

# SpecCにおける通信パターンと応用

-Channelを中心にして-

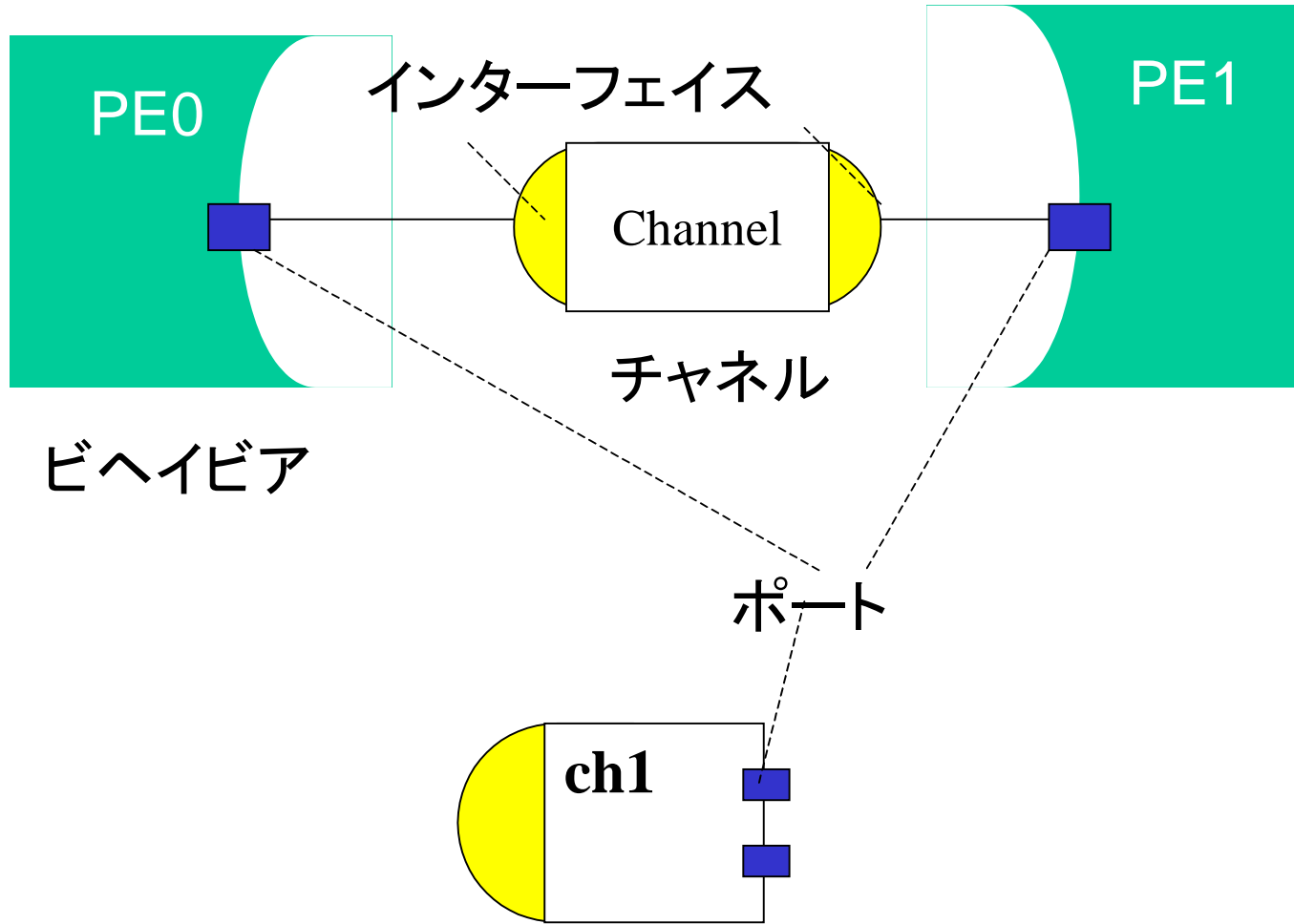
(株)インターデザイン・テクノロジー

岩政, 2001/8/31

- はじめに
- SpecCにおける通信パターン
  - 構造の観点から
  - 通信方法の観点から
- 通信に関する様々な応用
  - 例1: 通信合成
  - 例2: デザイン分割/IOドライバの利用
  - 例3: その他: IP活用、プロトコルスタック
- おわりに

- SpecCの通信(チャンネル)のパターンを整理
  - 構造、通信方法の観点から
- SpecCによる設計のなかでどのように使われるかを示す
  - 通信合成
  - デザイン分割(I/Oドライバ活用)
  - その他

# 記法

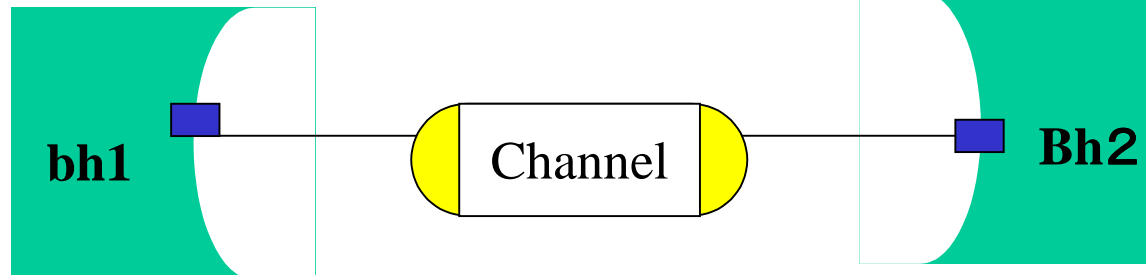


# 構造の視点から



- 単純なチャネル
- 階層チャネル
  - 子チャンネル
  - 子ビヘイビア
- ポート接続するチャネル
- 組合せ

# 単純なチャンネル

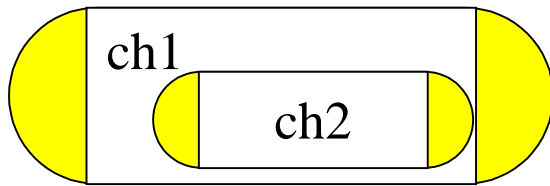


```
Behavior bh1(inout ifw ioch){  
    ioch.write(50);  
}
```

```
Behavior bh2(inout ifw ioch){  
    I = ioch.read();  
}
```

```
channel ch1 implements if1 if2 (void){  
    int val;  
    void write(int I){  
        val=I;  
    }  
    int read(){  
        return val;  
    }  
}
```

# 階層チャンネル1(子Channel)

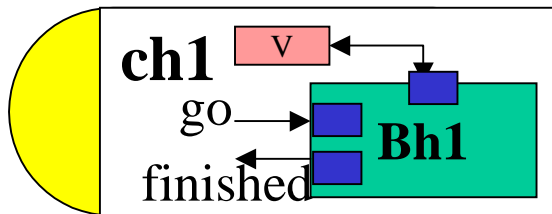


- チャンネルの中にチャンネル
- 抽象レベルの違い

```
channel ch1(void) implements
ifa_write, ifa_read
{
// Child channels
ch2 C0();

void write(int data[3]){
int cnt;
for(cnt=0;cnt<3; cnt++){
wait(ready);
C0.writeInt(data[cnt]);
notify(go);
}
}
```

# 階層チャンネル2(子Behavior)



- チャンネルの中にビヘイビア
- IP活用
- 機能隠蔽(ラッパ)

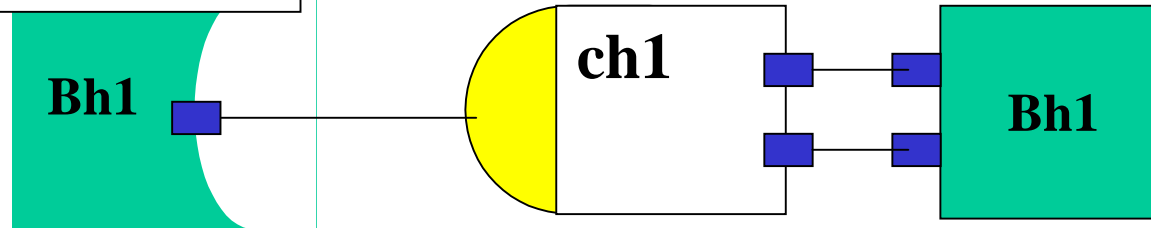
```
channel ch1(void) implements ioif
{
    int v;
    event go, stop;
    bh1 B0(go,stop,v);

    void init(void){
        B0.main();
    }
    void dosomething(int data){
        v = data;
        notify(go);
        wait(stop);
    }
}
```

# チャンネルのポート接続



```
behavior Bh1(inout ioif ch){  
    retu = dosmtg(50);  
}
```

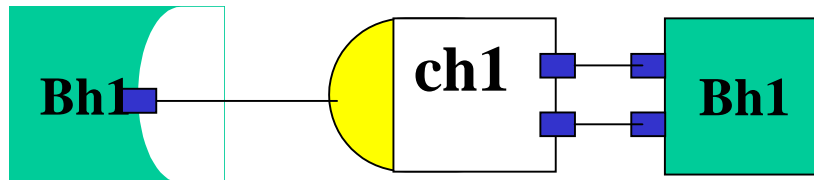


```
channel ch1(out event go, in event finished,  
inout int val) implements ioif
```

```
{  
int dosmtg(int i){  
    val = i;  
    notify go;  
    wait finished;  
    return val;  
}};
```

```
behavior Bh2(inout event go, inout event  
finished, inout int val)  
{  
void main( void ) {  
    wait go;  
    val = val * 20;  
    notify finished;  
}};
```

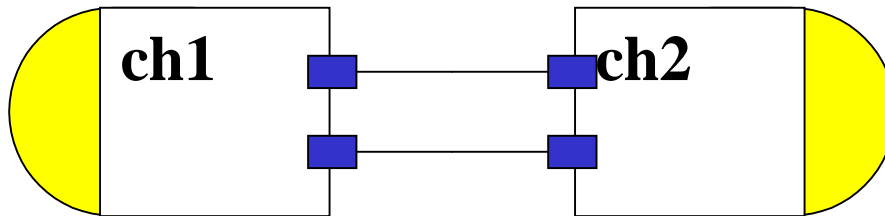
# トップレベルヘイビア



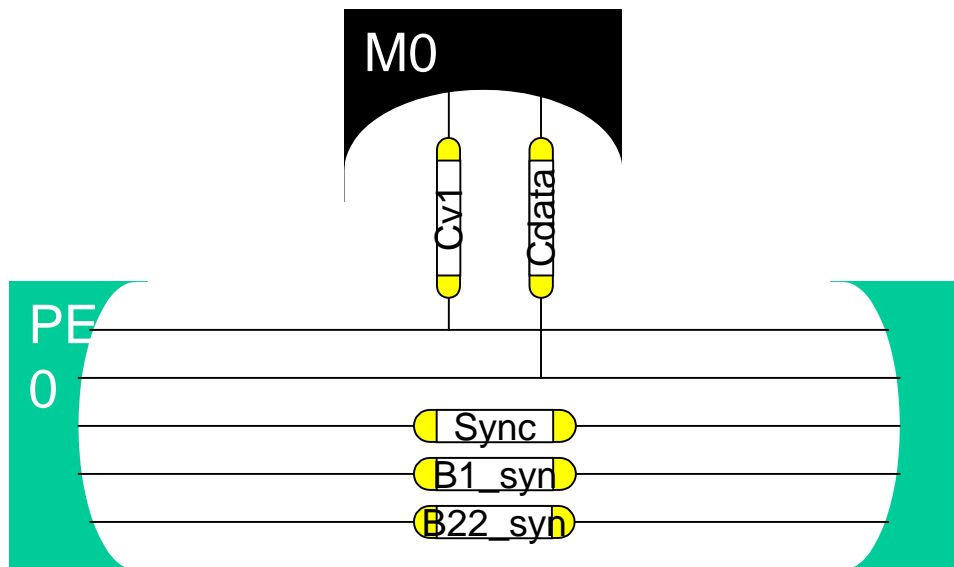
```
behavior Main(void)
{
    event go,finished;
    int    val;
    ch1 C0(go, finished, val);
    Bh2 B0(go, finished, val);
    Bh1 B1(C0);

    void main(void){
        par {
            B0.main();
            B1.main();
            :
        }
    }
}
```

# 組合せ例



- チャンネル同士がポート接続
- チャンネルをsplitするようになる



- N2N通信
- 通信における  
メモリアロケーション

# 通信内容の視点から



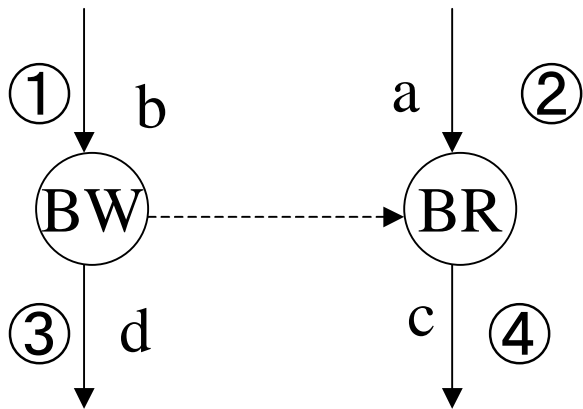
- 同期チャネルは基本パタンの組合せ
- 基本パターン
  - Blocking read/write
  - Non Blocking read/write
- 同期チャネルの例
  - ランデブ: <Block write, Block read>
  - 順序制約: <NB write, Block read>

*Gajski, et.al, "Specification and Design of Embedded Systems"*

# 同期チャネルの例



BW/BR:<event>



BR:書かれたら読める  
BW :読まれたら書ける  
<event>:イベント型

```
Channel sync(void) imp. IS IR{
void send(void){
    sending = true;
    if(!recving){
        wait(er);
    }
    sending=false;
    notify(ev);
}

void recv(void){
    recving = true;
    if(!sending){
        wait(ev);
    }
    recving=false;
    notify(er);
}
```

# 通信に関する応用例

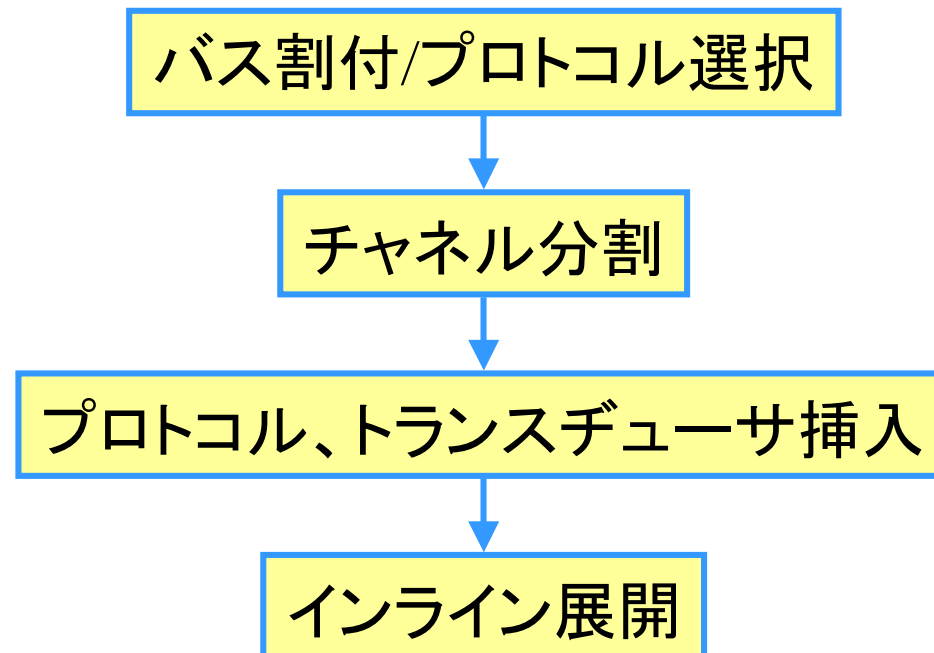


- 例1: 通信合成/リファインメント
- 例2: IOドライバを活用した通信リファインメント
- 例3: IP, Sプロトコルスタック

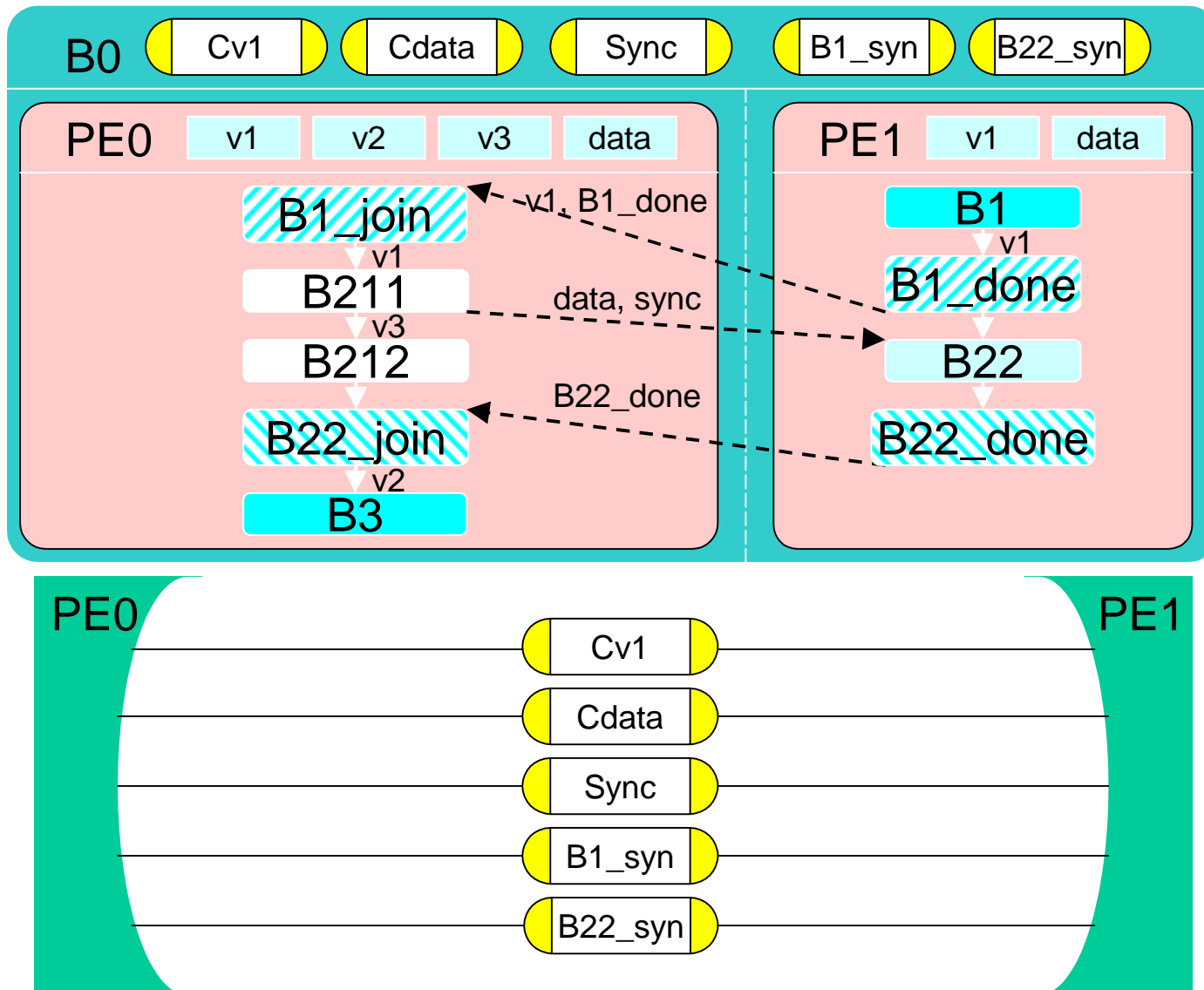
# 例1 : 通信合成における通信構造の リファインメント

Gajski, et.al, “System Design: A practical Guid with SpecC”

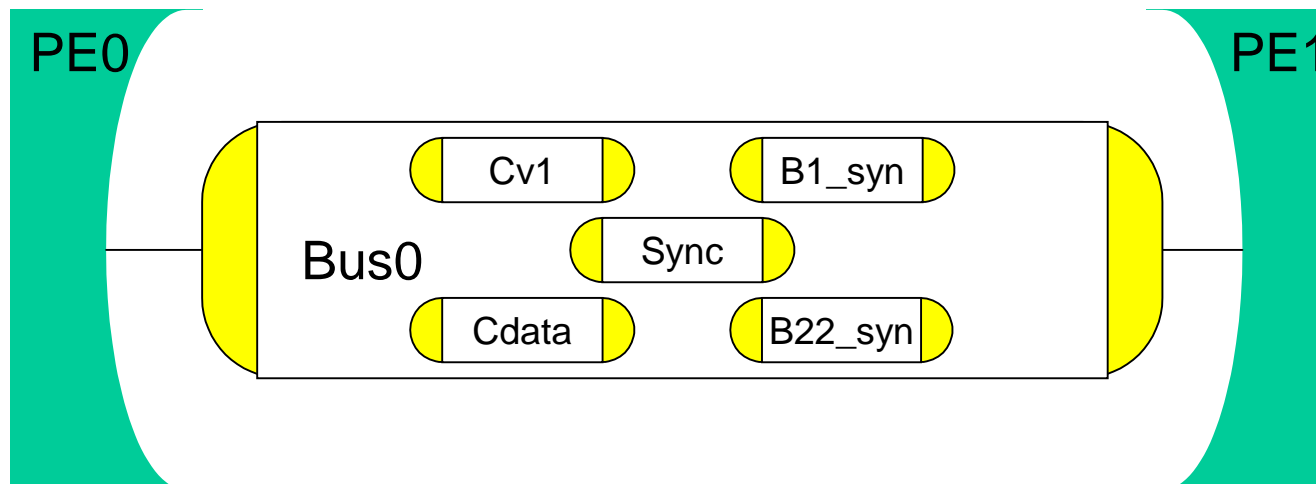
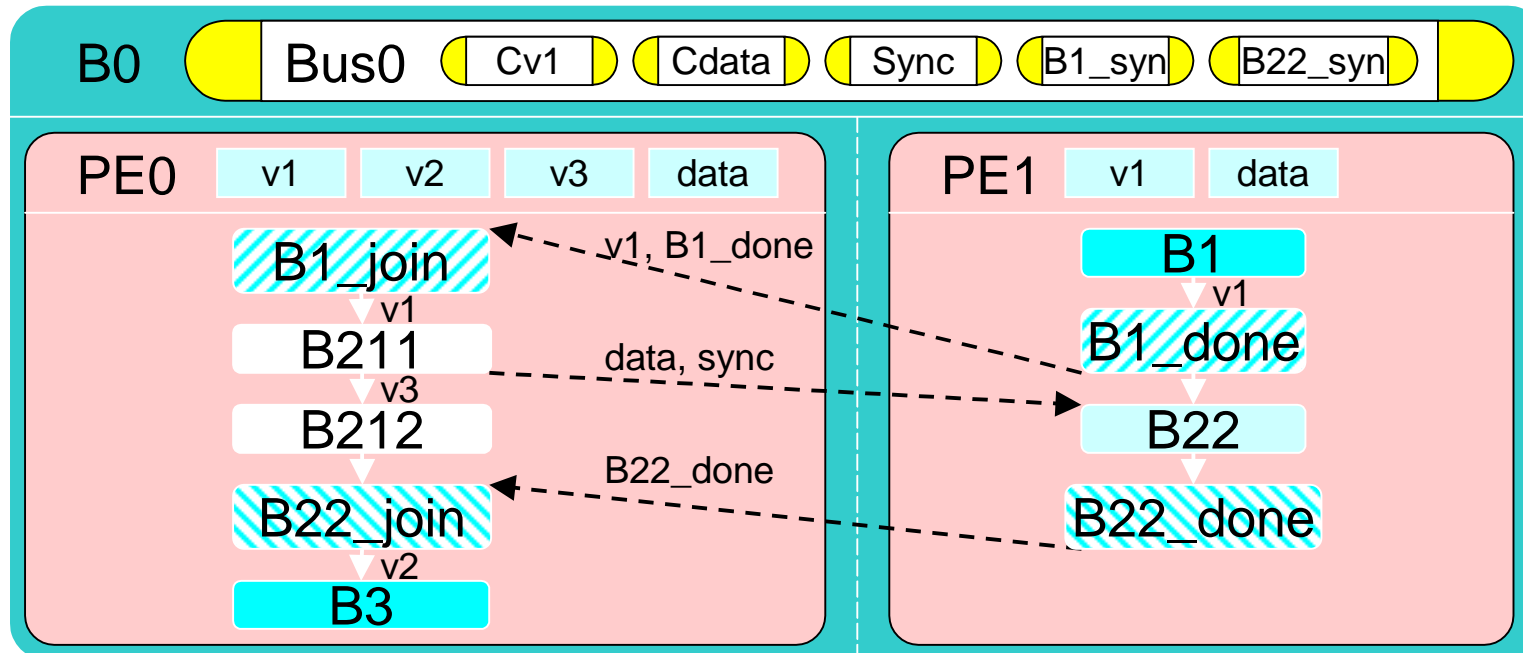
- ステップが進むにつれ、通信の構造もリファインされてゆく



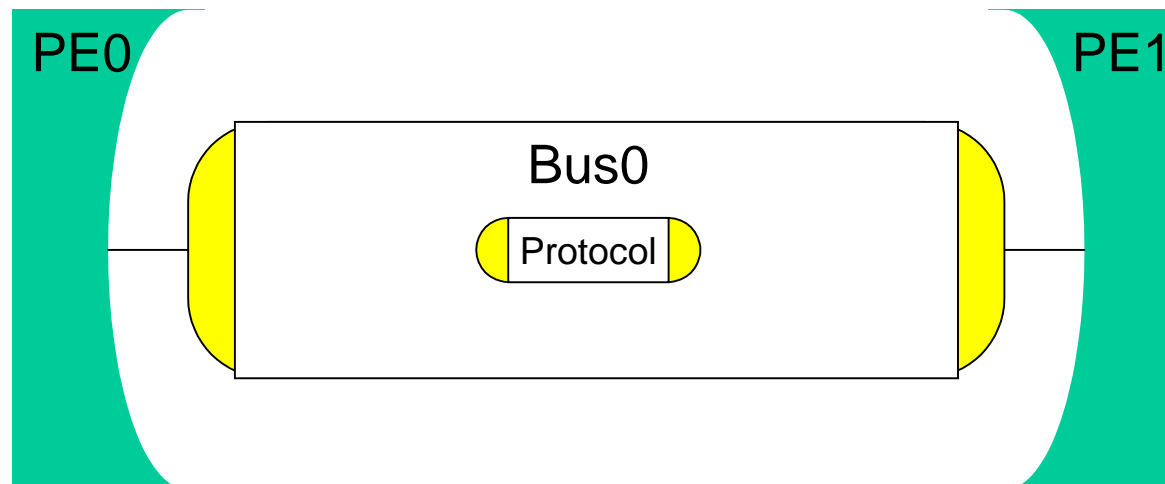
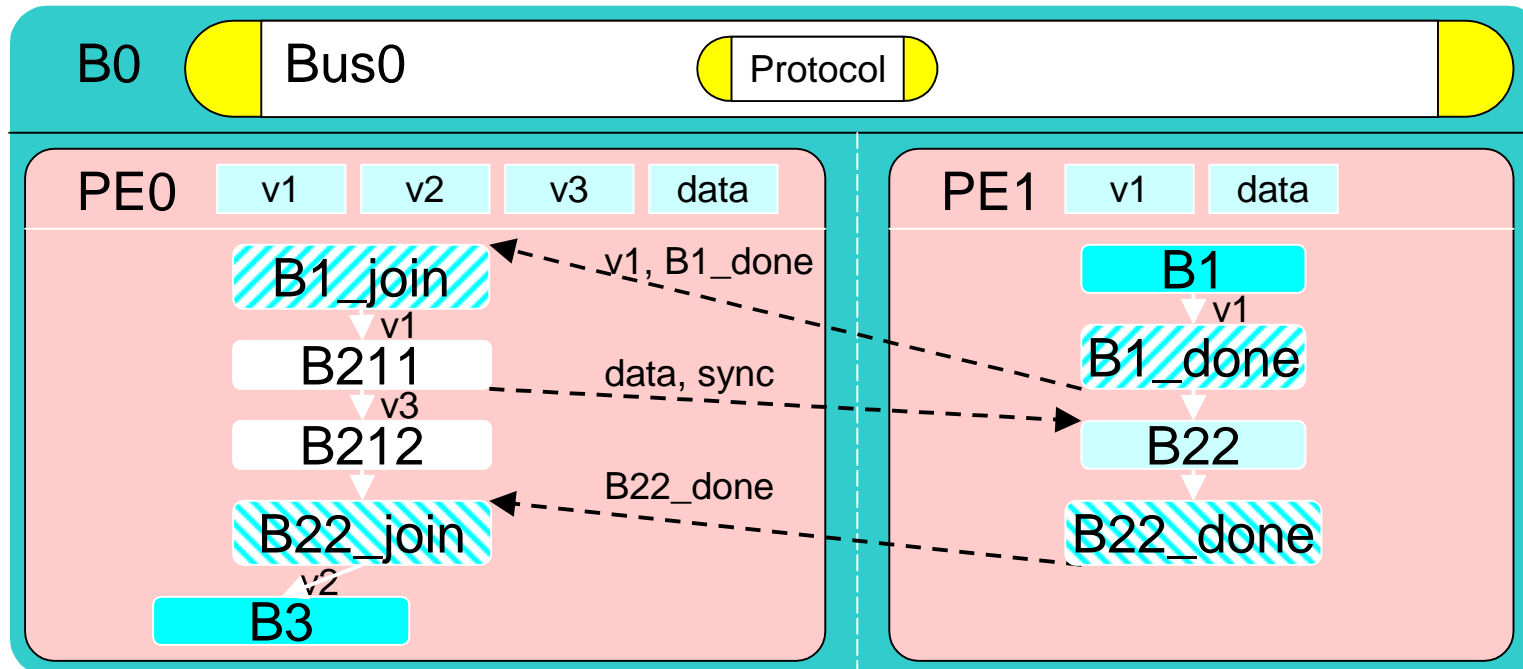
# 最初のモデル



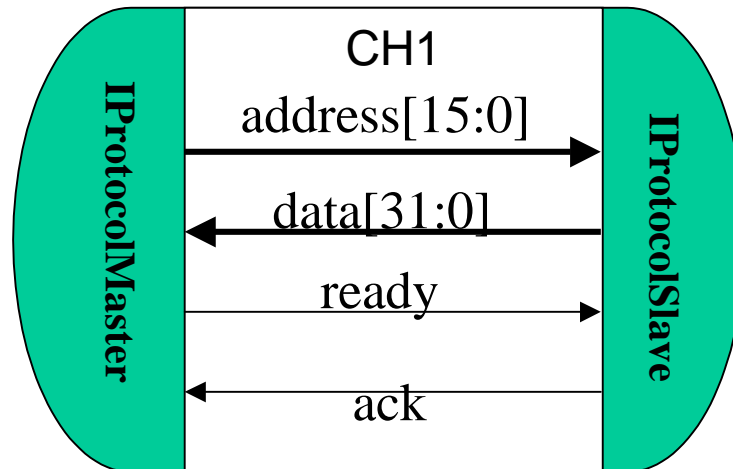
# バス割付/チャネル分割



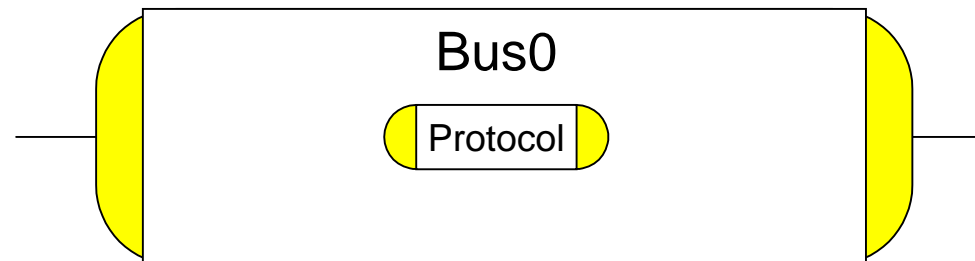
# プロトコル挿入



# プロトコル・チャンネル

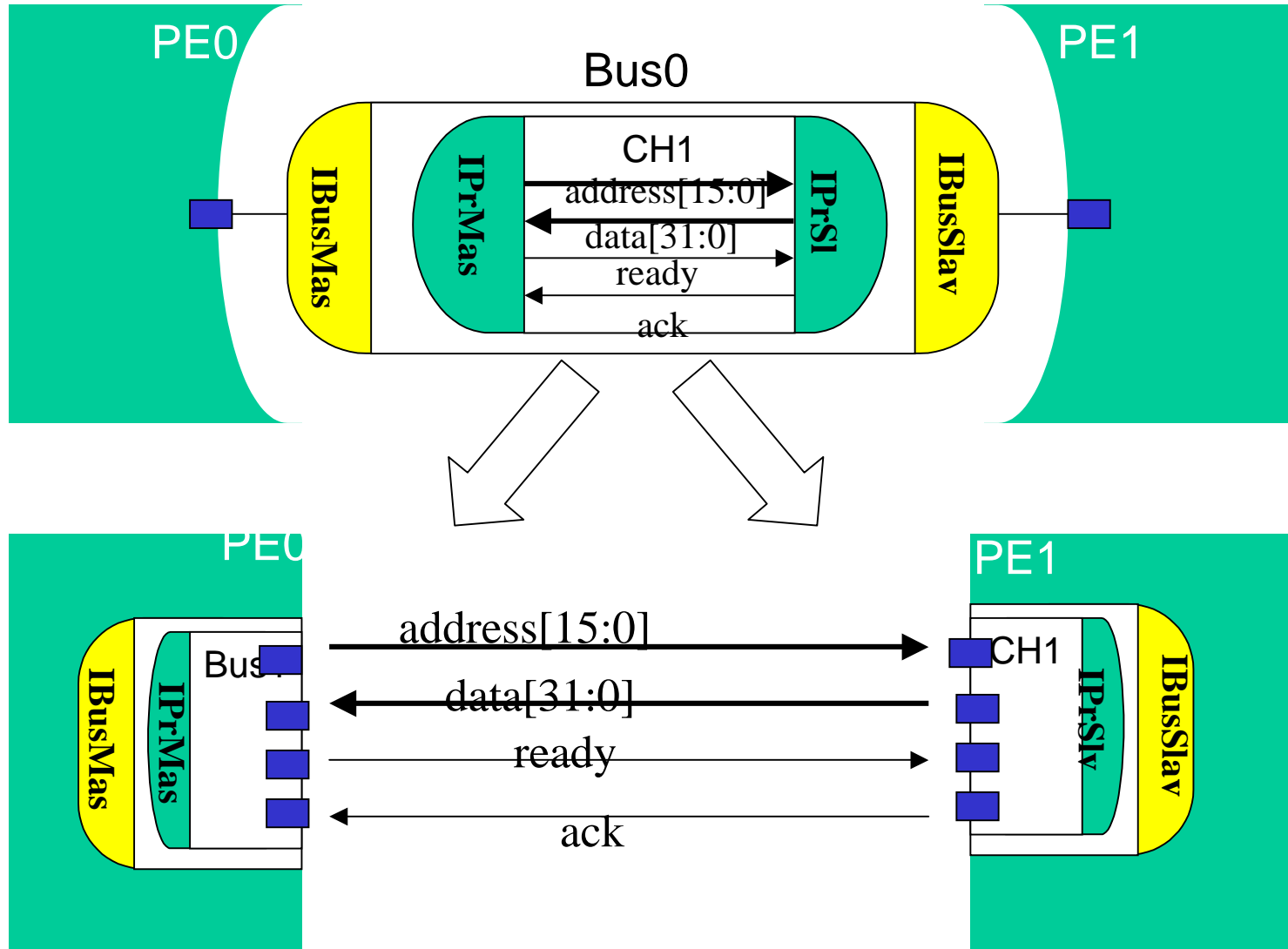


- プロトコルを介したメッセージ通信を実装
  - 同期
  - アービトレーション
  - 等々

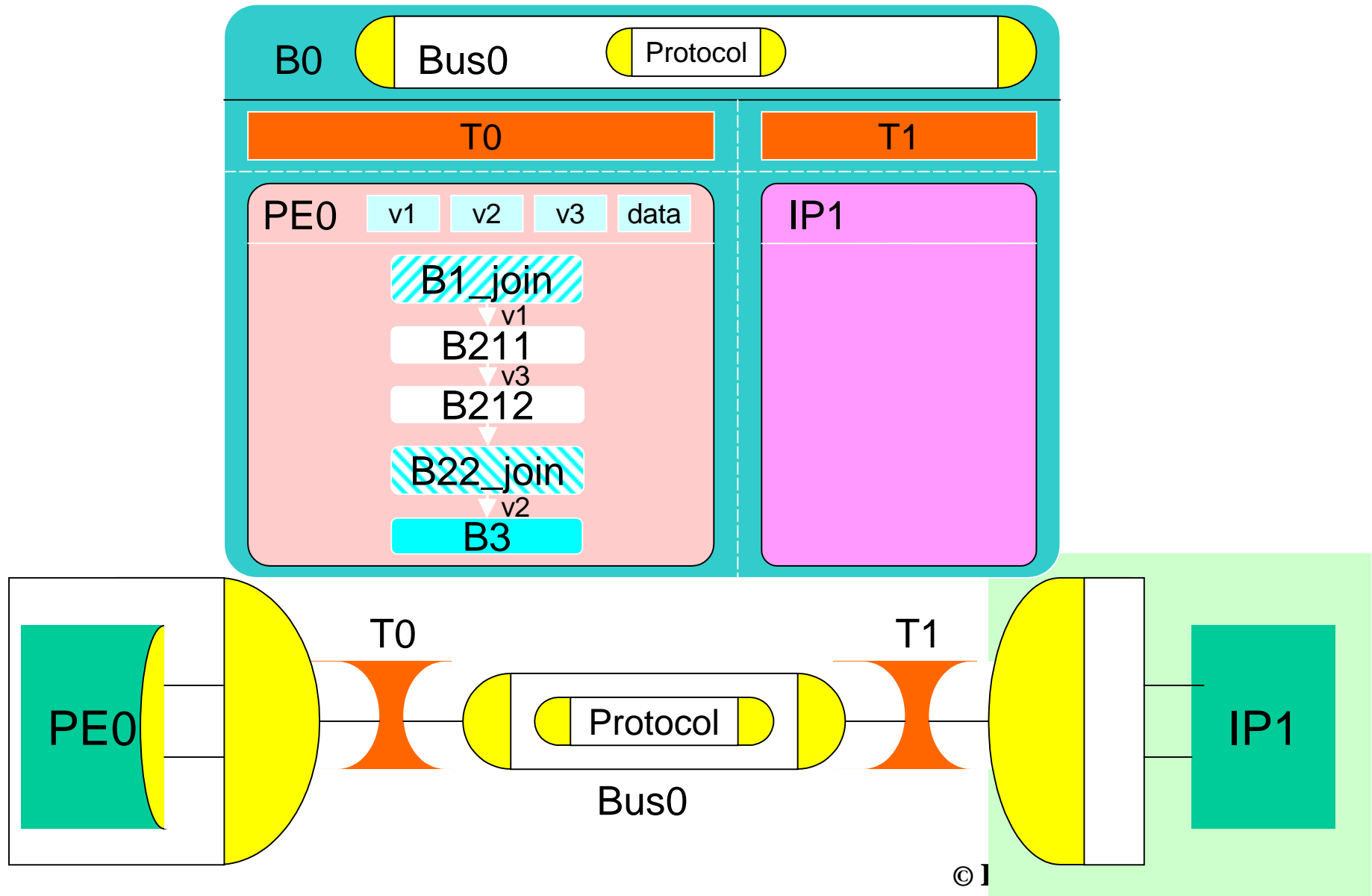


アプリケーションレイヤ・チャンネル

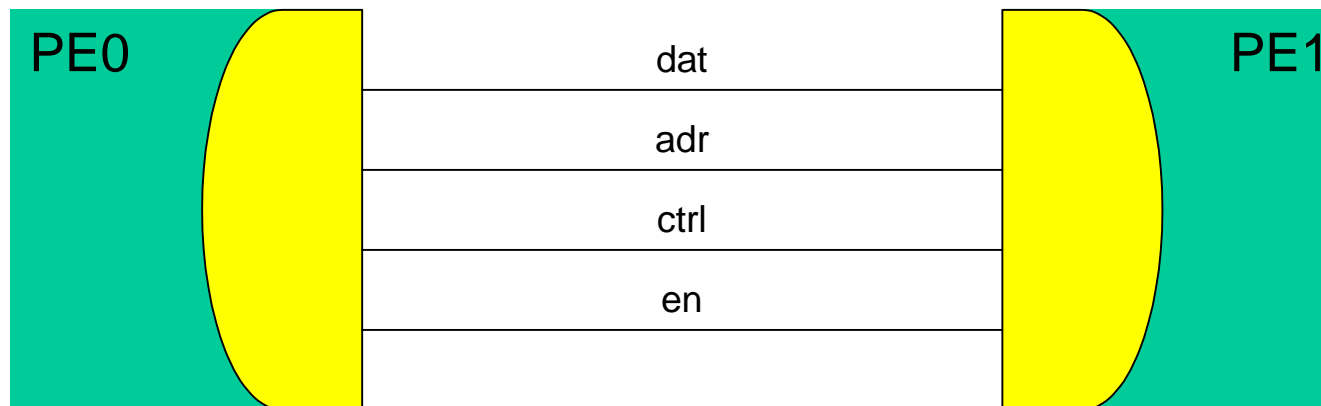
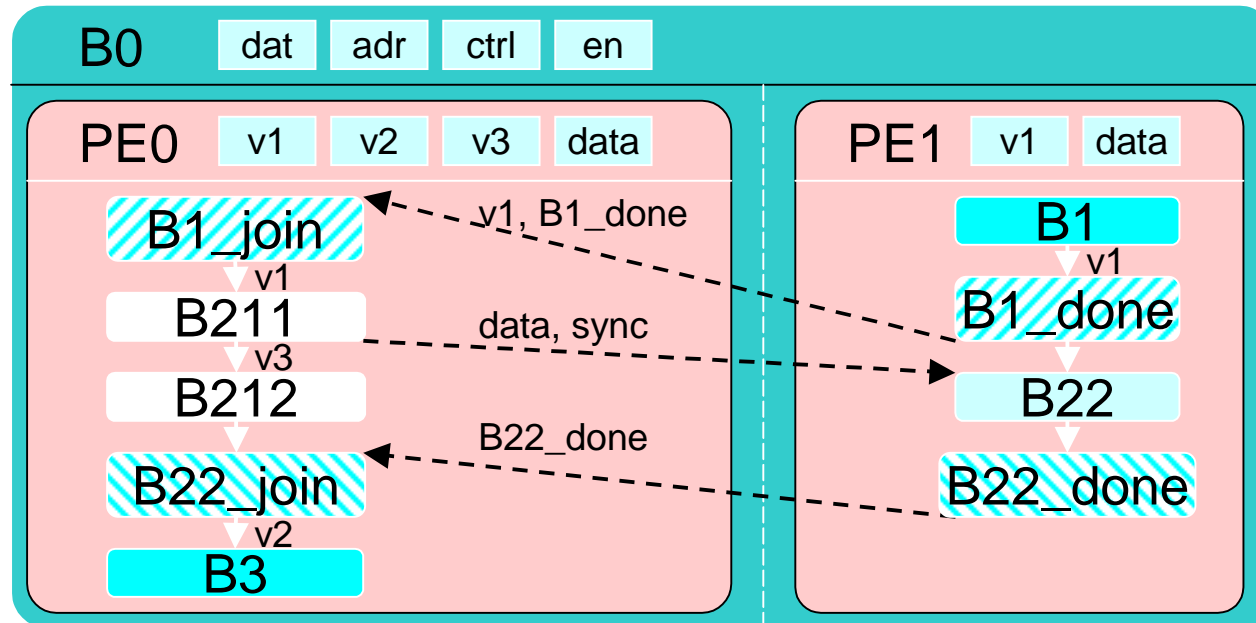
# インライン展開



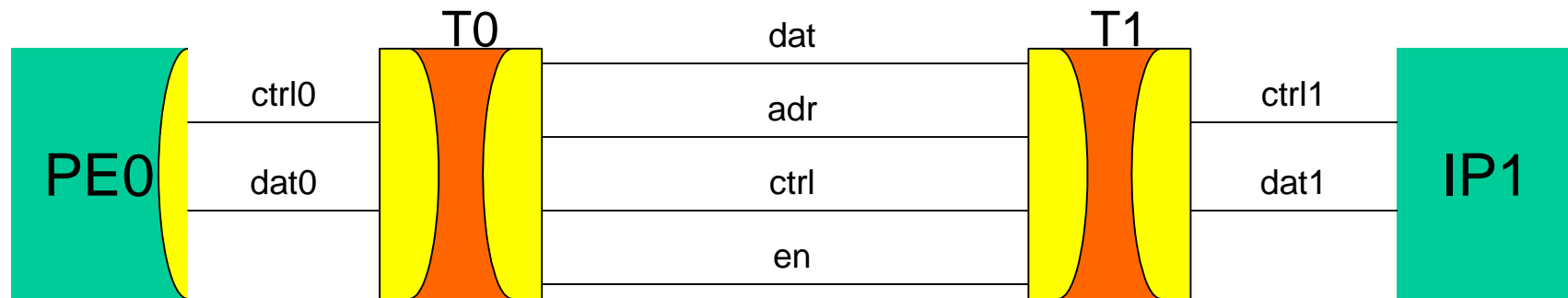
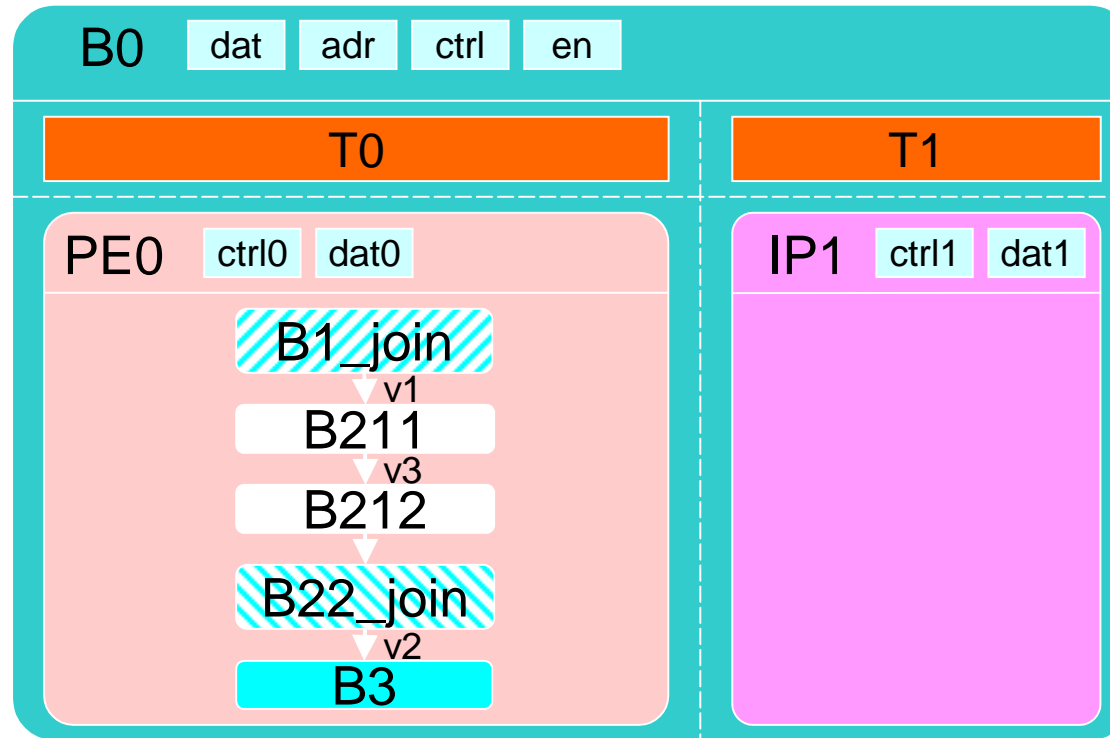
# トランスデューサとIP



# インライン展開: PE部品が合成可能

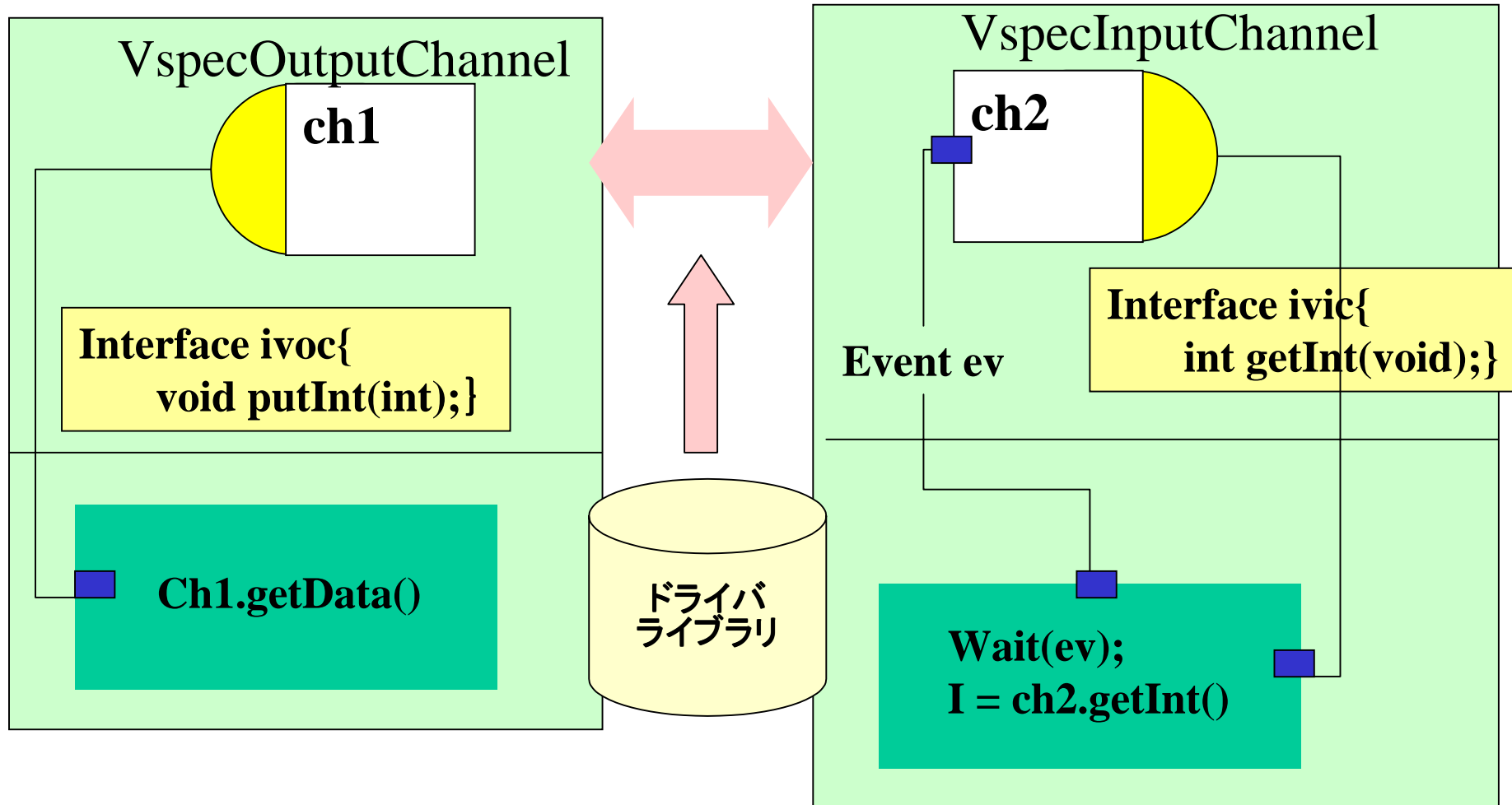


# PE部品がIPで合成不可能な場合



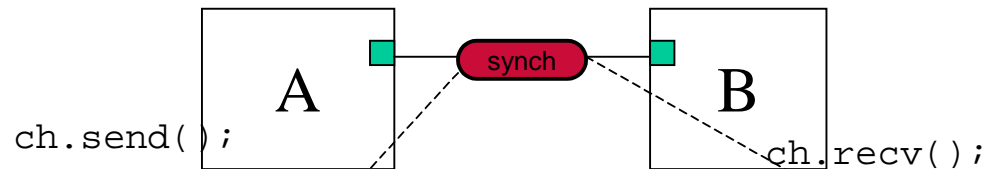
## 例2: デザイン分割における IOチャンネルをつかった通信リファインメント

# IOドライバ部品(VisualSpec)



- 外界とのIOを取り持つ組み込みチャンネル
- 様々なライブラリ(共有メモリ、TCPIP)を準備

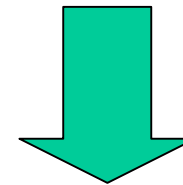
# 通信リファインメント /w IOドライバ



```
void send(void){
    sending = true;
    if(!recving){
        wait(er);
    }
    sending=false;
    notify(ev);
}

void recv(void){
    recving = true;
    if(!sending){
        wait(ev);
    }
    recving=false;
    notify(er);
}
```

•オリジナル  
(チャンネルを介した同期)

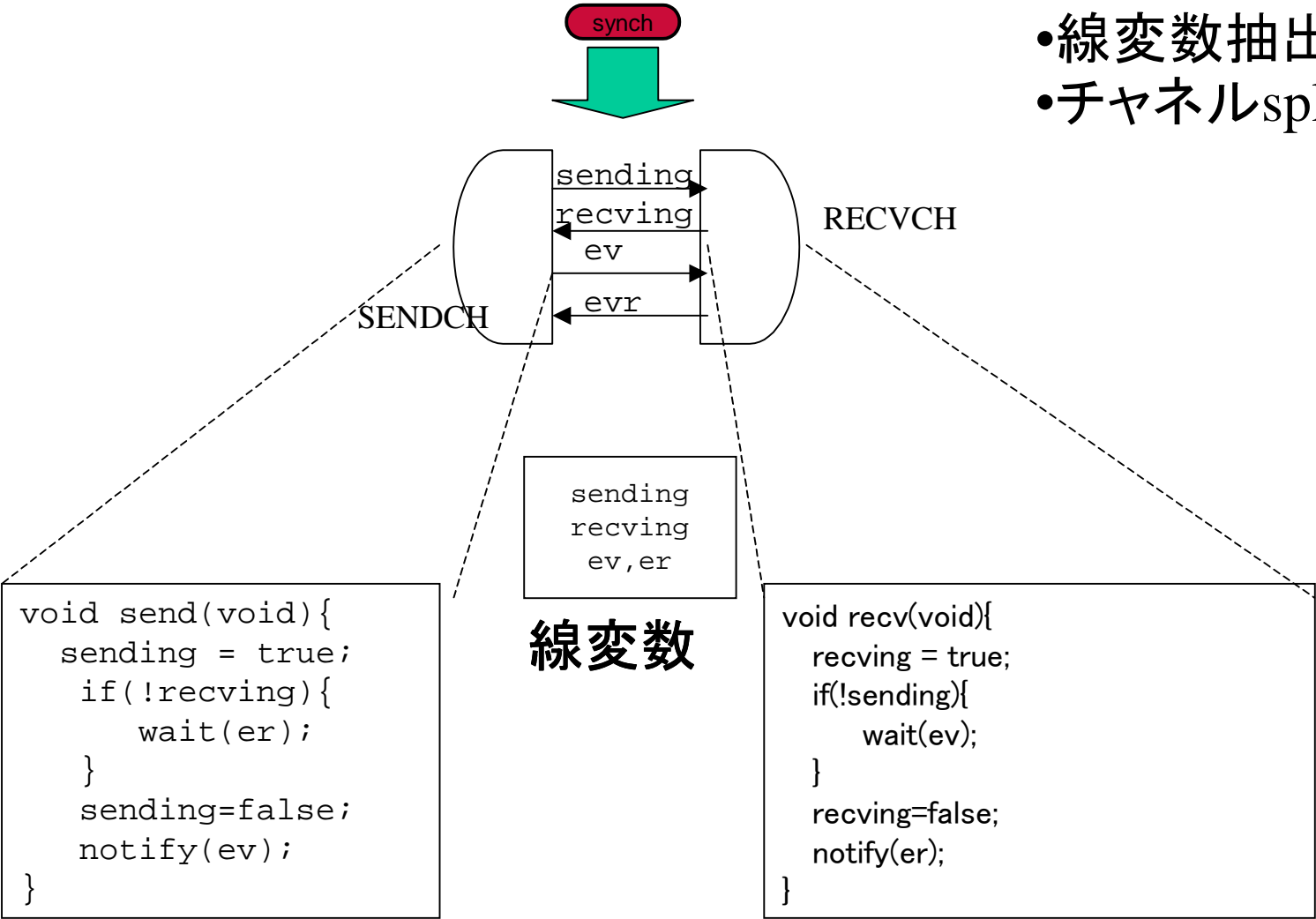


A,Bを2つのSWプロセスに分割する

# チャンネルsplitting



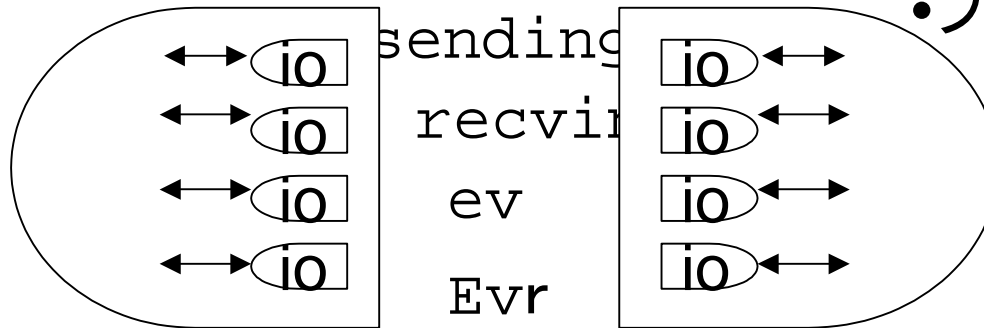
- 線変数抽出
- チャンネルsplit



# Ioチャンネルの挿入



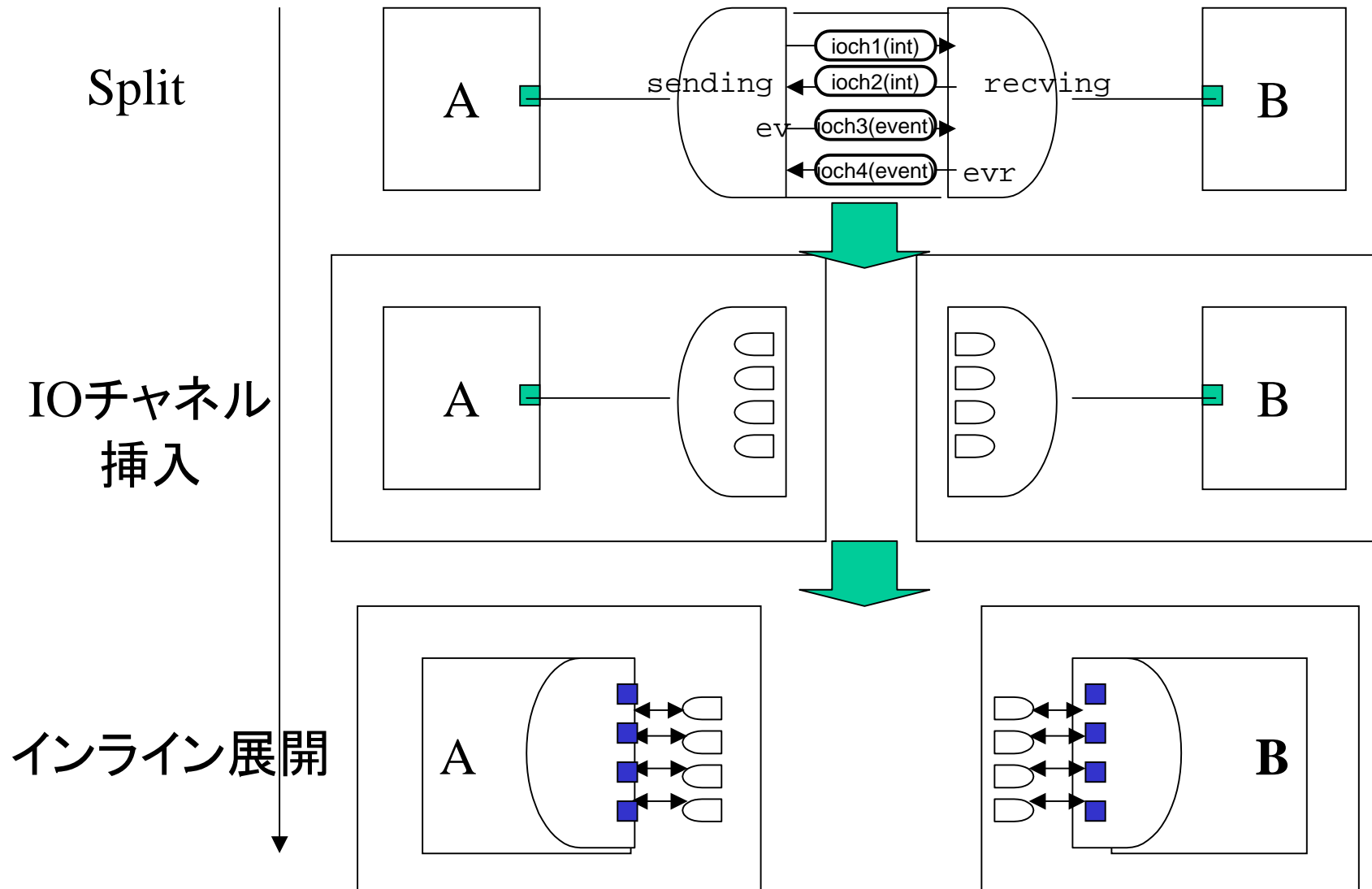
- Ioチャンネル挿入
- プロトコル修正



```
void send(void){
  ch1.putInt(1); //sending=1
  recving = ch2.getInt();
  if(!recving){
    wait(er);
  }
  ch1.putInt(0); //sending=false;
  ch3.putEvent(); //notify(ev);
}
```

プロトコル修正

# リファインメントの流れ

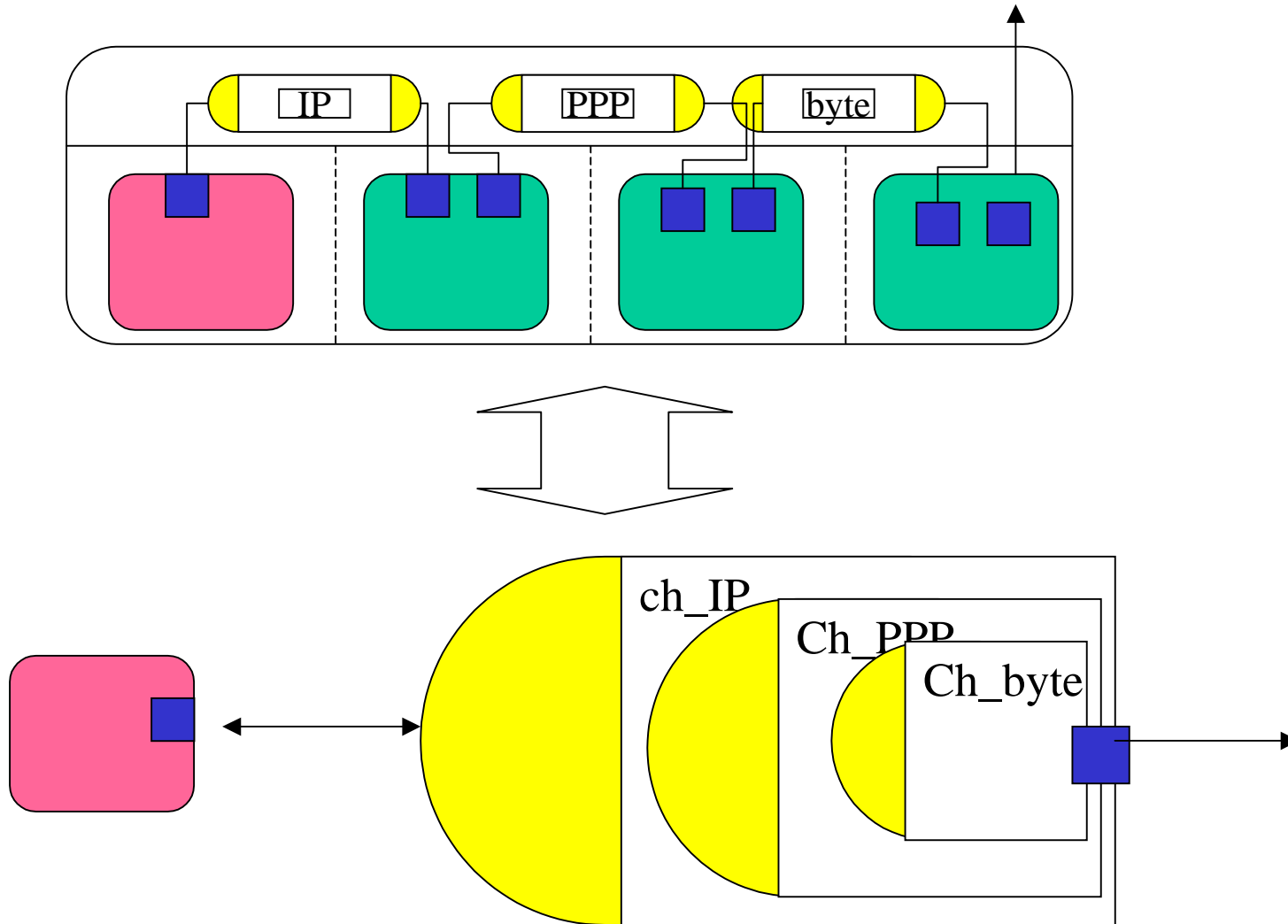


# IOドライバライブラリの例

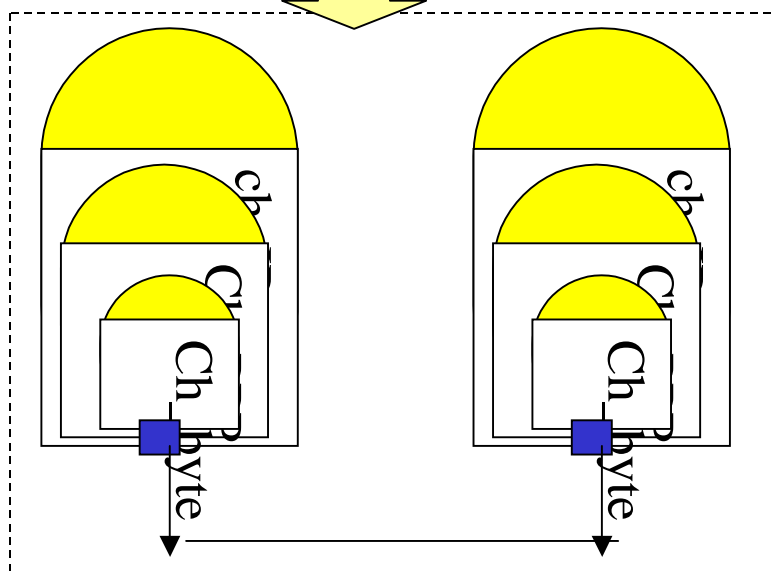
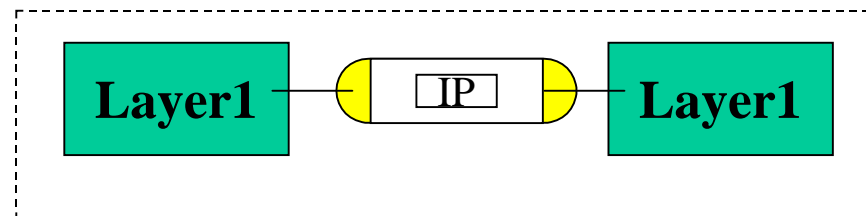
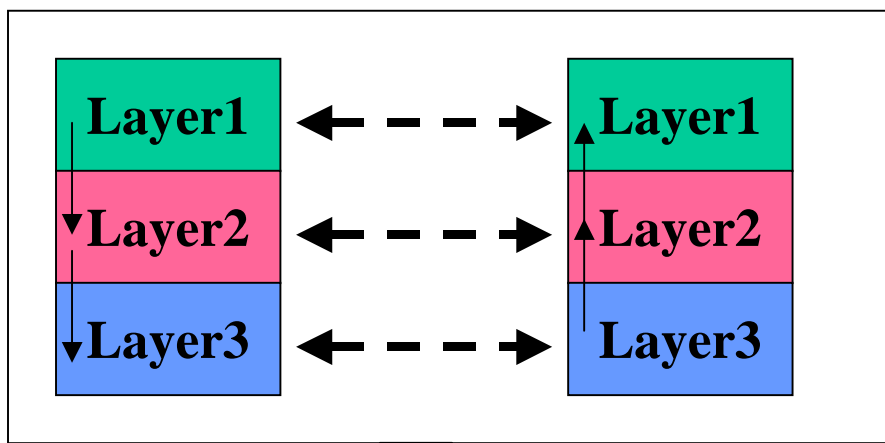


- 共有メモリライブラリ
  - 共有メモリを介して2プロセスが通信
  - 同一PC上で有効
- TCP/IPライブラリ
  - ソケット通信を介してプロセスが通信
  - ネットワーク上のPC間でも有効

# プロトコル変換



# プロトコルスタック？

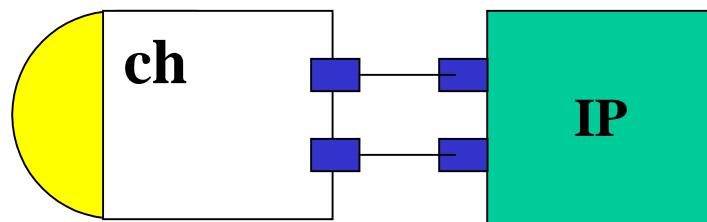


- 2つの側面
- (×)同一モデル
- 詳細度に応じてリファインされる
- どうやって切り替え??

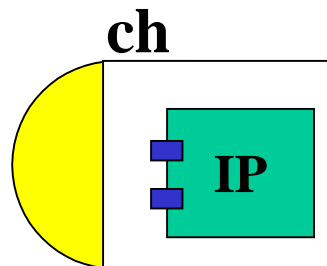
# IP活用としてのチャンネルの利用



- IPの使いかた(サービスレイヤ)も含めて部品化
- 使い方=チャンネルメソッド+チャンネル内の手順
- 手順は合成可能なSpecC記述で書く
  - →プロトコル変換部品(トランスデューサ)で合成



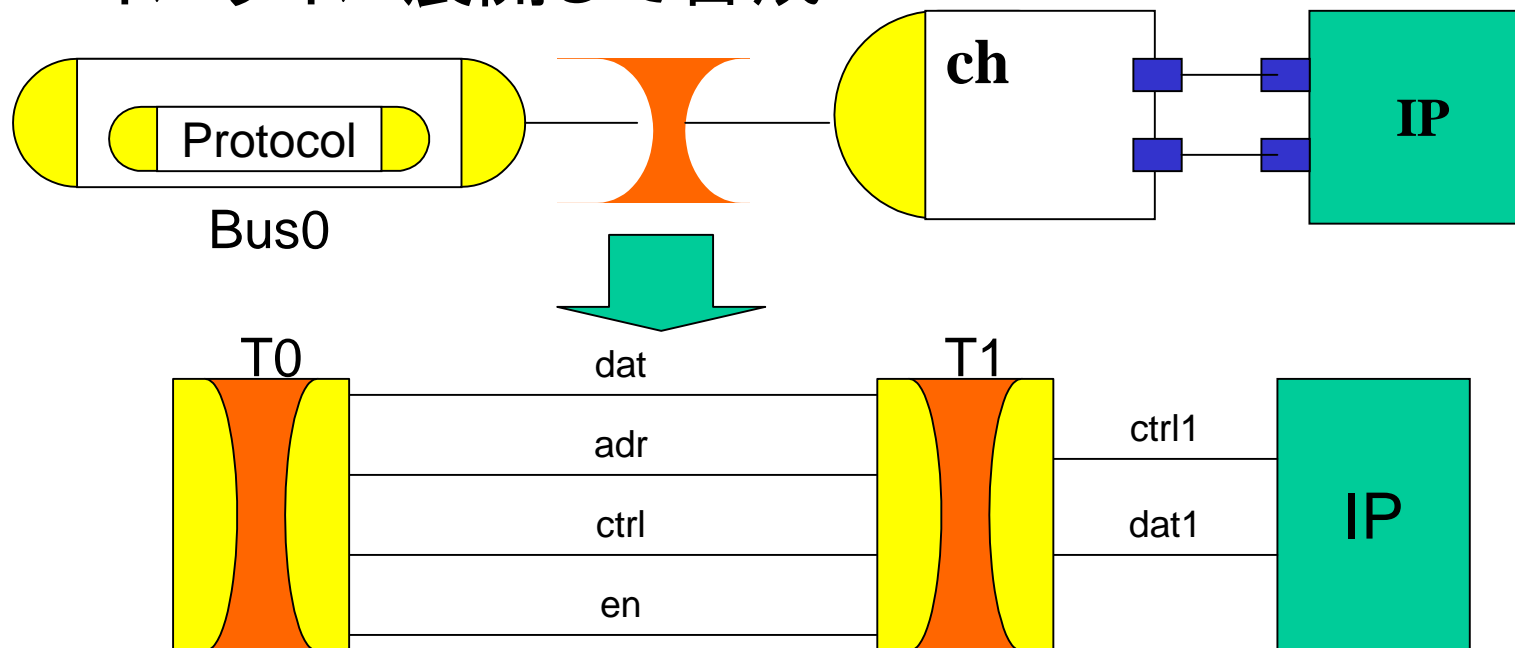
•ポート接続



•ラッパ

# トランスデューサ

- プロトコル変換部品
- IPと元のデザインでのチャンネルとのプロトコルアンマッチを解消する
- インライン展開して合成



- Channel通信の構造のバリエーション
  - TPOに応じて使い分ける
- 例を紹介
  - 通信合成で構造がリファインされる例
  - デザイン分割(IOドライバ活用)で通信がリファインされる例
  - プロトコルスタック→課題
- リファインコマンドとして実装