

知能ソフトウェア特論

Intelligent Software

## モデル検査 (1)

---

並行システムのモデリング

Model Checking (1)

Modeling Concurrent Systems

# 1. 並行システム (Concurrent systems)

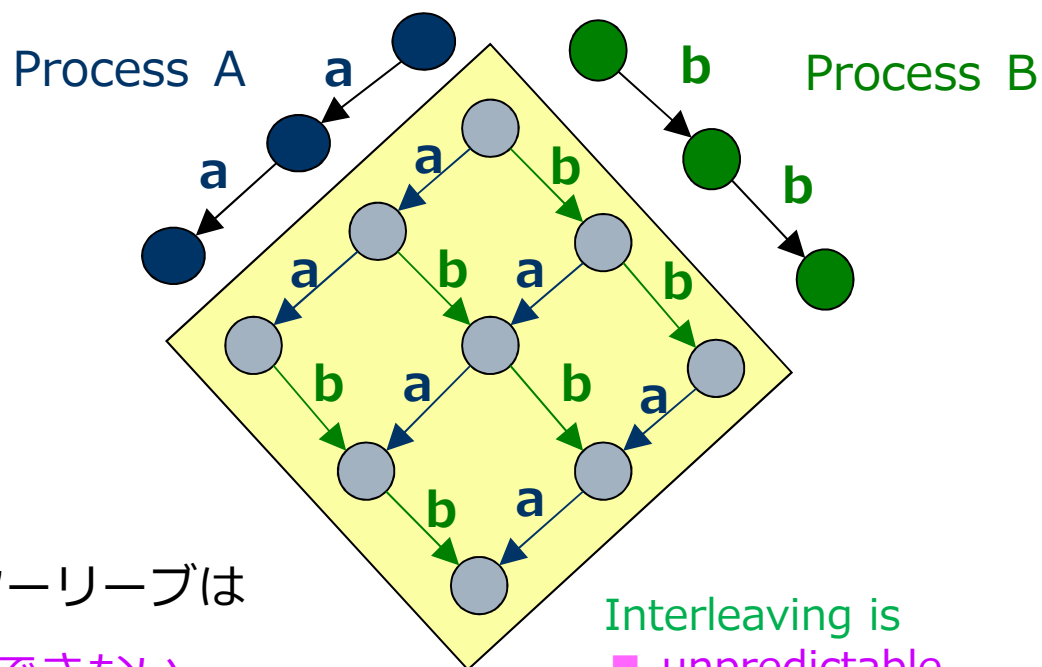
---

並行システム：  
複数のプロセス(逐次プログラムの実行)  
が並列(または擬似並列)に動作する計算シ  
ステム

A concurrent system is a computational system in which multiple processes (i.e., the executions of sequential programs) are run in parallel (or quasi-parallel).

---

# インタリーブ (Interleaving)



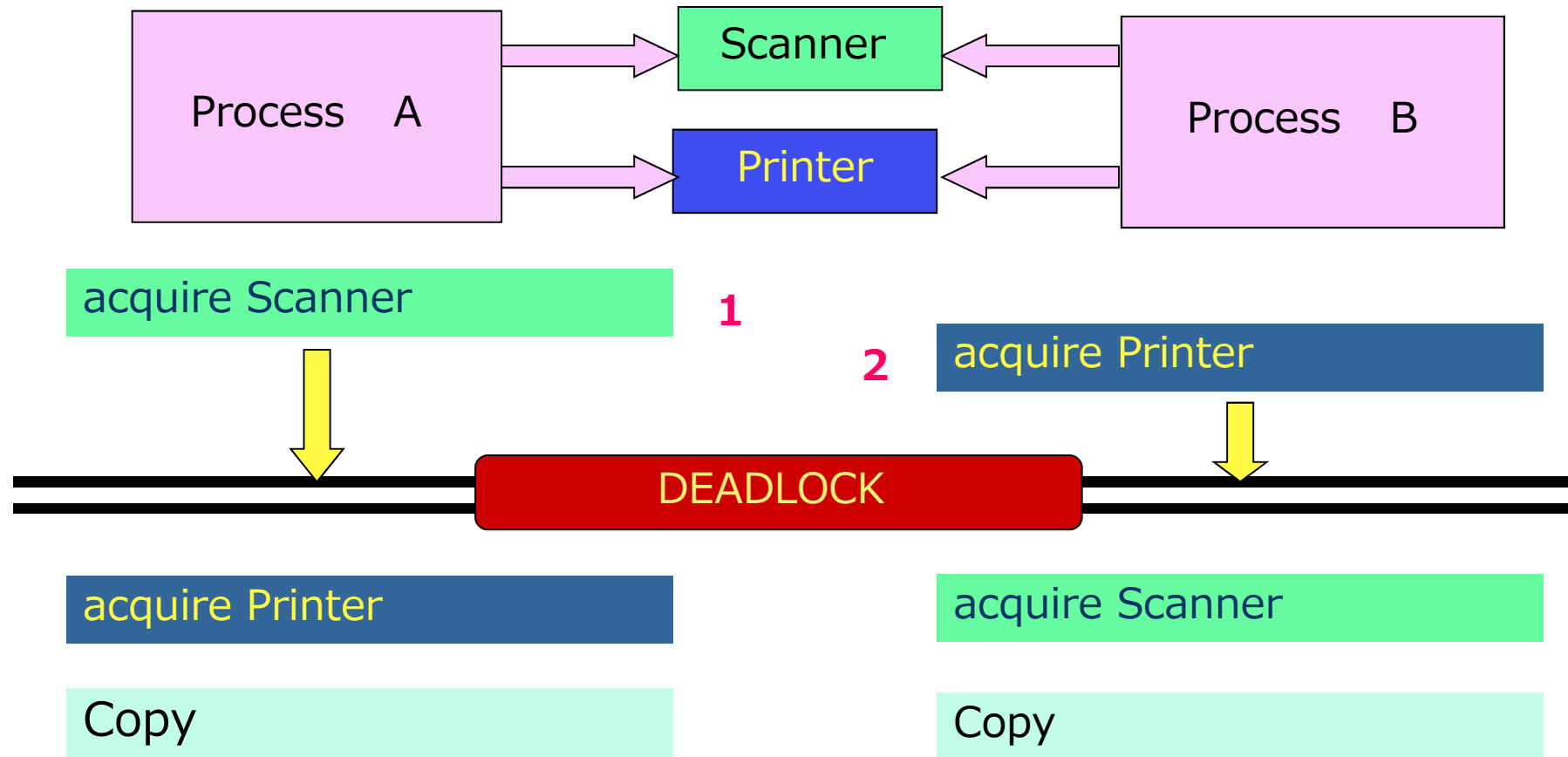
インタリーブは

- 予期できない
- 制御できない
- 膨大な数の実行経路を生じる

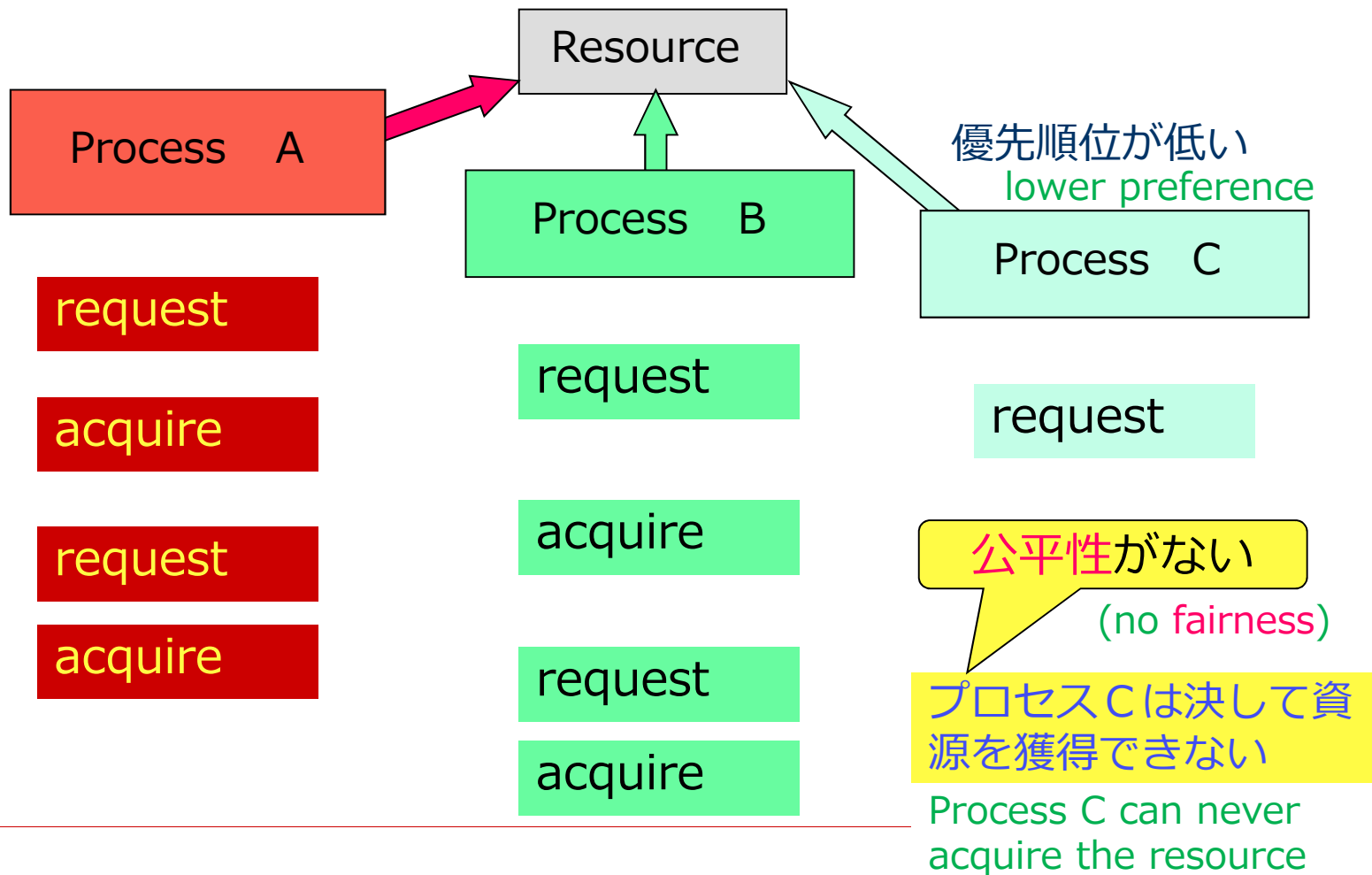
Interleaving is

- unpredictable,
  - uncontrollable, and
  - subject to yielding
- a huge number of computational paths.

# デッドロック (Deadlock)



# ライブロック, 飢餓, 公平性 (Livelock, Starvation, Fairness)



## 2. モデル検査の概要 (Overview of model checking)

---

入出力関係に着目した「**停止性 + 部分正当性**」の解析のみでは並行システムの正当性を言えない

→ **振る舞い** (途中の状態遷移) の考慮が必要

モデル検査：  
有限状態並行システムの**振る舞い**の検証を自動で行う技術

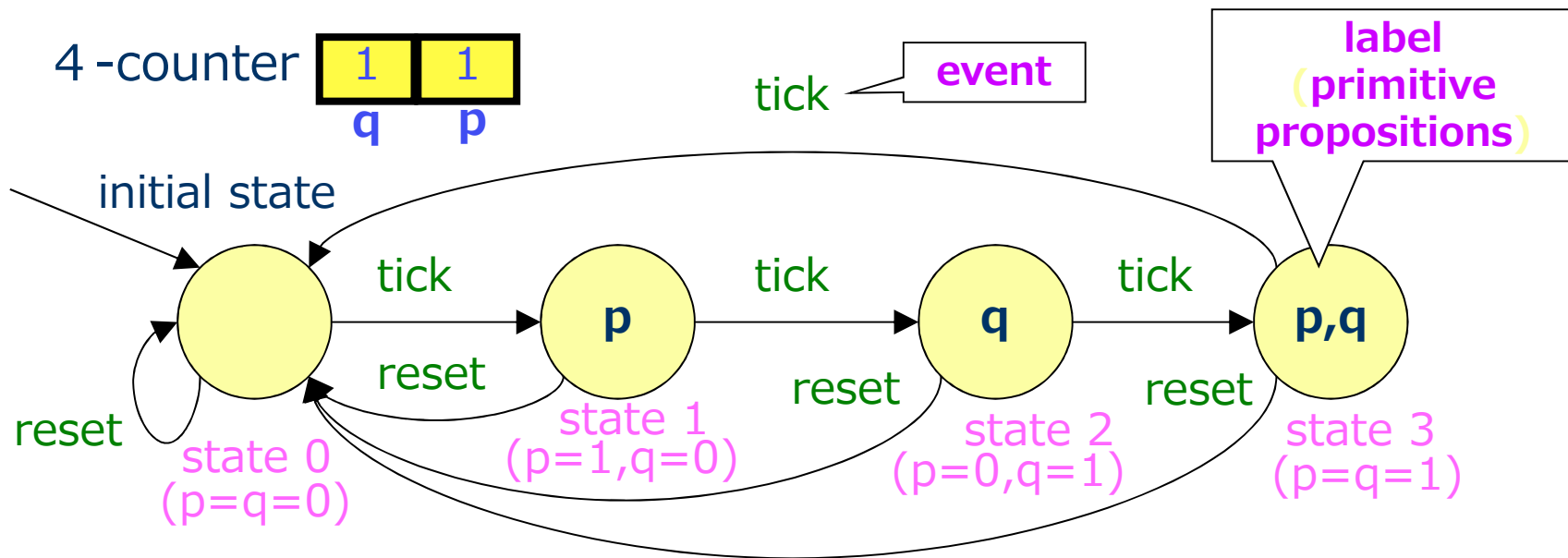
Analysis of input-output relationships such as **termination plus partial correctness** is not enough to prove the correctness of concurrent systems.

→ **Behaviors** (i.e., transitions of intermediate states) need to be considered.

Model checking is a technology that automatically **verifies the behaviors** of finite-state concurrent systems.

---

# 状態遷移系 (オートマトン) (State transition system; automaton)



状態数は数百万くらいはOK  
Millions of states are OK in practice.

## 検証できる主な性質 (Major verifiable properties)

---

モデル検査では主につぎの2つの性質を検証する

Basically, the following two properties can be verified by model checking.

### □ 安全性

「悪いことは決して起こらない」

例：このエレベータは、ドアが開いたまま昇降することは決してない

- **safety**, which ensures that the bad thing will never happen.

Example: This elevator will never start to move while its door is still open.

### □ 活性

「良いことはいつかは成り立つ」

例：資源を要求したら、いつか必ず取得できる

- **liveness**, which ensures that the good thing will eventually happen.

Example: If you request a resource, you will eventually acquire it.

---



## モデル検査の実施手順 (Model checking processes)

---

モデル検査はつぎの3ステップで実施する Model checking is conducted in the following three processes.

(1)モデリング (modeling)

(2)性質の記述 (description of properties)

(3)検査 (verification)

---

## (1) モデリング (Modeling)

---

- システムの振る舞いを表すモデル (状態遷移系) を記述する.
  - 記述には一般に, モデル記述言語を用いる.
  - プロセス代数に基づく言語やC言語風の言語などがある.
  - Describe a model (i.e., state transition system) that defines the behavior of the system.
  - A model description language is used for the description.
  - Languages based on process algebra and C-like ones are available.
-

## (2) 性質の記述 (Description of properties)

- システムが満たすべき性質（プロパティ）を記述する。
- Describe **properties** the system is expected to satisfy.
- 記述には一般的に、時間の概念を扱う**時相論理**を用いる。
- **Temporal logics**, which can represent the notion of time, are used for the description.
- 時相論理として、**LTL**と**CTL**がよく知られている。
- Two well-known temporal logics are **LTL** and **CTL**.

**LTL** (Linear Temporal Logic)

**CTL** (Computation Tree Logic)

【LTL Example】 **G(Req → F Ack)**

Reqが真になると、その後いつか必ずAckが真になる

Whenever Req is ON, then Ack will eventually become ON in the future.



It is **globally (always)** true that  
if Req is true then in **future** Ack will be true.

## (3) 検査 (Checking)

---

モデルが、記述された性質を満たすかどうか、**検査**する。

### モデル検査器

- 性質が成り立つ場合：検査終了
- 性質が成り立たない場合：  
反例（エラートレース）を出力

### モデル検査アルゴリズムの原理

すべての可能な入力について計算経路を網羅的に検査

Check to see if the model satisfies the described properties.

A model checker is used for checking.

- if the properties hold, the check is finished.
- Otherwise, a counterexample (error trace) is output.

The principle of model checking algorithms is to examine all possible computation paths for all possible input.

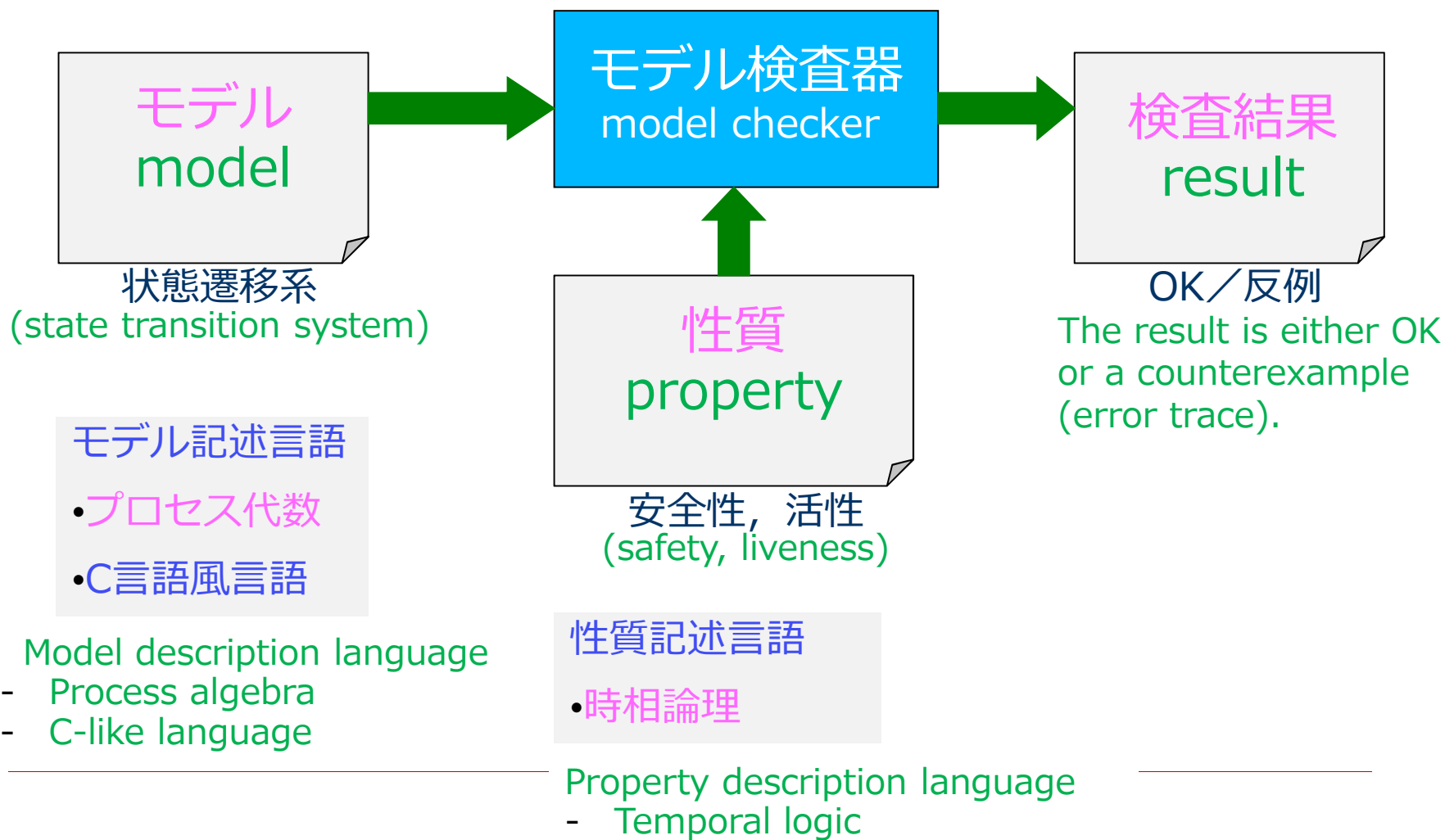
---

## モデル検査器の例 (Well-known model checkers)

---

- **SMV, NuSMV**
    - カーネギーメロン大学 (Carnegie Mellon University)
    - IEEE Futurebus+ standardのバグ発見 (Found bugs of an IEEE bus)
  - **SPIN**
    - ベル研究所 (Bell Laboratories)
    - ACM Software System Award受賞 (Received the ACM SS Award)
  - **LTSA**
    - ロンドン王立大学 (Imperial College London)
    - FSP言語, アニメーション (FSP Language, animation)
  - **Java PathFinder (JPF)**
    - NASA (National Aero. & Space Admin.)
    - Javaのバイトコードの検査 (Checks Java byte code)
-

# モデル検査器の概要 (Overview of model checker)



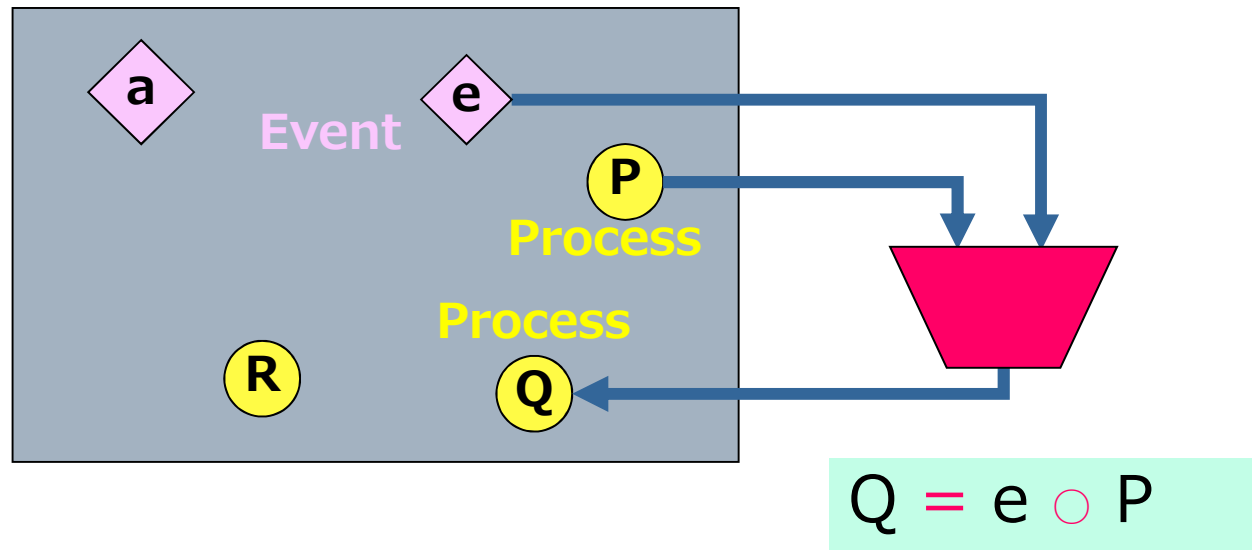
### 3. プロセス代数に基づく逐次プロセスのモデリング (Modeling sequential processes based on process algebra)

プロセス代数とは:

プロセスやイベント等の集合を台とし、  
プロセスの合成などの演算を定義した代数

process algebra:

an algebra consisting of processes  
and events as the universe together  
with operations such as process  
compositions.



## アクション接頭辞 (Action prefix)

---

**(a -> P)**

最初にアクション **a** を実行し, つぎにプロセス **P**と同じ振る舞いをするプロセス.

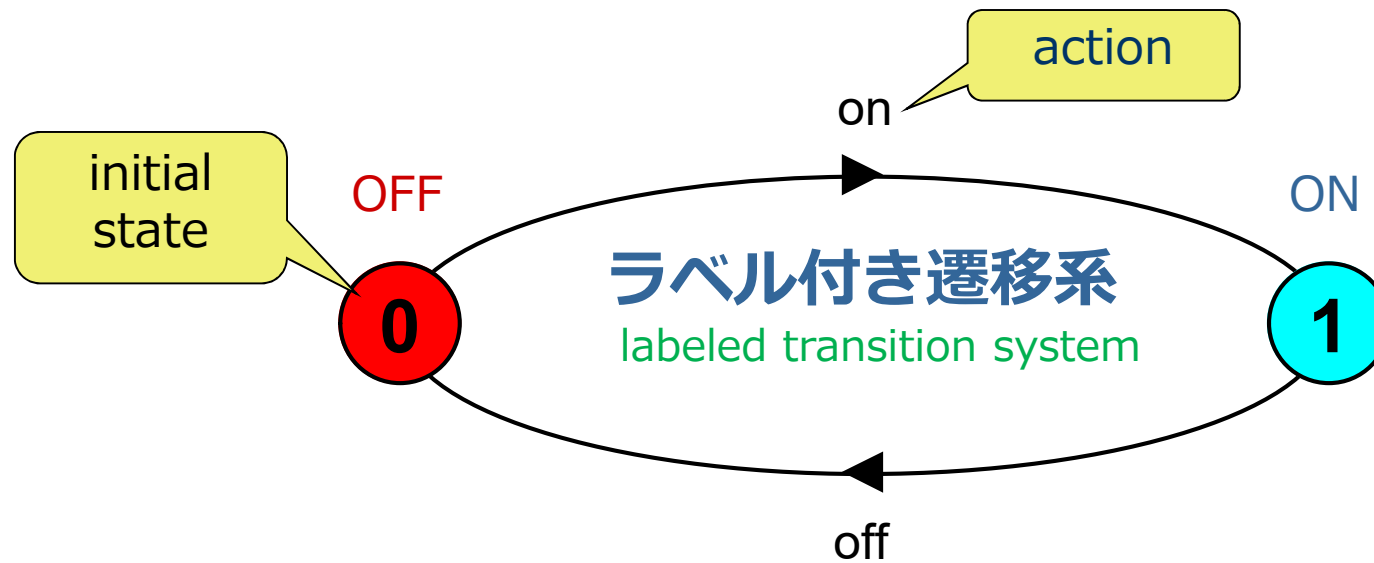
(a -> P) is a process that initially engages in the action **a** and then behaves exactly as described by the process **P**.

---



## スイッチの例(1) (Example of a binary switch)

---



トレース : 実行可能なアクションの列のこと

Trace: a possible action sequence

on→off→on→off→on→off→ …………….

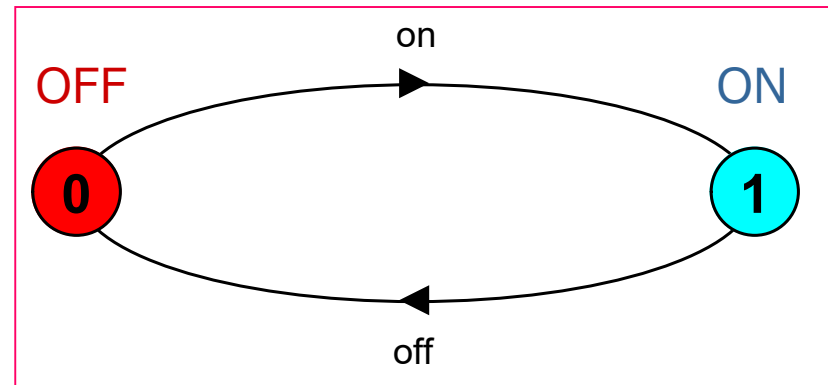
---

## スイッチの例(2)

SWITCH = OFF,  
OFF = (on -> ON),  
ON = (off -> OFF).

Process name  
in capitals

Action name  
in lower-case



代入によってより簡潔な表現を得る

(Simplify the expressions by substitution)

SWITCH = OFF,  
OFF = (on -> (off->OFF)).

SWITCH = (on->off->SWITCH).

## 選択 (Choice)

---

$(a \rightarrow P \mid b \rightarrow Q)$

最初にアクション  $a, b$  のいずれかを実行する。  
そのアクションが  $a$  ならつぎにプロセス  $P$  を実行し、  
 $b$  ならつぎにプロセス  $Q$  を実行するプロセス。

$(a \rightarrow P \mid b \rightarrow Q)$  is a process which initially engages in either  $a$  or  $b$ .  
The subsequent behavior is described by  $P$  if the first action was  $a$ ,  
and  $Q$  if the first action was  $b$ .

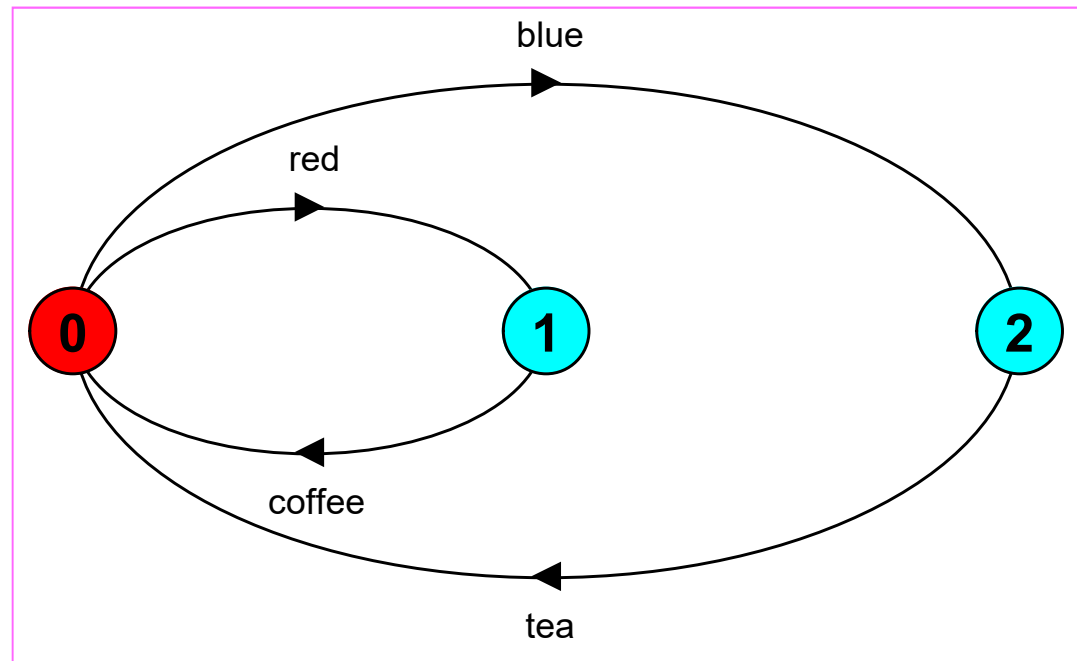
---

## 自動販売機の例 (Example of a vending machine)

DRINKS = ( red->coffee->DRINKS  
| blue->tea->DRINKS  
).

ボタンの色で飲み物を指定

Select a drink by the color of the button



## 4. 並行プロセスのモデリング (Modeling concurrent processes)

---

### プロセスの並列合成 (parallel composition of processes)

$(P||Q)$

プロセス P と Q の並行実行を表すプロセス。  
|| は並列合成演算子。

This expression means the process in which processes P and Q are executed in parallel.

The symbol || denotes the parallel composition operator.

---

# アクションのインタリーブ (Action interleaving)

かゆいところをかくプロセス

a process that itches

```
ITCH = (scratch->STOP).  
CONVERSE = (think->talk->STOP).
```

} 共有アクション無し  
no shared actions

会話するプロセス

a process that  
converses

```
|| CONVERSE_ITCH = (ITCH || CONVERSE).
```

並列合成は || で書き始める

並列合成されたプロセス

the process created by  
parallel composition

A parallel composition statement  
starts with ||.

プロセス間で共有されない  
アクションはインタリーブ  
される

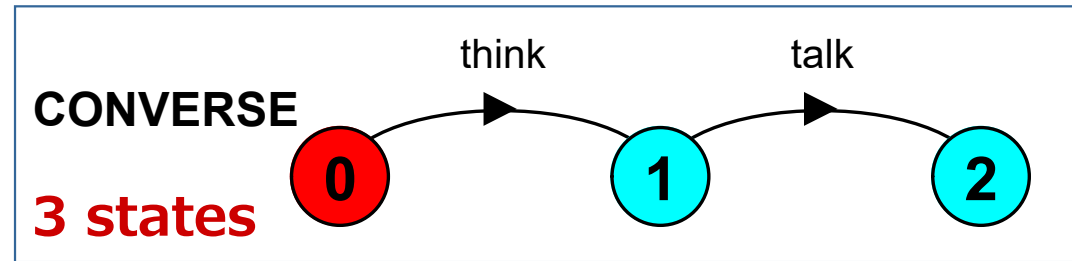
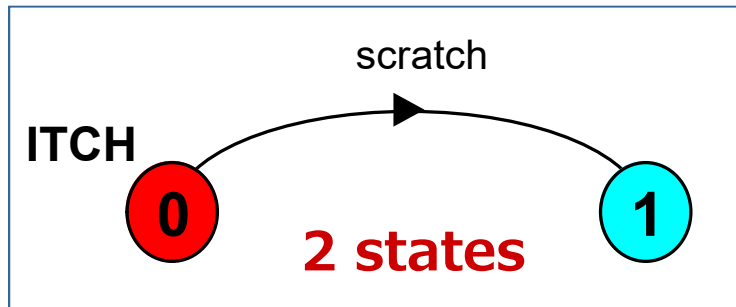
Actions not shared among  
the processes will be  
interleaved.

```
think→talk→scratch  
think→scratch→talk  
scratch→think→talk
```

} インタリーブによって  
可能となるトレース  
the traces made possible by  
interleaving

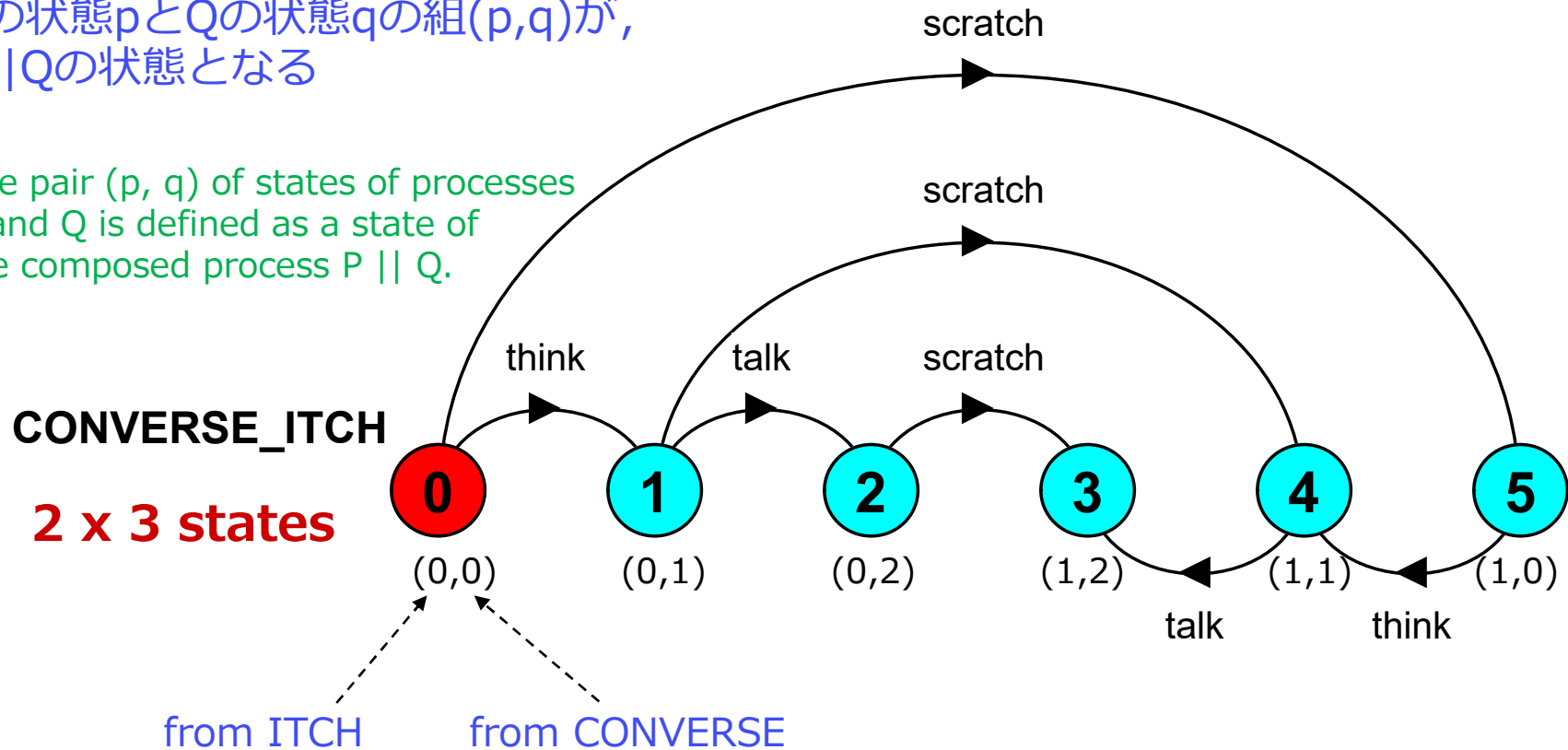
# ラベル付き遷移系の並列合成

(Parallel composition of labeled transition systems)



Pの状態pとQの状態qの組(p,q)が、 $P \parallel Q$ の状態となる

The pair (p, q) of states of processes P and Q is defined as a state of the composed process  $P \parallel Q$ .



# 共有アクションとインタラクション

(Shared action and interaction)

---

共有アクション：並列合成されたプロセスがもつ共通のアクション。

共有アクションによってプロセスのインタラクションをモデル化。

共有アクションは、それを共有するすべてのプロセスにおいて同時に同期して実行されなければならない。

A shared action is an action shared among the parallel processes.

Shared actions are used to model interactions among those processes.

Shared actions must be executed at the same time by all processes that share them.

In other words, those executions must be synchronized.

---



# インタラクションの例

(Example of interactions)

shared actions:  
ready, used

```
MAKER = (make->ready->used->MAKER).
```

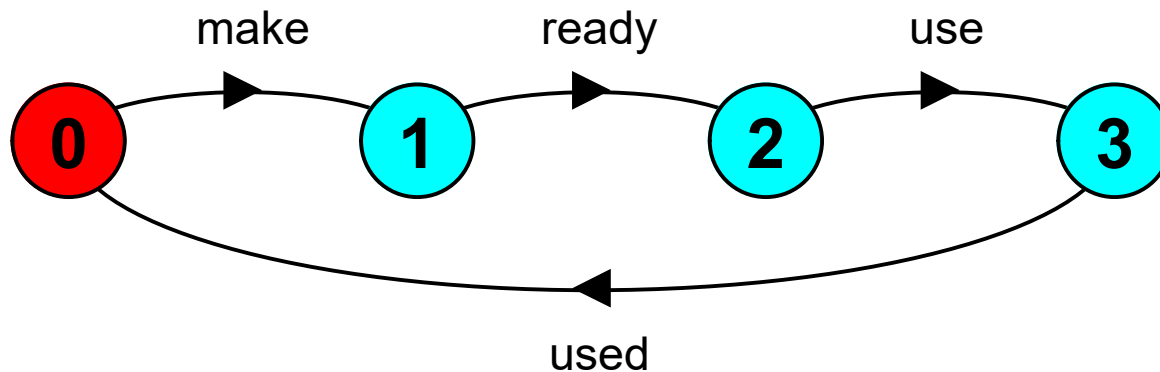
```
USER = (ready->use->used->USER).
```

```
|| MAKER_USER = (MAKER || USER).
```

3 状態

3 状態

3 x 3 状態?



4 状態

インタラクションは  
振舞いを制約する

Interactions constrain the behavior.

## 演習問題 3

### EXERCISE 3

二人のユーザーを表すプロセスUSER\_A, USER\_B, および共有資源を表すプロセスRESOURCE, およびそれらを並列合成したプロセスRESOURCE\_SHAREが, つぎのように定義されている. これら4つのプロセスのラベル付き遷移系を描画せよ.  
ヒント: a\_acquire, a\_release, b\_acquire, b\_release は共有アクションである.

The processes USER\_A and USER\_B, which represent two users, and RESOURCE, which represents a shared resource, are combined by the parallel composition to define RESOURCE\_SHARE. Draw the labeled transition systems for these four processes.

Hint: a\_acquire, a\_release, b\_acquire, b\_release are shared actions.

```
USER_A = (a_acquire -> a_use -> a_release -> USER_A).
USER_B = (b_acquire -> b_use -> b_release -> USER_B).
RESOURCE = IDLE,
    IDLE = ( a_acquire -> BUSY
            | b_acquire -> BUSY
            ),
    BUSY = ( a_release -> IDLE
            | b_release -> IDLE
            ).
|| RESOURCE_SHARE = (USER_A || USER_B || RESOURCE).
```