Domain-specific Model Checking with Bogor

SAnToS Laboratory, Kansas State University, USA

http://bogor.projects.cis.ksu.edu

Robby Matthew B. Dwyer John Hatcliff

Matthew Hoosier

Session IV: Bogor Customizations

Support

US Army Research Office (ARO)
US National Science Foundation (NSF)
US Department of Defense
Advanced Research Projects Agency (DARPA)

Boeing
Honeywell Technology Center
IBM
Intel

Lockheed Martin
NASA Langley
Rockwell-Collins ATC
Sun Microsystems

Outline

Customizations

- Extending BIR
  - Adding a Set Extension
  - Demo

Cadena

- Abstract API for CORBA Real-time Event Channel
- Priority scheduling & relative time
- Quasi-cyclic search

BogorVM

- a Java Virtual Machine and model checker in Bogor
- Modeling JVM instruction set as extensions
- Eclipse front-end

Customization Mechanisms

Bogor -- Extensible Modeling Language

Threads, Objects, Methods, Exceptions, etc.

+

Domain-Specific Abstractions

Core Modeling Language

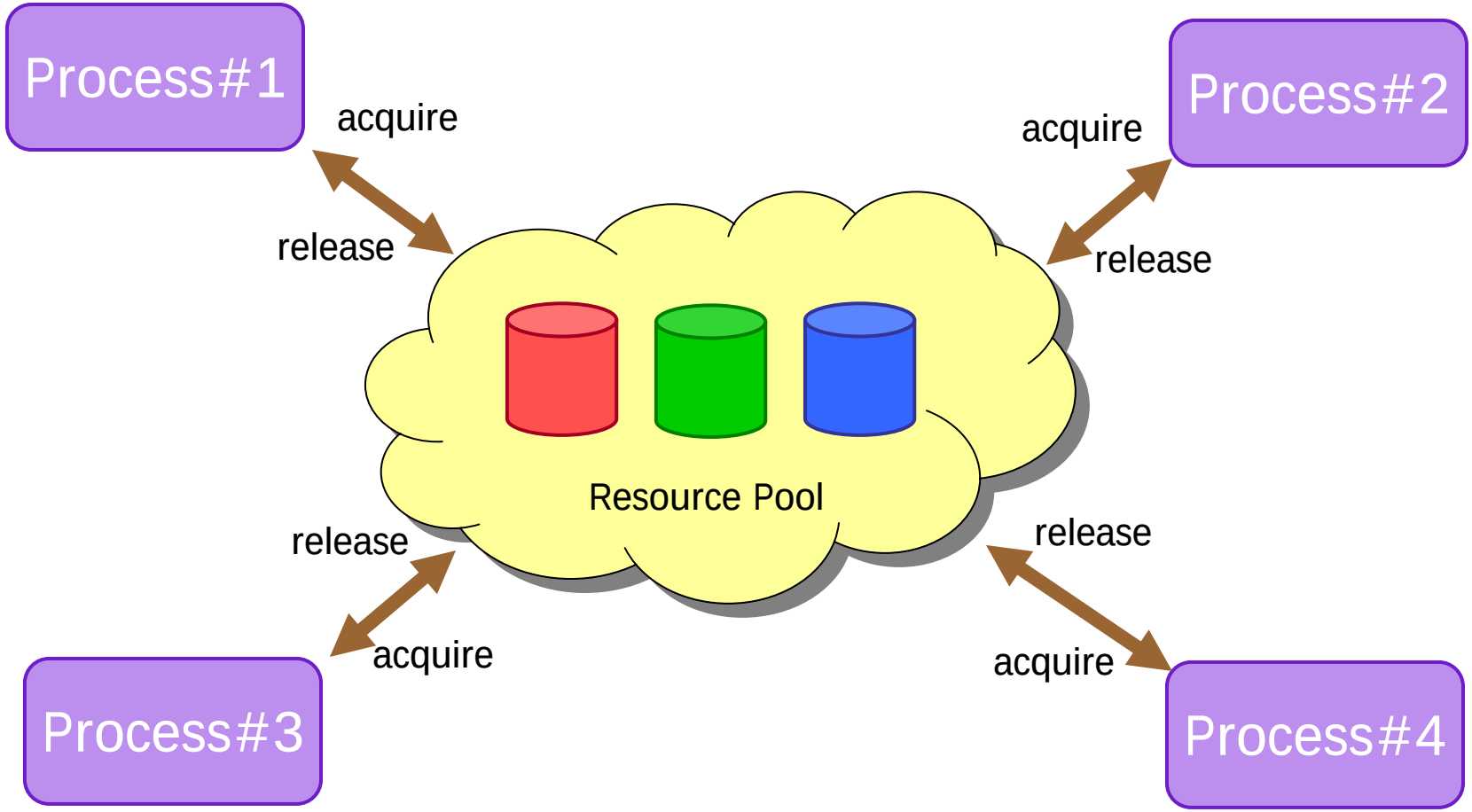
Bogor -- Customizable Checking Engine Modules

	Scheduling Strategy	*...existing modules...*	Domain-Specific Scheduler
State-space Exploration	State Representation	Domain-Specific Search	Domain-Specific State Rep.

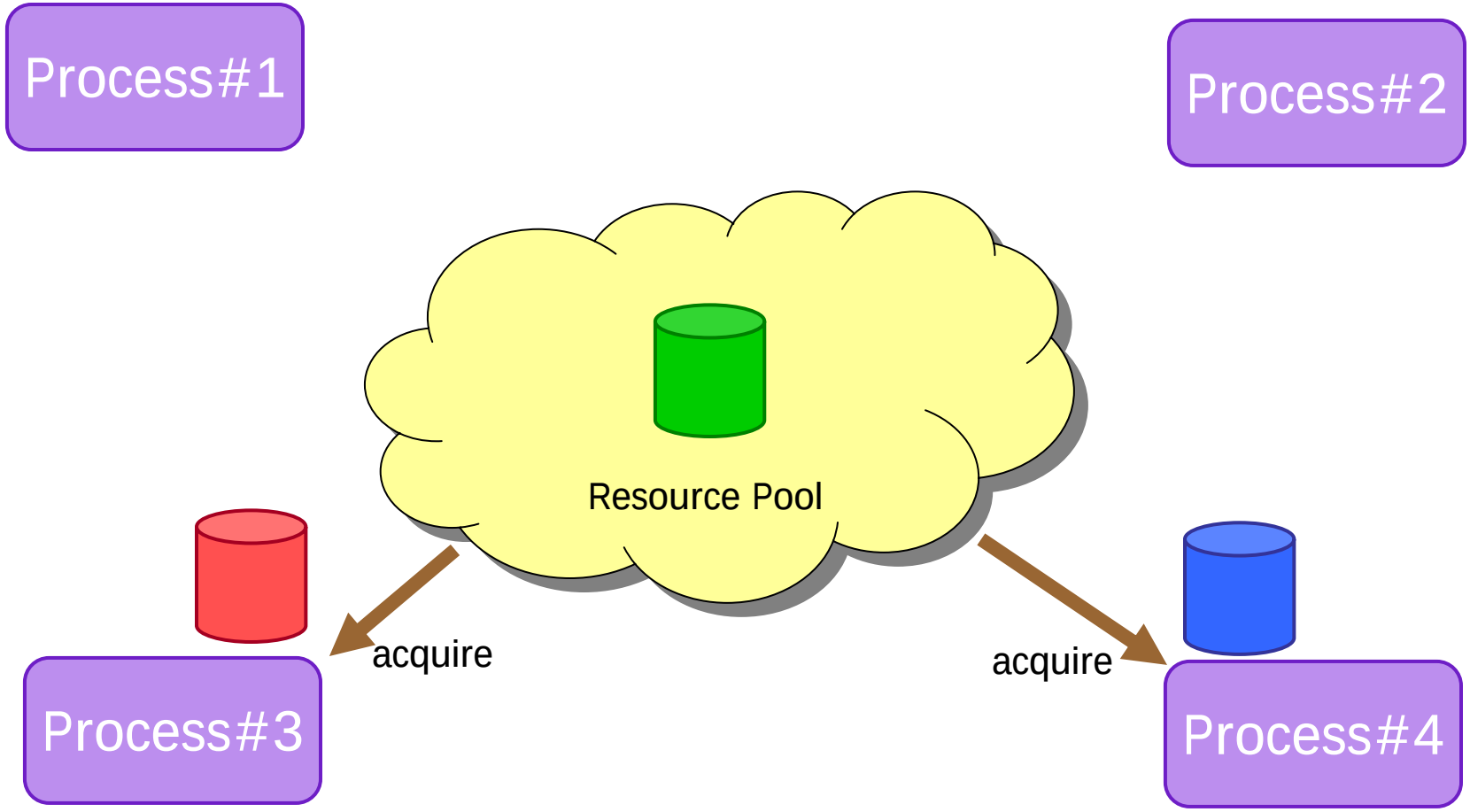
Core Checker Modules

Customized Checker Modules

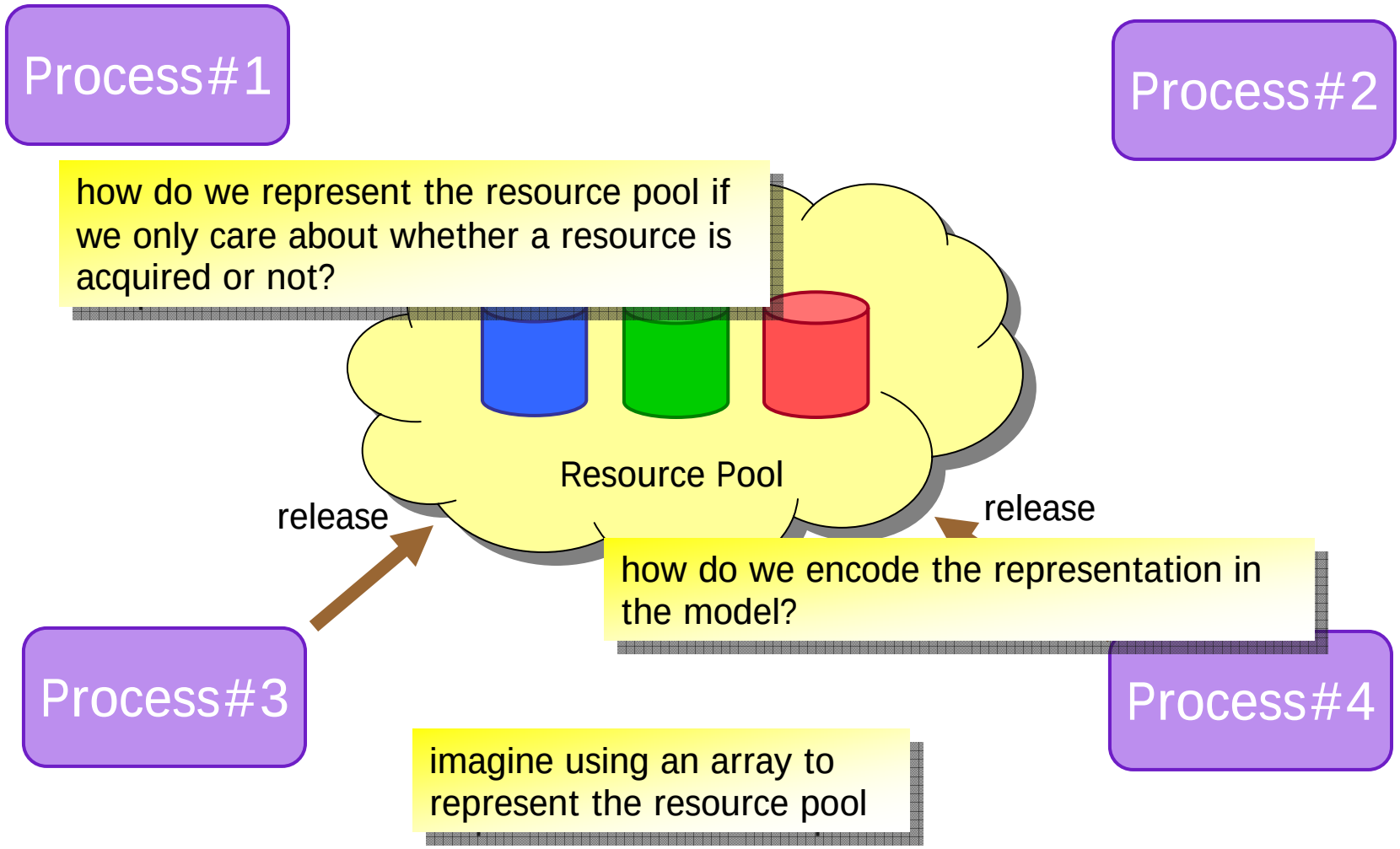
Extensible Input Language – Resource Contention Example



Extensible Input Language – Resource Contention Example

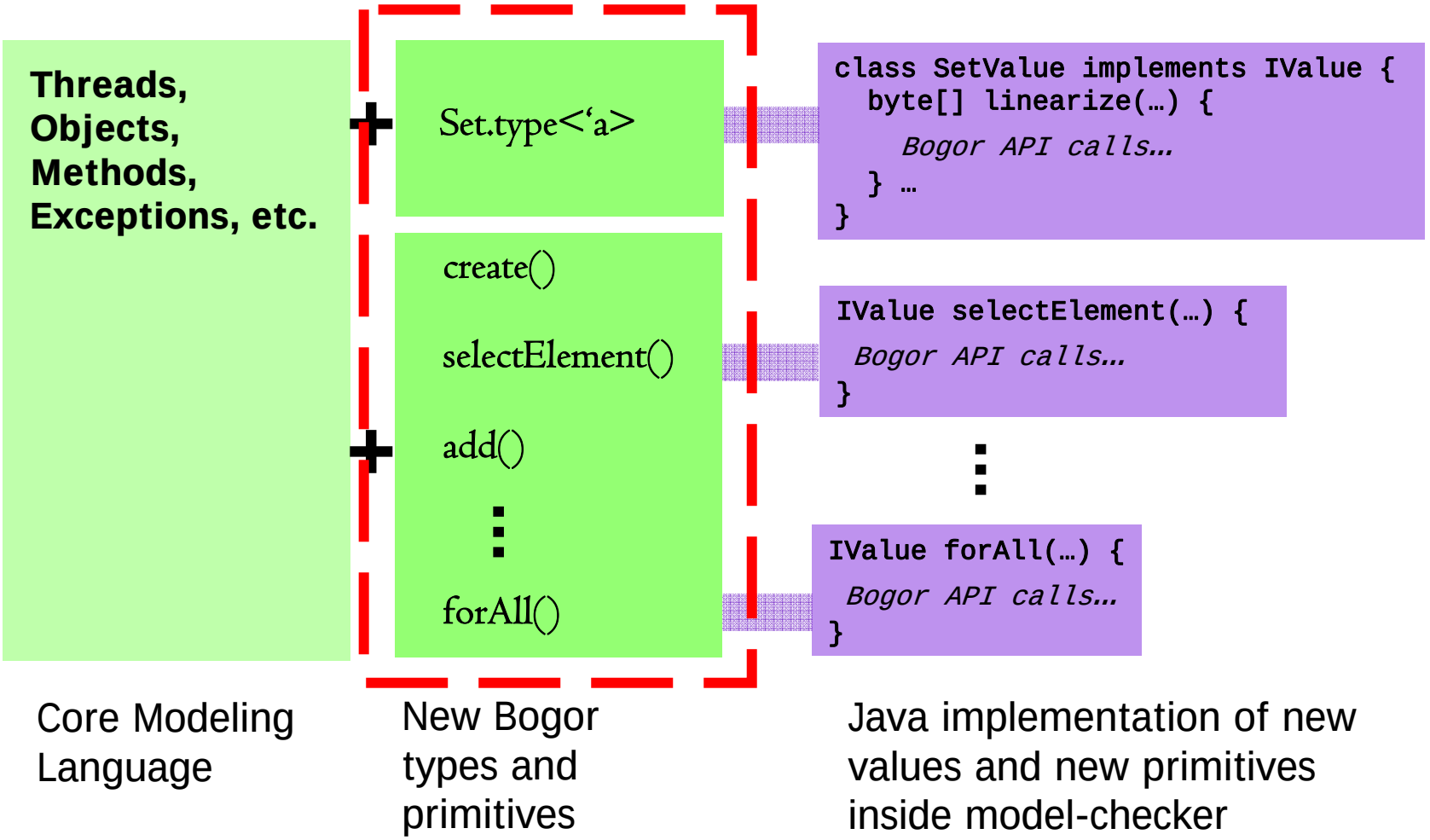


Extensible Input Language – Resource Contention Example



Domain-Specific Modeling

Bogor -- Extensible Modeling Language



Modeling Language Extensions

Bogor allows definitions of new abstract types and abstract operations as first-class constructs

A new *type* to represent polymorphic symmetric sets

```
extension Set for SetModule
{
  typedef type<'a>;

  expdef Set.type<'a> create<'a>('a ...);

  expdef 'a selectElement<'a>(Set.type<'a>);

  expdef boolean isEmpty<'a>(Set.type<'a>);

  actiondef add<'a>(Set.type<'a>, 'a);

  actiondef remove<'a>(Set.type<'a>, 'a);

  expdef boolean forAll<'a>('a -> boolean, Set.type<'a>);
}
```

Modeling Language Extensions

Bogor allows definitions of new abstract types and abstract operations as first-class constructs

```
extension Set for SetModule
{
  typedef type<'a>;

  expdef Set.type<'a> create<'a>('a ...);

  expdef 'a selectElement<'a>(Set.type<'a>);

  expdef boolean isEmpty<'a>(Set.type<'a>);

  actiondef add<'a>(Set.type<'a>, 'a);

  actiondef remove<'a>(Set.type<'a>, 'a);

  expdef boolean forAll<'a>('a -> boolean, Set.type<'a>);
}
```

Variable arity function for creating symmetric sets

Modeling Language Extensions

Bogor allows definitions of new abstract types and abstract operations as first-class constructs

```
extension Set for SetModule
{
  typedef type<'a>;

  expdef Set.type<'a> create<'a>('a ...);

  expdef 'a selectElement<'a>(Set.type<'a>);

  expdef boolean isEmpty<'a>(Set.type<'a>);

  actiondef add<'a>(Set.type<'a>, 'a);

  actiondef remove<'a>(Set.type<'a>, 'a);

  expdef boolean forAll<'a>('a -> boolean, Set.type<'a>);
}
```

Non-deterministically pick an element of the set to return

Modeling Language Extensions

Bogor allows definitions of new abstract types and abstract operations as first-class constructs

```
extension Set for SetModule
{
  typedef type<'a>;

  expdef Set.type<'a> create<'a>('a ...);

  expdef 'a selectElement<'a>(Set.type<'a>);

  expdef boolean isEmpty<'a>(Set

  actiondef remove<'a>(Set.type<'a>, 'a);

  expdef boolean forAll<'a>('a -> boolean, Set.type<'a>);
}
```

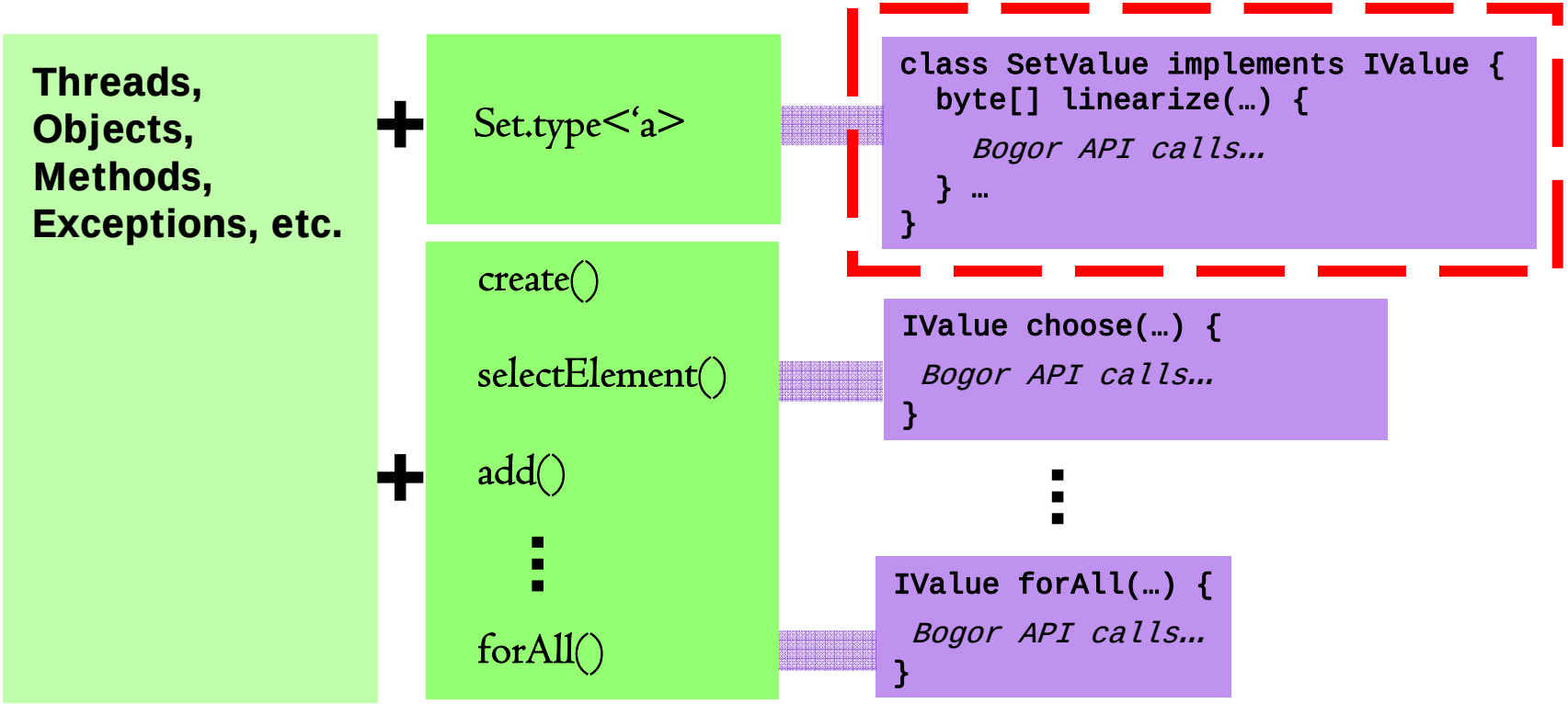
Higher-order function implements quantification over set elements

Predicate on set element

Set value to iterate over

Domain-Specific Modeling

Bogor -- Extensible Modeling Language



Core Modeling Language

New Bogor types and primitives

Java implementation of new values and new primitives inside model-checker

Extension Implementation

Implementing the set value for the set type

- operations on the value
  - *e.g.*, add, remove, isEmpty, *etc.*
- visitor pattern for the value
  - used for linearization, *etc.*
- linearization for state storage
  - fine-grained control over the value representation
- object model for counter-example display

Extension Implementation

Implementing the set value for the set type

```
public interface ISetValue
       extends INonPrimitiveExtValue {

  void add(IValue v);

  boolean contains(IValue v);

  IValue[] elements();

  boolean isEmpty();

  void remove(IValue v);

  ISetValue clone(Map<Object,Object> cloneMap);
}
```

Create an interface for all implementations of set values

Extension Implementation

Implementing the set value for the set type

```
public class DefaultSetValue implements ISetValue {
  protected HashSet<IValue> set = new HashSet<IValue>();
  …
  public void add(IValue v)

  public boolean contains(IV
    return set.contains(v);
  }

  public boolean isEmpty() { return set.size() == 0; }

  public void remove(IValue v) {set.remove(v); }

  public IValue[] elements() {
    IValue[] elements = set.toArray(new IValue[set.size()]);
    orderValues(elements);
    return result;
  }  …
```

A generic set value implementation capable of holding any element type

Extension Implementation

Implementing the set value for the set type

```
public class DefaultSetValue implements ISetValue {
  protected HashSet<IValue> set = new HashSet<IValue>();
  …
  public void add(IValue v) { se

  public boolean contains(IValue v) {
    return set.contains(v);
  }

  public boolean isEmpty() { return set.size() == 0; }

  public void remove(IValue v) {set.remove(v); }

  public IValue[] elements() {
    IValue[] elements = set.toArray(new IValue[set.size()]);
    orderValues(elements);
    return result;
  }  …
```

Defer to Java collection class

Extension Implementation

Implementing the set value for the

Most set operations are wrapper calls to **`HashSet`** methods

```
public class DefaultSetValue implements ISetV
  protected HashSet<IValue> set = new HashSet
  …
  public void add(IValue v) { set.add(v); }

  public boolean contains(IValue v) {
    return set.contains(v);
  }

  public boolean isEmpty() { return set.size() == 0; }

  public void remove(IValue v) {set.remove(v); }

  public IValue[] elements() {
    IValue[] elements = set.toArray(new IV
    orderValues(elements);
    return result;
  }  …
```

order elements for consistency

Extension Implementation

Implementing the set value for the set type

- operations on the value
  - *e.g.*, add, remove, isEmpty, *etc.*
- visitor pattern for the value
  - used for garbage collection, *etc.*
- linearization for state storage
  - fine-grained control over the value representation
- object model for counter-example display

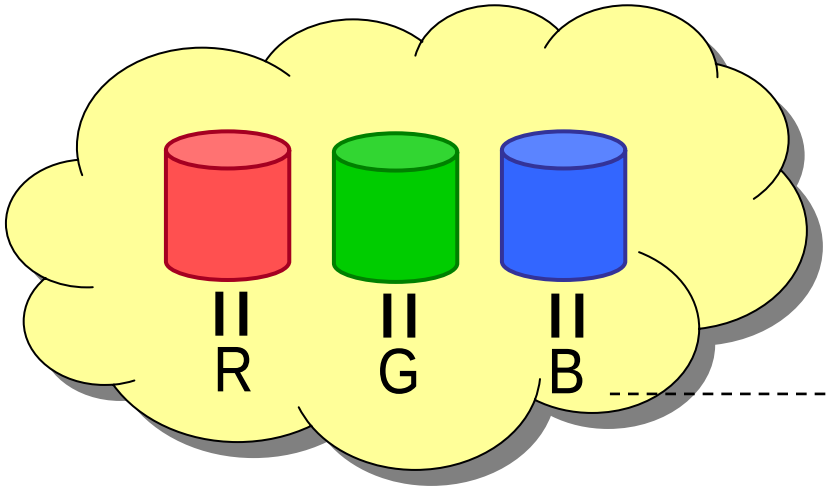
Extension Implementation

Implementing the linearization function

```
public class DefaultSetValue implements ISetValue {

  public byte[][] linearize(
    int bitsPerNonPrimitiveValue,
    ObjectIntTable<INonPrimitiveValue> nonPrimitiveValueIdMap,
    int bitsPerThreadId,
    IntIntTable threadOrderMap) {

    BitBuffer bb = new BitBuffer();

    vf.newVariedValueArray(elements())
      .linearize(
          false,
          bitsPerNonPrimitiveValue,
          nonPrimitiveValueIdMap,
          bitsPerThreadId,
          threadOrderMap,
          null, bb);

    return new byte[][] { bb.toByteArray() };
  }
  …
}
```

Convert the set value to a bit vector for state storage

Extensible Modeling Language – Set Extension Example (Semantics)



The state of the set consists of encodings of the references to resources

Suppose the resources are represented as integers R, G, B

<R,G,B>

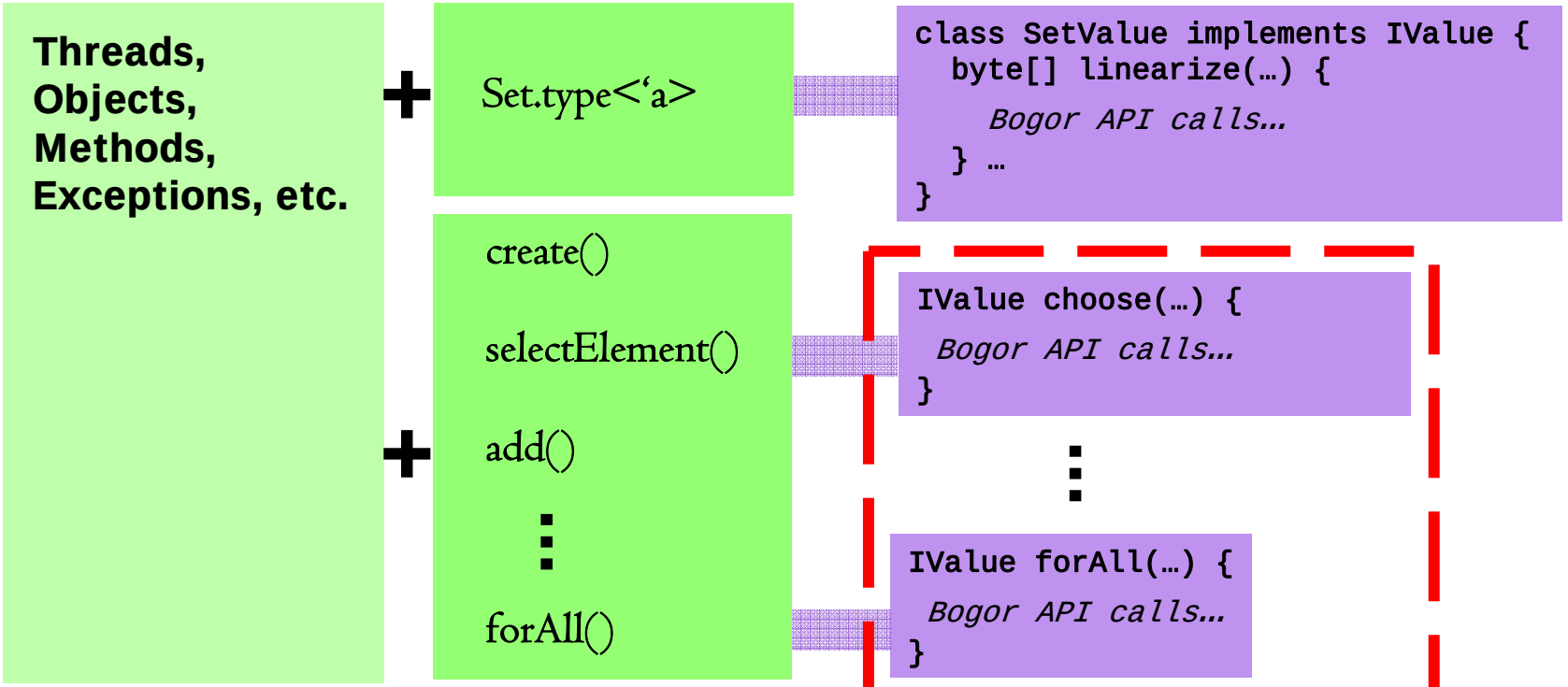
...convert to canonical order!

<B,G,R>

...now each equivalent representations of set can be detected by matching bit-pattern figure prints

Domain-Specific Modeling

Bogor -- Extensible Modeling Language



Extension Implementation

Implementing the set operations for the set type

- set module extension
- set creation
- non-deterministically choose a set element
- adding an element

Extension Implementation

Implementing the set operations for the set type

```
public class SetModule implements IModule {

  protected TypeFactory tf;
  protected IExpEvaluator ee;
  protected IValueFactory vf;
  protected IBacktrackingInfoFactory bf;
  protected ISchedulingStrategist ss;

  public String getCopyrightNotice() { return null; }

  public IMessageStore setOptions(String key, Properties options)
    { return new DefaultMessageStore(); }

  public IMessageStore connect(IBogorConfiguration bc) {
    tf = bc.getSymbolTable().getTypeFactory();
    ee = bc.getExpEvaluator();
    ss = bc.getSchedulingStrategist();
    vf = bc.getValueFactory();
    bf = bc.getBacktrackingInfoFactory();
    return new DefaultMessageStore();
  }
…
```

The set module must implement IModule

Extension Implementation

Implementing the set operations for the set type

```
public class SetModule implements IModule {

  protected TypeFactory tf;
  protected IExpEvaluator ee;
  protected IValueFactory vf;
  protected IBacktrackingInfoFactory bf;
  protected ISchedulingStrategist ss;

  public String getCopyrightNotice() { return null; }

  public IMessageStore setOptions(String key, Properties options)
    { return new DefaultMessageStore(); }

  public IMessageStore connect(IBogorConfiguration bc) {
    tf = bc.getSymbolTable().getTypeFactory();
    ee = bc.getExpEvaluator();
    ss = bc.getSchedulingStrategist();
    vf = bc.getValueFactory();
    bf = bc.getBacktrackingInfoFactory();
    return new DefaultMessageStore();
  }
…
```

Used for displaying copyright notices

Extension Implementation

Implementing the set operations for the set type

```
public class SetModule implements IModule {

  protected TypeFactory tf;
  protected IExpEvaluator ee;
  protected IValueFactory vf;
  protected IBacktrackingInfoFactory bf;
  protected ISchedulingStrategist ss;

  public String getCopyrightNotice() { return null; }

  public IMessageStore setOptions(String key, Properties options)
    { return new DefaultMessageStore(); }

  public IMessageStore connect(IBogorConfiguration bc) {
    tf = bc.getSymbolTable().getTypeFactory();
    ee = bc.getExpEvaluator();
    ss = bc.getSchedulingStrategist();
    vf = bc.getValueFactory();
    bf = bc.getBacktrackingInfoFactory();
    return new DefaultMessageStore();
  }
…
```

Used for setting options of Bogor modules

Extension Implementation

Implementing the set operations for the set type

```
public class SetModule implements IModule {

  protected TypeFactory tf;
  protected IExpEvaluator ee;
  protected IValueFactory vf;
  protected IBacktrackingInfoFactory bf;
  protected ISchedulingStrategist ss;

  public String getCopyrightNotice() { return null; }

  public IMessageStore setOptions(String key, Properties options)
    { return new DefaultMessageStore(); }

  public IMessageStore connect(IBogorConfiguration bc) {
    tf = bc.getSymbolTable().getTypeFactory();
    ee = bc.getExpEvaluator();
    ss = bc.getSchedulingStrategist();
    vf = bc.getValueFactory();
    bf = bc.getBacktrackingInfoFactory();
    return new DefaultMessageStore();
  }
…
```

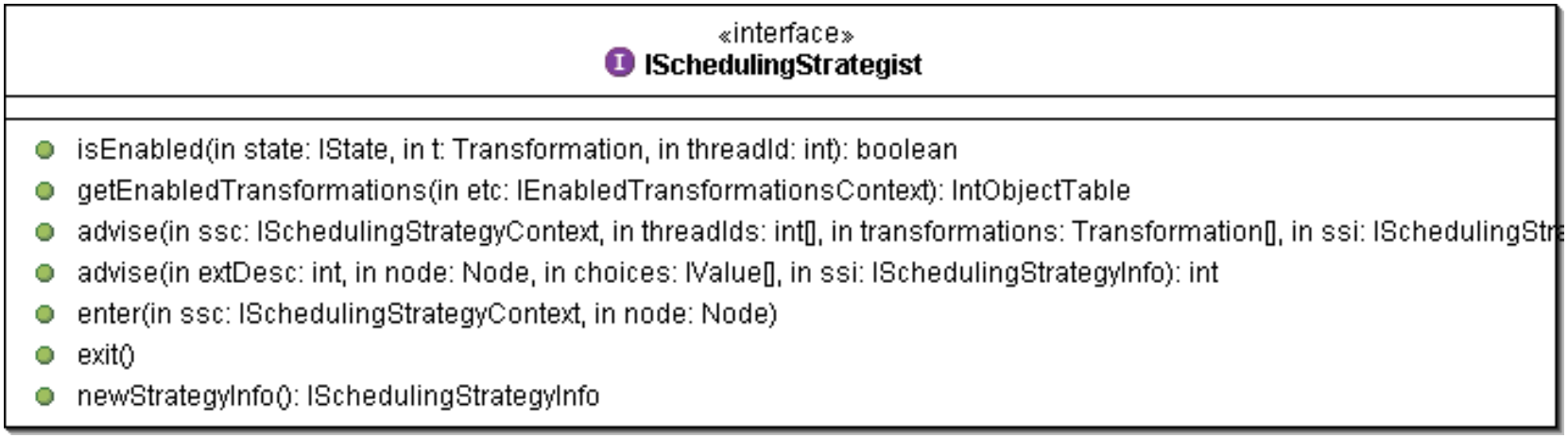
Connect to other Bogor modules

Extension Implementation

Implementing the set operations for the set type

- set module extension
- set creation
- non-deterministically choose a set element
- adding an element

ISchedulingStrategist



- Used to determine
  - enabled transitions: isEnabled(), getEnabledTransformations()
  - which transition to take: advise()
- create strategy info

Extension Implementation

Implementing the set operations for the set type

```
public class SetModule implements IModule {

  // expdef 'a choose<'a>(Set.type<'a>);
  public IValue choose(IExtArguments arg) {

    ISetValue set = (ISetValue) arg.getArgument(0);

    IValue[] elements = set.elements();

    int size = elements.length;

    int index = 0;

    if (size > 1) {
      index = ss.advise(arg.getExtDesc(),
                        arg.getNode(),
                        elements,
                        arg.getSchedulingStrategyInfo());
    }

    return elements[index];
  }
  …
```

Get the set value

Extension Implementation

Implementing the set operations for the set type

```
public class SetModule implements IModule {

  // expdef 'a choose<'a>(Set.type<'a>);
  public IValue choose(IExtArguments arg) {

    ISetValue set = (ISetValue) arg.getArgument(0);

    IValue[] elements = set.elements();

    int size = elements.length;

    int index = 0;

    if (size > 1) {
      index = ss.advise(arg.getExtDesc(),
                        arg.getNode(),
                        elements,
                        arg.getSchedulingStrategyInfo());
    }

    return elements[index];
  }
  …
```

Get the set elements

Extension Implementation

Implementing the set operations for the set type

```
public class SetModule implements IModule {

  // expdef 'a choose<'a>(Set.type<'a>);
  public IValue choose(IExtArguments arg) {

    ISetValue set = (ISetValue) arg.getArgument(0);

    IValue[] elements = set.elements();

    int size = elements.length;

    int index = 0;

    if (size > 1) {
      index = ss.advise(arg.getExtDesc(),
                        arg.getNode(),
                        elements,
                        arg.getSchedulingStrategyInfo());
    }

    return elements[index];
  }
  …
```

Ask the scheduler which one to pick if there are two or more elements

Extension Implementation

Implementing the set operations for the set type

- set module extension
- set creation
- non-deterministically choose a set element
- adding an element

Extension Implementation

Implementing the set operations for the set type

```
public class SetModule implements IModule {

  // actiondef add<'a>(Set.type<'a>, 'a);
  public IActionBacktrackingInfo add(IExtArguments arg) {
    ISetValue set = (ISetValue) arg.getArgument(0);

    IValue element = arg.getArgument(1);

    ISchedulingStrategyContext ssc =
           arg.getSchedulingStrategyContext();

    if (!set.contains(element)) {
      set.add(element);

      return new SetAdditionBacktrackingInfo(
        set, element, arg.getNode(), ssc.getStateId(),
        ssc.getThreadId(), arg.getSchedulingStrategyInfo());
    }
    else {
      return bf.createNoChangeBacktrackingInfo( … );
    }
  }
  …
```

Get the set and the element

Extension Implementation

Implementing the set operations for the set type

```
public class SetModule implements IModule {

  // actiondef add<'a>(Set.type<'a>, 'a);
  public IActionBacktrackingInfo add(IExtArguments arg) {
    ISetValue set = (ISetValue) arg.getArgument(0);

    IValue element = arg.getArgument(1);

    ISchedulingStrategyContext ssc =
           arg.getSchedulingStrategyContext();

    if (!set.contains(element)) {
      set.add(element);

      return new SetAdditionBacktrackingInfo(
        set, element, arg.getNode(), ssc.getStateId(),
        ssc.getThreadId(), arg.getSchedulingStrategyInfo());
    }
    else {
      return bf.createNoChangeBacktrackingInfo( … );
    }
  }
  …
```

Add the element if it is not already in the set value

Extension Implementation

Implementing the set operations for the set type

```
public class SetModule implements IModule {

  // actiondef add<'a>(Set.type<'a>, 'a);
  public IActionBacktrackingInfo add(IExtArguments arg) {
    ISetValue set = (ISetValue) arg.getArgument(0);

    IValue element = arg.getArgument(1);

    ISchedulingStrategyContext ssc =
           arg.getSchedulingStrategyContext();

    if (!set.contains(element)) {
      set.add(element);

      return new SetAdditionBacktrackingInfo(
        set, element, arg.getNode(), ssc.getStateId(),
        ssc.getThreadId(), arg.getSchedulingStrategyInfo());
    }
    else {
      return bf.createNoChangeBacktrackingInfo( … );
    }
  }
  …
```

Create the backtracking information

Extension Implementation

Implementing the set operations for the set type

```
public class SetModule implements IModule {

  // actiondef add<'a>(Set.type<'a>, 'a);
  public IActionBacktrackingInfo add(IExtArguments arg) {
    ISetValue set = (ISetValue) arg.getArgument(0);

    IValue element = arg.getArgument(1);

    ISchedulingStrategyContext ssc =
           arg.getSchedulingStrategyContext();

    if (!set.contains(element)) {
      set.add(element);

      return new SetAdditionBacktrackingInfo(
        set, element, arg.getNode(), ssc.getStateId(),
        ssc.getThreadId(), arg.getSchedulingStrategyInfo());
    }
    else {
      return bf.createNoChangeBacktrackingInfo( .
    }                                          ------
  }
  …
```

If the element is already in the set then do nothing

Extension Implementation

Implementing the set operations for the set type

```
public class SetModule implements IModule {

  protected static class SetAdditionBacktrackingInfo
     implements IActionBacktrackingInfo {

    ISetValue set;
    IValue element;

    …

    public void backtrack(IState state) {
      set.remove(element);
    }

    …
}
…
```

Backtrack by removing the element from the set

A BIR Example – Resource Contention

Demo

- Model with the Set extension
- Incorporating the extension in the Bogor Eclipse plugin
- Run the example
  - invalid end state
  - invariant checking

Assessments

- Bogor provides a clean and well-designed framework for extending its modeling language
  - Allows introduction of new abstract types and abstract operations as first-class construct
    - Complete control over value representation (linearization)
    - No double interpretation (the operations are executed as atomic actions in the model checker instead of being interpreted by the model checker)
  - Analogous to adding new instructions to a Virtual Machine
    - essentially, we are building abstract machines for particular domains

Outline

Customizations

Extending BIR

- Adding a Set Extension
- Demo

Cadena

- Abstract API for CORBA Real-time Event Channel
- Priority scheduling & relative time
- Quasi-cyclic search

BogorVM

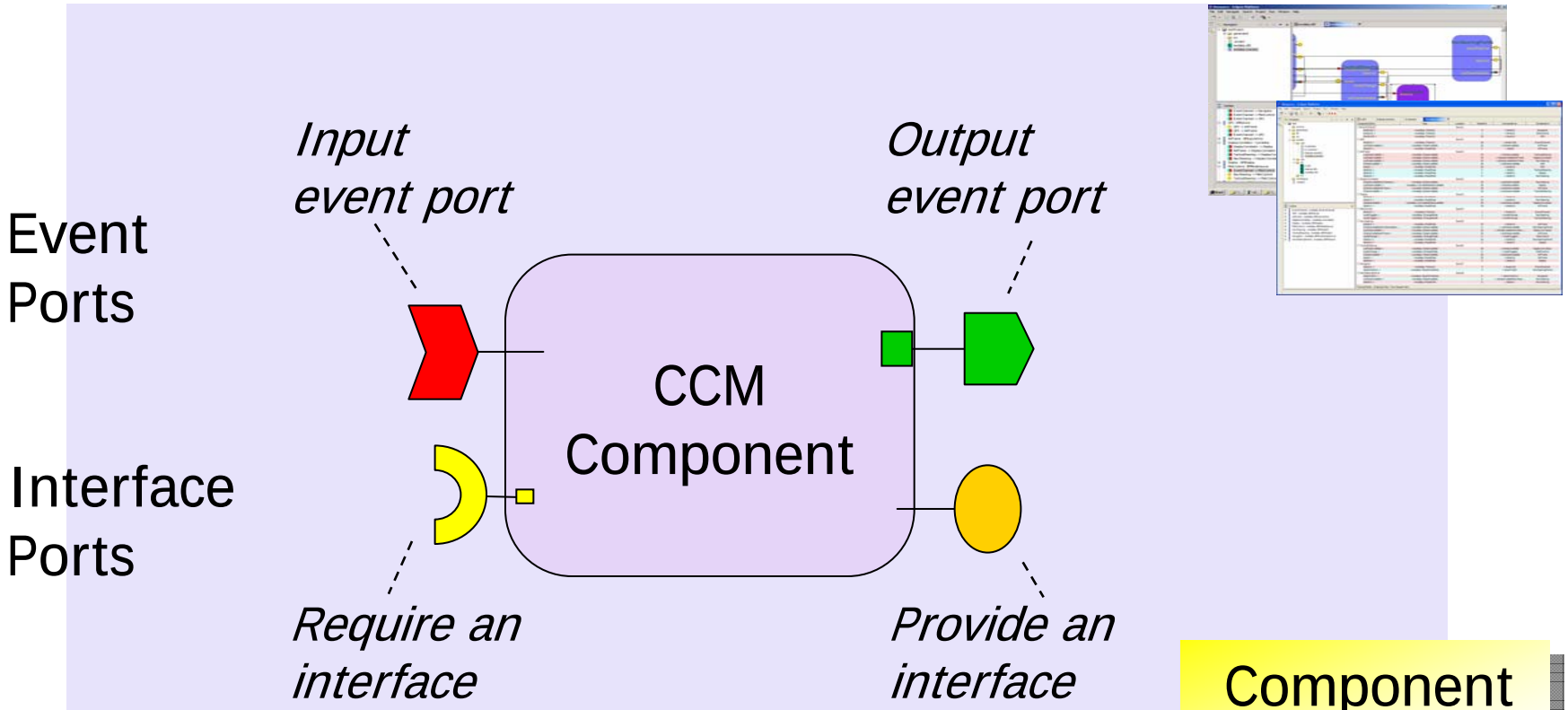
- a Java Virtual Machine and model checker in Bogor
- Modeling JVM instruction set as extensions
- Eclipse front-end

Avionics Mission Control Systems

- Mission-control software for Boeing military aircraft
- Boeing's Bold Stroke Avionics Middleware
  - ...built on top of ACE/TAO RT CORBA

- Provided with an Open Experimental Platform (OEP) from Boeing
  - a sanitized version of the real system
  - 100,000+ lines of C++ code (including RT CORBA middleware)
  - 50+ page document that outline development process and describe challenge problems
- One "Challenge Area"...
  - reasoning about system control-flow and system/component mode transitions
    - in essence, the sanitized version is ideal for model checking

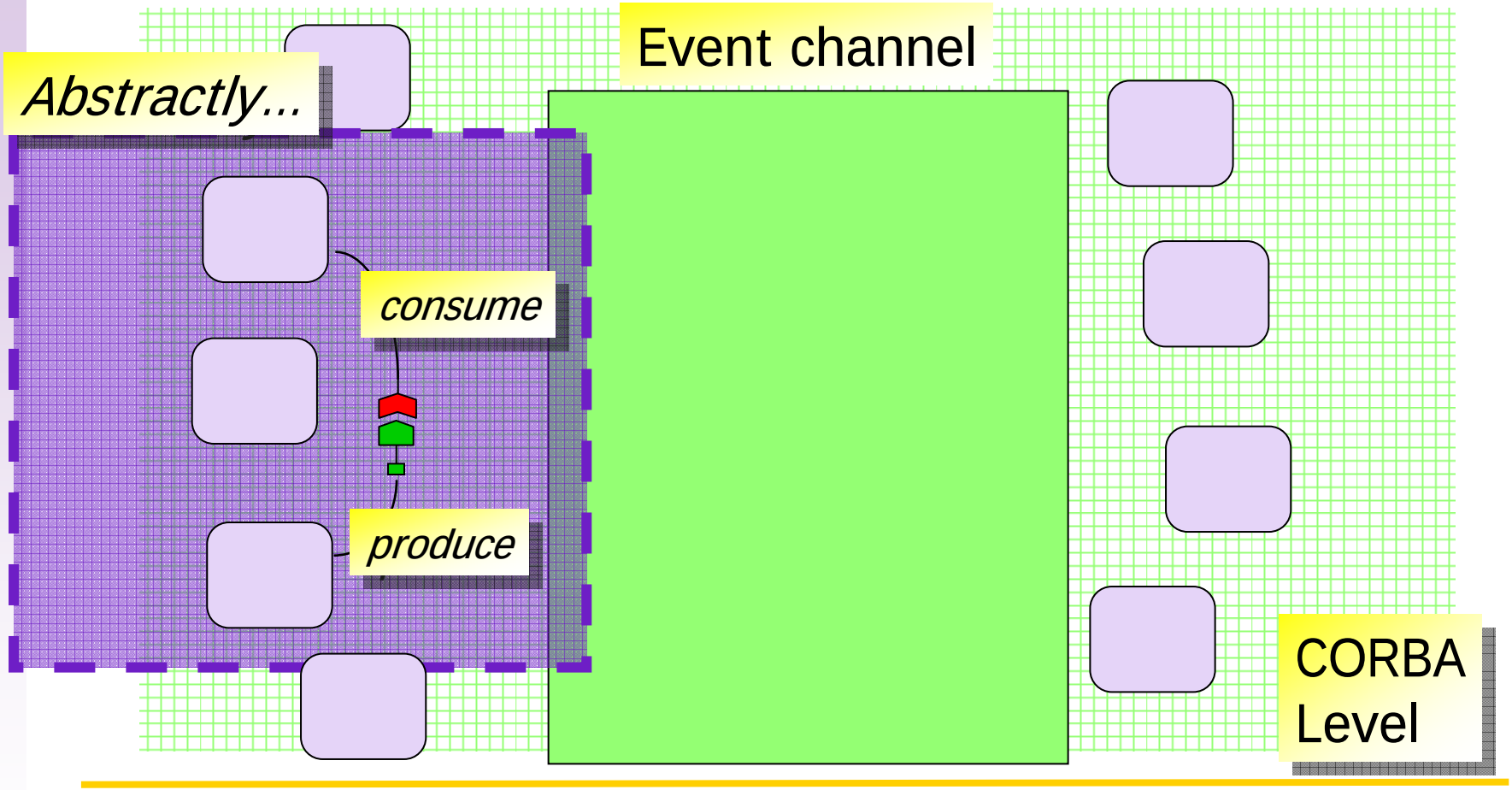
Component-based Design



Component Development

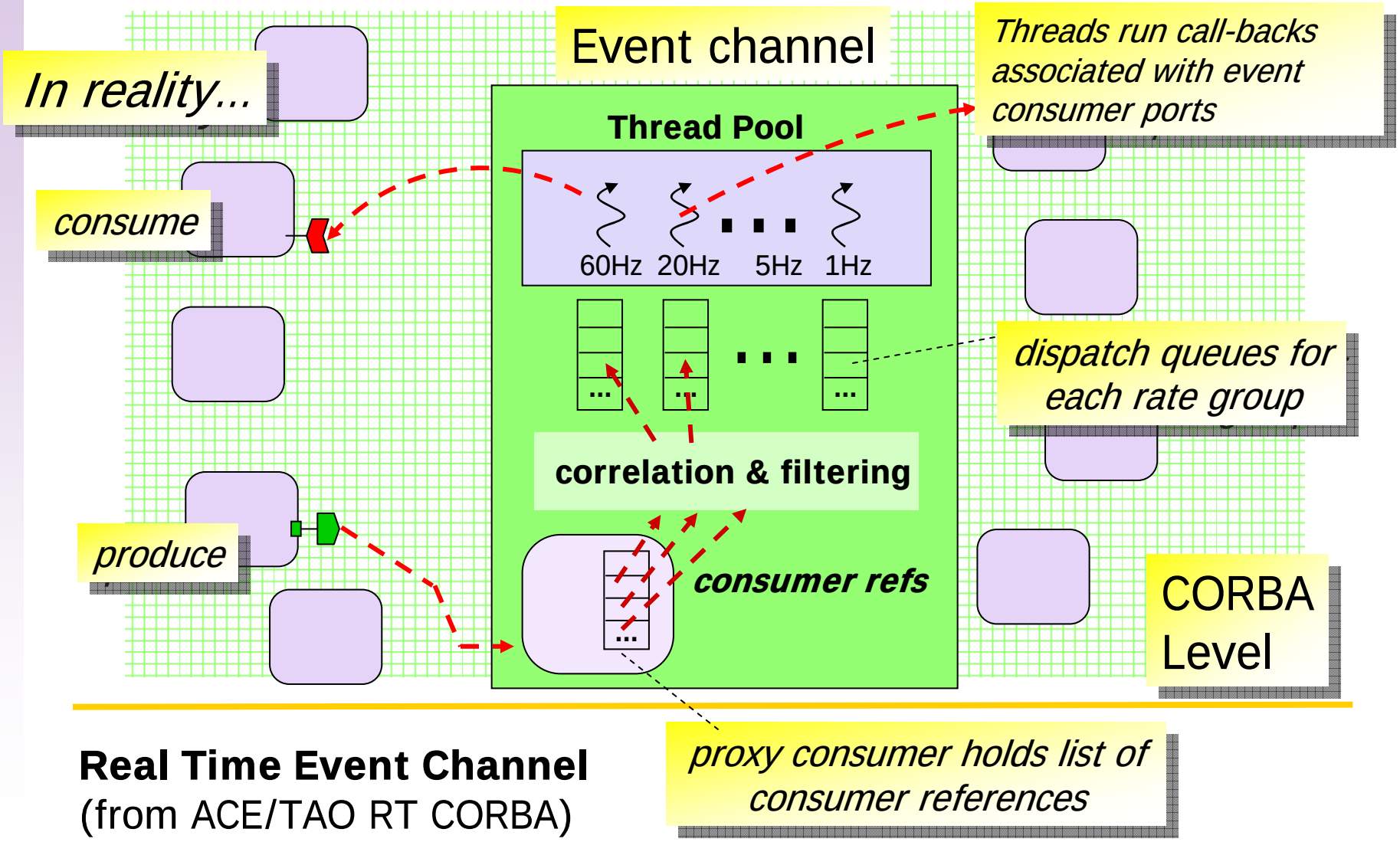
Cadena development environment allows model-based development of Bold Stroke applications using the CORBA Component Model (CCM)

RT Middleware-based Implementation



Real Time Event Channel
(from ACE/TAO RT CORBA)

RT Middleware-based Implementation



Real Time Event Channel
(from ACE/TAO RT CORBA)

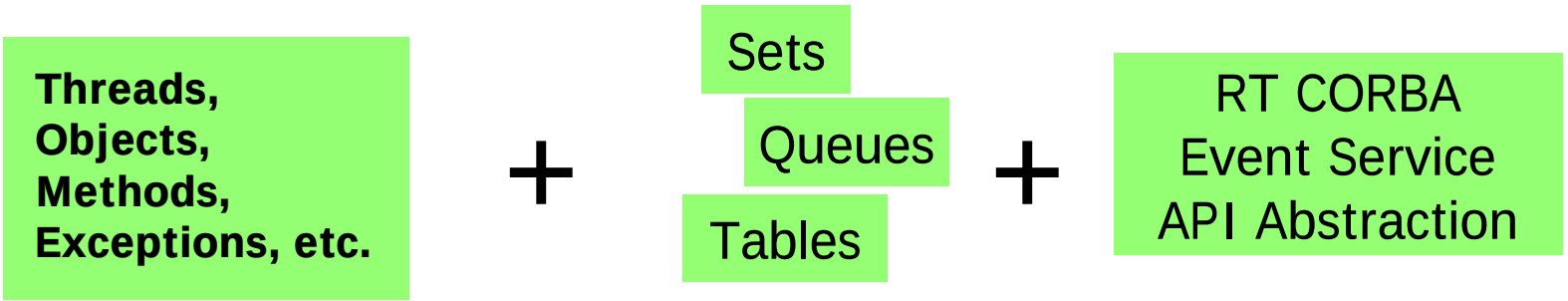
Effective for Boeing Software

How do we take the well-known explicit-state model-checking algorithms and enhance them to be effective for working directly on Boeing software?

- How do we model Boeing's mission control software?
- How do we reduce the number of paths/states explored?

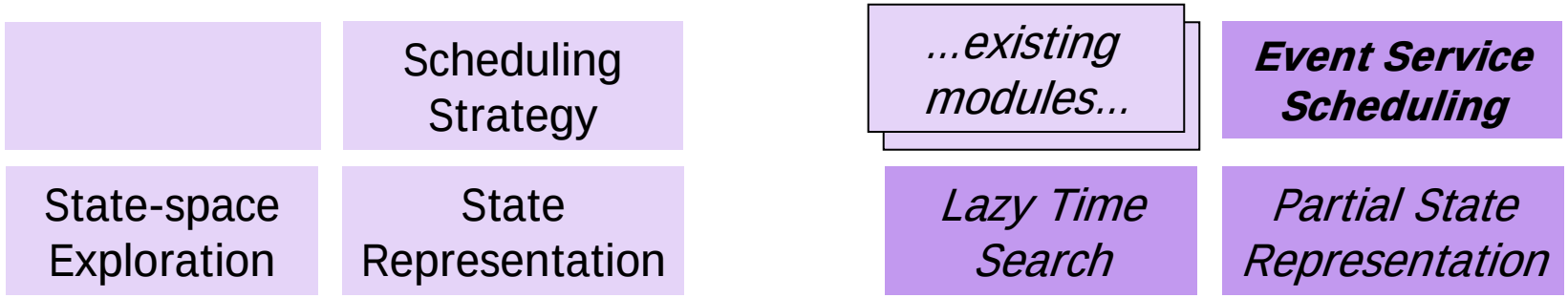
Bogor Customized To Cadena

Bogor -- Extensible Modeling Language



Core Modeling Language — Extensions — Domain-specific Abstractions

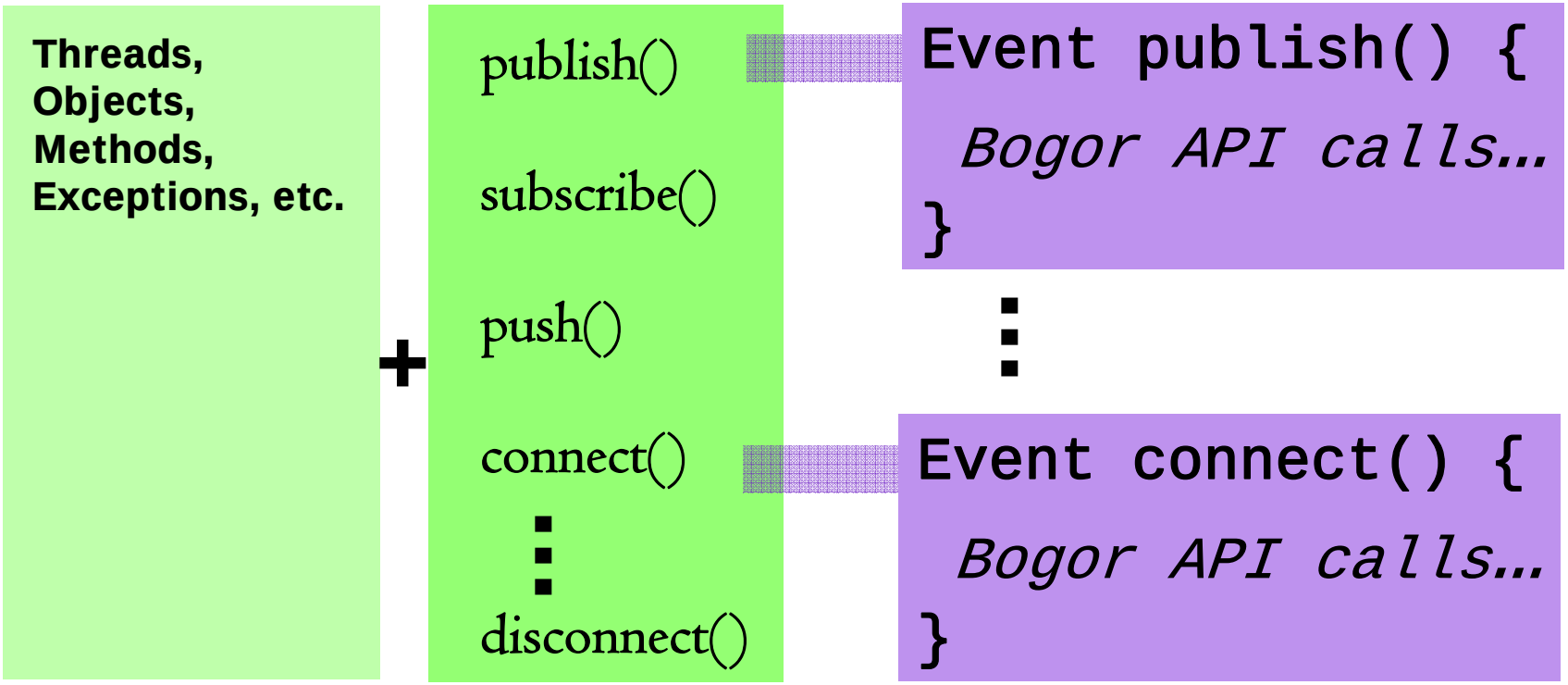
Bogor -- Customizable Checking Engine Modules



Core Checker Modules — Customized Checker Modules

Domain-Specific Modeling

Bogor -- Extensible Modeling Language

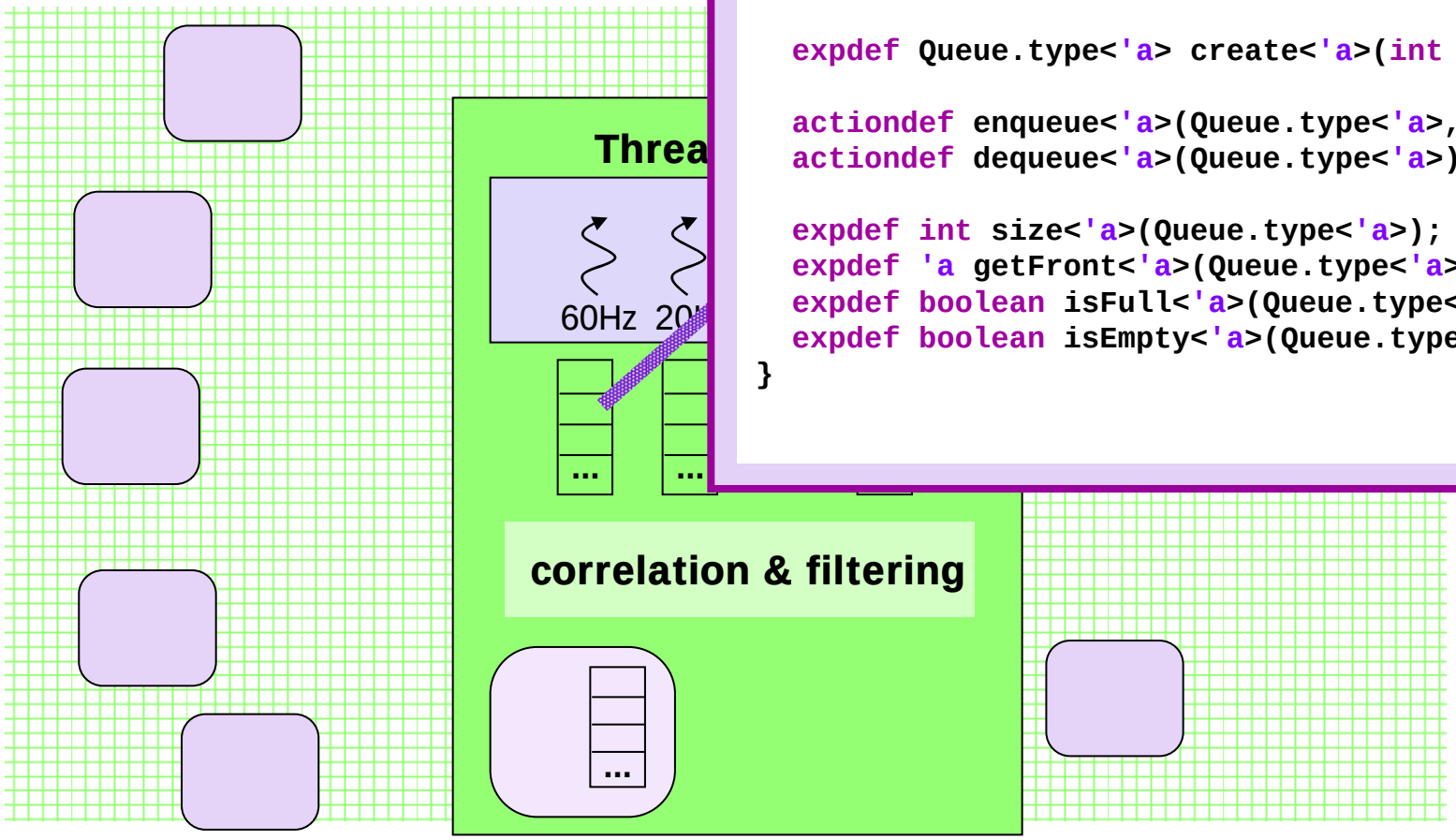


Core Modeling Language

New Bogor primitives corresponding to Event Channel API

Java implementation of new primitives inside model-checker

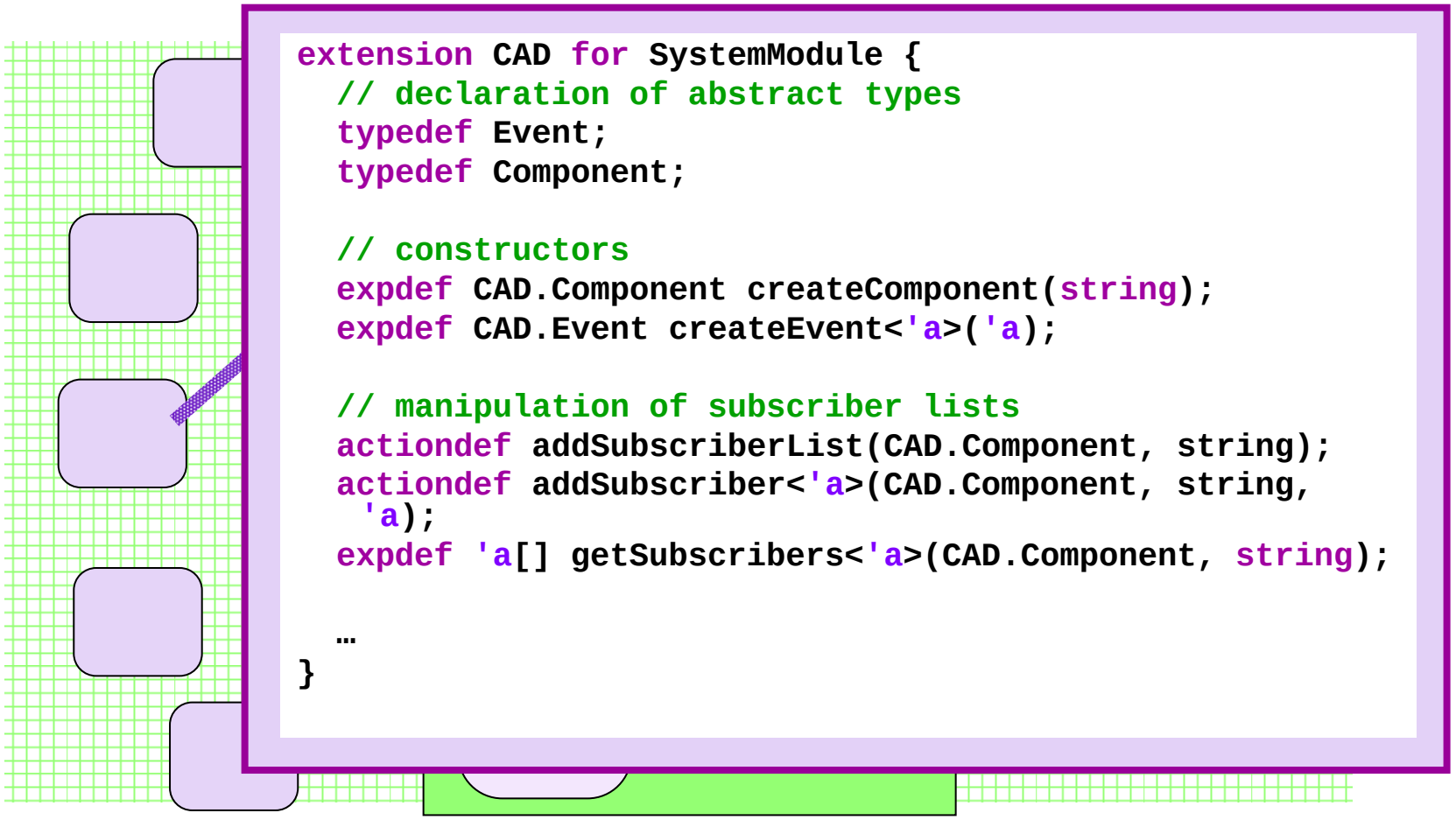
Bogor Modeling Extensions




```
extension Queue for cadena.Queue {
  typedef type<'a>;

  expdef Queue.type<'a> create<'a>(int size);

  actiondef enqueue<'a>(Queue.type<'a>, ‘a);
  actiondef dequeue<'a>(Queue.type<'a>);

  expdef int size<'a>(Queue.type<'a>);
  expdef 'a getFront<'a>(Queue.type<'a>);
  expdef boolean isFull<'a>(Queue.type<'a>);
  expdef boolean isEmpty<'a>(Queue.type<'a>);
}
```

Bogor extensions for representing event-channel queue data structures

Bogor Modeling Extensions



```
extension CAD for SystemModule {
  // declaration of abstract types
  typedef Event;
  typedef Component;

  // constructors
  expdef CAD.Component createComponent(string);
  expdef CAD.Event createEvent<'a>('a);

  // manipulation of subscriber lists
  actiondef addSubscriberList(CAD.Component, string);
  actiondef addSubscriber<'a>(CAD.Component, string,
    'a);
  expdef 'a[] getSubscribers<'a>(CAD.Component, string);

  …
}
```

Bogor extensions for representing CCM component API

Assessments of Previous Work

Cadena	dSPIN (ICSE'02)	Bogor (FMCO'02)
Boeing ModalSP ▪ 3 rate groups ▪ 8 components ▪ 125 events per hp	1.4 M states 58 sec 130 MB	9.1 K states 8.59 sec 1.61 MB
Boeing MediumSP ▪ 2 rate groups ▪ 50 components ▪ 820 events per hp		740 K states 3 min 21.5 MB

- want to check larger model
  - does not seem to scale well regardless aggressive reductions

See "*Model-checking Middleware-based Event-driven Real-time Embedded Software*". FMCO 2002

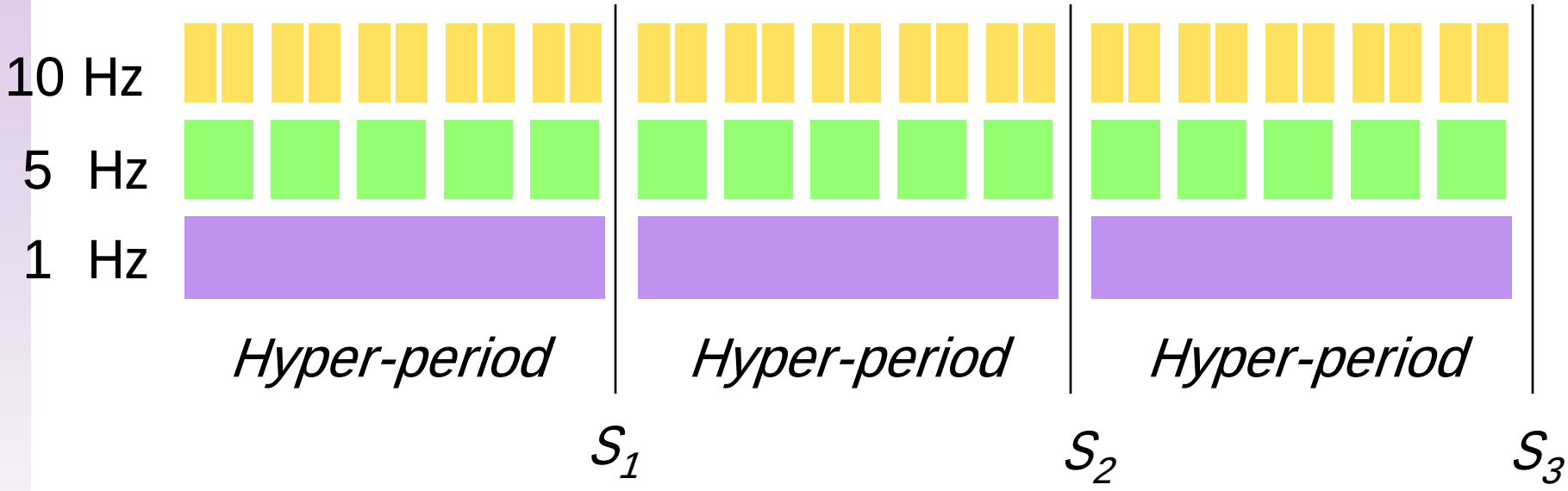
Key Observation

Leverage patterns of periodic computation

- use the structure of periodic systems to systematically drop states

Leveraging Periodic Structure

Periodic Tasks

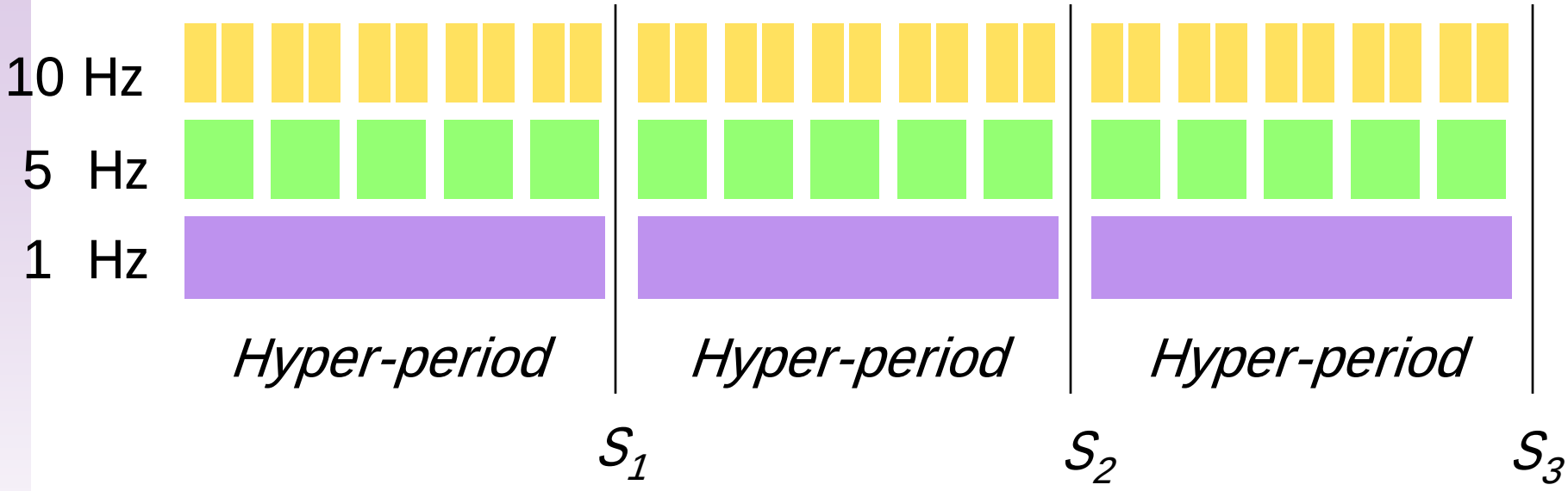


Basic Idea

- break the search into several regions
- divide the problem into smaller problems

Leveraging Periodic Structure

Periodic Tasks

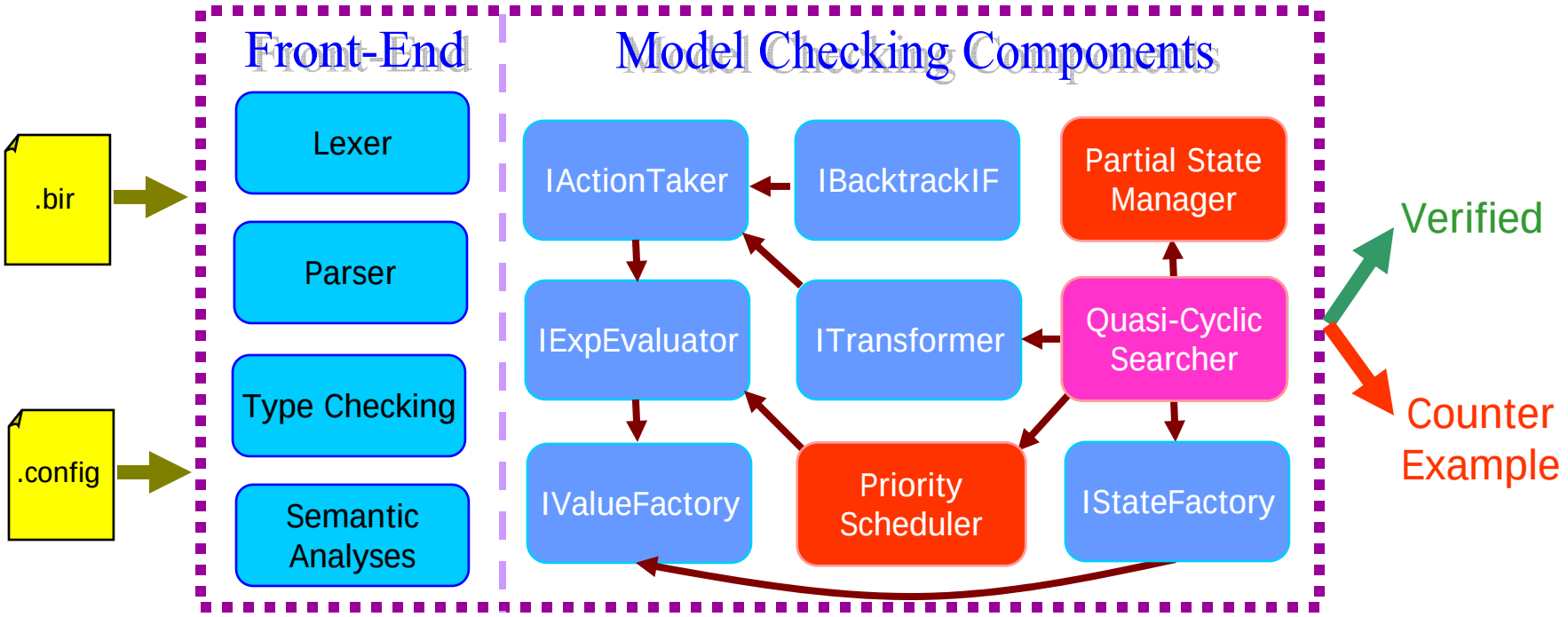


Macro-state Structure

- **Same at each macro-state**:
  - dispatch queues empty, threads idle, correlators are at initial state
- **Different**: component/system mode values are different

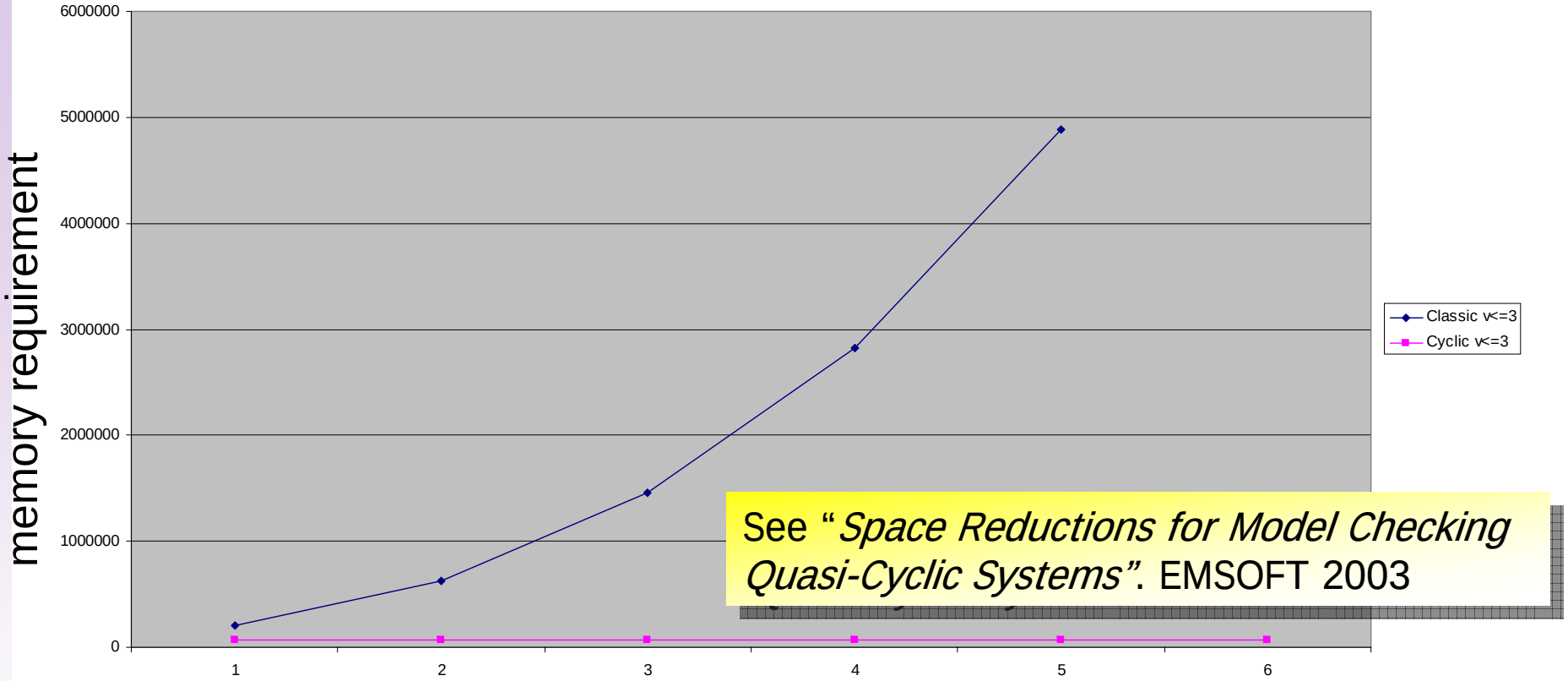
Domain-Specific Algorithms

Bogor -- Customizable Checking Engine Modules



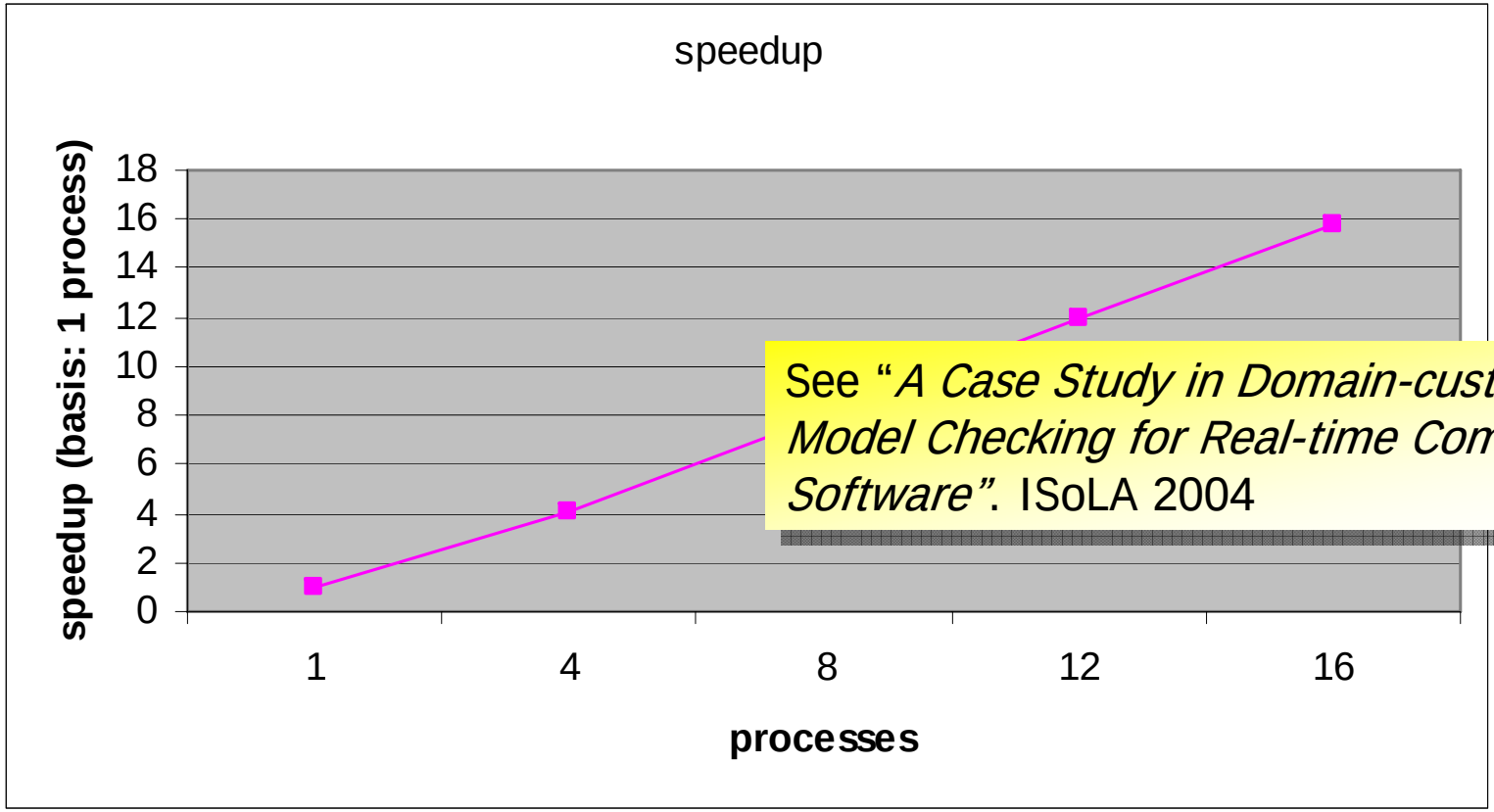
Bogor default modules are *unplugged* and *replaced* with state representation, scheduling and search strategies *customized to the Bold Stroke domain*

Scaling Boeing ModalSP



- both searches have exponential time growth
  - quasi-cyclic search takes more time (overlapping regions)
- parallel quasi-cyclic takes 25% less time than classical DFS

Distributed Quasi-cyclic search



See "*A Case Study in Domain-customized Model Checking for Real-time Component Software*". ISoLA 2004

- Adapt parallel version to distributed cluster
  - Centralized node for boundary state seen-set
  - Worker nodes to explore each autonomous region
- Almost-linear speed-up (somewhat higher memory than shared memory)

Outline

Customizations

Extending BIR

- Adding a Set Extension
- Demo

Cadena

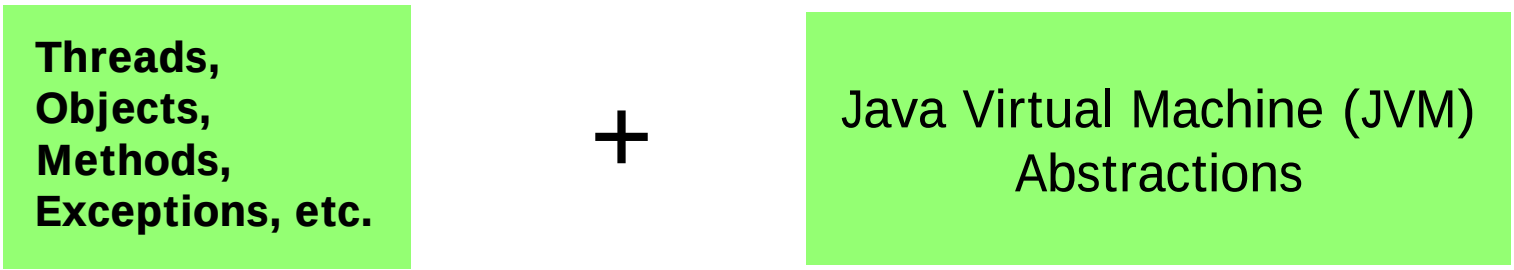
- Abstract API for CORBA Real-time Event Channel
- Priority scheduling & relative time
- Quasi-cyclic search

BogorVM

- a Java Virtual Machine and model checker in Bogor
- Modeling JVM instruction set as extensions
- Eclipse front-end

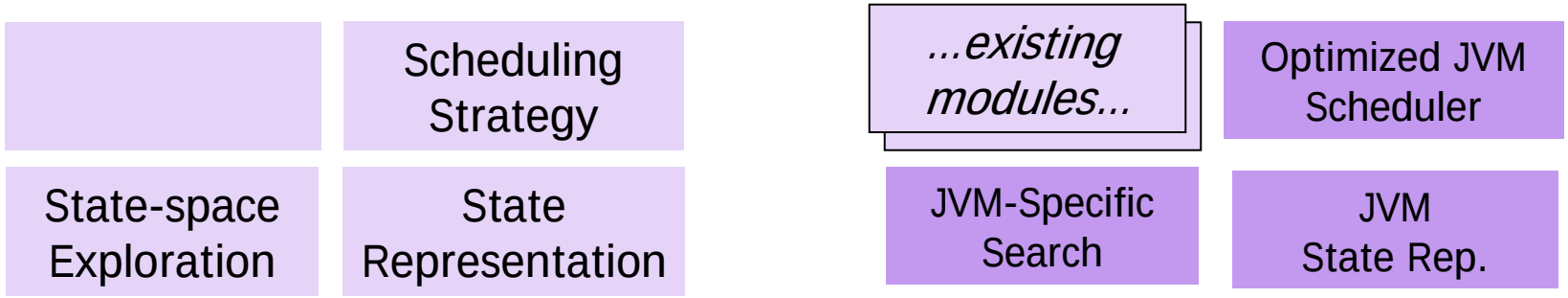
Customization Mechanisms

Bogor -- Extensible Modeling Language



Core Modeling Language

Bogor -- Customizable Checking Engine Modules



Core Checker Modules

Customized Checker Modules

Modelling Java Features – A Simple Deadlock Example

```
public class Deadlock {
  static Lock lock1;
  static Lock lock2;
  static int state;

  public static void main(String[] args) {
    lock1 = new Lock();
    lock2 = new Lock();
    Process1 p1 = new Process1();
    Process2 p2 = new Process2();
    p1.start();
    p2.start(); } }

class Process1 extends Thread {
  public void run() {
    Deadlock.state++;
    synchronized (Deadlock.lock1) {
      synchronized (Deadlock.lock2) {
        Deadlock.state++;
      }
    }
  }
}

class Lock { }

class Process2 extends Thread {
  public void run() {
    Deadlock.state++;
    synchronized (Deadlock.lock2) {
      synchronized (Deadlock.lock1) {
        Deadlock.state++;
      }
    }
  }
}
```

Modelling Java Classes

```
public class Deadlock {
  static Lock lock1;
  static Lock lock2;
  static int state;
  …
}
```

```
top record (|java.lang.Object|) {
  lock l;
}

record (|Deadlock|)
  extends (|java.lang.Object|) { }

int /|Deadlock.state|\;
(|Lock|) /|Deadlock.lock1|\;
(|Lock|) /|lock2|\;
```

- the record for `java.lang.Object` is modelled as the *top* record in BIR that contains a *lock* field
- the record for `Deadlock` extends `Object` and contains *no record field declaration* because there is no instance variables
- *static fields* in `Deadlock` are modeled as *global variables* in BIR

Modelling Methods

```
public static
        void main(String[] args) {
  …
}


public void run() {
  …
}
```

```
function
{|Deadlock.main([Ljava/lang/String;)
V|}(VM.F f) {
  …
}

function {|Process1.run()V|}
    (transient (|Process1|) this,
     VM.F f) {
  …
}
```

- Static methods are translated as functions that have only one parameter, i.e., VM.F (frame)
- Instance methods are translated as functions that have two parameters, the “this” variable, and the VM.F
- VM.F is contains information about the stack frame of the methods (*i.e.*, JVM local and stack slots)

Modelling VM Instructions

```
const Op { ACONST_NULL = 1; ICONST_M1 = 2; ICONST_0 = 3; ICONST_1 = 4; … }

extension VM for ...VM {
  actiondef zro (VM.F frame, int op);
  actiondef one (VM.F frame, int op, int);
  actiondef lcl (VM.F frame, int op, int localIndex);
  actiondef fld (VM.F frame, int op, string fieldName);
  actiondef typ <'a>(VM.F frame, int op);
  actiondef cmp (VM.F frame, int op, lazy boolean result);
  actiondef arr <'a>(VM.F frame, int dims);
  actiondef inc (VM.F frame, int var, int inc);
  …
}
```

- Bytecode instructions are categorized (similar to ASM):
  - **`zro`** for instructions without any parameter, e.g., **`ACONST_NULL`**
  - **`one`** for instructions with one integer parameter, e.g., **`BIPUSH`**
  - **`lcl`** for instructions that access local slots, e.g., **`ILOAD`**
  - **`fld`** for instruction that access object fields, e.g., **`GETSTATIC`**

Modelling VM Instructions (Cont'd)

```
const Op { ACONST_NULL = 1; ICONST_M1 = 2; ICONST_0 = 3; ICONST_1 = 4; … }

extension VM for ...VM {
  actiondef zro (VM.F frame, int op);
  actiondef one (VM.F frame, int op, int);
  actiondef lcl (VM.F frame, int op, int localIndex);
  actiondef fld (VM.F frame, int op, string fieldName);
  actiondef typ <'a>(VM.F frame, int op);
  actiondef cmp (VM.F frame, int op, lazy boolean result);
  actiondef arr <'a>(VM.F frame, int dims);
  actiondef inc (VM.F frame, int var, int inc);
  …
}
```

- Bytecode instructions are categorized (similar to ASM):
  - **`typ`** for instructions involving types, e.g., **`CHECKCAST`**
  - **`cmp`** for comparison instructions, e.g., **`IFEQ`**
  - **`arr`** for the **`MULTINANEWARRAY`** instruction
  - **`inc`** for increment/decrement instructions, i.e., **`IINC`**

VM Instruction Examples

```
function {|Process1.run()V|}(transient (|Process1|) this, VM.F f) {
  …

  loc l0$27: do {
               VM.max(f, 3, 2); VM.set<(|Process1|)>(f, "this", 0);
               VM.fld(f, Op.GETSTATIC, "/|Deadlock.state|\\");
             } goto l1;

  loc l1:    do independent { VM.zro(f, Op.ICONST_1); }
             goto l2;

  loc l2:    do independent { VM.zro(f, Op.IADD); }
             goto l3$27;

  loc l3$27: do { VM.fld(f, Op.PUTSTATIC, "/|Deadlock.state|\\"); }
             goto l4$28;

  loc l4$28: do { VM.fld(f, Op.GETSTATIC, "/|Deadlock.lock1|\\"); }
             goto l5;

  loc l5:    do independent { VM.zro(f, Op.DUP); }
             goto l6;

  loc l6:    do independent { VM.lcl(f, Op.ASTORE, 1); }
             goto l7;
  …
}
```

Implementing A VM Instruction - `zro`

```
public IActionBacktrackingInfo zro(IExtArguments args) {

  Frame f = (Frame) args.getArgument(0);

  int opcode = ((IIntValue) args.getArgument(1)).getInteger();

  switch (opcode) {

    case ACONST_NULL:
        return opACONST_NULL(args, f);

    case ICONST_M1:
        return opICONST_M1(args, f);

    …
 }
}
```

Implementing A VM Instruction – ACONST_NULL

```
protected IActionBacktrackingInfo opACONST_NULL(IExtArguments args,
                                                Frame f)
{ return pushValue(args, f, vf.newNullValue()); }

protected IActionBacktrackingInfo pushValue(IExtArguments args,
                                            Frame f, IValue v) {
  ISchedulingStrategyInfo ssi = args.getSchedulingStrategyInfo();
  Action a = (Action) args.getNode();
  f.pushStack(v);
  return vmbif.createPushPopStackBacktrackingInfo(args
              .getContainingTransition(), ssi, a, f, 1);
}

public IActionBacktrackingInfo createPushPopStackBacktrackingInfo(
        final ITransformationBacktrackingInfo parent,
        final ISchedulingStrategyInfo ssi, final Node n, final Frame f,
        final int num, final IValue... vs) {
  return new IActionBacktrackingInfo() {
    public void backtrack(IState state) {
      for (int i = 0; i < num; i++) { f.popStack(); }
      for (IValue v : vs) { f.pushStack(v); }
    }
  };
}
```

Implementing A VM Instruction – GETSTATIC, PUTFIELD

```
protected IActionBacktrackingInfo opGETSTATIC(IExtArguments args,
                                    IState s, Frame f, String fieldName) {
  ISchedulingStrategyInfo ssi = args.getSchedulingStrategyInfo();
  Action a = (Action) args.getNode();
  int globalIndex = st.getGlobalIndexTable().get(fieldName);
  f.pushStack(s.getGlobalValue(globalIndex));
  return vmbif.createPushPopStackBacktrackingInfo(args
              .getContainingTransition(), ssi, a, f, 1); }

protected IActionBacktrackingInfo opGETFIELD(IExtArguments args,
                                    IState s, Frame f, String fieldName) {
  ISchedulingStrategyInfo ssi = args.getSchedulingStrategyInfo();
  Action a = (Action) args.getNode();
  IValue ov = f.popStack();
  if (ov instanceof INullValue) { f.pushStack(ov);
                                  throw new NullPointerBogorException(); }
  IRecordValue rv = (IRecordValue) ov;
  RecordType rt = (RecordType) rv.getType();
  String qFieldName = resolveFieldName(new Pair<RecordType, String>(
                                                    rt, fieldName), 0);
  int fieldIndex = rt.getFieldIndex(qFieldName);
  f.pushStack(rv.getFieldValue(fieldIndex));
  return vmbif.createPushPopStackBacktrackingInfo(args
              .getContainingTransition(), ssi, a, f, 1, rv); }
```

Assessment

- BIR extension mechanism is rich enough to model JVM instructions
  - lifting BIR to the JVM level
  - direct representation of Java class files
    - more faithful execution model
    - less hoops to jump when translating counter examples
- simple translator from bytecode to BIR
- strict bytecode representation can be leveraged for optimizations (*almost* for free)
  - independent annotation, thread-locality for POR
- there is an up-front cost to model each instruction, but this is done only once

BogorVM-Eclipse Integration

- Main Goal
  - lower the entry barrier for using Bogor as a Java checking engine
- Features
  - incremental compilation of Java bytecode to BIR in Eclipse that is synchronized with Eclipse JDT build process
  - a viewer to display incrementally compiled BIR
  - a launcher in Eclipse to invoke Bogor and check the compiled code, and
  - a Java counter example display on top of Bogor's counter example display
- A prototype done by Yong Peng, MSE
- The user manual is available at: http://www.cis.ksu.edu/~yongpeng/phase3/UserManual.pdf
- Demo!

Build Your Own Model Checker!

Bogor Web Site

http://bogor.projects.cis.ksu.edu

- Significant “out of the box” capabilities for supporting modern software systems
- Novel extension and customization facilities
  - used by us
  - used by others
- Documentation and support
- Pedagogical material and courseware