Introduction to Behavioral Programming In Java

February 2012

Weizmann Institute of Science

Ben Gurion University

Team members

Past and present – more or less in chronological order – updated 6/2012.

- David Harel (*)
- Rami Marelly
- Hillel Kugler
- Shahar Maoz
- Itai Segall
- Yoram Atir
- Asaf Kleinbrot
- Dan Barak
- Avital Sadot
- Amir Kantor
- Michal Gordon
- Tal Berger
- Smadar Szekely (*)
- Assaf Marron (*)
- Gera Weiss (*)
- Daniel Barkan
- Guy Weiss
- Yaarit Natan
- Amir Nissim
- Robby Lampert
- Guy Wiener
- Yaniv Sa’ar
- Nir Eitan
- Guy Katz
- Michael Bar-Sinai
- Moshe Weinstock
- Ronen Brafman

Weizmann Institute
of Science

Ben Gurion University

(*) contact persons for the team

The Behavioral Programming Vision

Can complex software be developed from

simple threads of behavior

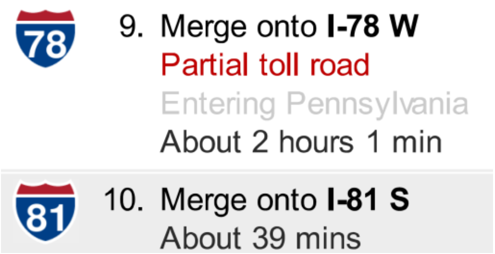
by

automatic interweaving ?

Humans interweave behavior threads all the time…

Driving Directions

…



9. Merge onto **I-78 W**
   Partial toll road
   Entering Pennsylvania
   About 2 hours 1 min
10. Merge onto **I-81 S**
    About 39 mins

…

Daily Schedule

…

Drive for 4 hrs.

Stop for Lunch

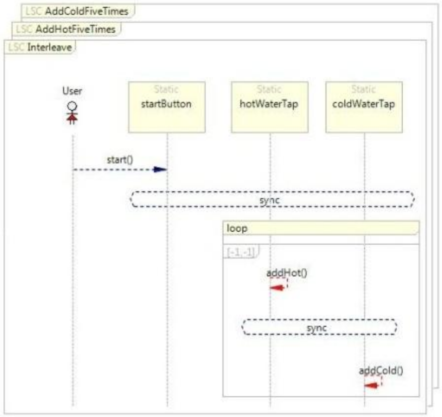
Drive for 5 hrs.

…

A 6-day trip from NYC to LA

… can software be developed this way?

LSC & BPJ: From requirements to code



LSC: A visual language for scenario specification

- Damm and Harel 2001, Harel and Marelly 2003
- Natural yet executable scenario-based specification
- Initially for requirement specification, evolved into a programming language
- PlayGo – an IDE for programming with LSC

BPJ: A package for programming scenarios in Java

(and equivalents for other languages)

```
class AddHotFiveTimes extends BThread {
    public void runBThread() {
        for (int i=1; i<=5; i++) {
            bSync(addHot, none, none);
        }
    }
}
```

- Harel, Marron, and Weiss 2010
- Bringing advantages of scenario-based specification to programming
- Integrate with & complement other paradigms (OOP, aspects, rule-based, agile, …).

Incremental development in Java with BPJ

Behavior Threads

Req. 3.1

```
class AddHotFiveTimes extends BThread {
    public void runBThread() {
        for (int i=1; i<=5; i++) {
            bSync(addHot, none, none);
        }
    }
}
```

Req. 5.2.9

```
class AddColdFiveTimes BThread {
    public void runBThread() {
        for (int i=1; i<=5; i++) {
            bSync(addCold, none, none);
        }
    }
}
```

Patch 7.1

```
class Interleave extends BThread {
    public void runBThread() {
        while (true) {
            bSync(none, addHot, addCold);
            bSync(none, addCold, addHot);
        }
    }
}
```

Why do we need this?

A key benefit: incremental development

Need to accommodate a cross-cutting requirement? **Add a module**

Need to refine an inter-object scenario? **Add a module**

Need to remove a behavior? **Add a module**

. . . ? **Add a module**

No need to modify existing code

Behavior execution cycle

1. All behavior threads (b-threads) post declarations:

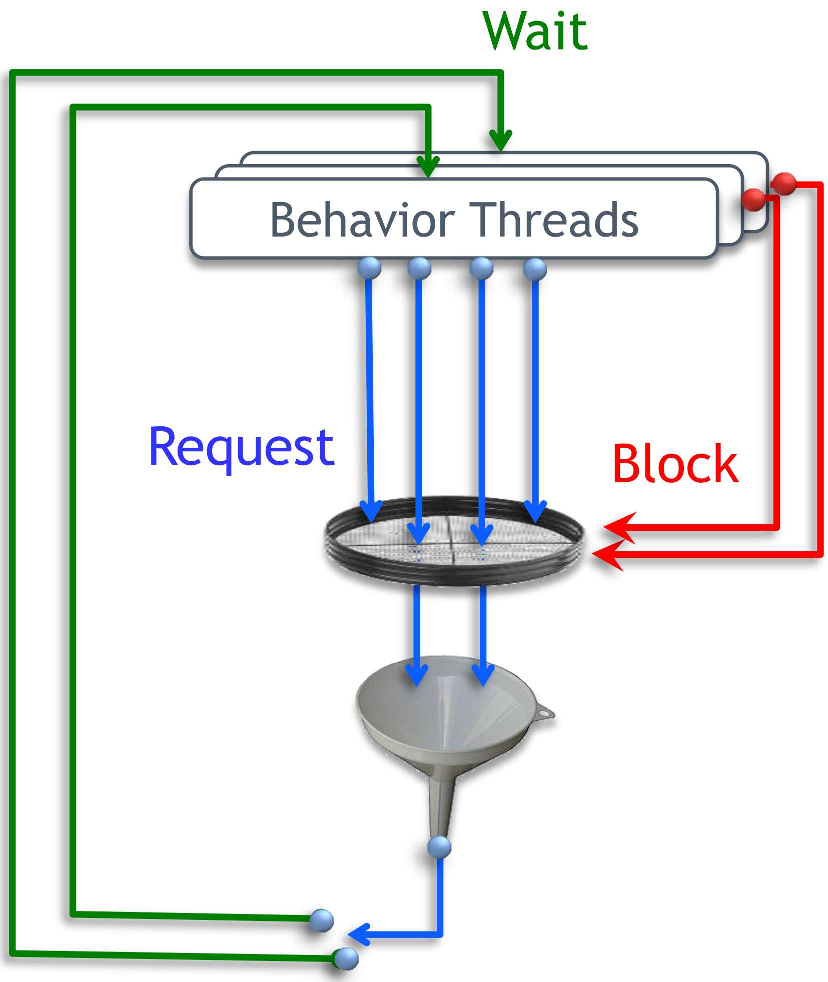
    - Request events: propose events to be considered for triggering;
    - Wait for events: ask to be notified when events are triggered;
    - Block events: temporarily forbid the triggering of events.

2. When all declarations are collected:

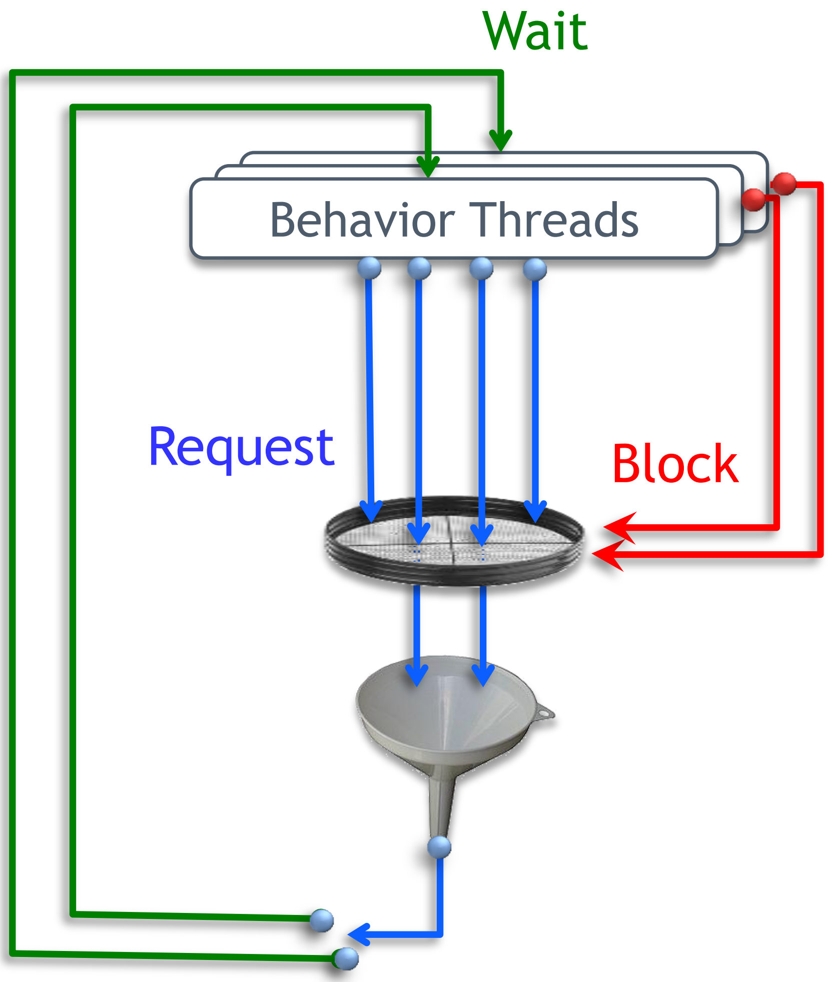
    An event that is **requested** and not **blocked** is selected.

    All b-threads **waiting** for this event can update their declaration

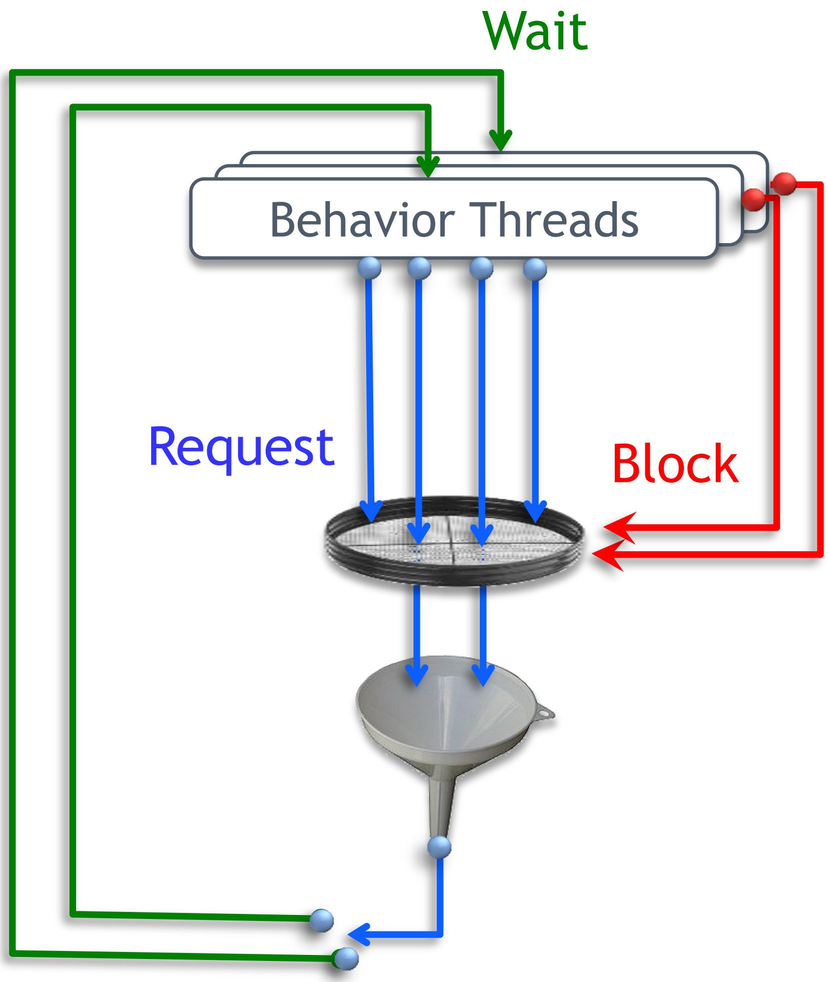
Behavior execution cycle



Behavior execution cycle



Behavior execution cycle



The BPJ Library and API

- B-threads are Java threads
- Events and event sets are Java objects and collections
- Development and execution do not require special environments
- Direct integration with other Java code:

```
class MyBThread extends BThread {
    void runBthread() {
        …
        bSync(requestedEvents, watchedEvents, blockedEvents);
        …
    }
}
```

- The transition system is implicit

Online: The Group's SVN

Example: Coding b-threads in Java

```
class AddHotFiveTimes extends BThread {
    public void runBThread() {
        for (int i=1; i<=5; i++) {
            bSync(addHot, none, none);
        }
    }
}
```

```
class AddColdFiveTimes BThread {
    public void runBThread() {
        for (int i=1; i<=5; i++) {
            bSync(addCold, none, none);
        }
    }
}
```

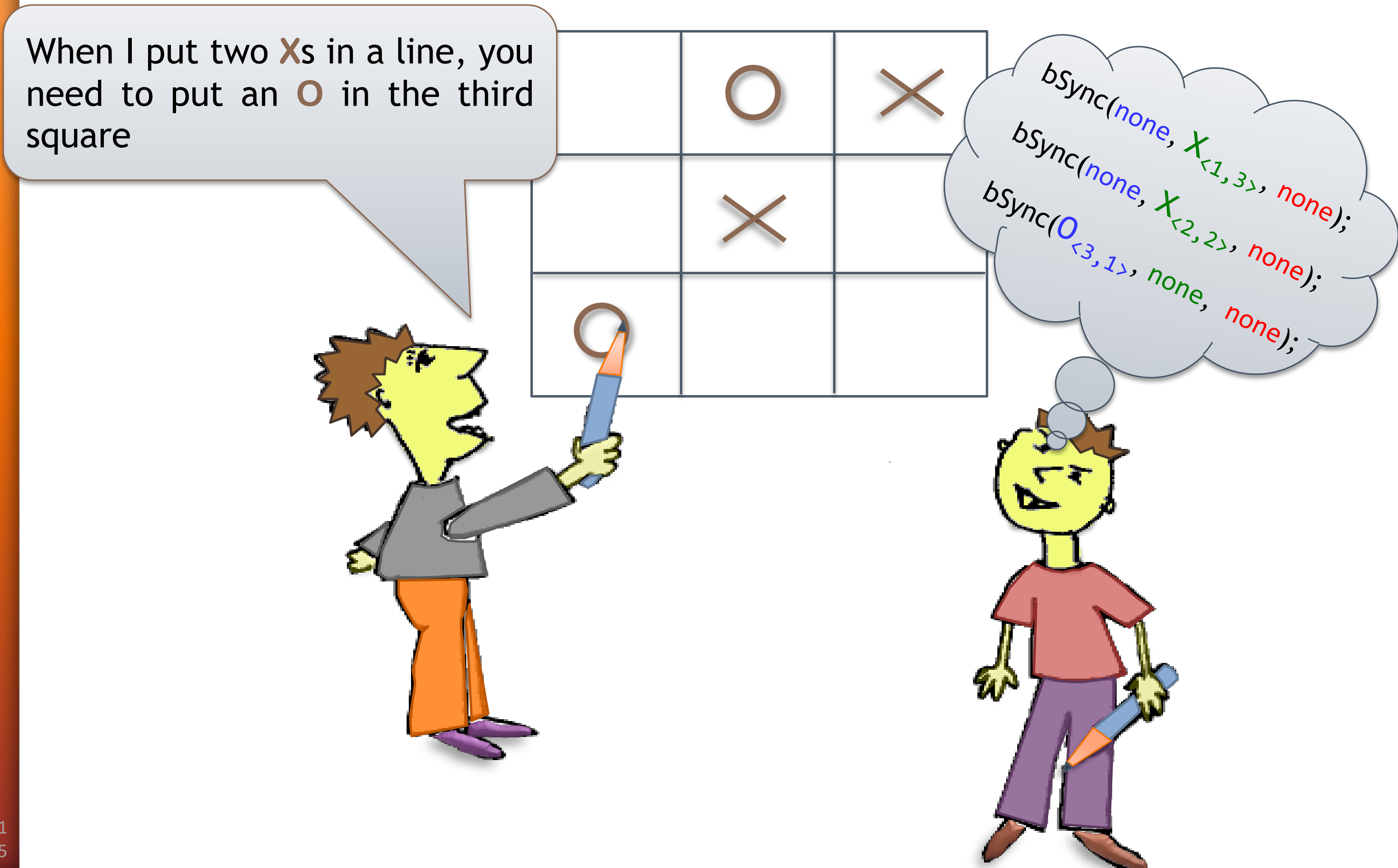
```
class Interleave extends BThread {
    public void runBThread() {
        while (true) {
            bSync(none, addHot, addCold);
            bSync(none, addCold, addHot);
        }
    }
}
```

```
addHot
  addCold
addHot
  addCold
addHot
  addCold
addHot
  addCold
addHot
  addCold
```

Main application: reactive systems

Complexity stems from the need to interleave many simultaneous behaviors

Alignment of code modules with requirements


When I put two Xs in a line, you need to put an O in the third square

```
bSync(none, X<1,3>, none);
bSync(none, X<2,2>, none);
bSync(O<3,1>, none, none);
```

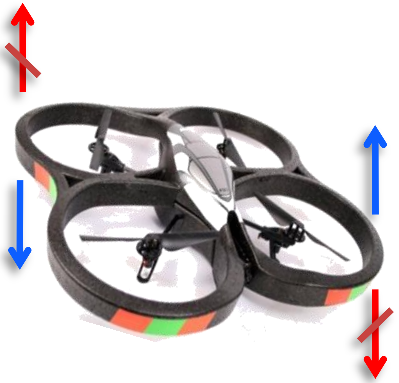
Each new game rule or strategy is added in a

separate b-thread

without changing existing code

Example: Flying a quadrotor helicopter

To correct the angle:



block SpeedUpR4

request SlowDownR4

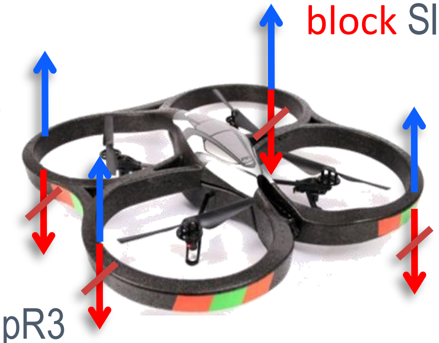
request SpeedUpR2

block SlowDownR2

To increase altitude:

request SpeedUpR1

block SlowDownR1



request SpeedUpR4

block SlowDownR4

request SpeedUpR2

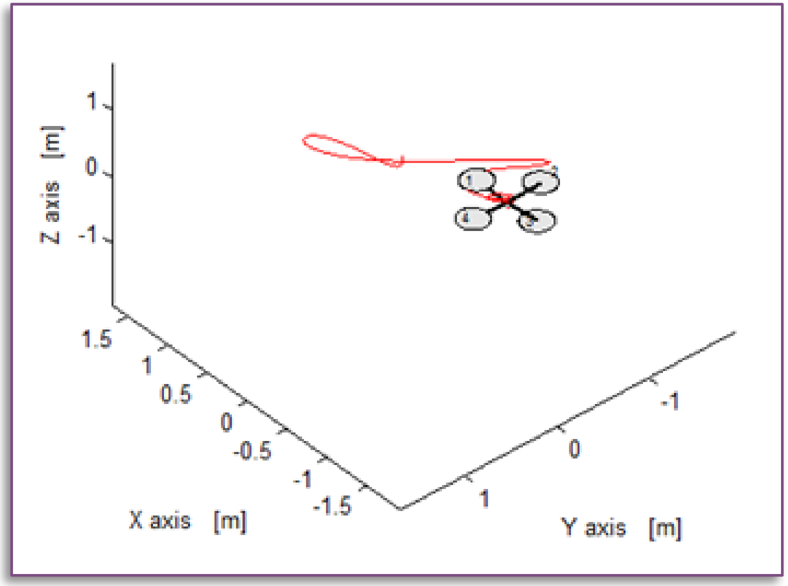
block SlowDownR2

request SpeedUpR3

block SlowDownR3

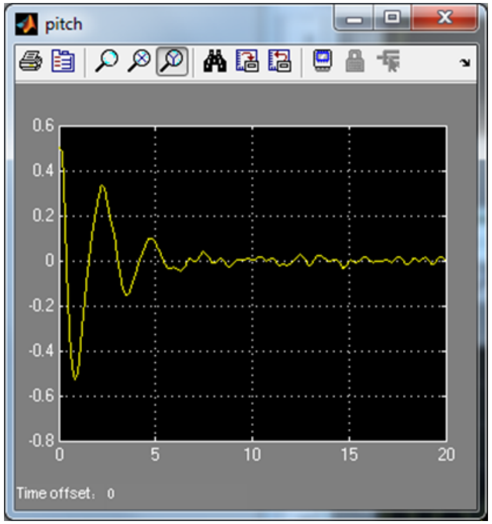
Selected event:
SpeedUpR2

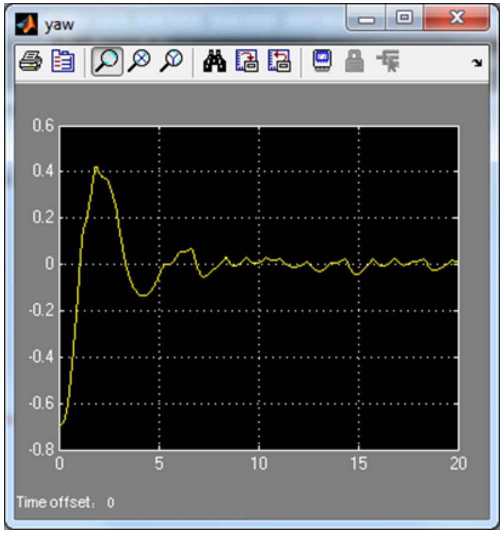
Balancing a quadrotor – behaviorally

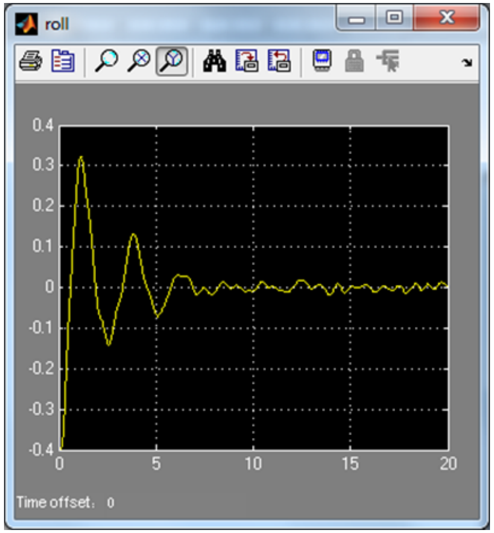


Results of applying the quadrotor Simulink model of Bouabdalla et al where a linear transformation box is replaced with behavior threads.

Pitch, yaw and roll angles are stabilized after a few seconds.



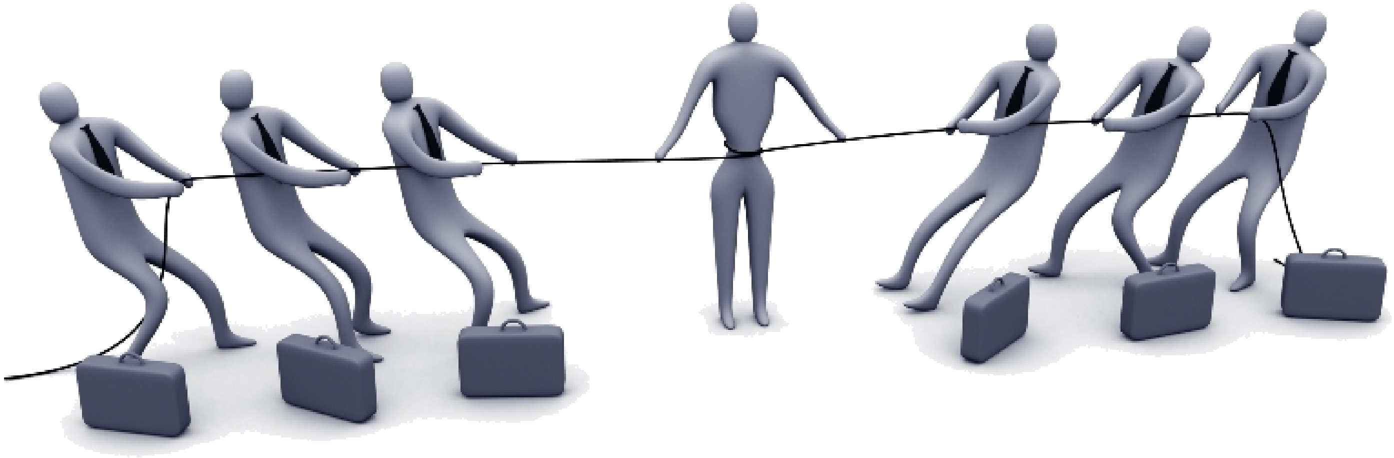




But…

- How do we know when we are done?

- When each module is programmed separately, how do we avoid conflicts?



- An answer: **Model Checking + Incremental Development**

b-Thread: Formal Definition

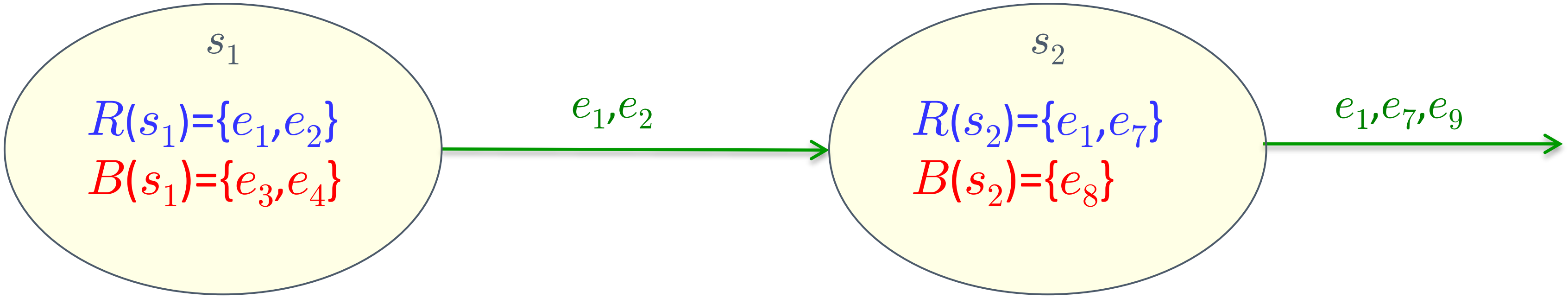
A **b-thread** is a tuple $\langle S,\ E,\ \rightarrow,\ init_i,\ R,\ B\ \rangle$

> Where $\langle S,\ E,\ \rightarrow, init\rangle$ is a transition system, and

> for each state s:

+ the set $R(s)$ models the **requested** events

+ the set $B(s)$ models the **blocked** events



The runs of a set of b-threads

Composition of the b-threads $\{\langle S_i, E_i, \rightarrow_i, init_i, R_i, B_i \rangle : i{=}1,\ldots,n\}$ is defined as a product transition system.

The composition contains the transition $\langle s_1, \ldots, s_n \rangle \xrightarrow{e} \langle s'_1, \ldots, s'_n \rangle$ if:

$$\underbrace{e \in \bigcup_{i=1}^{n} R_i(s_i)}_{e\ is\ requested} \qquad \bigwedge \qquad \underbrace{e \notin \bigcup_{i=1}^{n} B_i(s_i)}_{e\ is\ not\ blocked}$$

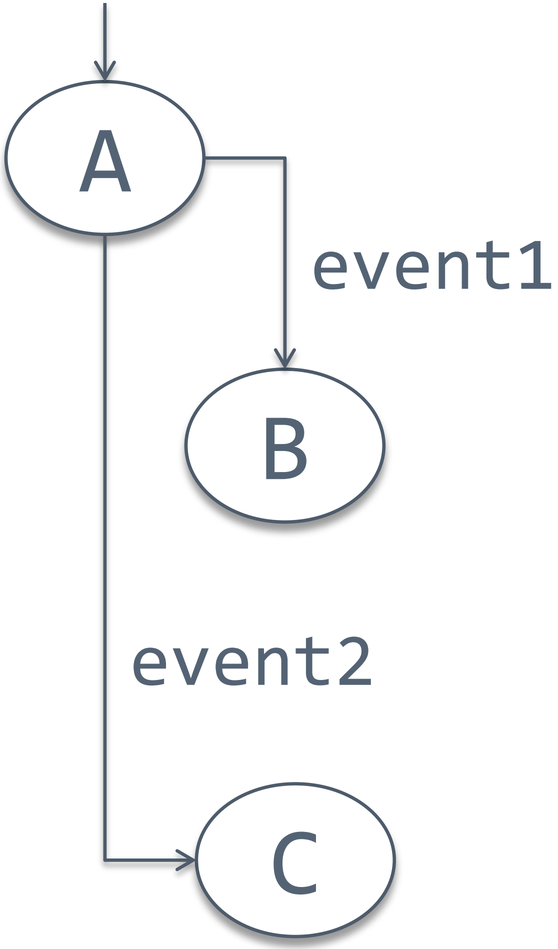
$$\bigwedge_{i=1}^{n} \Big(\underbrace{(e \in E_i \implies s_i \xrightarrow{e} s'_i)}_{affected\ b\text{-}threads\ move} \wedge \underbrace{(e \notin E_i \implies s_i = s'_i)}_{unaffected\ b\text{-}threads\ don't\ move} \Big)$$

Behavior Thread States

b-thread states at bSync

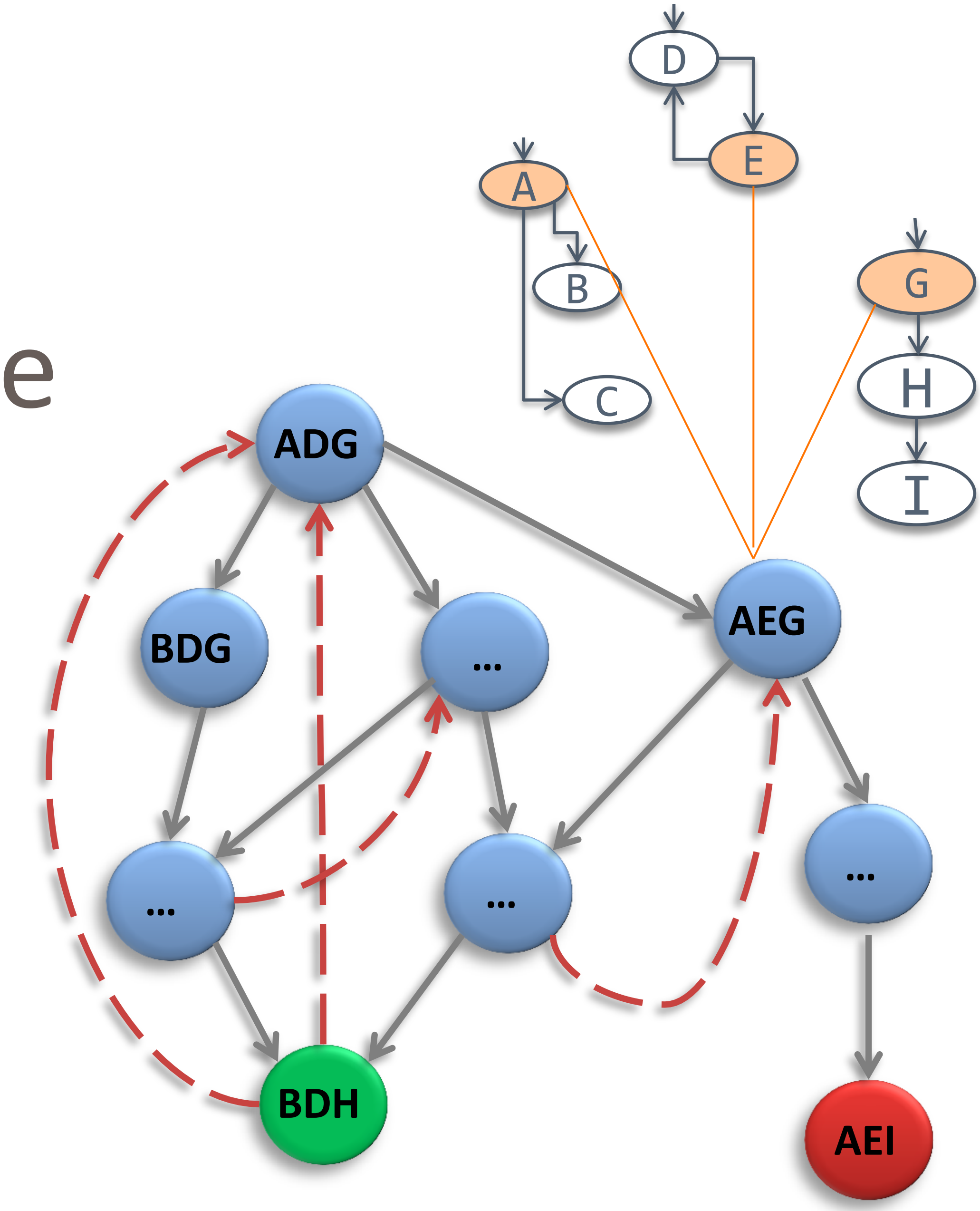
```
.
.
labelNextVerificationState( “A” );
bSync( … );
if( lastEvent == event1 ) {
        .
        .
        .
        labelNextVerificationState( “B” );
        bSync( … );
}

if( lastEvent == event2 ) {
        .
        .
        .
        labelNextVerificationState( “C” );
        bSync( … );
}
```

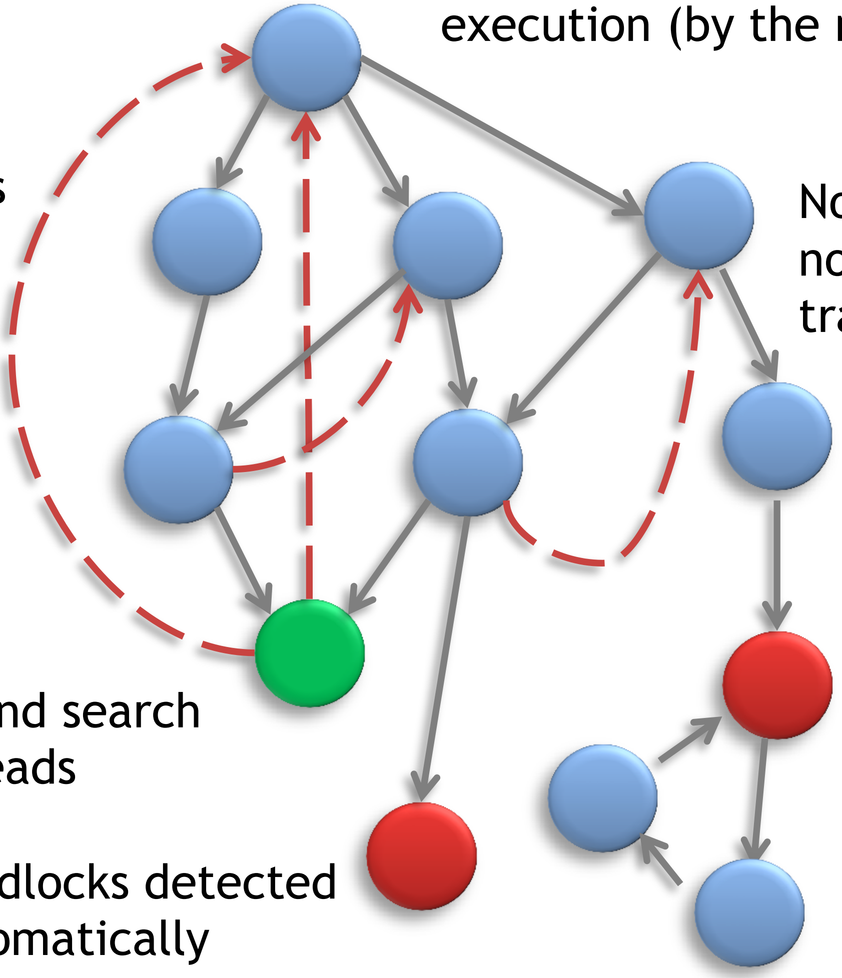


Behavioral Program State Graph

Program states are the Cartesian product of b-thread states



BPmc: Model-checking behavioral programs "in-vivo" (c.f. Java Path Finder)



Transition using standard execution (by the native JVM)

Backtrack using Apache `javaflow` continuations

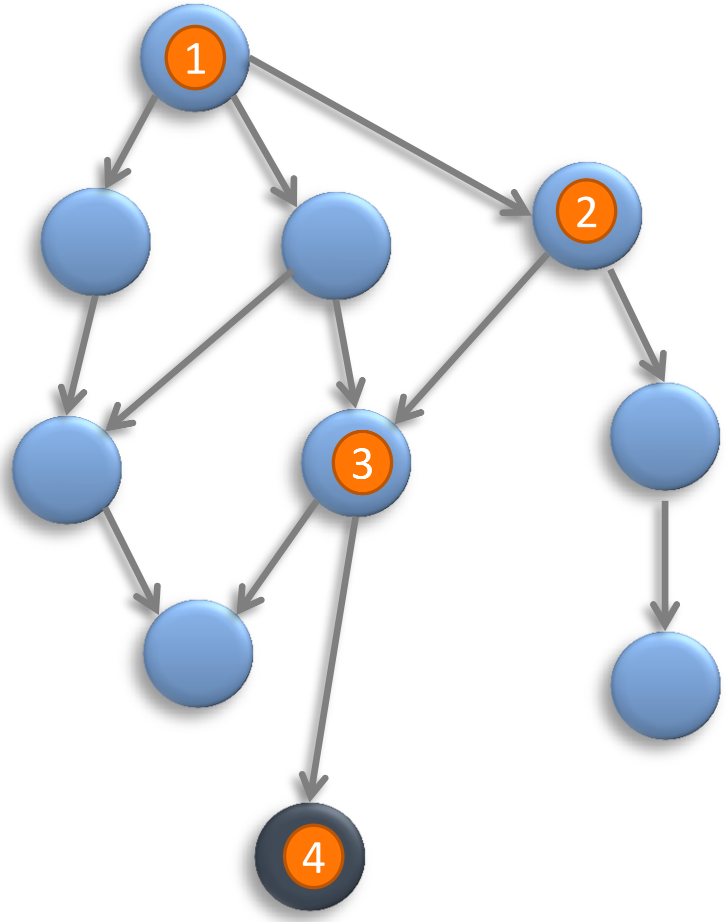
Notations for nondeterministic transitions

State tagging for safety and liveness properties by b-threads

State matching and search pruning by b-threads

Deadlocks detected automatically

Counterexample: A path to a bad state



Model-checker-assisted development of Tic-Tac-Toe

- Initial Development:
  - `DetectXWin, DetectOWin, DetectDraw`
  - `EnforceTurns`
  - `DefaultMoves`
  - `XAllMoves`

- Modify b-threads to prune search / mark bad states

- Model Check → Counterexample → Add b-thread / change priority:
  - `PreventThirdX`
  - `PreventXFork`
  - `PreventAnotherXFork`
  - `AddThirdO`
  - `PreventYetAnotherXFork`

X		O
O	O	
X	X	X

Counterexamples as scenarios

- Let $c=e_1, \dots, e_m, \dots, e_n$ be a counterexample
- Can generalize and code new b-threads or,
- Using counterexample in a patch behavior. E.g.,
  - Let e_m be the last event requested by the system
    - Wait for $e_1, \dots, e_{m-1}$
    - Block e_m
  - Other b-threads will take care of the right action, “the detour”.
  - Model-check again

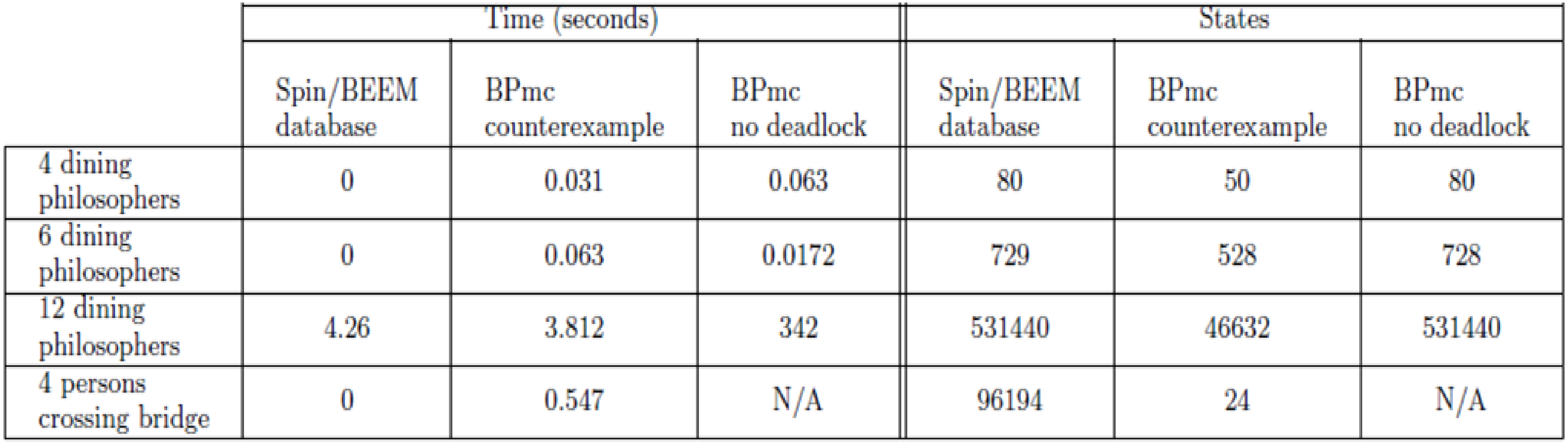
Other examples and experiences

» Bridge-crossing problem

» Dining Philosophers

» Scheduling in a signal-processing board

Initial Model-Checking Performance

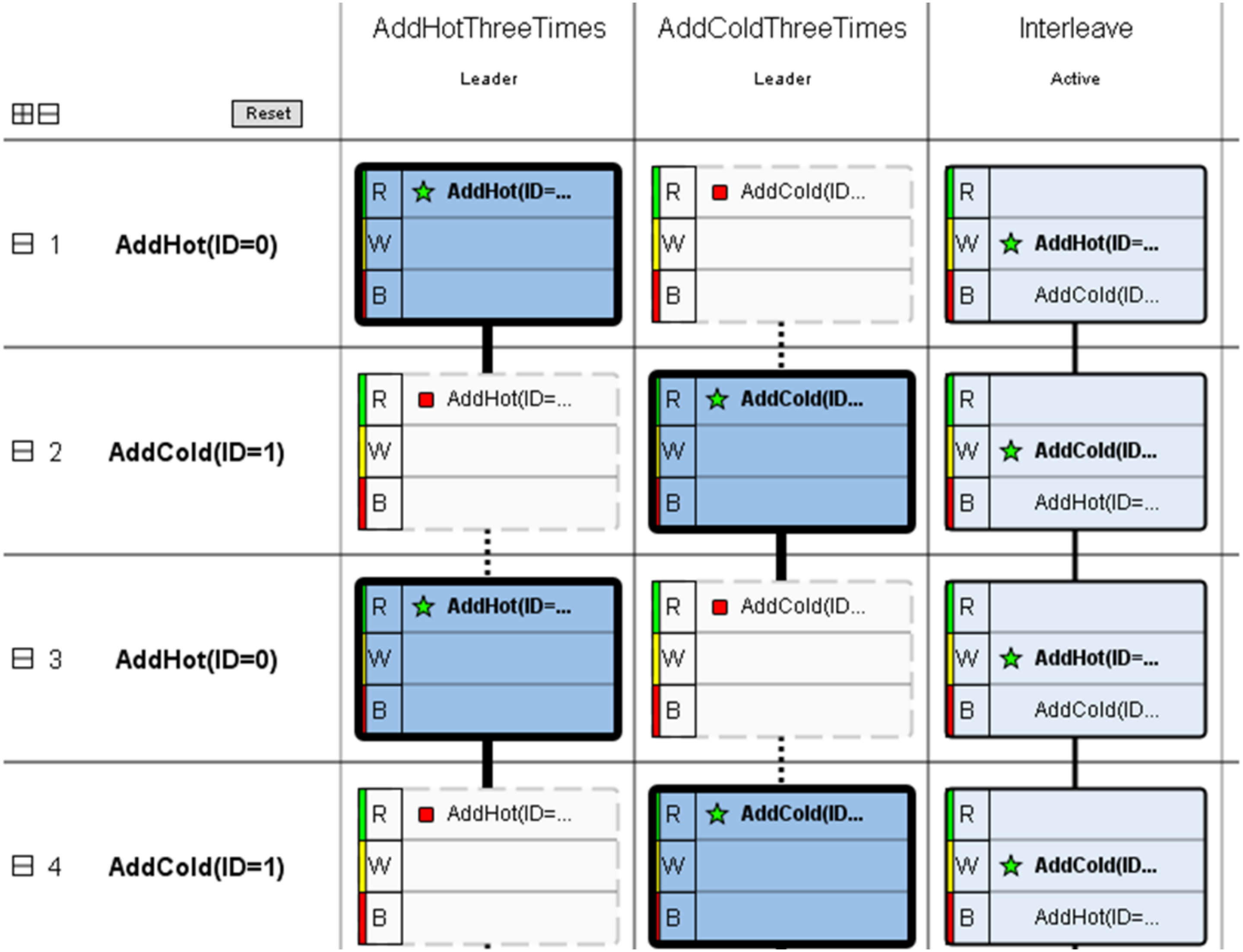


	Time (seconds)			States		
	Spin/BEEM database	BPmc counterexample	BPmc no deadlock	Spin/BEEM database	BPmc counterexample	BPmc no deadlock
4 dining philosophers	0	0.031	0.063	80	50	80
6 dining philosophers	0	0.063	0.0172	729	528	728
12 dining philosophers	4.26	3.812	342	531440	46632	531440
4 persons crossing bridge	0	0.547	N/A	96194	24	N/A

Limitations / opportunities of BPmc

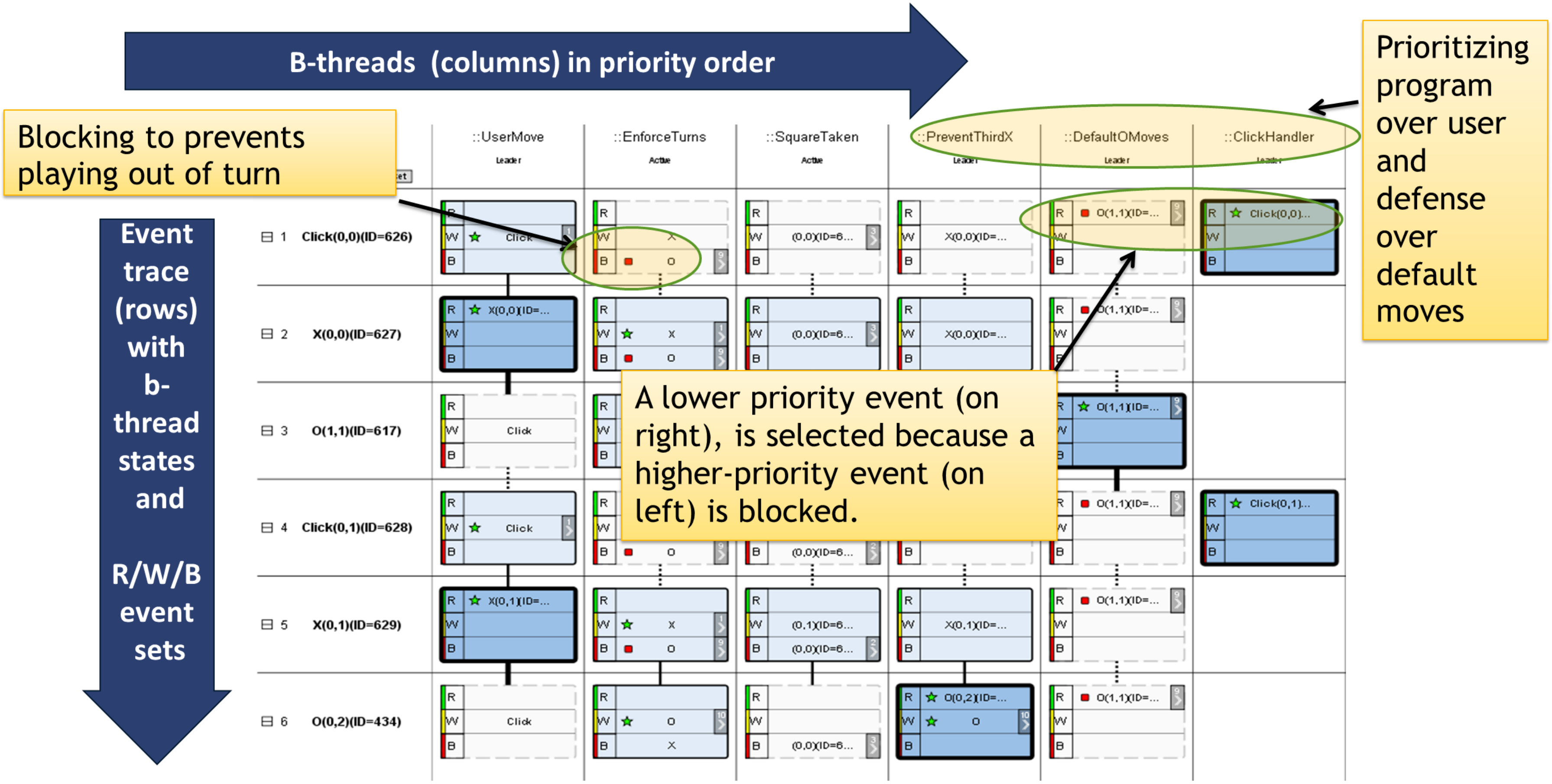
- » Abstracts program only per behavioral states
- » Dependent on application for state labeling
- » Single threading during model-checking
- » Dependency on Javaflow
- » No support for dynamic B-Threads
- » Application-dependent performance
- » Explicit MC only

Visualization and Comprehension



		AddHotThreeTimes Leader	AddColdThreeTimes Leader	Interleave Active
⊟ 1	**AddHot(ID=0)**	R ☆ **AddHot(ID=...** W B	R 🟥 AddCold(ID... W B	R W ☆ **AddHot(ID=...** B AddCold(ID...
⊟ 2	**AddCold(ID=1)**	R 🟥 AddHot(ID=... W B	R ☆ **AddCold(ID...** W B	R W ☆ **AddCold(ID...** B AddHot(ID=...
⊟ 3	**AddHot(ID=0)**	R ☆ **AddHot(ID=...** W B	R 🟥 AddCold(ID... W B	R W ☆ **AddHot(ID=...** B AddCold(ID...
⊟ 4	**AddCold(ID=1)**	R 🟥 AddHot(ID=... W B	R ☆ **AddCold(ID...** W B	R W ☆ **AddCold(ID...** B AddHot(ID=...

Visualization and Comprehension



B-threads (columns) in priority order

Event trace (rows) with b-thread states and

R/W/B event sets

Blocking to prevents playing out of turn

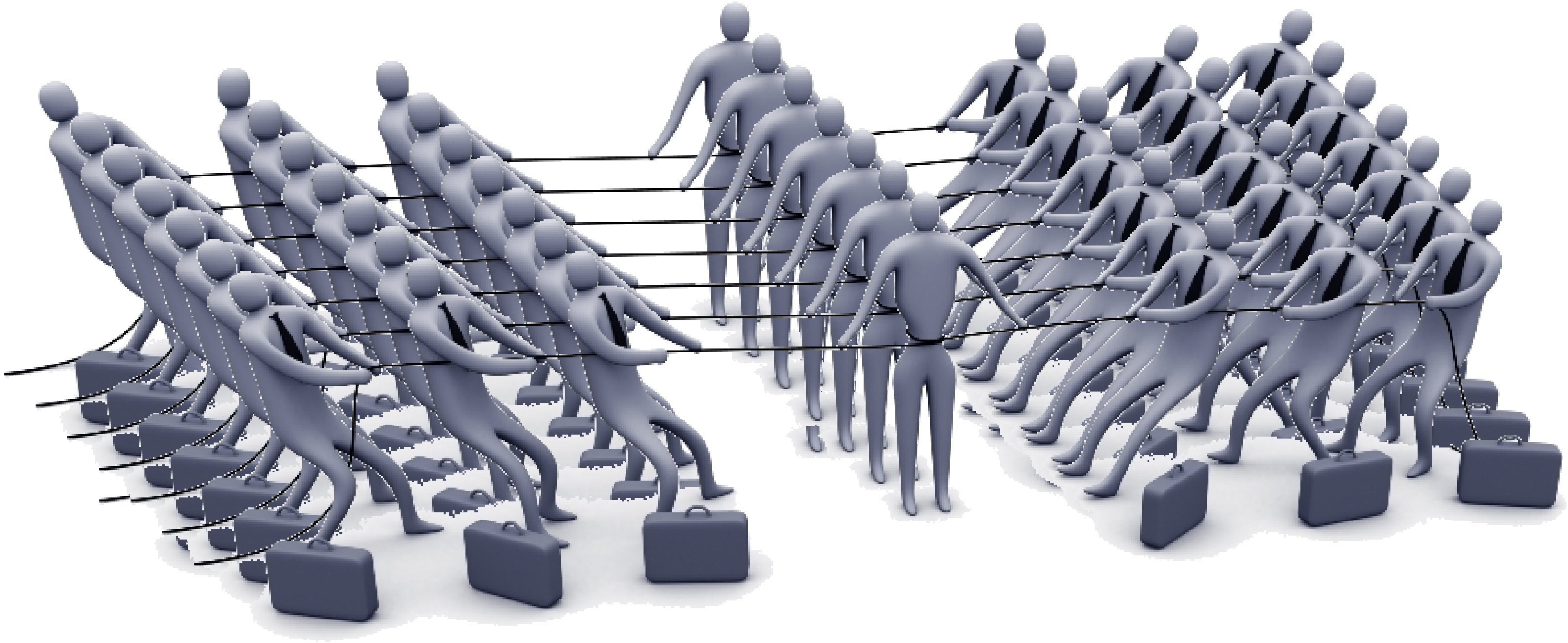
Prioritizing program over user and defense over default moves

A lower priority event (on right), is selected because a higher-priority event (on left) is blocked.

		::UserMove Leader	::EnforceTurns Active	::SquareTaken Active	::PreventThirdX Leader	::DefaultOMoves Leader	::ClickHandler Leader
⊟ 1	Click(0,0)(ID=626)	R W ☆ Click B	R W X B ■ O	R W (0,0)(ID=6... B	R W X(0,0)(ID=... B	R ■ O(1,1)(ID=... W B	R ☆ Click(0,0)... W B
⊟ 2	X(0,0)(ID=627)	R ☆ X(0,0)(ID=... W B	R W ☆ X B ■ O	R W (0,0)(ID=6... B	R W X(0,0)(ID=... B	R ■ O(1,1)(ID=... W B	
⊟ 3	O(1,1)(ID=617)	R W Click B	R W B			R ☆ O(1,1)(ID=... W B	
⊟ 4	Click(0,1)(ID=628)	R W ☆ Click B	R W B ■ O	B (0,0)(ID=6...	B	R ■ O(1,1)(ID=... W B	R ☆ Click(0,1)... W B
⊟ 5	X(0,1)(ID=629)	R ☆ X(0,1)(ID=... W B	R W ☆ X B ■ O	R W (0,1)(ID=6... B (0,0)(ID=6...	R W X(0,1)(ID=... B	R ■ O(1,1)(ID=... W B	
⊟ 6	O(0,2)(ID=434)	R W Click B	R W ☆ O B X	R W B (0,0)(ID=6...	R ☆ O(0,2)(ID=... W ☆ O B	R ■ O(1,1)(ID=... W B	

But still …

» Can it scale to large applications?

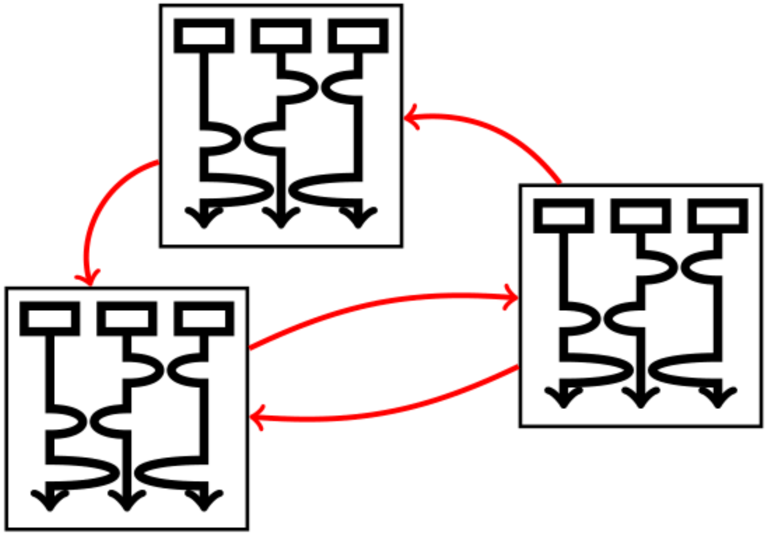


» … and what about external events?

Remote Events – Local Behavior

Real-life behavioral applications require distributed execution

- Asynchronous communication between nodes
- Synchronous collaboration inside nodes
- Each node has scenarios for handling remote events



Research Directions

> Compositional model-checking

> Run-time model-checking

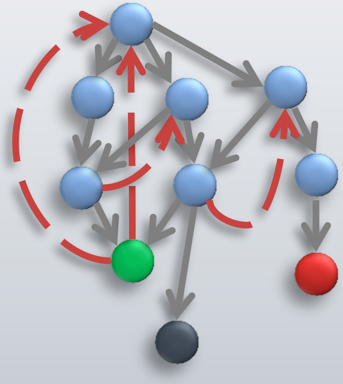
> Program synthesis and repair

> Intuitive programming platforms

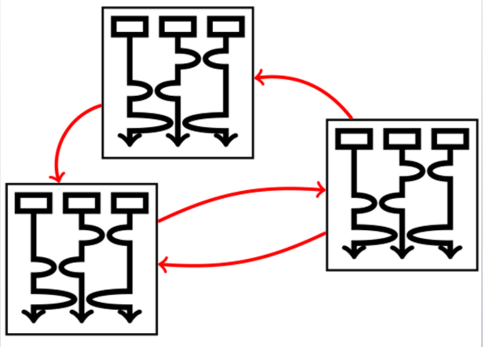
> Applications in robotics and hybrid control

> More

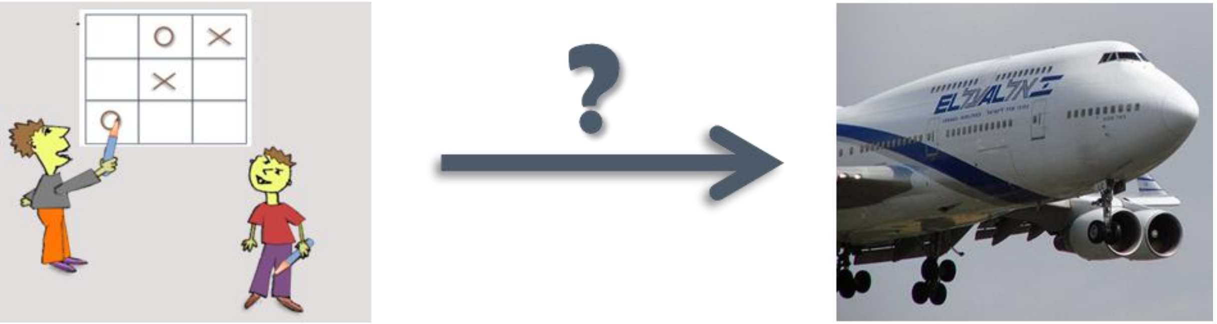
Interweaving



Verification



Scalability

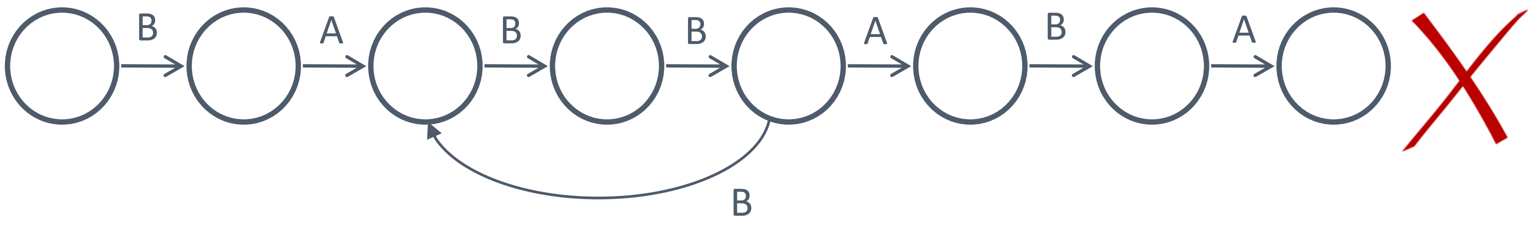


Thank You !

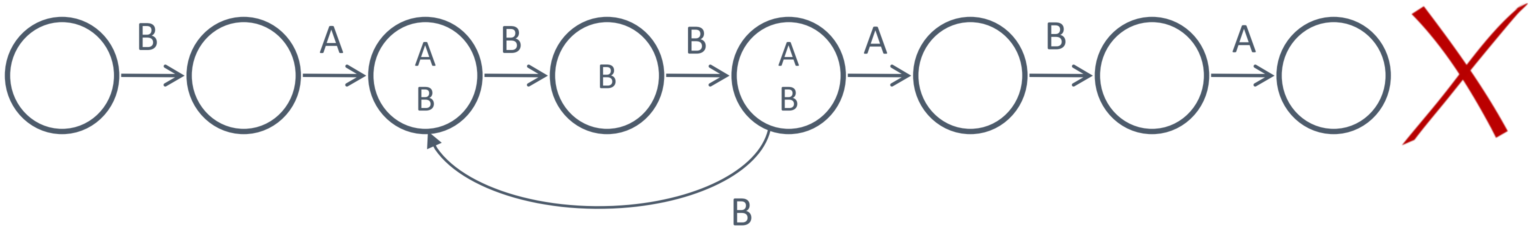
BACKUP SLIDES

BPmc support of fairness constraints (1 of 2)

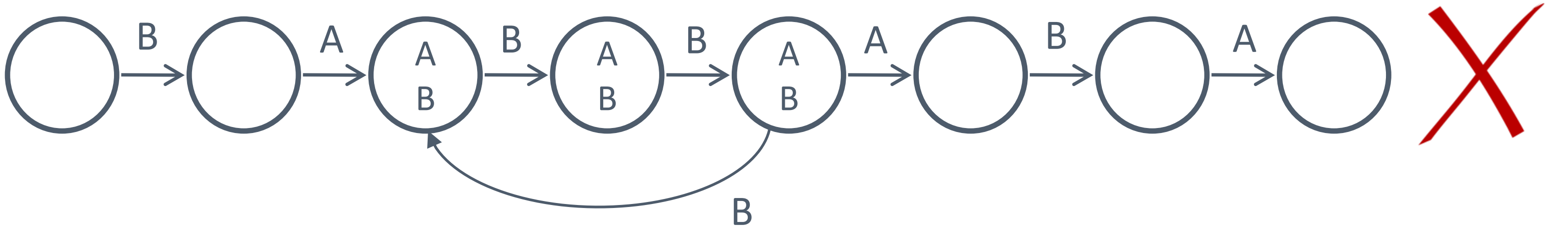
- Unconditional : “Every process gets its turn infinitely often”.



- Strong : “Every process that is enabled infinitely often gets its turn infinitely often”

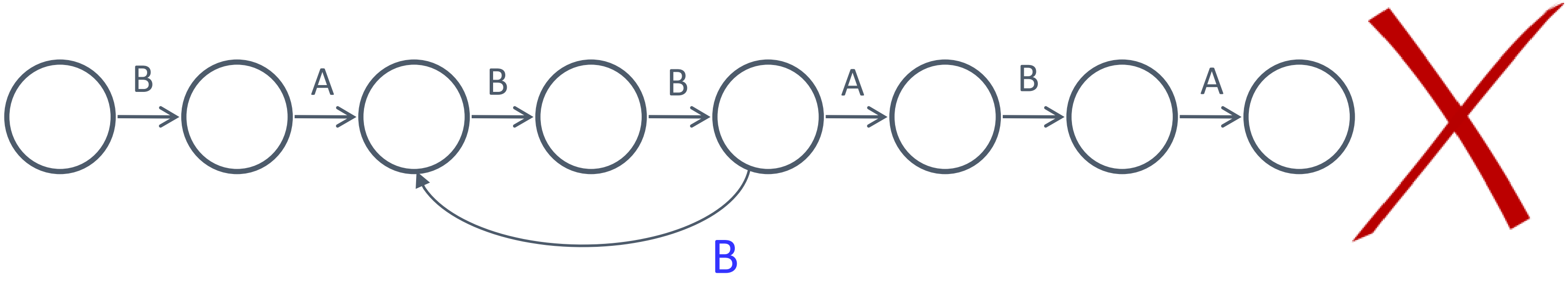


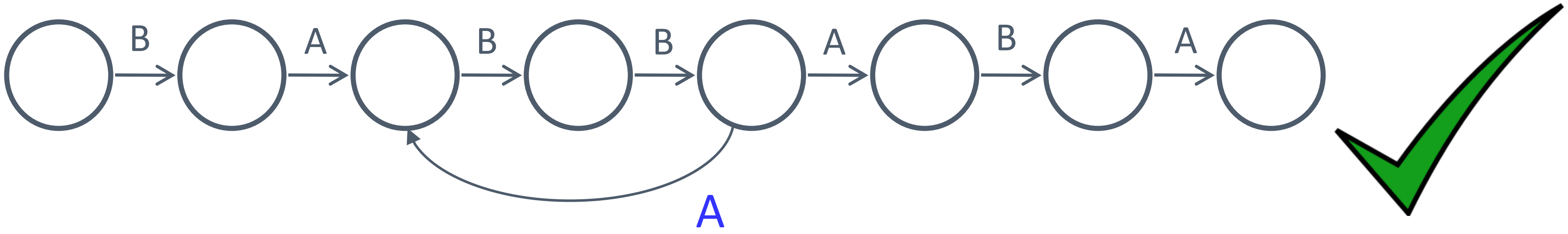
- Weak “Every process that is continuously enabled from a certain time instant on gets its turn infinitely often”



Liveness Testing with Fairness Constraints (2 of 2)

- » Input: fairness constraints as event sets
- » MC: Look for cold states only in *FAIR* cycles





Example of Behaviors in Tic-Tac-Toe

Move events: $X_{<0,0>}$, … , $X_{<2,2>}$, $O_{<0,0>}$, … , $O_{<2,2>}$

Game events: `OWin, XWin, Tie`

`EnforceTurns`: One player marks a square in a 3 by 3 grid with **X**, then the other marks a square with **O**, then it is **X**'s turn again, and so on;

`SquareTaken`: Once a square is marked, it cannot be marked again;

`DetectWin`: When a player marks three squares in a horizontal, vertical, or diagonal line, she wins;

`AddThirdO`: After marking two **Os** in a line, the **O** player should try to mark the third square (to win);

`PreventThirdX`: After the **X** player marks two squares in a line, the **O** player should try to mark the third square (to foil the attack);

`DefaultOMoves`: When other tactics are not applicable, player **O** should prefer the center square, then the corners, and mark an edge square only when there is no other choice;

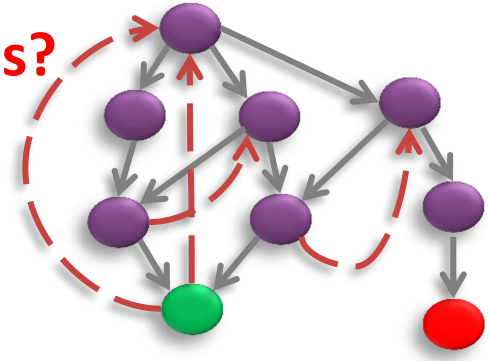
Javaflow

- http://commons.apache.org/sandbox/javaflow/
- Save a thread's stack in an object called a *continuation.*
- Can restore the continuation in any thread – and continue execution from there
- BPmc optionally serializes the continuation with all pointed objects
- See BP user guide

Some answers to common questions and challenges

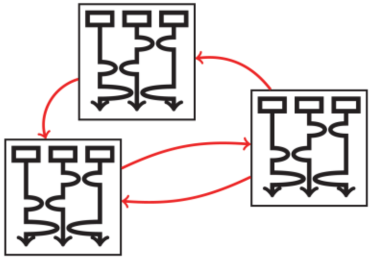
What about conflicting requirements?

- Model Checking
- Incremental development
- …



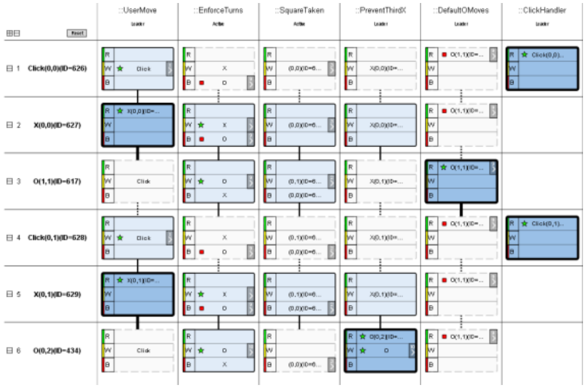
Scalability in terms number of behaviors and interleaving complexity?

- Agent oriented architectures
- Machine learning for event selection
- …



Comprehension of systems constructed by behavior composition?

- Trace visualization tool
- …



Reset	::UserMove Leader	::EnforceTurns Active	::SquareTaken Active	::PreventThirdX Leader	::DefaultOMoves Leader	::ClickHandler Leader
⊟ 1 Click(0,0)(ID=626)	R W ★ Click B	R W X B ■ O	R W (0,0)(ID=6... B	R W X(0,0)(ID=... B	R ■ O(1,1)(ID=... W B	R ★ Click(0,0)... W B
⊟ 2 X(0,0)(ID=627)	R ★ X(0,0)(ID=... W B	R W ★ X B ■ O	R W (0,0)(ID=6... B	R W X(0,0)(ID=... B	R ■ O(1,1)(ID=... W B	
⊟ 3 O(1,1)(ID=617)	R W Click B	R W ★ O B X	R W (0,1)(ID=6... B (0,0)(ID=6...	R W X(0,1)(ID=... B	R ★ O(1,1)(ID=... W B	
⊟ 4 Click(0,1)(ID=628)	R W ★ Click B	R W X B ■ O	R W (0,1)(ID=6... B (0,0)(ID=6...	R W X(0,1)(ID=... B	R ■ O(1,1)(ID=... W B	R ★ Click(0,1)... W B
⊟ 5 X(0,1)(ID=629)	R ★ X(0,1)(ID=... W B	R W ★ X B ■ O	R W (0,1)(ID=6... B (0,0)(ID=6...	R W X(0,1)(ID=... B	R ■ O(1,1)(ID=... W B	
⊟ 6 O(0,2)(ID=434)	R W Click B	R W ★ O B X	R W B (0,0)(ID=6...	R ★ O(0,2)(ID=... W ★ O B	R ■ O(1,1)(ID=... W B	