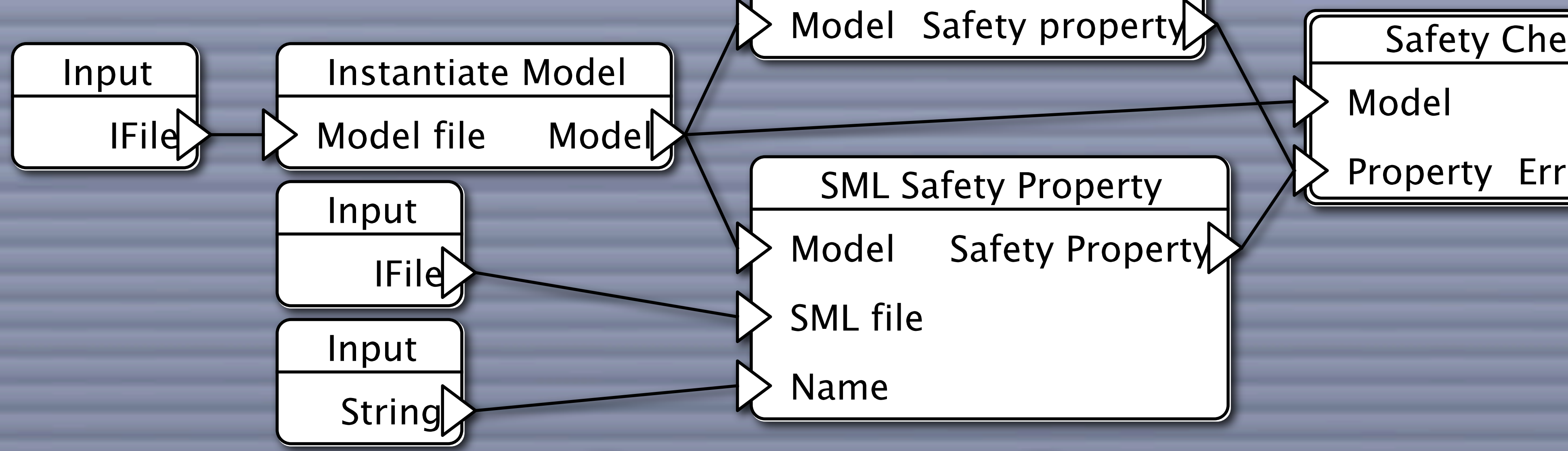
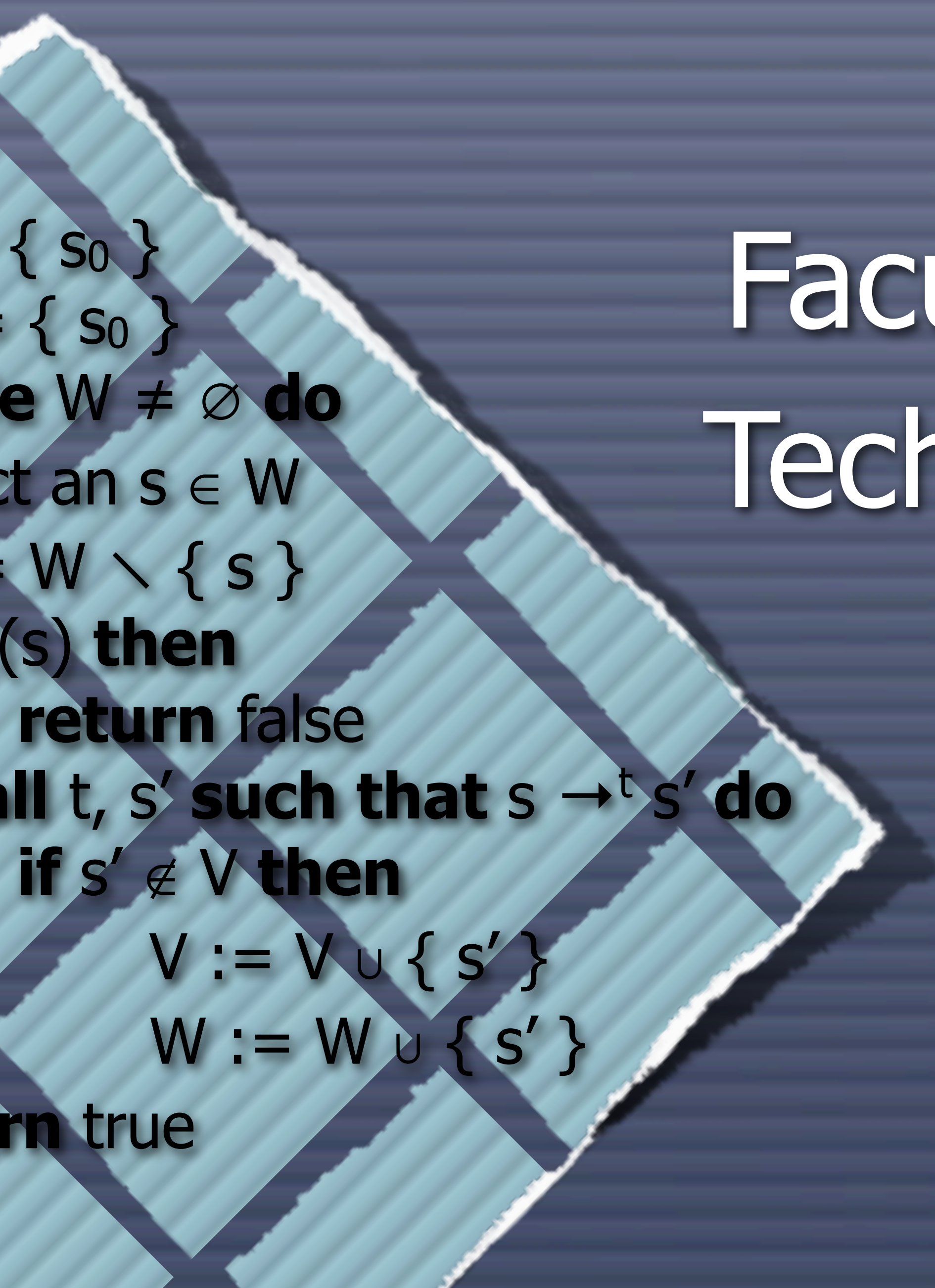
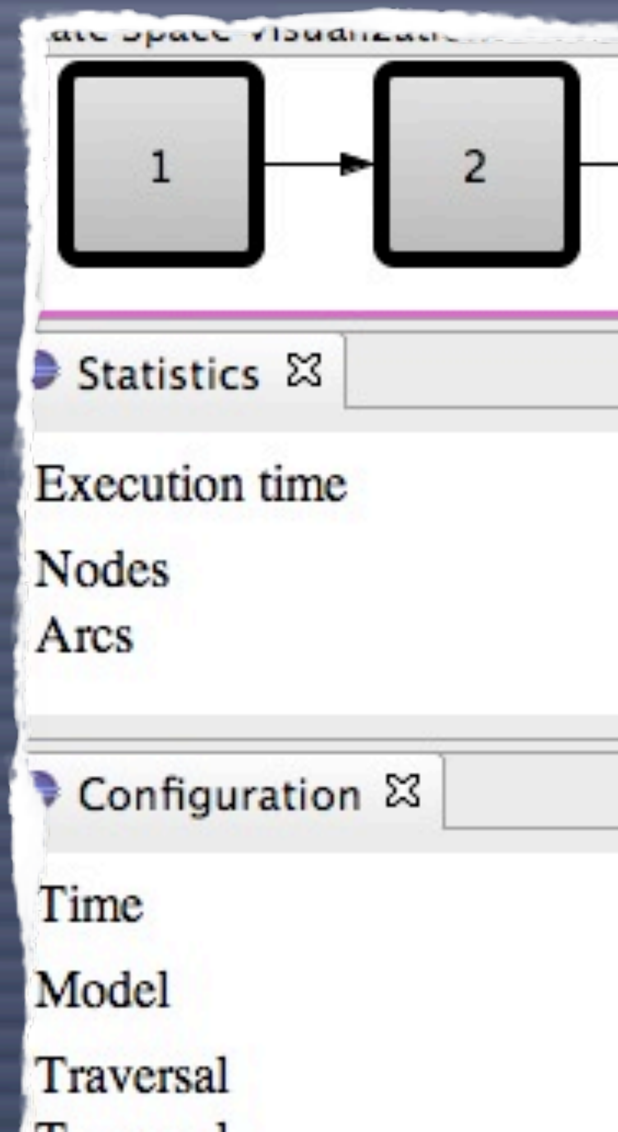


State Space Exploration and ASAP: Research Perspective

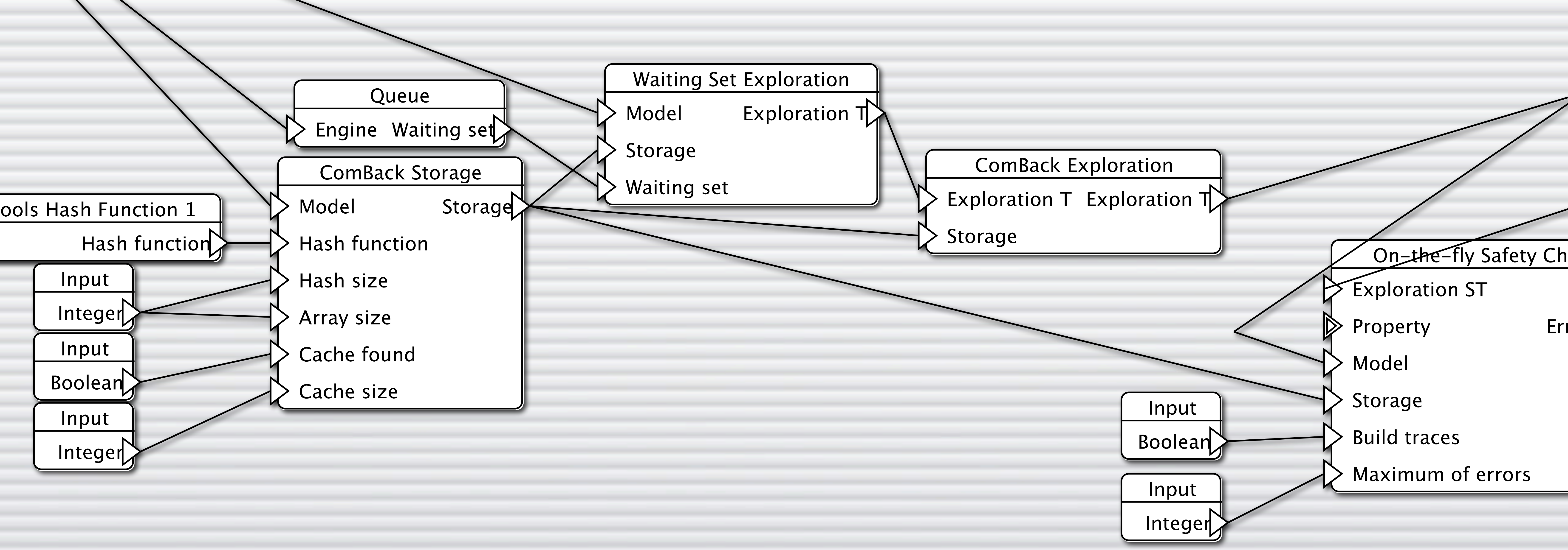
Michael Westergaard
Faculteit Wiskunde & Informatica
Technische Universiteit Eindhoven
m.westergaard@tue.nl





Outline

- Advanced methods in ASAP
- Integrating new methods
  - Briefly Access/CPN
- Benchmarking
- Status and outlook



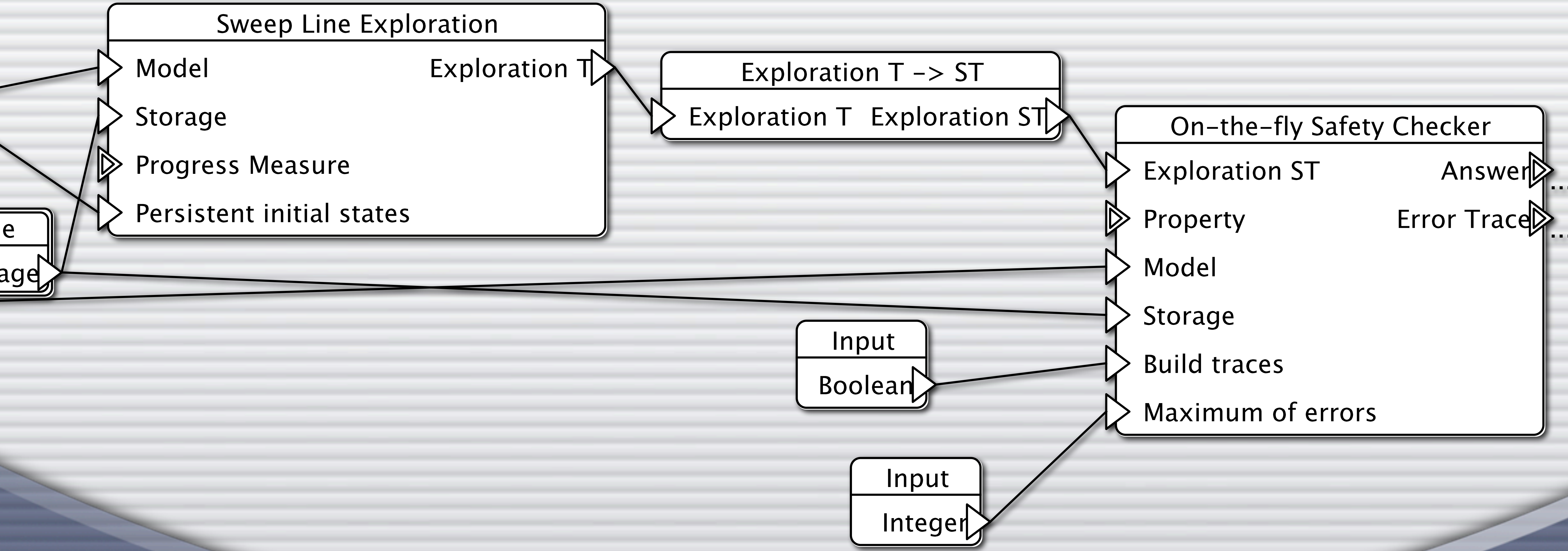
Example: The ComBack Method

Briefly:
The ComBack Method

- Use the hash compaction method
- Use a table of back-edges to resolve hash conflicts
- Use caching to improve speed

Demo:
The ComBack Method

- Create new from template
- Run check



Example:
The Sweep-line Method

Briefly:
The Sweep-line Method

- Uses notion of progress in model identified by a **progress measure**
- A conceptual **sweep-line** marks a border between states that have already been discovered
- Only states in front of the sweep-line is kept in memory

Briefly: A Progress Measure

- ASAP automatically generates a template progress measure (much like queries)
- We just have to fill in the blanks
- Let's use the number of eating philosophers as the **progress value**

Example: Progress Meassure

```
fun query (state, events) =
let
  fun query'New_Page { Waiting, Has_One, Eating,
                       Philosophers, Initialized,
                       Chopsticks } =
    List.length Eating
  fun query'state { New_Page} = query'New_Page New_Page
in
  query'state state
end
```

Demo:
The Sweep-line Method

- Create new from template
- Create progress measure
- Run check
- Combine with hash compaction

Example: The Sweep-line Method

- The sweep-line is defined completely outside of the ASAP main application (proof-of-concept, eat-your-own-dog-food, ...)
- Yet...
  - We can add it in the JoSEL editor
  - We can use it with the safety-checker
  - We can combine it with hash compaction
  - We can create progress measures as easily as we create safety properties
  - The progress measure shows up in the report

Overview

- Adding new methods to the GUI
  - Eclipse's plug-in system
- Adding new methods to the engine
- Extending JoSEL
- Adding entries to the report
- Briefly: ACCESS/CPN

Basically, this is Easy!

- ASAP is an Eclipse Rich Client application, so we have access to Eclipse’s plug-in mechanism
- This allows us to easily add new GUI elements (like the wizard for creating progress measures)
- This allows us to specify new points where the application can be extended

Eclipse's Plug-in System: Plug-ins

- **Plug-ins**: a program unit that provides a bounded functionality (e.g., the sweep-line method)
- **Dependencies**: a plug-in may (acyclically) depend on one or more other plug-ins (e.g., the sweep-line method depends on the generic state-space tool in ASAP)

Eclipse's Plug-in System: Extensions

- A plug-in may define zero or more **extension points** (e.g., new entries to add to the right-click menu in the index)
- An extension point can define any number of **details** (like the class implementing the wizard or when the menu entry should be enabled)
- An **extension point** provides an implementation of an extension point

Plug-ins in ASAP

- ASAP uses mostly standard or slightly specialized standard components
- These thus get a lot of extensibility automatically
  - E.g., adding an entry to the right-click menu of the queries folder for creating a progress measure

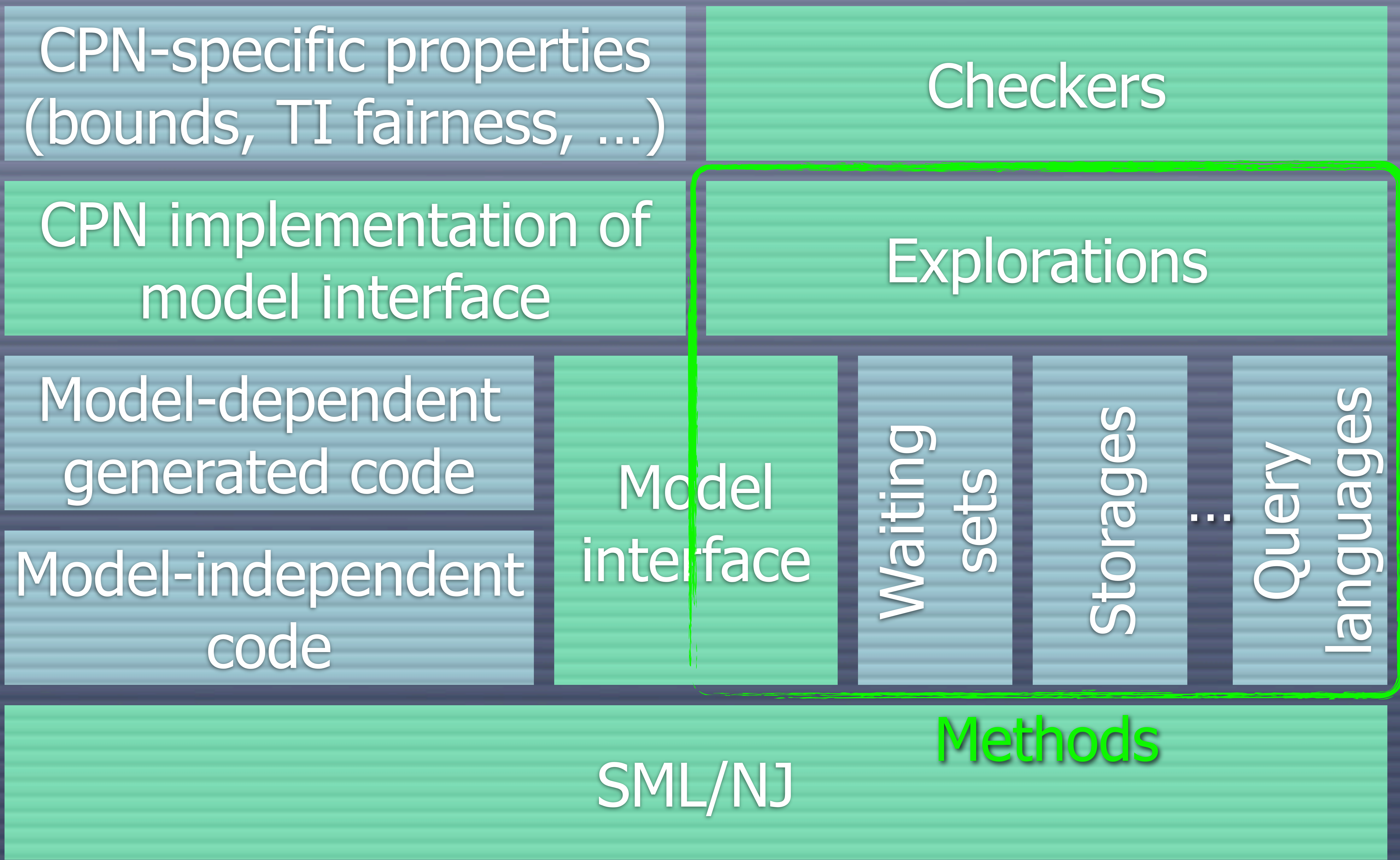
Interfaces

- In order to make this possible we need to adhere to the principle

Program to the interface, not the implementation

- A plug-in defining an extension point describes which values are allowed, including which interfaces they must implement
  - The plug-in only has access to implementations via the interface

State-space Tool of ASAP



Adding New Methods

- The state space engine of ASAP also introduces strict interfaces
  - Model, Storage, WaitingSet, Exploration (actually several explorations)
- Adding a new method should depend on these interfaces and implement interfaces (or define new interfaces and implement them)

Interfaces in SML

- SML uses **signatures** for interfaces
- Modules implementing interfaces are called **structures** or **functors**
  - Functors can explicitly depend on other structures and should be preferred over structures

```
fun sweep ([], storage, sVal, aVal) = (storage, sVal, aVal)
  | sweep (roots, storage, sVal, aVal) = let
      val _ = sweepHook (List.map #1 roots, storage)
      (*
       *  put root states into the queue (the toDel bit is set to false)
       *)
      val queue =
        List.foldl
            (fn ((s, id, trace), q) =>
              PQ.insert ((s, id, getProgress s, false, trace), q))
            (PQ.mkQueue (fn (_, _, prog, _, _) => prog)) roots
      val (toDel, roots, storage, sVal, aVal) =
        PQ.fold handleState queue (NONE, [], storage, sVal, aVal)
      val (storage, sVal, aVal) =
          toDel
```

Example: Sweep-line Exploration

Sweep-line Exploration Functor

```
(*
 *  default sweep line exploration
 *)
functor SweepLineExploration(
val markInitStatesAsPersistent : bool
structure Storage              : REMOVE_STORAGE
structure Model                : MODEL
structure Measure              : PROGRESS_MEASURE
   where type state = Model.state * Model.event list
sharing type Model.state = Storage.item
): SWEEP_LINE_EXPLORATION = struct
```

Sweep-line Exploration Functor

We require:
a boolean

```
(*
 *  default sweep line exploration
 *)
functor SweepLineExploration(
val markInitStatesAsPersistent : bool
structure Storage              : REMOVE_STORAGE
structure Model                : MODEL
structure Measure              : PROGRESS_MEASURE
   where type state = Model.state * Model.event list
sharing type Model.state = Storage.item
): SWEEP_LINE_EXPLORATION = struct
```

Sweep-line Exploration Functor

```
(*
 *  default sweep line exploration
 *)
functor SweepLineExploration(
val markInitStatesAsPersistent : bool
structure Storage              : REMOVE_STORAGE
structure Model                : MODEL
structure Measure              : PROGRESS_MEASURE
   where type state = Model.state * Model.event list
sharing type Model.state = Storage.item
): SWEEP_LINE_EXPLORATION = struct
```

We require:
a boolean
a storage

Sweep-line Exploration Functor

We require:
a boolean
a storage
a model

```
(*
 *  default sweep line exploration
 *)
functor SweepLineExploration(
val markInitStatesAsPersistent : bool
structure Storage              : REMOVE_STORAGE
structure Model                : MODEL
structure Measure              : PROGRESS_MEASURE
   where type state = Model.state * Model.event list
sharing type Model.state = Storage.item
): SWEEP_LINE_EXPLORATION = struct
```

Sweep-line Exploration Functor

```
(*
 *  default sweep line exploration
 *)
functor SweepLineExploration(
val markInitStatesAsPersistent : bool
structure Storage              : REMOVE_STORAGE
structure Model                : MODEL
structure Measure              : PROGRESS_MEASURE
   where type state = Model.state * Model.event list
sharing type Model.state = Storage.item
): SWEEP_LINE_EXPLORATION = struct
```

We require:
a boolean
a storage
a model
a progress measure

Sweep-line Exploration Functor

```
(*
 *  default sweep line exploration
 *)
functor SweepLineExploration(
val markInitStatesAsPersistent : bool
structure Storage              : REMOVE_STORAGE
structure Model                : MODEL
structure Measure              : PROGRESS_MEASURE
   where type state = Model.state * Model.event list
sharing type Model.state = Storage.item
): SWEEP_LINE_EXPLORATION = struct
```

We require:
a boolean
a storage
a model
a progress measure

We provide:
a sweep-line
exploration

Sweep-line Exploration Interfaces

- The “PROGRESS_MEASSURE” signature is defined by the sweep-line plug-in (and only applicable for the sweep-line method)
- The “SWEEP_LINE_EXPLORATION” signature is defined by the sweep-line plug-in, but extends the “TRACE_EXPLORATION” provided by ASAP

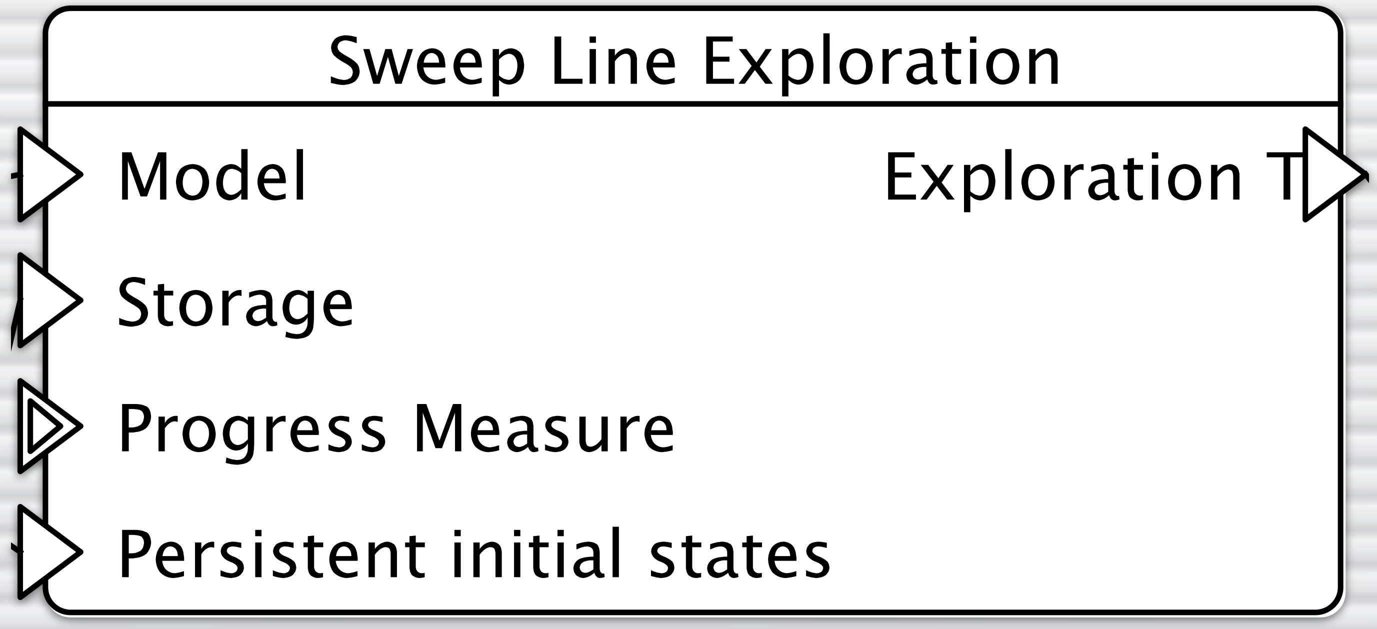
Sweep-line Exploration Interfaces

- The “PROGRESS_MEASSURE” signature is defined by the sweep-line plug-in (and only applicable fo
- The “SWEE
  signature is -
  in, but exte ”
  provided by

Or, reiterating an earlier point:
The sweep-line method
depends on previously defined
interfaces and implements one
of these interfaces

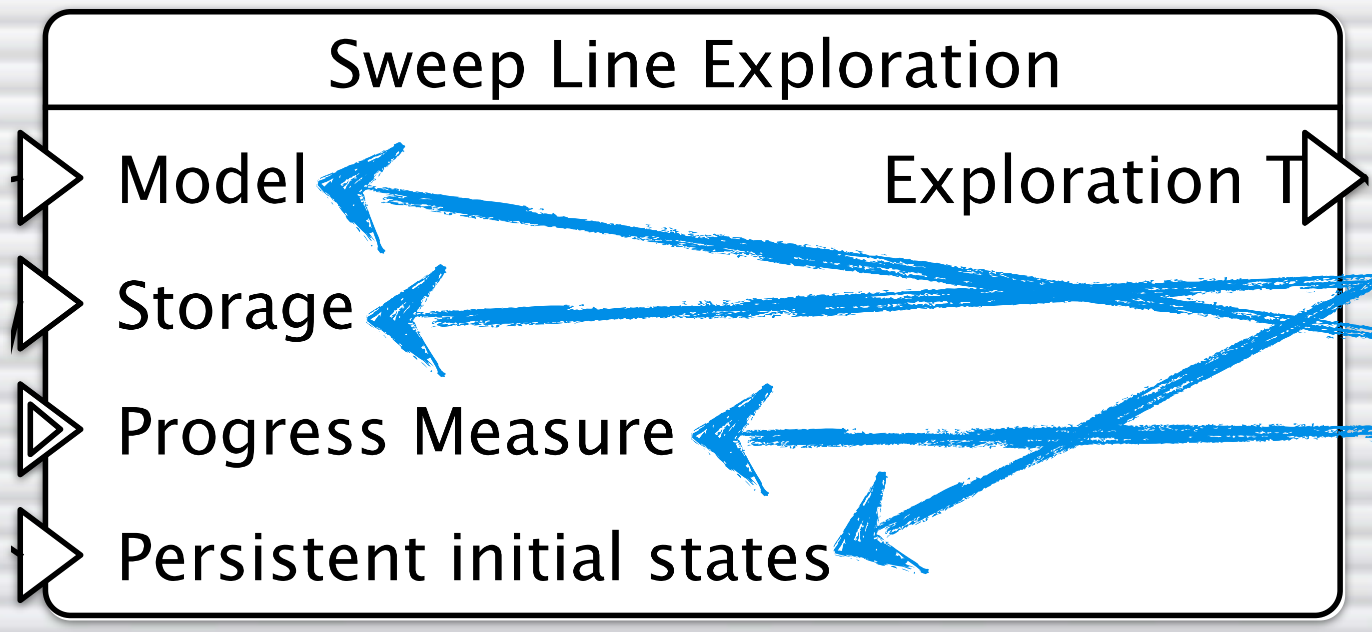
Extending JoSEL

- When we have developed a new method, we wish to integrate it into the GUI of ASAP
- JoSEL can be extended by adding new tasks (ASAP defines an extension point for this)
- We basically need to create a task for each functor we create
- EMF makes all the boiler-plate code for us, and ASAP contains abstract classes that do most of the work



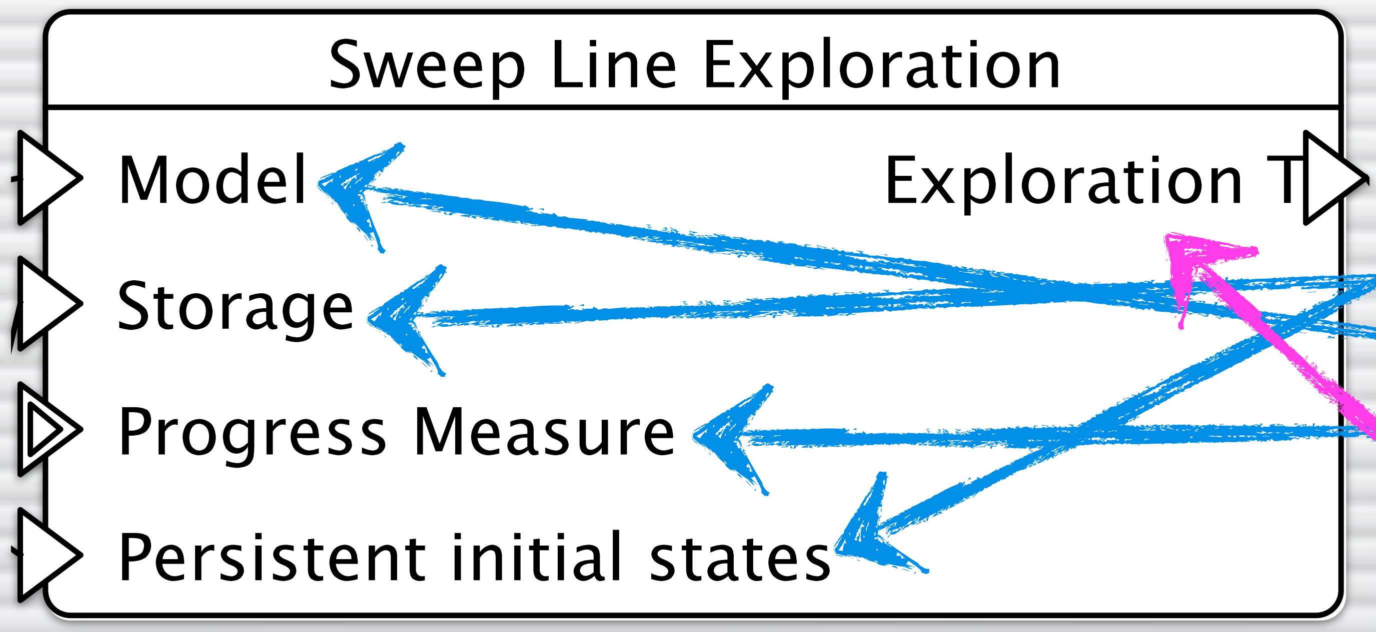

```
(*
 *  default sweep line exploration
 *)
functor SweepLineExploration(
val markInitStatesAsPersistent : bool
structure Storage               : REMOVE_STORAGE
structure Model                 : MODEL
structure Measure               : PROGRESS_MEASURE
   where type state = Model.state * Model.event list
sharing type Model.state = Storage.item
): SWEEP_LINE_EXPLORATION = struct
```

Example:
Sweep-line Exploration




```
(*
 *  default sweep line exploration
 *)
functor SweepLineExploration(
val markInitStatesAsPersistent : bool
structure Storage              : REMOVE_STORAGE
structure Model                : MODEL
structure Measure              : PROGRESS_MEASURE
   where type state = Model.state * Model.event lis
sharing type Model.state = Storage.item
): SWEEP_LINE_EXPLORATION = struct
```

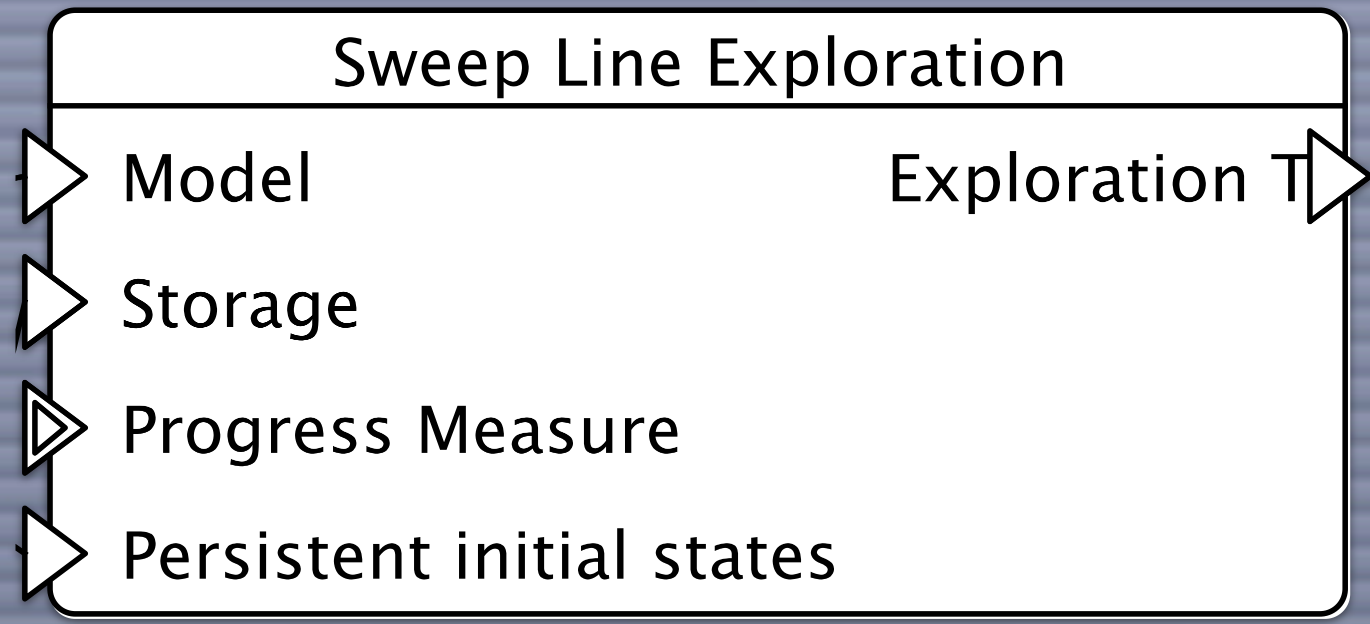
Example:
Sweep-line Exploration




```
(*
 *  default sweep line exploration
 *)
functor SweepLineExploration(
val markInitStatesAsPersistent : bool
structure Storage              : REMOVE_STORAGE
structure Model                : MODEL
structure Measure              : PROGRESS_MEASURE
   where type state = Model.state * Model.event list
sharing type Model.state = Storage.item
): SWEEP_LINE_EXPLORATION = struct
```

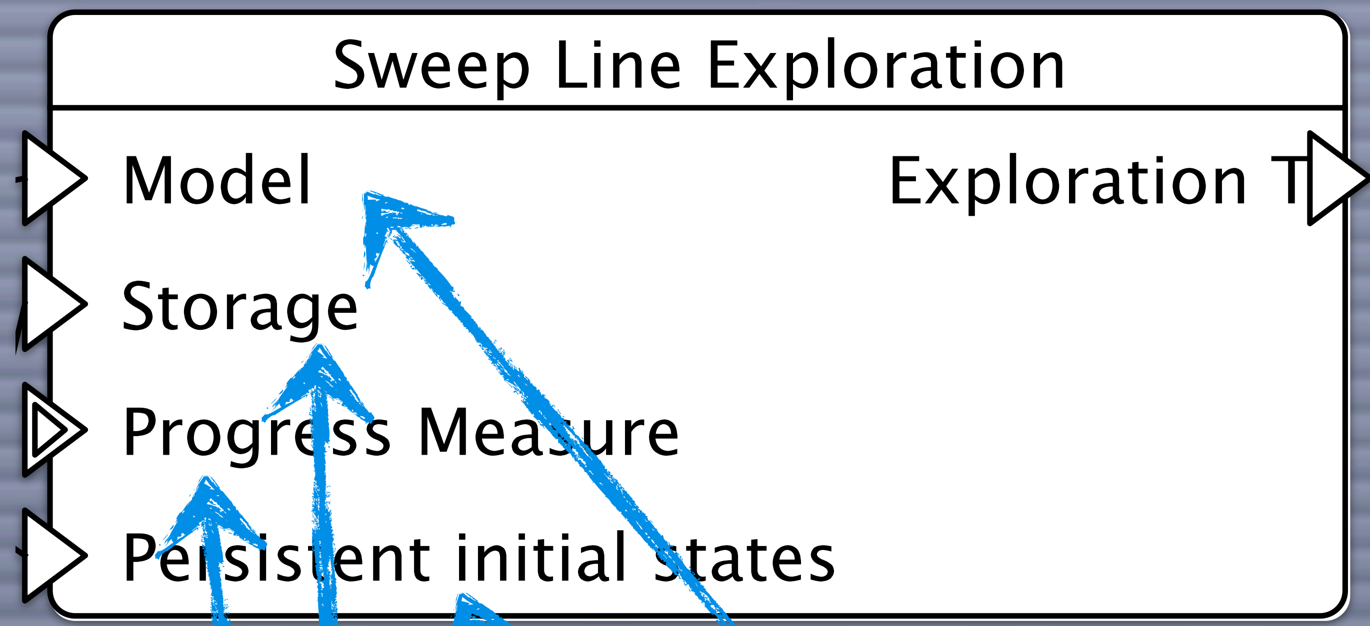
Example:
Sweep-line Exploration

Tasks



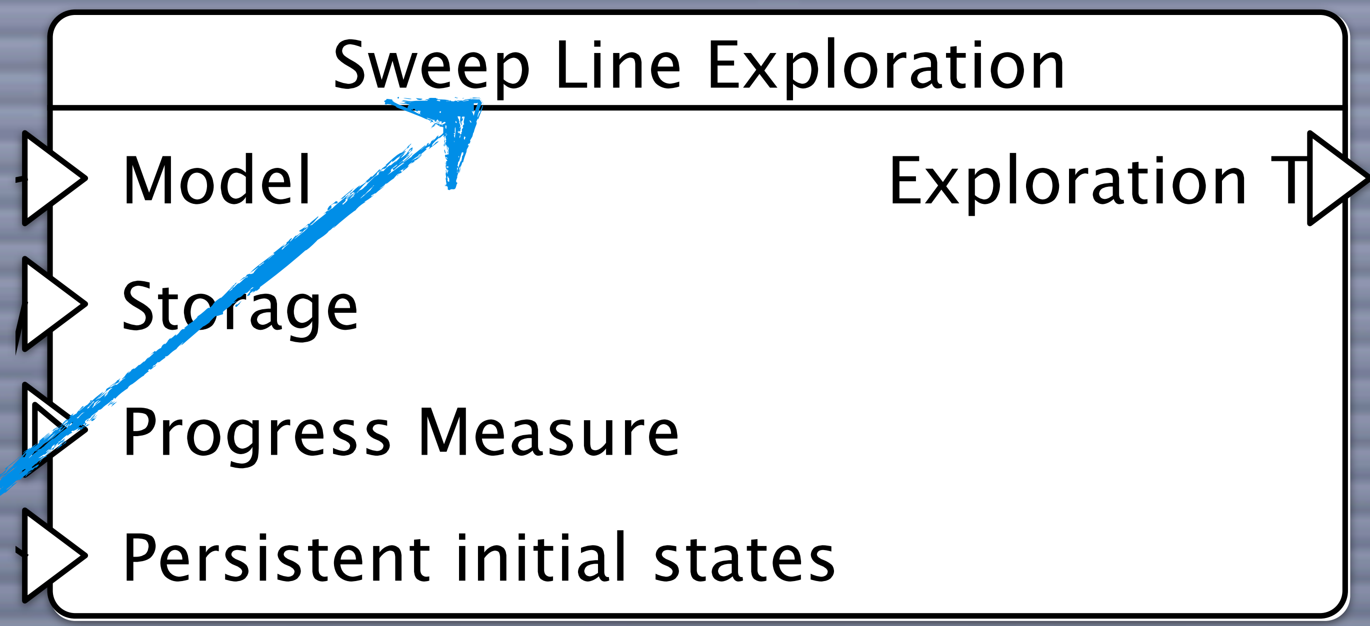

```java
public ValueDescription[] getParameters() {
    return new ValueDescription[] { ExecutionFactory.eINSTANCE.createValueDescription("Model", MLModel.class),
            ExecutionFactory.eINSTANCE.createValueDescription("Storage", MLRemoveStorage.class),
            ExecutionFactory.eINSTANCE.createValueDescription("Progress Measure", MLProgressMeasure.class),
            ExecutionFactory.eINSTANCE.createValueDescription("Persistent initial states", Boolean.class) };
}
```

Tasks




```
public ValueDescription[] getParameters() {
    return new ValueDescription[] { ExecutionFactory.eINSTANCE.createValueDescription("Model", MLModel.class),
            ExecutionFactory.eINSTANCE.createValueDescription("Storage", MLRemoveStorage.class),
            ExecutionFactory.eINSTANCE.createValueDescription("Progress Measure", MLProgressMeasure.class),
            ExecutionFactory.eINSTANCE.createValueDescription("Persistent initial states", Boolean.class) };
}
```

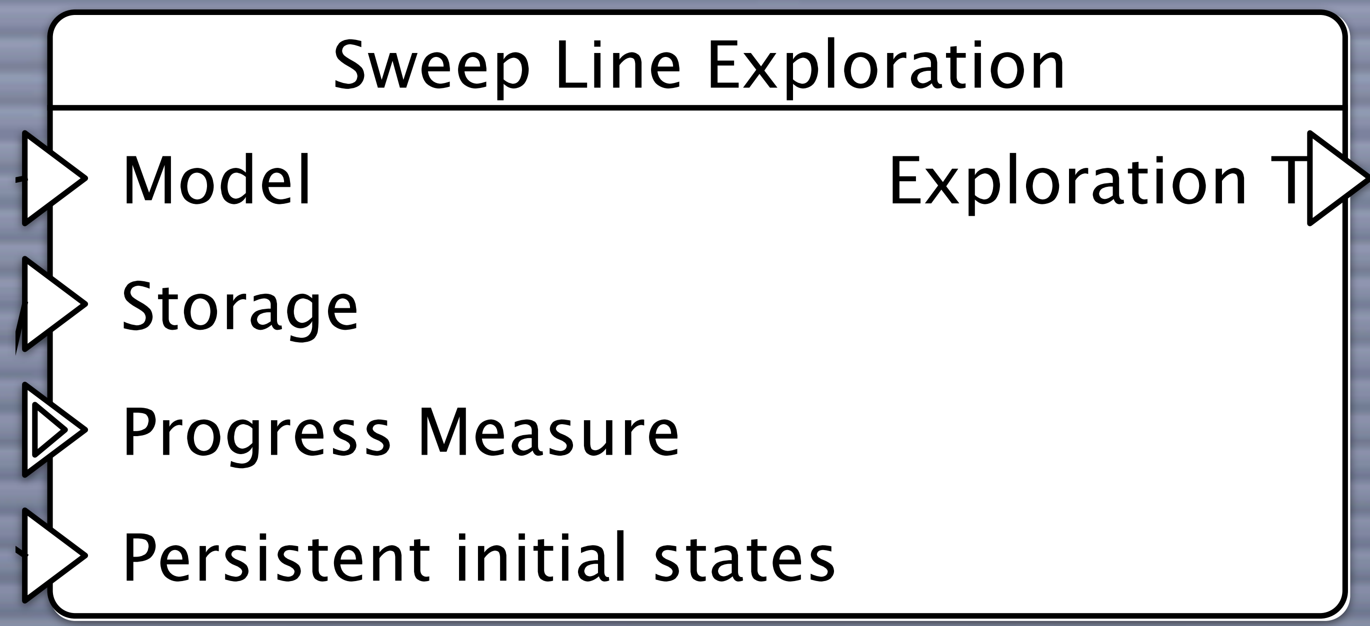
Tasks




```java
public ValueDescription[] getParameters() {
    return new ValueDescription[] { ExecutionFactory.eINSTANCE.createValueDescription("Model", MLModel.class),
            ExecutionFactory.eINSTANCE.createValueDescription("Storage", MLRemoveStorage.class),
            ExecutionFactory.eINSTANCE.createValueDescription("Progress Measure", MLProgressMeasure.class),
            ExecutionFactory.eINSTANCE.createValueDescription("Persistent initial states", Boolean.class) };
}
```

```java
public String getName() {
    return "Sweep Line Exploration";
}
```

Tasks

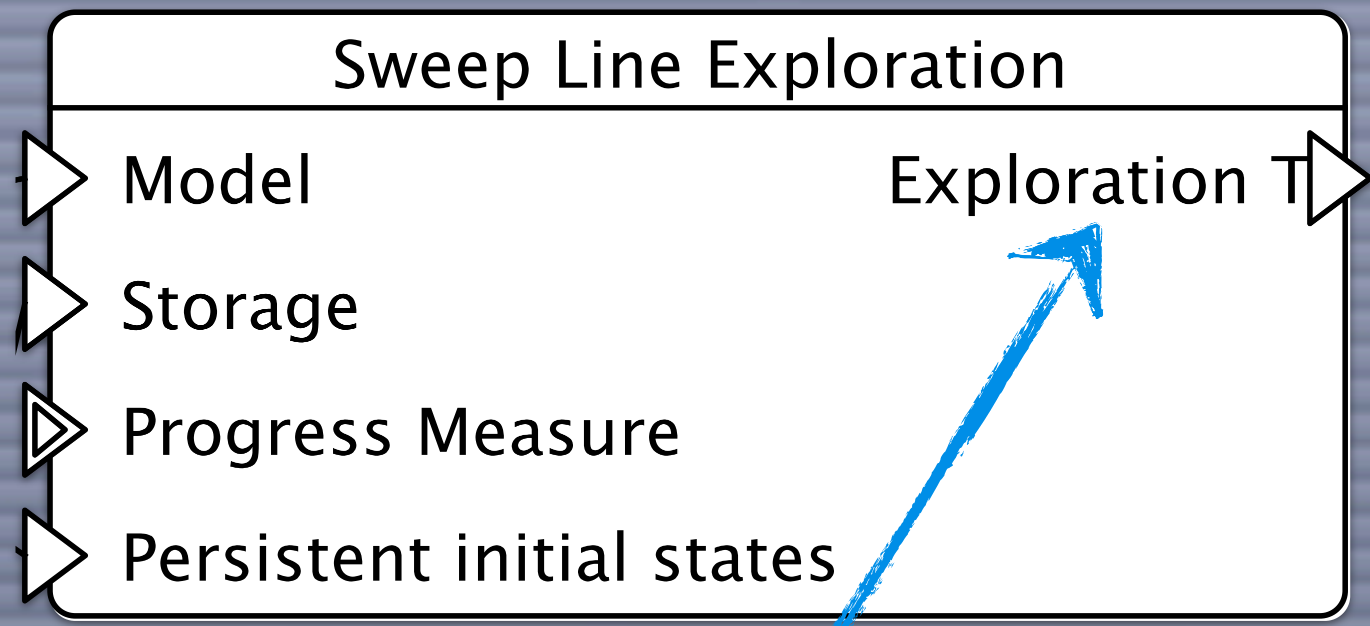



```
public ValueDescription[] getParameters() {
    return new ValueDescription[] { ExecutionFactory.eINSTANCE.createValueDescription("Model", MLModel.class),
            ExecutionFactory.eINSTANCE.createValueDescription("Storage", MLRemoveStorage.class),
            ExecutionFactory.eINSTANCE.createValueDescription("Progress Measure", MLProgressMeasure.class),
            ExecutionFactory.eINSTANCE.createValueDescription("Persistent initial states", Boolean.class) };
}
```

```
public String getName() {
    return "Sweep Line Exploration";
}
```

```
public String getFunctorName() {
    return "SweepLineExploration";
}
```

Tasks



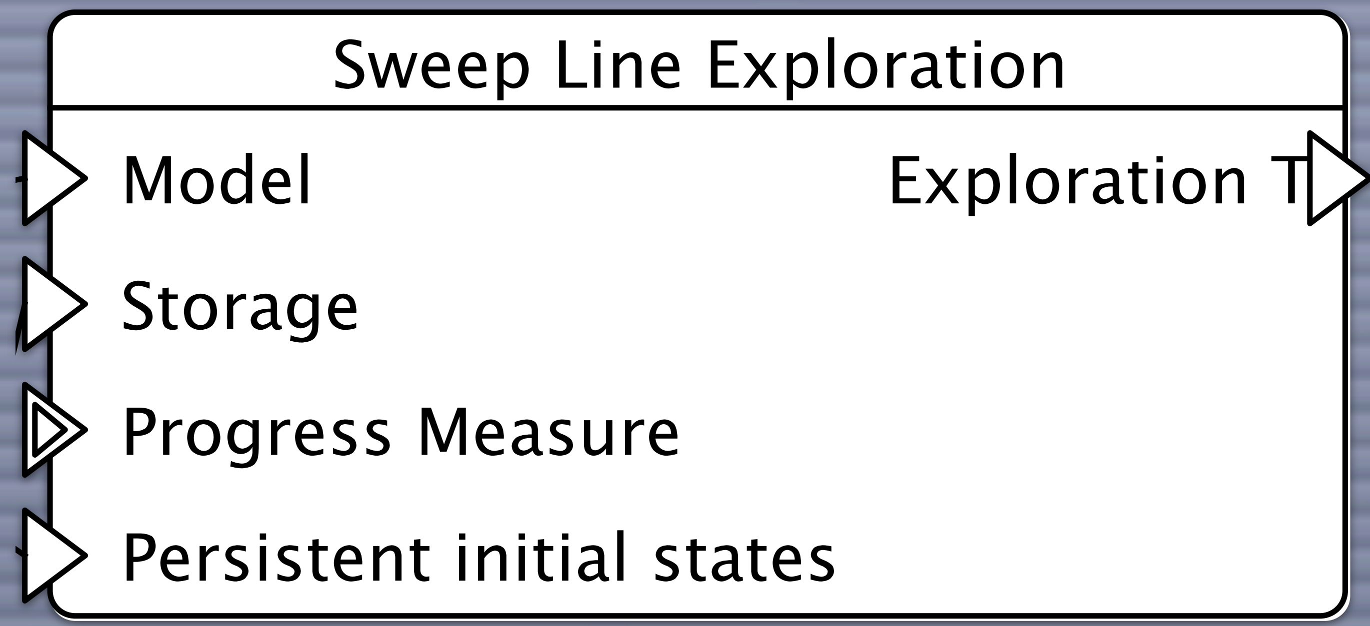

```
public ValueDescription[] getParameters() {
    return new ValueDescription[] { ExecutionFactory.eINSTANCE.createValueDescription("Model", MLModel.class),
            ExecutionFactory.eINSTANCE.createValueDescription("Storage", MLRemoveStorage.class),
            ExecutionFactory.eINSTANCE.createValueDescription("Progress Measure", MLProgressMeasure.class),
            ExecutionFactory.eINSTANCE.createValueDescription("Persistent initial states", Boolean.class) };
}
```

```
public String getName() {
    return "Sweep Line Exploration";
}
```

```
public String getFunctorName() {
    return "SweepLineExploration";
}
```

```
public ValueDescription getReturnType() {
    return ExecutionFactory.eINSTANCE.createValueDescription("Exploration T", MLTraceExploration.class);
}
```

Tasks




```java
public ValueDescription[] getParameters() {
    return new ValueDescription[] { ExecutionFactory.eINSTANCE.createValueDescription("Model", MLModel.class),
            ExecutionFactory.eINSTANCE.createValueDescription("Storage", MLRemoveStorage.class),
            ExecutionFactory.eINSTANCE.createValueDescription("Progress Measure", MLProgressMeasure.class),
            ExecutionFactory.eINSTANCE.createValueDescription("Persistent initial states", Boolean.class) };
}
```

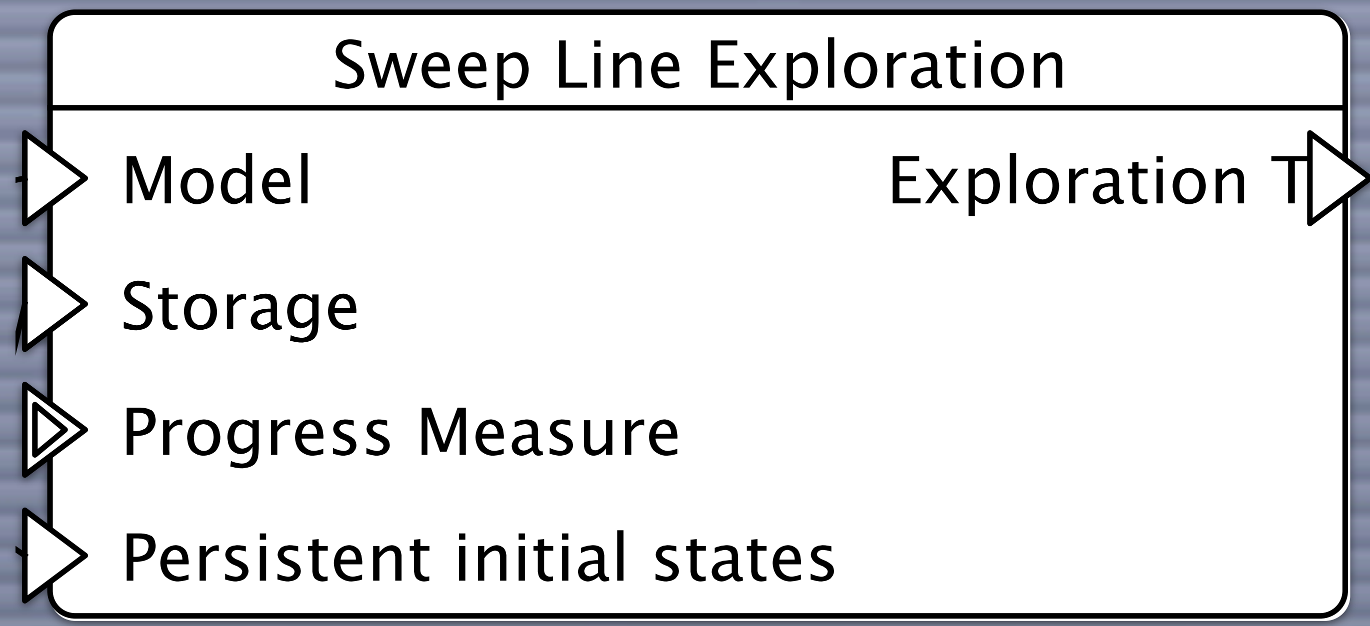
```java
public String getName() {
    return "Sweep Line Exploration";
}
```

```java
public String getFunctorName() {
    return "SweepLineExploration";
}
```

```java
public ValueDescription getReturnType() {
    return ExecutionFactory.eINSTANCE.createValueDescription("Exploration T", MLTraceExploration.class);
}
```

```java
public <V, E> MLTraceExploration executeTask(final MLModel<V, E> model, final MLRemoveStorage storage,
        final MLProgressMeasure<V, E> progressMeasure, final Boolean setInitAsPersistent) throws Exception {
    final MLTraceExploration result = new MLTraceExploration(model.getSimulator(), null);
    model.getSimulator().evaluate(
            result.getDeclaration() + " = IntermediateStatsExploration"
            + "(structure JavaExecute = JavaExecute"
            + " structure Exploration = StoppableExploration"
            + "(structure JavaExecute = JavaExecute"
            + " structure Exploration = " + getFunctorName()
            + "(structure Model = " + model.getStructureName()
            + " structure Storage = " + storage.getStructureName()
            + " structure Measure = " + progressMeasure.getStructureName()
            + " val markInitStatesAsPersistent = " + setInitAsPersistent + ")))");
```

Tasks




```java
public ValueDescription[] getParameters() {
    return new ValueDescription[] { ExecutionFactory.eINSTANCE.createValueDescription("Model", MLModel.class),
            ExecutionFactory.eINSTANCE.createValueDescription("Storage", MLRemoveStorage.class),
            ExecutionFactory.eINSTANCE.createValueDescription("Progress Measure", MLProgressMeasure.class),
            ExecutionFactory.eINSTANCE.createValueDescription("Persistent initial states", Boolean.class) };
}
```

```java
public String getName() {
    return "Sweep Line Exploration";
}
```

```java
public String getFunctorName() {
    return "SweepLineExploration";
}
```

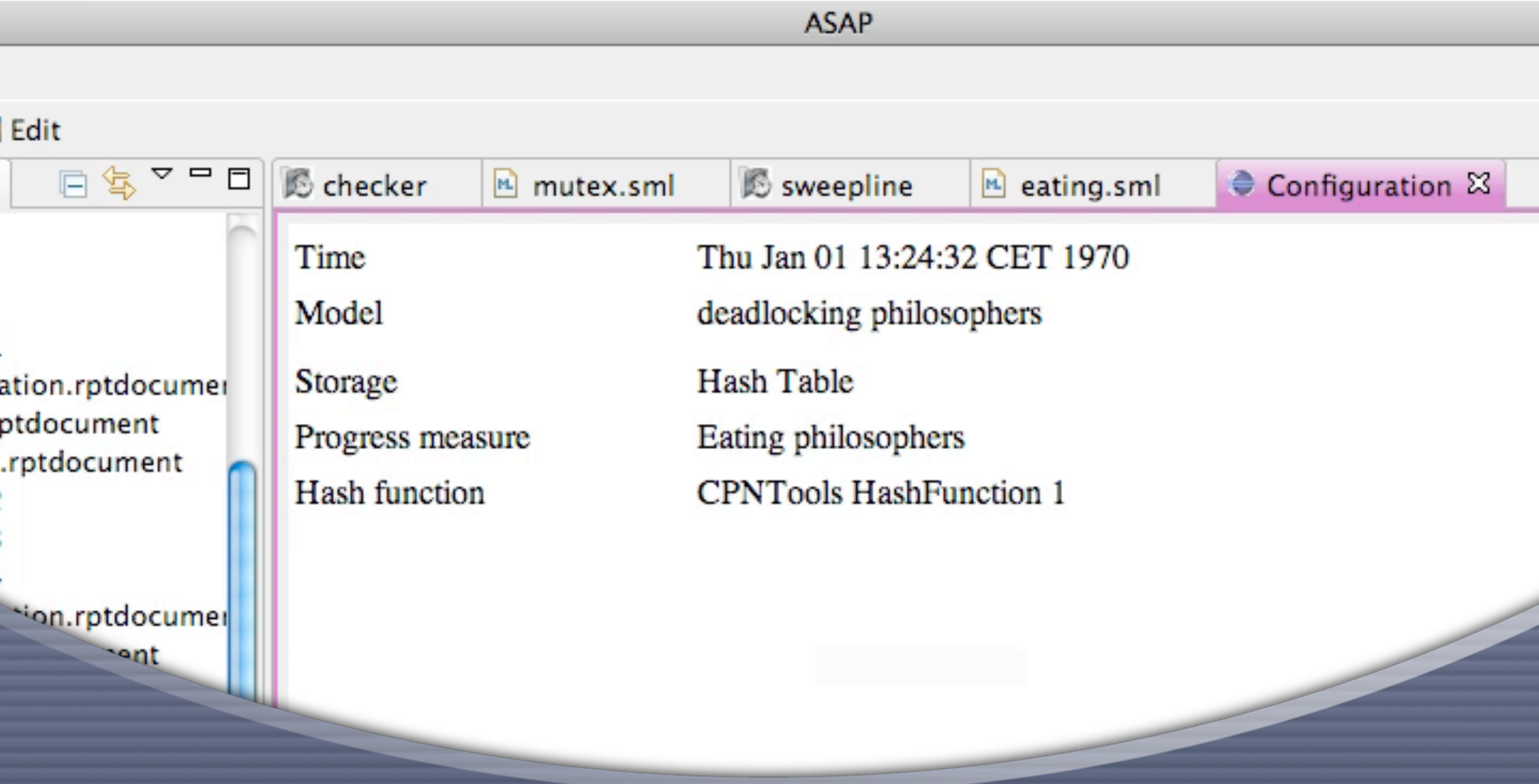
```java
public <V, E> MLTraceExploration executeTask(final MLModel<V, E> model, final MLRemoveStorage storage,
        final MLProgressMeasure<V, E> progressMeasure, final Boolean setInitAsPersistent) throws Exception {
    final MLTraceExploration result = new MLTraceExploration(model.getSimulator(), null);
    model.getSimulator().evaluate(
            result.getDeclaration() + " = IntermediateStatsExploration"
            + "(structure JavaExecute = JavaExecute"
            + " structure Exploration = StoppableExploration"
            + "(structure JavaExecute = JavaExecute"
            + " structure Exploration = " + getFunctorName()
            + "(structure Model = " + model.getStructureName()
            + " structure Storage = " + storage.getStructureName()
            + " structure Measure = " + progressMeasure.getStructureName()
            + " val markInitStatesAsPersistent = " + setInitAsPersistent + ")))");
    result.addChild(model, storage, progressMeasure);
    return result;
}
```

Reporting in ASAP

- ASAP automatically gathers information about every execution in a database (either an in-memory database or MySQL)
- The standard report is created using a standard report generating tool (BIRT)
  - ASAP is able to automatically assemble a report based on report fragments

Adding New Entries to the Report

- Add a new value entry to the database
- Make sure the value is gathered during execution
- (Make a new report item model and report item presentation)
- Make a fragment showing your value

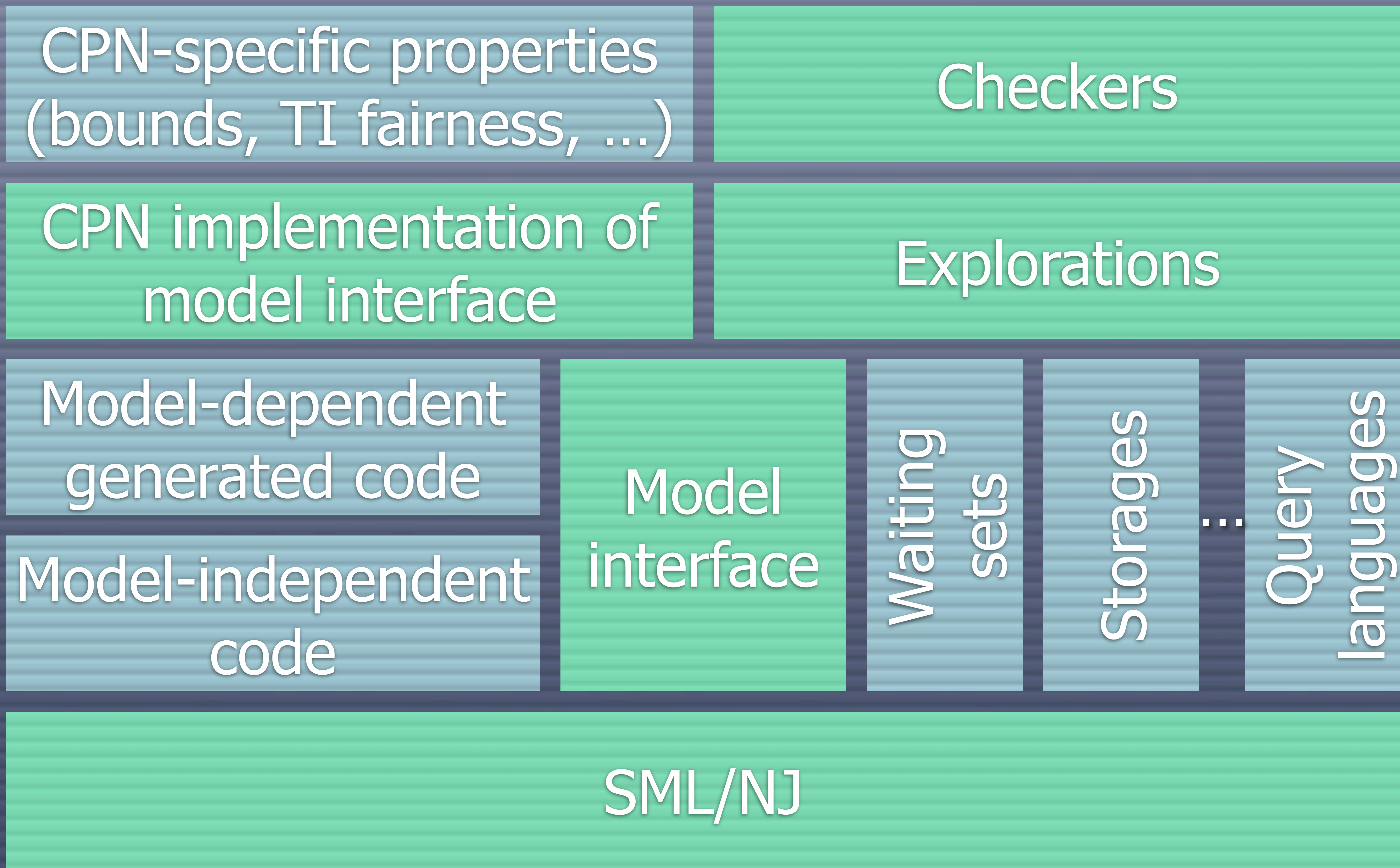


Example: Progress Measure

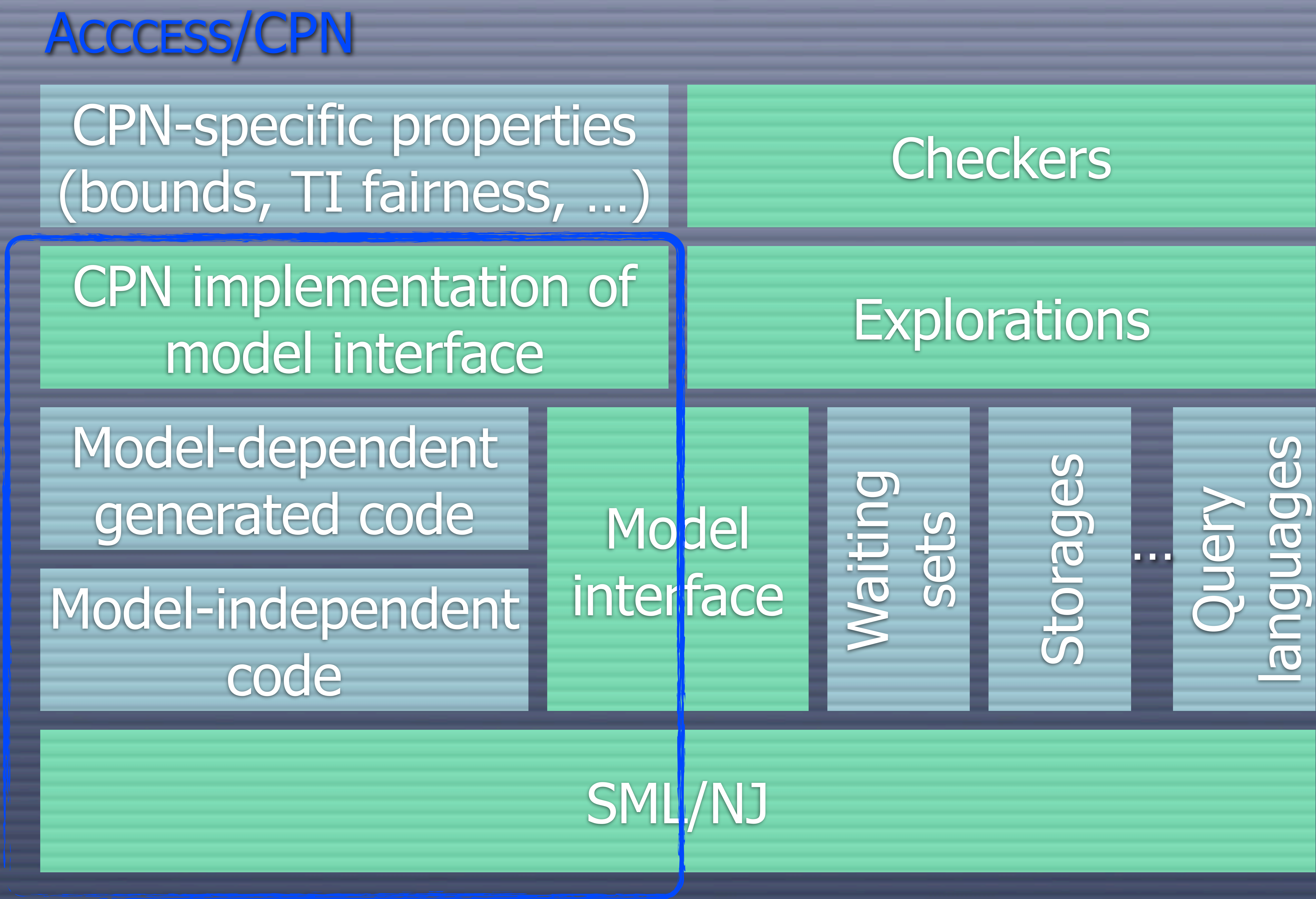
ACCESS/CPN

- We have isolated the library used by ASAP to load CPN models as well as the interface used by the state space engine
- These two parts together are distributed under the name ACCESS/CPN

ACCESS/CPN

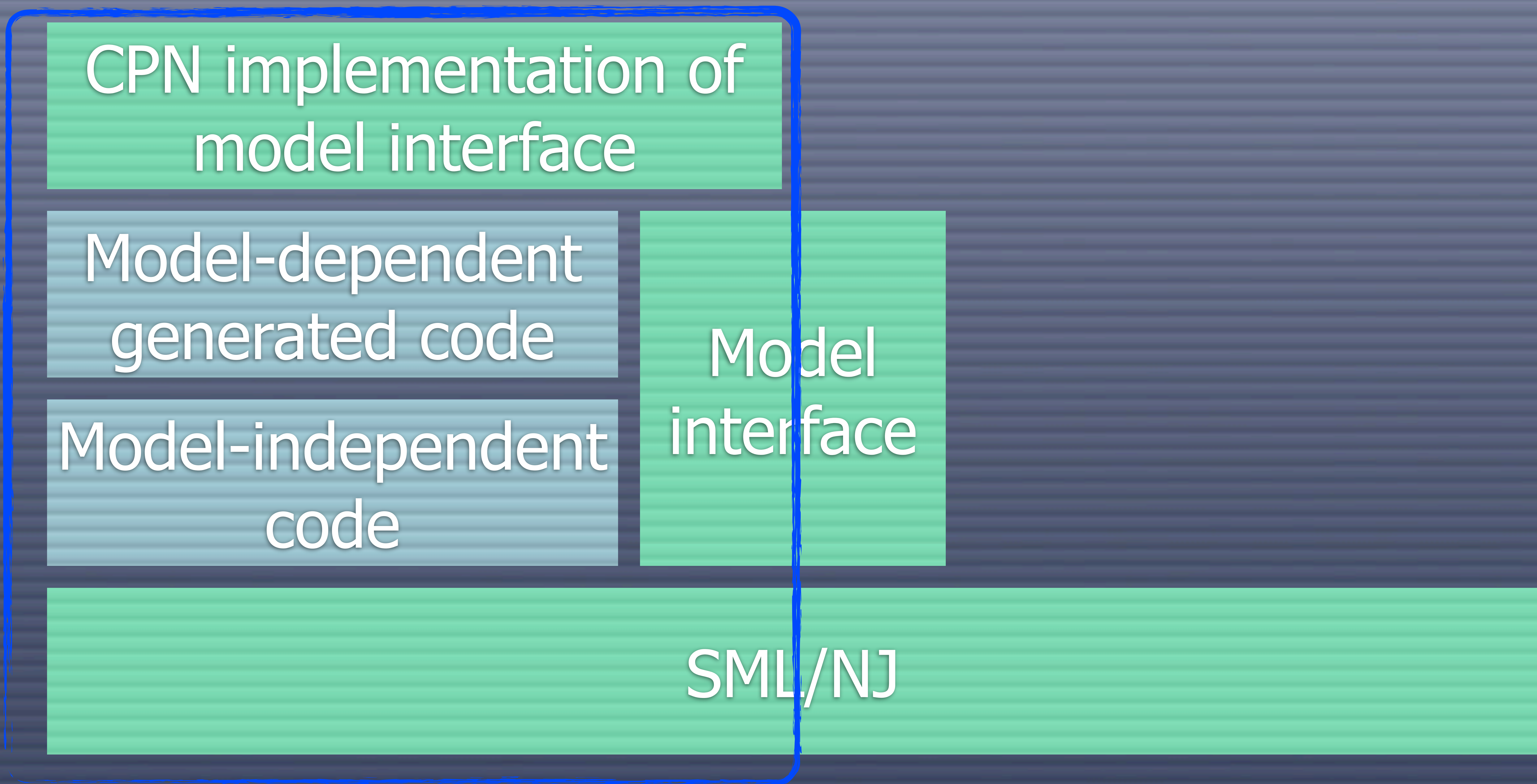

CPN-specific properties (bounds, TI fairness, ...)
Checkers
CPN implementation of model interface
Explorations
Model-dependent generated code
Model-independent code
Model interface
Waiting sets
Storages
...
Query languages
SML/NJ

ACCESS/CPN


ACCCESS/CPN
CPN-specific properties (bounds, TI fairness, ...)
Checkers
CPN implementation of model interface
Explorations
Model-dependent generated code
Model-independent code
Model interface
Waiting sets
Storages
...
Query languages
SML/NJ

Access/CPN

Acccess/CPN



Access/CPN Features

- With Access/CPN you can:
  - Load models from CPN Tools
  - Simulate models programmatically (both automatic and "manual")
  - Inspect and change state
  - Evaluate SML code
  - Build a state space tool :-)

Access/CPN Uses

- ASAP
- Integration into ProM (R. Mans & M. Netjes)
- Cosimulation of SystemC and CP-nets
- Various master's theses
- ...

Benchmarking

- Benchmarking comes in (at least) 3 variants
  - Improve code during development
  - Compare methods
  - Improve models
- We will focus on the two former

Measurements

- We can measure three things
  - Time spent in a function
  - Times a function is called
  - Amount of memory used

Comparing Methods

- We basically want to know
  - How long did it take to run the entire exploration
  - How much memory did we use

Improving Implementation

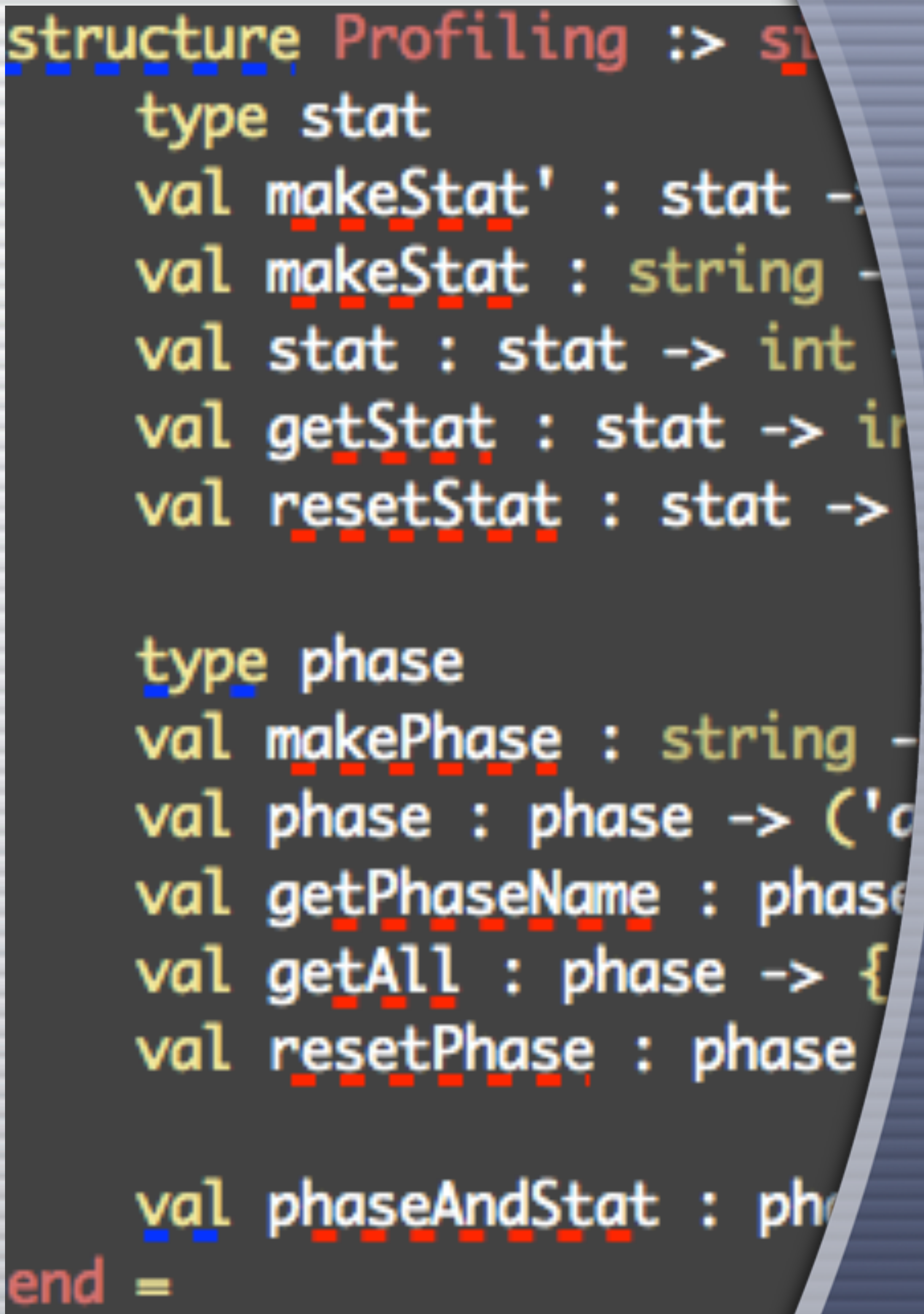
- We want to know
  - What are the hot-spots (time and memory)
  - How does one implementation compare to another
- This also encompasses our requirements for comparing methods

SML/NJ Profiler

- Basically non-existing
- There is an infrastructure allowing us to instrument code
  - Makes it difficult to control granularity

Custom Structure

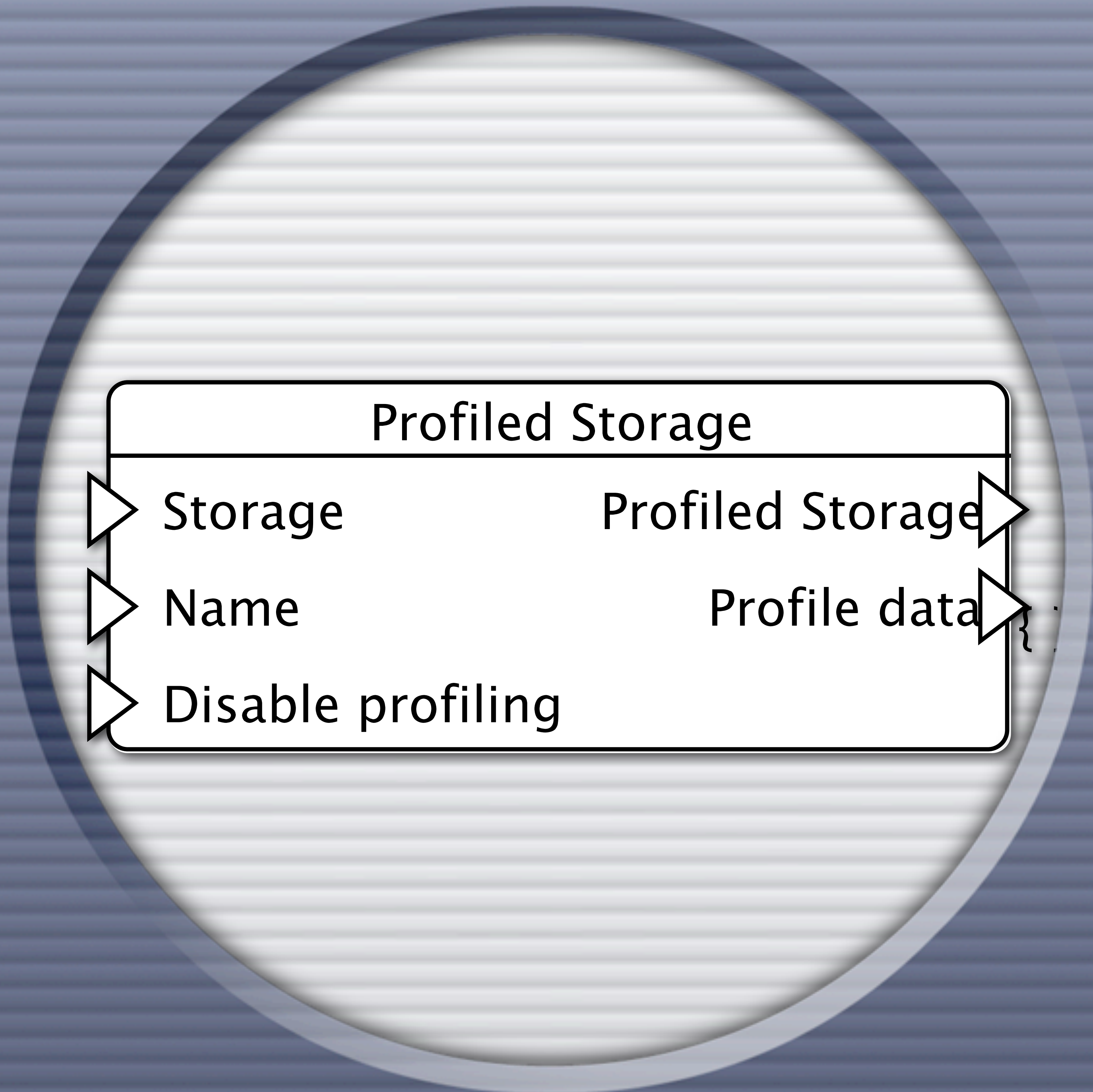
- Based on SML/NJ's own internal compilation framework
- Includes a hook in the compiler we have made ourselves

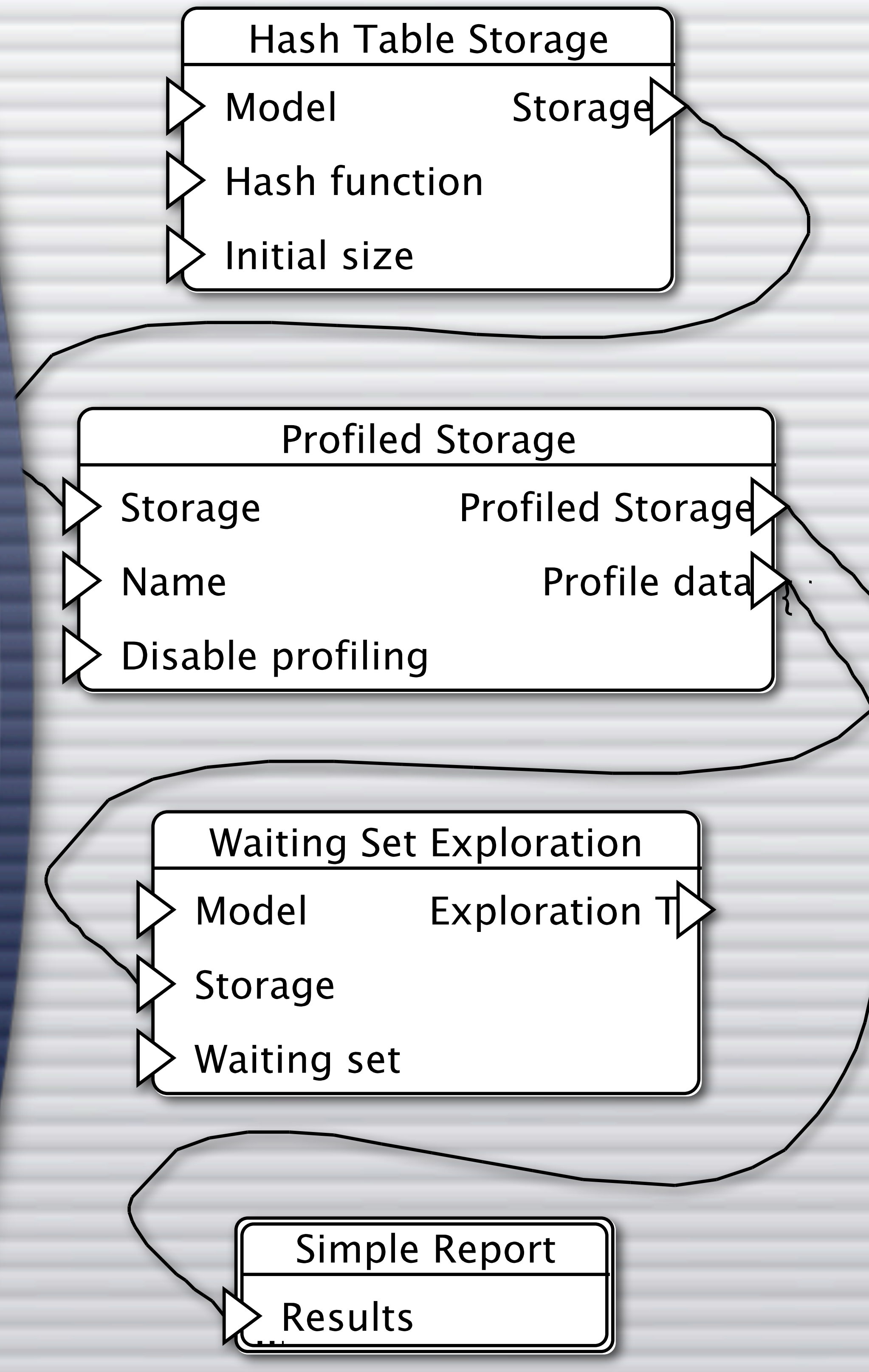


Time Profiling

Keep track of counts (stat)

Measure total time for a single function execution (phase)


Profiled Storage
Storage
Profiled Storage
Name
Profile data
Disable profiling


Hash Table Storage
Model
Storage
Hash function
Initial size
Profiled Storage
Storage
Profiled Storage
Name
Profile data
Disable profiling
Waiting Set Exploration
Model
Exploration T
Storage
Waiting set
Simple Report
Results

Strategy

```
structure Profiling :> sig
    type stat
    val makeStat' : stat -> string -> stat
    val makeStat : string -> stat
    val stat : stat -> int -> unit
    val getStat : stat -> int
    val resetStat : stat -> unit

    type phase
    val makePhase : string -> phase
    val phase : phase -> ('a -> 'b) -> ('a -> 'b)
```

Stats

Create new stat

```
structure Profiling :> sig
    type stat
    val makeStat' : stat -> string -> stat
    val makeStat : string -> stat
    val stat : stat -> int -> unit
    val getStat : stat -> int
    val resetStat : stat -> unit

    type phase
    val makePhase : string -> phase
    val phase : phase -> ('a -> 'b) -> ('a -> 'b)
      tPhaseName : phase -> string
             hase -> { usr : real, sys: real
```

Stats

Stats

```
structure Profiling :> sig
    type stat
    val makeStat' : stat -> string -> stat
    val makeStat : string -> stat
    val stat : stat -> int -> unit
    val getStat : stat -> int
    val resetStat : stat -> unit

    type phase
    val makePhase : string -> phase
    val phase : phase -> ('a -> 'b) -> ('a -> 'b)
```

Create new stat

Increment a stat

```
structure Profiling :> sig
    type stat
    val makeStat' : stat -> string -> stat
    val makeStat : string -> stat
    val stat : stat -> int -> unit
    val getStat : stat -> int
    val resetStat : stat -> unit

    type phase
    val makePhase : string -> phase
    val phase : phase -> ('a -> 'b) -> ('a -> 'b)
```

Create new stat

Increment a stat

Get value of stat

Stats

Stats

```
structure Profiling :> sig
    type stat
    val makeStat' : stat -> string -> stat
    val makeStat : string -> stat
    val stat : stat -> int -> unit
    val getStat : stat -> int
    val resetStat : stat -> unit
```

Create new stat

Increment a stat

Get value of stat

Reset stat

Stats

```
structure Profiling :> sig
    type stat
    val makeStat' : stat -> string -> stat
    val makeStat : string -> stat
    val stat : stat -> int -> unit
    val getStat : stat -> int
    val resetStat : stat -> unit
```

Create new stat

Increment a stat

Get value of stat

Reset stat

Create stat as a substat of another

```
functor ProfiledStorage (
structure Subject : STORAGE
val name: string
) : sig
    include STORAGE
    include PROFILING

    val storageStat : stat
    val storagePhase : phase
end =
struct
    structure Profiling = ProfilingHelp(val name = name)
    open Profiling
    open Subject
```

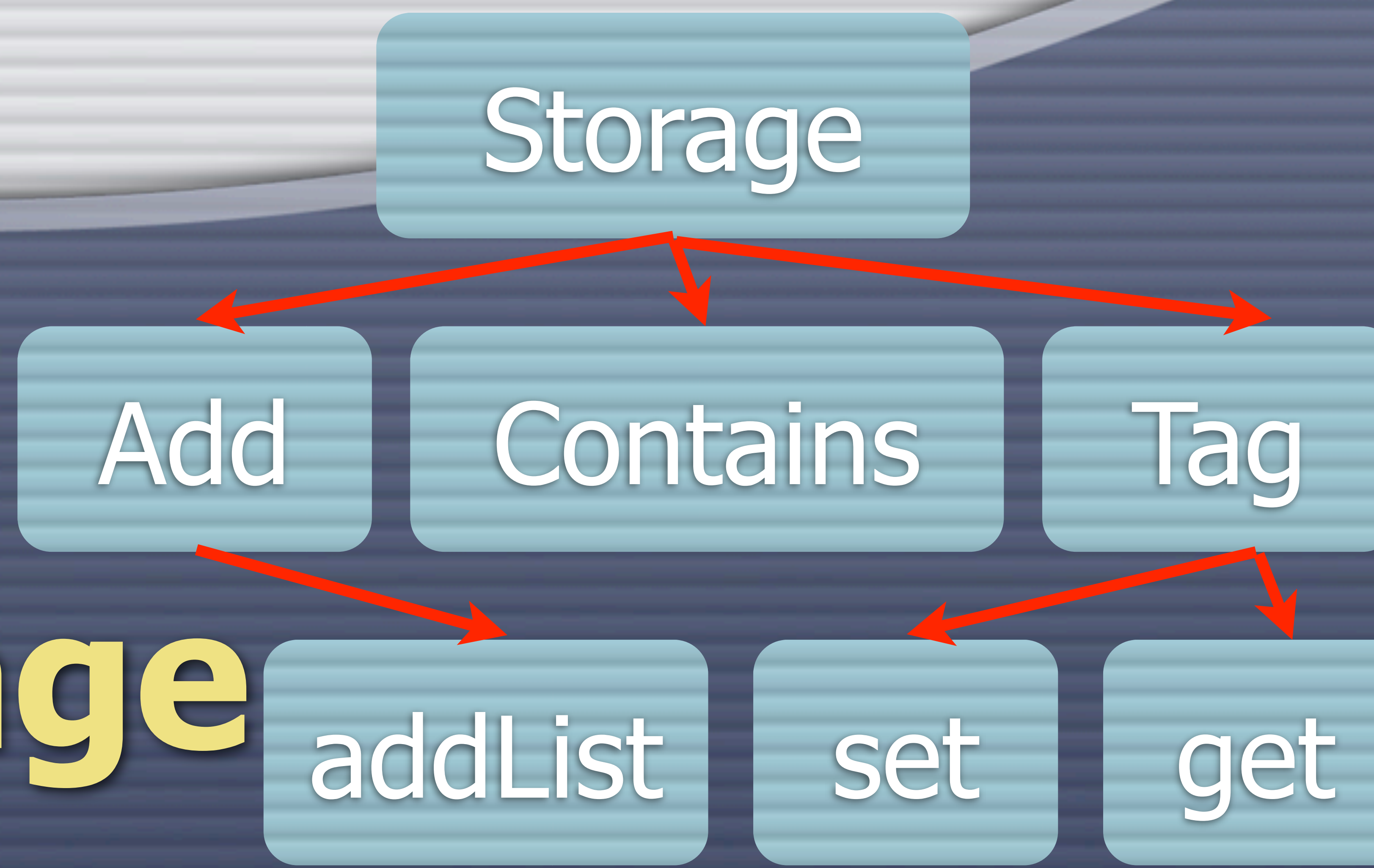
Example: Profiled Storage

```
end =
struct
    structure Profiling = ProfilingHelp(val name = name)
    open Profiling
    open Subject

    val storageStat = makeStat "Storage"
    val addStat = makeStat' storageStat "Storage.add"
    val addListStat = makeStat' addStat "Storage.addList"
    val containsStat = makeStat' storageStat "Storage.contains"
    val tagStat = makeStat' storageStat "Storage.tag"
    val getTagStat = makeStat' tagStat "Storage.getTag"
    val setTagStat = makeStat' tagStat "Storage.setTag"
```

Example: Profiled Storage

```
end =
struct
    structure Profiling = ProfilingHelp(val name = name)
    open Profiling
    open Subject

    val storageStat = makeStat "Storage"
    val addStat = makeStat' storageStat "Storage.add"
    val addListStat = makeStat' addStat "Storage.addList"
    val containsStat = makeStat' storageStat "Storage.contains"
    val tagStat = makeStat' storageStat "Storage.tag"
    val getTagStat = makeStat' tagStat "Storage.getTag"
    val setTagStat = makeStat' tagStat "Storage.setTag"
```

Example: Profiled Storage



```
val stat : stat -> int -> unit
val getStat : stat -> int
val resetStat : stat -> unit
```

```
type phase
val makePhase : string -> phase
val phase : phase -> ('a -> 'b) -> ('a -> 'b)
val getPhaseName : phase -> string
val getAll : phase -> { usr : real, sys: real, gc: real }
val resetPhase : phase -> unit

val phaseAndStat : phase -> stat -> ('a -> 'b) -> ('a -> 'b)
```

Phases

Phases

Create a new phase

```
type phase
val makePhase : string -> phase
val phase : phase -> ('a -> 'b) -> ('a -> 'b)
val getPhaseName : phase -> string
val getAll : phase -> { usr : real, sys: real, gc: real }
val resetPhase : phase -> unit

val phaseAndStat : phase -> stat -> ('a -> 'b) -> ('a -> 'b)
```

Phases

Create a new phase

Get name, timings, and reset a phase

```
type phase
val makePhase : string -> phase
val phase : phase -> ('a -> 'b) -> ('a -> 'b)
val getPhaseName : phase -> string
val getAll : phase -> { usr : real, sys: real, gc: real }
val resetPhase : phase -> unit

val phaseAndStat : phase -> stat -> ('a -> 'b) -> ('a -> 'b)
```

Phases

```
type phase
val makePhase : string -> phase
val phase : phase -> ('a -> 'b) -> ('a -> 'b)
val getPhaseName : phase -> string
val getAll : phase -> { usr : real, sys: real, gc: real }
val resetPhase : phase -> unit

val phaseAndStat : phase -> stat -> ('a -> 'b) -> ('a -> 'b)
```

Create a new phase

Get name, timings, and reset a phase

Given a function, create a new function which starts the timer for phase on invocation and stops on termination

Phases

```
type phase
val makePhase : string -> phase
val phase : phase -> ('a -> 'b) -> ('a -> 'b)
val getPhaseName : phase -> string
val getAll : phase -> { usr : real, sys: real, gc: real }
val resetPhase : phase -> unit

val phaseAndStat : phase -> stat -> ('a -> 'b) -> ('a -> 'b)
```

Create a new phase

Get name, timings, and reset a phase

Given a function, create a new function which starts the timer for phase on invocation and stops on termination

Start phase and increment stat

```
    fun emptyStorage a b = phaseAndStat storagePhase storageStat Subject.emptyStorage a b
    fun add a = phaseAndStat storagePhase addStat Subject.add a
    fun addList a = phaseAndStat storagePhase addListStat Subject.addList a
    fun contains a = phaseAndStat storagePhase containsStat Subject.contains a
    fun contains' a = phaseAndStat storagePhase containsStat Subject.contains' a
    fun isEmpty a = phaseAndStat storagePhase storageStat Subject.isEmpty a
    fun numItems a = phaseAndStat storagePhase storageStat Subject.numItems a
    fun getTag a = phaseAndStat storagePhase getTagStat Subject.getTag a
    fun getTag' a = phaseAndStat storagePhase getTagStat Subject.getTag' a
    fun setTag a = phaseAndStat storagePhase setTagStat Subject.setTag a
    fun setTag' a = phaseAndStat storagePhase setTagStat Subject.setTag' a
end
```

Example: Profiled Storage

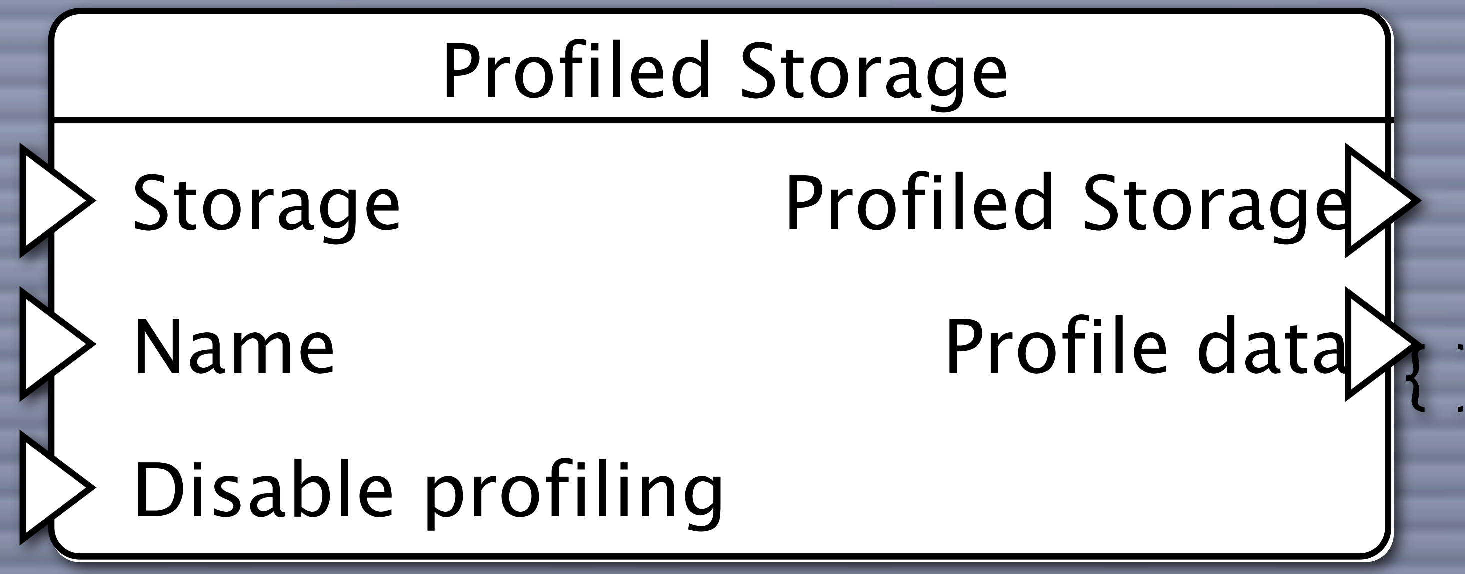
Memory Profiling

- Difficult as SML is functional
- Strategy: Sample memory "once in a while"
- We need to perform a full garbage collection, as everything is allocated on the heap ⇒ we cannot do this too often

Two Methodologies

- Register a signal handler doing everything
  - Completely transparent to programmer
- Use same strategy as for time (wrap base structures)
  - Greater control

We prefer the latter approach
1) for consistency and
2) for performance reasons



Collecting Results

- We simply connect the “Profile data” output port to the standard report
- Results automatically appear in the “Statistics” section with same control as for the rest of the gathered information
- Using the “Disable profiling” port, we can set up tasks for user control

Evaluation

- Quick-and-dirty solution that works
- Easy to extend – but requires a bit of manual work
- Provides a nice high-level view
- Needs work to provide low-level view

Short-term New Features

- Even faster analysis
- Support for timed CPN models
- Standard report
- On-line drawing of graph

Even Faster Analysis

- CPN Tools and ASAP currently uses SML/NJ
- In the labs we have ported parts of the simulator to SML compiler, MLton
- MLton can cut runtime down to 50% - 70%
- MLton does **not** allow dynamic code generation and you must ask all questions in advance
  - We seek a reasonable way to use SML/NJ in the initial phases (interactive investigation) and MLton for hardcore number crunching in later phases

Even Faster Analysis

- MLton provides a much better profiler than SML/NJ, leading to new insights
  - We use 30% - 50% of the time generating random numbers to choose bindings fairly
  - After eliminating the above, we use 20% - 30% of the time converting between multi-set representations

Ev s

When doing state space analysis, we need to investigate **all** bindings, so we do not care about a fair order

- filer than SML/NJ, leading to new insights
  - We use 30% - 50% of the time generating random numbers to choose bindings fairly
  - After eliminating the above, we use 20% - 30% of the time converting between multi-set representations

Even Faster Analysis

- MLton provides a much better profiler than SML/NJ, leading to new insights
  - We use 30% - 50% of the time generating random numbers to choose bindings fairly
  - After eliminating the above, we use 20% - 30% of the time converting between multi-set representations

Even Faster Analysis

- MLt... profiler than SML... s
- W... ne ge... to choose bi... fully
- After eliminating the above, we use 20% - 30% of the time converting between multi-set representations

Most likely it is possible to just use the one of the state space tool when not doing simulation

Even Faster Analysis

- MLton provides a much better profiler than SML/NJ, leading to new insights
  - We use 30% - 50% of the time generating random numbers to choose bindings fairly
  - After eliminating the above, we use 20% - 30% of the time converting between multi-set representations

Even Faster Analysis

- MLton provides a much better profiler than SML/NJ, leading to new insights
  - We use 30% - 50% of the time generating random numbers to choose bindings fairly
  - After eliminati
    - 30% of the t
    multi-set repre

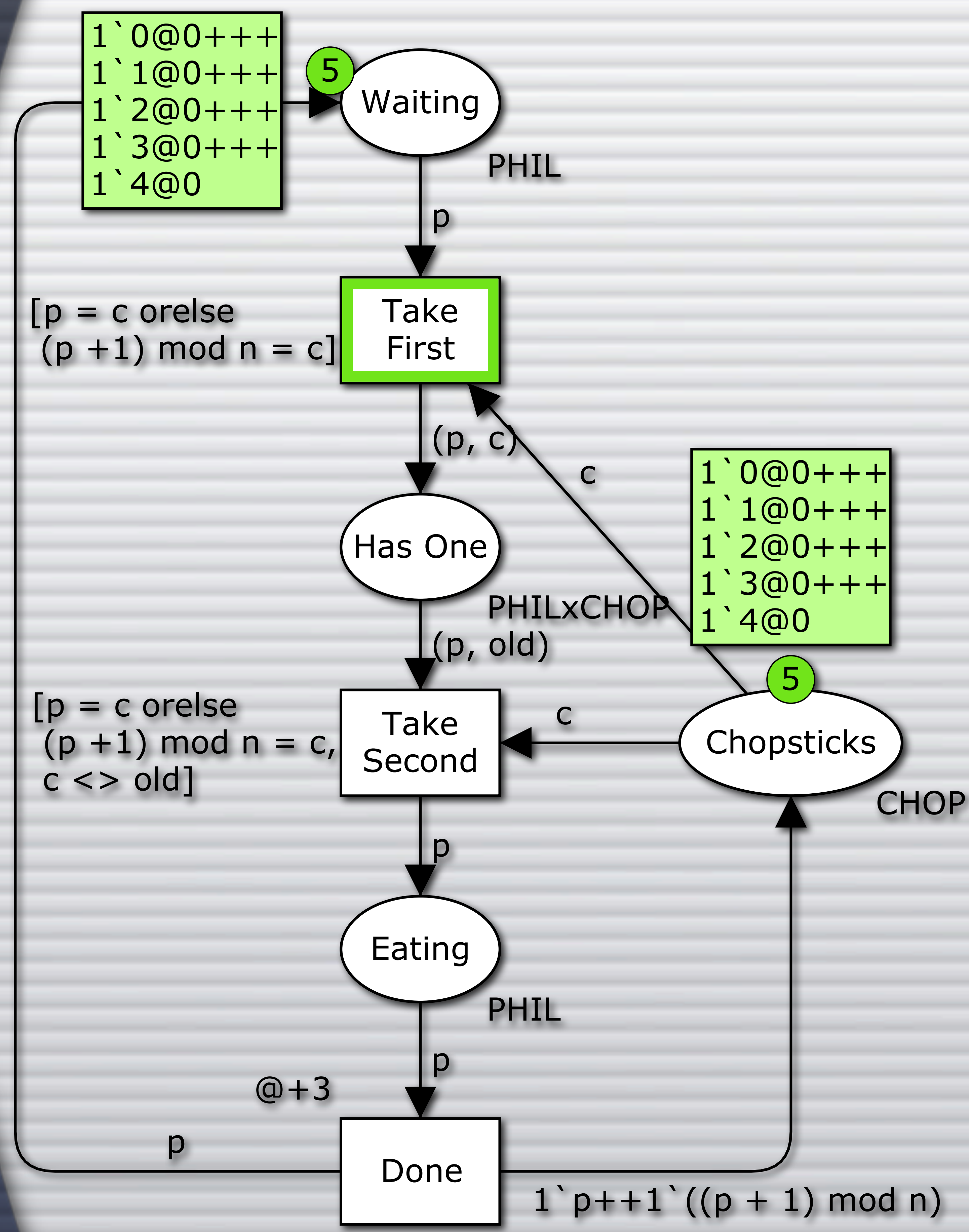
Using all of this information has allowed us to speed the tool up by a factor 6-8 (compared to the speed we have seen today)

Support for Timed Models

- We need to add time information to the state descriptor (easy)
- We need to make all the utility functions understand the time value (hash-functions, serializations, etc.)
(easy but mind numbingly dull)
- We need to implement time equivalence (easy)

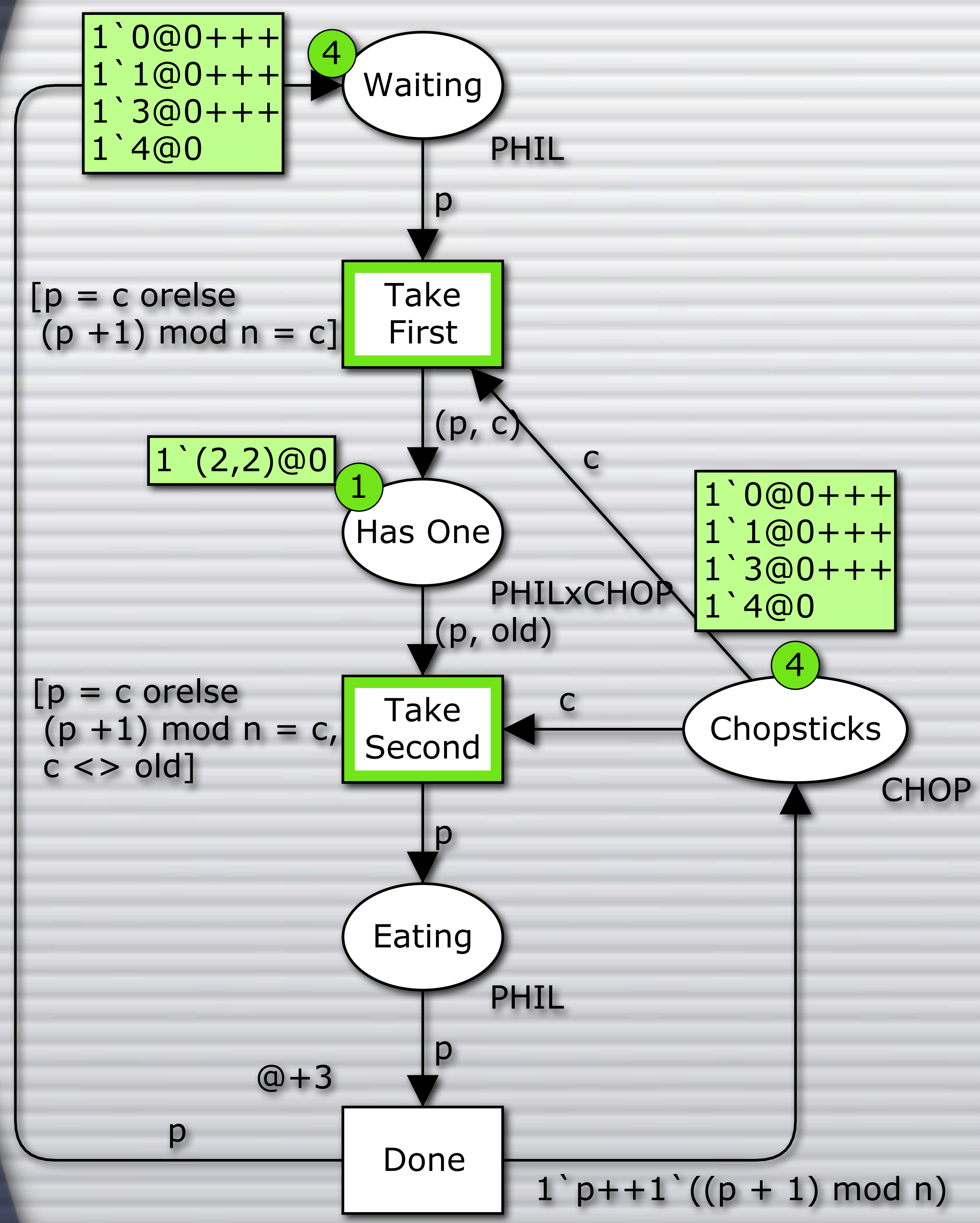
Support for Timed Models

- Models with finite untimed behavior can have infinite timed state spaces
- Time equivalence does not record absolute time stamps, only the difference between the current time and the time stamp



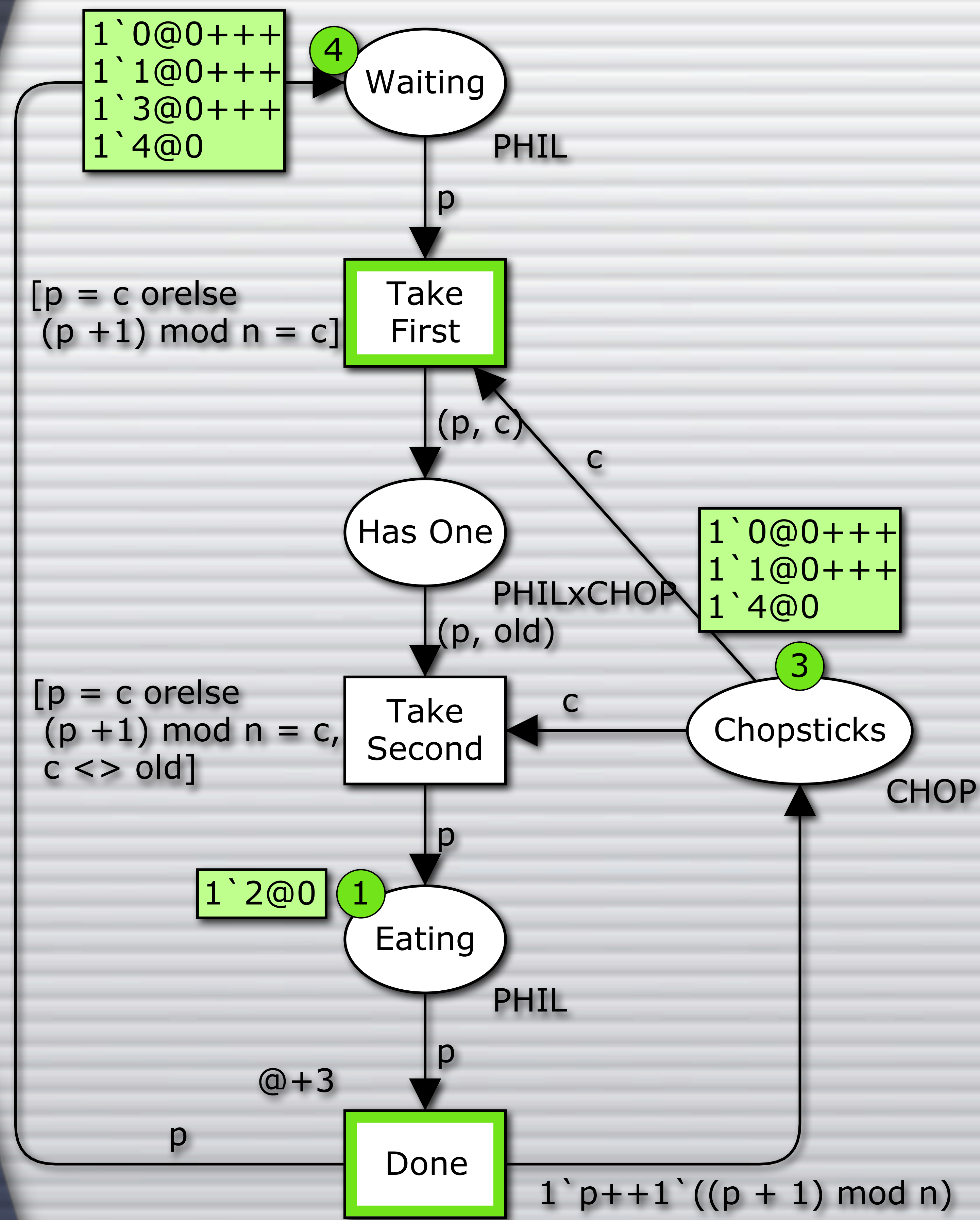
Support for Timed Models

- Models with finite untimed behavior can have infinite timed state spaces
- Time equivalence does not record absolute time stamps, only the difference between the current time and the time stamp



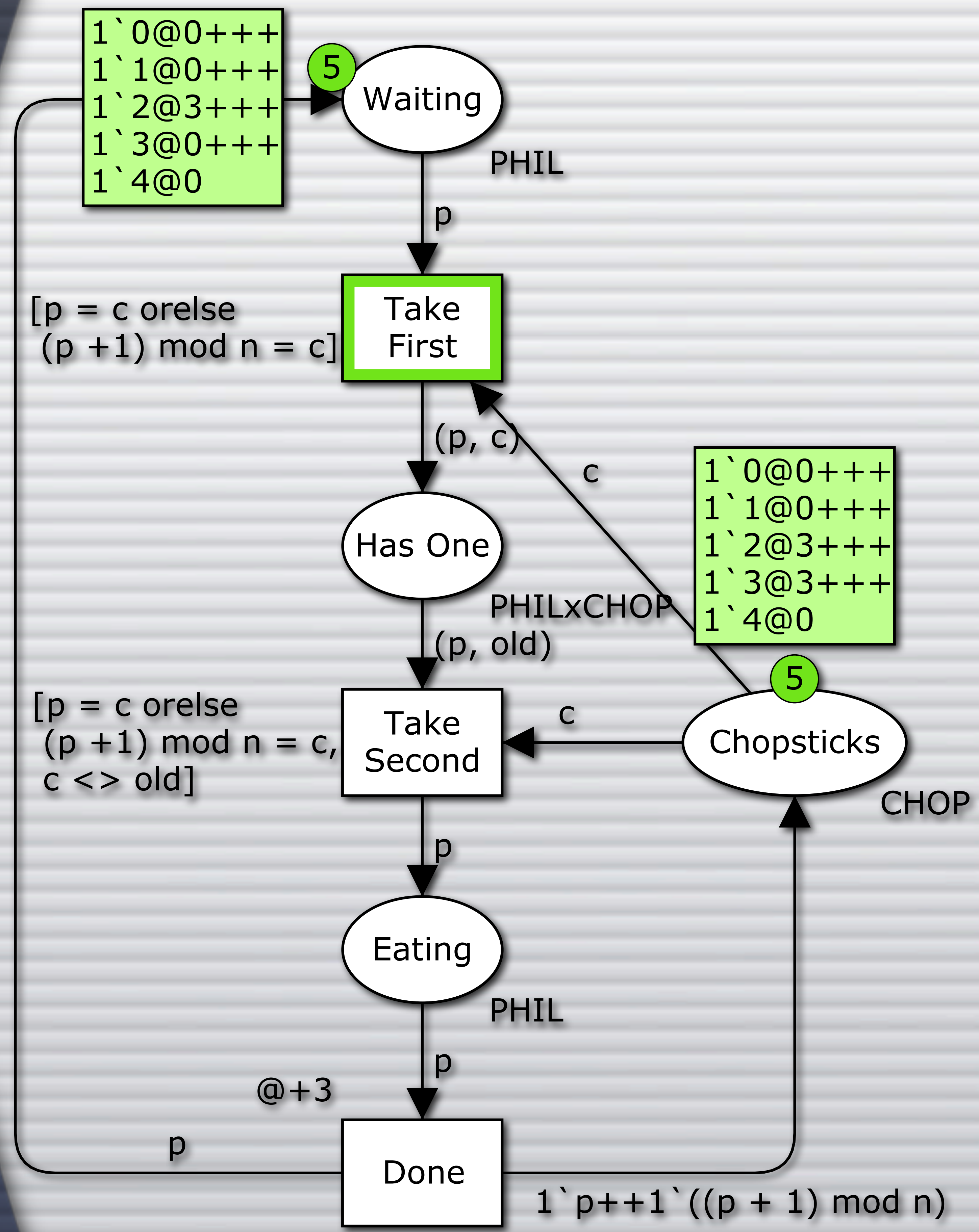
Support for Timed Models

- Models with finite untimed behavior can have infinite timed state spaces
- Time equivalence does not record absolute time stamps, only the difference between the current time and the time stamp



Support for Timed Models

- Models with finite untimed behavior can have infinite timed state spaces
- Time equivalence does not record absolute time stamps, only the difference between the current time and the time stamp



Standard Report

- Most of the engine is there (exploration, SCC graphs, extensible reporting engine)
- The properties need to be generated from the model, and code needs to be written for this

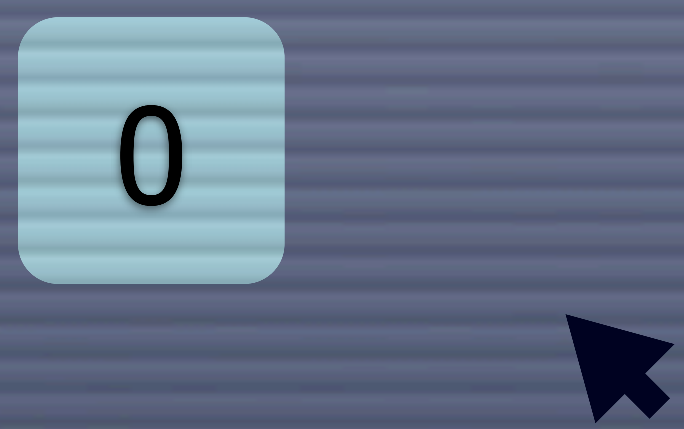
On-line Drawing of Graphs

- Now, graphs are drawn all at once
- Normally, we'll just want to explore parts of the graph interactively (like in CPN Tools)

On-line Drawing of Graphs

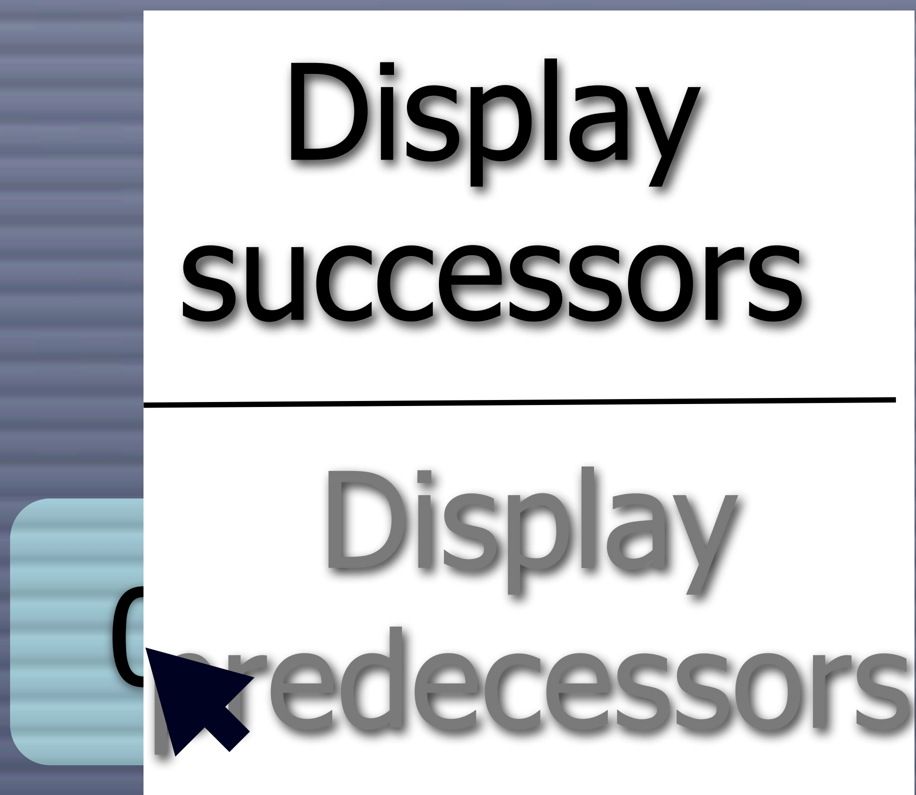
- ASAP keeps a real representation of the graph fragments it draws in memory (instead of just state numbers)
- If we only require to be able to draw out-going nodes, we do not even have to precompute the entire graph

On-line
Drawing of Graphs

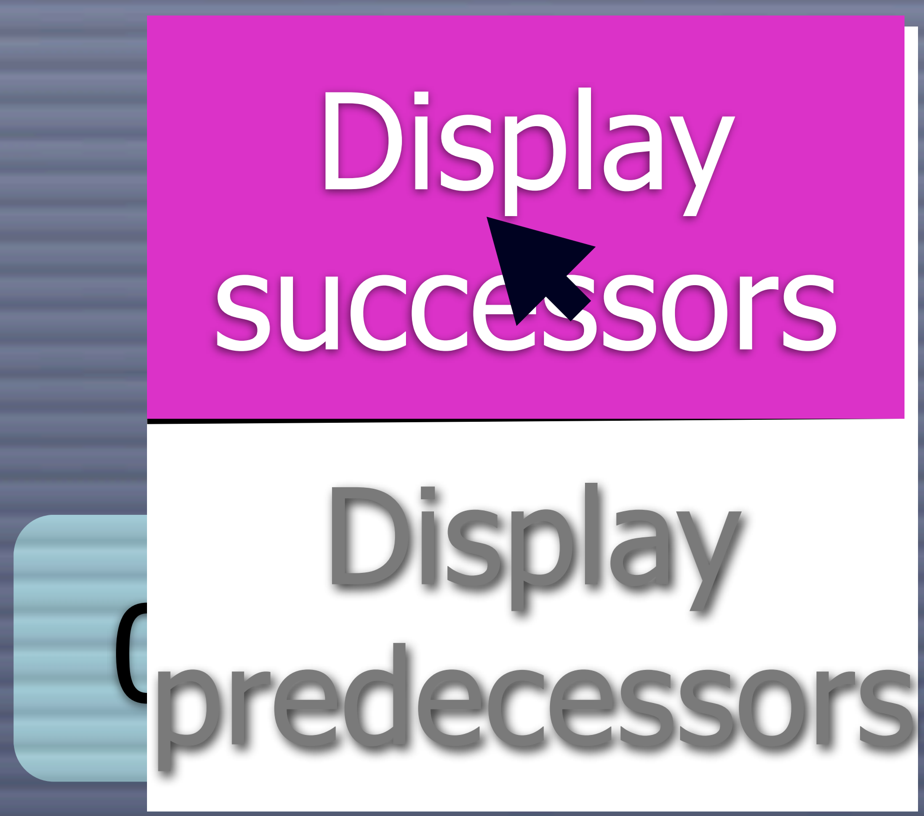


On-line
Drawing of Graphs

On-line Drawing of Graphs



On-line
Drawing of Graphs

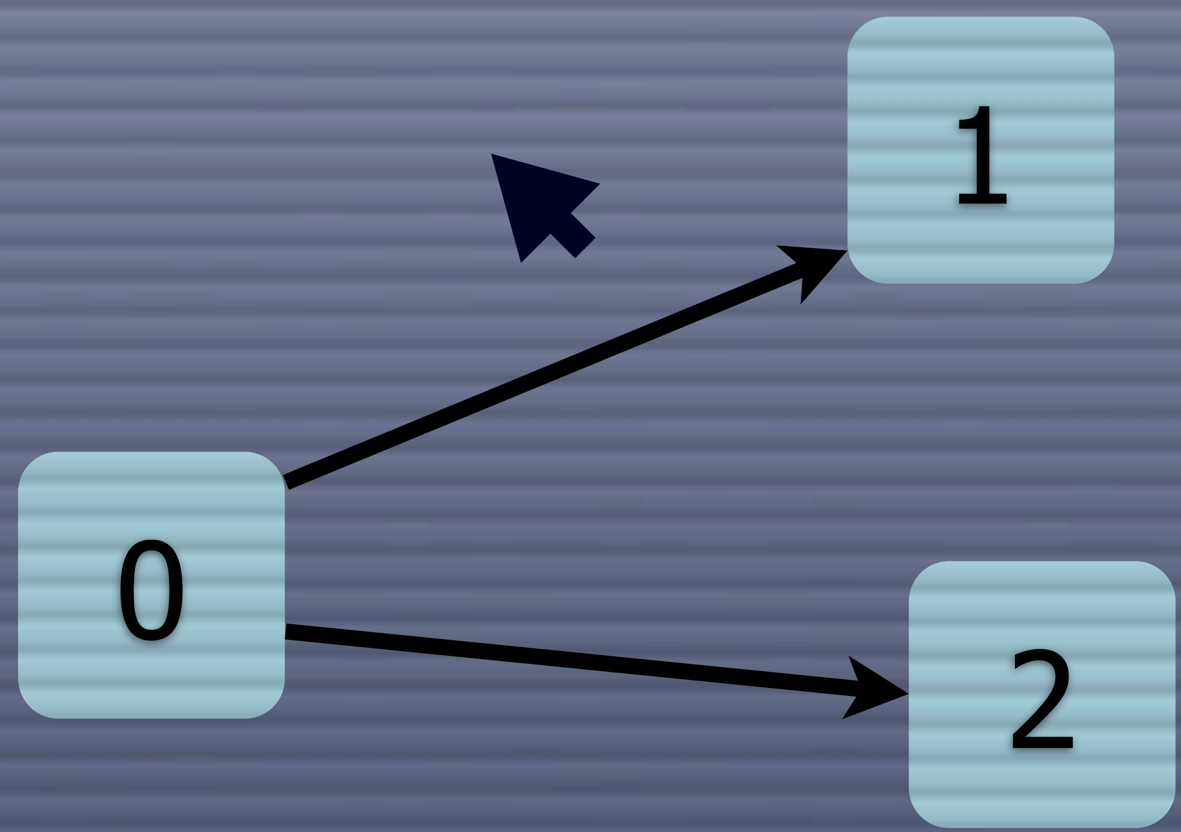


On-line
Drawing of Graphs

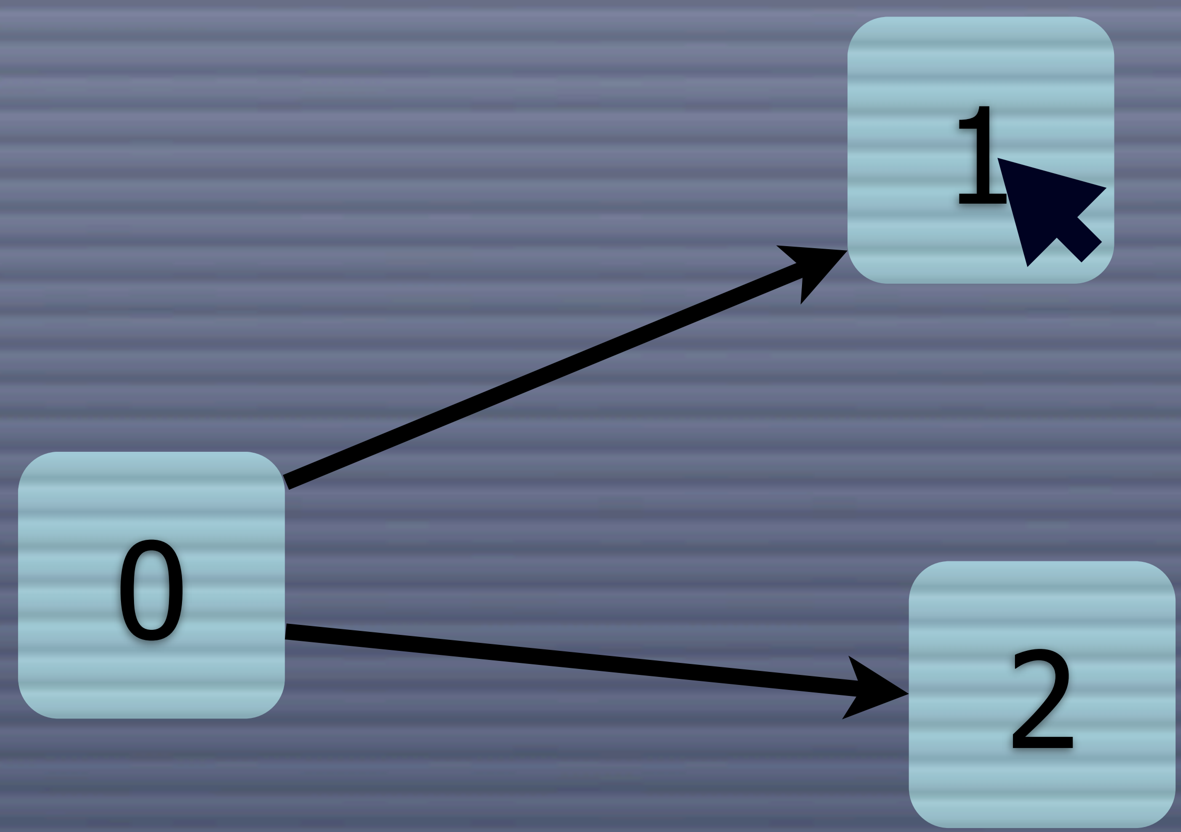
0

A representation of state 0 is sent to state space engine, which calculates and returns the successors

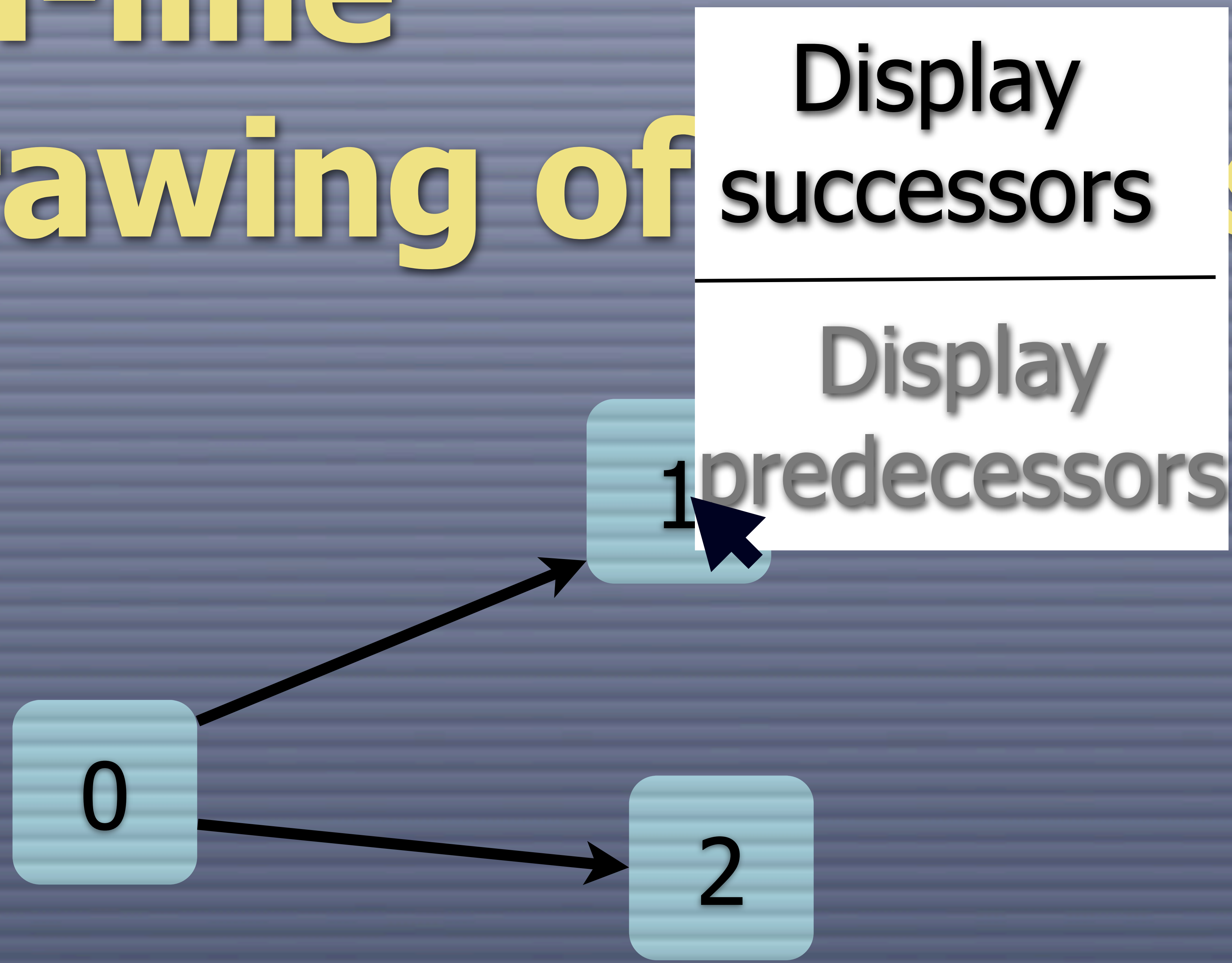
On-line
Drawing of Graphs



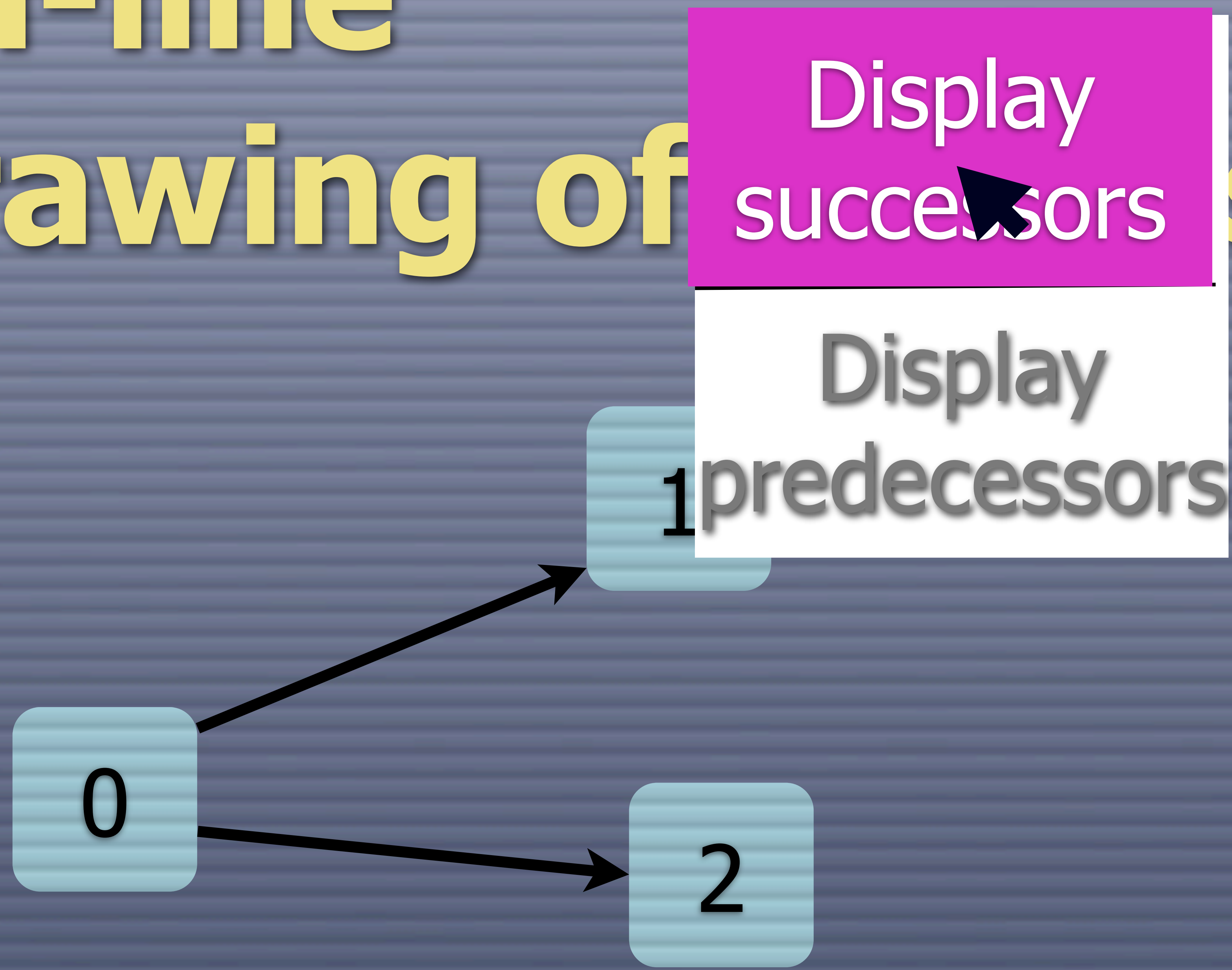
On-line
Drawing of Graphs



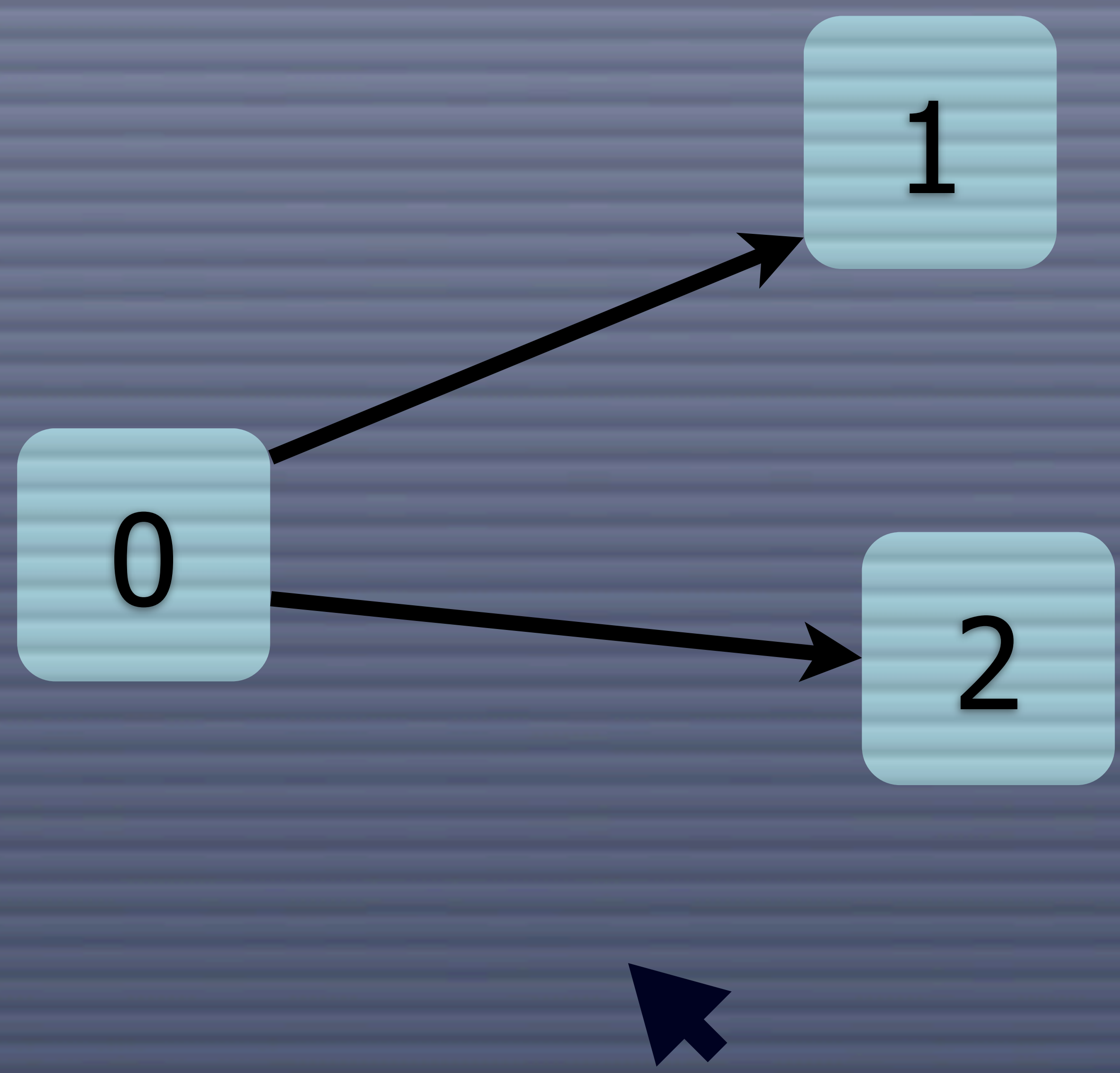
On-line Drawing of s



On-line Drawing of s

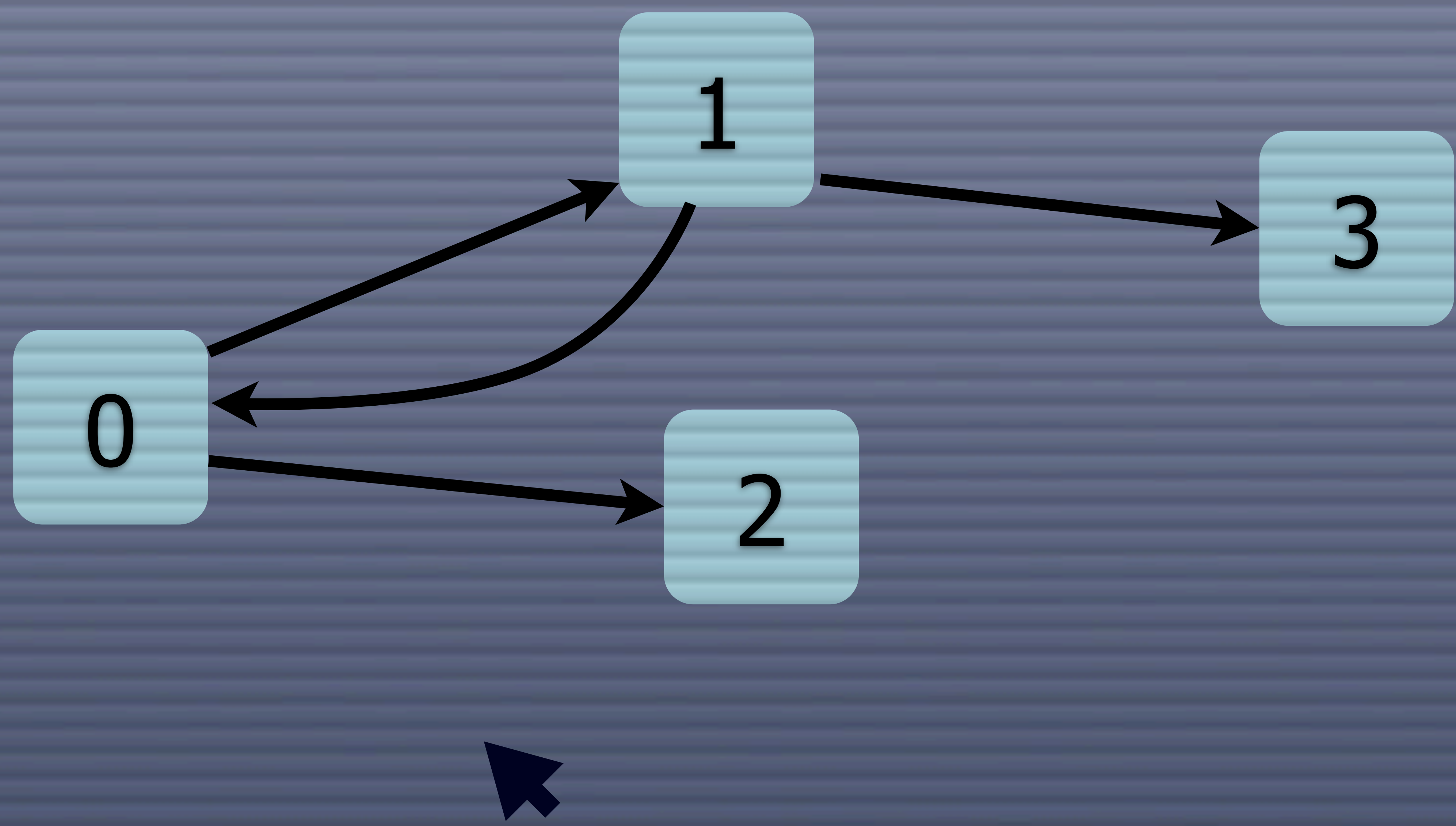


On-line Drawing of Graphs



A representation of state 1 is sent to state space engine, which calculates and returns the successors

On-line Drawing of Graphs



Longer-term New Features

- Off-line CTL analysis
- Distributed (safety) checking
- Extend JoSEL (syntactical sugar, language extensions)
- Integrate CPN viewer
- More user friendly ways to specify properties

Get It!

- ASAP can be downloaded from
  www.cs.au.dk/CPnets/projects/ascoveco/asap.html
- Access/CPN can be downloaded from
  www.cs.au.dk/CPnets/projects/ascoveco/accesscpn

Get It!

Thanks for you attention!

- ASAP can be downloaded from
  www.cs.au.dk/CPnets/projects/ascoveco/asap.html
- Access/CPN can be downloaded from
  www.cs.au.dk/CPnets/projects/ascoveco/accesscpn