CAL Actor Language: Motivation

- Writing simple actors should be simple.
  - Ptolemy II API very rich
  - actor writing requires considerable skill
  - **BUT: Actors have a lot of common structure.**
- Models should allow efficient code generation.
  - actor descriptions contain a lot of "admin" code

➔ **Generate** actors from a more abstract description.

  - reduces amount of code to be written
  - makes writing actors more accessible
  - reduces error probability
  - makes code more versatile
    - retargeting (other platforms, new versions of the Ptolemy API)
    - analysis & composition

Src: J. Janneck, CAL – An actor language, 2003

Simple actors

```
actor ID () In ==> Out :

    action [a] ==> [a] end
end
```

Name, Parameter(s), Ports

```
actor A (k) Input1, Input2 ==> Output:

    action [a], [b] ==> [k*(a + b)] end
end
```

Port Pattern(s)

```
actor Merge ()
    Input1, Input2 ==> Output:

    action Input1: [x] ==> [x] end
    action Input2: [x] ==> [x] end
end
```

non-deterministic example

```
actor C ()
    Double Input ==> Double Output:

  action [a] ==> [a] where a >= 0 end

  action [a] ==> [-a] where a < 0 end
end
```

deterministic example

actor firing ≡ execution of one *enabled* action

Src: J. Janneck, CAL – An actor language, 2003

Stateful Actors

```
actor Sum () Input ==> Output:

    sum := 0;

    action [a] ==> [sum]
    do
        sum := sum + a;
    end
end
```

action
- **input patterns**
  declaring variables
- **guard**
  specifying enabling conditions
- **output expressions**
  computing output tokens
- **body**
  modifying the actor state

Action Guards

```
actor FairMerge ()
    Input1, Input2 ==> Output:

    s := 0;

    action Input1: [x] ==> [x]
    guard s = 0
    do
        s := 1;
    end

    action Input2: [x] ==> [x]
    guard s = 1
    do
        s := 0;
    end
end
```


Src: J. Janneck, CAL – An actor language, 2003

Action schedules

```
actor FairMerge ()
    Input1, Input2 ==> Output:

    s := 0;

    action Input1: [x] ==> [x]
    guard s = 0
    do
        s := 1;
    end

    action Input2: [x] ==> [x]
    guard s = 1
    do
        s := 0;
    end
end
```

```
actor FairMerge ()
    Input1, Input2 ==> Output:

  A: action Input1: [x] ==> [x] end
  B: action Input2: [x] ==> [x] end

    schedule fsm State0:
        State0 (A) --> State1;
        State1 (B) --> State0;
    end
end
```

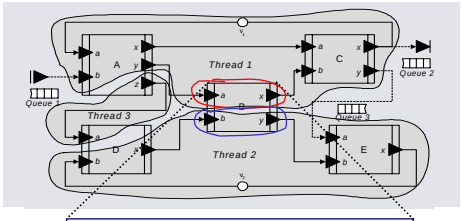
```
actor FairMerge ()
    Input1, Input2 ==> Output:

  A: action Input1: [x] ==> [x] end
  B: action Input2: [x] ==> [x] end

    schedule regexp
        (A B)*
    end
end
```


Src: J. Janneck, CAL – An actor language, 2003

Executing CAL: Discovering concurrency




```
actor B () a, b ==> x, y:

    s := <something>;

    action a:[v] ==> x:[f(v, s)] end

    action b:[v] ==> y:[g(v)]
    do
        s := h(v, s);
    end
end
```

Src: J. Janneck, CAL – An actor language, 2003

References & Further Reading

- Dataflow Process Networks
  - E. A. Lee and T. M. Parks, Dataflow Process Networks, Proc. of the IEEE, vol. 83, no. 5, pp. 773-801, May, 1995. http://ptolemy.eecs.berkeley.edu/publications/papers/95/processNets
  - Krzysztof Kuchcinski, Class notes, EDA 380, Design of Embedded Systems, 2002
- Actor-oriented Modeling:
  - Actor-oriented Models for Codesign: Balancing Re-Use and Performance, Edward A. Lee and Stephen Neuendorffer, in Formal Methods and Models for System Design, Kluwer, 2004 http://ptolemy.eecs.berkeley.edu/publications/papers/04/FormalCodesign/FormalCodesign.pdf
- Caltrop Project/Cal Actor Language
  - Janneck et al., http://embedded.eecs.berkeley.edu/caltrop
  - CAL Language Report, J. Eker, J. Janneck http://embedded.eecs.berkeley.edu/caltrop/docs/LanguageReport/CLR-1.0-r1.pdf