5. パイプライン化計算

5.1 パイプライン技法

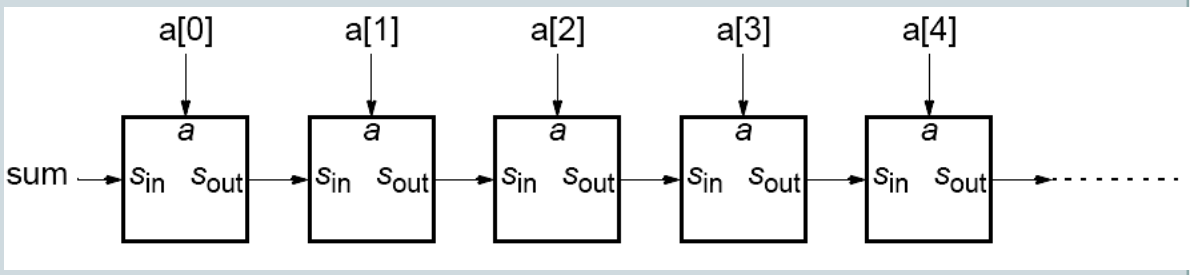
- 問題を連続して実行する一連のタスクに分割する
- 各タスクは別のプロセッサで実行される
- パイプラインの一つのステージは一つのタスクに対応する
- 一種の機能分割方式

例:総和のループ

```
1.   for(i=0; i<n; i++)
2.     sum = sum + a[i];
```

ループを展開すると

```
1.   sum = sum + a[0];
2.   sum = sum + a[1];
3.   sum = sum + a[2];
4.   sum = sum + a[3];
5.   sum = sum + a[4];
```

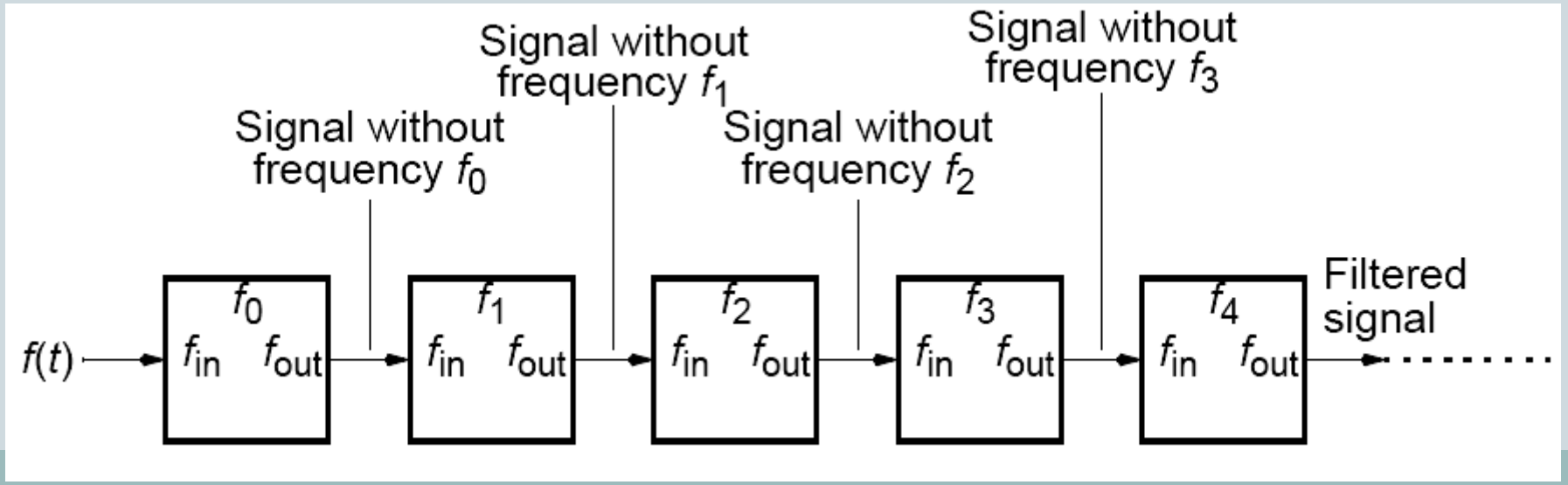


- 各ステートメントをステージに対応させる。
- ステージiは　s_out = s_in + a[i];　を実行する

5.1 パイプライン技法

例：周波数フィルタ

- 入力信号f(t)から特定の周波数(f_0, f_1, ...)を除去する
- 周波数フィルタのパイプライン処理
  - 各ステージが特定の周波数を担当する
  - 周波数毎の振幅図作成に応用できる



5.1 パイプライン技法

パイプライン処理は次の状況にあるときに速度向上が期待できる

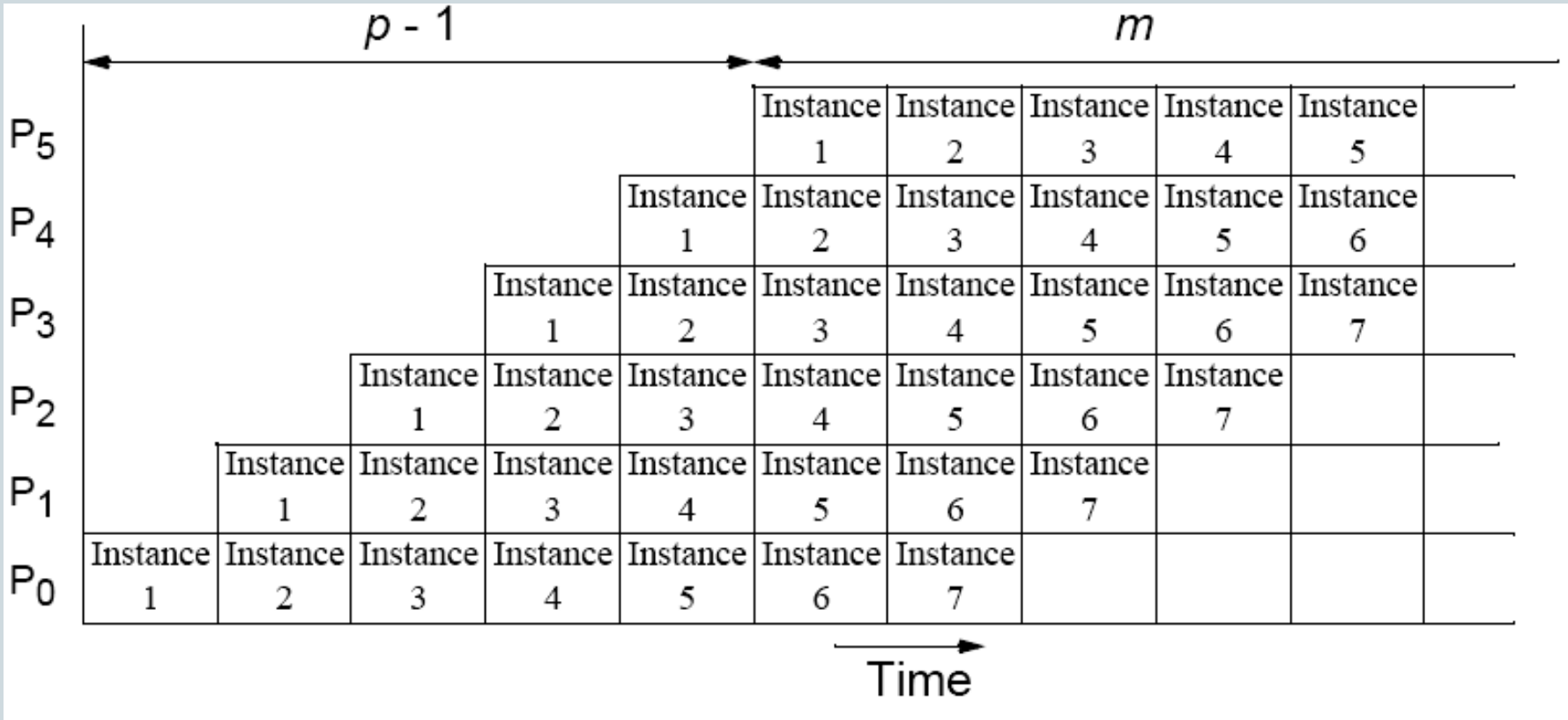
- Type1: 複数のインスタンスが実行されるとき
- Type2: それぞれが複数の操作を必要とする一連のデータを処理しなければならないとき
- Type3: 次のプロセスを起動するための情報をプロセスが全ての内部操作を完了する以前に先へ渡せるとき

5.1 パイプライン技法(Type1)(1)

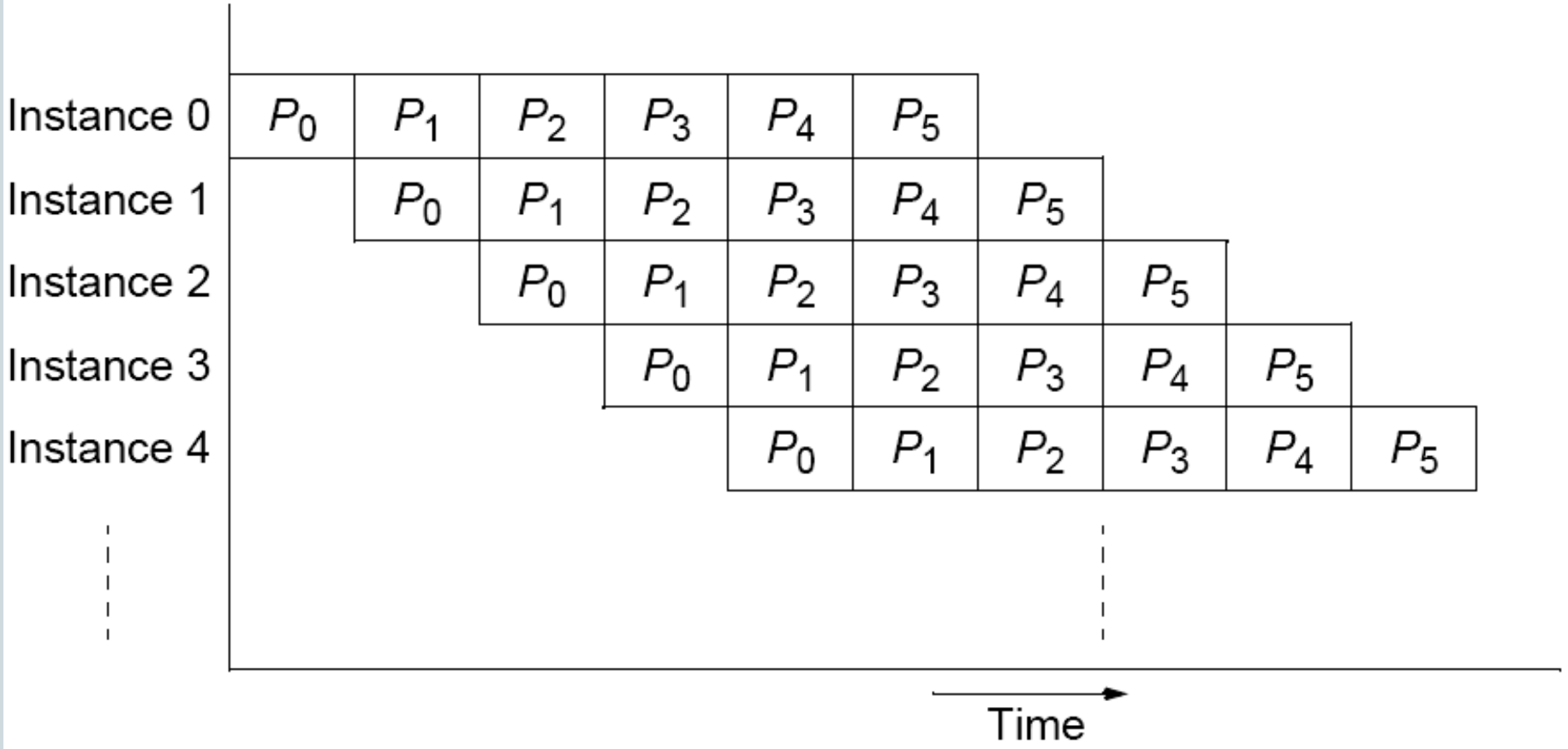
Type1: 一つのプログラムを複数のパラメータで(複数回)実行する

- 1パイプラインサイクル
  - 各プロセスは同一時間でそのタスクを完了すると仮定したときの時間
- p個のプロセスでm個のインスタンスを実行するためのパイプラインサイクル数
  - m + p −1
- 平均サイクル数
  - (m + p −1)/m
  - mが大きいとき 1サイクルに近づく
- 最初のp−1サイクル(パイプラインレイテンシー)後はサイクル毎に完了

5.1 パイプライン技法(Type1)(2)



5.1 パイプライン技法(Type1)(3)

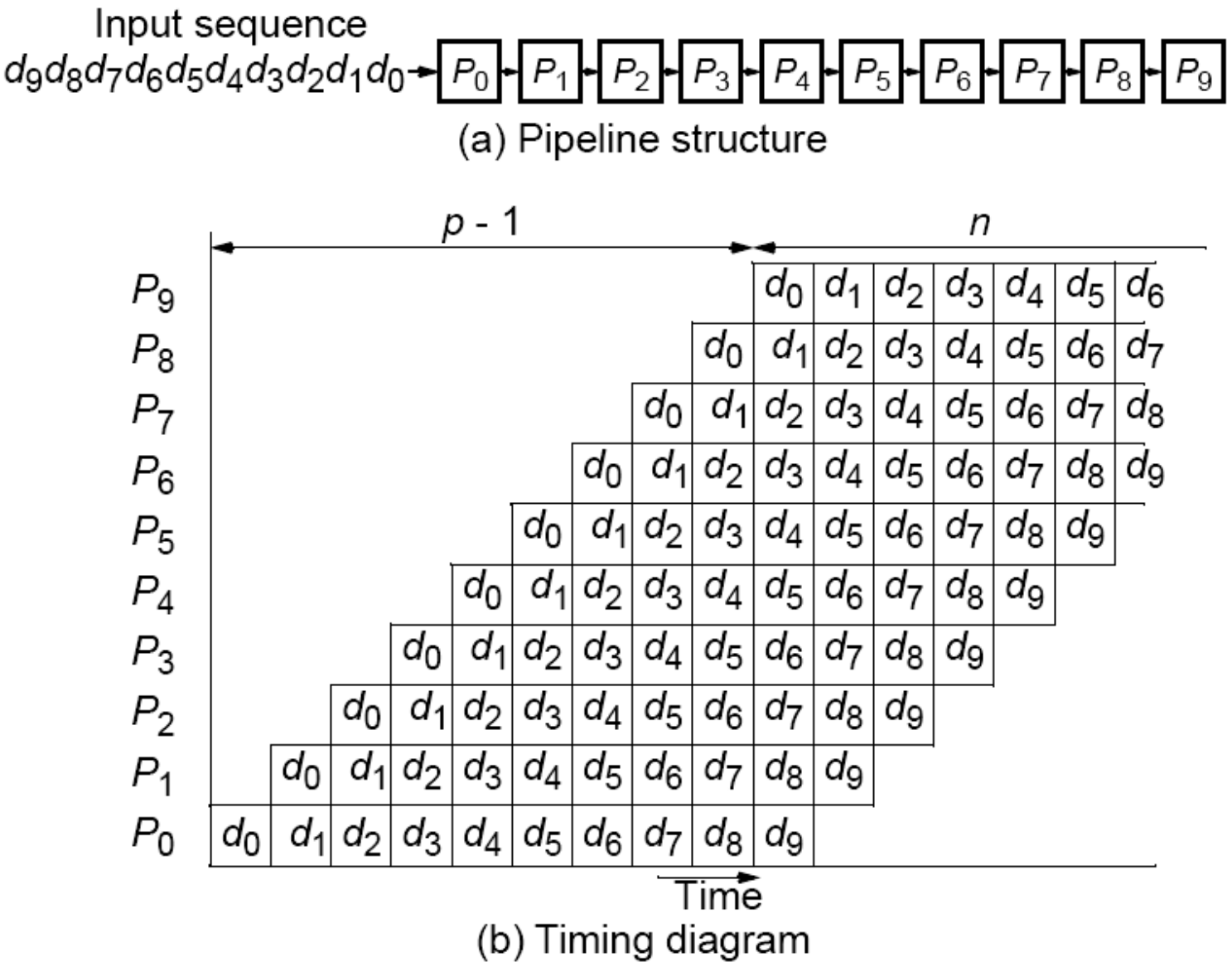


5.1 パイプライン技法(Type2)(1)

Type2: 一つのプログラムの中で複数のデータ処理を行う

- nデータのときの実行時間は、パイプラインサイクルが等しいと仮定する
  - (p-1) + n

5.1 パイプライン技法(Type2)(2)

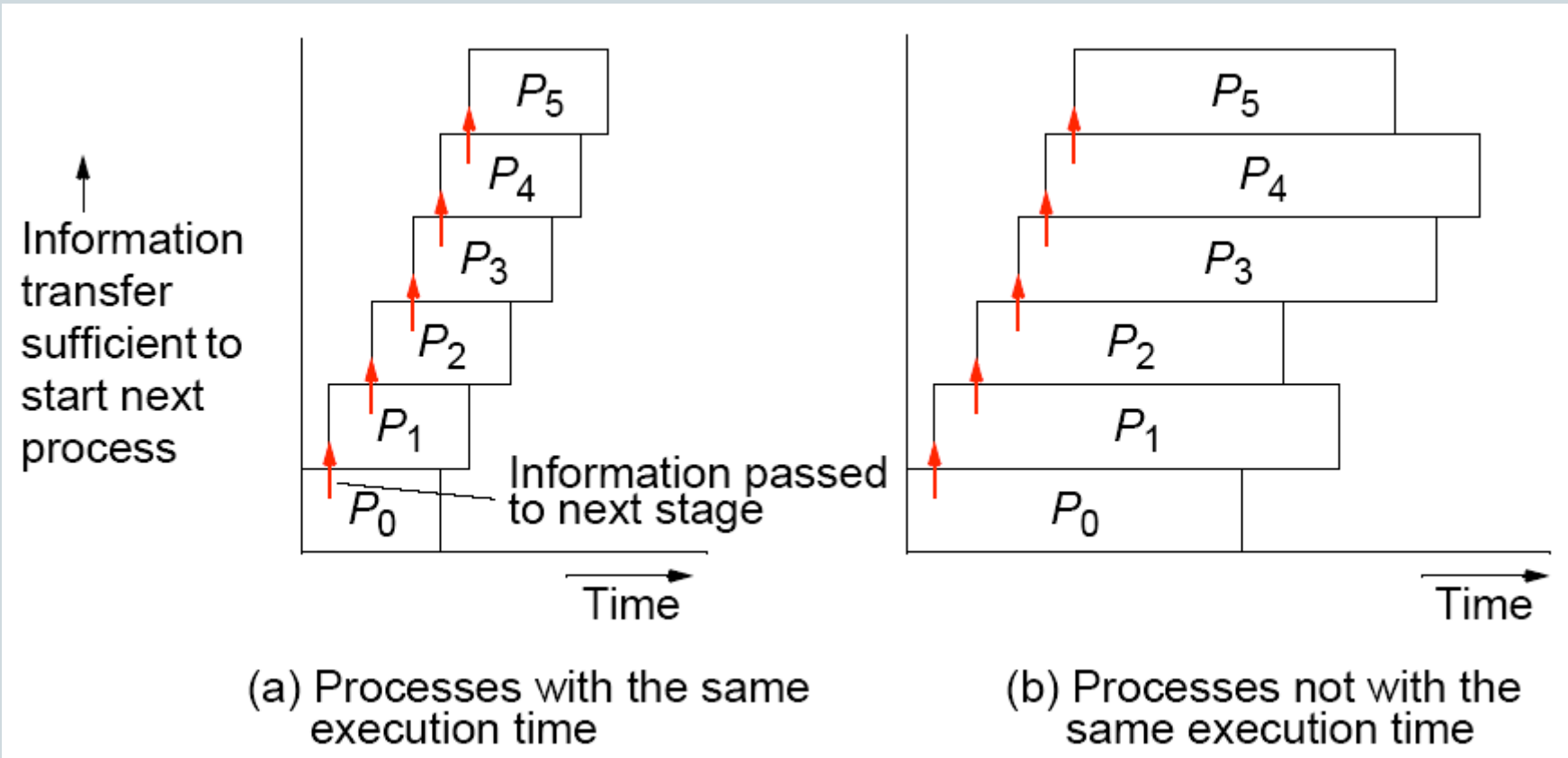


5.1 パイプライン技法(Type3)(1)

Type3: 問題のインスタンスは一つしかないが，各プロセスがその完了前に次のプロセスに情報を渡せる

- UNIXのpipeを並列実装
  - ls -al | sort | a2ps

5.1 パイプライン技法(Type3)(2)



5.2 パイプライン化応用のための計算プラットフォーム

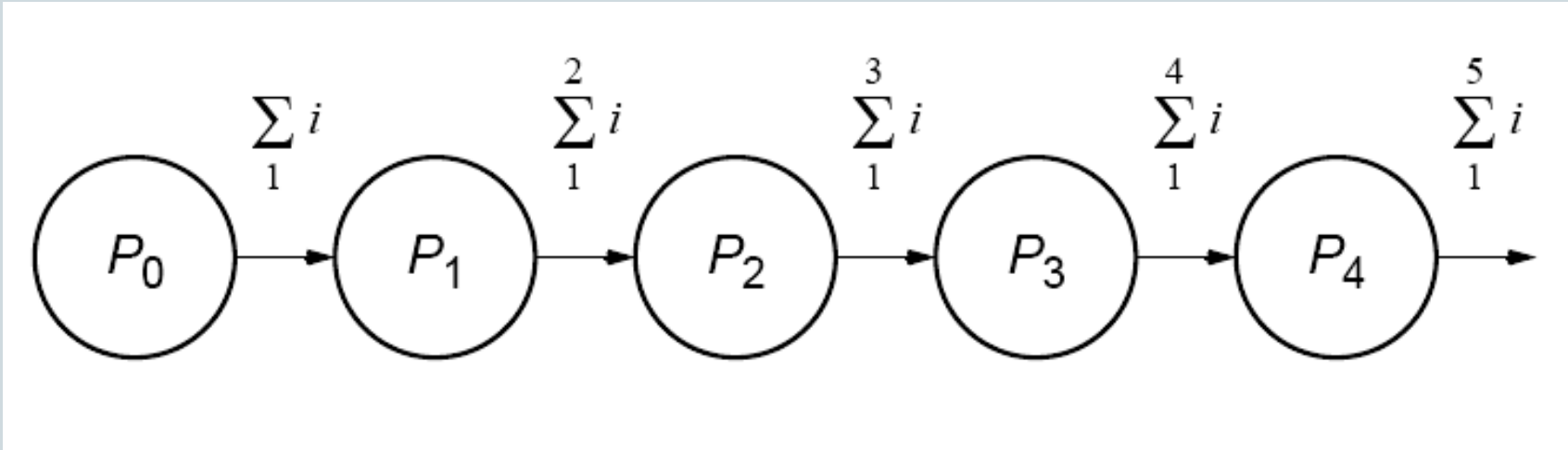
パイプライン処理の理想的な結合構造

- ライン，リング
- メッシュ，ハイパーキューブにも埋め込める
- ネットワーク結合コンピュータには不向きかも

5.3 パイプラインプログラムの例(数の加算)(1)

5.3.1 数の加算

- 各プロセスは1個の数を保持する
- 各プロセスは左から入ってきた累積和と保持データを加算する
- 右のプロセスに渡す



5.3 パイプラインプログラムの例(数の加算)(2)

P_i (0<i<n-1):

```
1.    recv(&accumulation, P_{i-1});
2.    accumulation = accumulation + number;
3.    send(&accumulation, P_{i+1});
```

P_0:

```
1.    send(&number, P_1);
```

P_{n-1}:

```
1.    recv(&accumulation, P_{n-2});
2.    accumulation = accumulation + number;
```

SPMD:

```
1.    if(process > 0){
2.      recv(&accumulation, P_{i-1});
3.      accumulation = accumulation + number;
4.    }
5.    if(process<n-1)) send(&accumulation, P_{i+1});
```

5.3 パイプラインプログラムの例(数の加算)(2)

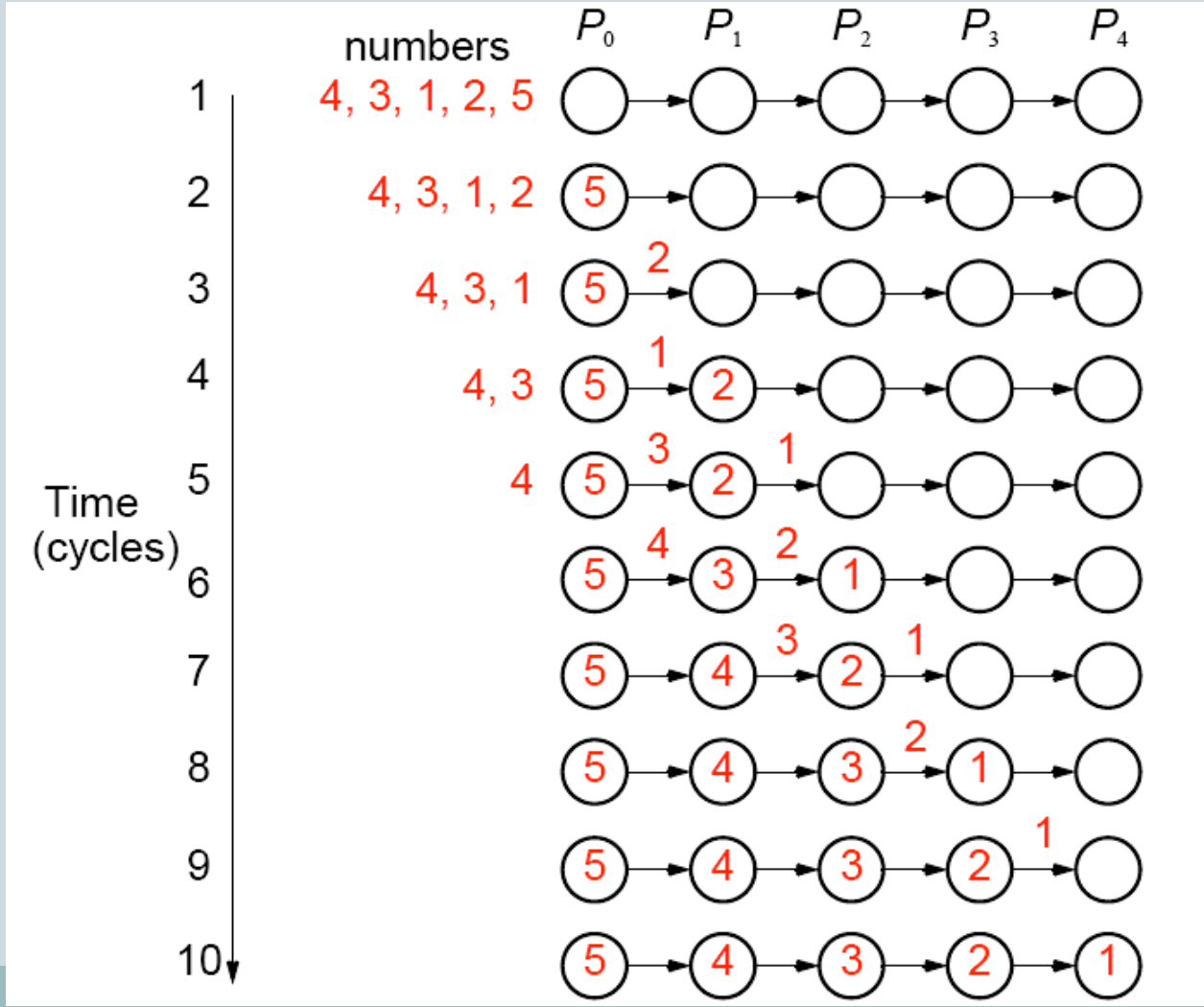
解析

- 各ステージで一個の数を加算する
- データ数をnとする．すなわち，n=p
- 1インスタンスの場合
  - 1パイプラインサイクルは一回の加算と左からと右への2回の通信
  - t_comp = 1
  - t_comm = 2(t_startup + t_data)
  - 全実行時間は
  - t_total = (2(t_startup + t_data) + 1)n
- 複数インスタンスの場合
  - 加算するn個の数がm組あるとすると
  - t_total = (2(t_startup + t_data) + 1)(m + n − 1)
  - mが大きいとき，平均実行時間は
  - t_ave = (t_total)/m −> 2(t_startup + t_data) + 1

5.3 パイプラインプログラムの例(ソート)(1)

パイプラインソート
(挿入法の並列版)

- それまでに受信した数のうち最大数を記憶してそれより小さい数を全て次へ送る



5.3 パイプラインプログラムの例(ソート)(2)

```
1.   recv(&number, P_{i-1})
2.   if(number>x){
3.     send(&x, P_{i+1});
4.     x=number;
5.   }else send(&number, P_{i+1});
```

```
1.   right_procno = n-i-1;   /* no of processes to the right */
2.   recv(&x, P_{i-1});
3.   for(j=0; j<right_procno; j++){
4.     recv(&number, P_{i-1});
5.     if(&number > x){
6.       send(&x, P_{i+1});
7.       x=number;
8.     }else send(&number, P_{i+1});
9.   }
```

5.3 パイプラインプログラムの例(ソート)(2)

解析

逐次計算

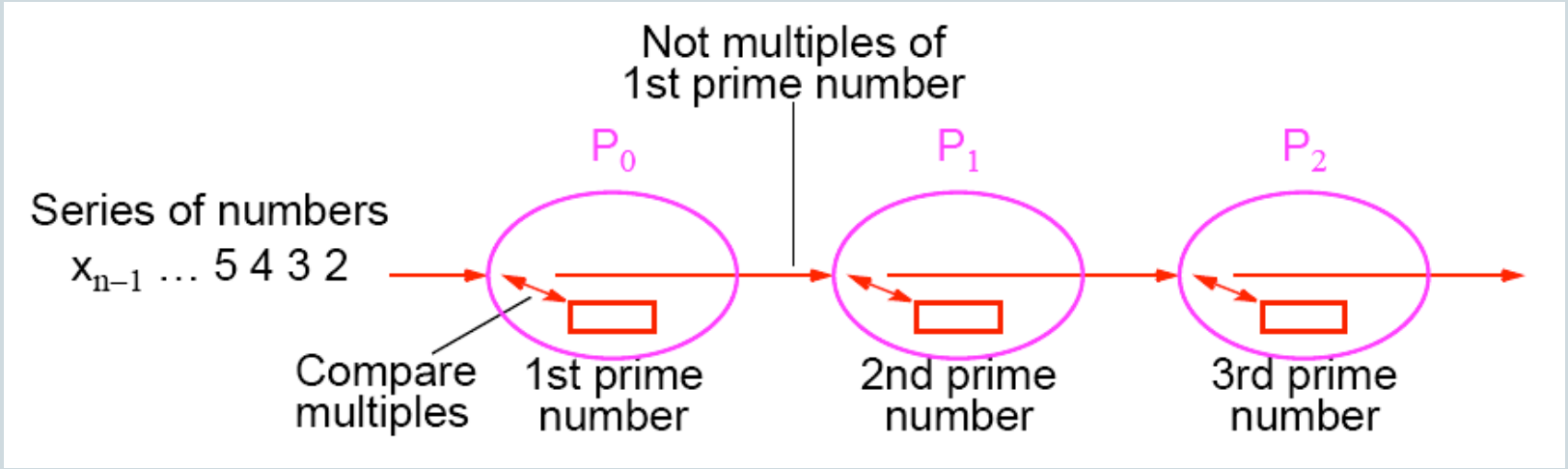
- $t_s = (n-1)+(n-2)+\ldots+2+1=n(n+1)/2$

並列実装

- パイプライン数n, ソートされる数nとすると
- パイプラインサイクル数
  - $n+n-1= 2n-1$
- 各パイプラインサイクル
  - $t_{comp} = 1$
  - $t_{comm} = 2(t_{startup} + t_{data})$
- 全実行時間
  - $t_{total} = (t_{comp}+t_{comm})(2n-1)=(1+2(t_{sartup}+t_{data}))(2n-1)$

5.3 パイプラインプログラムの例(素数発生)(1)

1. 2以上の整数列を生成する
2. 最初の数を素数として保存
3. 保存した素数の倍数を整数列から削除
4. 2に戻る



5.3 パイプラインプログラムの例(素数発生)(2)

エラトステネスのふるい

- 逐次コード

```
1.  for(i=2; i<n; i++)
2.    prime[i]=1;                    /* Initialize array */
3.  for(i=2; i<=sqrt_n; i++)         /* for each number */
4.    if(prime[i]==1)                /* identified as prime*/
5.      for(j=i+i; j<n; j=j+i)       /* delete all multiples*/
6.        prime[j]=0;              /* includes already done */
```

- 並列コード

```
1.  recv(&x, P_i-1);
2.  for(i=0; i<n; i++){
3.    recv(&number, P_{i-1});
4.    if(number == terminator) break;
5.    if((number % x)!=0) send(&number, P_{i+1});
6.  }
```