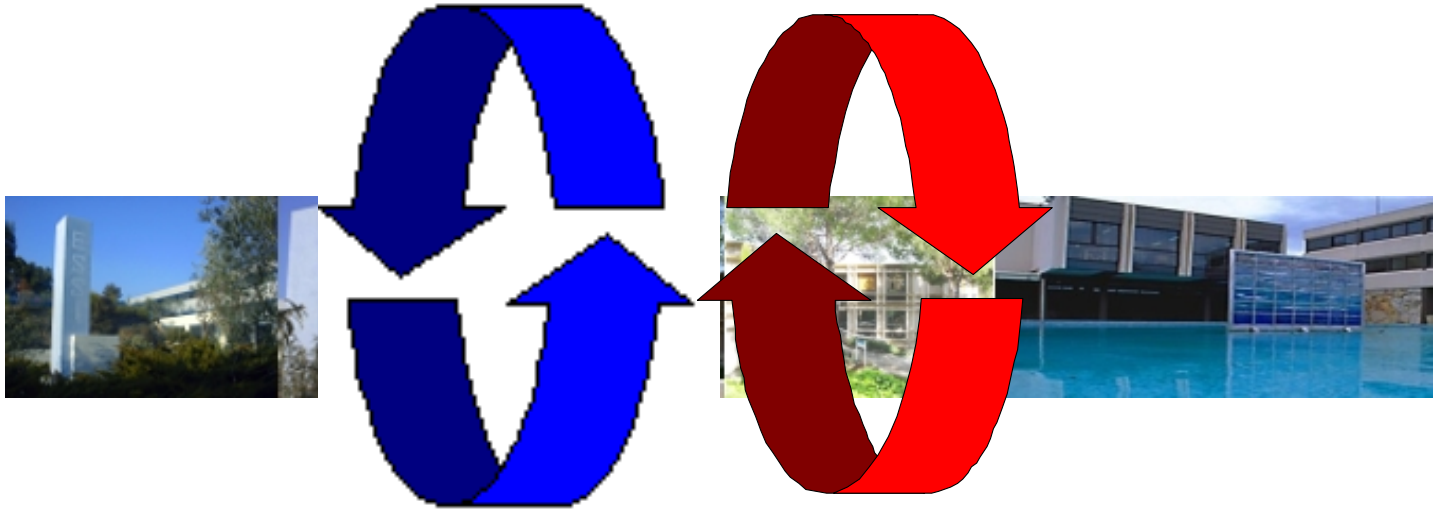
Modélisation des processus

riveill@unice.fr

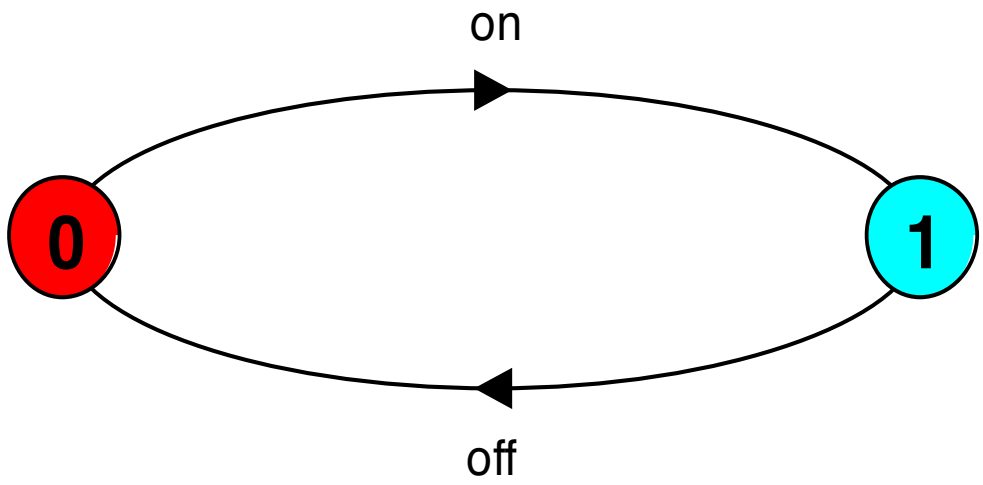
http://www.i3s.unice.fr/~riveill



Step 1 : modeling sequential processes

Modeling processes

- A process is the execution of a sequential program. It is modeled as a finite state machine which transits from state to state by executing a sequence of atomic actions.
- A light switch LTS

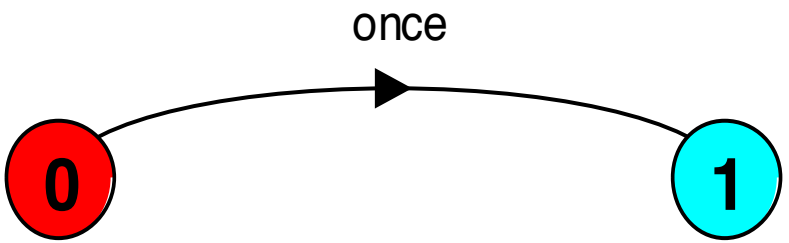


- A sequence of actions or *trace*
  - `on→off→on→off→on→off→ .....`
- *Can finite state models produce infinite traces?*

FSP - action prefix

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

- ONESHOT state machine (terminating process)
  - `ONESHOT = (once -> STOP).`

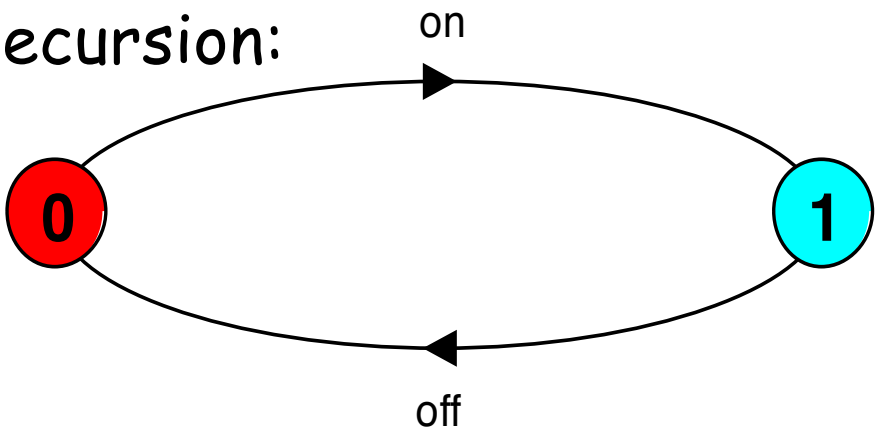


Convention:
- actions begin with lowercase letters
- PROCESSES begin with uppercase letters

FSP - action prefix & recursion (*infinite traces)*

- Repetitive behaviour uses recursion:

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



- Substituting to get a more succinct definition:

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

- And again:

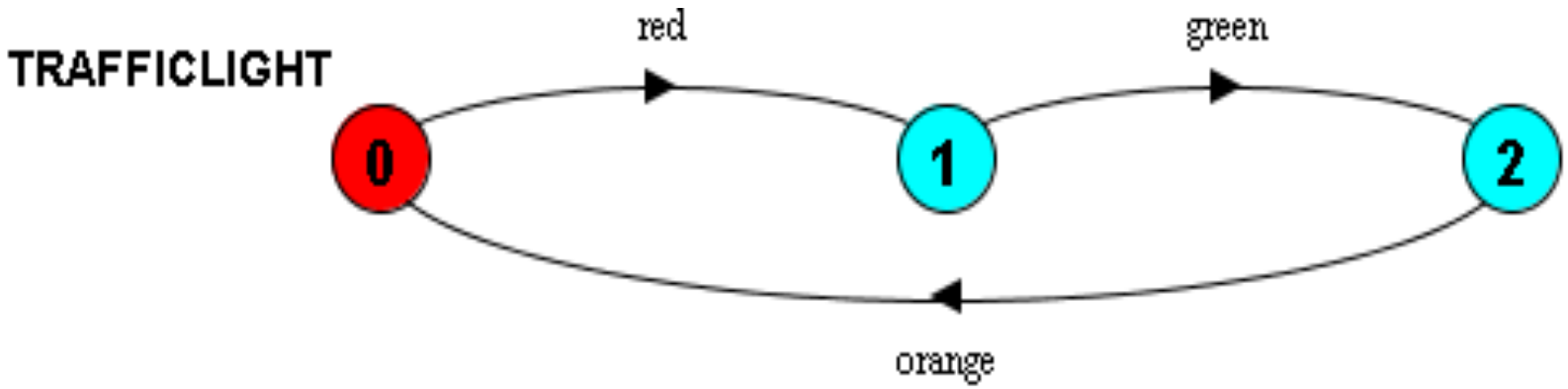
```
SWITCH = (on->off->SWITCH).
```

TEST

- Model in FSP a traffic light

```
TRAFFICLIGHT = (red->green->orange->TRAFFICLIGHT).
```

- Design in LTS a traffic light



- What is the trace ?
  - `red→green→orange→red→green` …
- What is the alphabet ?
  - `{red, green, orange}`

FSP - choice

- If **x** and **y** are actions then **(x-> P | y-> Q)** describes a process which initially engages in either of the actions **x** or **y**. After the first action has occurred, the subsequent behavior is described by **P** if the first action was **x** and **Q** if the first action was **y**.

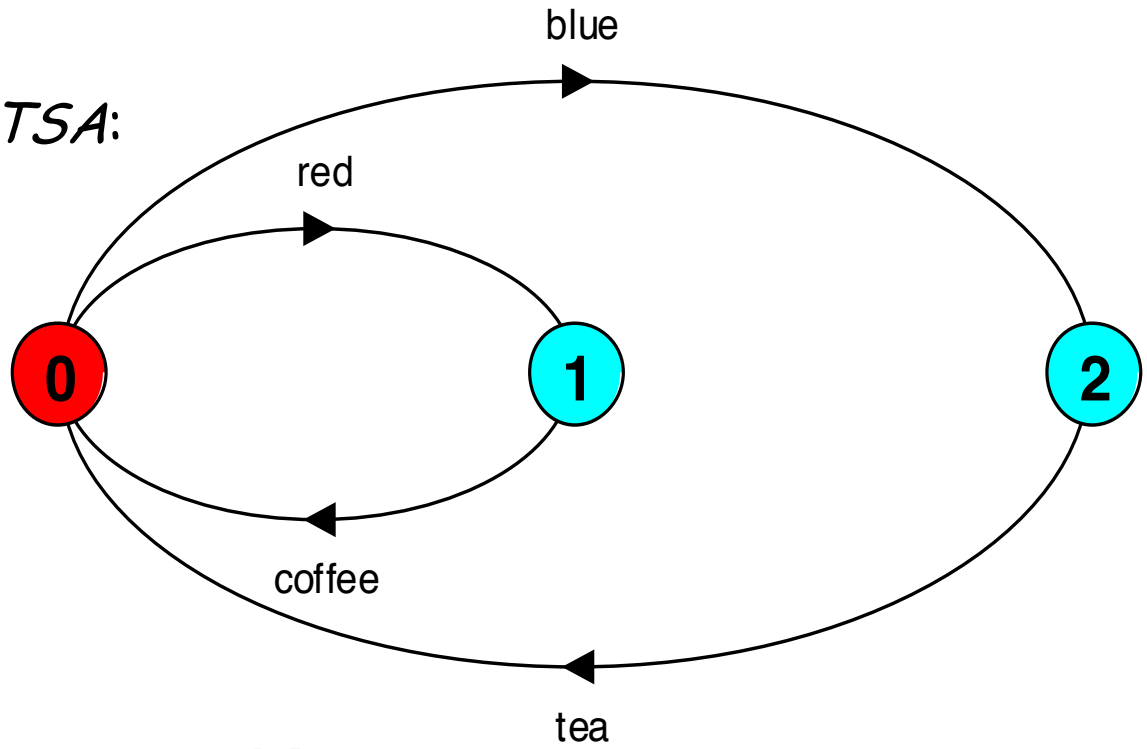
- *Who or what makes the choice?*
- *Is there a difference between input and output actions?*

FSP - choice

- FSP model of a drinks machine :

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

- LTS generated using *LTSA*:



- Possible traces?

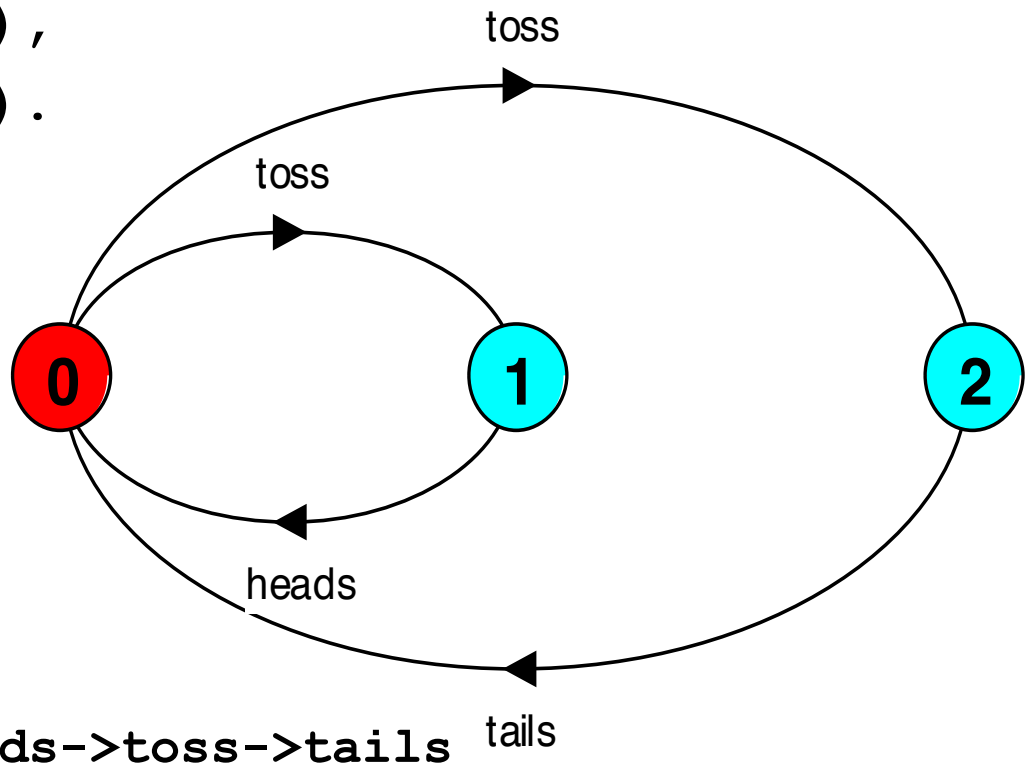
```
red->coffee->blue->tea->blue->tea->
```

Non-deterministic choice

- Process (x-> P | x -> Q) describes a process which engages in x and then behaves as either P or Q.

```
COIN = (toss->HEADS|toss->TAILS),
HEADS= (heads->COIN),
TAILS= (tails->COIN).
```

- Tossing a coin.



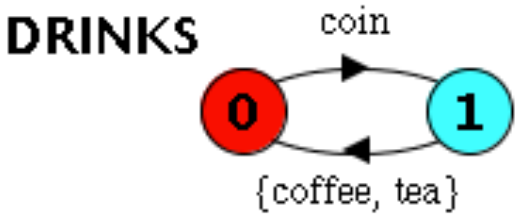
- Possible traces?
  - `toss->heads->toss->heads->toss->tails`

TEST

- Model in FSP a random drinks machine

```
DRINKS = (coin->{coffee,tea}->DRINKS).
```

- Design in LTS a drinks machine



- What is the trace ?
  - `coin->coffee->coin->coffee->coin->tea->...`
- What is the alphabet ?
  - `{coin,coffee,tea}`

FSP - indexed processes and actions

- Single slot buffer that inputs a value in the range 0 to 3 and then outputs that value:

```
BUFF = (in[i:0..3]->out[i]-> BUFF).
```

- equivalent to

```
BUFF = (in[0]->out[0]->BUFF
       |in[1]->out[1]->BUFF
       |in[2]->out[2]->BUFF
       |in[3]->out[3]->BUFF
       ).
```

indexed actions generate labels of the form *action.index*

- or using a process parameter with default value:

```
BUFF(N=3) = (in[i:0..N]->out[i]-> BUFF).
```

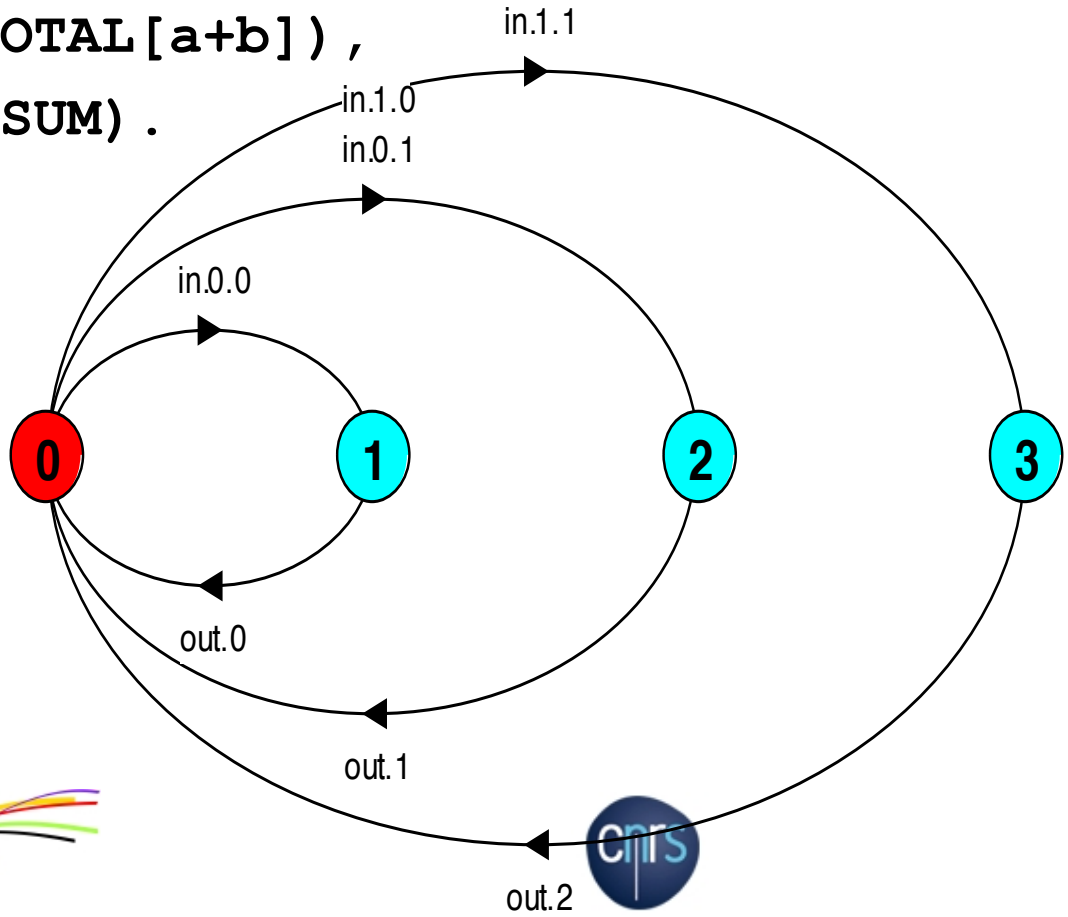
- Alphabet = {in.0, in.1, in.2, in.3,
  out.0, out.1, out.2, out.3}

FSP - indexed processes and actions

- index expressions to model calculation:

```
const N = 1
range T = 0..N
range R = 0..2*N
SUM = (in[a:T][b:T]->TOTAL[a+b]),
TOTAL[s:R] = (out[s]->SUM).
```

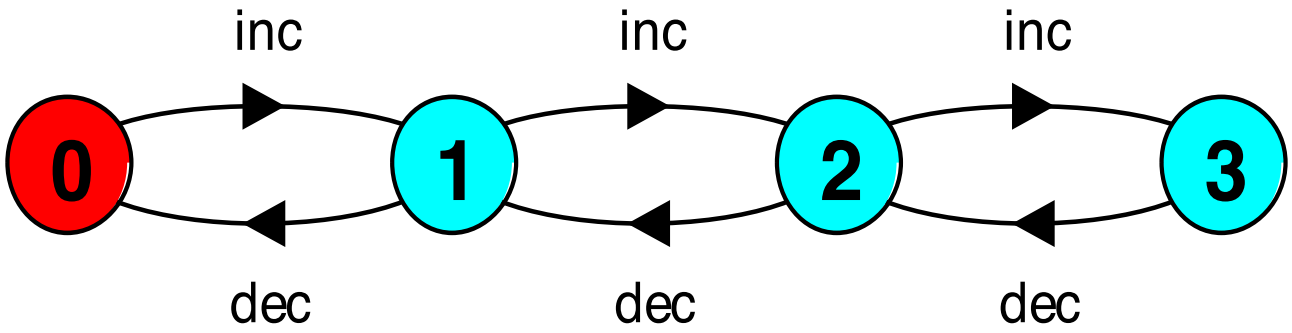
Local indexed process definitions are equivalent to process definitions for each index value



FSP - guarded actions

- The choice **(when B x -> P | y -> Q)** means that when the guard **B** is true then the actions **x** and **y** are both eligible to be chosen, otherwise if **B** is false then the action **x** cannot be chosen.

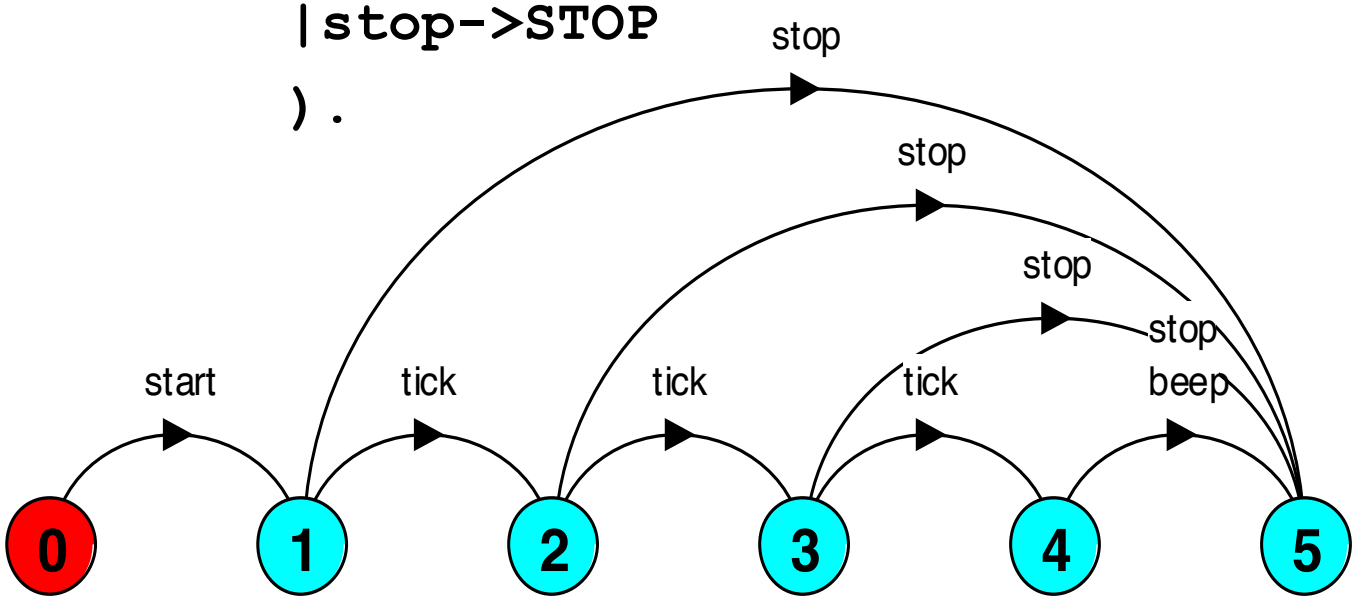
```
COUNT  (N=3)     = COUNT[0],
COUNT[i:0..N] = (when(i<N)  inc->COUNT[i+1]
                |when(i>0)  dec->COUNT[i-1]
                ).
```



FSP - guarded actions

- A countdown timer which beeps after N ticks, or can be stopped.

```
COUNTDOWN  (N=3)     = (start->COUNTDOWN[N]),
COUNTDOWN[i:0..N] =
            (when(i>0)  tick->COUNTDOWN[i-1]
            |when(i==0)beep->STOP
            |stop->STOP
            ).
```



FSP - process alphabets

- The alphabet of a process is the set of actions in which it can engage.
- Process alphabets are implicitly defined by the actions in the process definition.
- The alphabet of a process can be displayed using the LTSA alphabet window.

```
Process:
     COUNTDOWN
Alphabet:
     { beep,
       start,
       stop,
       tick
     }
```

FSP - process alphabet extension

- Alphabet extension can be used to extend the implicit alphabet of a process:

  - *implicit*

    ```
    WRITER = (write[1]->write[3]->WRITER).
    ```

  - Alphabet of `WRITER` is the set `{write.1, write.3}`

  - *explicit*

    ```
    WRITER = (write[1]->write[3]->WRITER)
                +{write[0..3]}.
    ```

  - Alphabet of `WRITER` is the set `{write[0..3]}`

FSP - process alphabet adjustment

- The implicit alphabet of a process can be extended and/or reduced, by two kinds of suffixes to a process description P:
  - extension `P + {…}`
  - hiding `P\{…}`
- Examples:

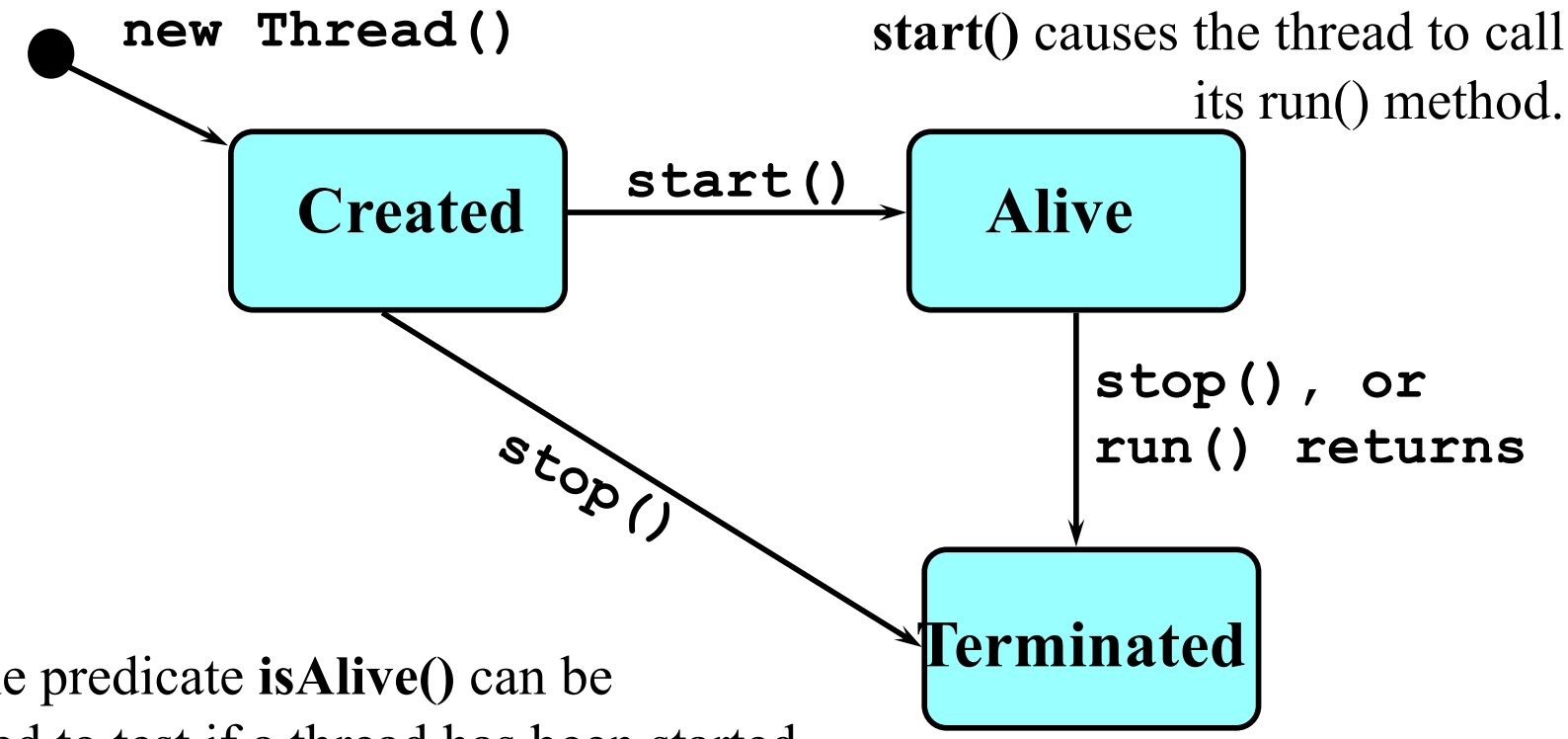
  `MEALS = (breakfast->lunch->dinner->MEALS)\{lunch}.`
  - Now “lunch” becomes an internal action, *tau*, not visible nor shared.
  - You want to eat alone.

  LISTEN = ({latin, jazz, pop}->LISTEN)+{hiphop}.
  - You do not want to listen to hiphop, and block on hiphop actions.

- We make use of alphabet extensions in later lectures

thread life-cycle in Java

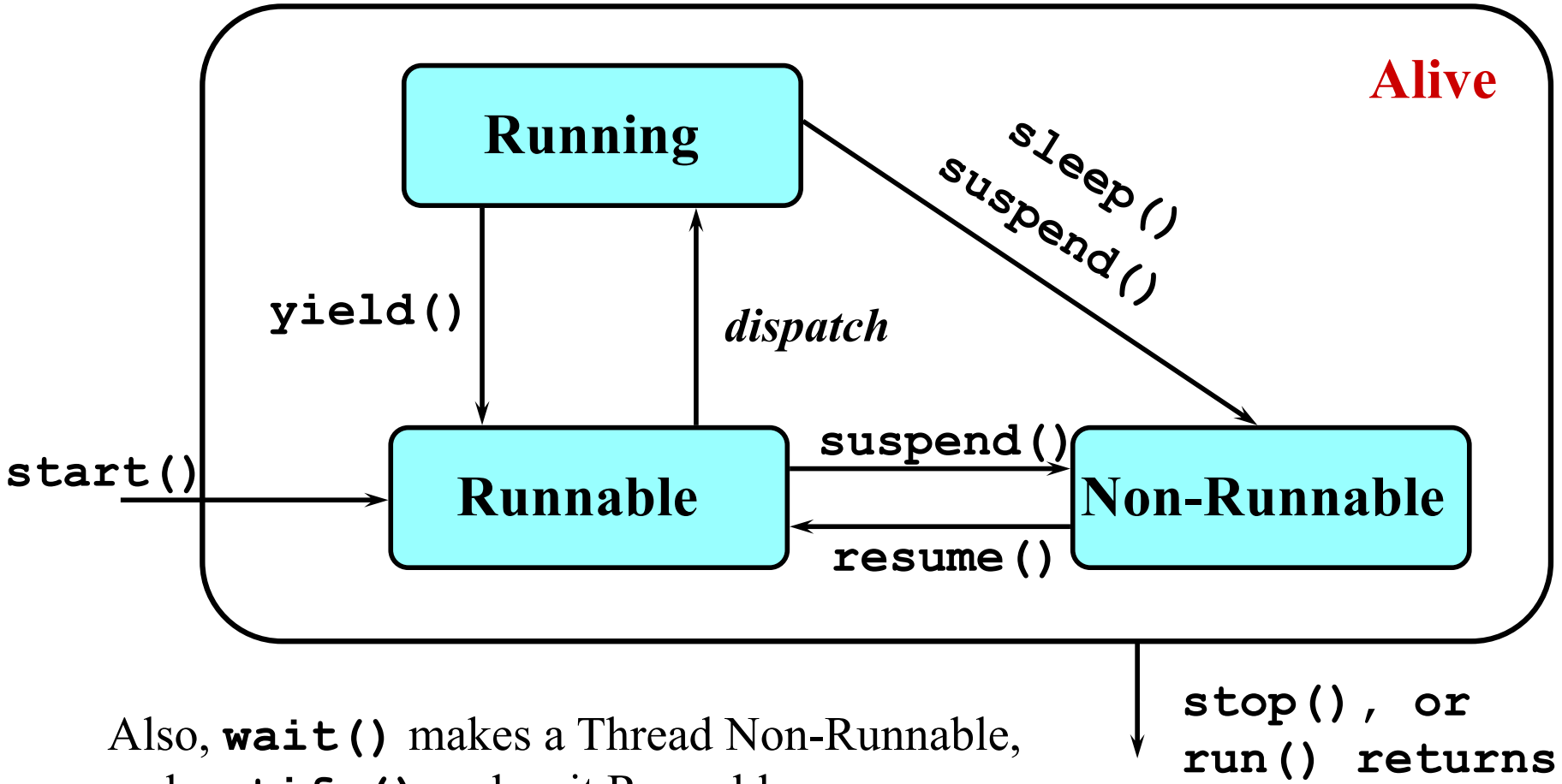
An overview of the life-cycle of a thread as state transitions:



The predicate **isAlive()** can be used to test if a thread has been started but not terminated. Once terminated, it cannot be restarted (cf. mortals).

thread alive states in Java

Once started, an **alive** thread has a number of substates :



Also, **`wait()`** makes a Thread Non-Runnable, and **`notify()`** makes it Runnable (used in later chapters).

Java thread lifecycle - an FSP specification

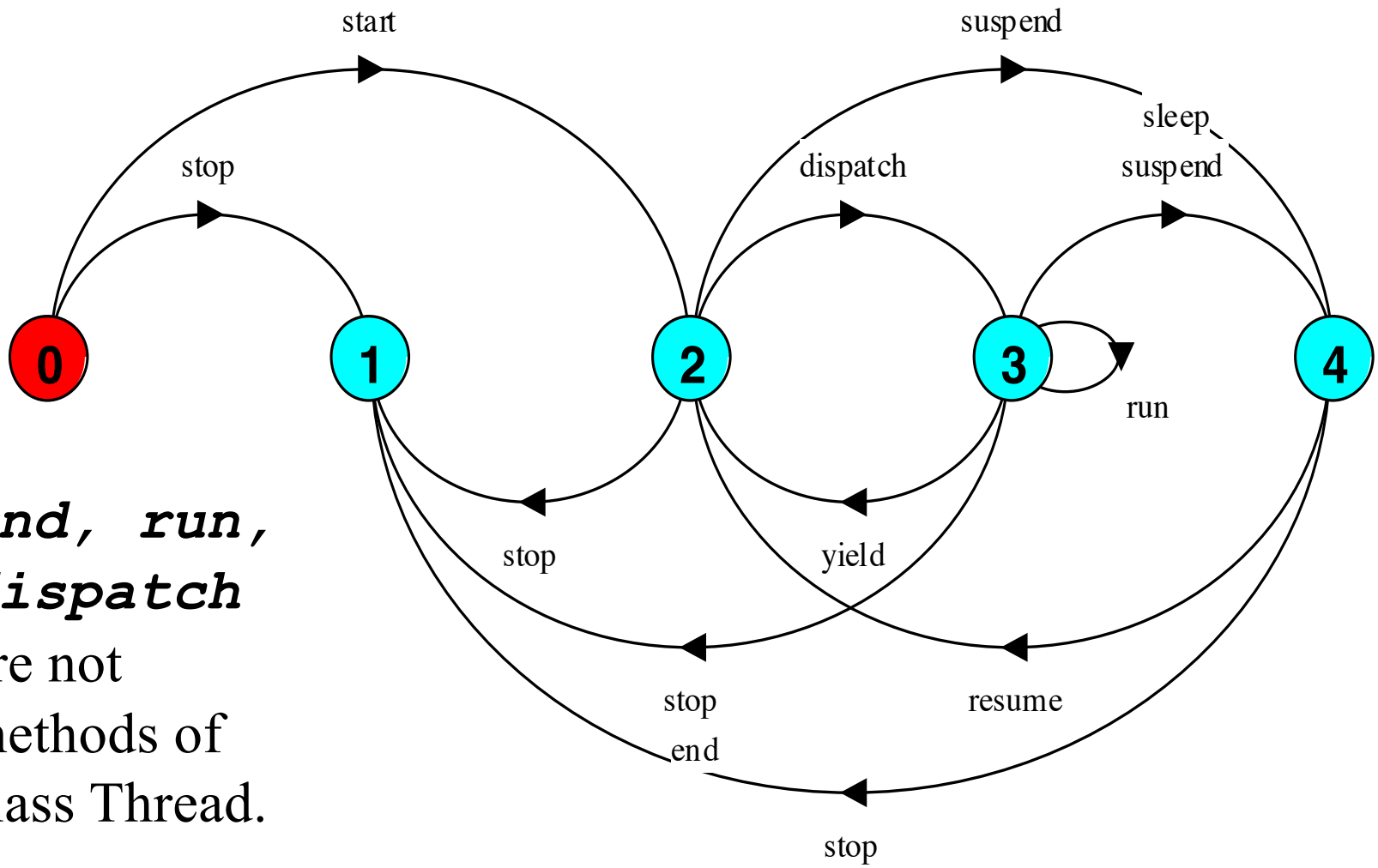
```
THREAD          = CREATED,
CREATED         = (start                ->RUNNABLE
                  |stop                 ->TERMINATED),
RUNNING         = ({suspend,sleep}      ->NON_RUNNABLE
                  |yield                ->RUNNABLE
                  |{stop,end}           ->TERMINATED
                  |run                  ->RUNNING),
RUNNABLE        = (suspend              ->NON_RUNNABLE
                  |dispatch             ->RUNNING
                  |stop                 ->TERMINATED),
NON_RUNNABLE    = (resume               ->RUNNABLE
                  |stop                 ->TERMINATED),
TERMINATED      = STOP.
```

Université Nice Sophia Antipolis

i3S sophia antipolis

cnrs

Java thread lifecycle - an FSP specification



end, run, dispatch are not methods of class Thread.

Université Nice Sophia Antipolis

States 0 to 4 correspond to **`CREATED`**, **`TERMINATED`**, **`RUNNABLE`**, **`RUNNING`**, and **`NON-RUNNABLE`** respectively.