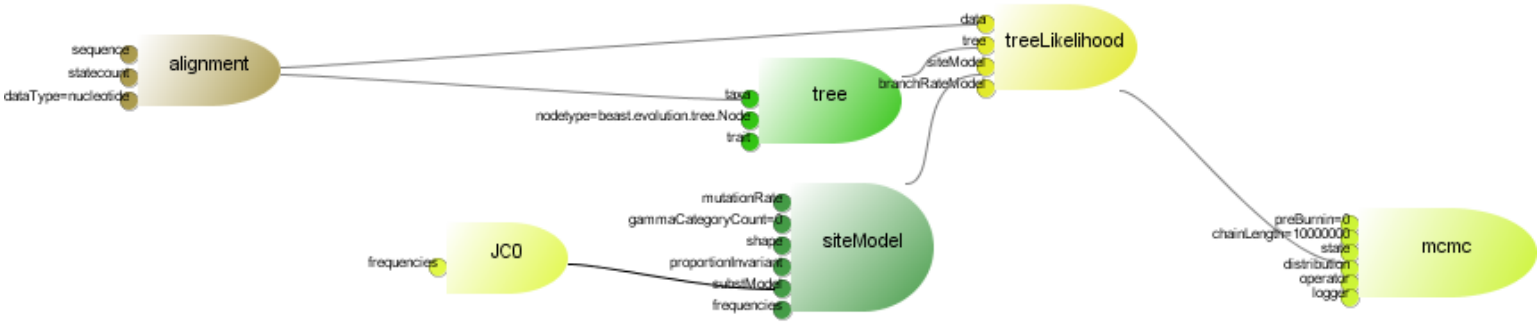
Beast II 101: Part 1

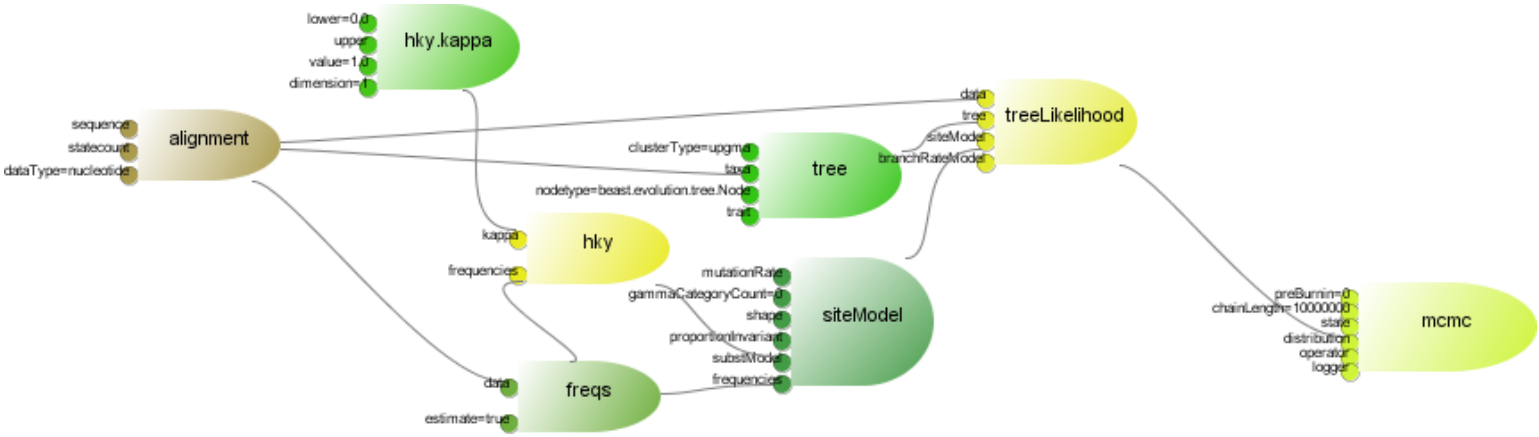
Remco R. Bouckaert
remco@cs.{auckland|waikato}.ac.nz
Department of Computer Science
University of Auckland & University of Waikato

Jukes Cantor



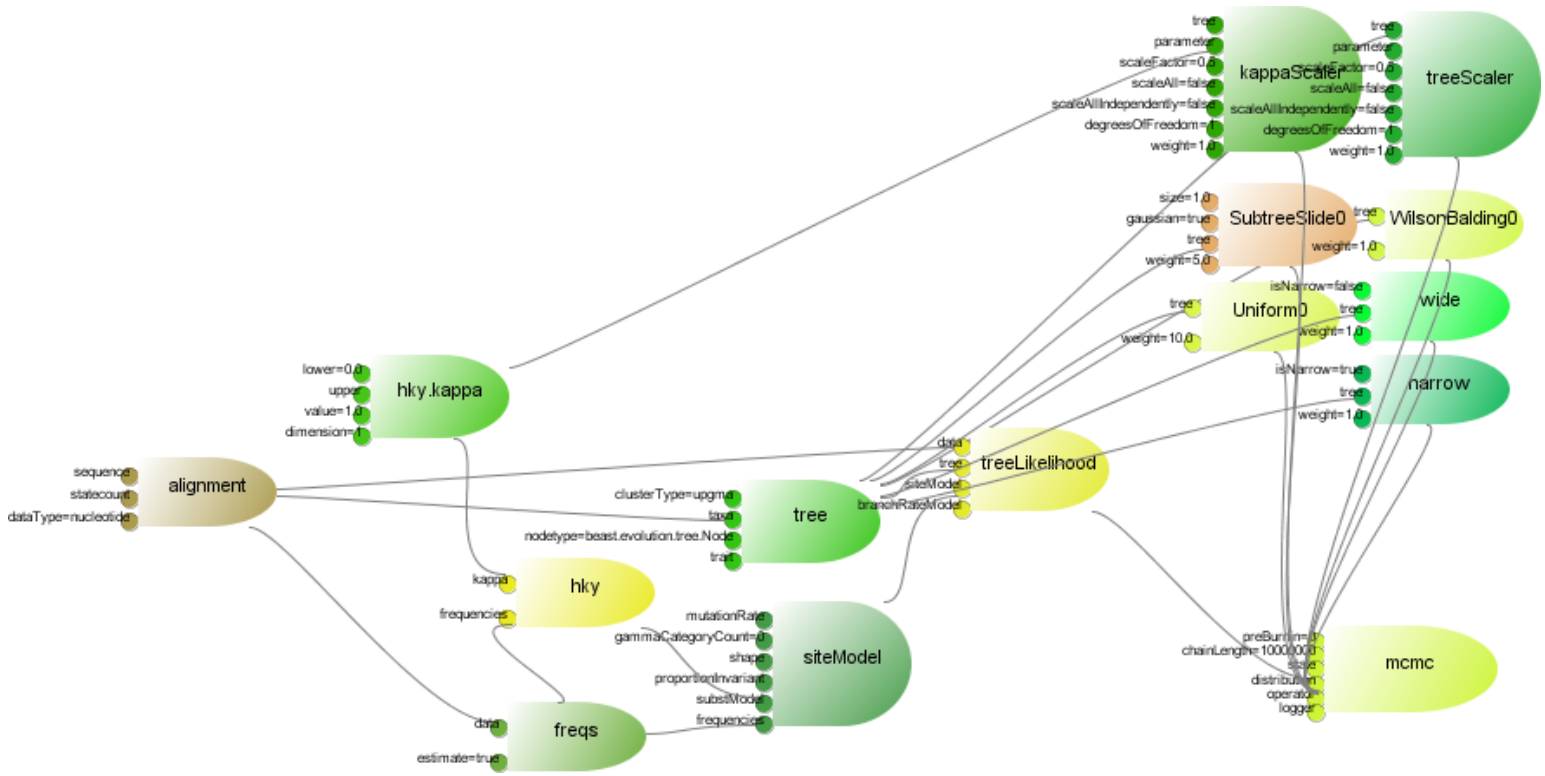
All objects are Plugins - connected to each other through Inputs

HKY



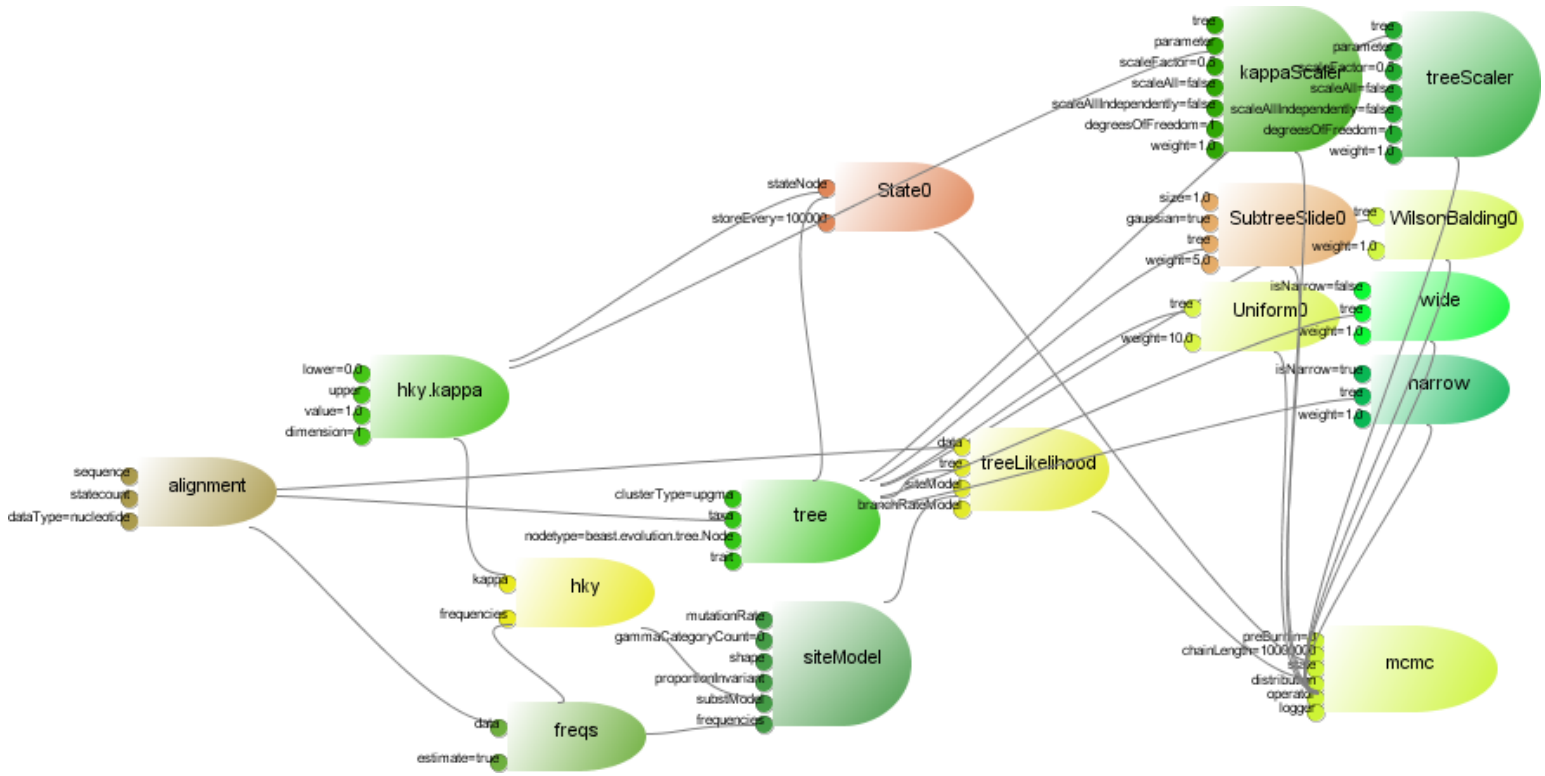
Adding kappa parameter and frequencies

Adding operators



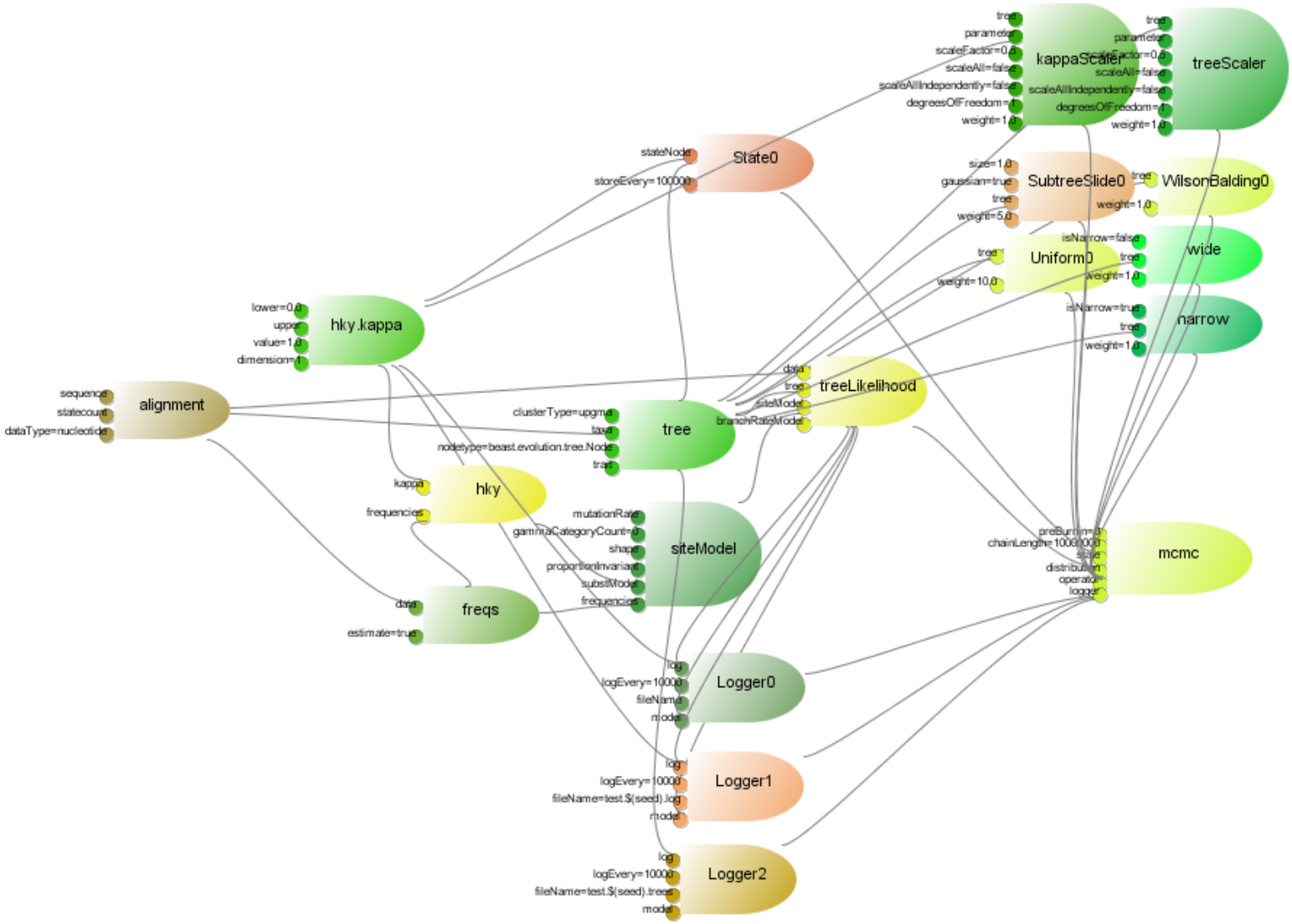
Operate on kappa parameter and tree

Adding State



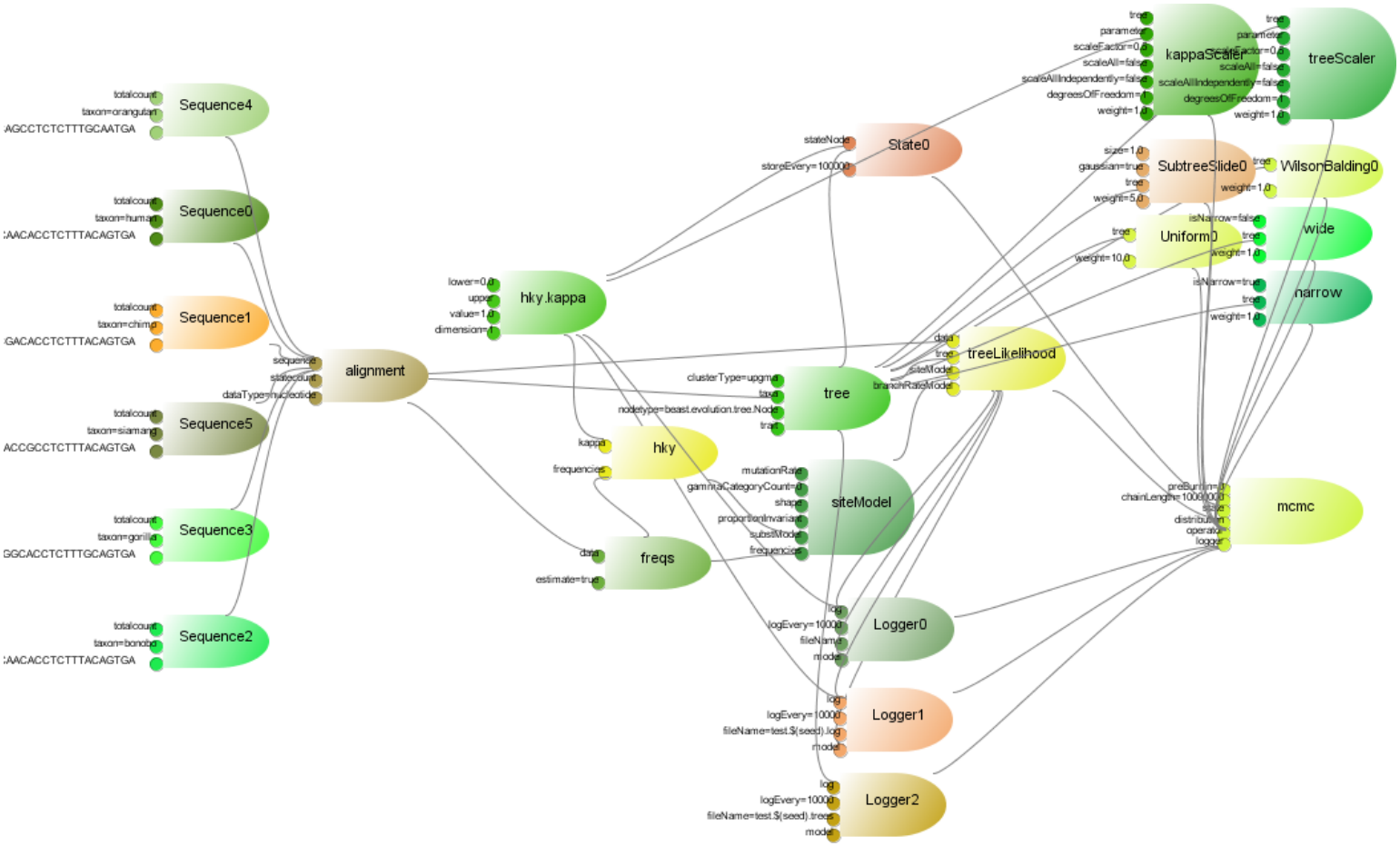
The state contains every Plugin that operators work on

Adding Loggers



3 loggers: screen, trace and trees

Adding Sequences



Inputs to alignments, which takes care of the patterns, DataType and set of Taxon names

What Beast 2 does

...supposed to do...

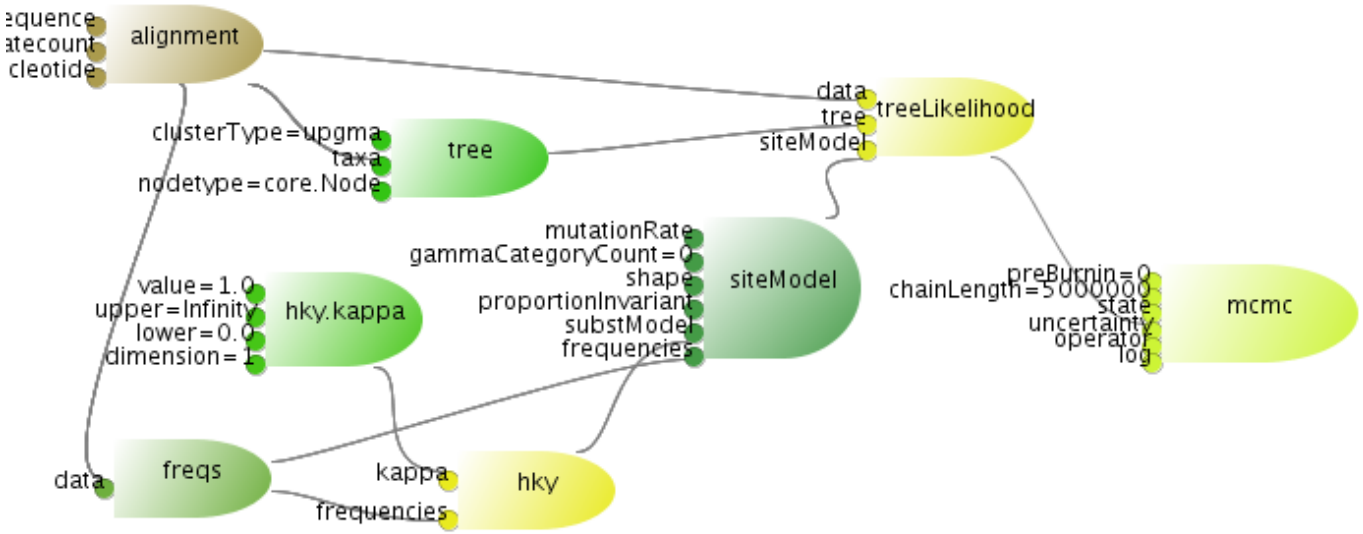
- The kind of Bayesian analysis as per citations on the Beast 1 wiki.
- Beauti 2: GUI to specify analysis.
- Provide a platform to develop add-ons - powerful interface, easy extensible XML, templates for Beauti.
- Sequence generator for simulation studies.
- Documentation for all the above – from user to developer, XML tweaker, etc.

What Beast 2 doesn't

- Post-analysis processing like Tracer, tree annotator, tree log analyser, DensiTree, KML producer
- Most non-Bayesian analysis
- Laundry

Phylosophy

Everything is a plug-in



Plug-ins provide...

- connection with with other plug-ins/values through ’inputs’
- validation
- documentation
- ’XML parsing’

Plugin class

```
@Description("Description goes here")
public class Plugin {
    public void initAndValidate()

    public String getDescription()
    public String getCitations()

    public String getID()
    public void setID(String sID)

} // class Plugin
```

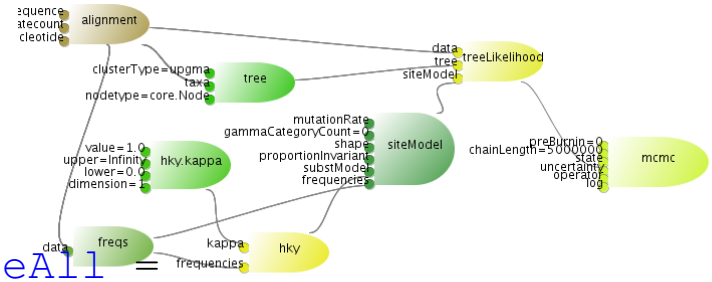
A minimal plugin

```
@Description("Description of MyPlugin goes here")
public class MyPlugin extends Plugin {
    public Input<Integer> m_value = new Input<Integer>("value",
        "value used by my plugin");

    public void initAndValidate() throws Exception {
        // go check stuff and
        // do stuff that normally goes in a constructor
    }

} // class MyPlugin
```

HKY Plugin

```
@Description("HKY85 (Hasegawa, Kishino & Yano, 1985) substitution model o
@Citation("Hasegawa, M., Kishino, H and Yano, T. 1985. Dating the human-a
        "Journal of Molecular Evolution 22:160-174.")
public class HKY extends 'Plugin' {
    public Input<Frequencies> m_freqs = new Input<Frequencies>("frequenci
    public Input<Parameter> m_kappa = new Input<Parameter>("kappa", "kapp

    @Override public void initAndValidate() throws Exception {
        initialiseEigen();
    }

     public void getTransitionProbabilities(Node node,
                double fStartTime,
                double fEndTime,
                double fRate,
                double[] matrix) {...}

    @Override
    protected boolean requiresRecalculation() {...}

    @Override public void store() {...}
    @Override public void restore() {... }
} // class HKY
```

Inputs



Simple primitives

```
public Input<Boolean> m_pScaleAll =
    new Input<Boolean>("scaleAll",
        "if true, all elements of a parameter are scaled, otherwise one i
```

Other plugins

```
public Input<Frequencies> m_freqs =
    new Input<Frequencies>("frequencies",
        "frequencies of nucleotide letters");
```

Multiple inputs

```
public Input<List<Parameter>> m_pParameters =
    new Input<List<Parameter>>("parameter",
        "parameter, part of the state",
        new ArrayList<Parameter>());
```

Inputs

Enumerations

```
final static String [] UNITS =  {"year", "month", "day"};

public Input<String> m_sUnits = new Input<String>("units",
        "name␣of␣the␣units␣in␣which␣values␣are␣posed,␣" +
                "used␣for␣conversion␣to␣a␣real␣value.␣This␣can␣be␣" +
        Arrays.toString(UNITS) + "␣(default␣'year')",
        "year",
        UNITS);
```

Input validation

Default: OPTIONAL (see previous slide)

If input is REQUIRED:

```
public Input<Parameter> m_kappa =
    new Input<Parameter>("kappa",
        "kappa parameter in HKY model",
        Validate.REQUIRED);

public Input<List<Operator>> m_operators =
    new Input<List<Operator>>("operator",
        "operator for generating proposals in MCMC state space",
        new ArrayList<Operator>(), Validate.REQUIRED);
```

If input is XOR:

```
public Input<Tree> m_pTree =
    new Input<Tree>("tree",
        "if specified, all tree branch length are scaled");
public Input<Parameter> m_pParameter =
    new Input<Parameter>("parameter",
        "if specified, this parameter is scaled"
        , Validate.XOR, m_pTree);
```

State

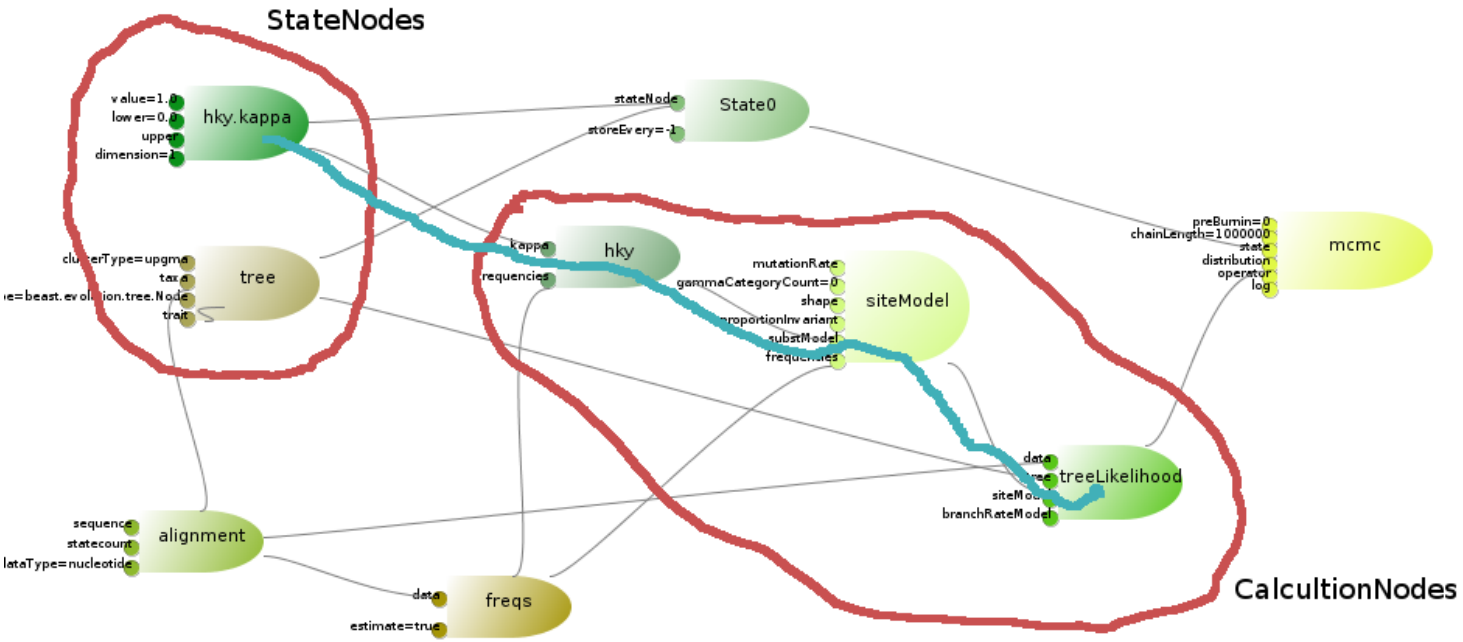
- State is explicit in XML & as object (unlike Beast 1)
- Contains StateNodes, e.g., parameters and trees
- Operators work on the StateNodes

```
public double proposal() throws Exception {...}
```

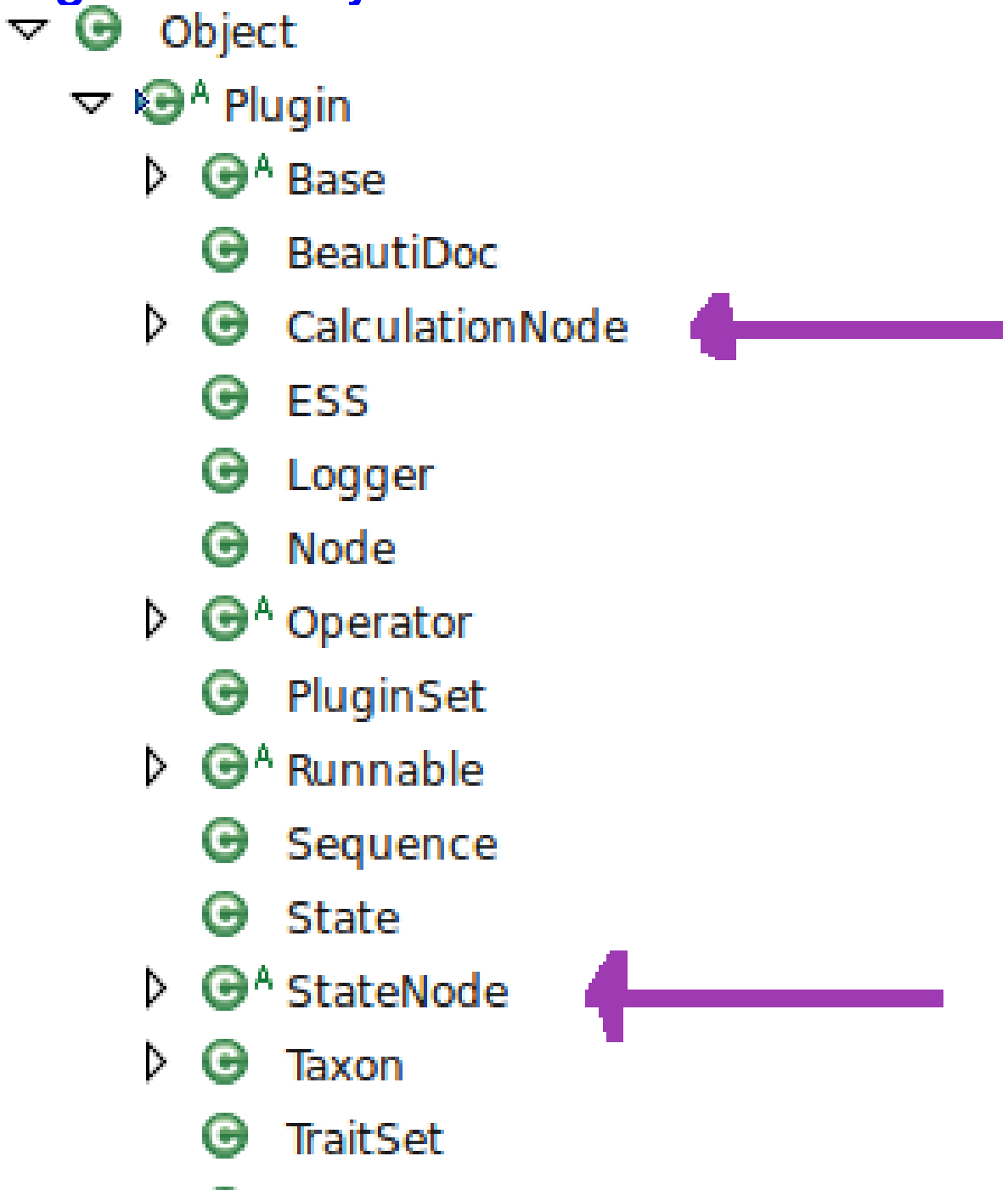
- State can be stored to disk/restored
- State can store/restore itself for MCMC proposals

MCMC Library

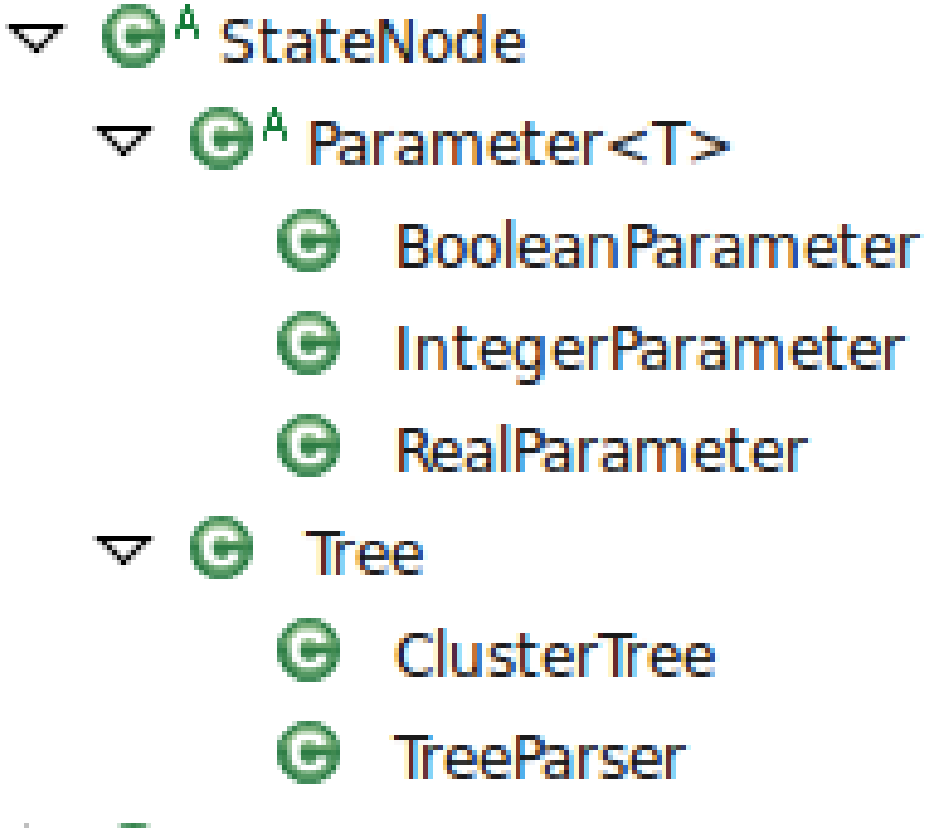
StateNode vs CalculationNode



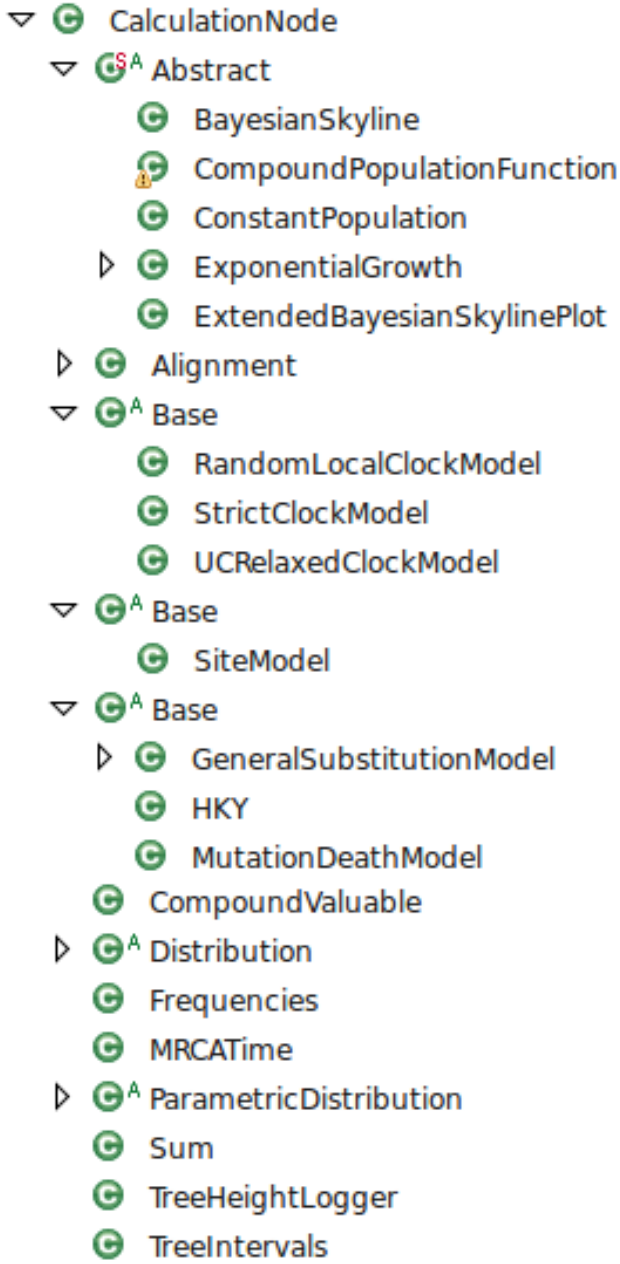
Plugin hierarchy



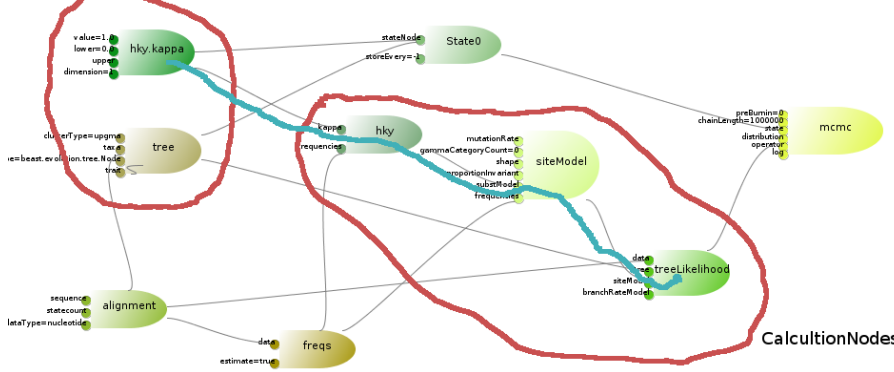
StateNode hierarchy



CalculationNode hierarchy



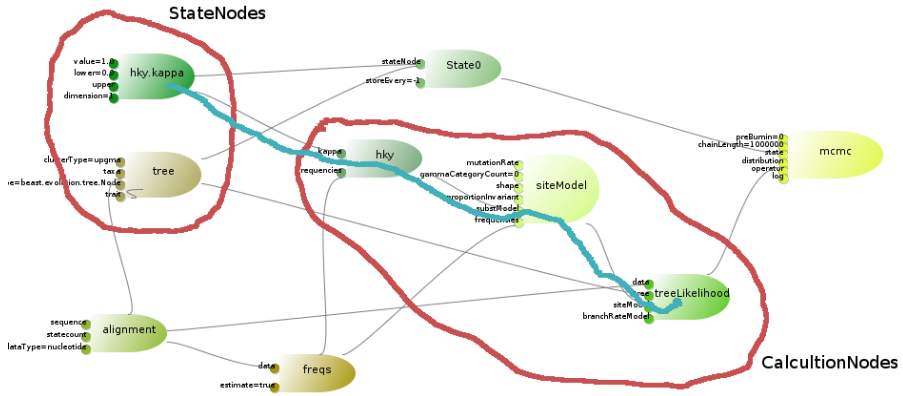
MCMC loop



Propose new state

```
logP = calculateLogP();
if (new state is acceptable)
        // do something

else
        // do something else
```

MCMC loop effect on state nodes



Store state

Propose new state

logP = calculateLogP();

if (new state is acceptable)

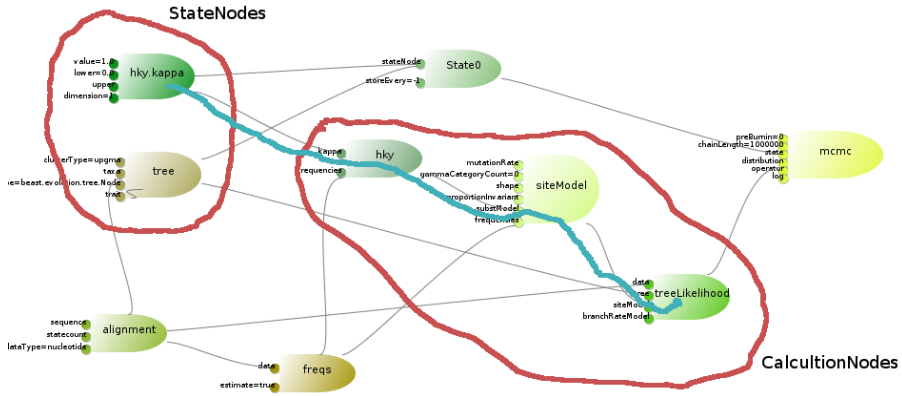
accept state

else

restore state

mark state clean

MCMC loop effect on calculation nodes



Store state

Propose new state

store calculation nodes

check dirtyness calculation nodes

logP = calculateLogP();

if (new state is acceptable)

    accept state

    mark calculation nodes clean

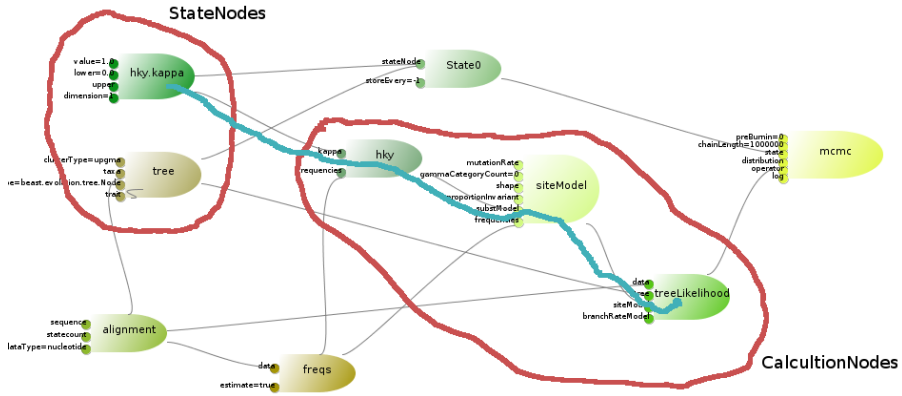
else

    restore state

    restore calculation nodes

mark state clean

MCMC loop
CalculationNode method calls



Store state

Propose new state

store calculation nodes store()

check dirtyness requiresRecalculation()

logP = calculateLogP();

if (new state is acceptable)

    accept state

    mark calculation nodes clean accept()

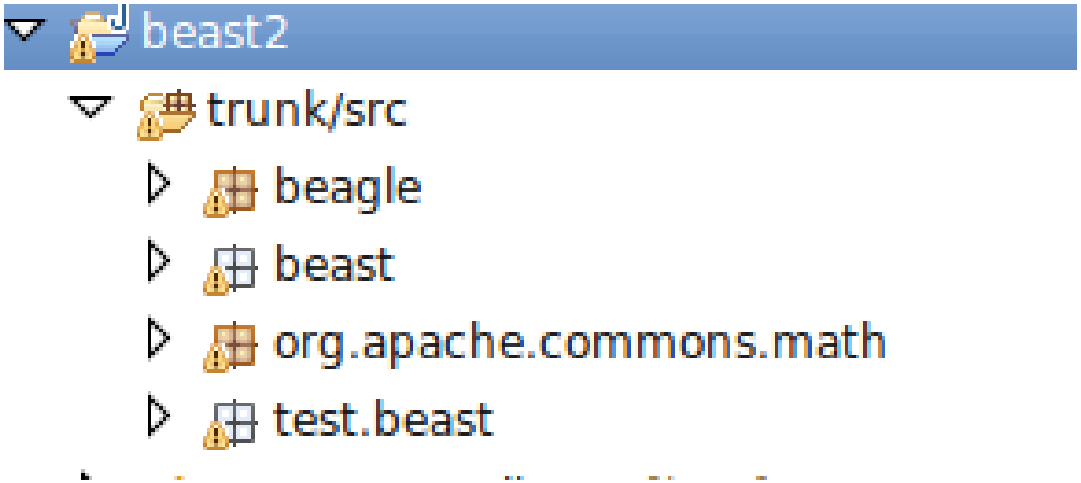
else

    restore state

    restore calculation nodes restore()

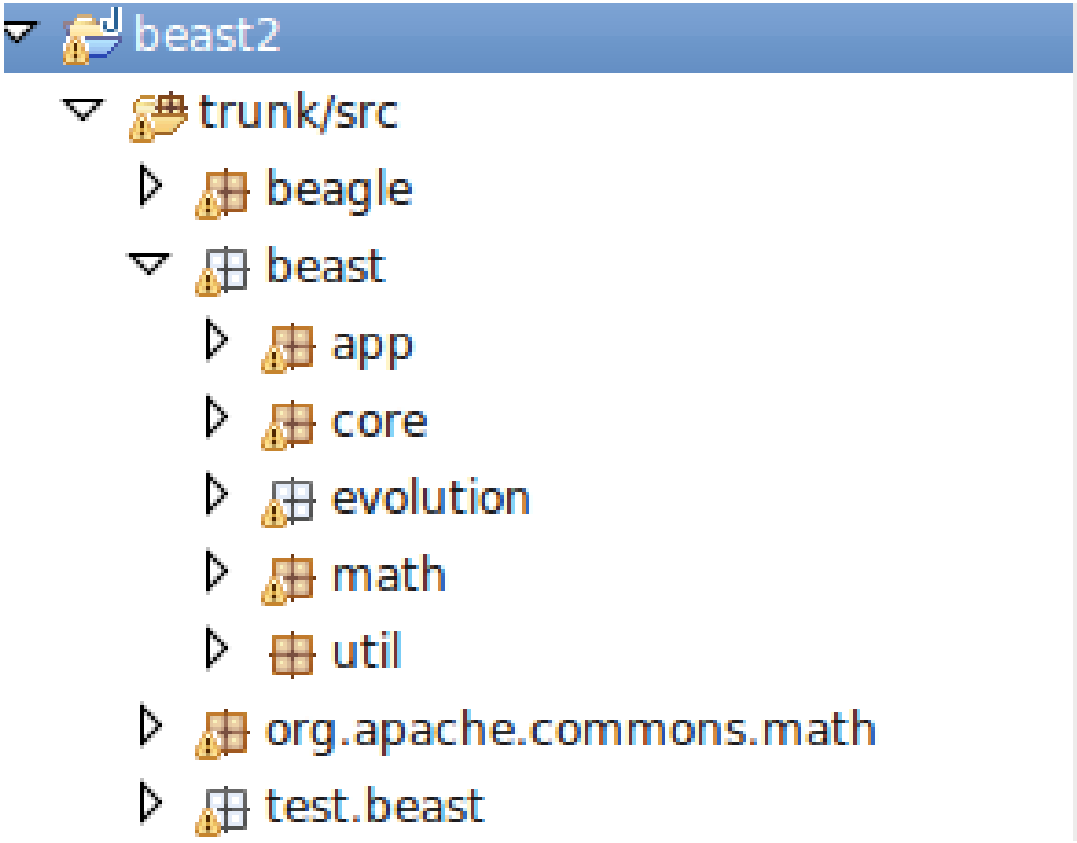
mark state clean

Beast class structure



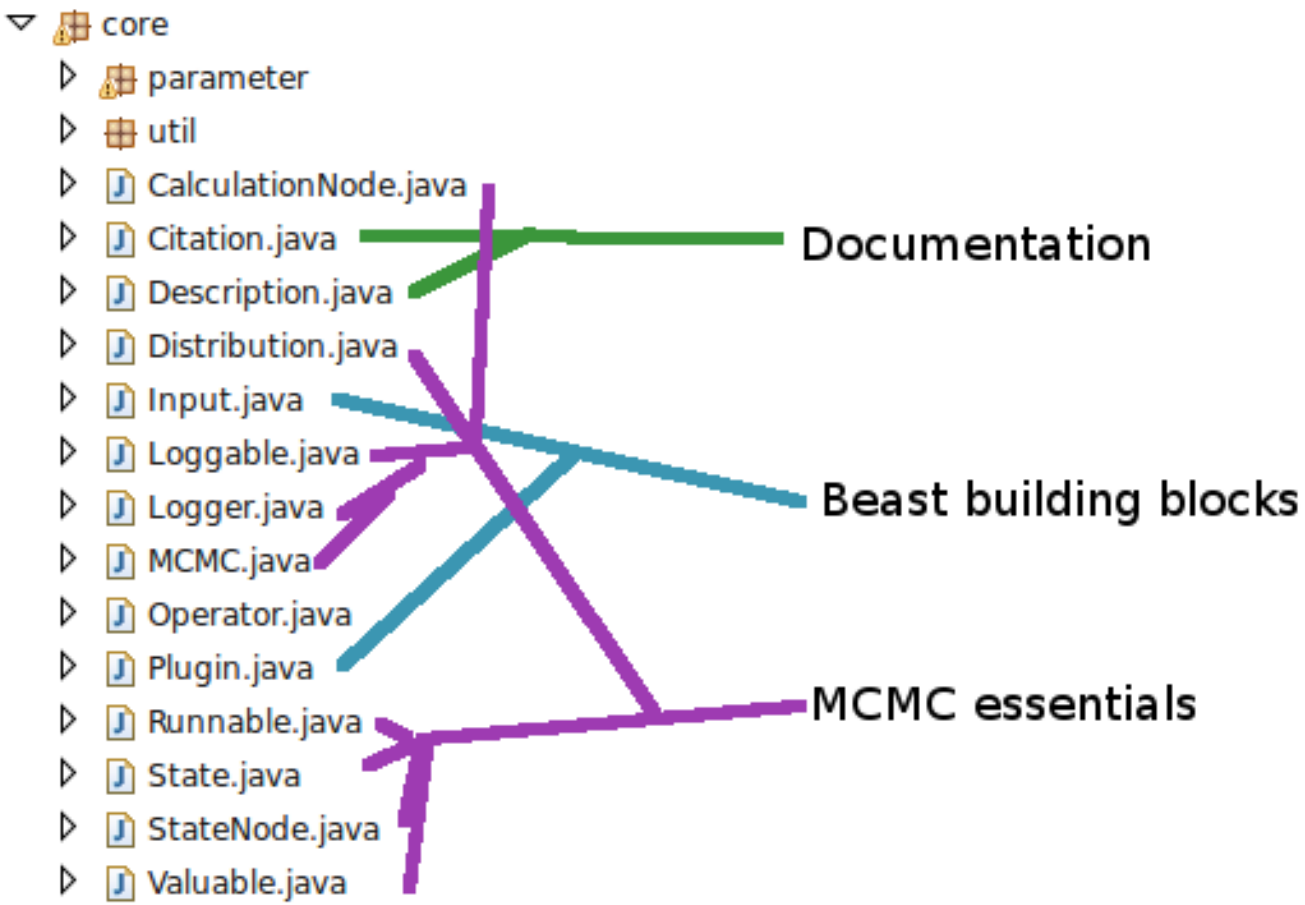
beast - main beast classes
beagle and apache libraries
test - for junit tests

Beast core class structure

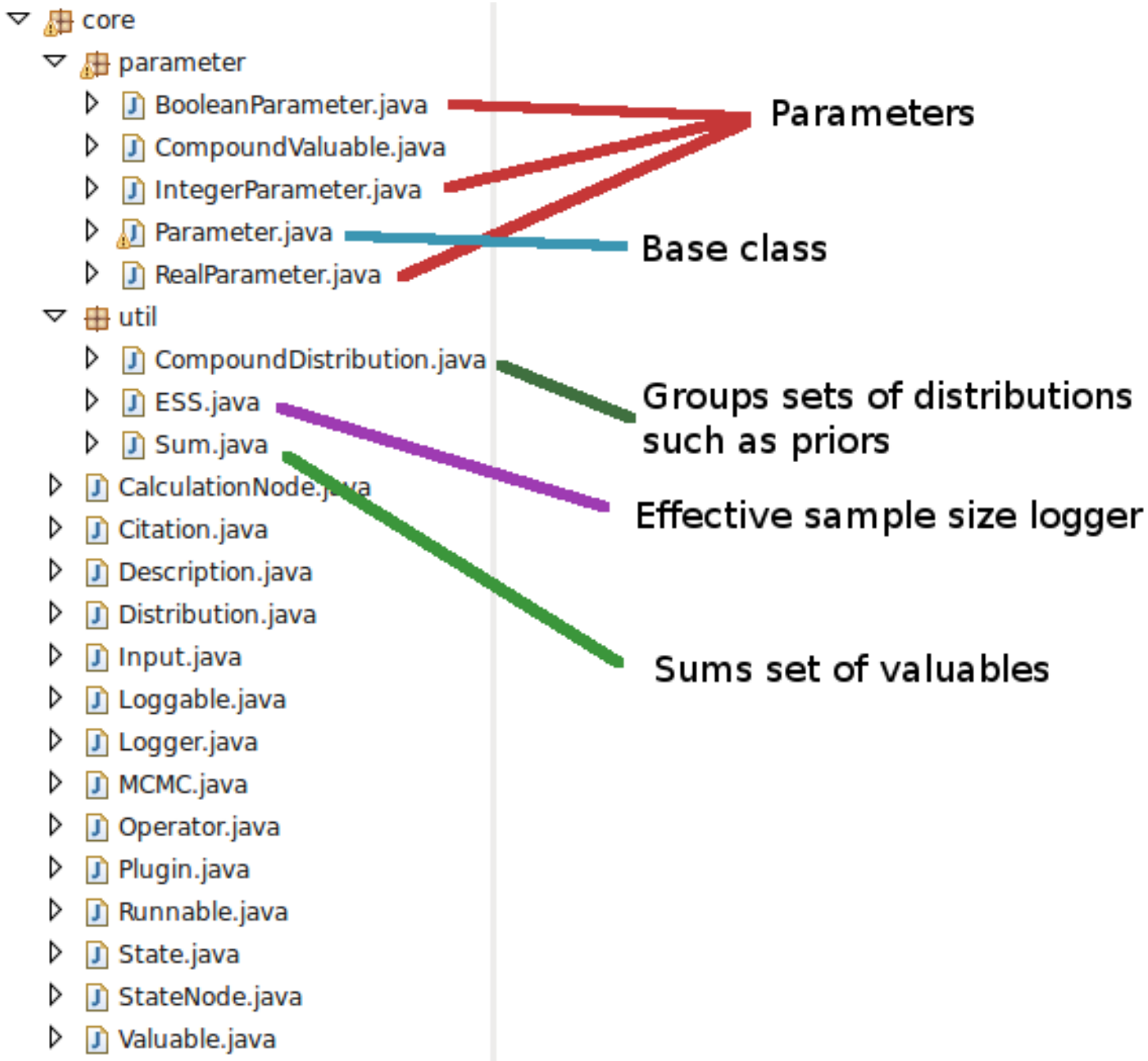


app - applications like BeastMCMC, Beauti, SequenceGenerator
core, evolution - MCMC and evolution libraries
math - mathematical classes
util - utilities like parsers, XML producers, random nr generator, class discovery.

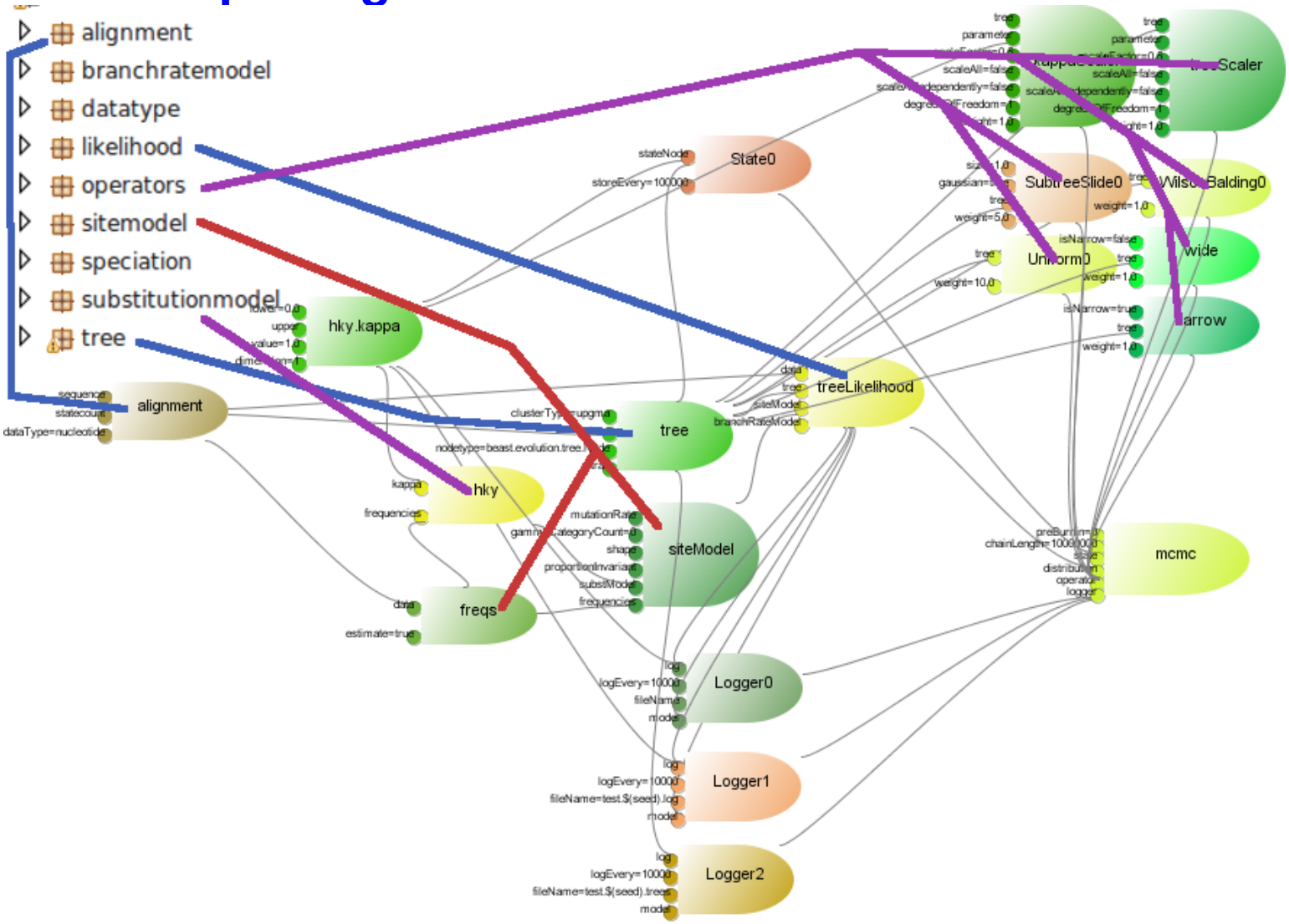
Beast core class structure



Beast core class structure

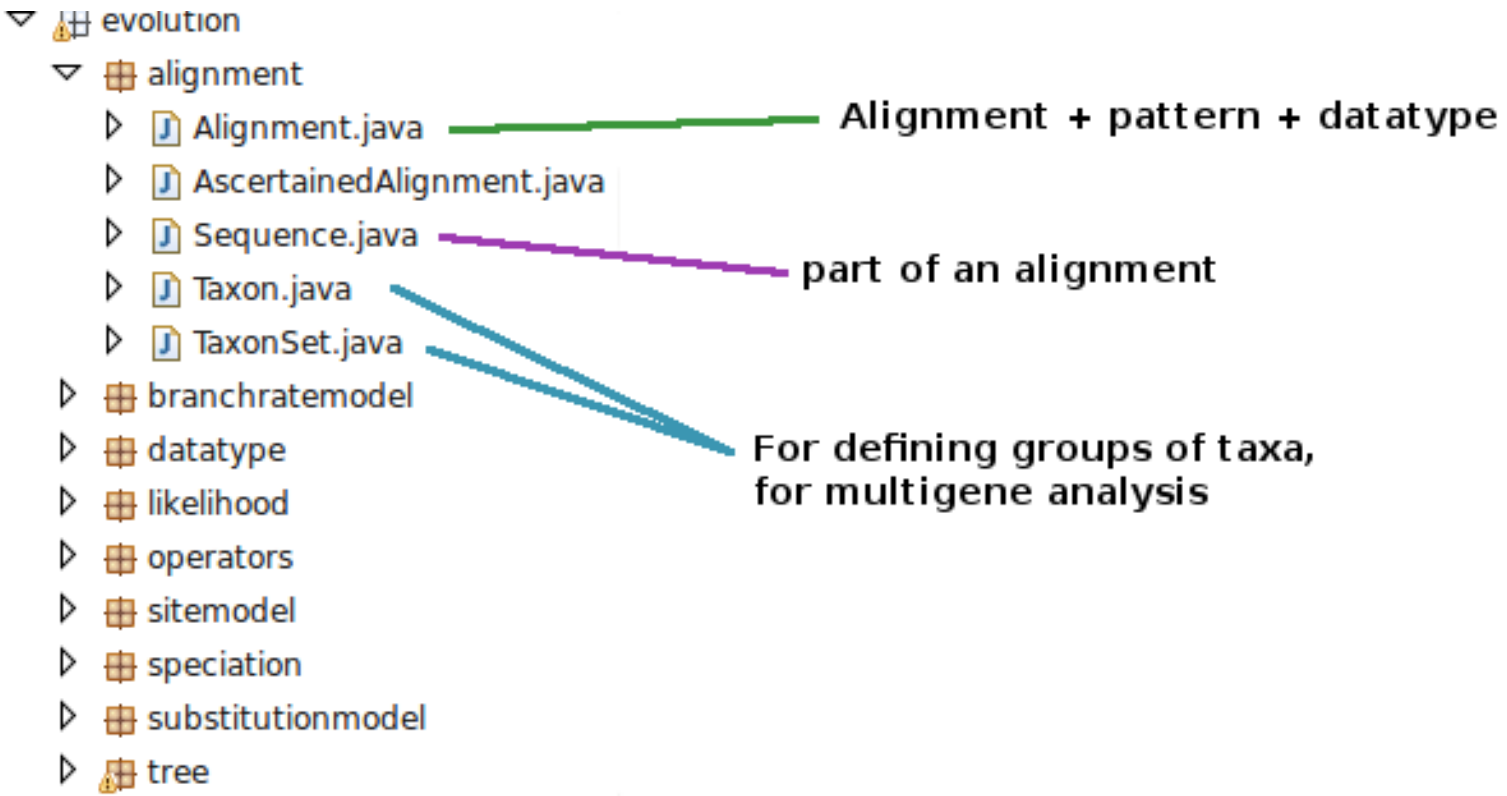


Evolution packages

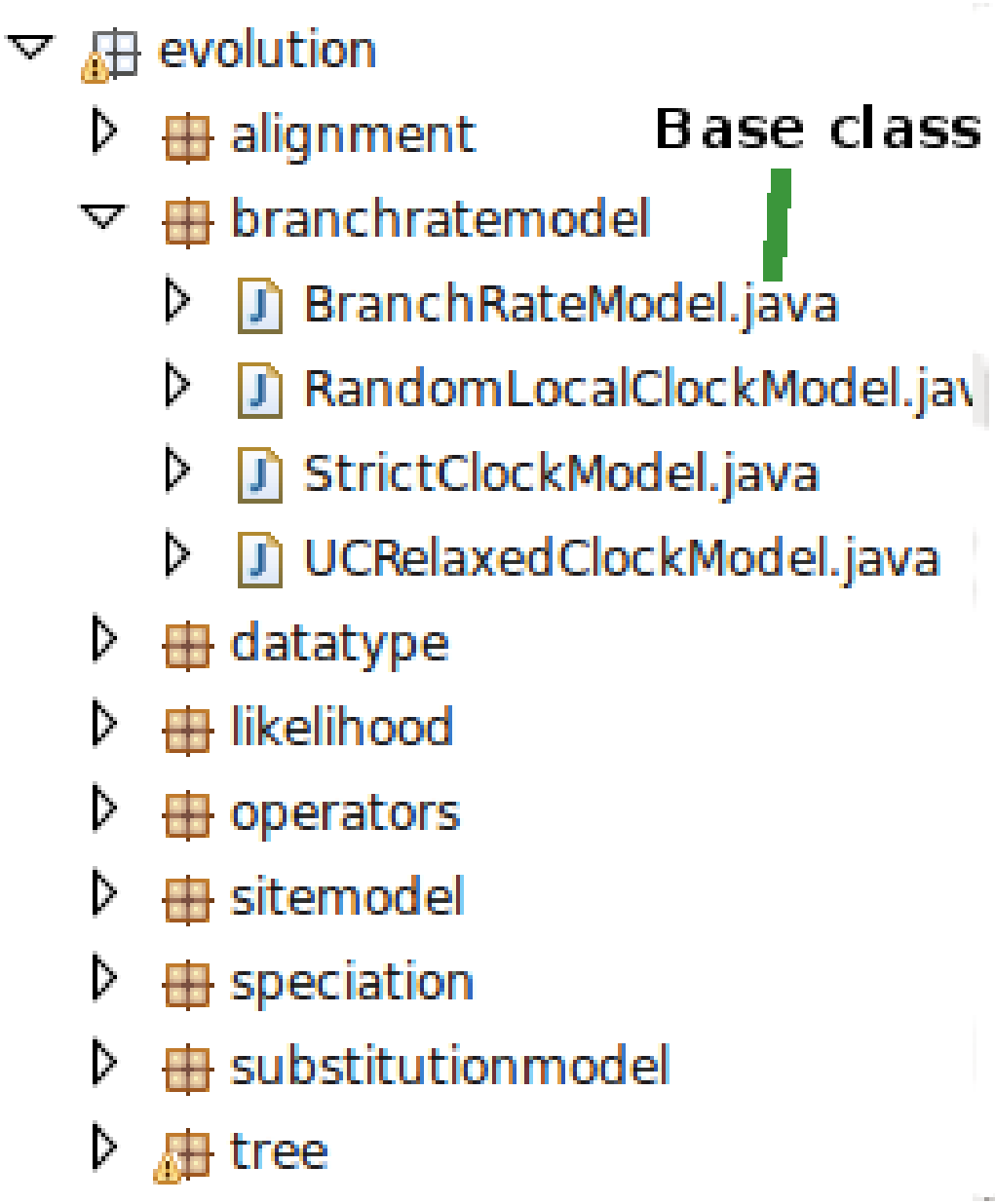


Important classes you might want to derive from: SubstitutionModel, Operator, BranchRateModel, Coalescent, SpeciationLikelihood, (DataType, Alignment, SiteModel).

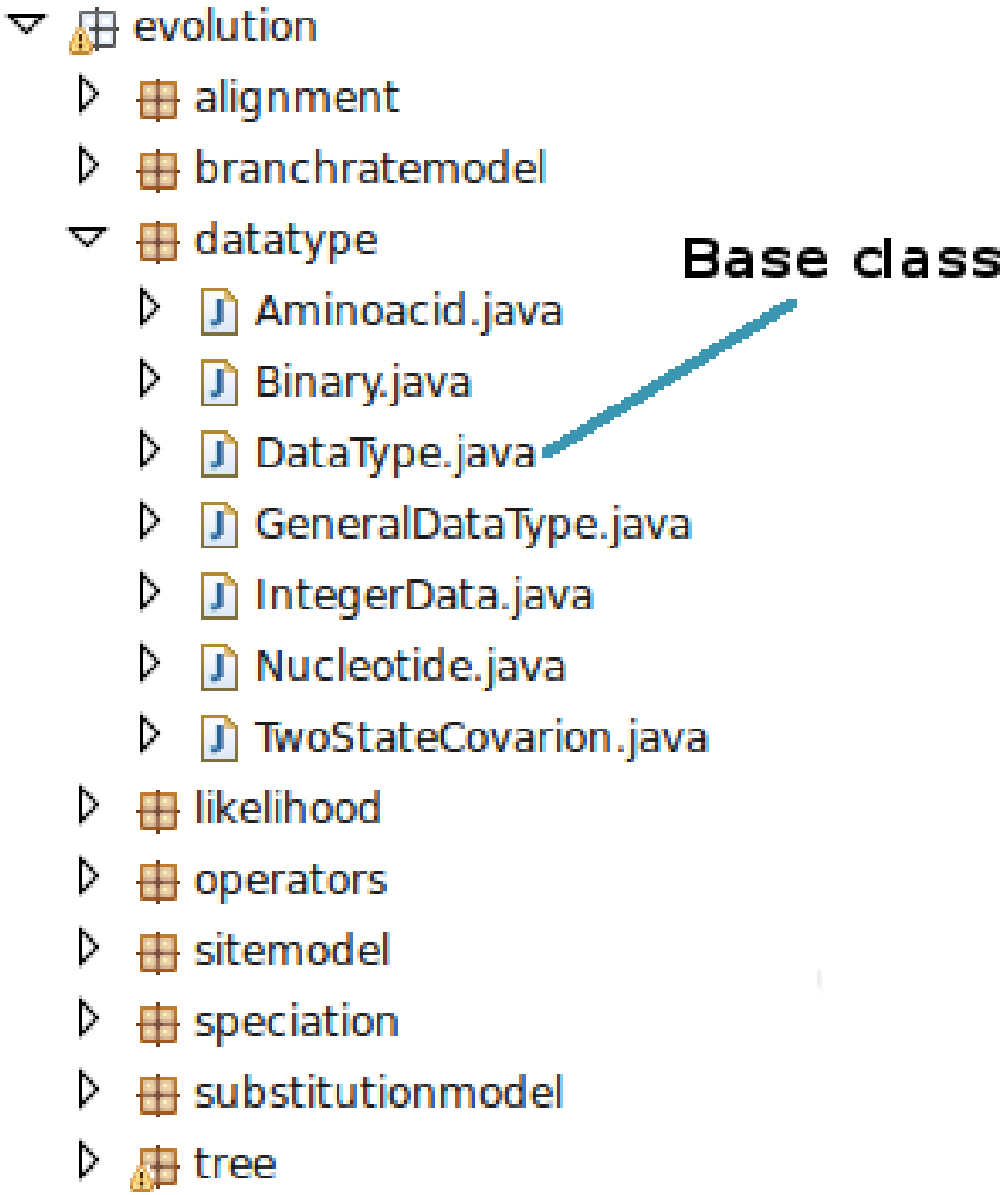
Evolution - alignment classes



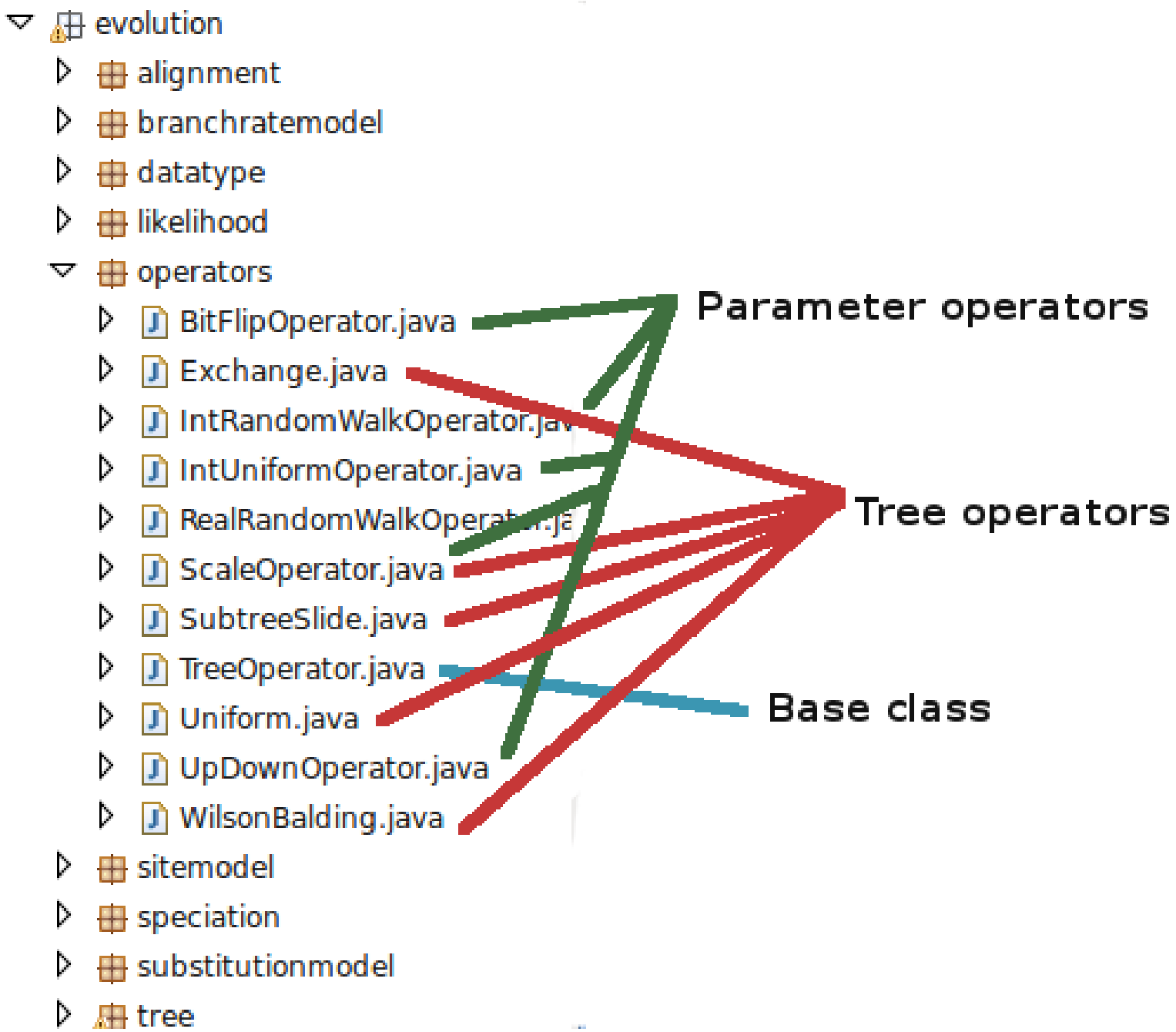
Evolution - branch rate model classes



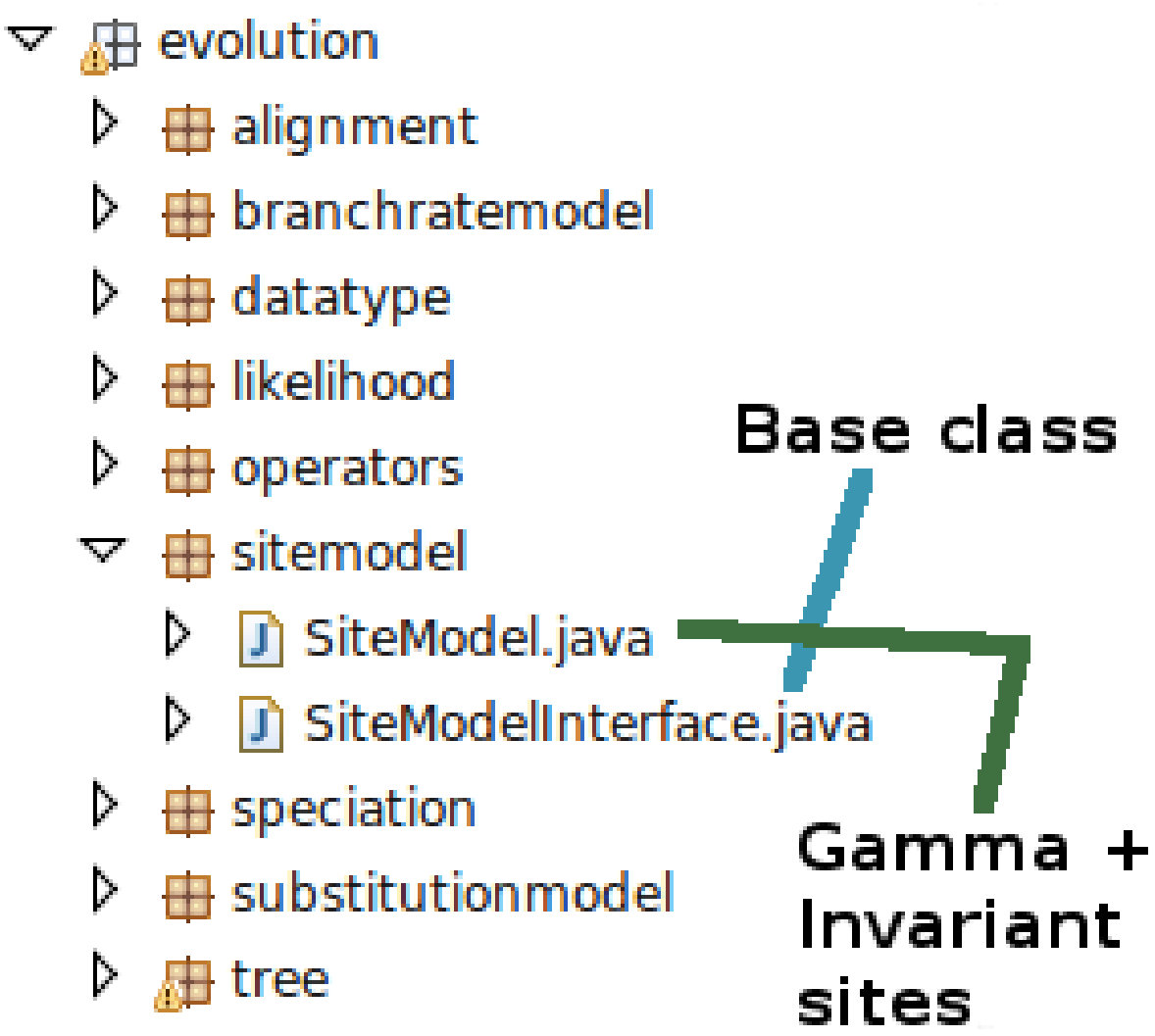
Evolution - data type classes



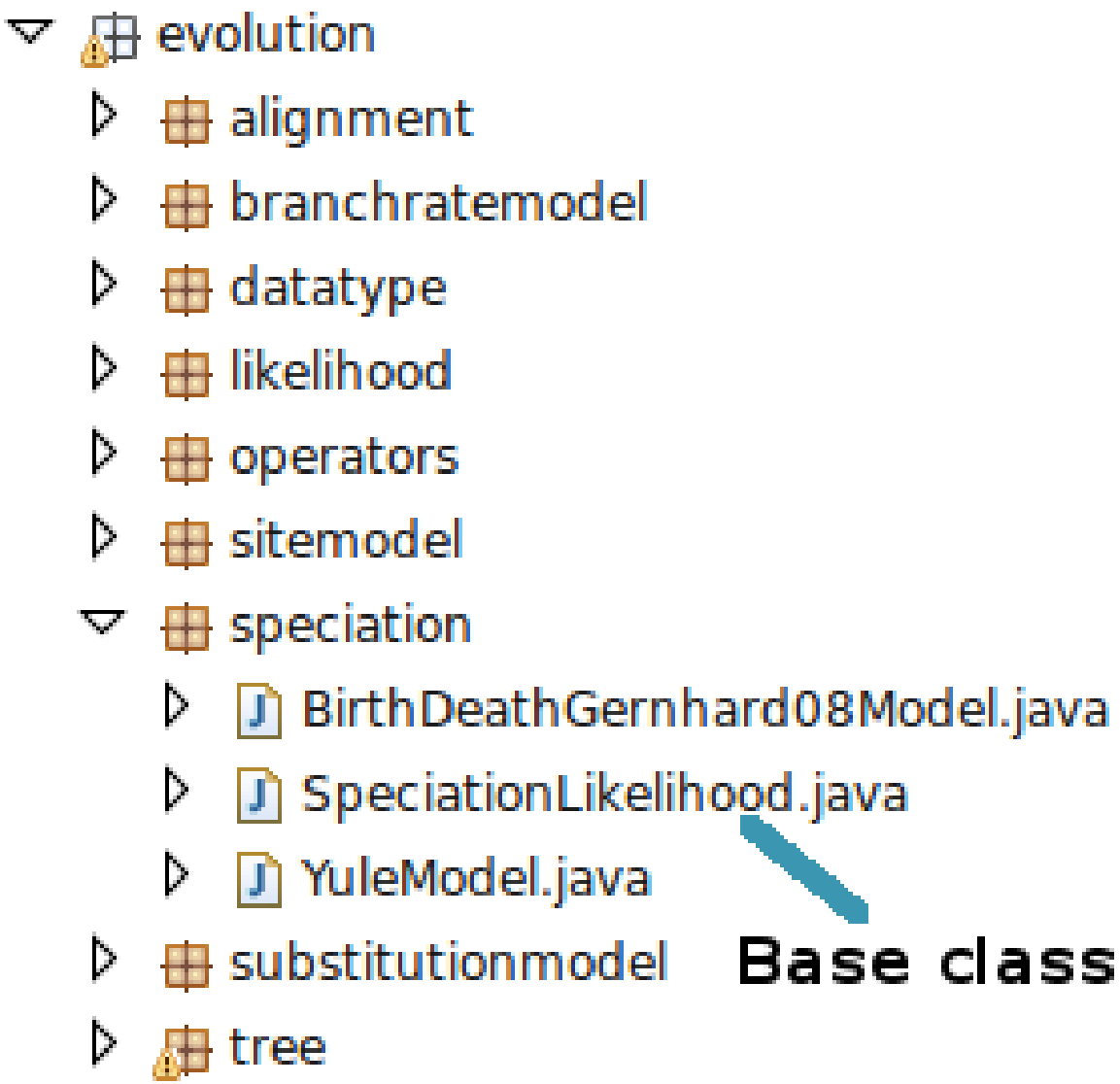
Evolution - operator classes



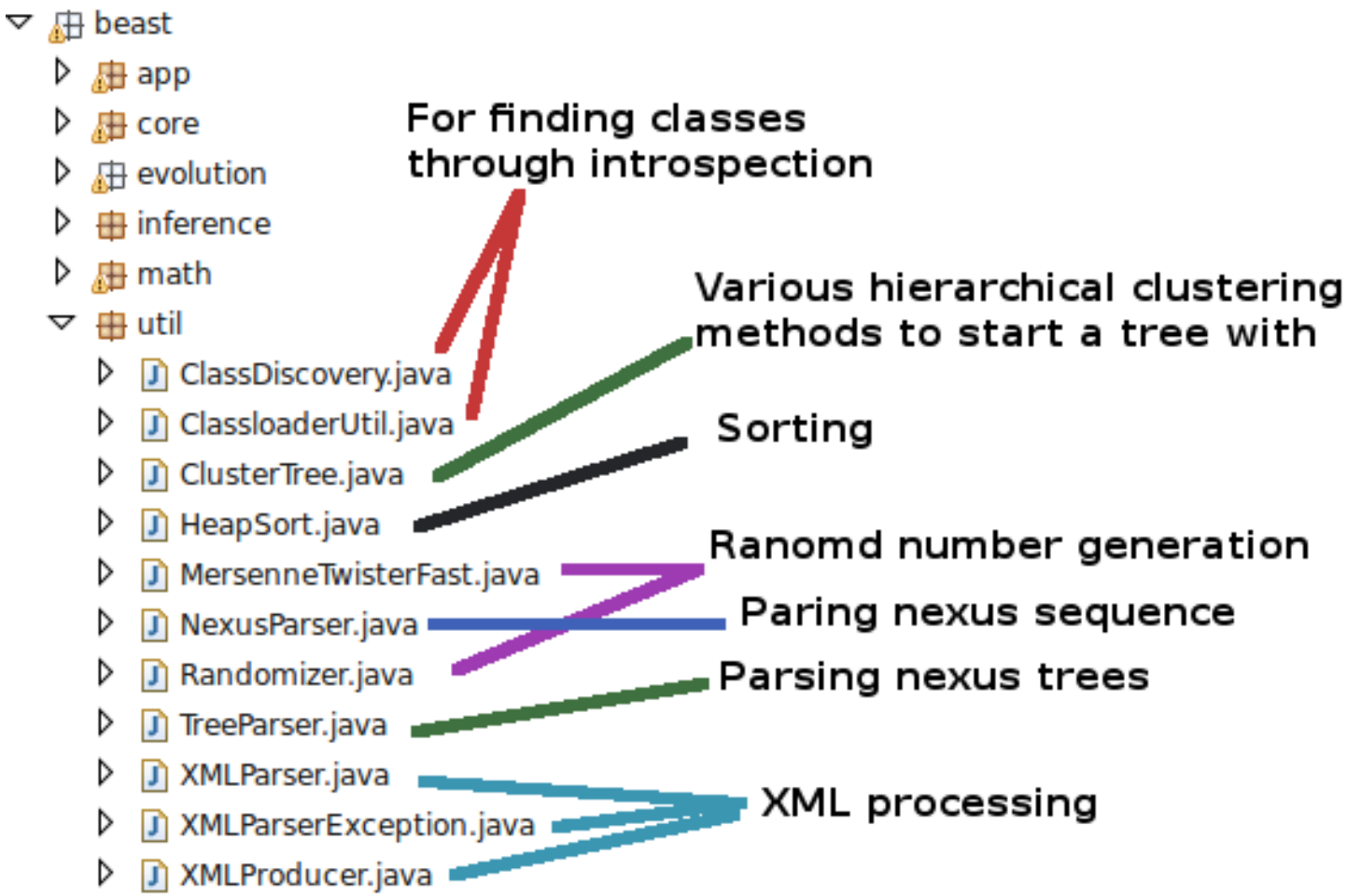
Evolution - site model classes



Evolution - speciation classes



Utility classes



Variable naming

In case you wondered where those funny names came from...

Variable name format: `<scope><type><name>`

scope

- `m_` prefix for member variables
- `g_` globals = static member variables
- none otherwise

type

- `s` string
- `f` floating point number (double or float)
- `n` number
- `i` indicator
- `b` boolean
- `p` pointer to object

Basic plugin layout

```
@Description("Some sensible description of the Plugin")
public class MyPlugin extends Plugin {
    <!-- inputs first -->
    public Input<RealParamater> m_p = new Input<>...;

    <!-- members next -->
    private Object m_o;

    <!-- initAndValidate -->
    @Override
    public void initAndValidate() {...}

    <!-- class specific methods -->

    <!-- Overriding methods -->
}
```

Accessing inputs *for reading, not writing!*

Let there be an input:

```
public Input<RealParamater> m_p = new Input<>...;
```

To get the parameter of input `m_p`, use `m_p.get()`.

To get the value of the parameter, use `m_p.get().getValue()`.

Alternatively

```
RealParamater p = m_p.get();
double fValue = p.getValue();
```

requiresRecalculation()

```
public boolean requiresRecalculation() {
        // for StateNode inputs only
        if (m_stateNodeInput.get().somethingIsDirty()) {
                return true;
        }

        // for CalculationNode inputs only
        if (m_calculationNodeInput.get().isDirtyCalculation()) {
                return true;
        }
        return false;
}
```

Lean CalculationNode

```
boolean m_bNeedsUpdate; // flag to indicate internal state is up to date
public void initAndValidate() {m_bNeedsUpdate = true;}
// CalculationNode specific interface that returns results
public Object calculateSomeThing() {
        if (m_bNeedsUpdate) {
                update();
        }
        return someThing;
}
void update() {
        someThing = ...;
        m_bNeedsUpdate = false;
}
public boolean requiresRecalculation() {
        if (someInputIsDirty()) {
                m_bNeedsUpdate = true;
                return true;
        }
        return false;
}
public void store() {super.store();}
public void restore() {
        m_bNeedsUpdate = true;
        super.restore();
}
```

Fat CalculationNode

As lean CalculationNode, but actually storing something

```
Object m_intermediateResult;
Object m_storedIntermediateResult;

public void initAndValidate() {
        // reserve space for result objects
        m_intermediateResult = new ...;
        m_storedIntermediateResult = new ...;
}

public void store() {
        // copy m_intermediateResult to m_storedIntermediateResult
        ...
        super.store();
}
public void restore() {
        // m_bNeedsUpdate = true; <- don't need this now
        Object tmp = m_intermediateResult;
        m_intermediateResult = m_storedIntermediateResult;
        m_storedIntermediateResult = tmp;
        super.restore();
}
```

Adding a Substitution model

Extend SubstitutionModel.Base class

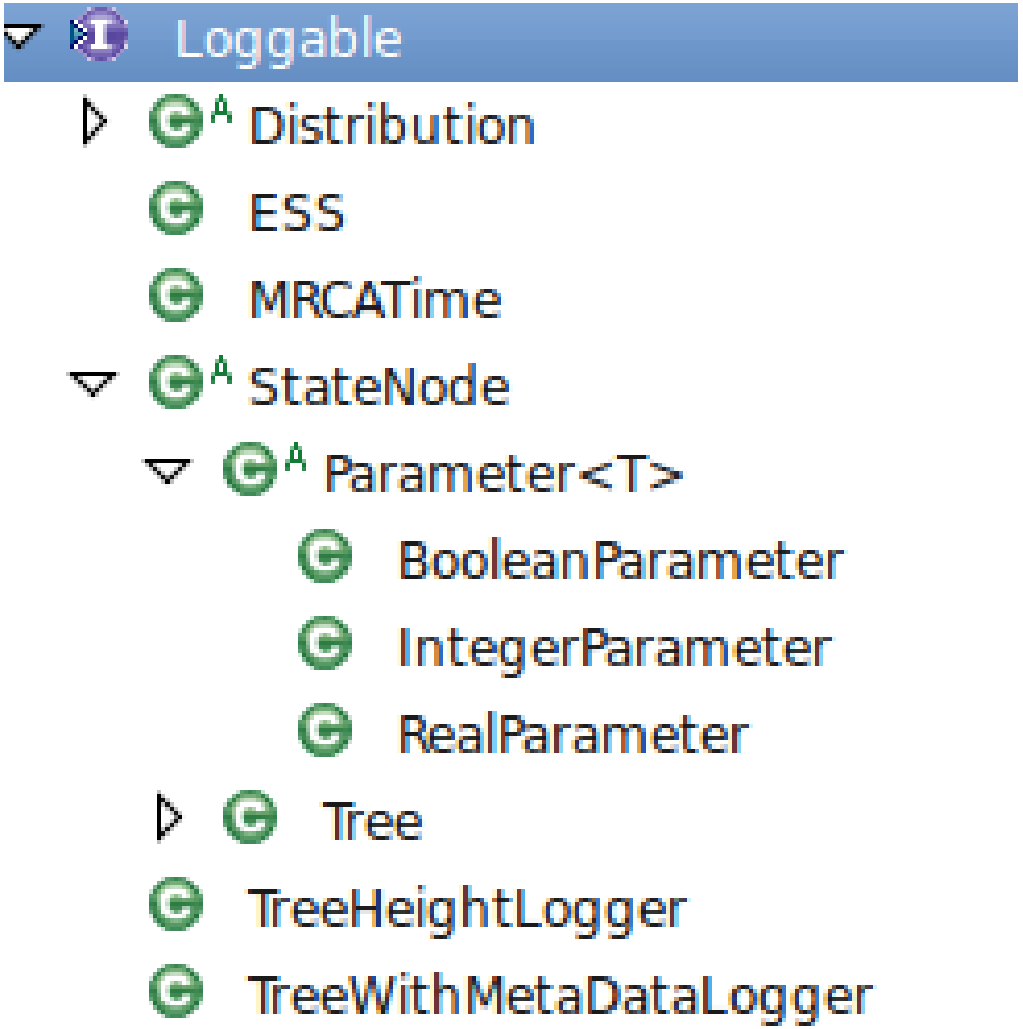
- A substitution model should implement `getTransitionProbabilities(Node node, double fStartTime, double fEndTime, double fRate, double[] matrix)`
- Typically, `fRate * (fEndTime - fStartTime)` is the distance t in e^{Qt} and `Node` can be ignored.
- Results should go in the `matrix`: note this is represented as array.
- `SubstitutionModel` is a `CalculationNode`, so it may be worth implementing `store/restore/requireRecalculation`

Adding an Operator: Extend Operator class

- An operator should have at least one input with a `StateNode` to operate on.
- An operator should implement `proposal()` which changes the State.
- `proposal()` should return the Hastings ratio.
- Return `Double.NEGATIVE_INFINITY` if the proposal is invalid/doomed (don't throw Exceptions).
- Implement `optimize()` if auto-optimization applies.

Adding a logger: Implement Loggable interface

```
void init(PrintStream out) throws Exception;

void log(int nSample, PrintStream out);

void close(PrintStream out);
```



Input rule of base class is not what you want.

If an Input is REQUIRED for a base class you want to override, but for the derived class this Input should be OPTIONAL, set the Input to OPTIONAL in the constructor. E.g. for a SNPSequence that derives from Sequence, but for which m_sData is optional, add a constructor

```
public SNPSequence() {
        m_sData.setRule(Validate.OPTIONAL);
}
```

Note that the constructor needs to be public, to prevent IllegalAccessExceptions on construction by e.g. the XMLParser.

Input parameter dimension is unknown...

...but a CalculationNode can easily find out.

Then, in the `initAndValidate()` method of the `CalculationNode`, call `parameter.setDimension(x)`

Input parameter *value* is unknown...

...but a CalculationNode can easily find out.

Then, in the `initAndValidate()` method of the `CalculationNode`, create a new Parameter X, and use `input.get().assignFromWithoutID(X)`

```
@Override
public void initAndValidate() throws Exception {
    // determine dimension, number of Nodes in a tree here
        int nNodes = m_tree.get().getNodeCount();

    // create new Parameter
        IntegerParameter positions = new IntegerParameter("0", 0, Integer
        for (int i = 0; i < nNodes; ++) {
                int iPosX = ...;
                positions.setValue(i, iPosX);
        }

    // copy values to the input
        m_positions.get().assignFromWithoutID(positions);

}
```

Trees with traits

For a tree with n leaf nodes, so $2n - 1$ nodes in total

- Easiest: associate a parameter with dimension $2n - 1$ to the tree
  - Leaf nodes are numbered $0, \ldots, n - 1$
  - Internal nodes are numbered $n, \ldots, 2n - 1$
  - Root node is not treated as special internal node (no number guaranteed)
- Harder: Derive from class `Node` and process as meta-data

Common errors

1. `Input` **is not declared public.**

If `Input`s are not public, they cannot get values assigned by for instance the `XMLParser`.

Common errors

1. `Input` **is not declared public.**
If `Input`s are not public, they cannot get values assigned by for instance the `XMLParser`.

2. **Type of input is a template class (other than** `List`**).**
Thanks to limitations of Java introspection and the way Beast II is set up, Inputs should be of a type that is concrete, and apart from `List<T>` no template class should be used.

Common errors

1. `Input` **is not declared public.**
If `Input`s are not public, they cannot get values assigned by for instance the `XMLParser`.

2. **Type of input is a template class (other than** `List`**).**
Thanks to limitations of Java introspection and the way Beast II is set up, Inputs should be of a type that is concrete, and apart from `List<T>` no template class should be used.

3. **Store/restore do not call** `super.store()`/`super.restore()`**.**
Obviously, not calling store/restore on super classes may result in unexpected behavior.

Uncommon errors

- Forgot to set `m_bNeedsUpdate` flag in `requiresRecalculation()`. This way the `CalculationNode` will never update its internal state and will always return the same (initially calculated) result.
- always return `false` from `requiresRecalculation()`. This way `CalculationNode`s downstream may never think of recalculating themselves.

Both issues will not be picked up during the debugging phase of the MCMC loop.