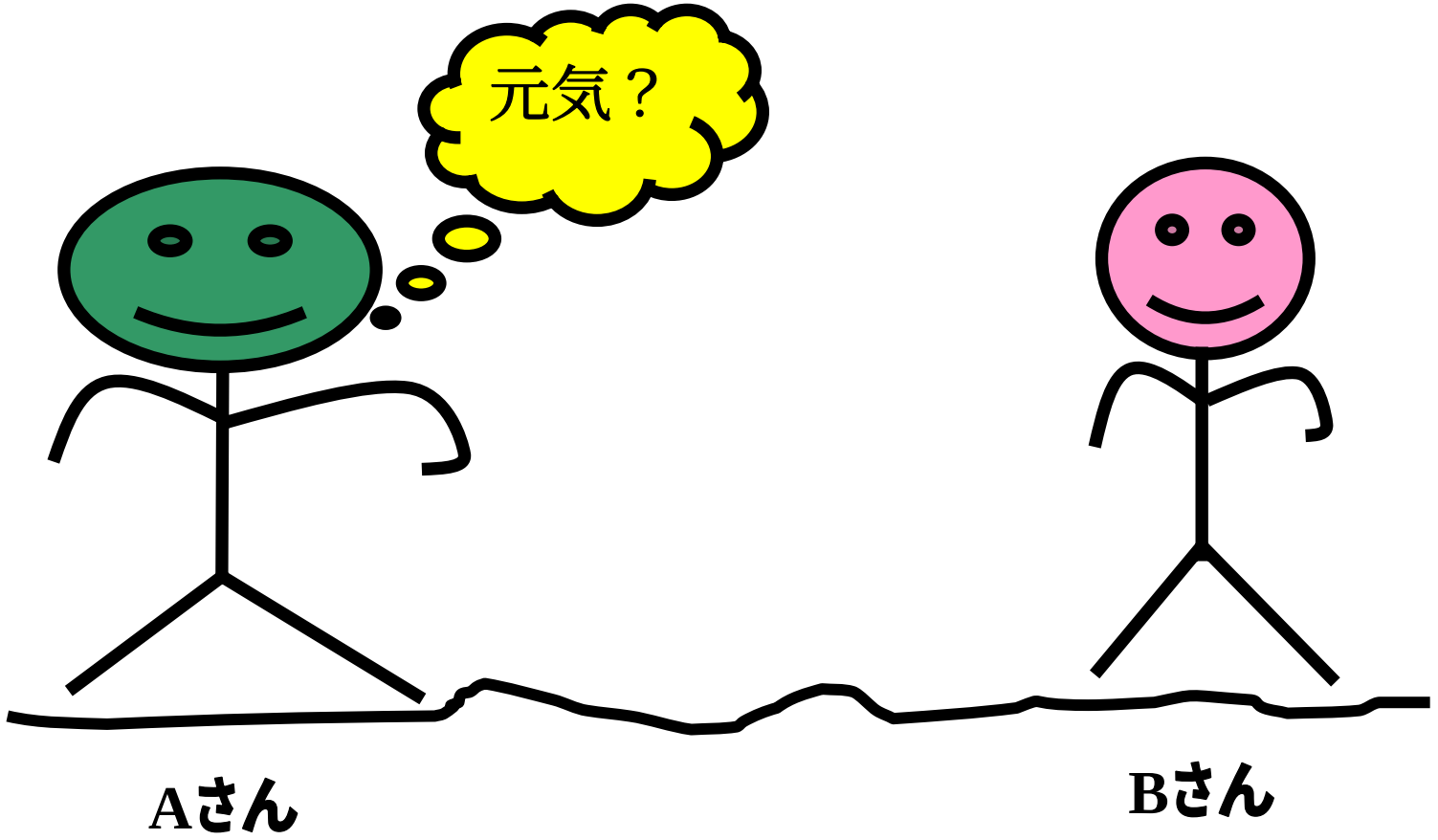
JCSPによる CSPモデルプログラミングの 実際

北海道職業能力開発大学
制御技術科　　中原 博史

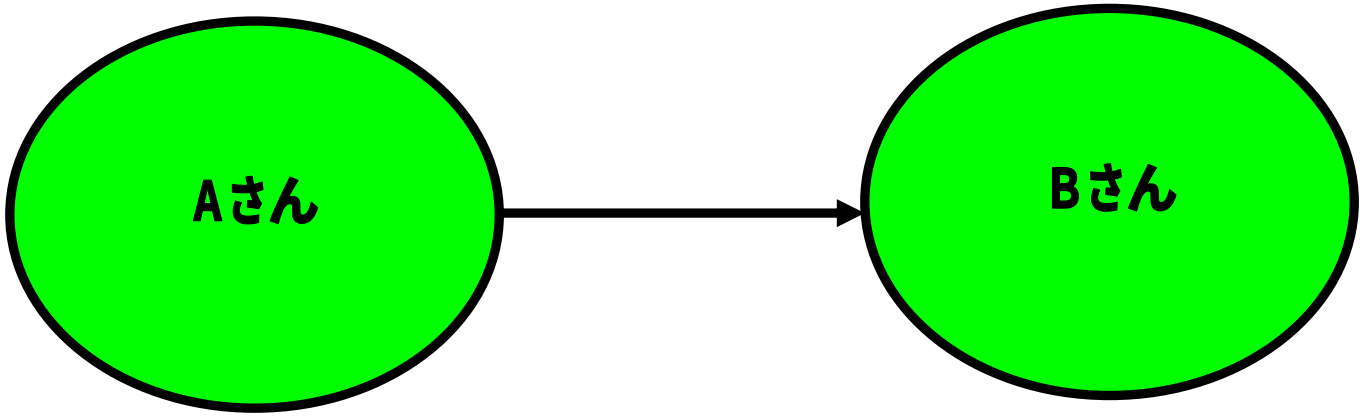
予　　定

1．CSPモデル
2．JCSP（CSP for JAVA）でCSPモデルの記述
3．実行
4．チャネルの転送スピード
5．ネットワーク対応、分散配置、仮想チャネル
6．仮想チャネルの転送スピード
7．JCSPの利用例

CSPモデルの例

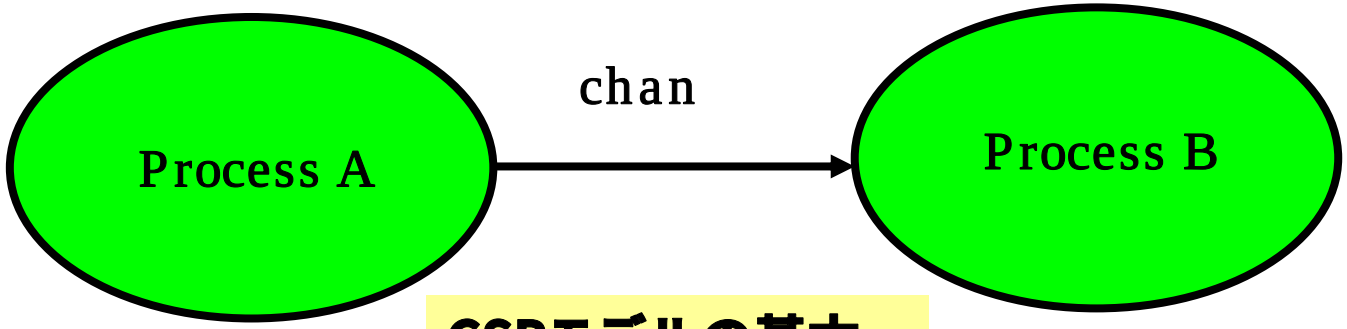


CSPモデルの例



モデル化すると・・・

CSPモデルの基本



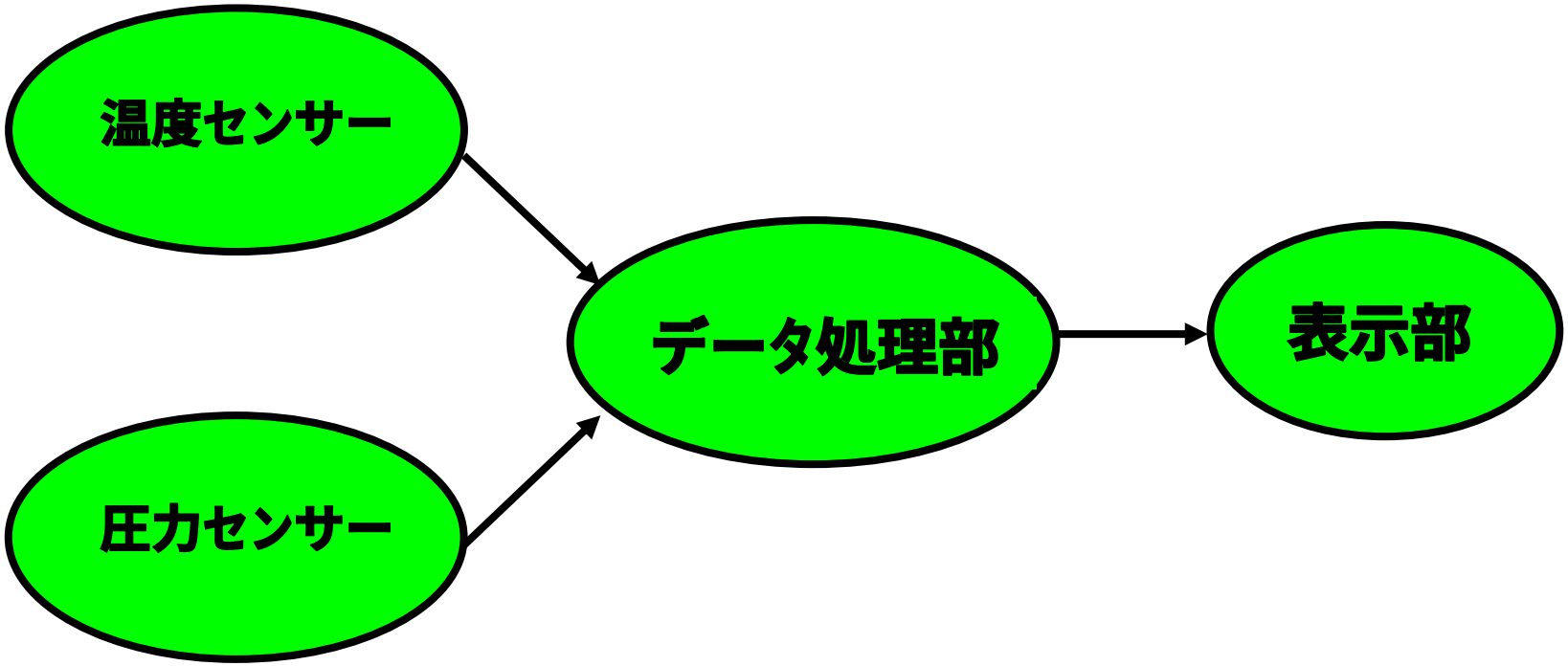
CSPモデルの基本

通信機能を持ったプログラム　　：プロセス
プロセス間を接続するデータの通信路：チャネル

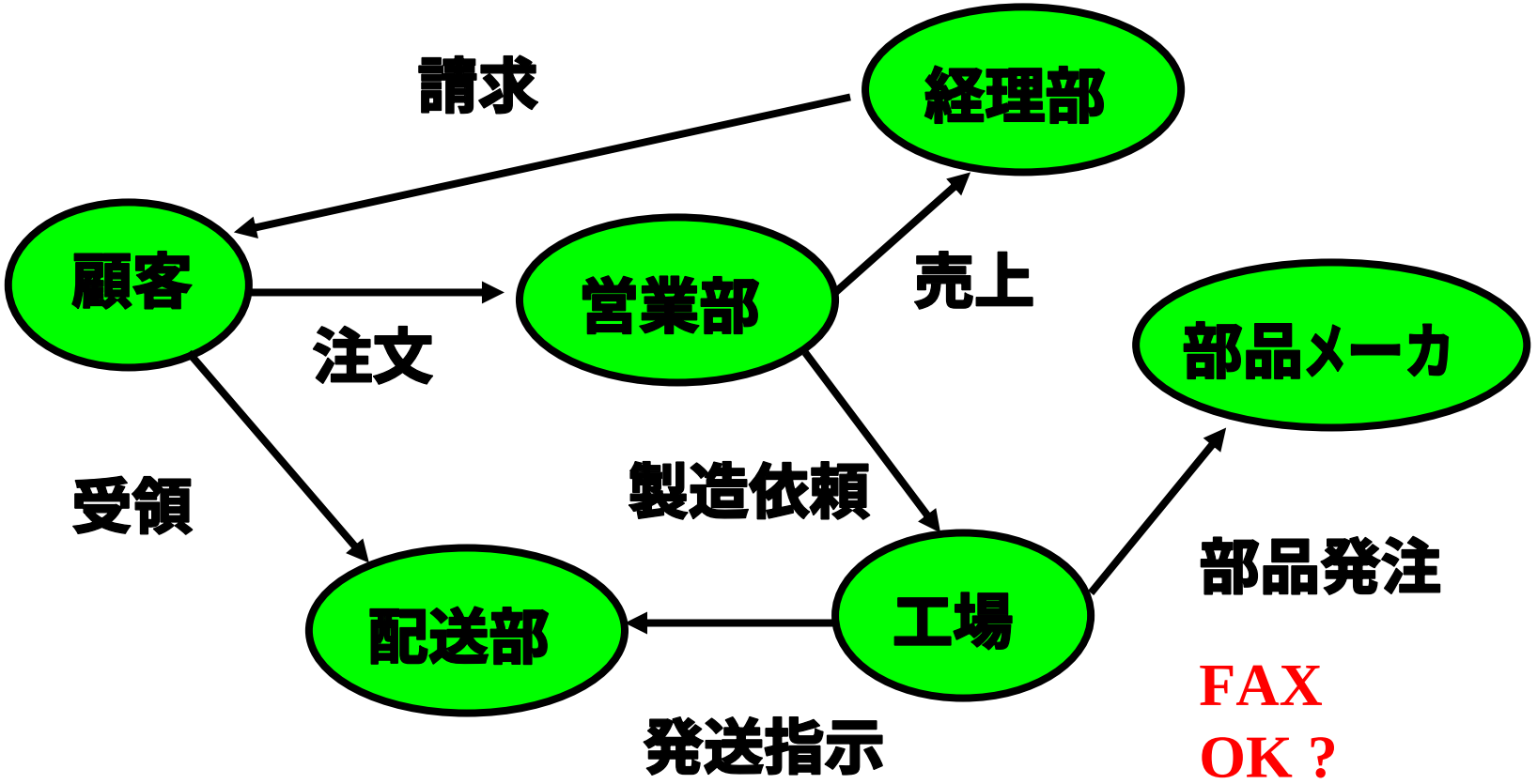
プロセス：円や楕円
チャネル：矢印

データフローダイアグラム
(DFD)

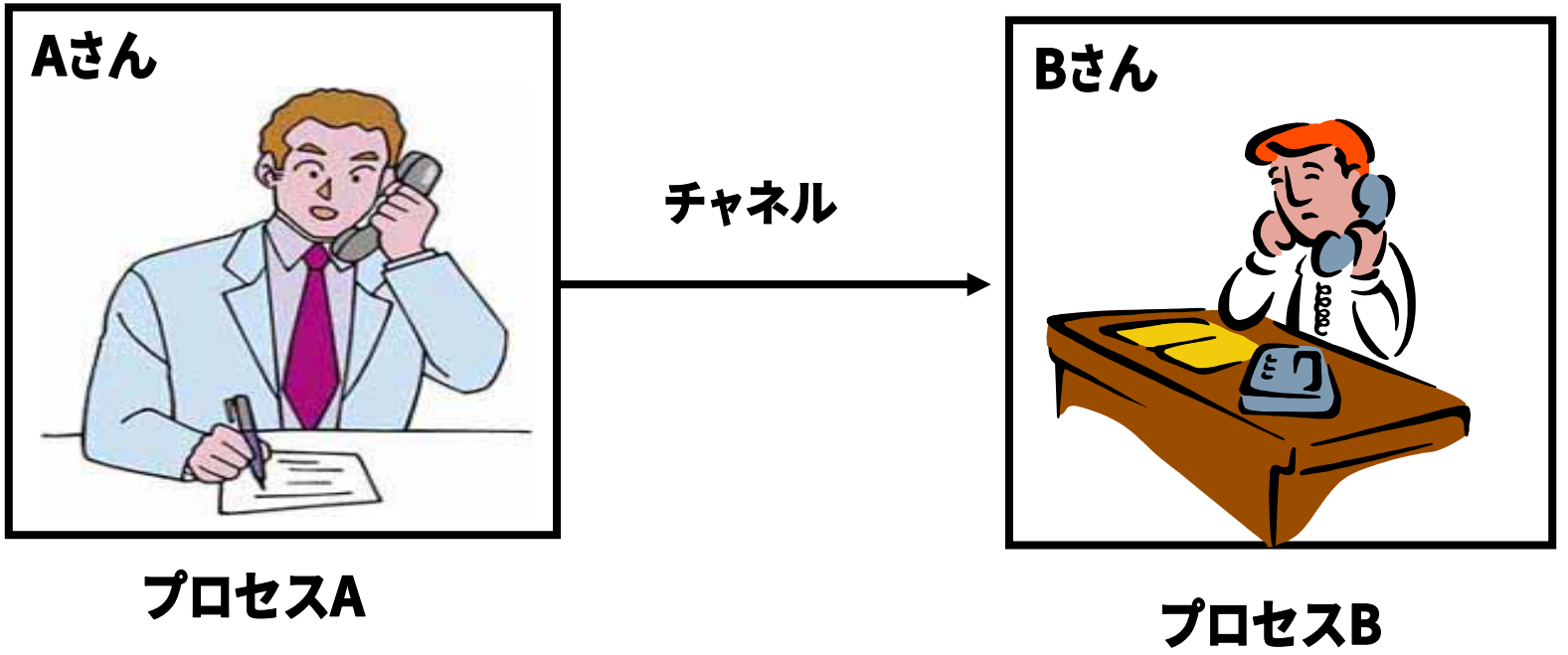
計測システムのDFD



ある会社のDFD

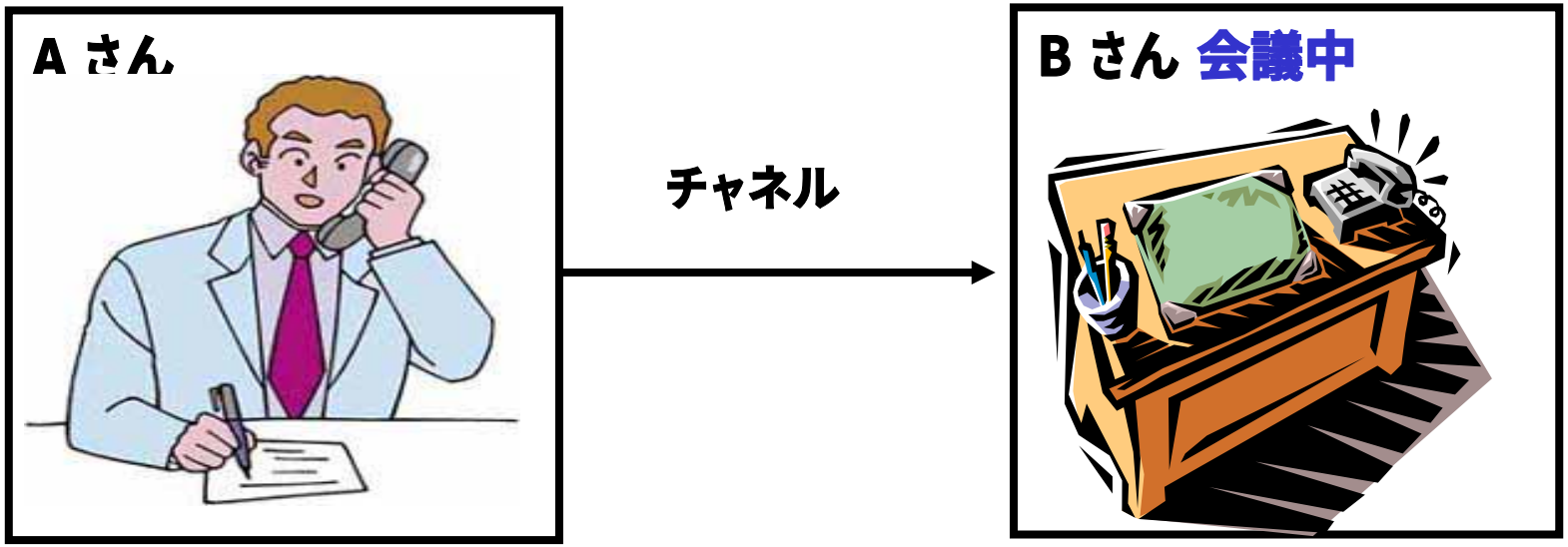


よりCSPモデルに近い例



FAXではなく電話 ！？

よりCSPモデルに近い例



相手がでない 、A さんは話せません ！！
相手が出るまで待たされる ！！

CSPモデルの特徴

1．チャネルには、バッファがない。

2．チャネルのデータの流れは、一方向

3．チャネルによる接続形態は、一対一

バッファがない。送信側は・・・

CSPモデルの基本

- 送信データは、受信側が直接受けるしかない
- 受信側がReady状態でなければ、送信側は待たされる
- 受信側がReady状態であれば（or になれば）、送信する
- 送信側が送信しているときは、受信側は受信している

同時 に動作！！

バッファがない。受信側は・・・

CSPモデルの基本

- 受信データは、送信側が直接送るしかない
- 送信側がReady状態でなければ、受信側は待たされる
- 送信側がReady状態であれば（or になれば）、受信する
- 受信側が受信しているときは、送信側は送信している

同時 に動作！！

バッファがないので、さらに

CSPモデルの基本

・送信側と受信側は、互いに独立して動作しているが、
送信と受信は、論理的に同時である

ここで同期する

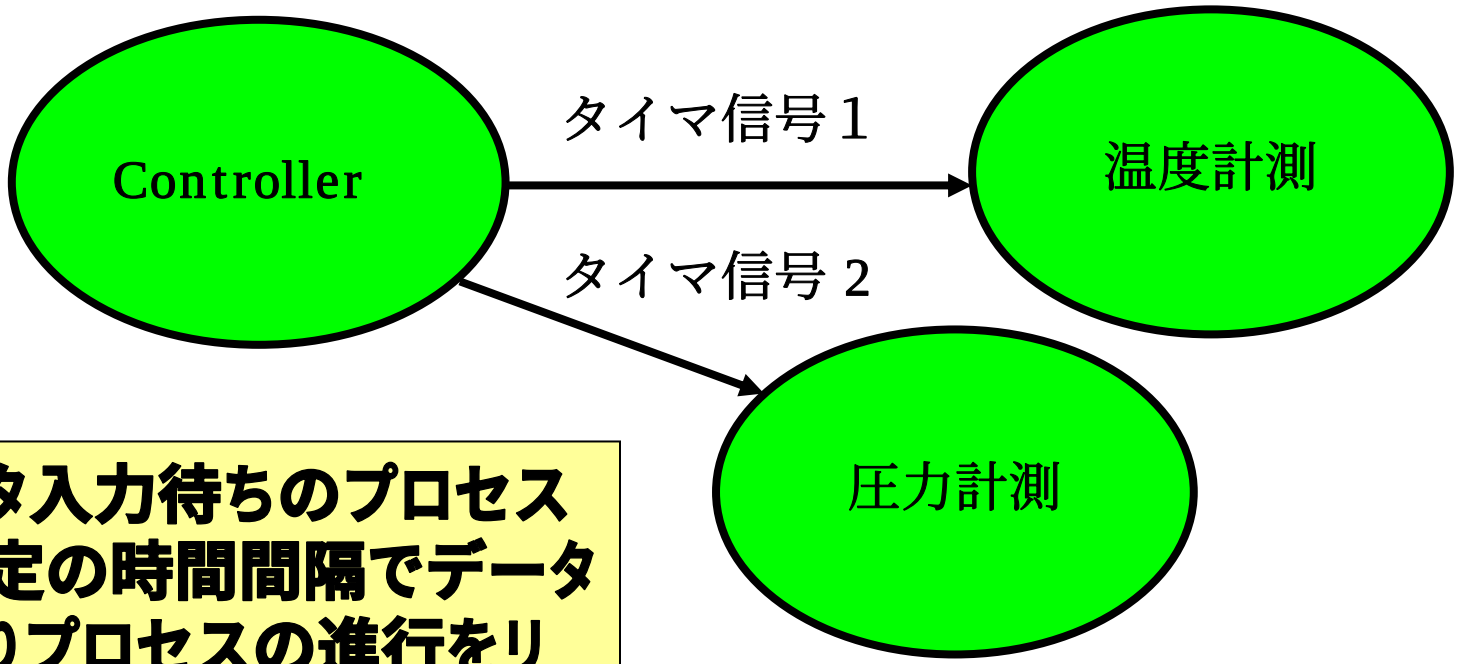
並列プロセス

バッファがないので、結局

CSPモデルの基本

・CSPモデルの並列プロセスは、チャネルを介した
メッセージパッシングにより同期がとられながら進行する
・チャネルドリブン

こんな事ができそう



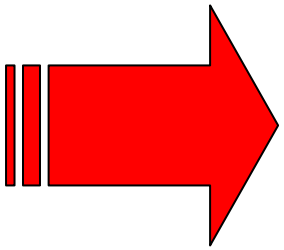
データ入力待ちのプロセスに特定の時間間隔でデータを送りプロセスの進行をリアルタイムに制御する

タイマ信号1：0.2秒間隔

タイマ信号2：0.5秒間隔

こんな事も起きます

- お互いに電話を待つ
- お互いに電話を掛ける
- いない人からの電話を待つ
- 使われていない電話に掛ける



並列プロセス特有の膠着状態

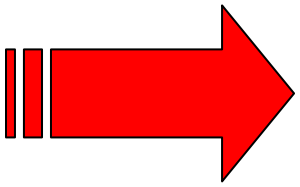
デッドロック

CSP : Communicating Sequential Processes

C.A.R. Hoareが、並列プロセスの挙動を記述・解析した。

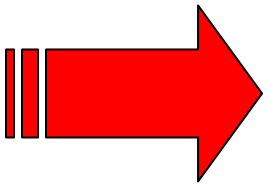
CSPをコンピュータ言語へ

CSPモデルを
コンピュータ言語へ



並列処理言語
OCCAM

OCCAMのための
並列処理用プロセッサ



並列処理プロセッサ
Transputer

OCCAM/Transputerの組み合わせにより実用的でスケーラビリティに富んだ並列処理システムが出現した

CSP for JAVA

最も簡単なCSPモデル

チャネルが1本、プロセスが2つ

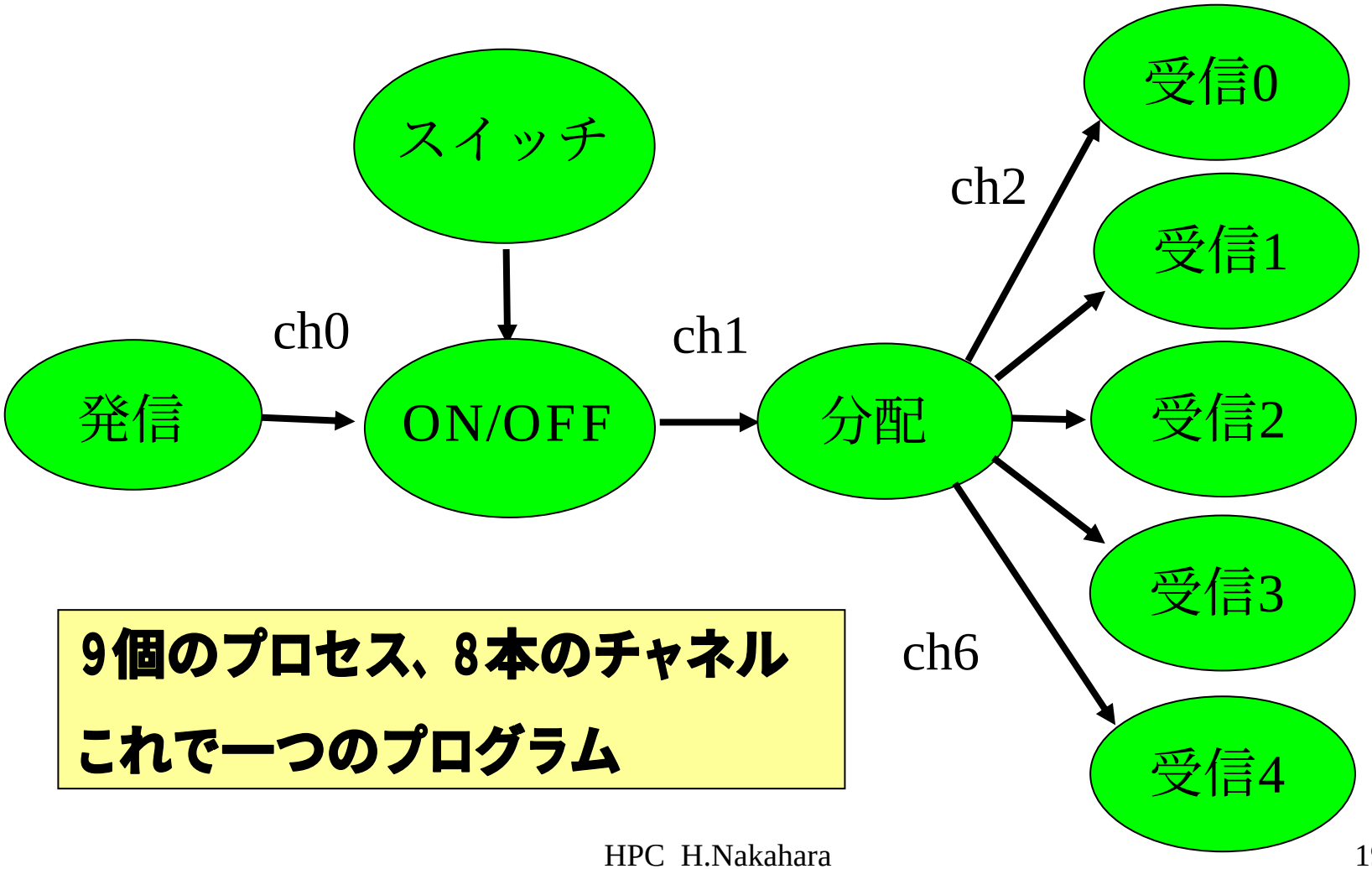
とは言うもののJAVAで・・・

- 簡単に記述できるのか？
- 正しく動作するのか？

CSP for JAVA

JCSP

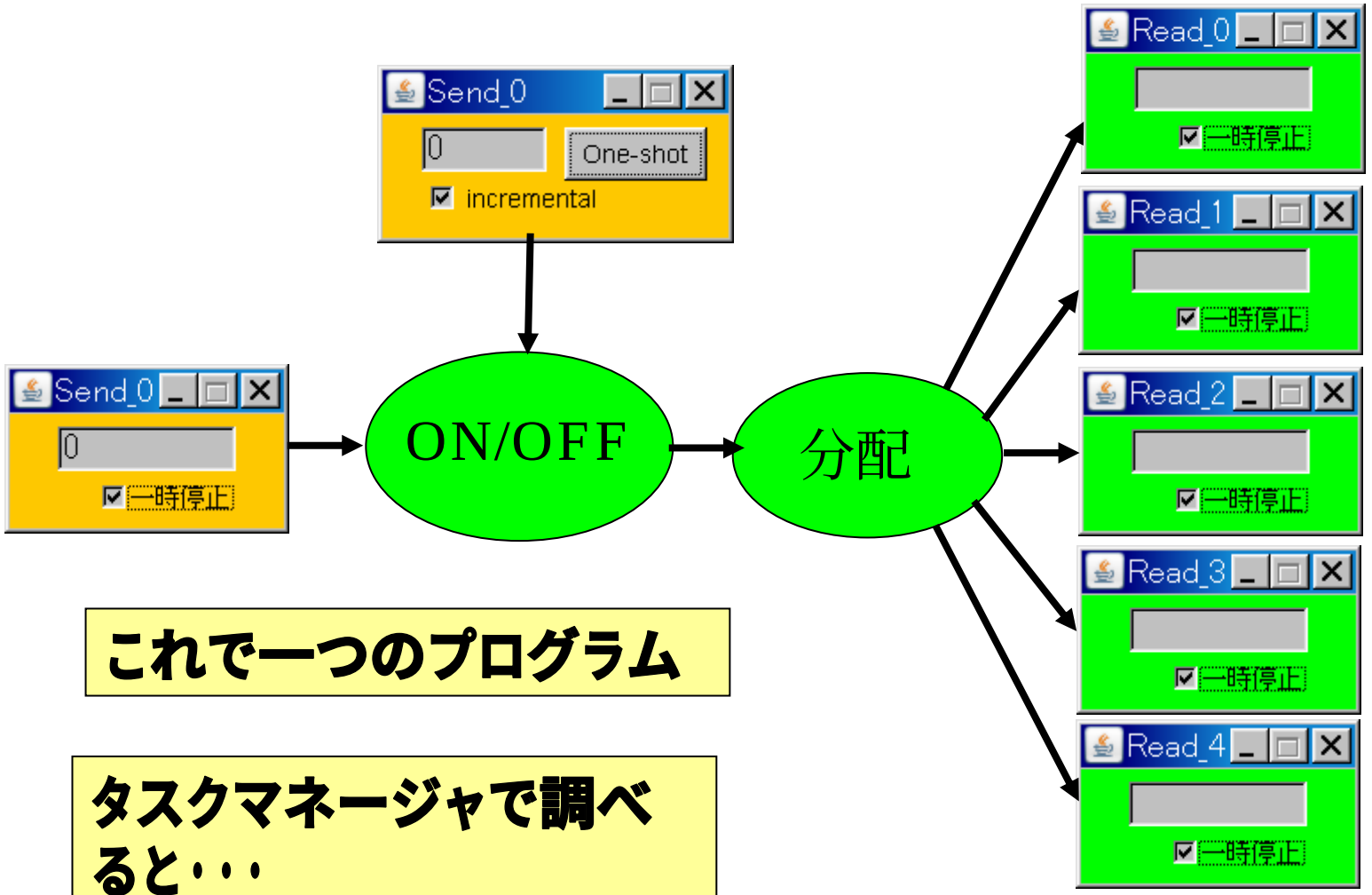
プログラム例



9個のプロセス、8本のチャネル

これで一つのプログラム

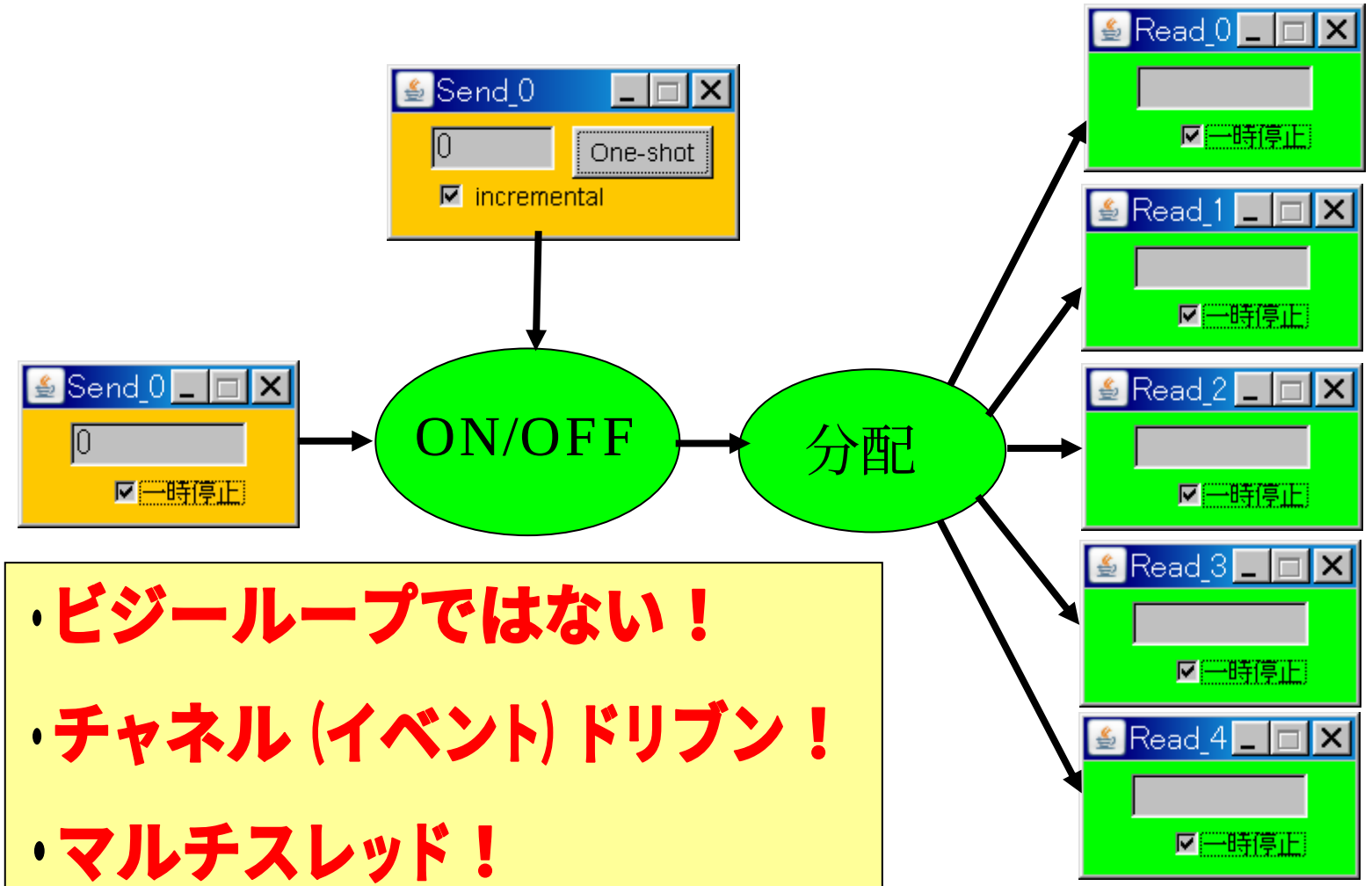
CSPモデルプログラム



これで一つのプログラム

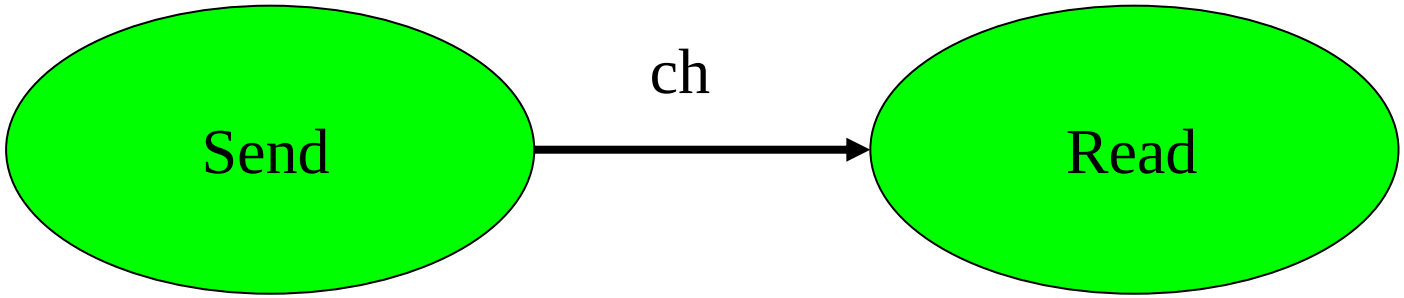
タスクマネージャで調べると・・・

CSPモデルプログラム



- ビジーループではない！
- チャネル（イベント）ドリブン！
- マルチスレッド！

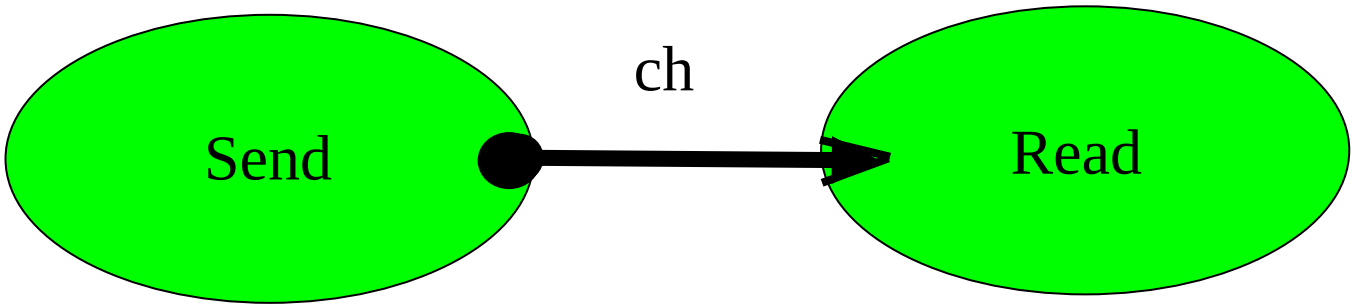
CSPモデルプログラムの基本



チャネルに整数を出力

チャネルから整数を入力し、表示する

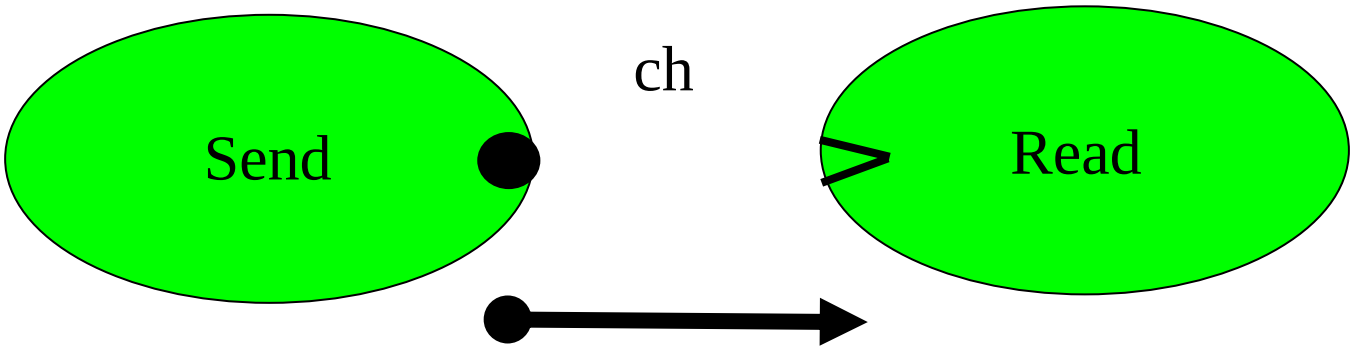
JCSP流で解釈すると



チャネルに整数を出力

チャネルから整数を入力し、表示する

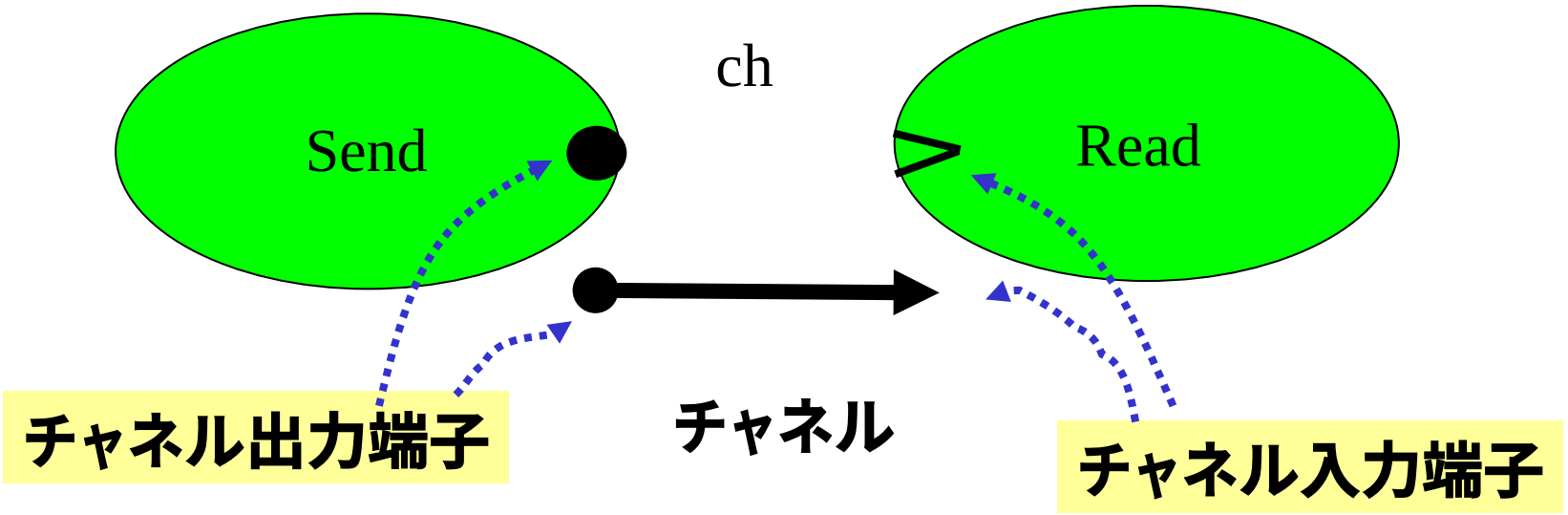
JCSP流で解釈すると



チャネルに整数を出力

チャネルから整数を入力し、表示する

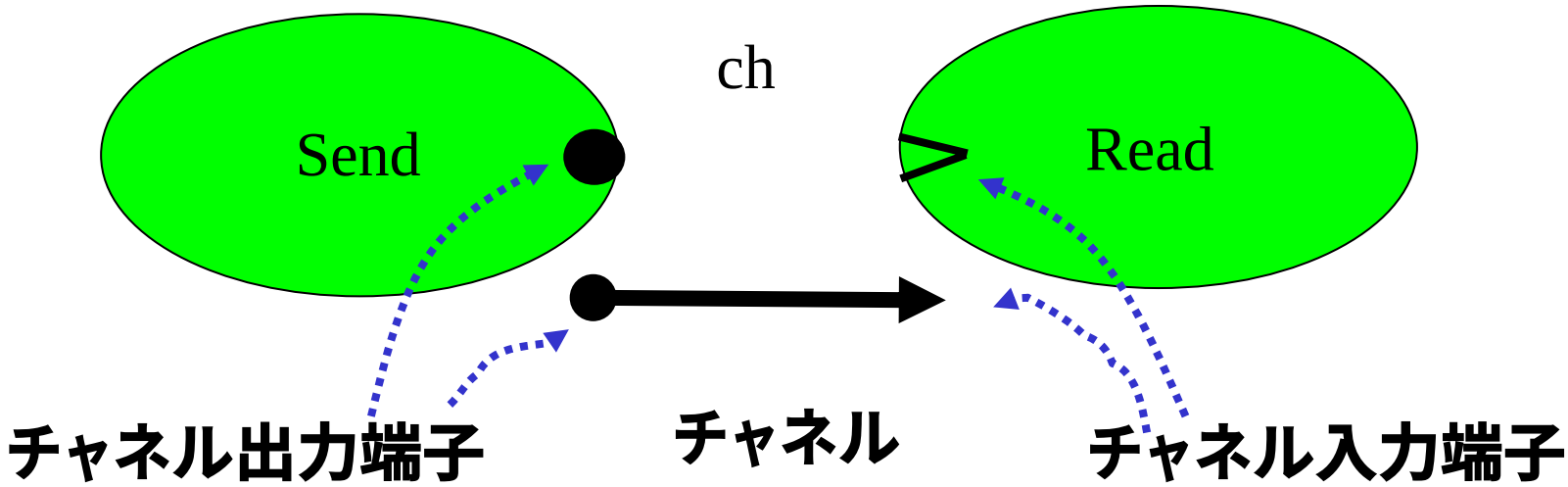
JCSPでは・・・



Sendプロセスは、チャネル出力端子を引数に持つ

Readプロセスは、チャネル入力端子を引数に持つ

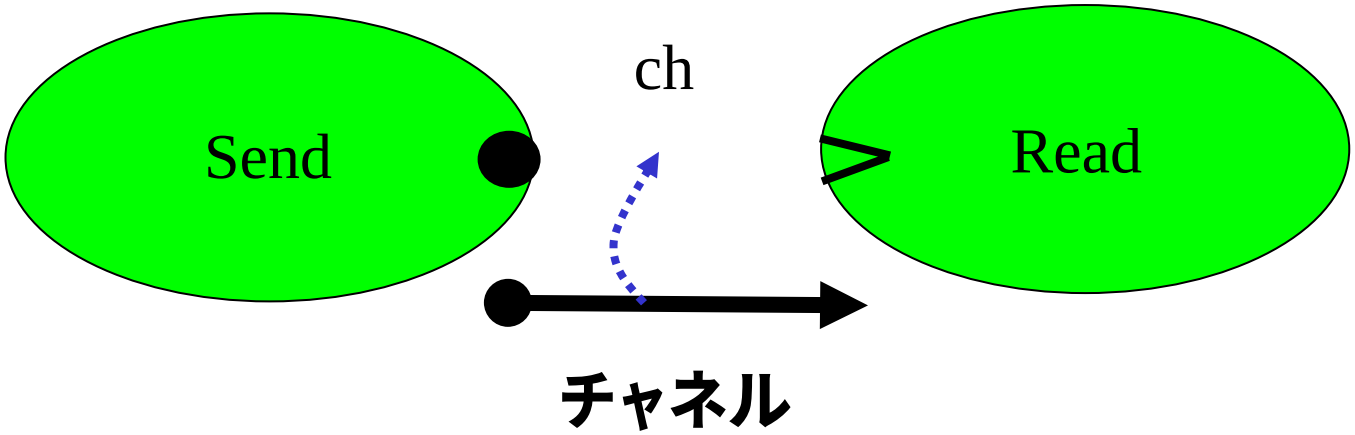
JCSPのプロセス



Send**プロセス**は、チャネル出力端子を引数に持つ**クラス（オブジェクト）**

Read**プロセス**は、チャネル入力端子を引数に持つ**クラス（オブジェクト）**

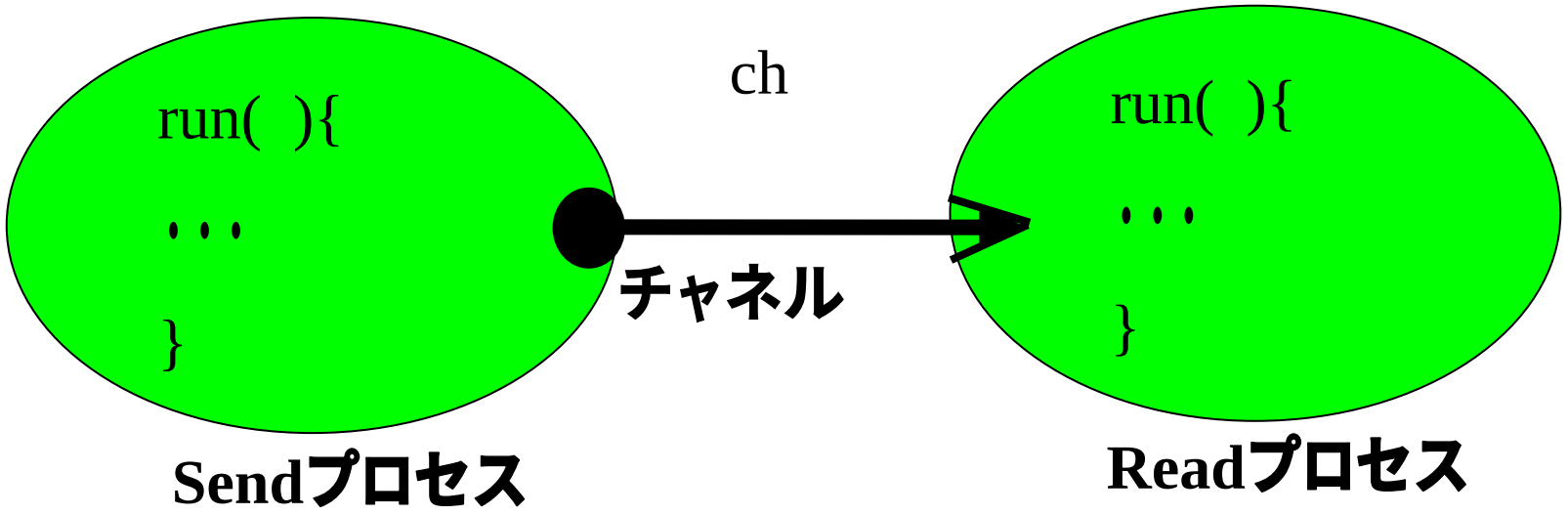
プロセス間の接続



メインプログラムでは、

チャネルを生成して、その入出力端子をそれぞれのプロセスに渡すことによってプロセス間のチャネル接続を実現している。

プロセスの処理内容は、・・・



プロセスは、唯一の実行メソッドrun( )を持つ。

処理する内容（プログラム）を、全てrun( )メソッドに記述する。

そして、必要に応じてチャネル端子からデータを送受信する。

Sendプロセスの例

```
import org.jcsp.lang.*;      //ver1.1
class Send implements CSProcess{
      private ChannelOutput out;
      public Send(ChannelOutput out){
           this.out=out;    }
     public void run( ) {
       int i=0;
       while (true){
             i=i+1;
            out.write (new Integer ( i ) ) ;   }
 }
}
```

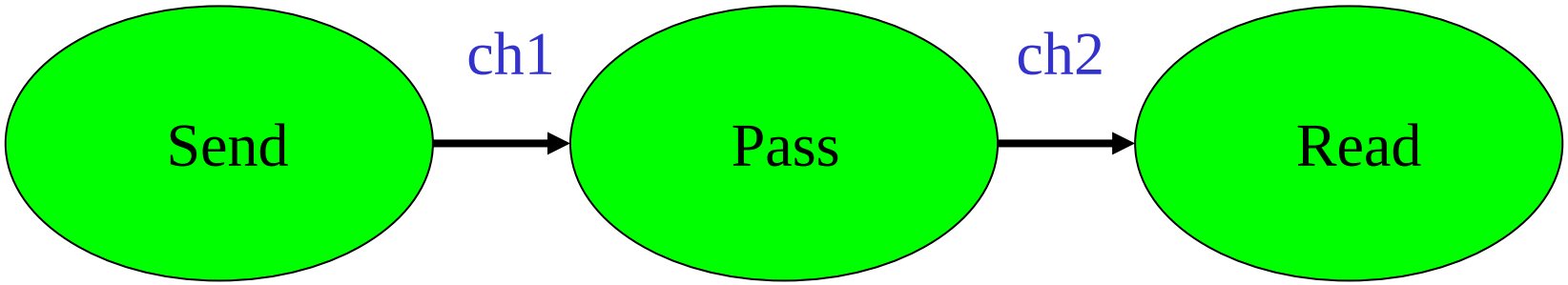
Readプロセスの例

```
import org.jcsp.lang.*;      //ver1.1
class Read implements CSProcess{
    private ChannelInput in;
    public Read(ChannelInput in){
          this.in=in;   }
    public void run( ) {
         Integer i;
        while (true){
               i=(Integer)in.read();
               System.out.println( "data="+i.intValue( ) ) ;   }
  }
}
```

メインプログラムの例

```
import org.jcsp.lang.*;      //ver1.1
public class ParaMain{
   public static void main( String[] arg){
     One2OneChannel chan=Channel.one2one ( ) ;  // ver1.1
      Send send=new Send( chan.out( ) );
      Read read=new Read( chan.in( ) );
     CSProcess[] csp=new CSProcess[]{send,read};
     Parallel Para=new Parallel(csp);
     Para.run();    //パラレルプロセスの実行
   }
}
```

② SendPassReadプログラム



チャネルに整数を10個出力

チャネルから入力したデータを即出力する

チャネルから整数を入力し、表示する

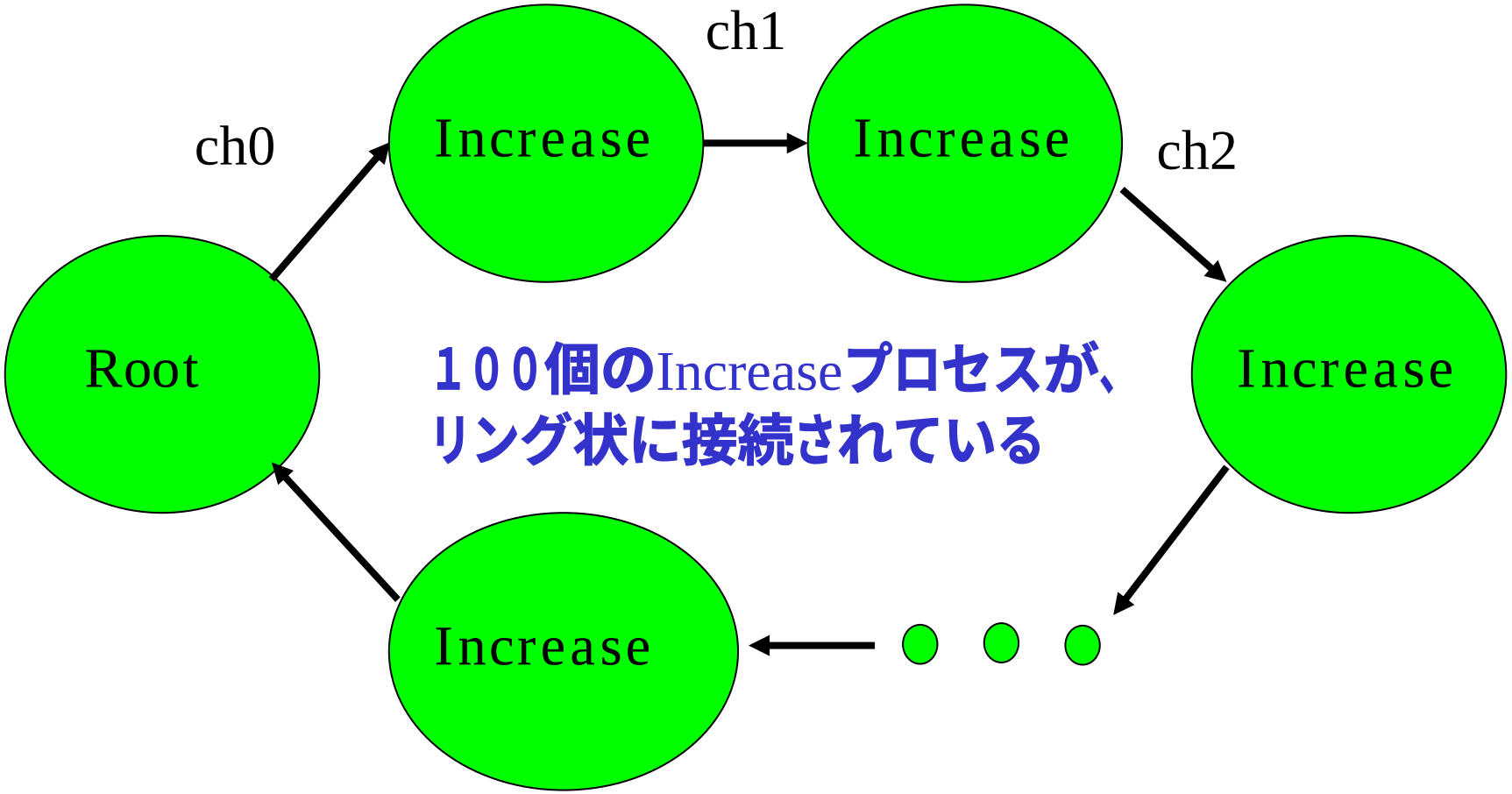
Passプロセス

```
class Pass implements CSProcess{
    private ChannelInput in;   private ChannelOutput out;
    public Pass(ChannelInput inch,ChannelOutput outch){
        this.in=inch;
        this.out=outch;    }
    public void run(){
       Integer N;       int i;
       for(i=0;i<10;i=i+1){
           N=(Integer)in.read();
           out.write(N);   }
    }
}
```

SendPassReadのメイン

```
public static void main( String[] arg){
    One2OneChannel ch1=Channel.one2one ( ) ;  // ver1.1
    One2OneChannel ch2=Channel.one2one();
     Send send=new Send(ch1.out());
     Pass pass=new Pass(ch1.in(),ch2.out());
     Read read=new Read(ch2.in());
     CSProcess[] csp=new CSProcess[]{send,read,pass};
     Parallel Para=new Parallel(csp);
    Para.run();
 }
```

③リング状の並列プロセス



Increaseプロセス

```
class Increase implements CSProcess{
     private ChannelInput in ;    private ChannelOutput out;
     public Increase(ChannelInput in,ChannelOutput out){
            this.in=in;      this.out=out;     }
    public void run(){
        int  I , d;          Integer ii;
        for (i=0;i<10;i=i+1){
              ii=(Integer)in.read();
              d=ii.intValue()+1;
              out.write( new Integer ( d ) ) ;    }
   }
}
```

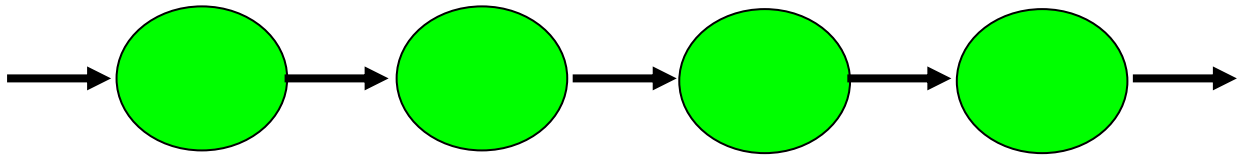
Rootプロセス

```
class Root implements CSProcess{
    private ChannelOutput out;   private ChannelInput in;
    public Root(ChannelInput in,ChannelOutput out){
                        this.out=out;  this.in=in;     }
     public void run(){
         Integer ii;  int i=0;
         for (i=0;i<10;i=i+1){
                ii=new Integer(i);
                out.write(ii);
                ii=(Integer)in.read();
               System.out.println( "data="+ii.intValue());   }
  }
}
```

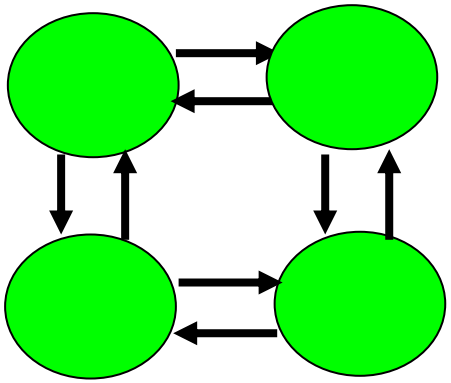
Ring100のメイン

```
public static void main(String[] args){
    int N=100;                    // Increaseプロセスの数
   One2OneChannel[] ch=Channel.one2oneArray(N+1);
    CSProcess[] CspRing=new CSProcess[N+1];
    int i;
    CspRing[0]=new Root(ch[N].in(),ch[0].out());  // ルート
    for (i=1;i<=N;i=i+1){
       CspRing[i]= new Increase(ch[i-1].in(),ch[i].out());      }
    Parallel Ring=new Parallel(CspRing);
    Ring.run();
   }
}
```

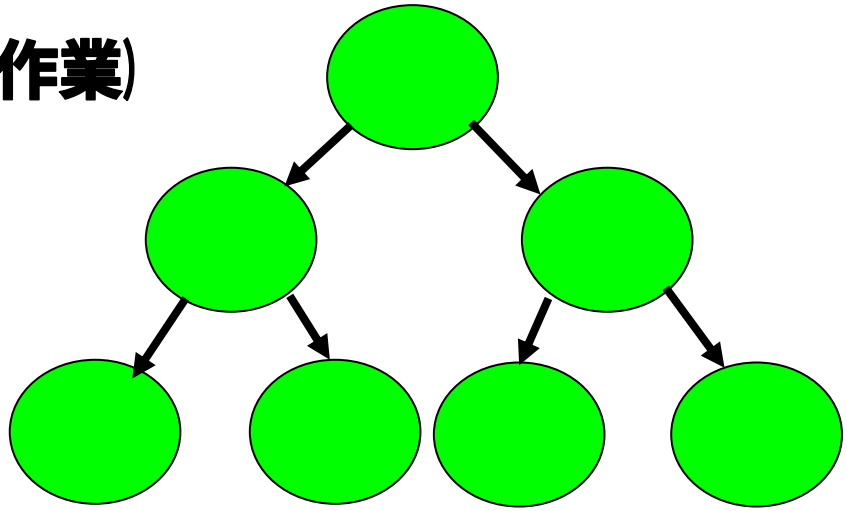
色々なトポロジー



パイプライン (流れ作業)

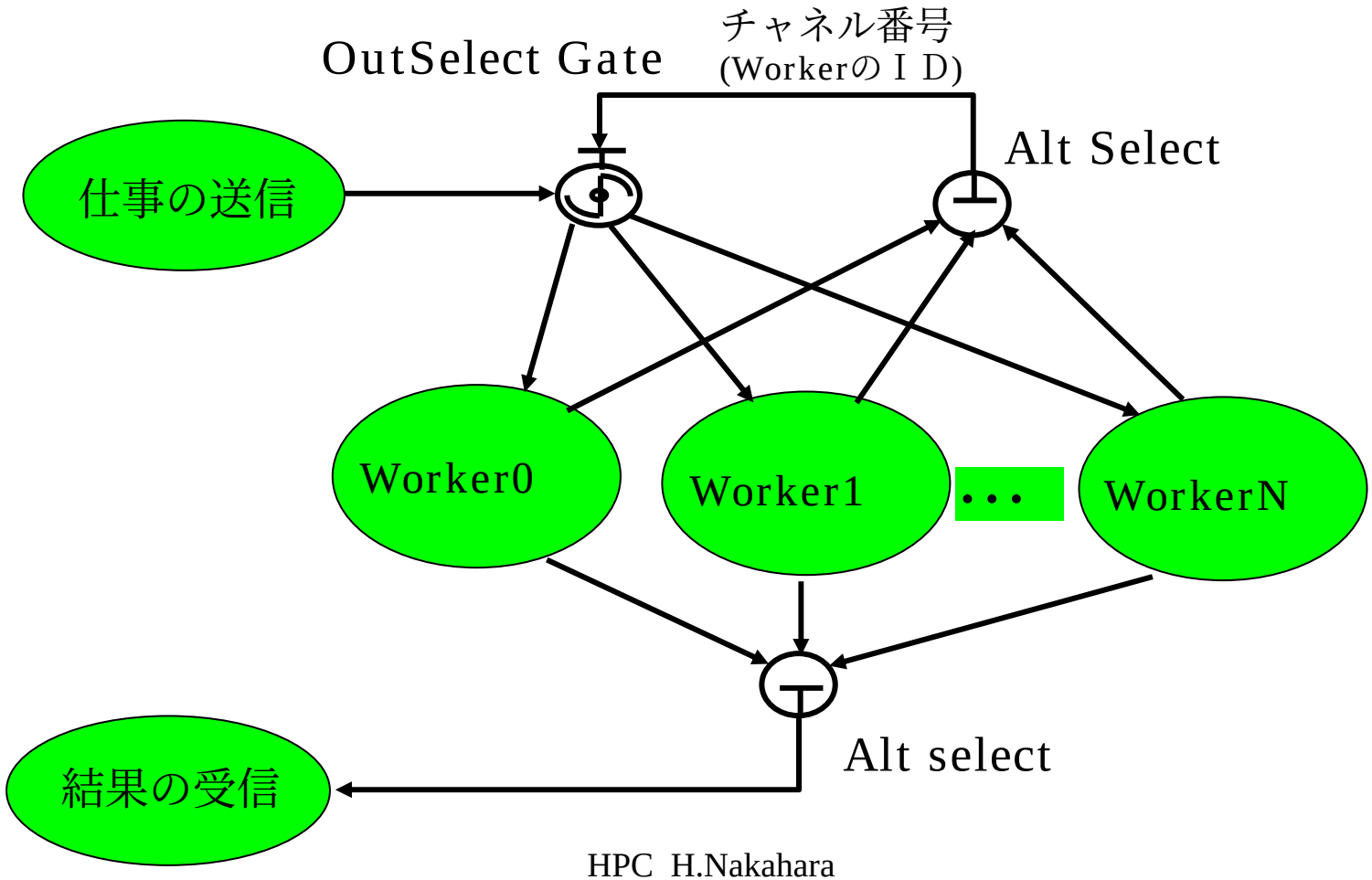


メッシュ (領域分割)

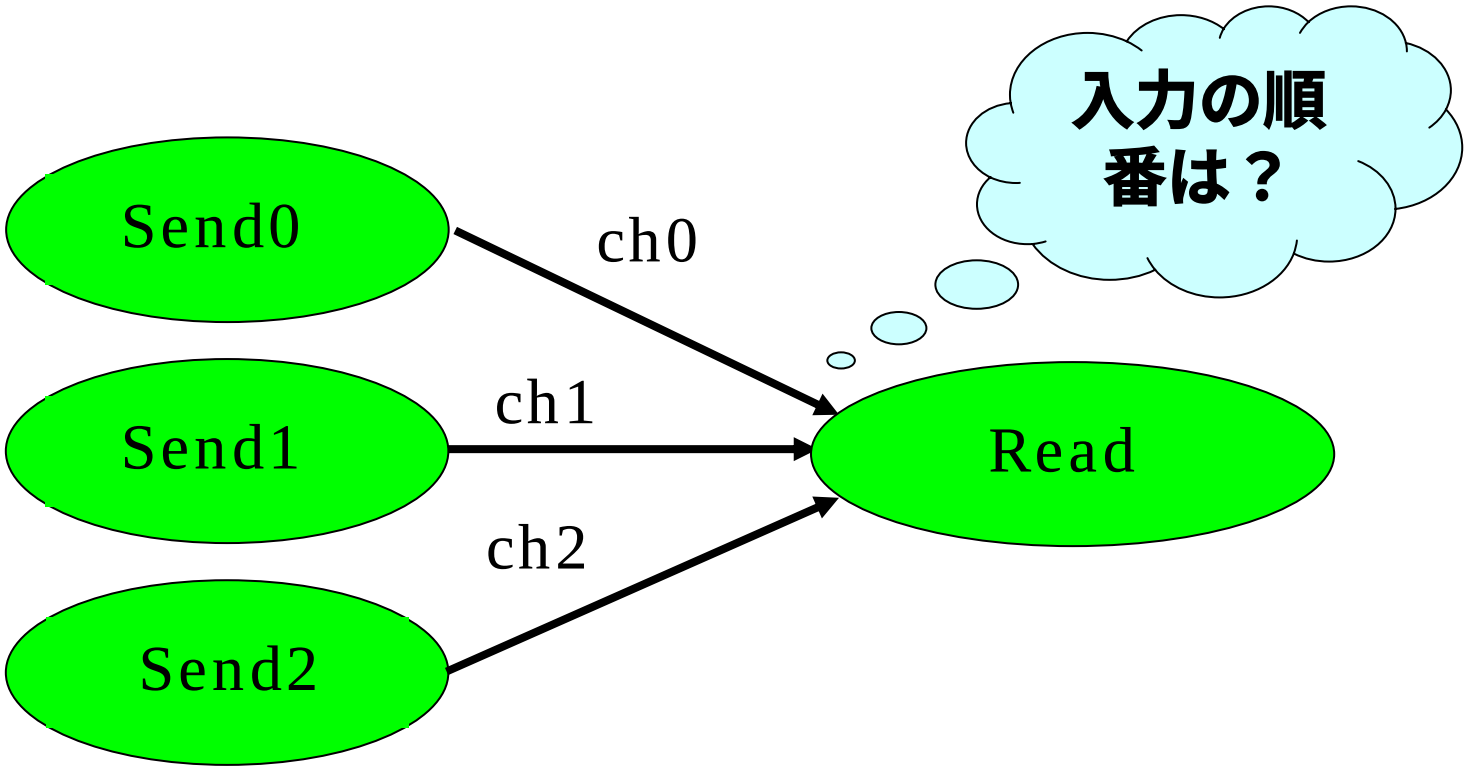


ツリー

プロセッサファーム


チャネル番号
(WorkerのＩＤ)
OutSelect Gate
Alt Select
仕事の送信
Worker0
Worker1
・・・
WorkerN
Alt select
結果の受信

④ Proc3to1

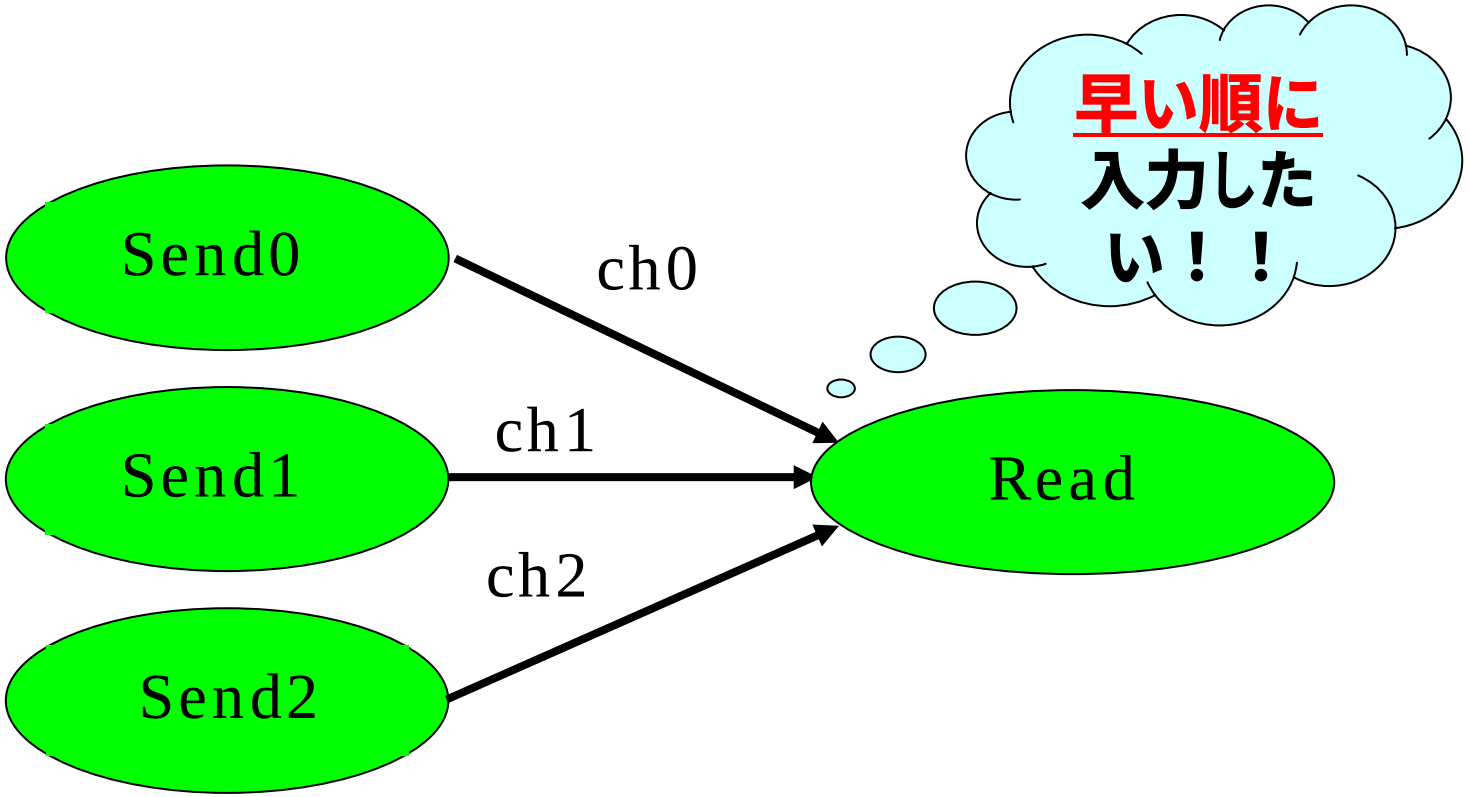


3対1接続の例

④ Proc3to1のReadプロセス

```
public class Read implements CSProcess{
   ・・・一部省略
 public void run(){
     Integer ii;    int i;
     for (i=0;i<10;i=i+1){
         ii=(Integer)in0.read();
           System.out.println("   チャネル0  data= "+ii.intValue() );
         ii=(Integer)in1.read();
           System.out.println("   チャネル1 data= "+ii.intValue() );
         ii=(Integer)in2.read();
           System.out.println("   チャネル2 data= "+ii.intValue() );  }
  }
```

⑤ Alternativeクラス

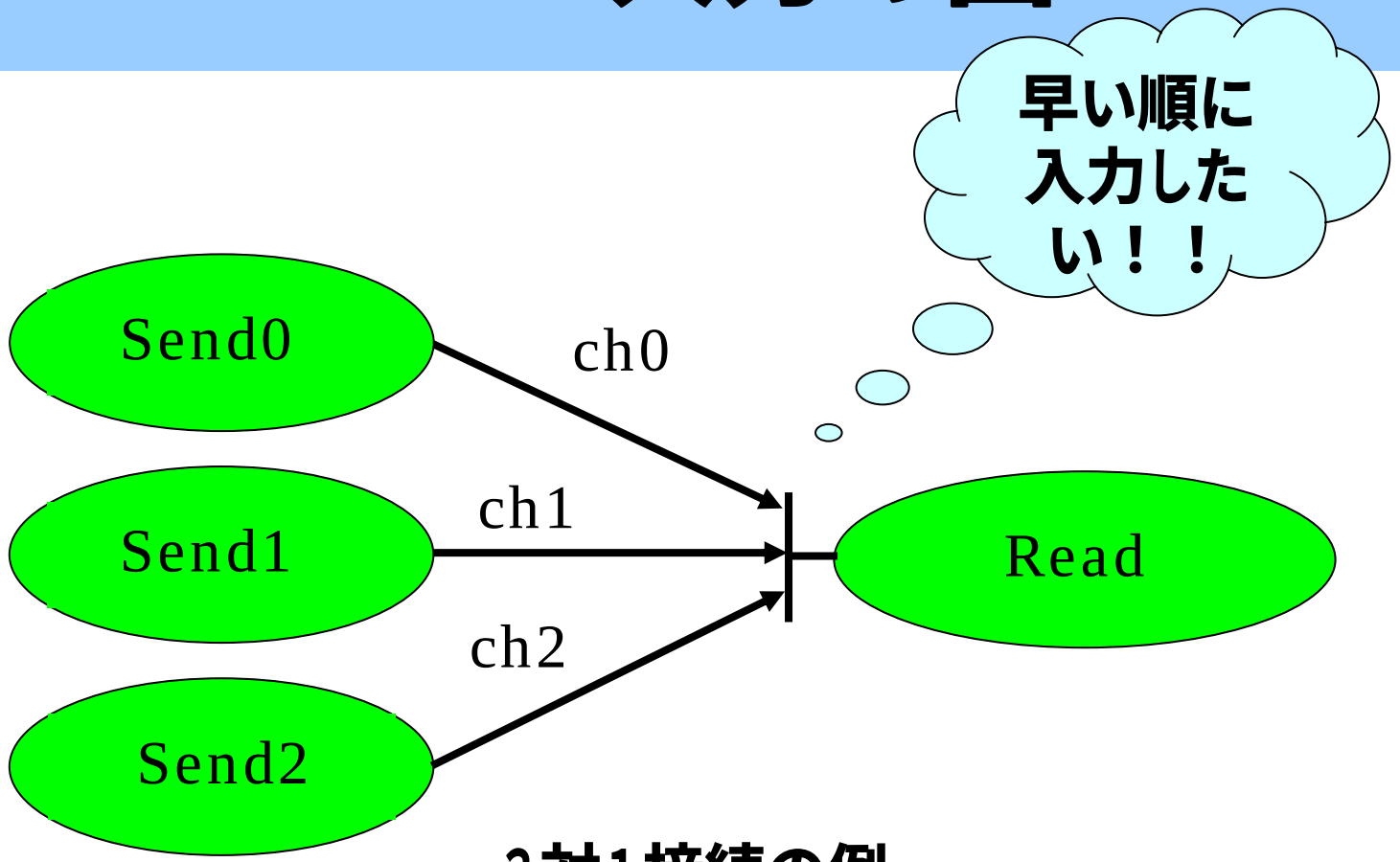


3対1接続の例

Alternativeクラスの例

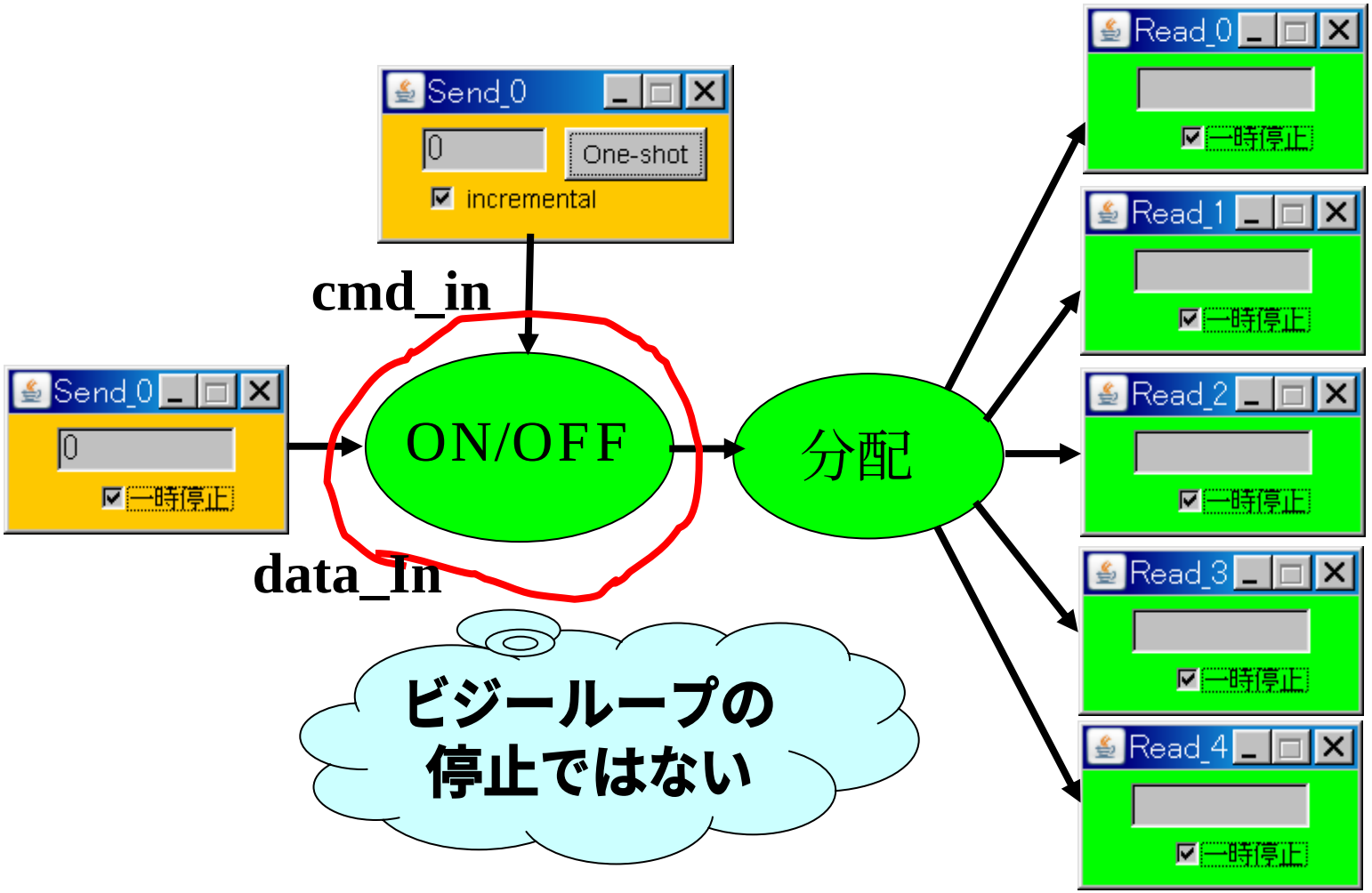
```
class AltselectRead implements CSProcess{
 ・・・一部省略
public void run(){
  String str ; int i,index;
  Guard[] gd=new  Guard[]{(Guard)in0,(Guard)in1,(Guard)in2};
  Alternative alt=new Alternative(gd);
  for (i=0;i<30;i=i+1){
    index=alt.select();
    switch(index){
          case 0 :  str =(String)in0.read() ;   break;
          case 1 :  str=(String)in1.read() ;   break;
          case 2 :  str=(String)in2.read() ;  break;  }
  }
```

ALT入力の図



3対1接続の例

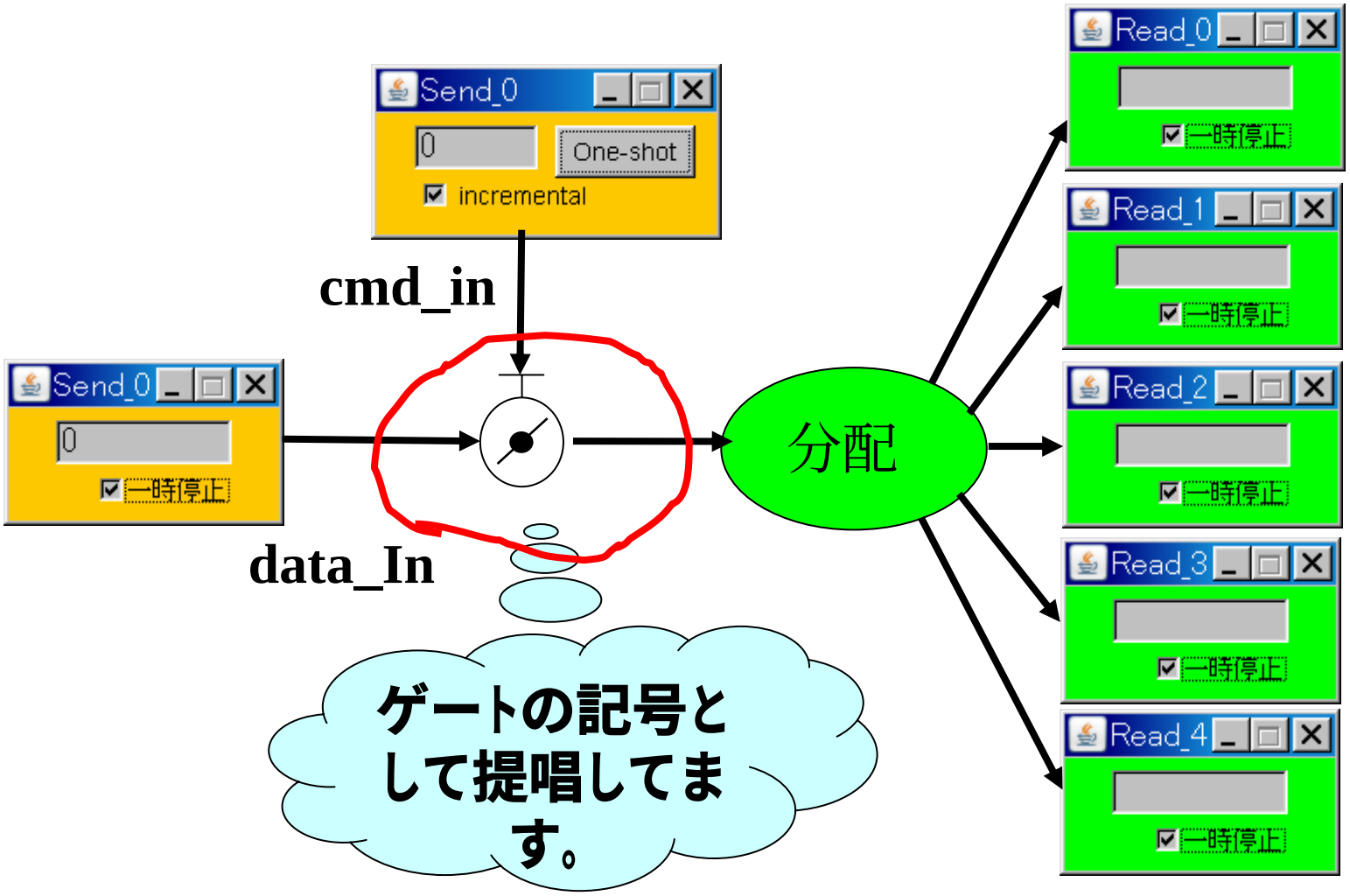
ゲートプロセス


Send_0
0
One-shot
incremental
cmd_in
Send_0
0
一時停止
ON/OFF
分配
data_In
ビジーループの
停止ではない
Read_0
一時停止
Read_1
一時停止
Read_2
一時停止
Read_3
一時停止
Read_4
一時停止

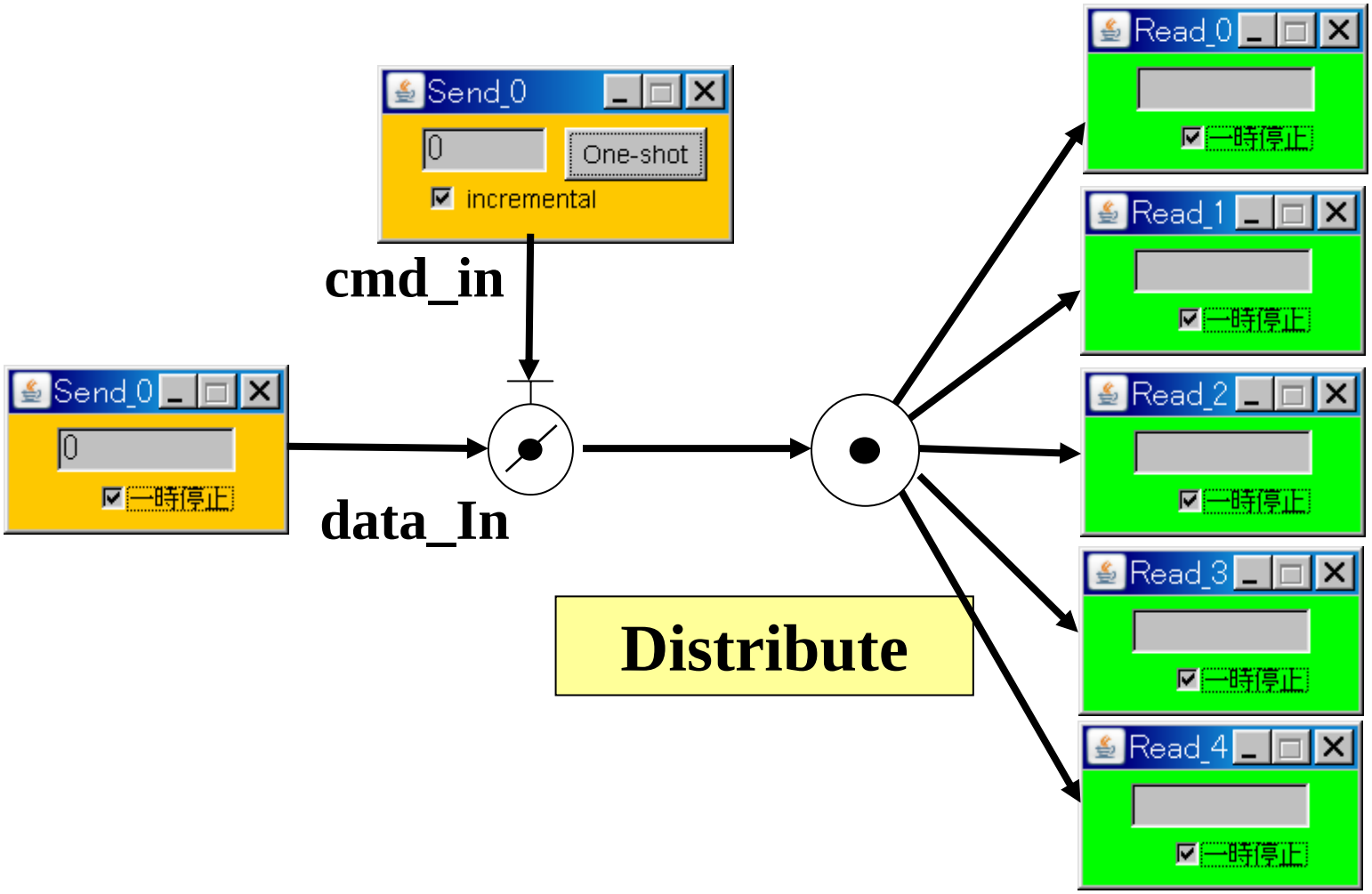
ゲートプロセスのrunメソッド

```
public void run(){
     Guard[] chnG=new Guard[]{cmdin,datain};
     Alternative alt=new Alternative(chnG);
     while(true){
        switch(alt.priSelect()){
           case 0:  cmdin.read();  // 一回目（停止）の受信
                    cmdin.read();  // 二回目（開始）の受信を待つ
                     break;
           case 1:  dataout.write(datain.read()) ;  //通過！！
                    break;
        }
 }
```

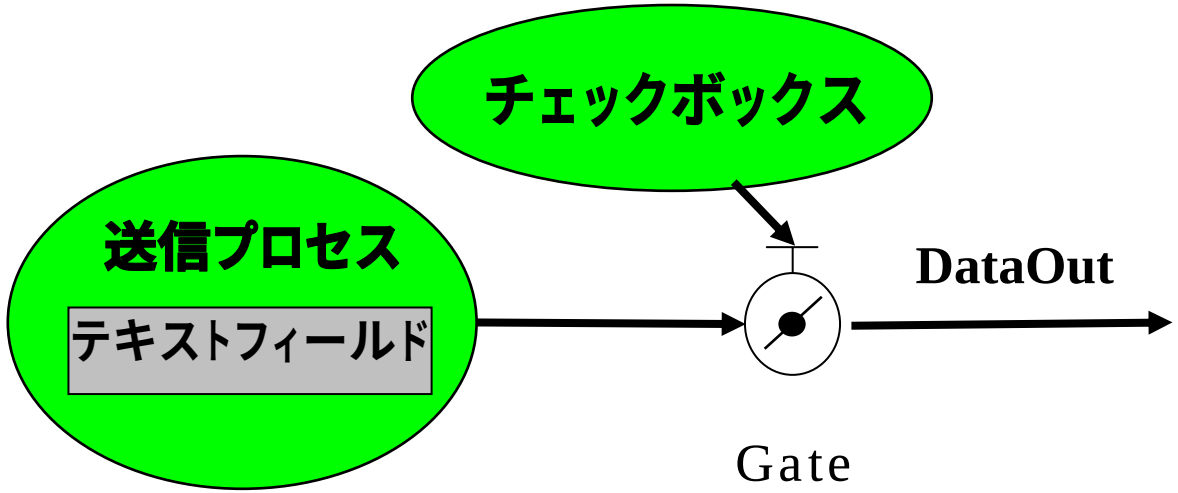
ゲートプロセス


Send_0
0
One-shot
incremental
cmd_in
Send_0
0
一時停止
data_In
分配
Read_0
一時停止
Read_1
一時停止
Read_2
一時停止
Read_3
一時停止
Read_4
一時停止
ゲートの記号として提唱してます。

Gate、Distribute

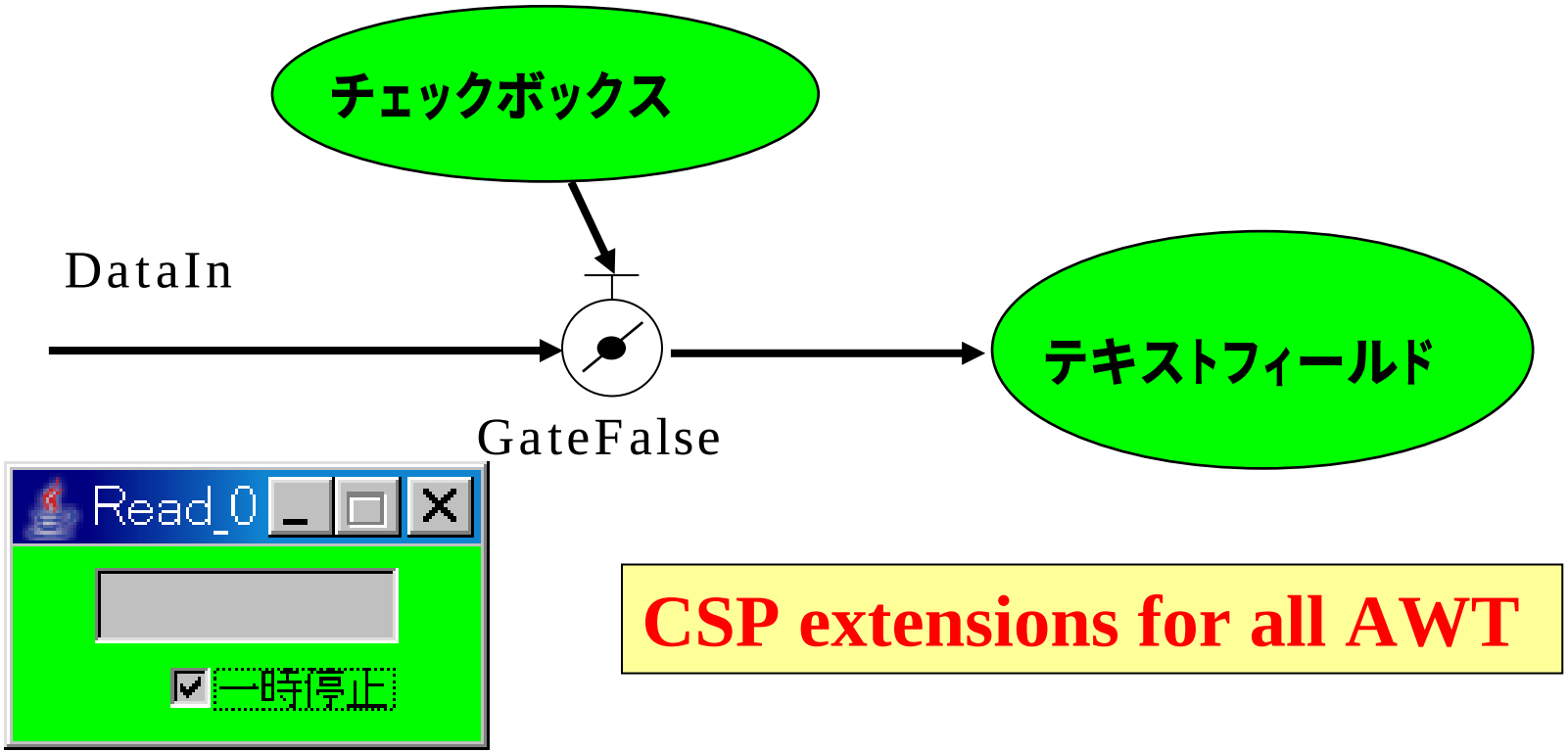


SendFormの構成例
ActiveCheckbox etc



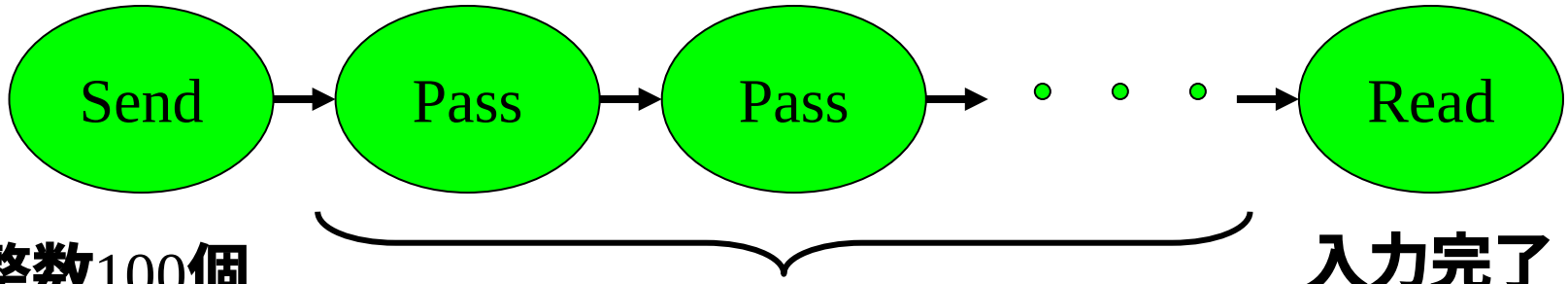
CSP extensions for all AWT

ReadFormの構成例
ActiveTextField、・・・ etc



CSP extensions for all AWT

転送時間の測定例



整数100個

開始時刻 : st

1000個のPassプロセス

入力完了時刻 : et

転送時間=(et - st) /データ数/プロセス数

=(et - st) /100/1000

データサイズに依存しない

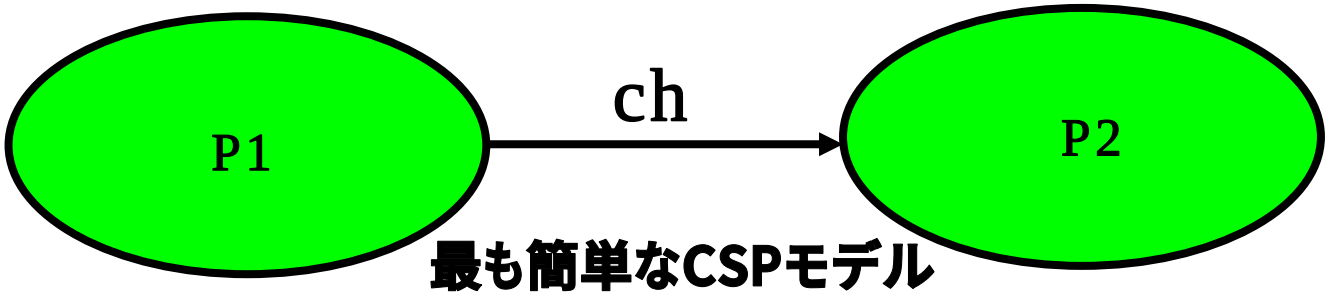
リファレンスの転送

Pentium4 : 2000MHz : JAVA1.5の場合、12(μs)

ネットワークで実行

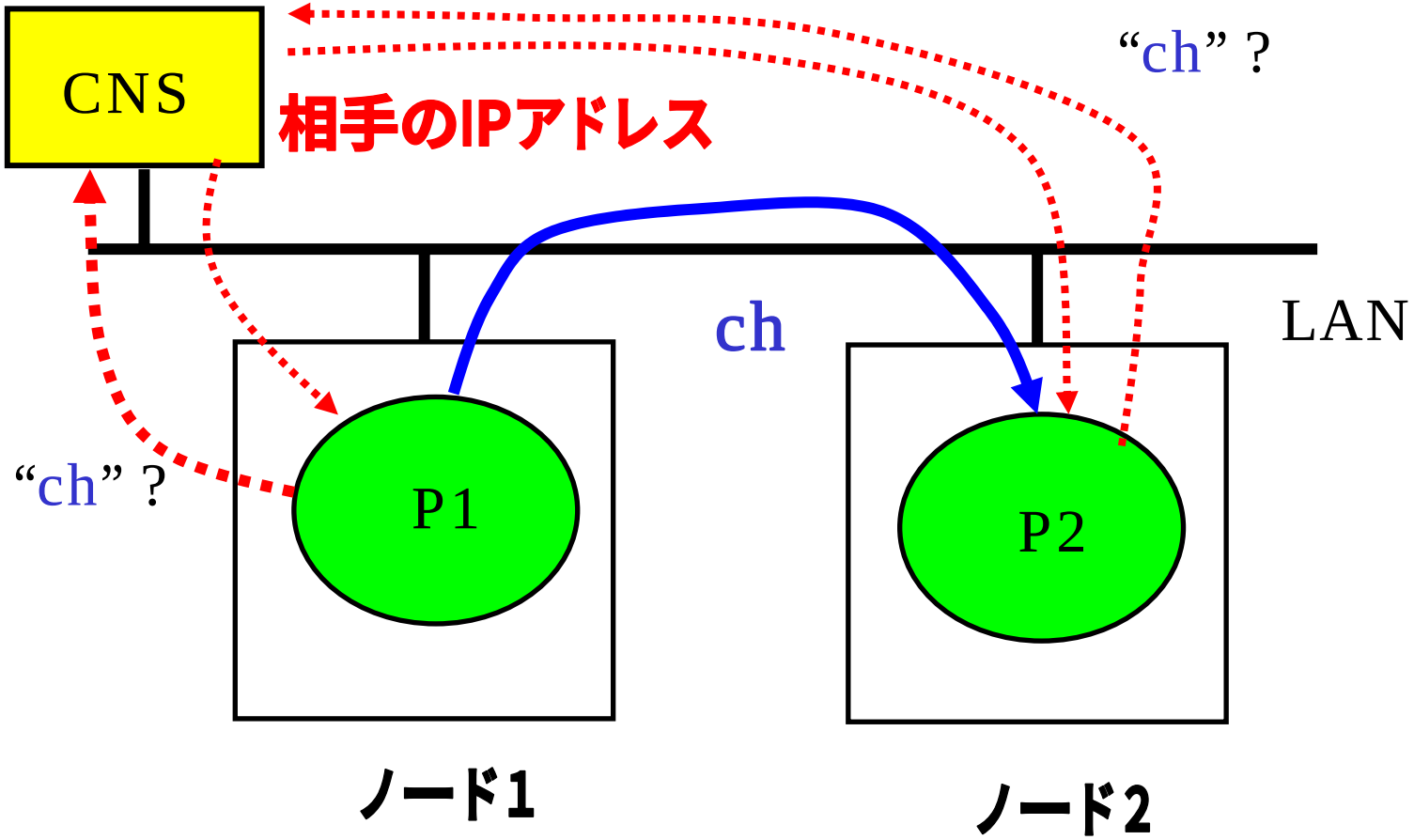
特　長

- チャネルネームサーバ（CNS）の利用
- プロセス間を、仮想チャネルで接続
- プロセスを任意のノードで起動できる（分散配置）
- プロセスのプログラムを変更する必要はない　！　！

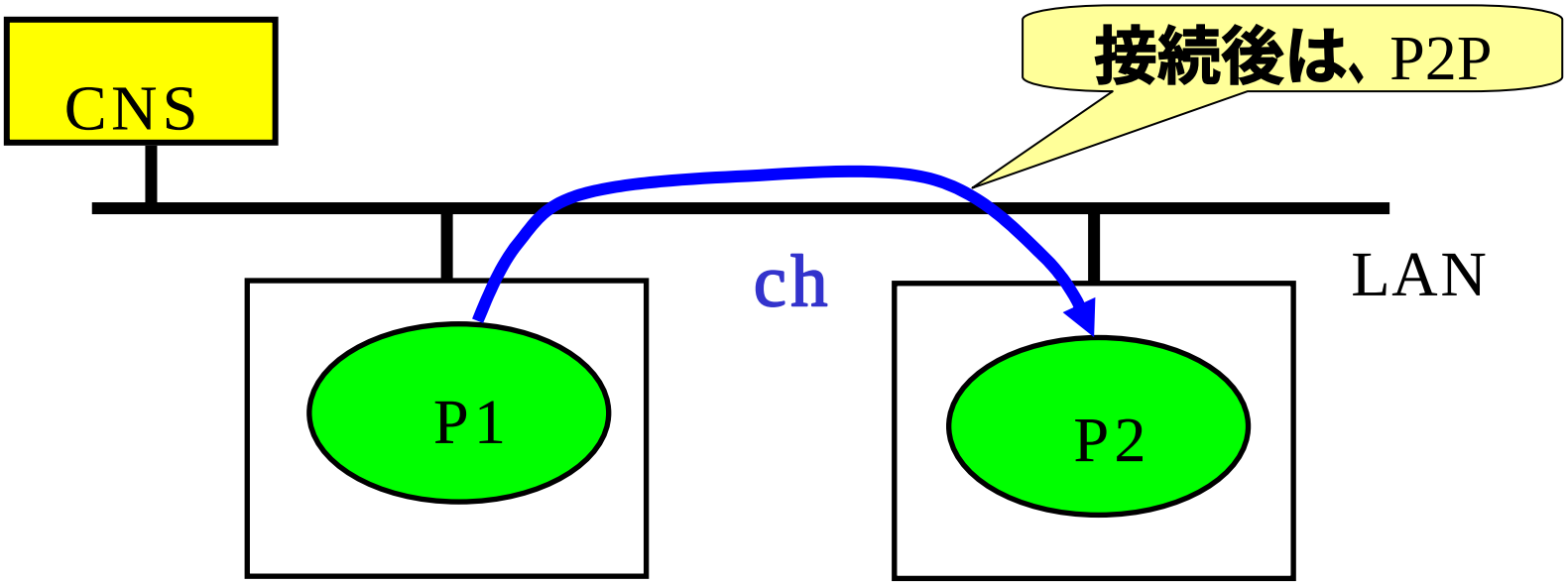


最も簡単なCSPモデル

チャネルネームサーバ(CNS)



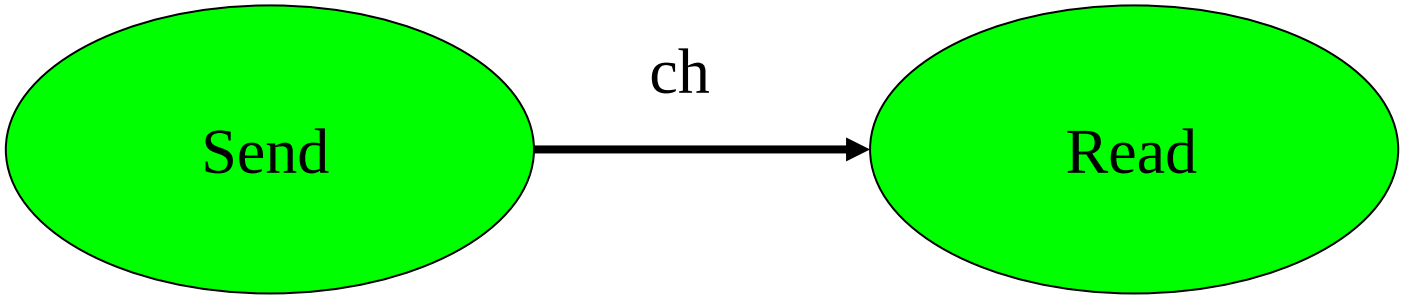
接続後は、P2P



・P1とP2は、直接的に仮想チャネルで接続される。

・仮想チャネルを介したメッセージパッシングによる同期がとられる。

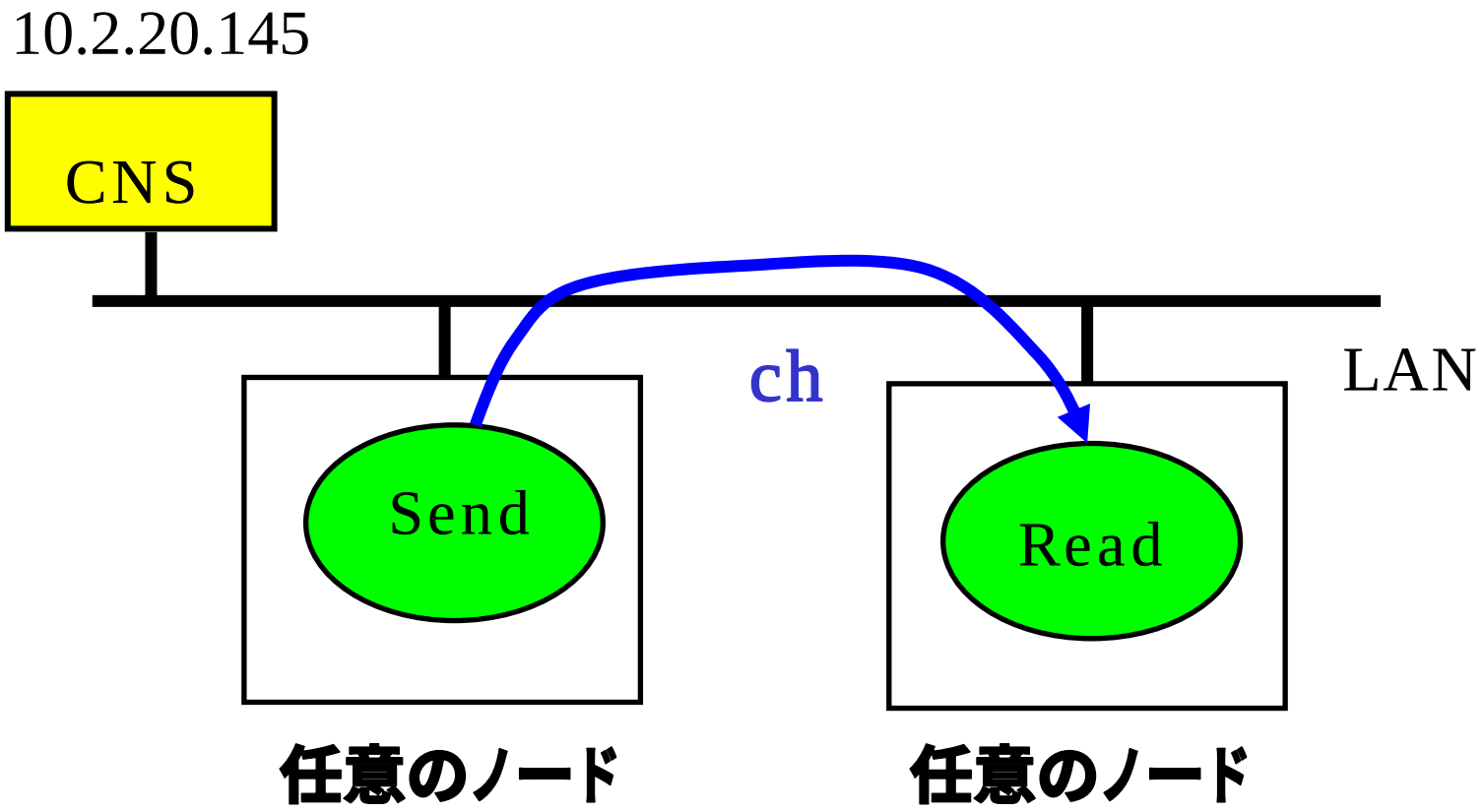
プログラム例



1台の計算機の場合

```
public static void main( String[] arg){
    One2OneChannel ch=Channel.one2one();
    Send send=new Send(ch.out());
    Read read=new Read(ch.in());
    CSProcess[ ] csp=new CSProcess[ ]{send,read};
    Parallel Para=new Parallel(csp);
    Para.run( );
 }
```

2台の計算機の場合



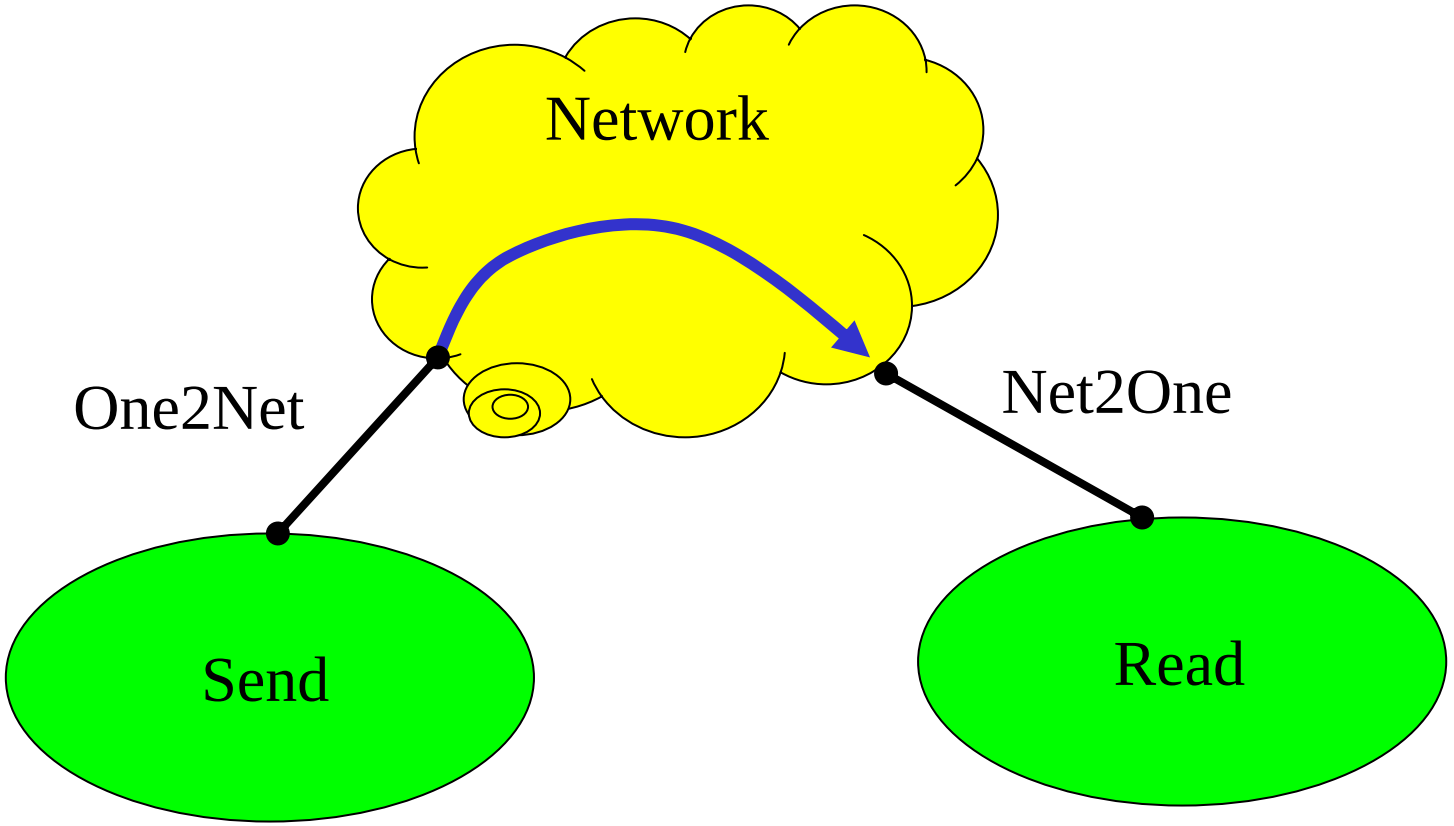
Sendプロセスの起動

```
public  static void main(String[ ] args )  {
      System.setProperty(“org.jcsp.tcpip.DefaultCNSServer",
                                                    "10.2.20.145")  ;
       try {      Node.getInstance().init( );   }
      catch (NodeInitFailedException e) {
            System.out.println("Node init failed¥n" + e);
             System.exit(-1)  ;     }
      new Send ( CNS.createOne2Net ( "ch" ) ) . run( )  ;
 }
```

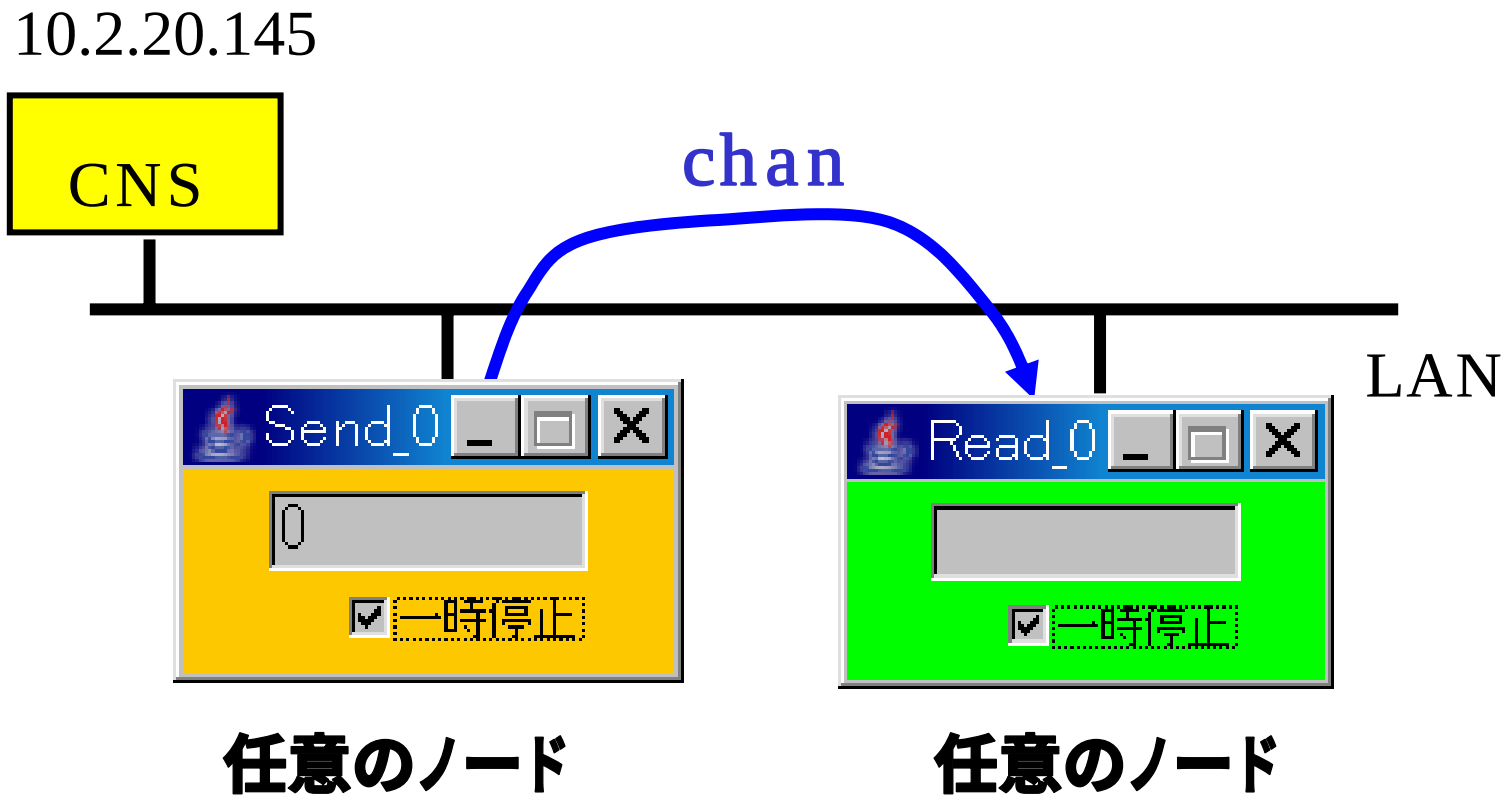
Readプロセスの起動

```
public  static void main(String[ ] args )  {
      System.setProperty(“org.jcsp.tcpip.DefaultCNSServer",
                                                  " 10.2.20.145 ")  ;
       try {      Node.getInstance().init( );   }
      catch (NodeInitFailedException e) {
            System.out.println("Node init failed¥n" + e);
             System.exit(-1)  ;     }
      new Read ( CNS.createNet2One ( "ch" ) ) . run( )  ;
 }
```

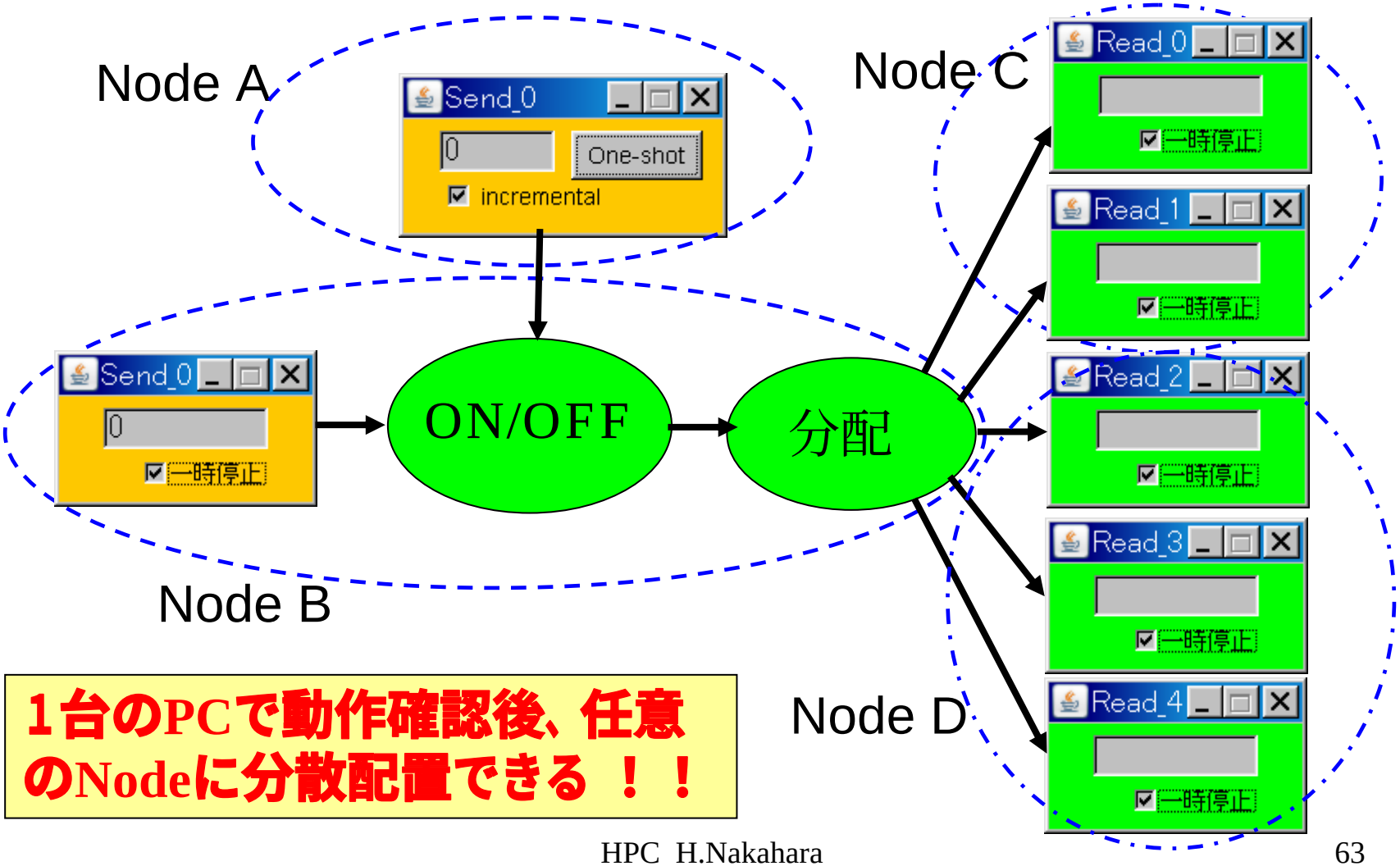
仮想チャネル



1. ネットワークチャネルで同期

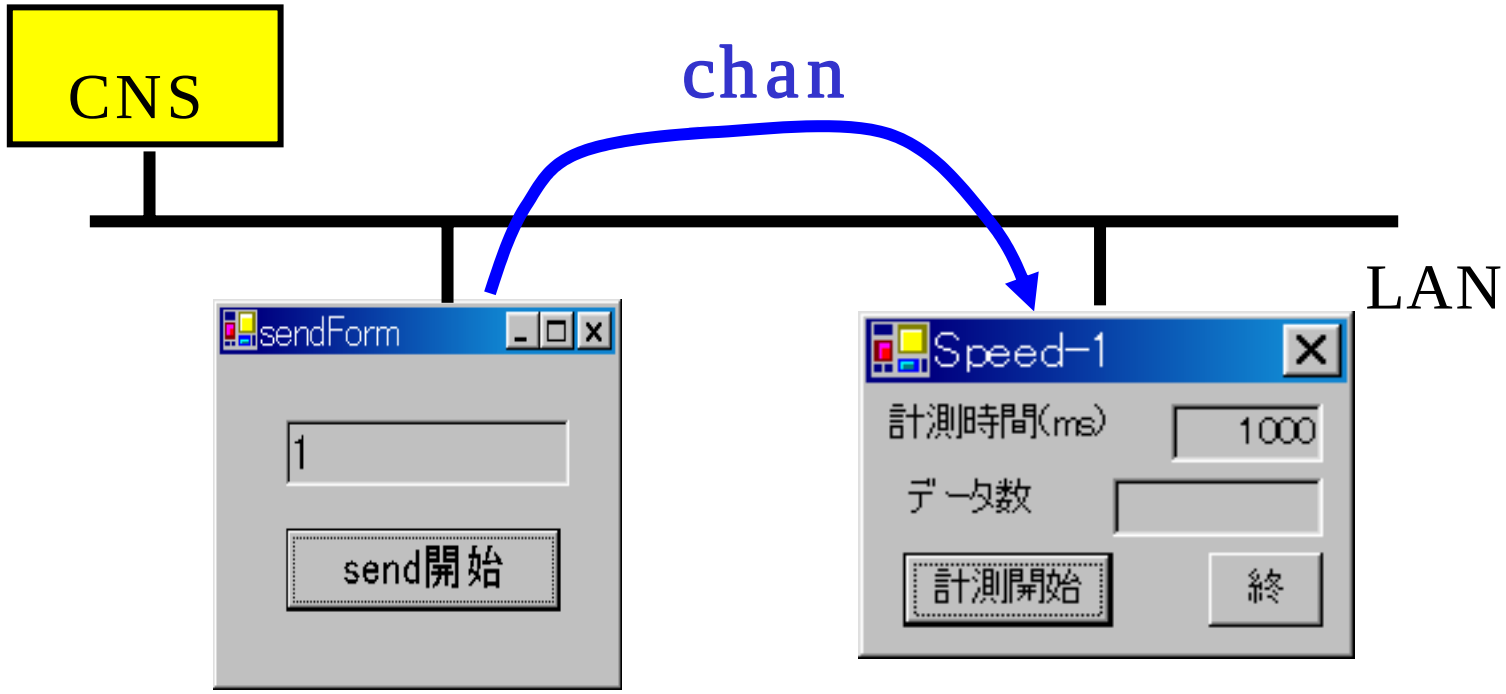


2. コンカレント処理と分散処理



1台のPCで動作確認後、任意のNodeに分散配置できる ！！

仮想チャネルの速度測定(J#.NET)



整数の出力

データ数のカウント

仮想チャネルの速度測定

CPU　：　Intel Celeron 1300MHz

LAN　：　100BASE **ハブ**

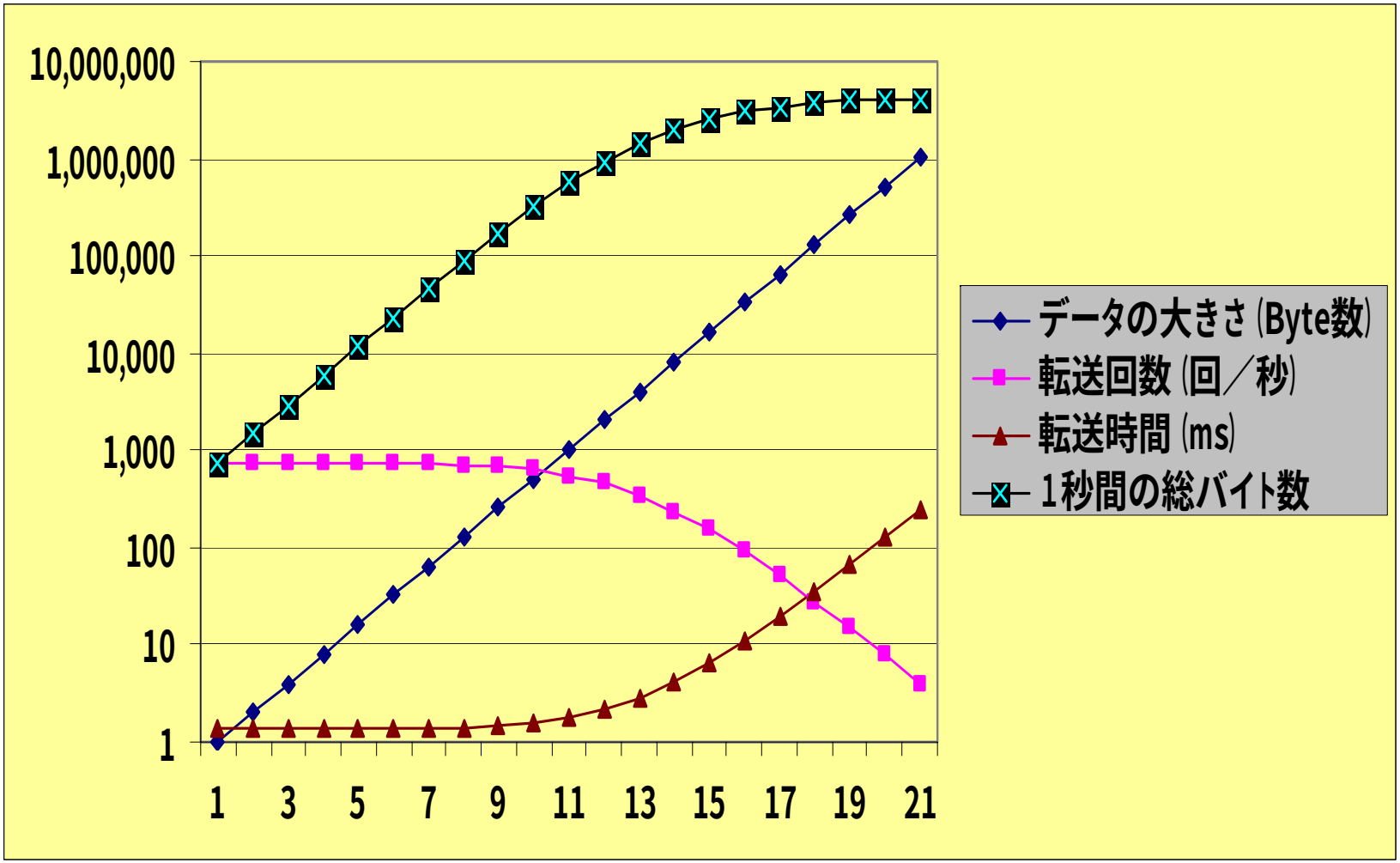
OS　：　Windows XP

言語　：　J#.NET (Visual Studio 2003)

1秒間に 740**個の**int**型データを受信**

1/740=1.35（ms）

データサイズと転送回数その１



転送速度.NETその２（参考）

CPU　：　Intel　P4　3000MHz

LAN　：　1000BASE ハブ

OS　：　Windows XP

言語　：JIBUシステム (C#.NET，Visual Studio 2005)

1秒間に2,000～4,000回のfloat型データを受信

0.5～0.25（ms /回）

測定値 (1) その1

データの大きさ (Byte数)	転送回数 (回／秒)	転送時間 (ms)	1秒間の総バイト数
1	740	1.35	740
2	740	1.35	1480
4	740	1.35	2960
8	740	1.35	5920
16	733	1.36	11728
32	733	1.36	23456
64	723	1.38	46272
128	710	1.41	90880
256	678	1.47	173568
512	630	1.59	322560
1,024	549	1.82	562176

測定値 (2) その１

データの大きさ (Byte数)	転送回数 (回／秒)	転送時間 (ms)	1秒間の総バイト数
2,048	454	2.20	929792
4,096	346	2.89	1417216
8,192	236	4.24	1933312
16,384	157	6.37	2572288
32768	94	10.64	3080192
65536	52	19.23	3407872
131072	28	35.71	3670016
262144	15	66.67	3932160
514288	8	125.00	4114304
1028576	4	250.00	4114304

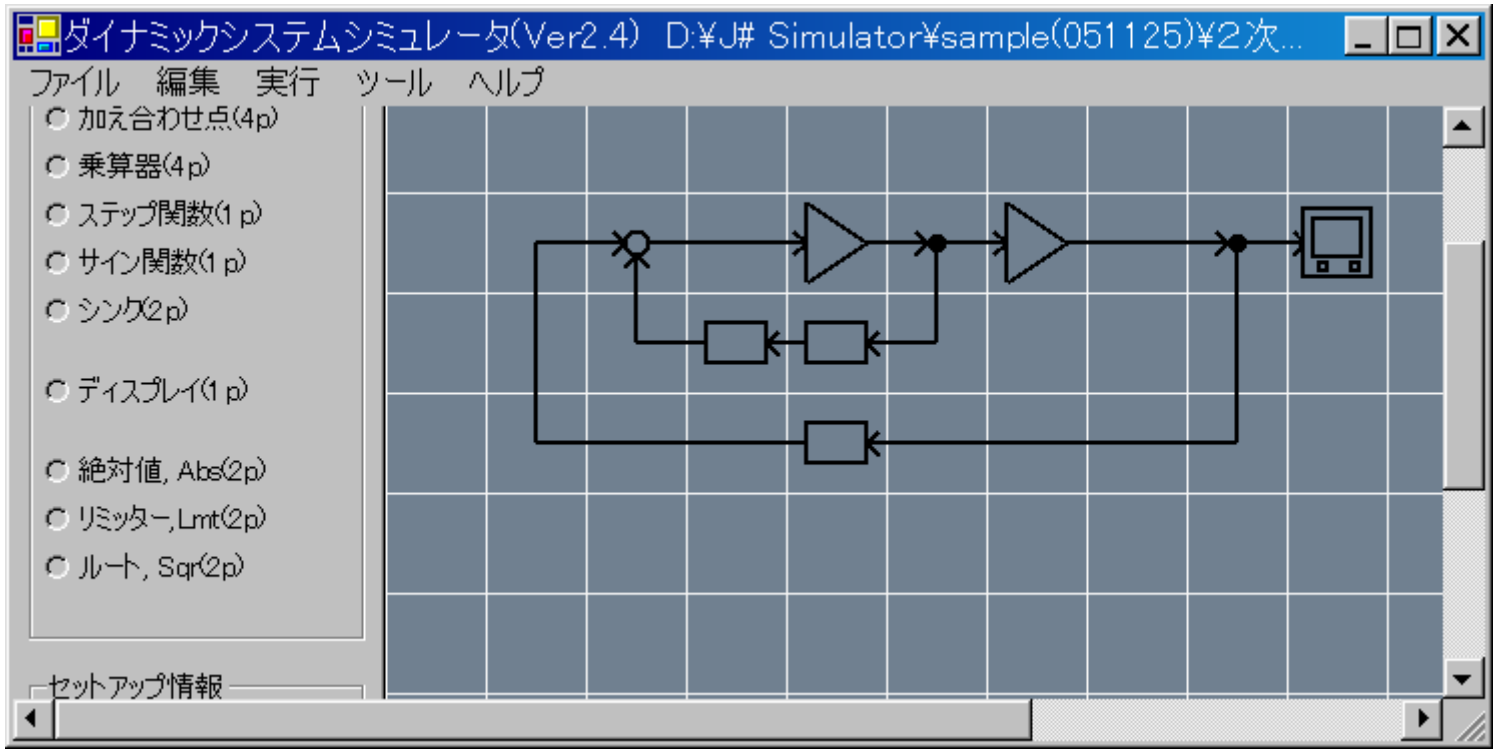
ＪＣＳＰの特長

- 100％ JAVAライブラリ
- 正統的なCSPモデルをサポート
- 並列実行と分散実行に対応（パラレルもコンカレントもOK）
- スレッドの概念はない
- シンプルなモデル、容易で安全なプログラム開発
- ノード内は、リファレンスを転送、高速
- イベント（チャネル）ドリブン、ビジーループと無縁
- CSP extensions for all AWT
- マルチコア時代・ネットワーク時代に最適なツール

ＪＣＳＰの利用例

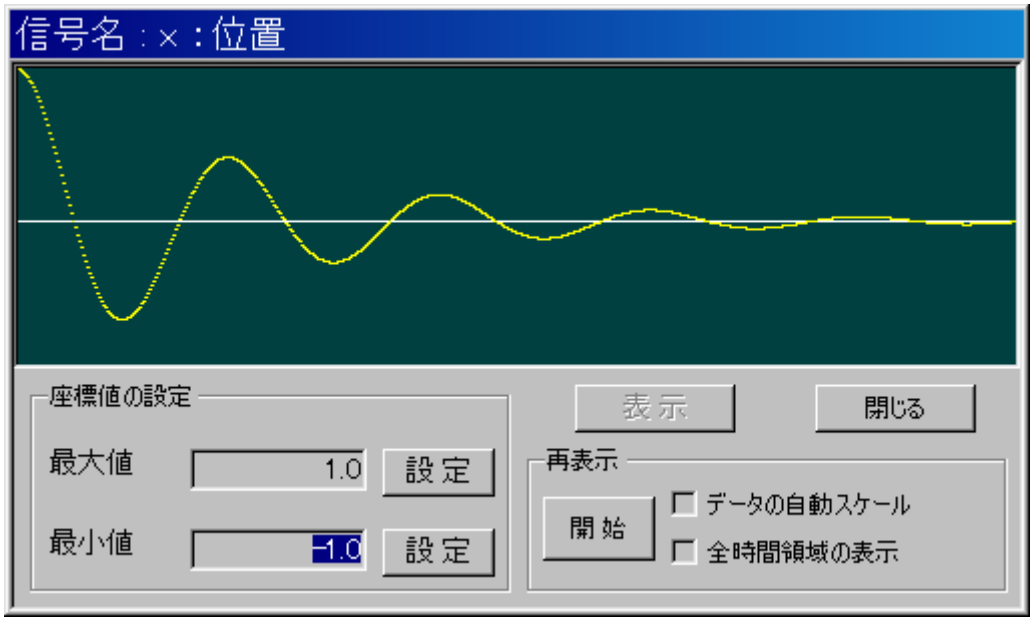
ダイナミックシステムシミュレータ

ブロックオリエンテッド



ＪＣＳＰの利用例

ダイナミックシステムシミュレータ



ＪＣＳＰの利用例

ダイナミックシステムシミュレータ

・ブロック線図をGUIにて作成

・各ブロックがプロセス

・線がチャネル

・GUIからチャネルの接続情報を取得

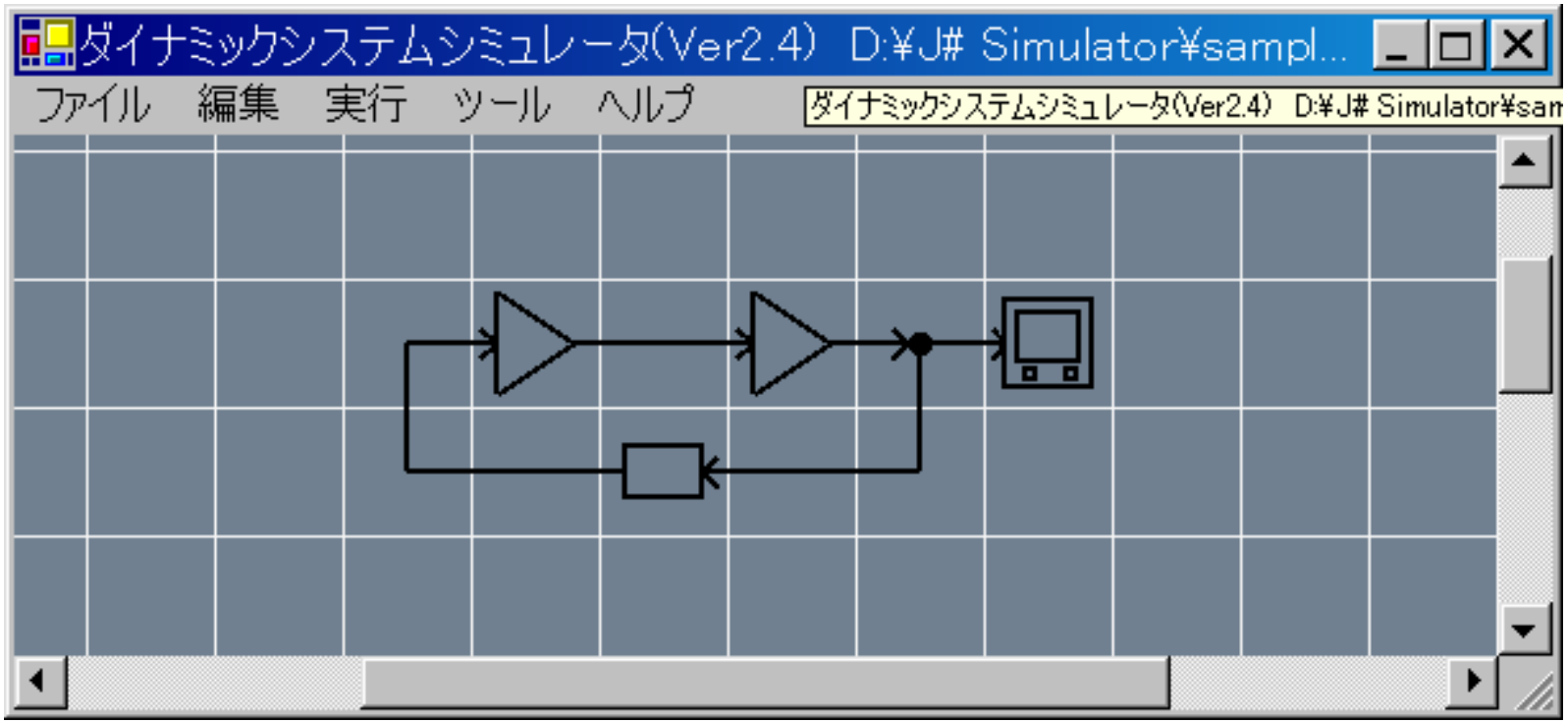
・各ブロック（プロセス）を生成、チャネル接続、並列実行

一番難しそうな並列実行部は以外と簡単 ！ ！

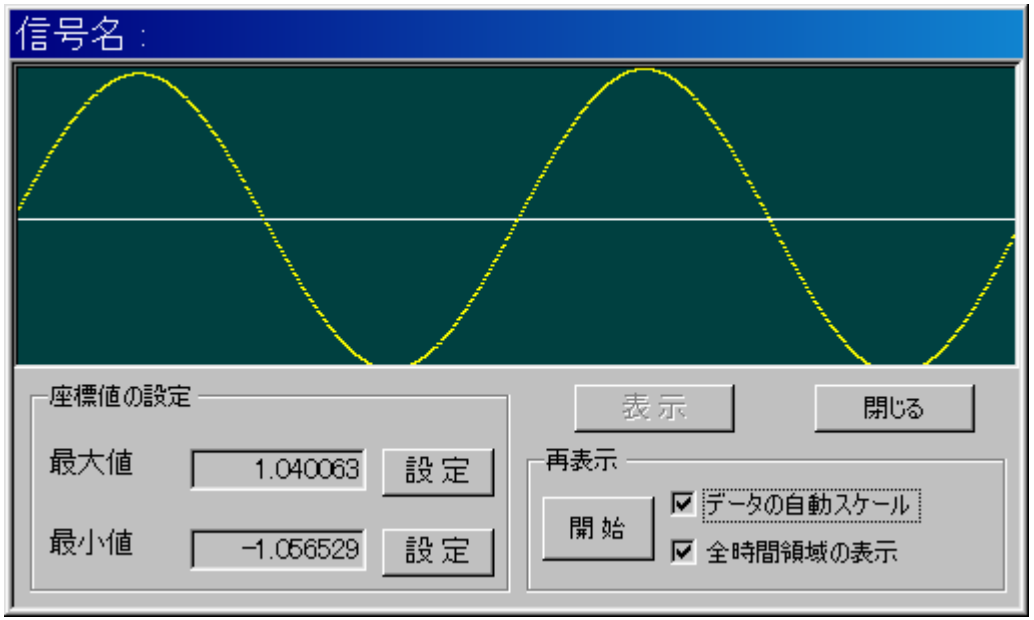
なぜデッドロックしない？

$$\frac{d^2x}{dt^2} + x(t)=0$$

$$x(t) = \sin(t)$$



なぜデッドロックしない？



積分器のrunメソッド

```
public void run(){
    Float xx;    float x , y ;    float t = 0.0f;
    while ( t < exetime){
        ParIO.run();
        xx = ( Float) chIn.obj;
        x = xx.floatValue();
        y = y + x * dt;
        chOut.obj = new Float(y);
        t = t + dt;
    }
}
```