# CSPモデルの優位性

産業技術総合研究所
情報技術研究部門
磯部祥尚

第9回CSP研究会（2012年3月17日）

# 講演内容

## 1. CSPモデルの特徴

- CSPモデルとは？
- 同期型メッセージパッシング通信
- イベント駆動
- 通信相手（チャネル）の自動選択

## 2. CSPモデルの実装

- ライブラリ/言語
- CSPモデルの実装例
- ローカル/ネットワークチャネル

## 3. CSPモデルの検証

- CSPモデルの記述例
- 検証ツール
- 振舞いの等しさ

## 4. CSPモデルベース開発

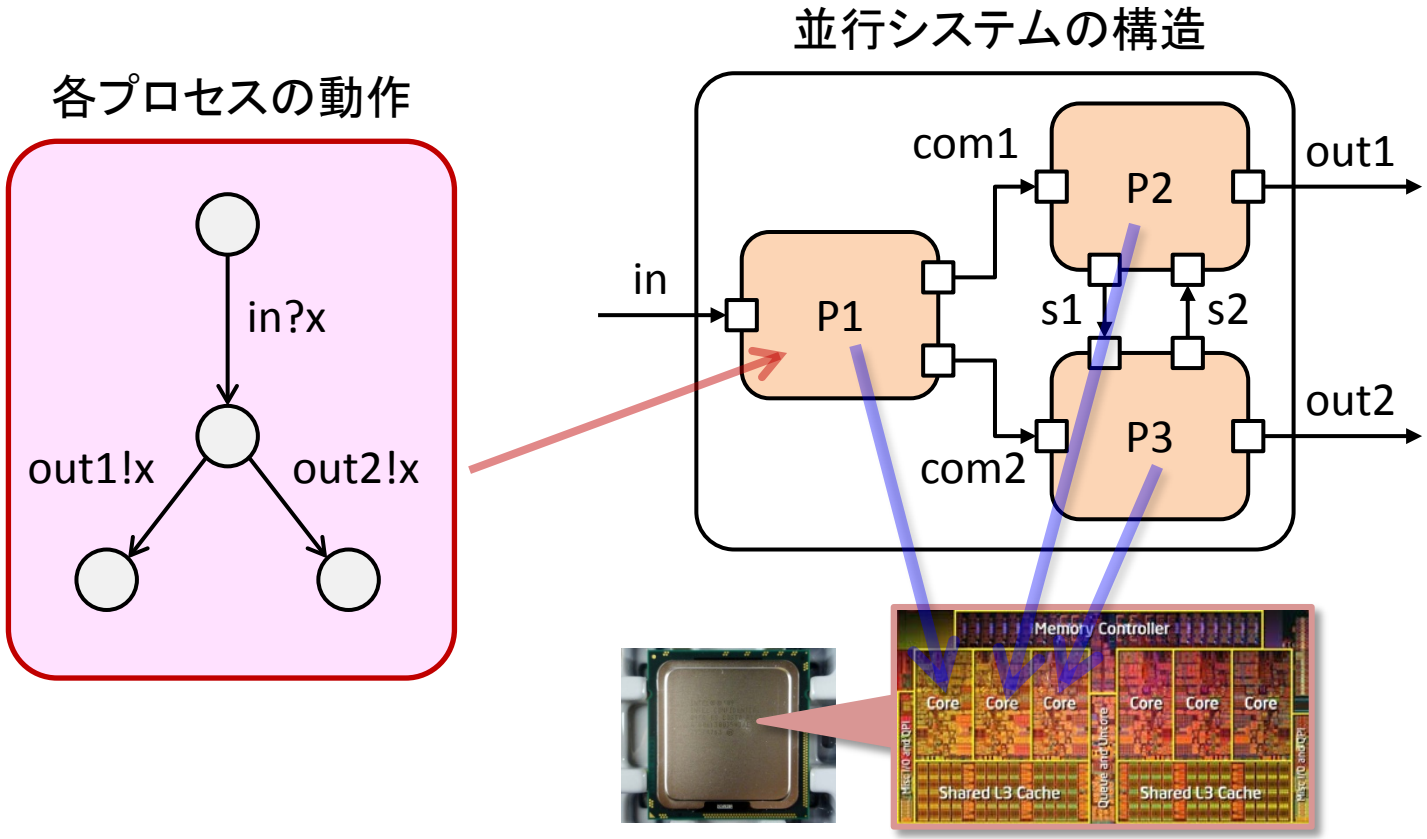
- 並列プログラミングの難しさ
- CSPによるモデル化、検証、実装
- まとめ

# CSPモデルの特徴

- CSPモデルとは？
- 同期型メッセージパッシング通信
- イベント駆動
- 通信相手（チャネル）の自動選択

# CSPモデルとは？

- **CSPモデル：並行システムを表現する方法のひとつ**
  - 並行システムの構造（チャネルの接続関係）を表現できる
  - 各プロセスの動作（入出力、計算の順序関係）を表現できる




例：6コアCPU　Core i7-980X

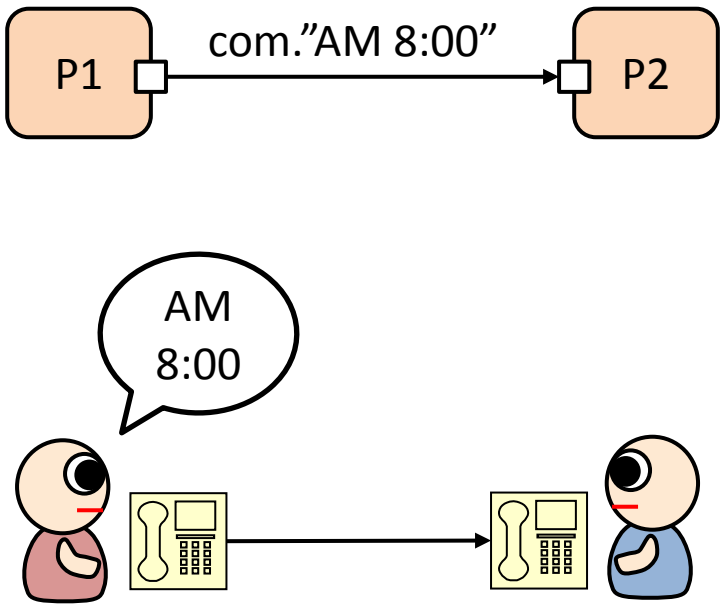
# CSPモデルの特徴（同期型メッセージパッシング通信）

■ CSPモデルはチャネルを用いた同期型メッセージパッシング通信方式を採用

○ 確実に情報が相手に伝わる（動作がわかりやすい）

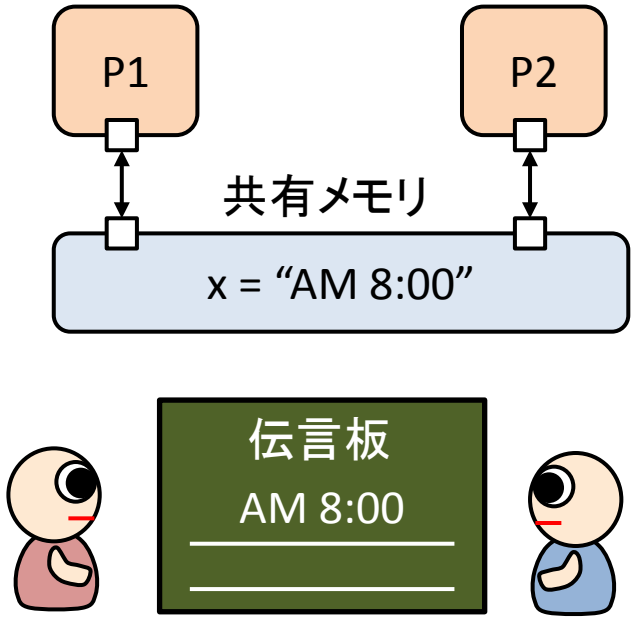
× 共有メモリ通信よりも処理が重い

同期型メッセージパッシング通信




話したメッセージは確実に伝わる
（不在ならば話せない）

共有メモリ通信

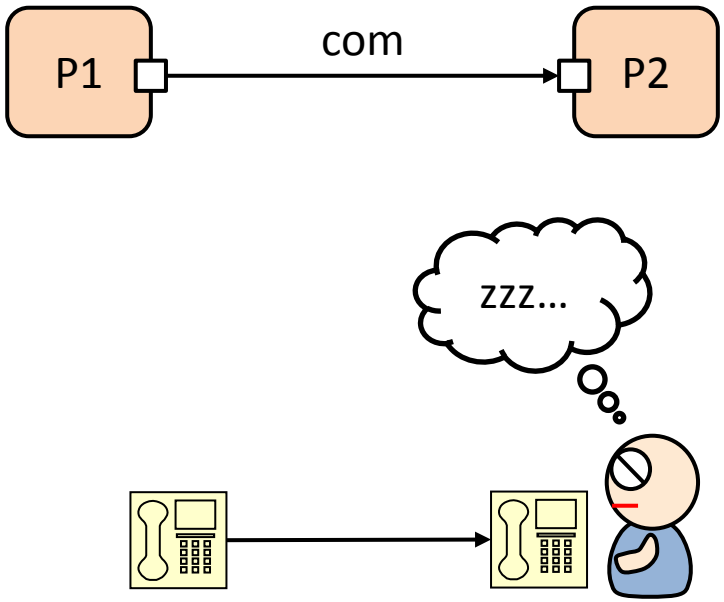



書いたメッセージが伝わるか不安
（読み書きのタイミングが難しい）

# CSPモデルの特徴（イベント駆動）

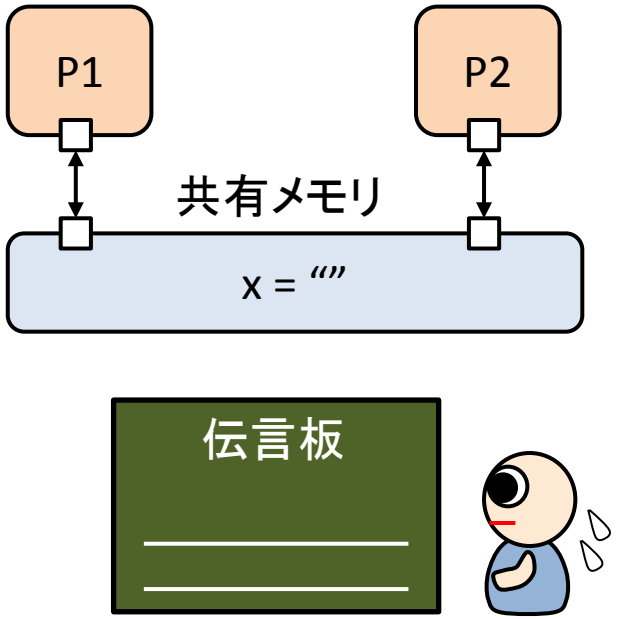
- **CSPモデルはイベント駆動を採用**
  - 同期型メッセージパッシング通信＋イベント駆動：待ちの間のCPU負荷はかからない
  - 共有メモリ＋ポーリング：定期的に書込みの有無をチェックするためCPU負荷がかかる

イベント駆動




電話が鳴るまでは眠っていてよい

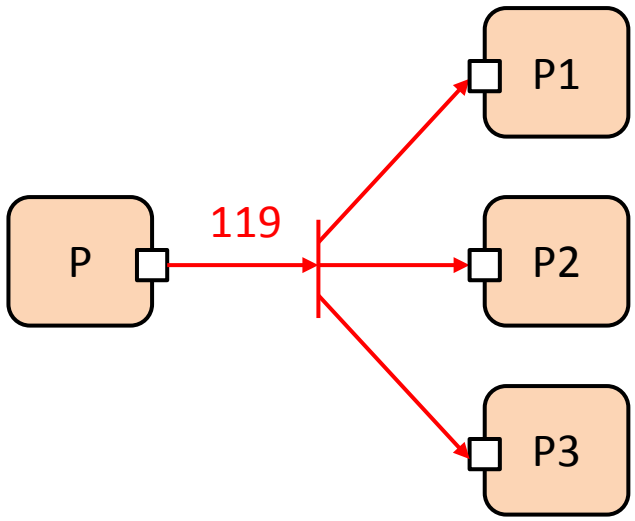
ポーリング



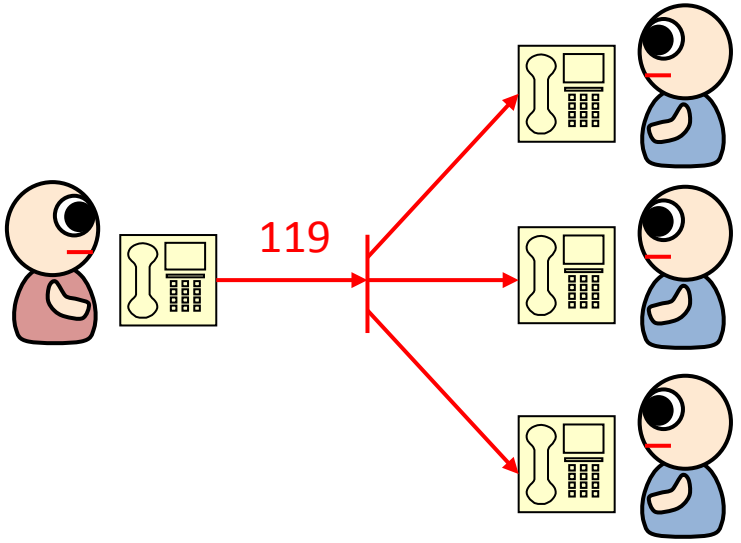

書込みの有無を定期的にチェック
（何もしないのに忙しい）

# CSPモデルの特徴(通信相手の自動選択)

■ CSPモデルは通信可能な相手を自動的に選択して通信可能



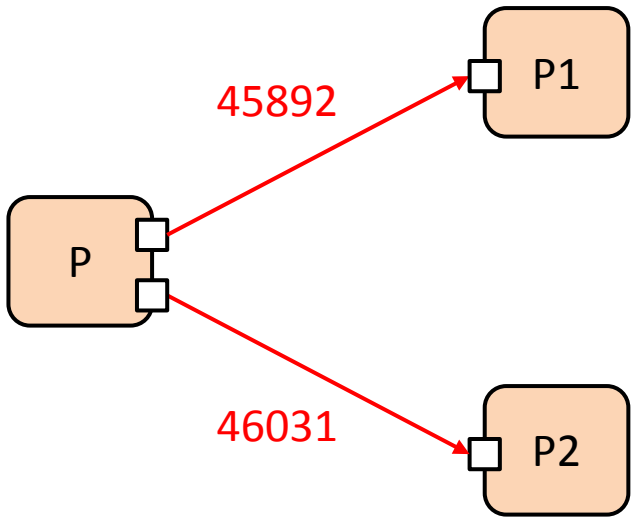

受信可能なプロセスを選択して送信する(全てが受信不可のときは、ひとつが受信可能になるまで送信を延期)



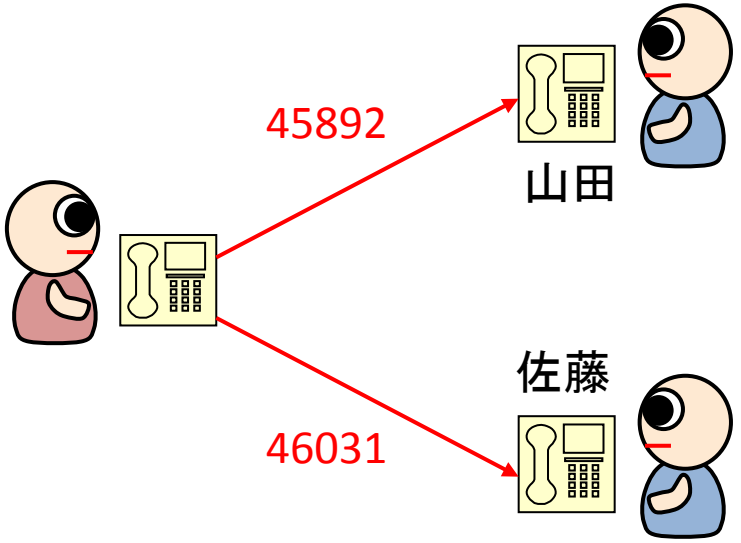

代表電話(手の空いている人が受ける)

# CSPモデルの特徴(通信チャネルの自動選択)

■ CSPモデルは通信可能なチャネルを自動的に選択(外部選択)して通信可能




受信可能なプロセスが接続されている
チャネルを選択して送信する




2人に同時に電話をかけて出た方とだ
け会話する(このような電話はない？)

# CSPモデルの実装

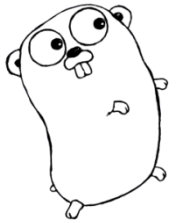
- ライブラリ/言語
- 実装例
- ローカル/ネットワークチャネル

# CSPモデルを実装するためのライブラリ/言語

- CSPモデルを実装するための**ライブラリ**や**プログラミング言語**が公開されている。

| ライブラリ | 言語 | 研究開発元 |
|---|---|---|
| JCSP | Java | ケント大学（QuickStone） |
| C++CSP | C++ | ケント大学 |
| CHP | Haskell | ケント大学 |
| PyCSP | Python | トロムソ大学＆コペンハーゲン大学 |
| Python-CSP | Python | ウォルバーハンプトン大学 |

- **Go言語: 2009年秋にGoogleが発表したプログラミング言語**
  - ✓ コンパイル言語のように速く、スクリプト言語のように使い易い。
  - ✓ CSPに基づく並列処理記述が言語レベルで可能。

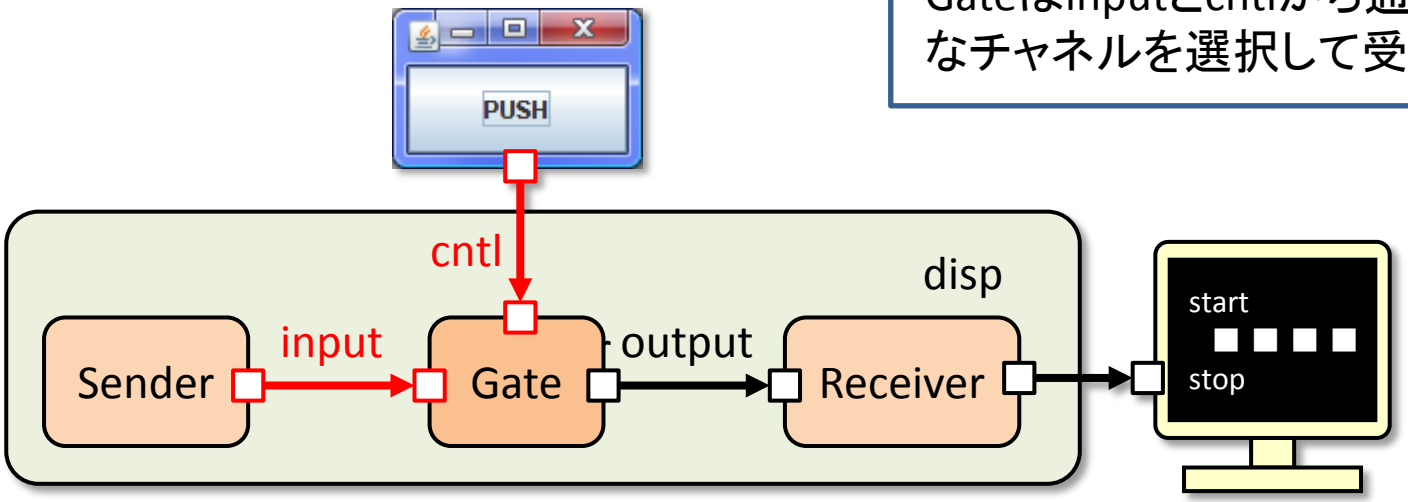


- **XMOS : 2008年からXMOS社が販売しているプロセッサ**
  - ✓ イベント駆動型マルチスレッドプロセッサ（XS1-G4, 4コア, 32スレッド）
  - ✓ CSPに基づくC言語風の実装言語XCによるハードウェア実装

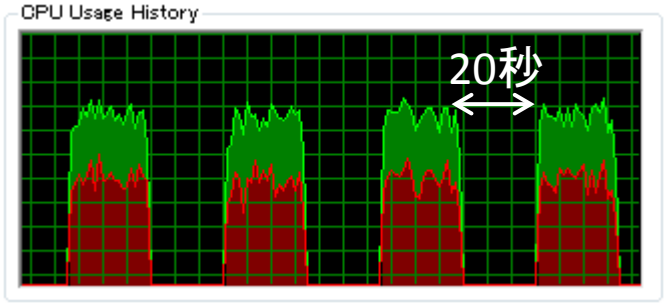
# CSPモデルのJCSP（Java）による実装例

- **Sender, Gate, Receiverから構成される並行システム**
  - GateはSenderのデータをReceiverに転送する
  - 転送の中断と再開を外部から制御できる

GateはinputとcntlからGateは通信可能なチャネルを選択して受信する



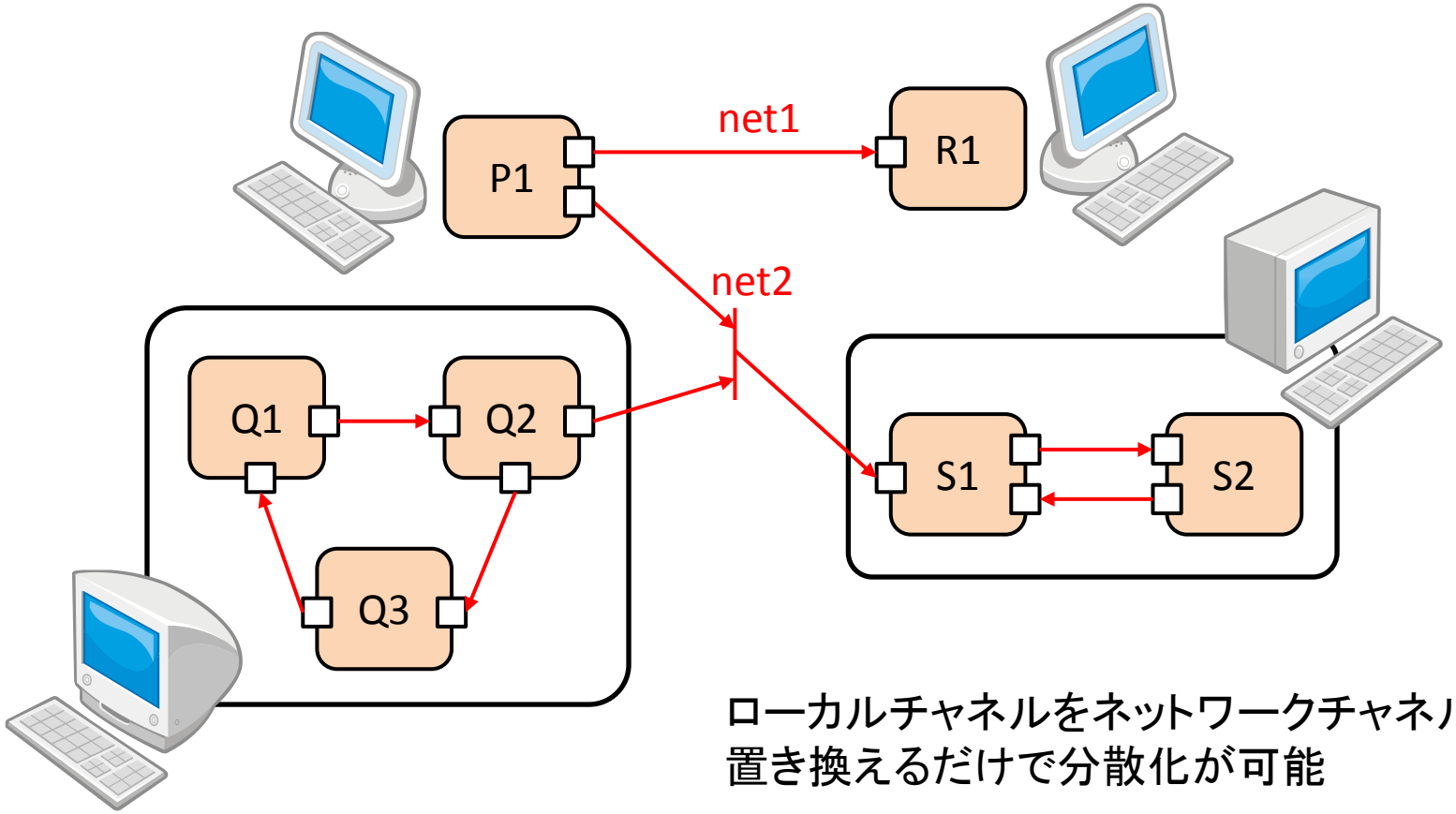

- **20秒ごとに転送の中断と再開を繰り返すとCPU負荷は下記のようになる**




転送を中断中のCPU負荷は0

# ローカルチャネルとネットワークチャネル

- JCSP（Java）では複数のプロセスを**複数のPCで分散実行する**ことも簡単
  - ローカルチャネル：1つのプログラム内の複数のプロセスを接続する
  - ネットワークチャネル：異なるプログラム（PC）間の複数のプロセスを接続する




ローカルチャネルをネットワークチャネルに
置き換えるだけで分散化が可能

# CSPモデルの検証

- CSPモデルの例
- 検証ツール
- 振舞いの等しさ

# 並行システムのCSPモデルの例

■ CSPでは並行システムの構造や動作を“式”で表現する。

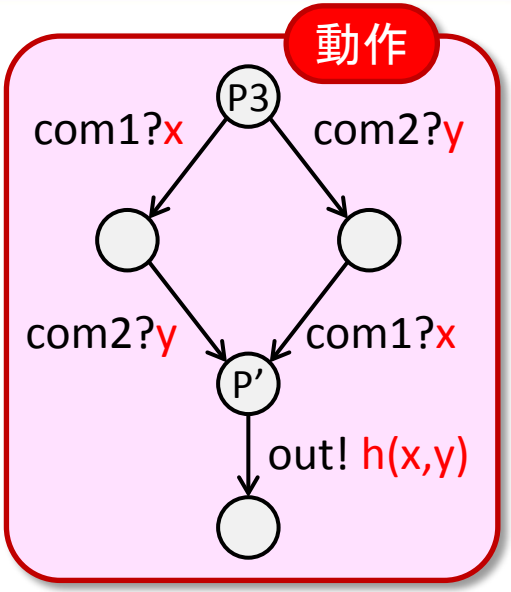
CSPモデル

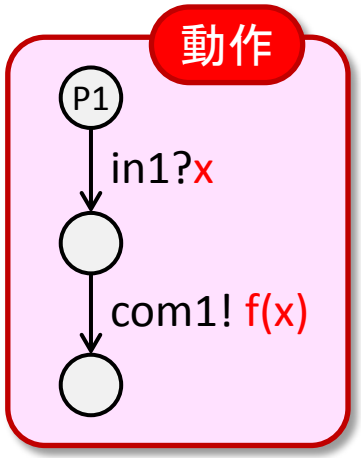
動作 P1 = in1?x → com1! f(x) → STOP

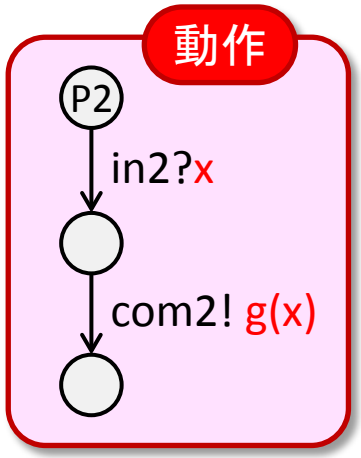
動作 P2 = in2?x → com2! g(x) → STOP

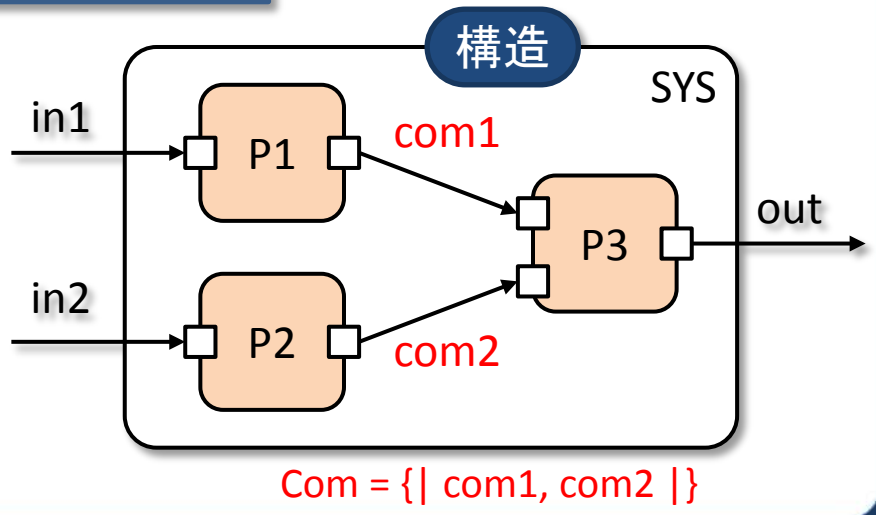
動作 P3 = com1?x → com2?y → P’(x,y)
□ com2?y → com1?x → P’(x,y)
P’(x,y) = out! h(x,y) → STOP

構造 SYS = ((P1 ||| P2) [|Com|] P3) \ Com



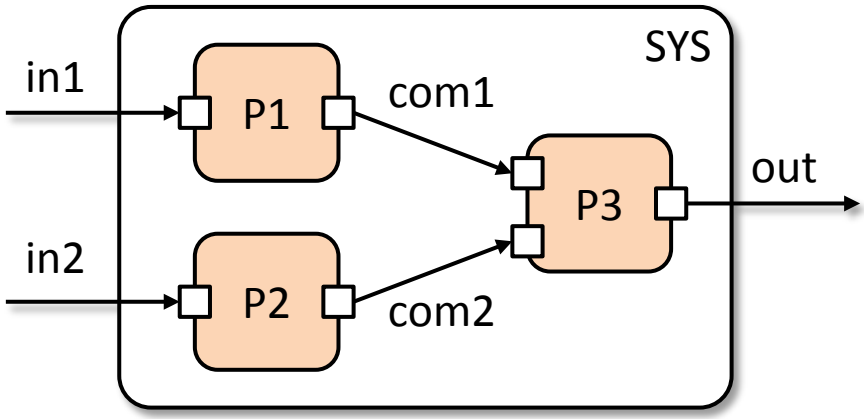



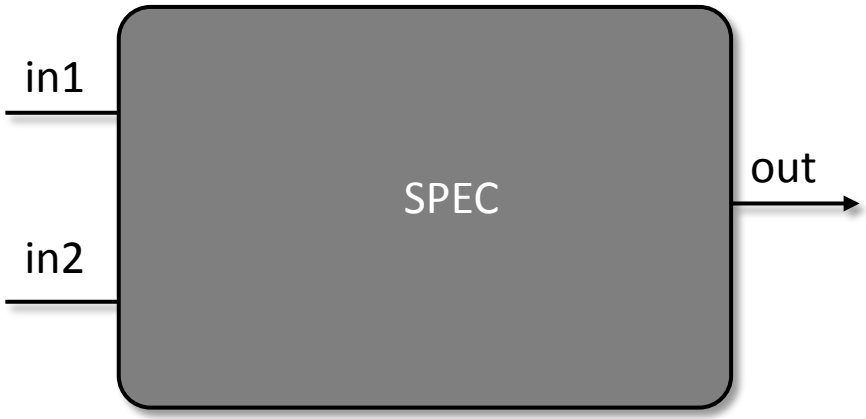




Com = {| com1, com2 |}

# CSPモデルの検証

■ CSPでは並行システムの動作の等しさを証明できる




並行

P1 = in1?x → com1! f(x) → STOP

P2 = in2?x → com2! g(x) → STOP

P3 = com1?x → com2?y → P'(x,y)
□ com2?y → com1?x → P'(x,y)

P'(x,y) = out! h(x,y) → STOP

SYS = ((P1 ||| P2) [|Com|] P3) \ Com

逐次

SPEC = in1?x → in2?y → Q(x,y)
□ in2?y → in1?x → Q(x,y)

Q(x,y) = out! h(f(x),g(y)) → STOP

この例では、下記の等しさを証明できる

SYS = SPEC

# CSPベースのモデル検査ツールFDR（Oxford大学）

■ モデル検査ツール FDRを使うとSYSとSPECの等しさ（詳細化関係）を自動判定できる。

並行

P1 = in1?x → com1! f(x) → STOP

P2 = in2?x → com2! g(x) → STOP

P3 = com1?x → com2?y → P'(x,y)
□ com2?y → com1?x → P'(x,y)

P'(x,y) = out! h(x,y) → STOP

SYS = ((P1 ||| P2) [|Com|] P3) \ Com

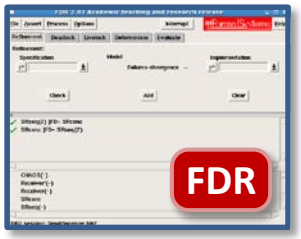
逐次

SPEC = in1?x → in2?y → Q(x,y)
□ in2?y → in1?x → Q(x,y)

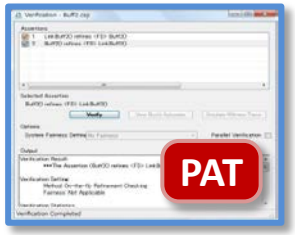
Q(x,y) = out! h(f(x),g(y)) → STOP

assert SYS = SPEC（検証項目）

SYS = SPEC ?

入力 → 

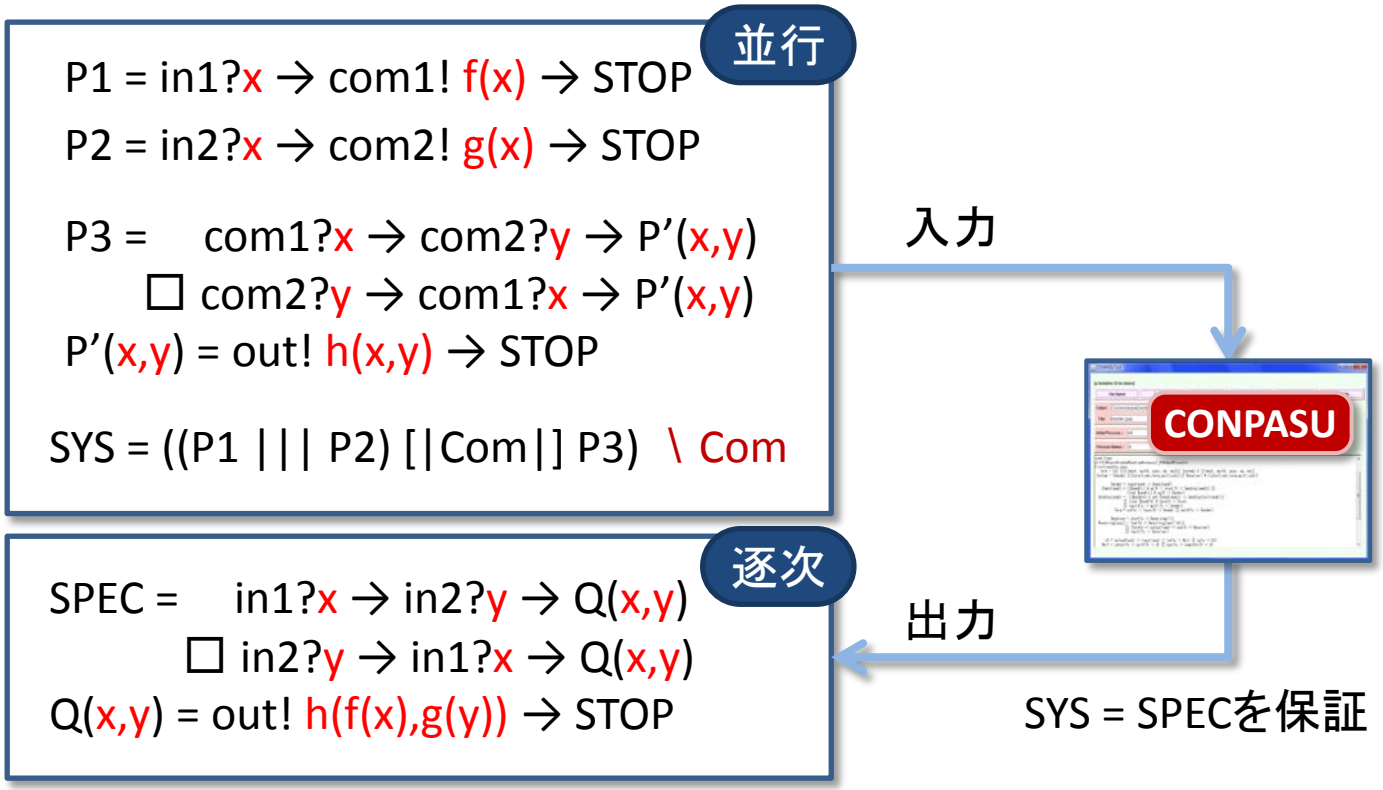
→ 出力 → True




（シンガポール大学のモデル検査ツール）

# CSPベースの解析ツールCONPASU（産総研）

■ 解析ツール CONPASUを使うとSYSからSPEC（に近い記述）を自動生成できる。

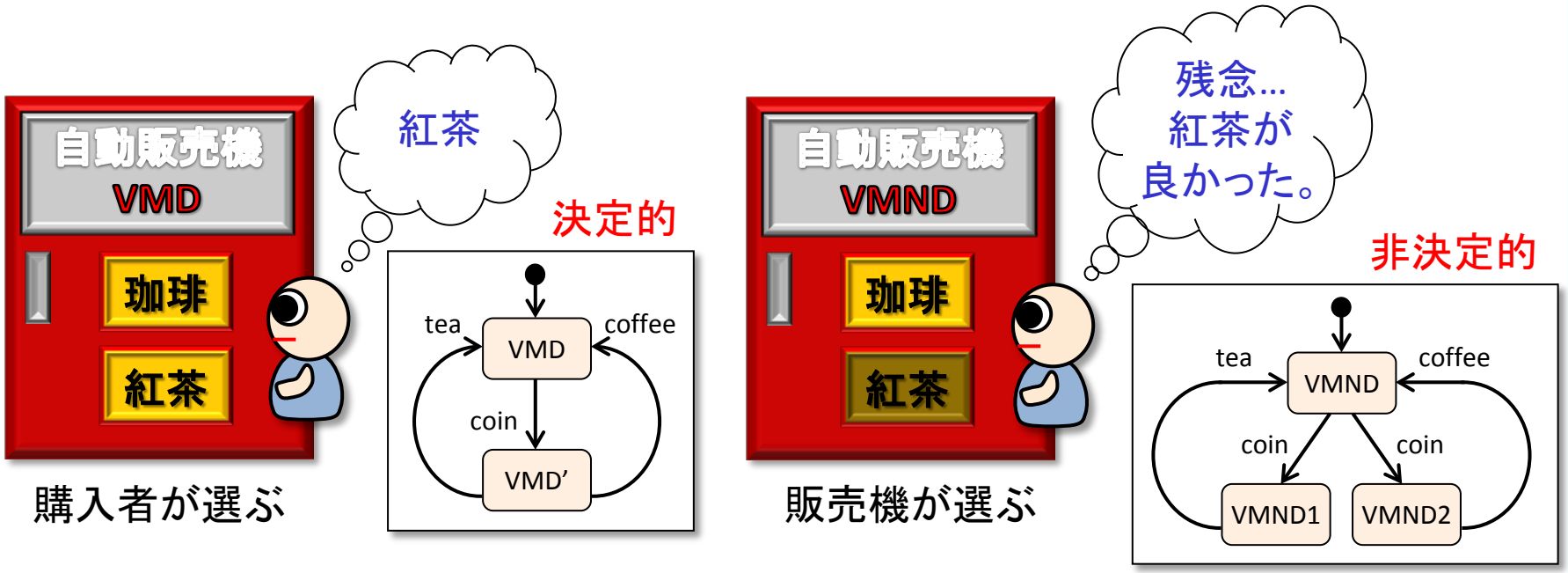



■ SYSの記述は比較的機械的に作成できるが、**SPECの記述は意外と難しい**

# CSPモデルの等しさの特徴

■ CSPの等しさは決定的選択と非決定的選択を区別できる。

コインを入れた後に、珈琲か紅茶を購入できる自動販売機



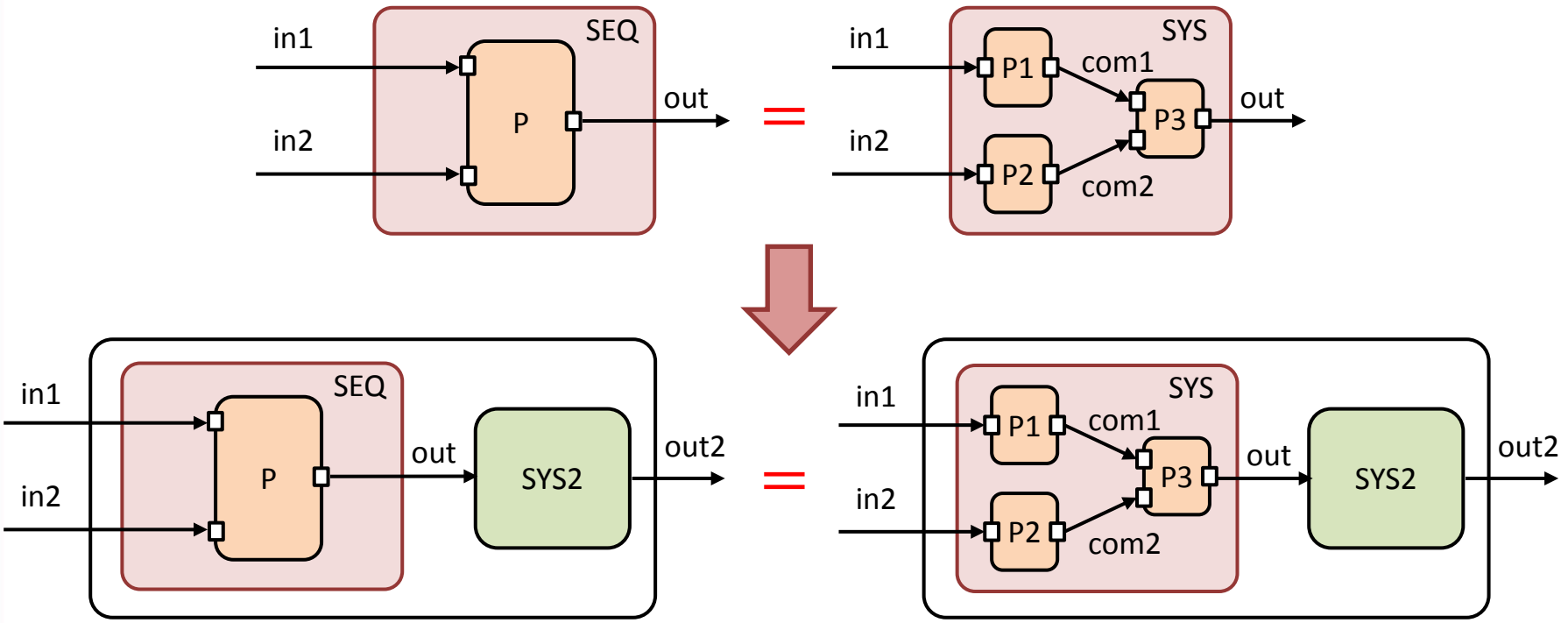

CSPでは　VMD ≠ VMND

（注：実行トレースだけでは区別できない）

# CSPモデルの等しさの保存

■ CSPの等しさは合成後も保存される（部品単位での検証が可能）。

SEQ ＝ SYS　⇒　(SEQ ||[Out]| SYS) \ Out ＝ (SYS ||[Out]| SYS) \ Out



# CSPモデルベース開発

- 並列プログラミングの難しさ
- CSPによるモデル化、検証、実装
- まとめ

# 並列プログラミングの難しさ

- 「マルチコアシステムでのJava並行性バグのパターン」のウェブサイト（2010.12.21）:
  http://www.ibm.com/developerworks/jp/java/library/j-concurrencybugpatterns/

  - Java並行プログラミングにおける間違えやすい様々なバグのパターンとその回避策が紹介されている。
  - そのようなプログラムをモデル化して検査することは可能であるが、根本的なところに難しさがある。
  - CSPベースの言語/ライブラリを使うことで振舞いが明確になり、このようなバグも抑えられる。また、検証ツールも使える。

# CSPモデルの検証と実装

■ 並行システムをCSPでモデル化し、FDRで検証し、JCSPで実装する

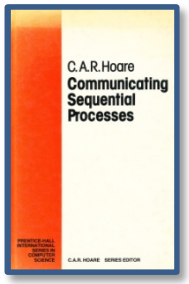
理論
CSP

→

検証
FDR

→

実装
JCSP




Communicating
Sequential
Processes,
Hoare, 1985


プロセス代数 CSP:
並行システムを記述・解析するための理論

モデル検査器 FDR:
CSPモデルの詳細化関係を検証するツール

Oxford大学、Formal Systems
（アカデミックライセンスは無料）

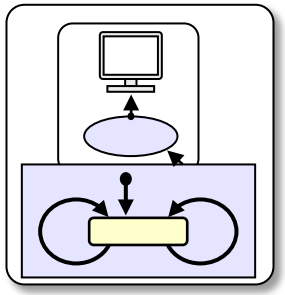
Javaライブラリ JCSP:
CSPモデルをJavaで実装するためのライブラリ

Kent大学
（2006年秋からオープンソース）

# モデル化、検証、実装の流れ

■ 並行システムをCSPでモデル化し、FDRで検証し、JCSPで実装する

逐次システムや仕様



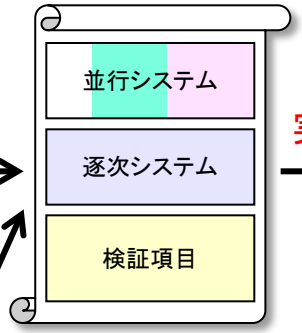
モデル化

CSPモデル

```
SRseq(K) = SendRec(0,0,0,K)
SendRec(n,m,i,K)
= i<K & disp!send.n
  → SendRec((n+1)%N,m,i+1,K) □
  i>0 & disp!receive.m
  → SendRec(n,(m+1)%N,i-1,K)
```

検証

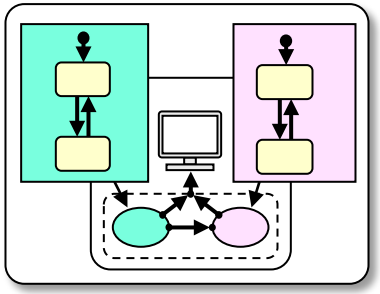
FDRスクリプト




実行

FDR

並行システムや仕様



モデル化

CSPモデル

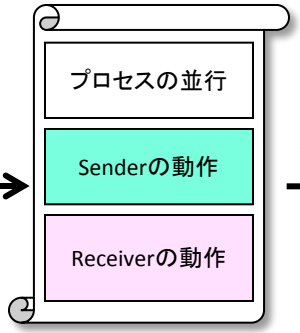
```
SRconc
= (Sender(0) [| {| chan |} |]
   Receiver(0))＼{| chan |}

Sender(n)
 = disp!send.n → Sender'(n)
Sender'(n)
 = chan!n → Sender((n+1)%N)

Receiver(m)
= chan?m → Receiver'(m)
Receiver'(m)
= disp!receive.m → Receiver(m)
```

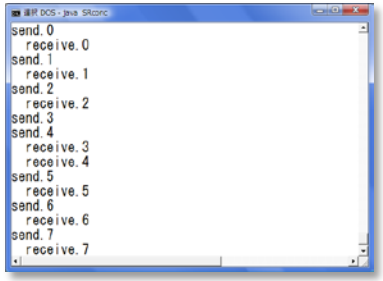
実装

JCSPプログラム




実行

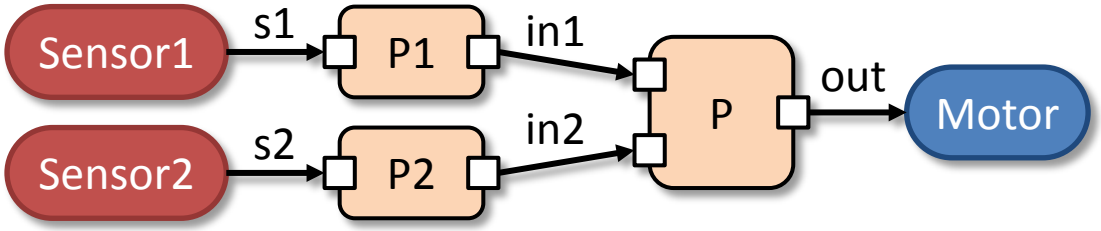
Java



# まとめ

■ 並行システムの開発にCSPモデルを利用する ありがたみ

- 設計の視点 → 設計コストを削減できる
  - 同期チャネルによる通信 （例：タイミング制御が容易）
  - 通信相手（チャネル）の自動選択 （例：複数の入出力処理が容易）




- 検証の視点 → 実装前に不具合を発見できる
  - モデル検査器FDR, PATによる不具合発見
  - 解析ツールCONAPSU（開発中）による並行動作の可視化

- 実装の視点 → CSPモデルを実装できる
  - CSPモデルベースの言語 / ライブラリ（JCSP, Go, …）による実装
  - イベント駆動によるCPU負荷の削減