Ltd.

本日の全体像



Copyright BACKSLASH DESIGN Co., Ltd.

本セッションの内容

BACKSLASH DESIGN

1. よく見かける二大課題
   - 動解析ファースト
   - ソフトウェア疲労
2. 末広がり vs 構造化
3. 静解析ファースト：データ構造の設計
4. 設計図から考えよう

Copyright BACKSLASH DESIGN Co., Ltd.

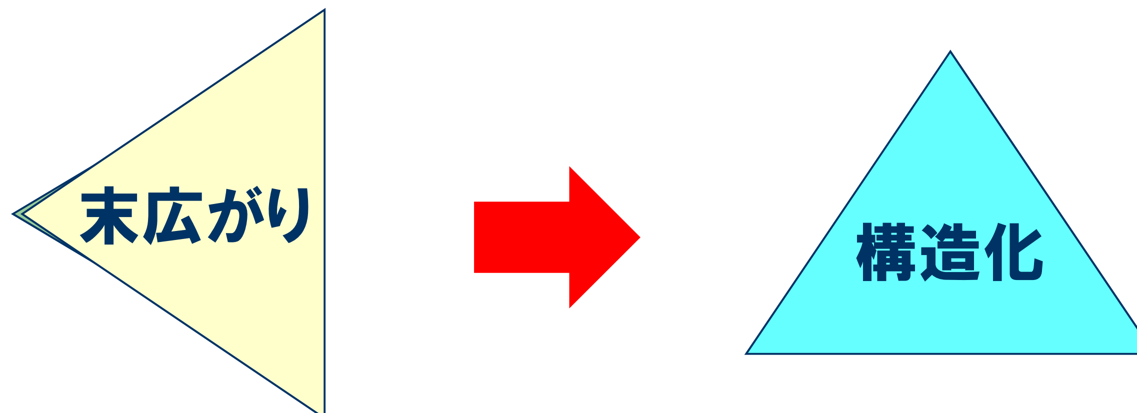
BACKSLASH DESIGN

1. よく見かける二大課題

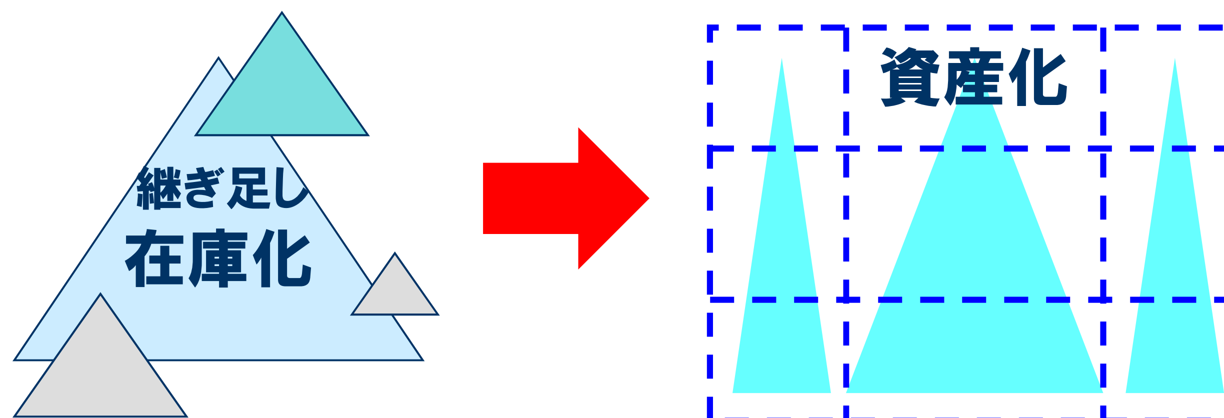
Copyright BACKSLASH DESIGN Co., Ltd.

二大課題

- アセンブラ的プログラムからモジュール化プログラムへの転換



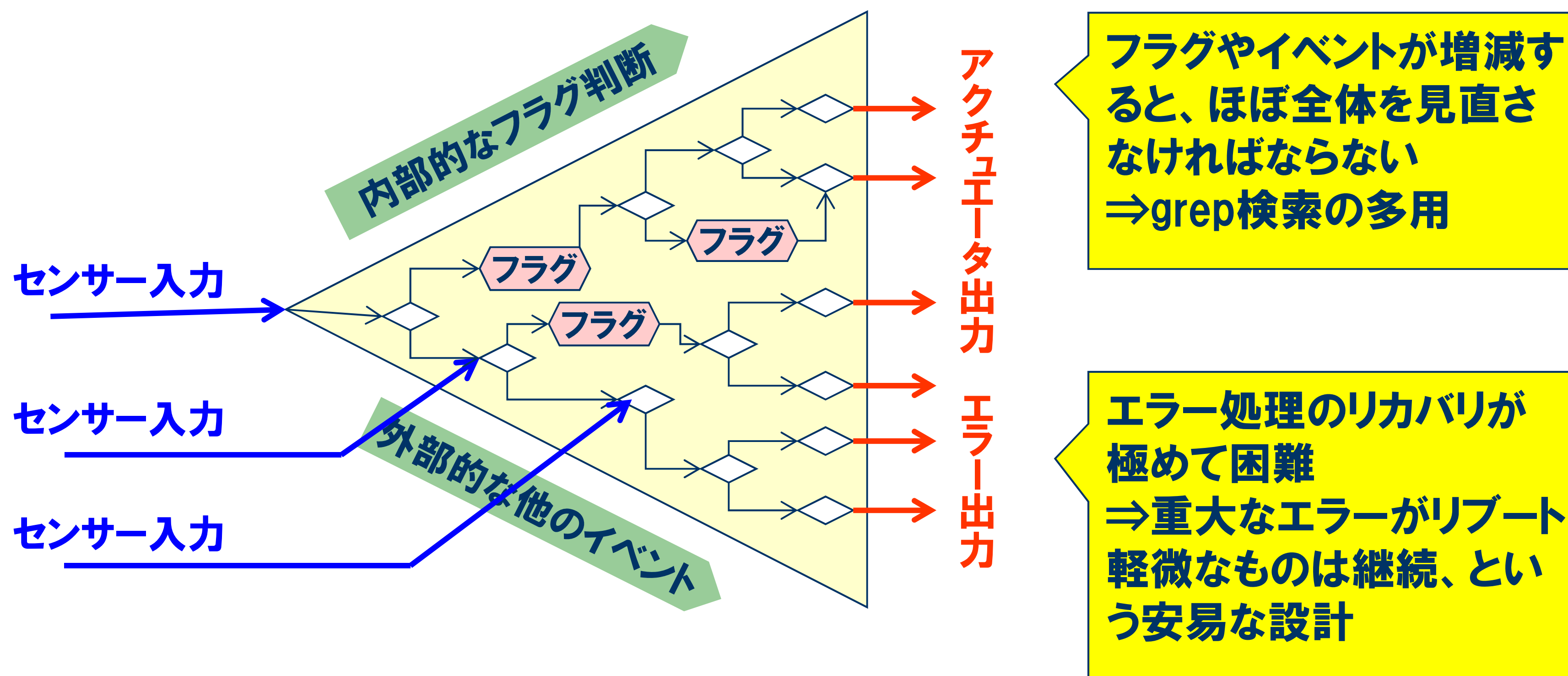
- 在庫化プログラムから資産化プログラムへの転換



Copyright BACKSLASH DESIGN Co., Ltd.

第一の問題：動解析ファースト

- 制御仕様をそのままプログラミングすると「末広がり」になる
  - 内部的なフラグ判断の積み重ね
  - 外部的な他のイベントの積み重ね



Copyright BACKSLASH DESIGN Co., Ltd.

第二の問題：ソフトウェア疲労

BACKSLASH DESIGN

ID	症状	別名	現象
S1	一筆書き	モンスター関数	ひとつの関数やファイルが長い 作った人にしかわからない（作った人も？）
S2	クローン	切り貼りプログラミング	同じコード断片やデータ構造が点在している 修正漏れが発生する（ここにもあった、、、）
S3	神様データ	グローバルデータ フラグ	すべてを支配しているデータが存在する 修正の波及範囲が大きい
S4	中央集権	責務過多 肥満児	ひとつのファイルにたくさんの関数がある いつも同じファイルを修正している
S5	スパゲッティ	くもの巣 もぐらたたき	いろいろな関数を呼び出している 副作用が発生する（こちらを直すとあちらが）
S6	老舗温泉旅館	場当たり的 ルール無用	階層を越えた呼び出し／命名規則がない 引継ぎができない（そもそも説明できない）
S7	一枚岩	モノリシック フレンド	#includeしているファイルが多い／循環依存 分割コンパイルができない

Copyright BACKSLASH DESIGN Co., Ltd.

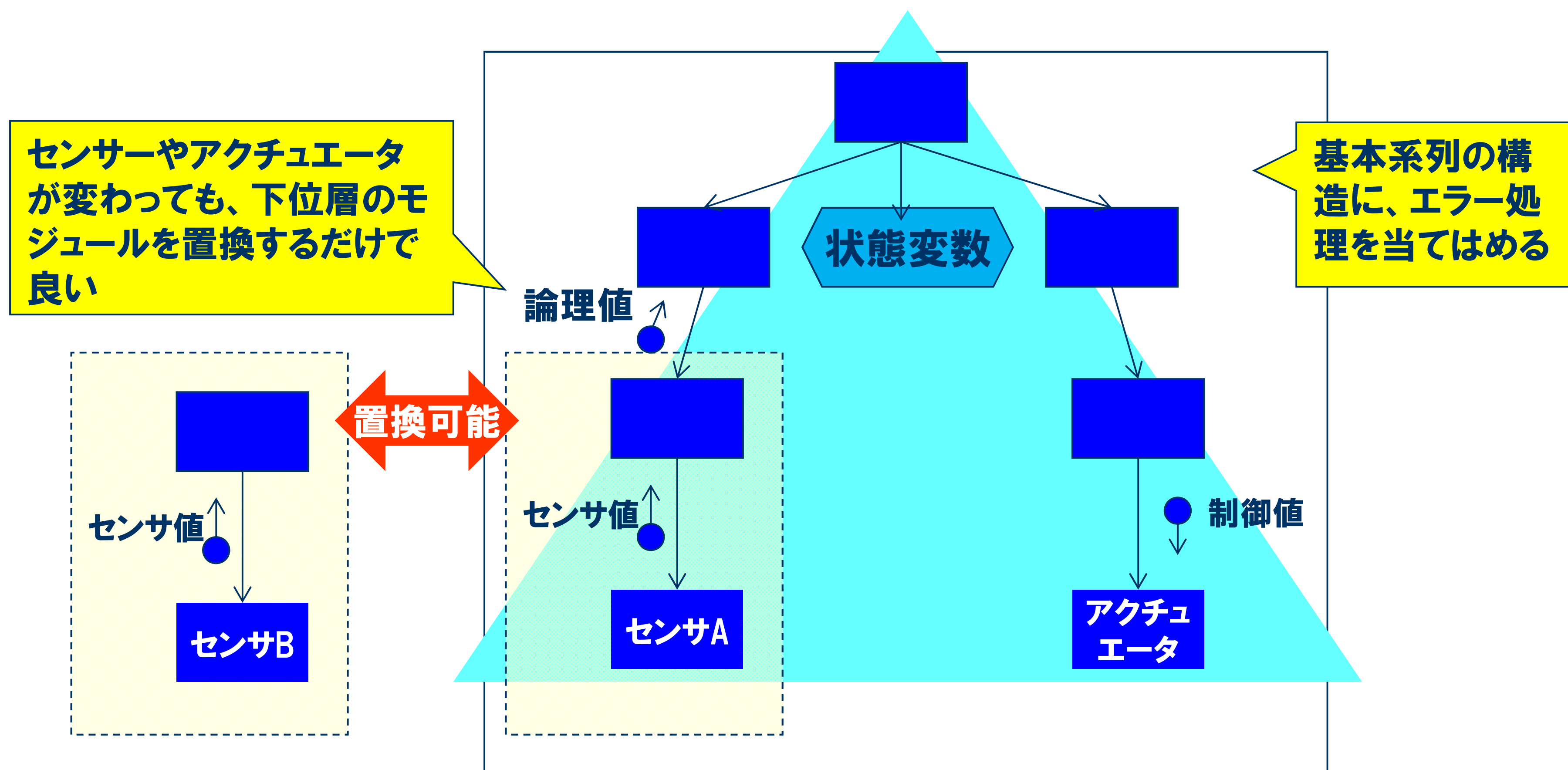
BACKSLASH DESIGN

2. 末広がり vs 構造化

Copyright BACKSLASH DESIGN Co., Ltd.

設計すると：構造化プログラム

- 論理的な階層化設計
  - 上位層で、状態変数を定義
  - 下位層で、センサーやアクチュエータのプリミティブな制御のみ



Copyright BACKSLASH DESIGN Co., Ltd.

プログラミングスタイルの違い

BACKSLASH DESIGN

	アセンブラ的C	モジュール的C
中心ファイル	実装ファイル [.c] 中心	ヘッダファイル [.h] 中心
設計の主題	.実装ファイルで、関数呼び出しのインタフェースを設計	ヘッダファイルで、ファイル間のインタフェースを設計
ヘッダファイルの位置づけ	共通の定義	実装ファイルとペア 公開する関数と変数を定義
変数	グローバル	ファイル内にカプセル化
main関数	サイクリック実行を制御（mainループ）	トップのBOSSモジュールを起動するだけ

動解析ファースト

静解析ファースト

Copyright BACKSLASH DESIGN Co., Ltd.

BACKSLASH DESIGN

3. 静解析ファースト：データ構造の設計

Copyright BACKSLASH DESIGN Co., Ltd.

データ構造設計していない極端な例

- 0～255個の定数の取得
  - 256個のcase文
- 「配列」使おうよ
  - 256個のテーブル
  - 1行で変換

```
// ガロア体上の置換表で変換
public int convert_galois(int a)
{
        switch(a)   {
                // 0行
                case 0x00:a = 0x63;break;
                case 0x01:a = 0x7c;break;
                case 0x02:a = 0x77;break;
                case 0x03:a = 0x7b;break;
                case 0x04:a = 0xf2;break;
                ・・・・・・・・・・・・・・・・・・・・
                ・・・・・・・・・・・・・・・・・・・・
                case 0xfe:a = 0xbb;break;
                case 0xff:a = 0x16;break;
        }
        return a;
}
```

```
// ガロア体上の置換表で変換
public int convert_galois(int a)
{
        a = array[a];
         return a;
}
```

データ構造とアルゴリズム：
データ構造を設計すれば、
制御ロジックはシンプルになる

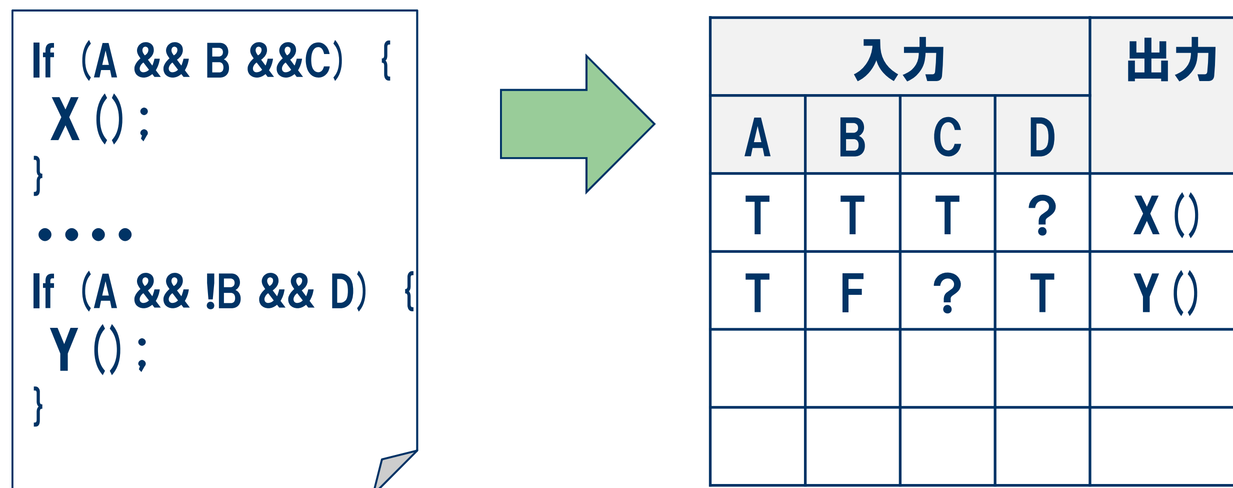
Copyright BACKSLASH DESIGN Co., Ltd.

フラグのグループ化、そして、名前付け

BACKSLASH DESIGN

- プリミティブなフラグを括る
  - 状態変数や決定表という設計ができる

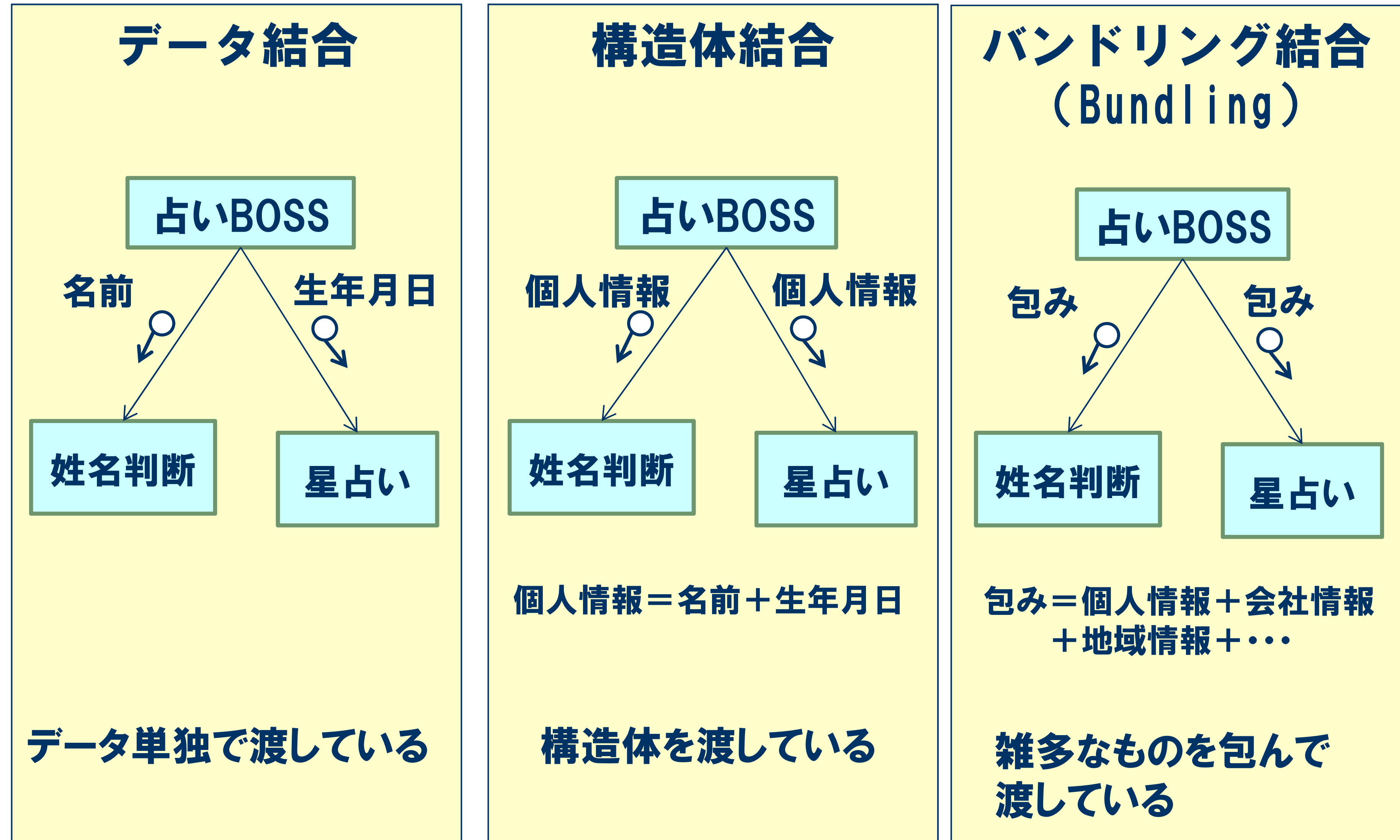
- 決定表
  - 一緒に判断している条件を並べてグループ化する
  - 決定表を作り、決定表に名前を付ける



入力				出力
A	B	C	D	
T	T	T	?	X()
T	F	?	T	Y()

Copyright BACKSLASH DESIGN Co., Ltd.

構造体で括る、かつ、必要なものだけ渡す



Copyright BACKSLASH DESIGN Co., Ltd.

設計はデータ辞書で

- 3つの基本構造は、連結、選択、繰り返し

構造	説明	表記法	C言語	活用例
連結	複数の要素を一つにまとめる	A=B+C	構造体	同じ目的のメンバを集めた構造体
選択	複数の取りうる値がある	A= [B|C]	列挙型	状態変数、モード変数コマンド種別
繰り返し	要素が複数個存在する	A=3 {B} 10	配列	変換テーブル

Copyright BACKSLASH DESIGN Co., Ltd.

BACKSLASH DESIGN

4. 設計図から考えよう

Copyright BACKSLASH DESIGN Co., Ltd.

コード中心と設計中心

	コード中心	設計中心
プログラム形状	末広がり	構造的
プログラム規模	一人で把握できる規模まで （アセンブラ的規模）	スケーラブル （一人規模から複数人規模まで）
プログラム理解	属人的 （作った人以外は理解が困難）	他の人でも理解しやすい （引継ぎや複数人開発が容易）
プログラミングの考え方	アセンブラ的C	モジュール的C
設計の考え方	動解析ファースト （入力イベントと複数のフラグを判断して処理が連なる）	静解析ファースト （論理構造があり、上位モジュールが下位モジュールを呼び出す）
生産性	低い。 ソースコードの動きを追いかけて、変更箇所を特定する	高い。 図面で変更箇所を特定する
品質	不安定。 修正の都度、アドホックに検索して影響範囲を特定する	安定。 図面で影響範囲を限定する

Copyright BACKSLASH DESIGN Co., Ltd.

「末広がり」から「構造化」へ移行するには

BACKSLASH DESIGN

- 論理的なモジュール構造を設計する
  - 抽象化と概念化で、上位層のモジュールを発見する
  - 正常系のシナリオで、安定した論理構造を作る
    - エラー系で構造を作るといびつになり、長持ちしない
- 状態遷移の設計をする
  - フラグではなく状態変数を作る
- エラーの伝播ルートを設計する
  - 上位⇔中間⇔下位の伝播ルールを決める

Copyright BACKSLASH DESIGN Co., Ltd.