パネル

# MEC(Multi-access Edge Commuting)

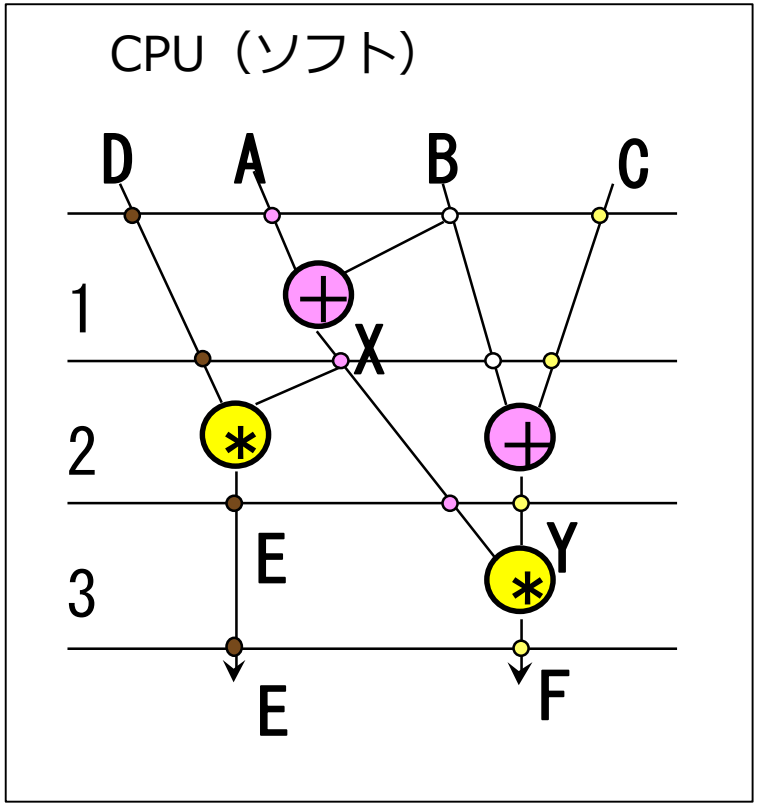
# 複数FPGAからなるシステム と HLS（高位合成）

NEC（日本電気)
若林一敏、中村寿彦、高橋渡、酒井完

# 目次

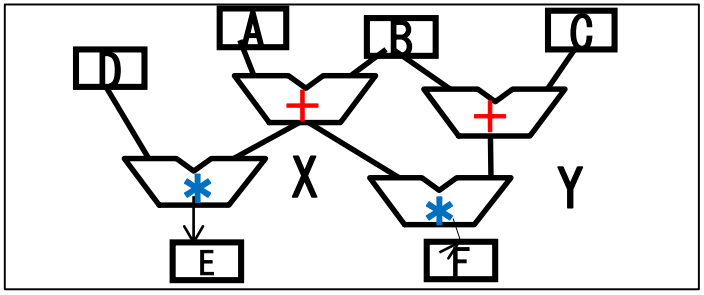
- １．高位合成の出す回路（FSMD)は、複数FPGAからなるボード（FICボード）向き
- ２．FICボードの設計環境として必要なこと

# 高位合成とは　C/C++記述→RTL　(→FPGA)

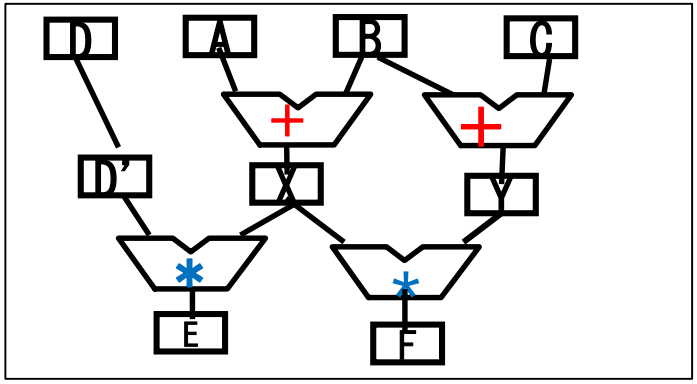



HW（回路）

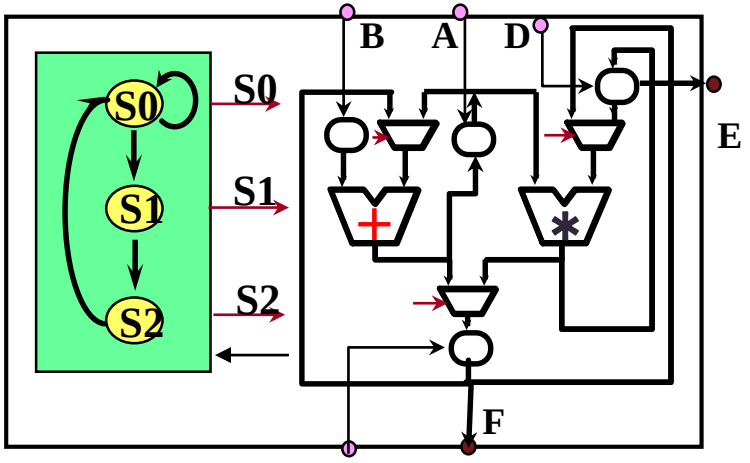
**1サイクル**
**並列回路**




DII=1の
**パイプライン**
**レイテンシ2**



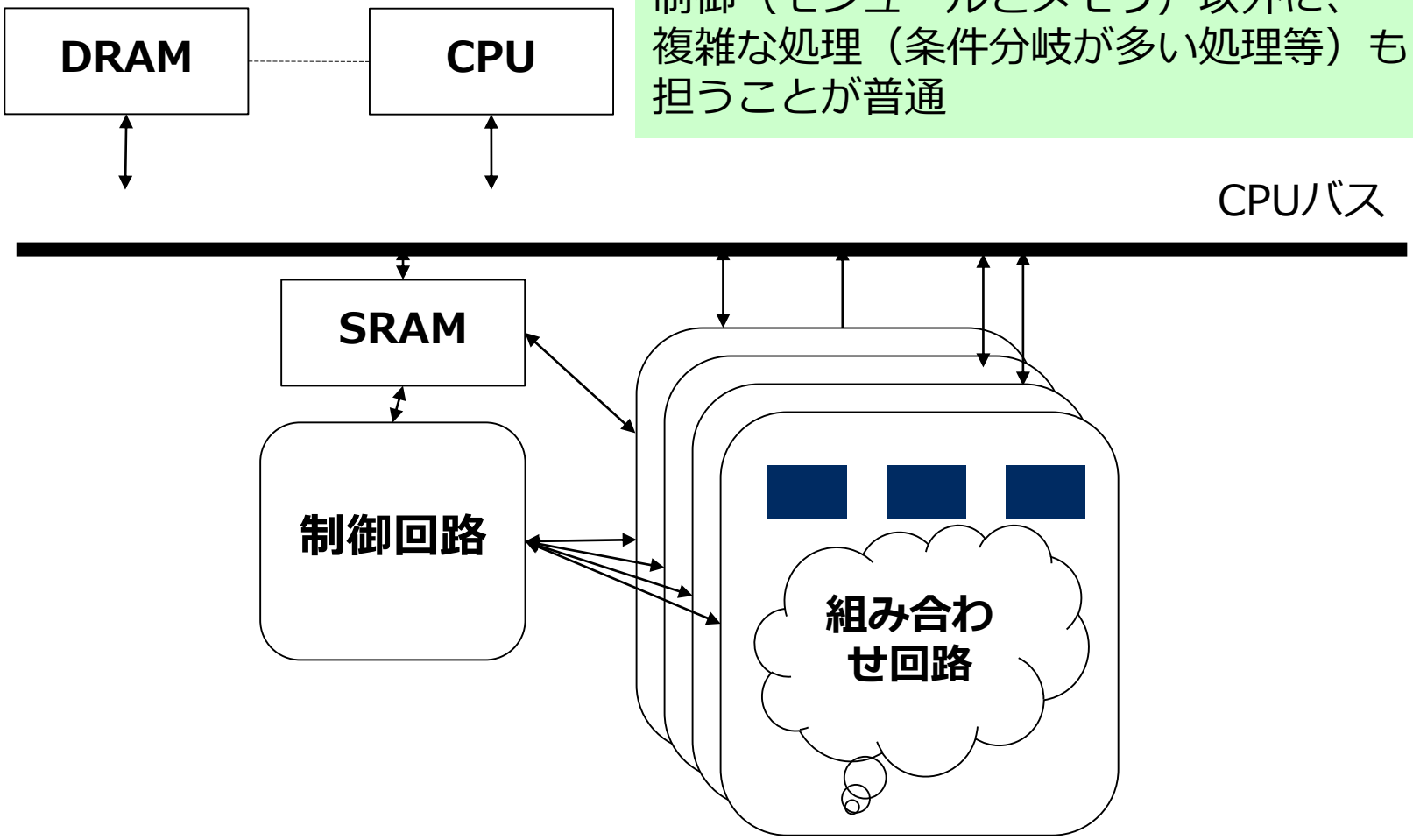

**3サイクル**
**順序回路**
DII=3の
**パイプライン**



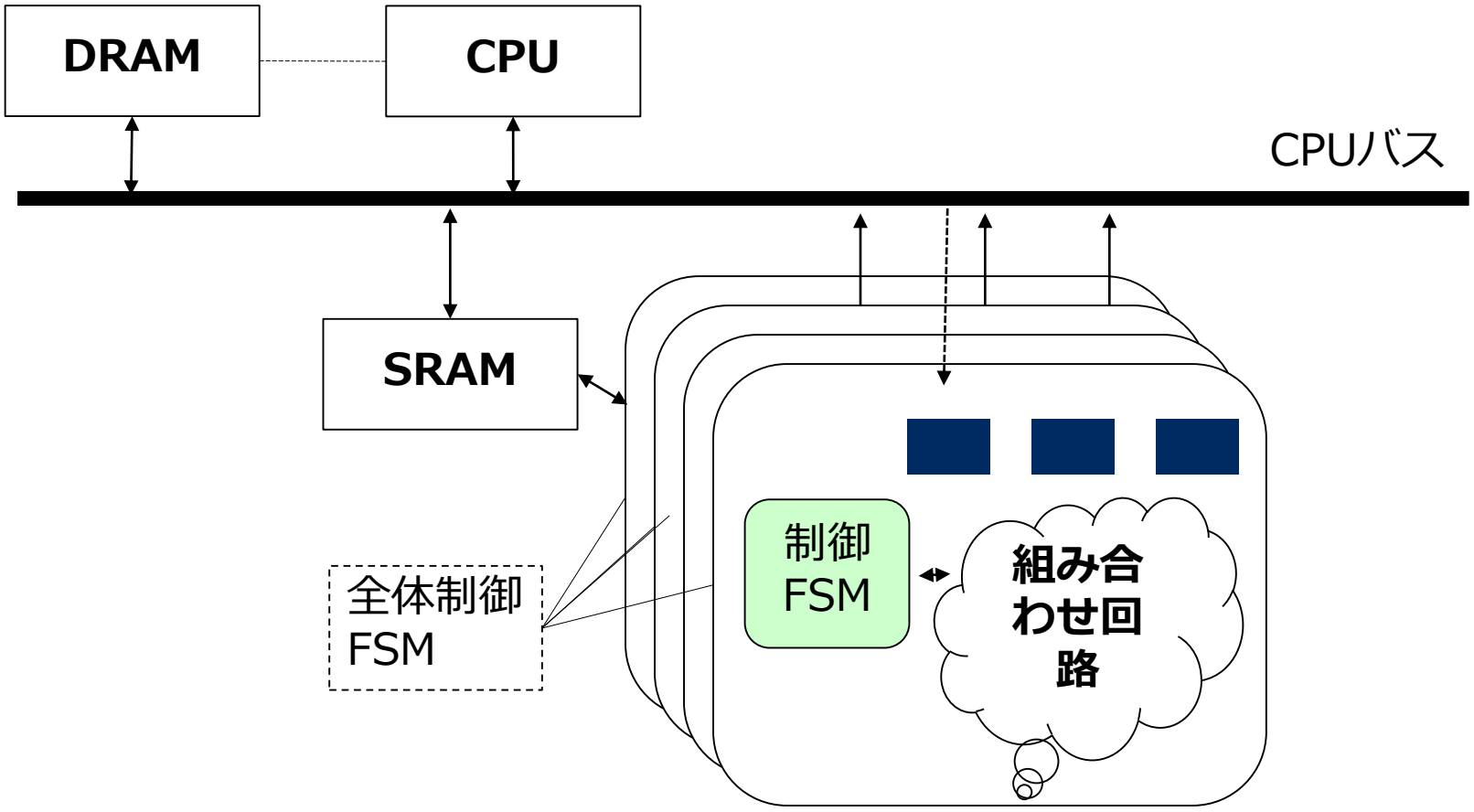

MECでは、
FSM+Dが活躍できる

# RTL設計と高位合成のイメージの違い（RTL設計）

制御（モジュールとメモリ）以外に、
複雑な処理（条件分岐が多い処理等）も
担うことが普通



# RTL設計と高位合成のイメージの違い（高位合成回路）



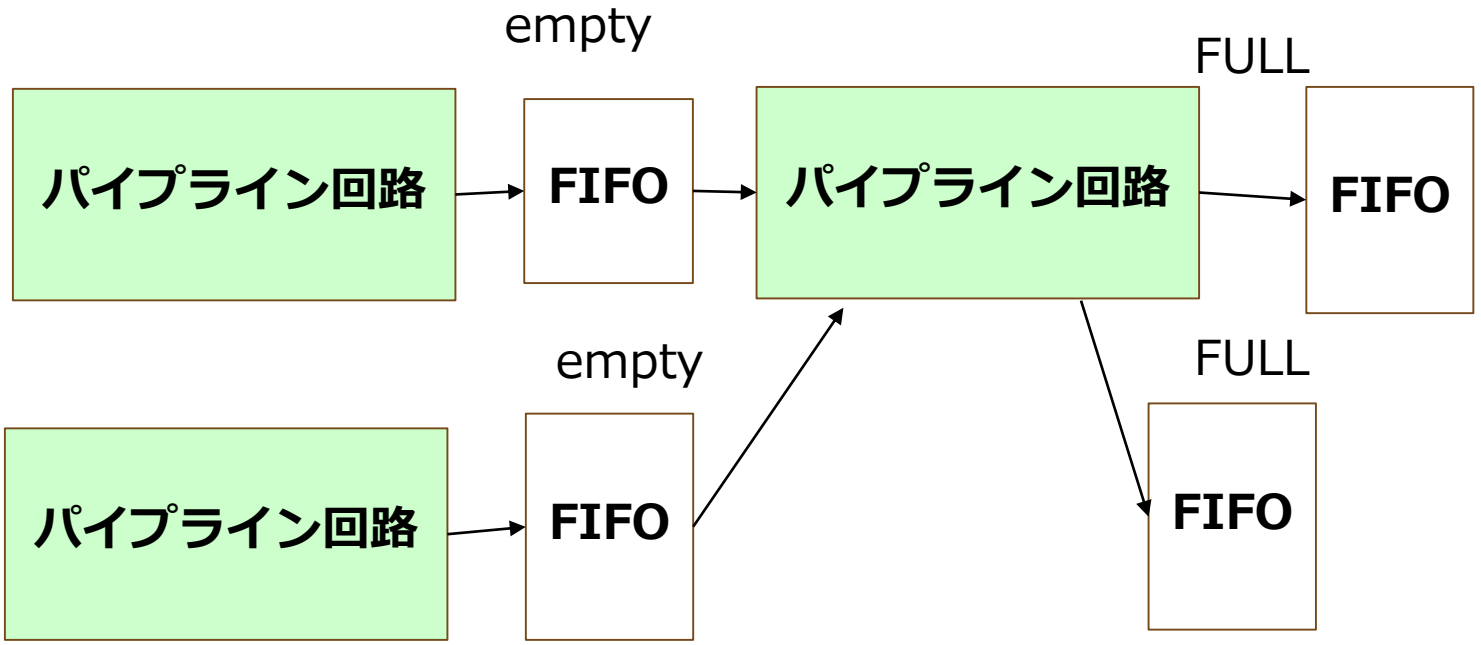

各モジュールが制御をもって独立に動作可能なので、MECに向いている

# 高位合成はストール制御は自動生成（C上に制御の記述無し）

パイプライン回路の入力、出力につながるFIFO

入力FIFOのEMPTY、出力FIFOのFULLによる
パイプラインストール制御は自動生成
・入力が止まっても、フラッシュ動作も自動生成

→C設計、高位合成は、MECに向いている



# 目次

1．高位合成の出す回路（FSMD)は、MEC向き

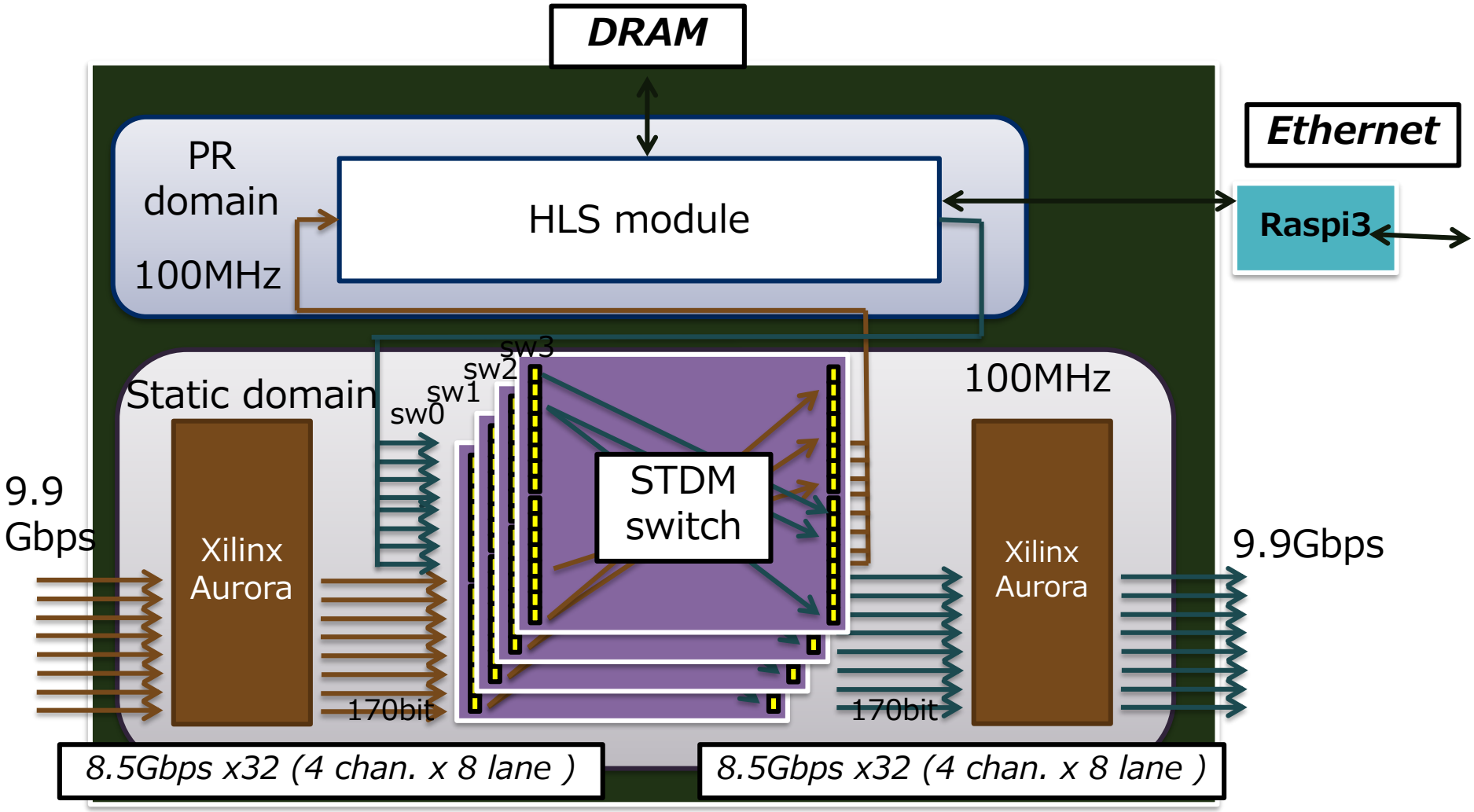
➡ 2．FICボードの設計環境

■ 複数のFPGAを自動で使えるようにしたい
- 通信部分の自動合成
- 時分割多重で、端子数を削減

■ 普通のC/C++から自動並列化したい
- SystemCでは、プロセスレベルの並列性は人手指定
  （sc_moule, sc_cthread等で指定）

■ 本プロジェクト新FPGAへの対応
  演算器の遅延・面積等のライブラリ整備等

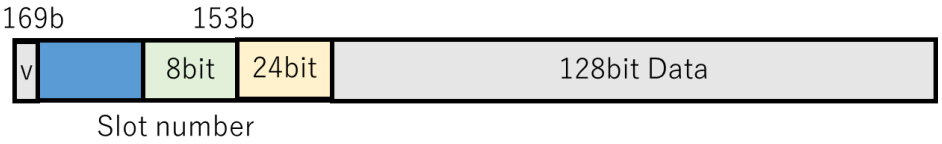
# FICの構成



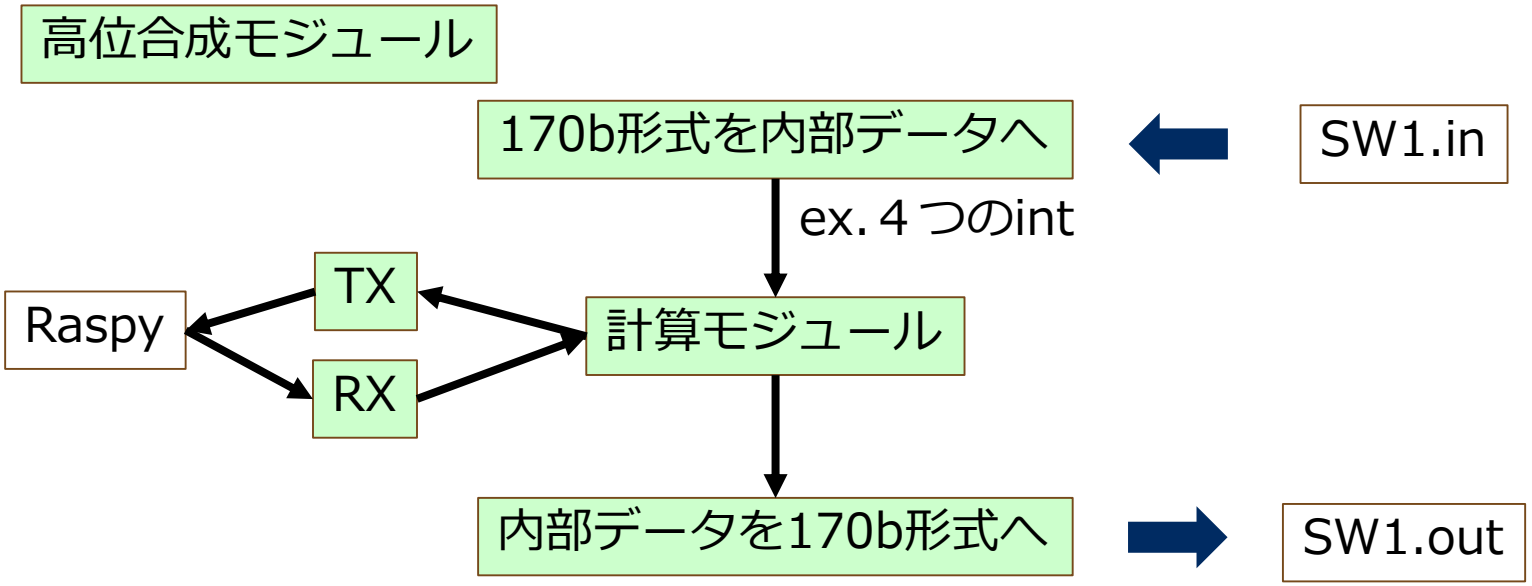

現状は、ボード間通信はHLSモジュールは担当しない（Aurora IPで通信）
まずは、AXI streamを基本で合成

# 接続I/Fの自動（半自動）生成

Auroraの形式



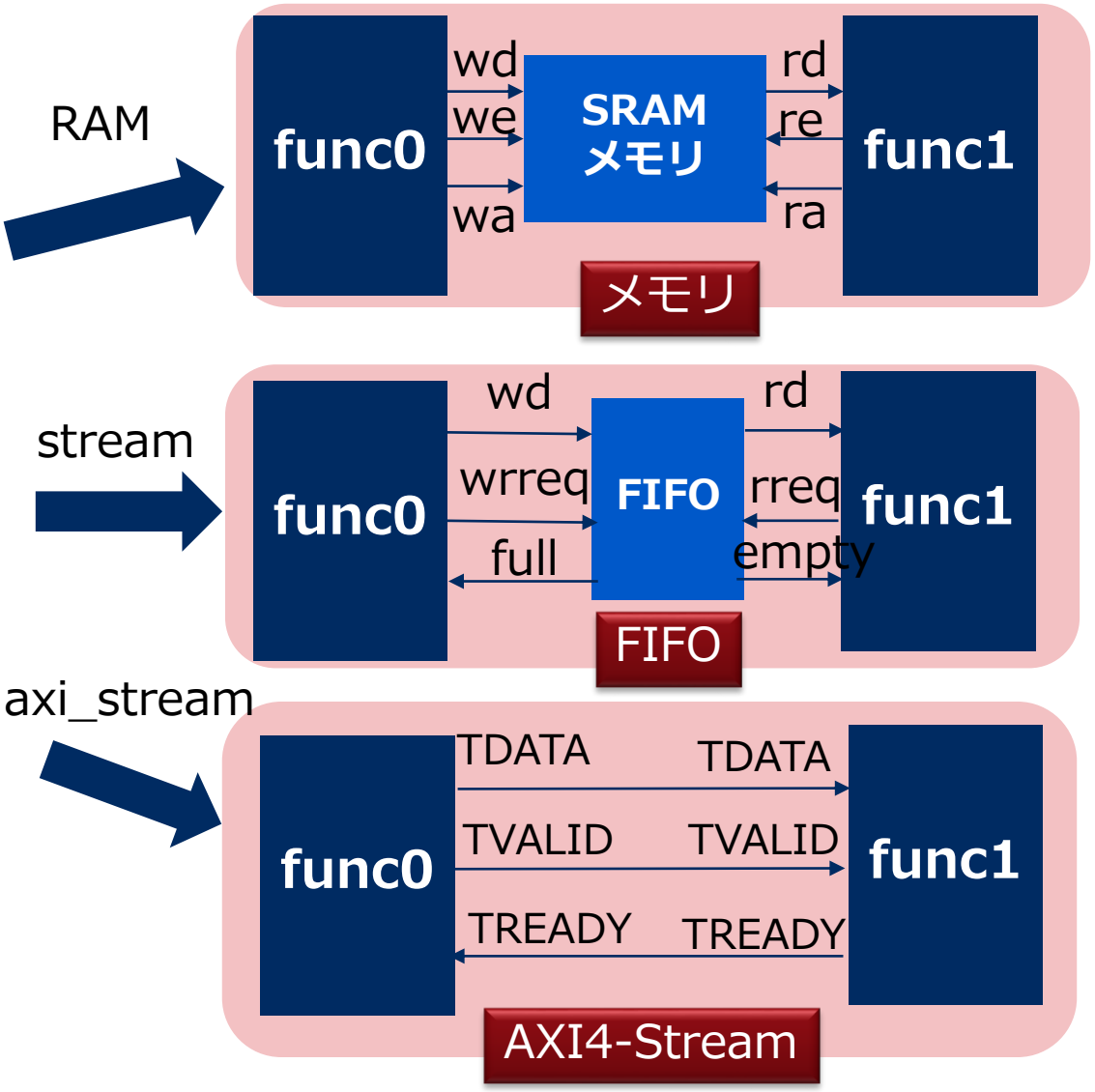

ボード間通信は170b形式→計算モジュール内のデータを170ｂ形式に変換



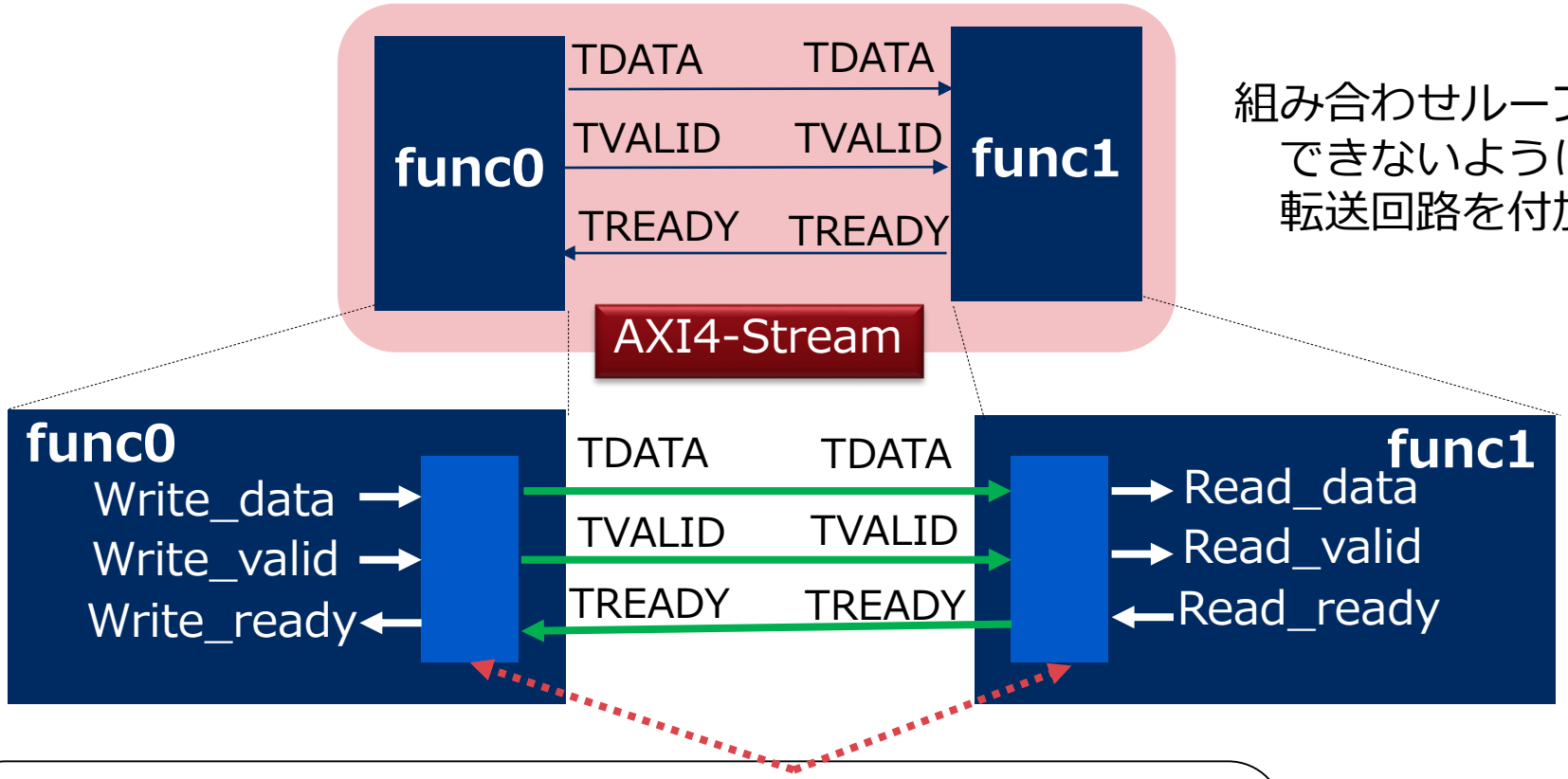

合成用クラスを作成する等を行い、計算モジュールを設計するだけで
SWやRaspyとのI/F回路やデータをパック、アンパックする回路を自動生成

# 順次アクセスの配列をAXI4-Streamなど様々なIFに変更可能

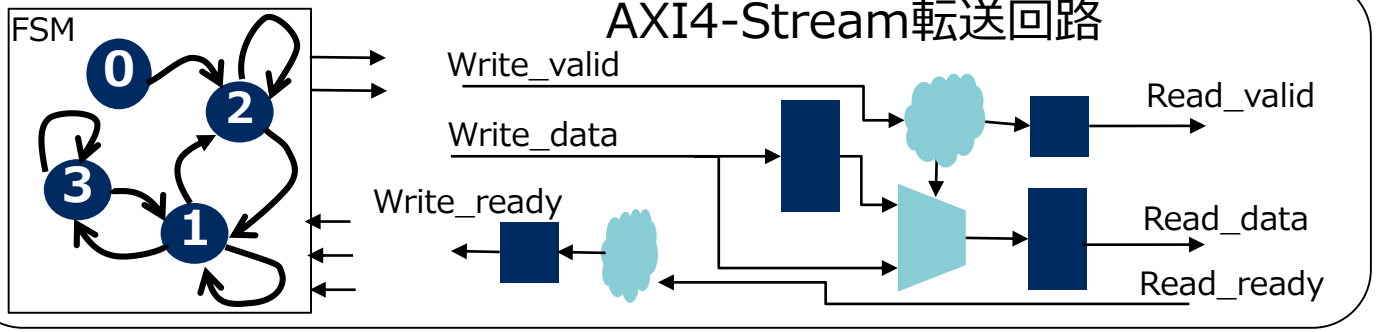
```
//cyber folding=1
func0()
for (int i = 0; i < N; i++) {
        ary[i] = x[i]+…….;
}
//cyber folding=1
func1()
for (int I = 0; I < N; i++) {
        y[i] = ary[i];
 }

//cyber process_pipeline
void func_top ( ){
        sc_uint<8> ary[N];
        func0 (in0, ary);
        func1 (ary, out0);
}
```



# AXI4-Stream対応の転送回路を自動生成



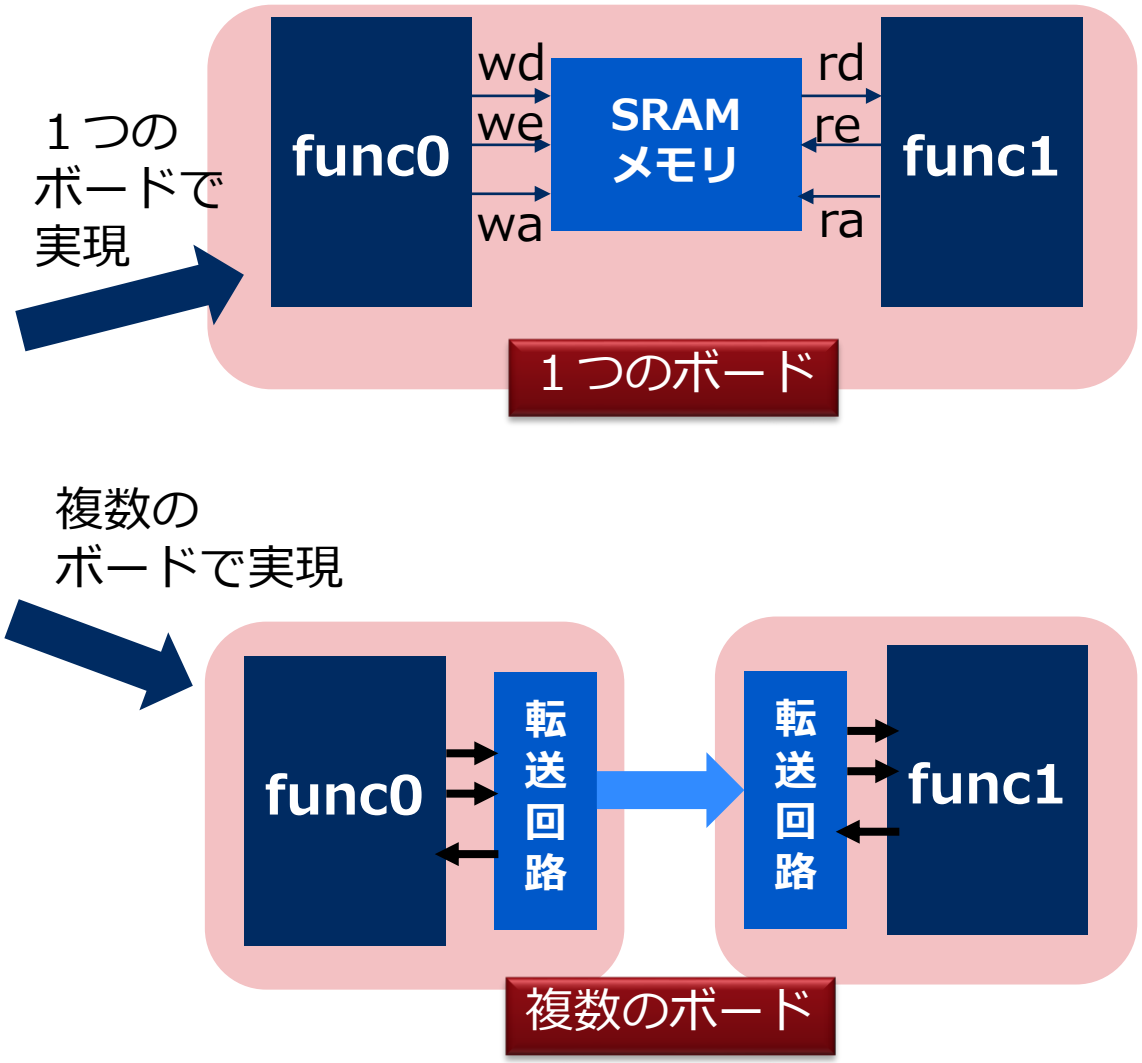

組み合わせループが
できないように
転送回路を付加



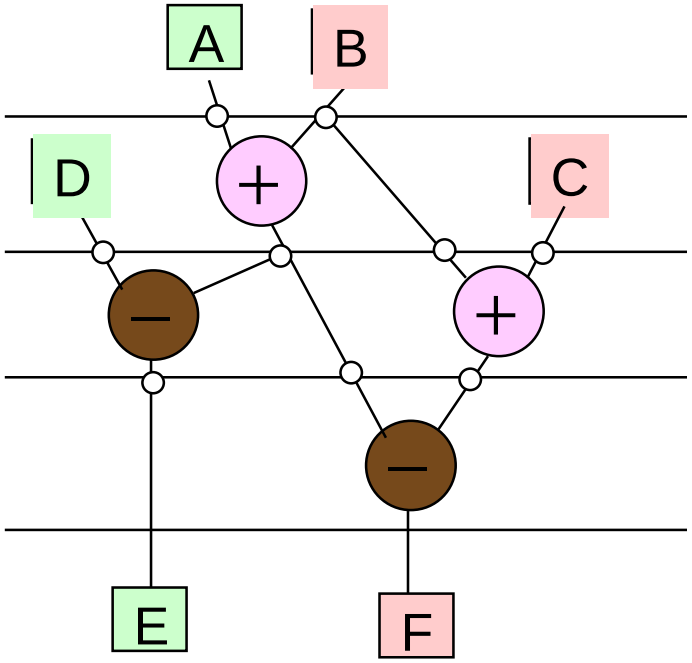

転送回路の出力は
レジスタにする
ことでループを回避

# 複数ボードへの分割：転送回路を自動生成

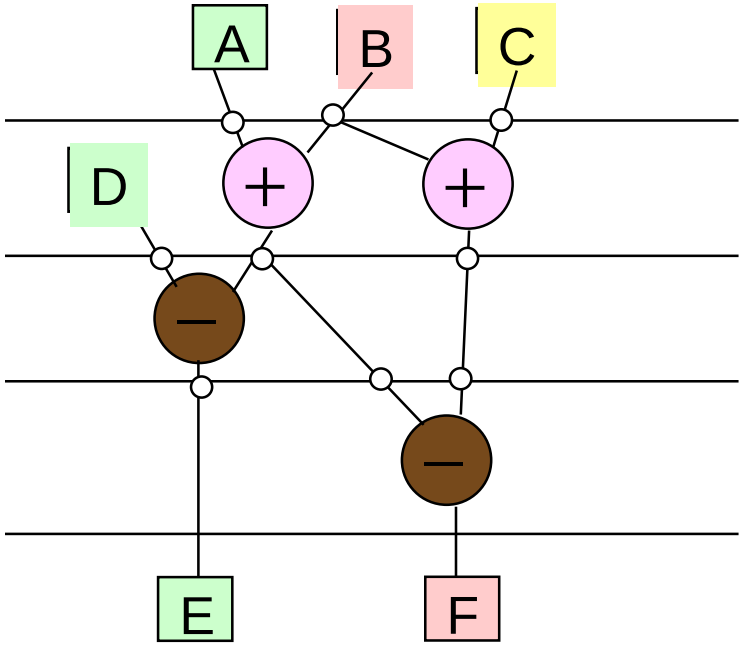
```
//cyber folding=1
func0()
for (int i = 0; i < N; i++) {
        ary[i] = x[i]+…….;
}
//cyber folding=1
func1()
for (int I = 0; I < N; i++) {
        y[i] = ary[i];
 }

//cyber process_pipeline
void func_top ( ){
        sc_uint<8> ary[N];
        func0 (in0, ary);
        func1 (ary, out0);
}
```



# MECの時分割通信への対応　→ポートシェア機能拡張




**2ポートで合成**




**３ポートで合成**

上記のように、論理的なin,out端子を、少数の物理ポートを時分割で利用するように合成可能
→　MECの時分割通信に利用する

# 当初　目標

- 複数のFPGAを自動で使えるようにしたい
  - 通信部分の自動合成
  - 時分割多重で、端子数を削減

- 普通のC/C++から自動並列化したい
  - プロセスレベル並列の指定
    SystemCでは、プロセスレベルの並列性は人手指定
    （sc_moule, sc_cthread等で指定）

- 自動階層設計（FPGA内の垂直、水平並列化）

# プロセスレベルのパイプラインの記述方（C/C++の場合）

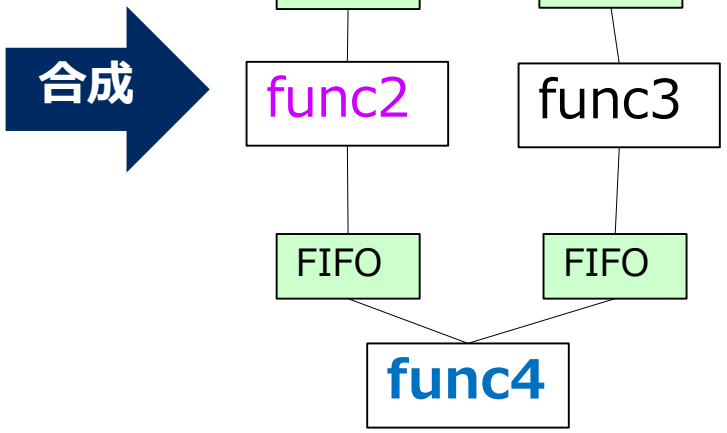
プロセス（モジュール）レベルのパイプライン化

C/C++には、モジュールがどのように接続するかという「構造記述力」がない。→　合成用コメントで表現。

以下の、process_pipelineは、関数を並列処理し、間をFIFOで接続する意味

```
//cyber process_pipeline
main(){
   x=input(a);
   z=func1(x,y);
   s=func2(z,c);
   t=func3(z,…);
   func4(s,t,…)
}
```

左のC記述を、下の構造と解釈




SystemCの場合は、sc_moduleで任意の接続構造が書ける

func1,2,3,4,は、パイプライン合成
それらを、パイプラインでつなげる

# 並列プロセスをC/C++で、どう記述するか？（構造記述）

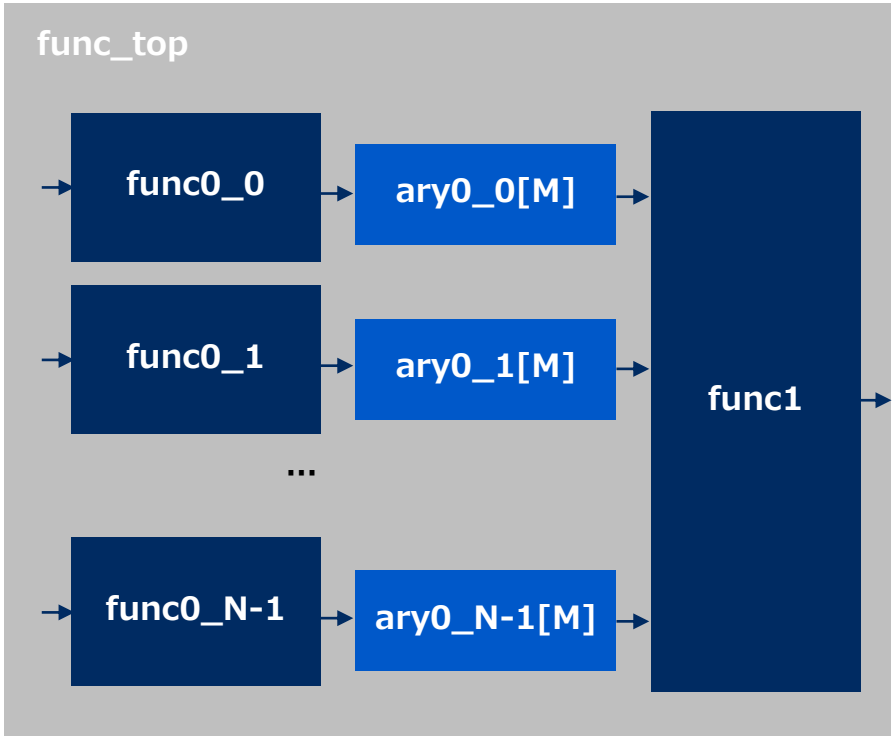
## Cの記述を構造記述として読み取る（合成プラグマとセット）

上のC記述を、#process_pipelineという合成コメントで、
インスタンスがN個ある回路を合成（現状）→

```
/* Cyber func = process_pipeline */
void func_top (...) {

  sc_uint <16> ary0[N][M] /* Cyber expand_dim=2*/;

  /* Cyber unroll_times = all */
  for (int i = 0; i < N; i++) {
     func0 <A_BIT, B_BIT, ... > (..., ary0[ i ], ...);
  }

  func1 <A_BIT, B_BIT, ... > (..., ary0, ...);
}
```

**配列次元展開**
**：配列の並列アクセス**

**プロセス関数ループ展開**
**：モジュールの配列化**



# 大きなプログラムから、複数プロセスを合成

どう分割するか？

１）高位合成に適したプロファイラ（GPROF等の高位合成用）を開発し、何分割すれば、所望の性能をだせそうかを予測

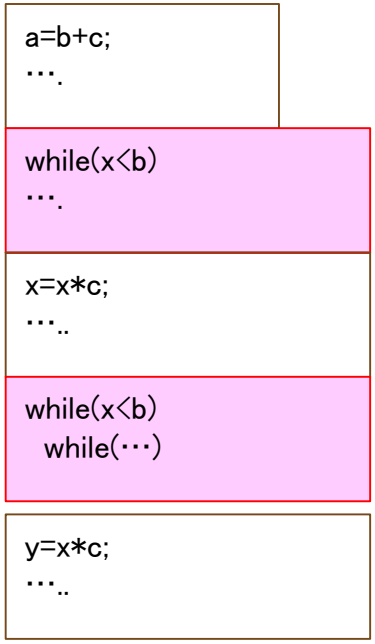
２）１）で指定された分割で、高位合成

３）サイクル精度モデルを生成して、サイクル性能をチェック

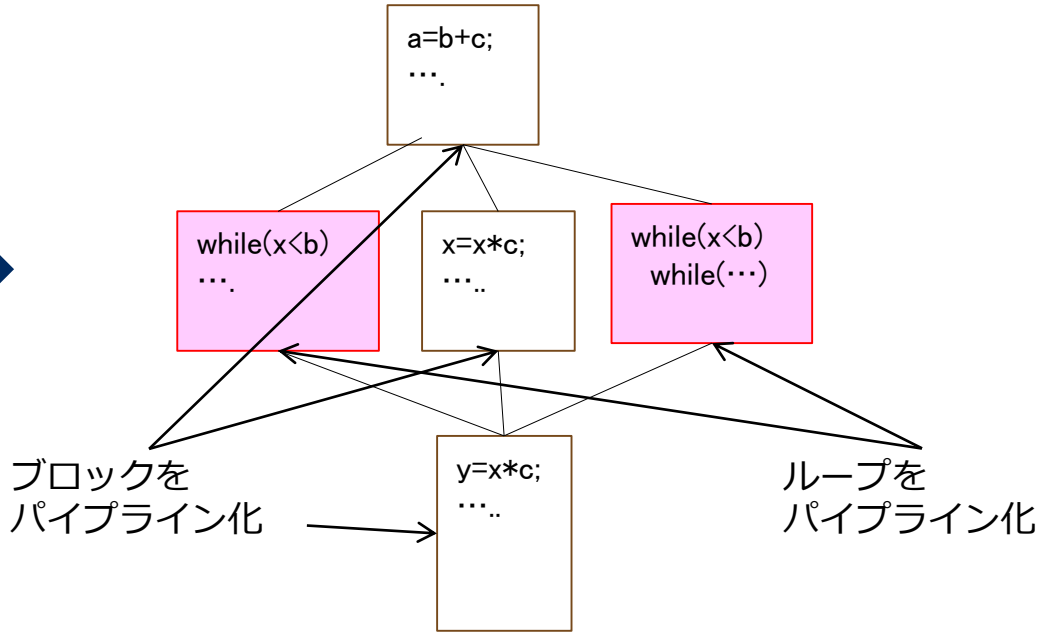
４）２，３を繰り返す（半自動化を目指す）

# 大きなプログラムを自動分割、多数のFPGAへ

C/C++プログラム




合成

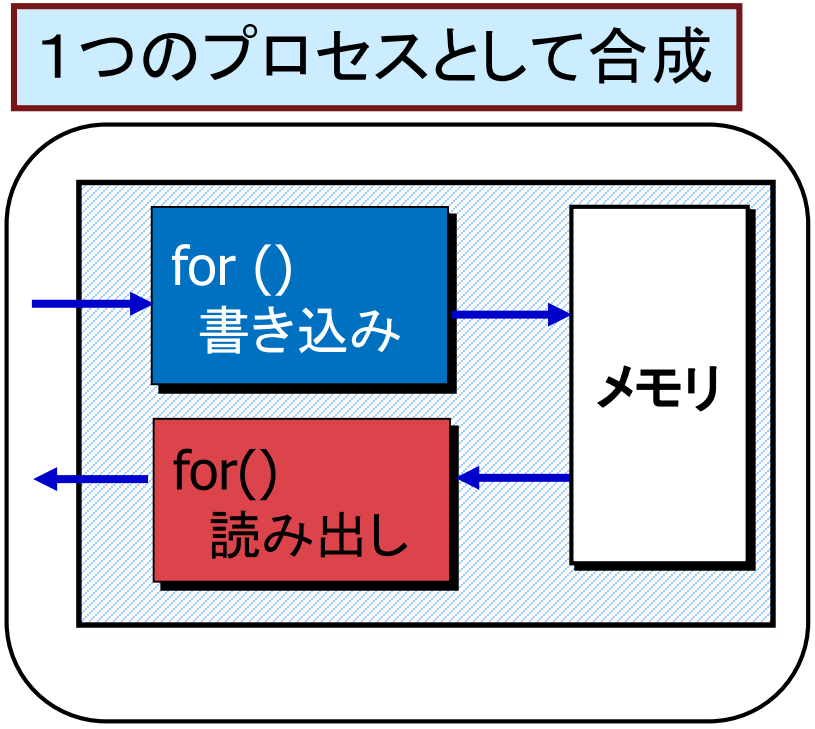



関数、ループ等をパイプラン合成、
それらを、パイプラン動作
→多数のFPGAで　高速実行

→FPGA間は、時分割多重やシリアル化等

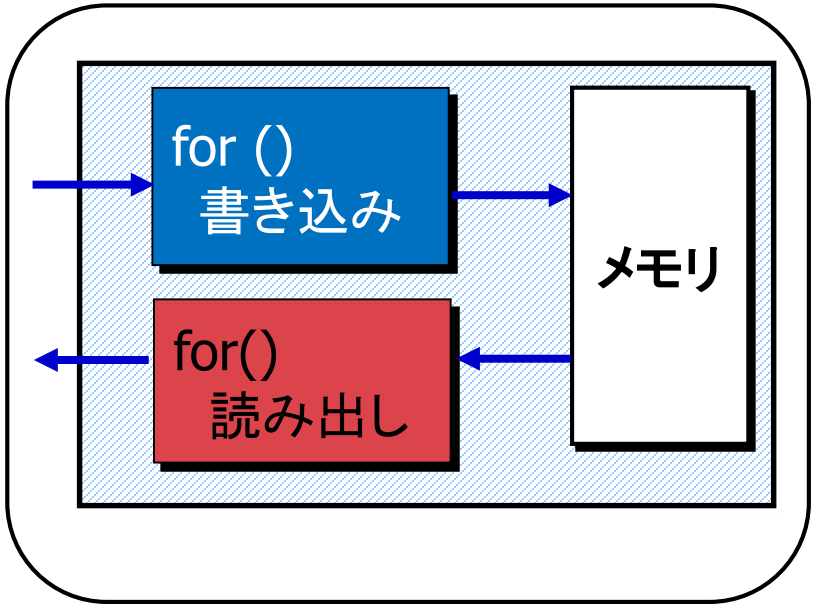
# メモリアクセスのさらなる高速化(メモリ→FIFO変換)

```
//cyber folding=1
func0()
for (int i = 0; i < N; i++) {
        ary[i] = x[i]+…….;
}
//cyber folding=1
func1()
for (int I = 0; I < N; i++) {
        y[i] = ary[i];
 }
                    ← 並列性を人手指定
//cyber process_pipeline
void func_top ( ){
        sc_uint<8> ary[N];
        func0 (in0, ary);
        func1 (ary, out0);
}
```



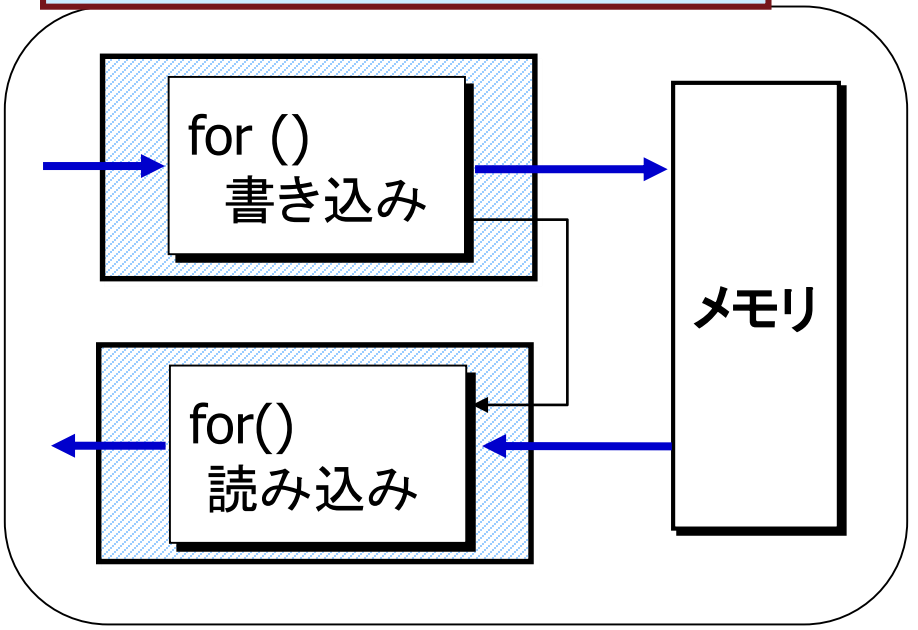
# プロセスレベルの並列性：3種類の例

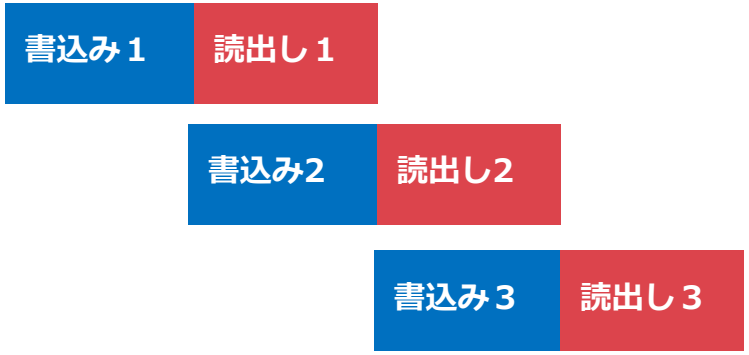
## 1つのプロセスとして合成





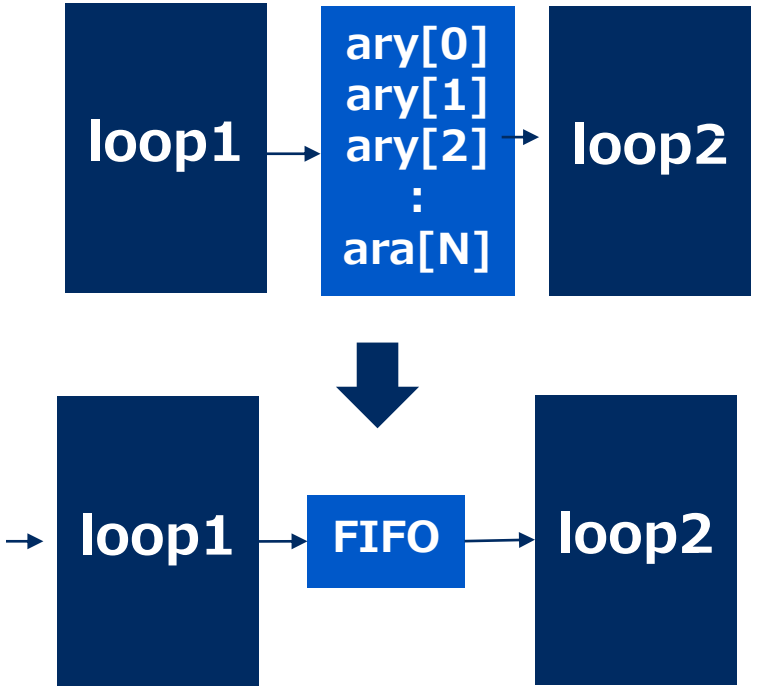
## 2つのプロセスとして合成



# メモリアクセスのさらなる高速化(メモリ→FIFO変換)

```
//cyber folding=1
func0()
for (int i = 0; i < N; i++) {
        ary[i] = x[i]+…….;
}
//cyber folding=1
func1()
for (int I = 0; I < N; i++) {
        y[i] = ary[i];
 }

                    ― 並列性を人手指定
//cyber process_pipeline
void func_top ( ){
        sc_uint<8> ary[N];
        func0 (in0, ary);
        func1 (ary, out0);
}
```




# ランダムアクセスの場合は、ピンポンバッファ生成

## メモリを共有するプロセス間で、プロセスレベルのパイプライン化が可能

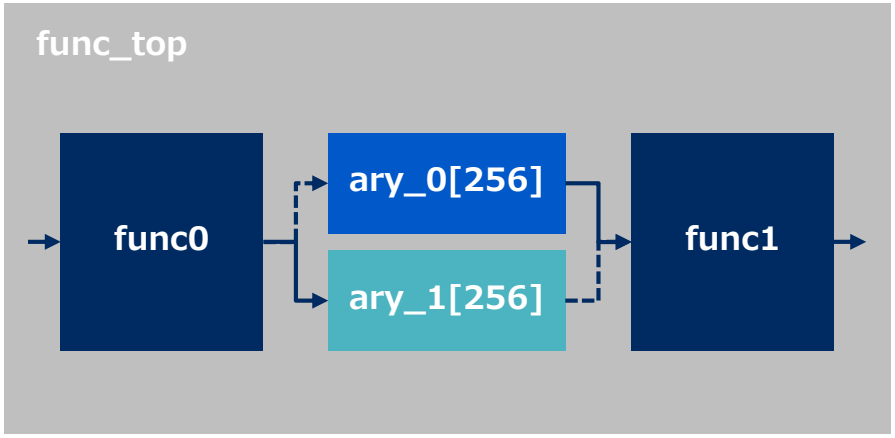
・C 記述

```
/* Cyber func = process_pipeline */
void func_top (...) {

   sc_uint <32> ary [256] /* Cyber pingpong = YES */;

   func0 (..., ary, ...);

   func1 (..., ary, ...);
}
```

ary[] へ Write アクセス

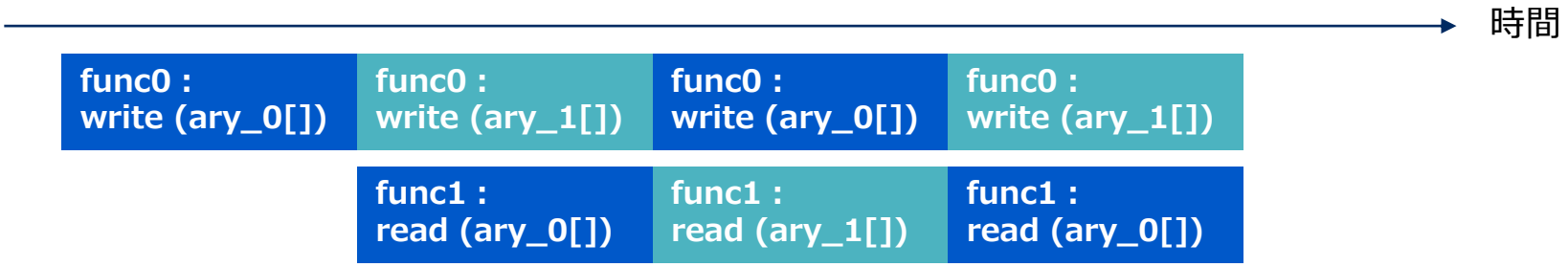
ary[] へ Read アクセス

・回路イメージ




メモリの自動2面化、制御論理自動生成

・動作イメージ

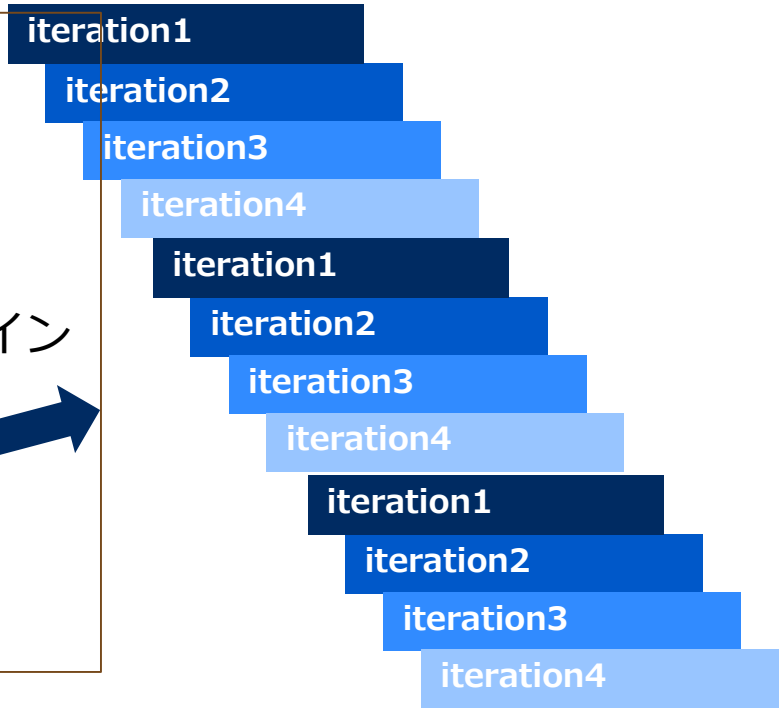


# 2重ループのパイプライン（単純なネスト構造の場合）

内側ループだけをパイプライン化

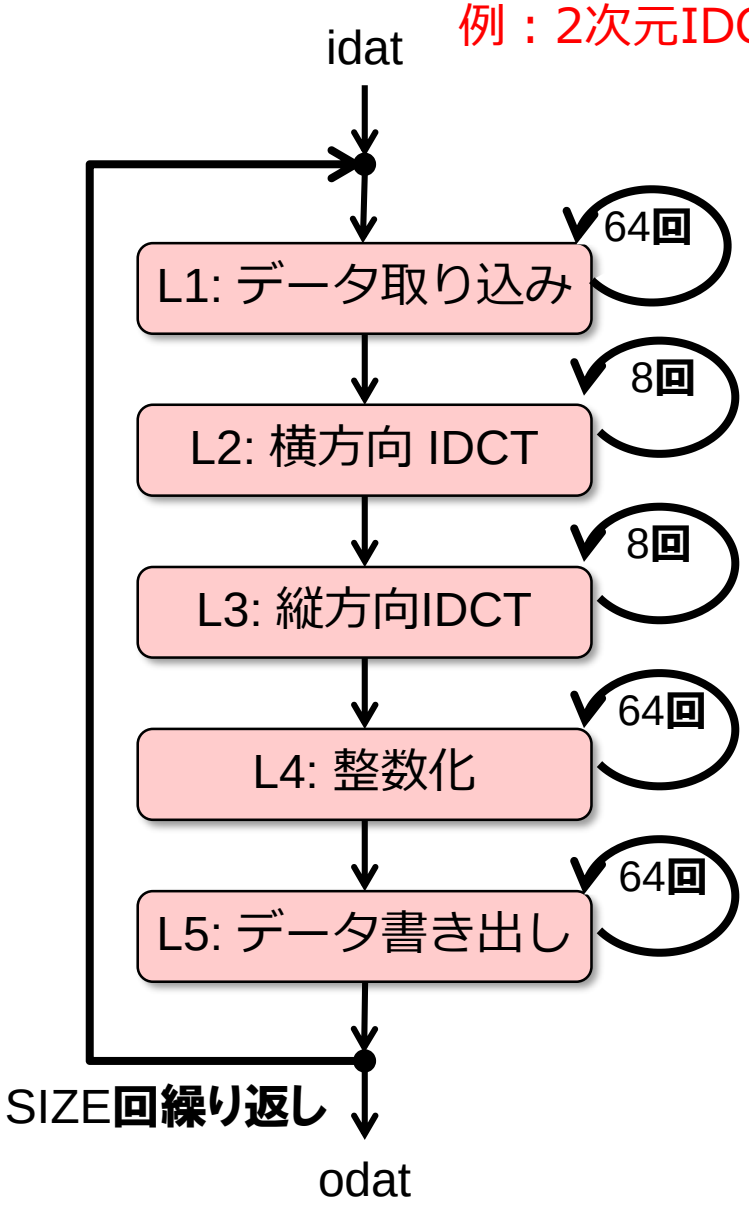

ネスト解消し、1重ループにしてパイプラン化

```
//単純なネストループ
for(x=0..5)
  for (y=0..8)//nest展開on/off
       f1(x,y);

//ネストを自動解消して、パイプライン
for (i=0..40) 1重ループ化

   x=
   y=
   f1(x,y);
```



# 一般的な2重ネストループ

例：2次元IDCT




```
L0:for(unsigned int j=0;j<SIZE;j++){
      bank = (j%2) ;
      iary_s = iary[bank]; c2r_s = c2r[bank];
      r2f_s = r2f[bank]; oary_s = oary[bank];
    L1:for(unsinged char i=0;i<64;i++){
         iary_s[i] = idat.get();  }
    L2:for(unsinged char i=0;i<8;i++){
         column(iary_s+i, c2r_s+i);}
    L3:for(unsinged char i=0;i<8;i++){
         row(c2r_s+(i*8), r2f_s+(i*8));}
    L4:for(unsinged char i=0;i<64;i++){
         data_t val = r2f_s[i];
         val = ((val<0) ? (val-8) : (val+8))/16;
         oary_s[i] = val;}
    L5:for(unsinged char i=0;i<64;i++){
         odat.put(oary_s[i]);}
}
```

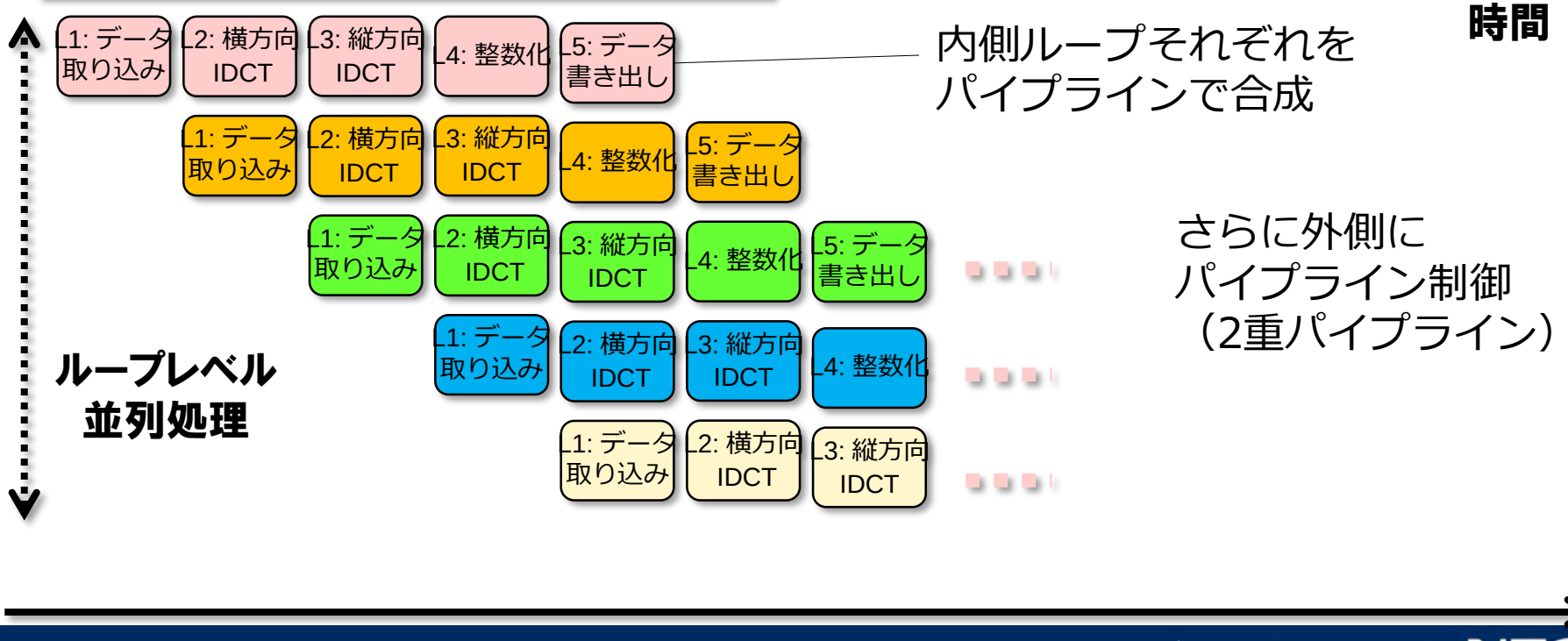
# パイプラインを2階層で合成する

## 逐次実行

L1: データ取り込み | L2: 横方向IDCT | L3: 縦方向IDCT | L4: 整数化 | L5: データ書き出し | L1: データ取り込み | L2: 横方向IDCT | ...

各ループはパイプライン合成、外側が逐次

## ループパイプライン実行



# Orchestrating a brighter world

世界の想いを、未来へつなげる。

未来に向かい、人が生きる、豊かに生きるために欠かせないもの。
それは「安全」「安心」「効率」「公平」という価値が実現された社会です。

NECは、ネットワーク技術とコンピューティング技術をあわせ持つ
類のないインテグレーターとしてリーダーシップを発揮し、
卓越した技術とさまざまな知見やアイデアを融合することで、
世界の国々や地域の人々と協奏しながら、
明るく希望に満ちた暮らしと社会を実現し、未来につなげていきます。


Orchestrating a brighter world
NEC