Blueprint

Flexible specification and construction of FlexiNet binders

Peter Bagnall

Purpose

- To provide a generic way of specifying binders
- Allow a “default” binder to be easily adapted
- Examples may be changing buffer sizes, low-level protocols, serializers
- Builds object graphs in a well defined manner, so could be used for other elements which need to be flexibly constructed
- Ultimately, to simplify writing binders

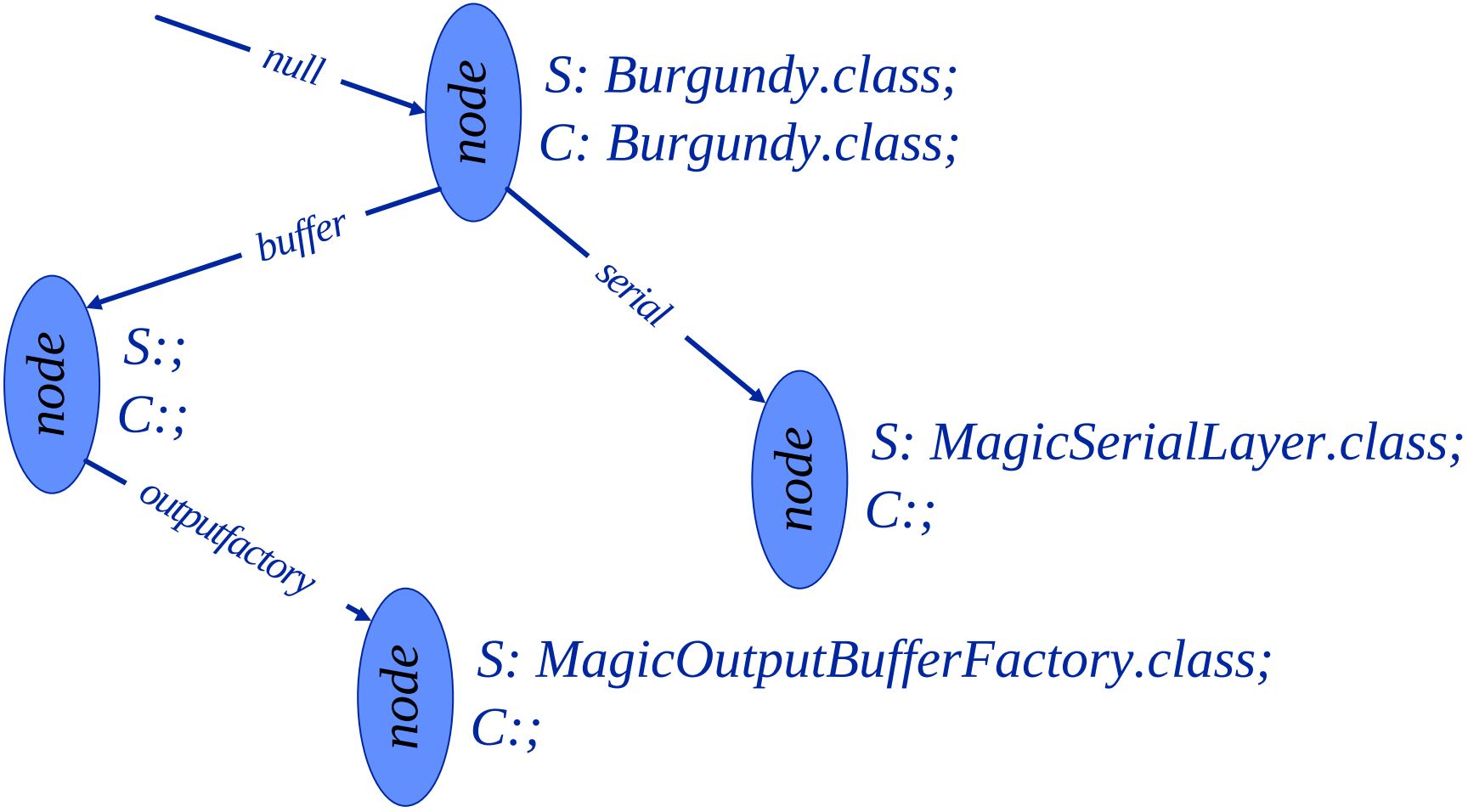
Brief Overview (1)

- Evolved from FlexiProps
- Blueprints are hierarchical structures (plus symbolic links)
- Each node has multiple suggestions
- Each node has multiple constraints
- Resolution decides which suggestions to take to meet the constraints, or reports failure if not possible

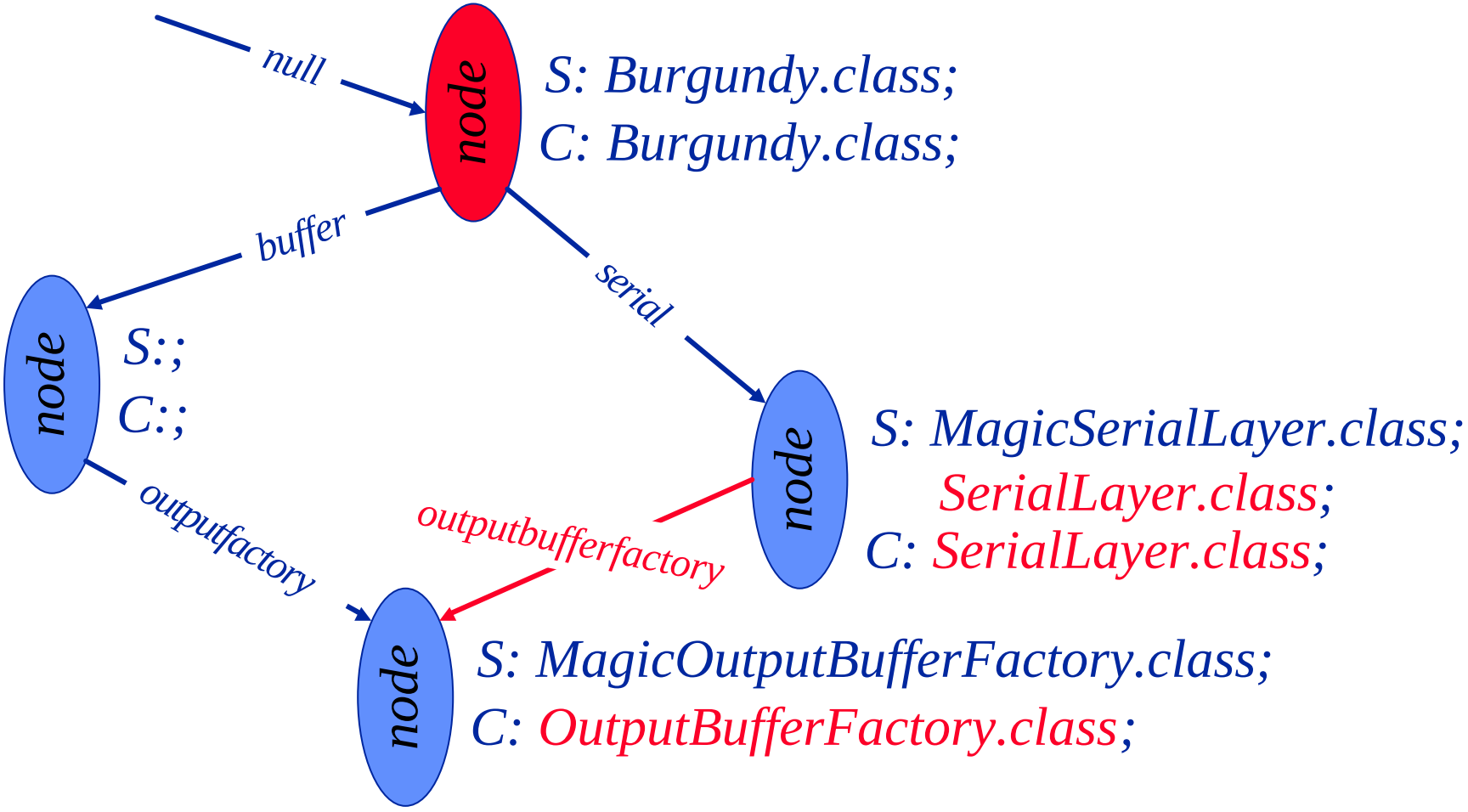
Brief Overview (2)

- Resolved Blueprint may be recorded for later use
- It can then be “constructed” which builds any instances required and initialises all objects, using the Blueprint as the source of the initialisation parameters
- Finally the constructed object(s) can be obtained for use

A Blueprint for Burgundy



Implied Specifications

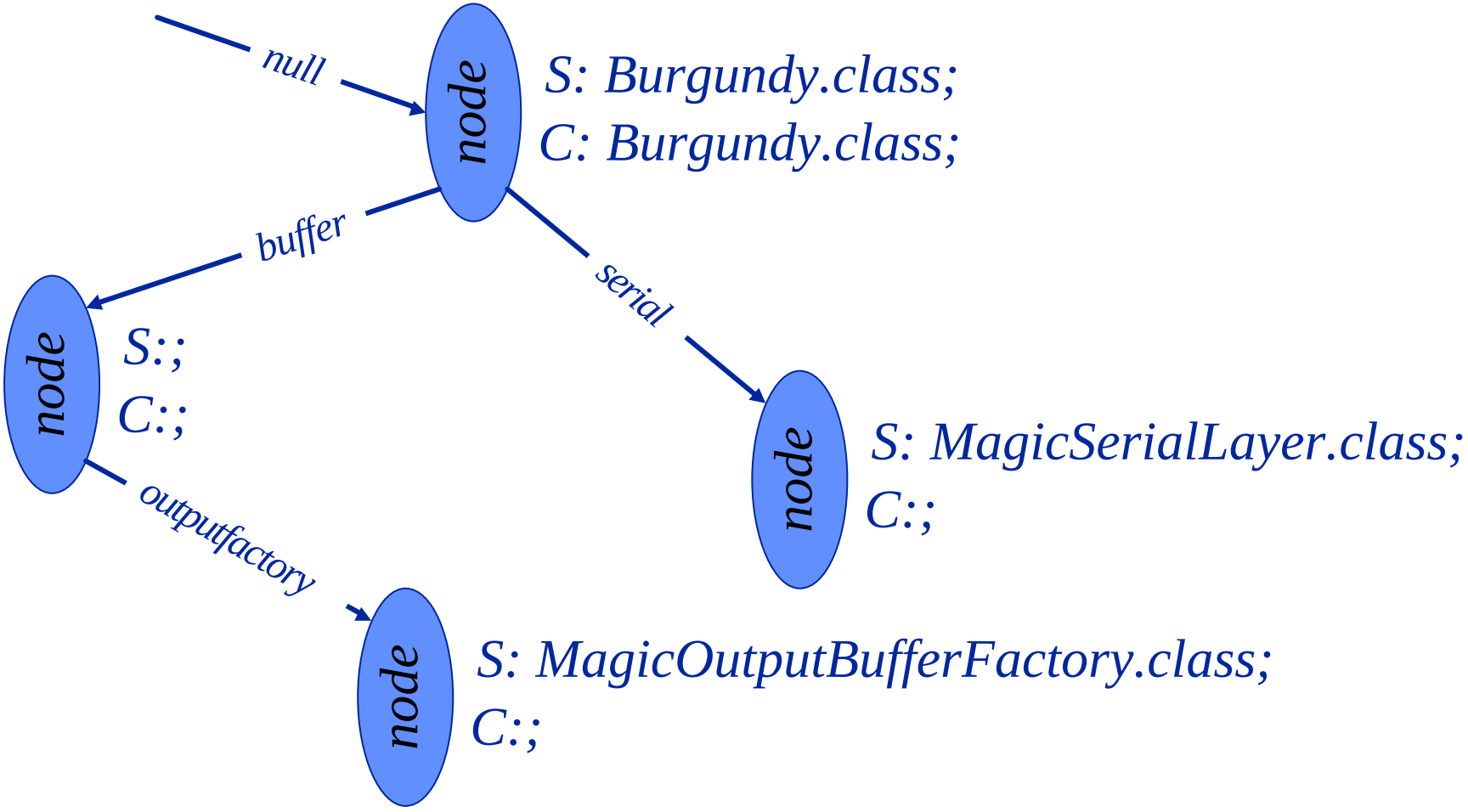


Example - Building a Burgundy

```
Blueprint bp=new Blueprint();
bp.require(null,Burgundy.Class);
bp.suggest("serial",MagicSerialLayer.class);
bp.suggest("buffer.outputfactory",
                  MagicOutputBufferFactory.class);
bp.construct();
Binder binder=(Binder)bp.get(null);
```

And then **`binder`** is usable

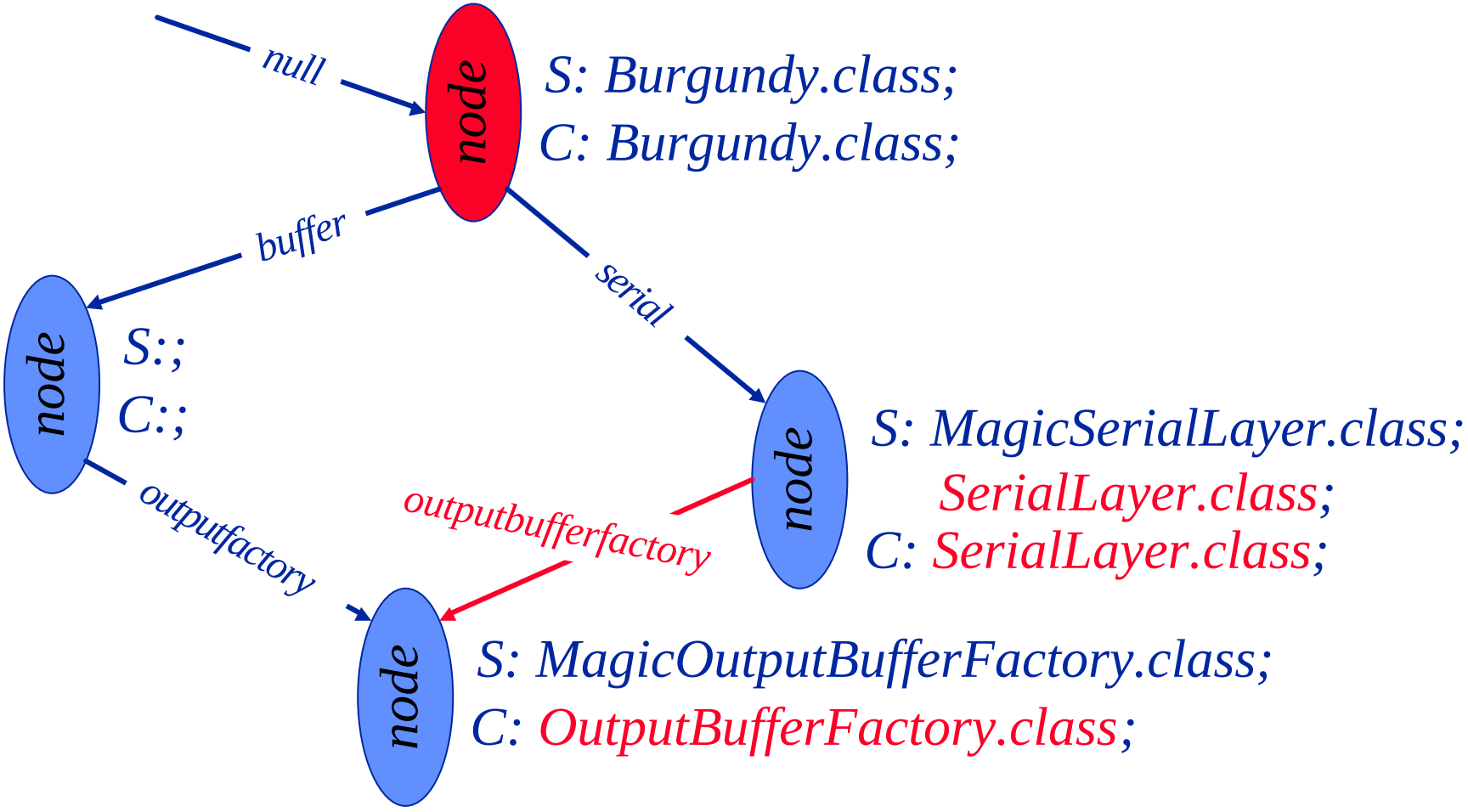
A Blueprint for Burgundy



Inside Burgundy

```
setRequirements(Blueprint b) {
   b.require("serial",                SerialLayer.Class);
   b.link(“.outputbufferfactory”,”buffer.outputfactory”);
   ...
   b.constrain(”buffer.outputfactory",
                              OutputBufferFactory.Class);
   ...
}
```

Implied Specifications



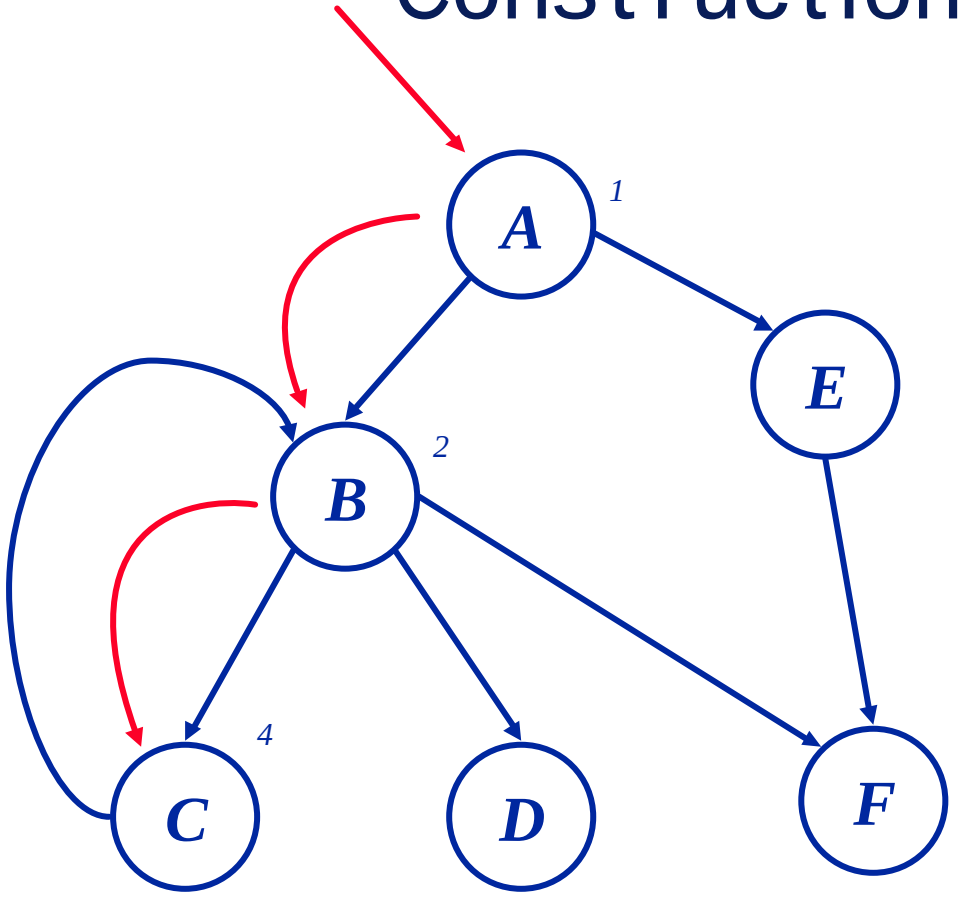
Construction - Resolution

- Starting at the root the tree is walked. At each node…
  - a suggestion is selected, (in the order they were set)
  - if is conflicts with any constraints then next suggestion is tried
  - if the suggestion is a “Resolving class” it is asked to specify any suggestions or constraints on its children.
    - To do this the class must implement the static method setRequirements(Blueprint bp)
    - c.f. - Java Beans properties - set methods.

Construction - Resolution

- It then attempts to resolve its first child node.
- When the child completes it calls its parent's "complete" method if successful, or returns with a failure flag if unsuccessful.
- If successful the parent then calls the next child, and so on.
- Eventually the root receives a complete call, and has no more work to do, in which case the call returns, which causes the stack to unwind.

Construction - Resolution



Starting position

A = 1
B = 2 | 3
C = 4 | 5
D = 6
E = 7
F =

Current state

A = 1
B = 2
C = 4

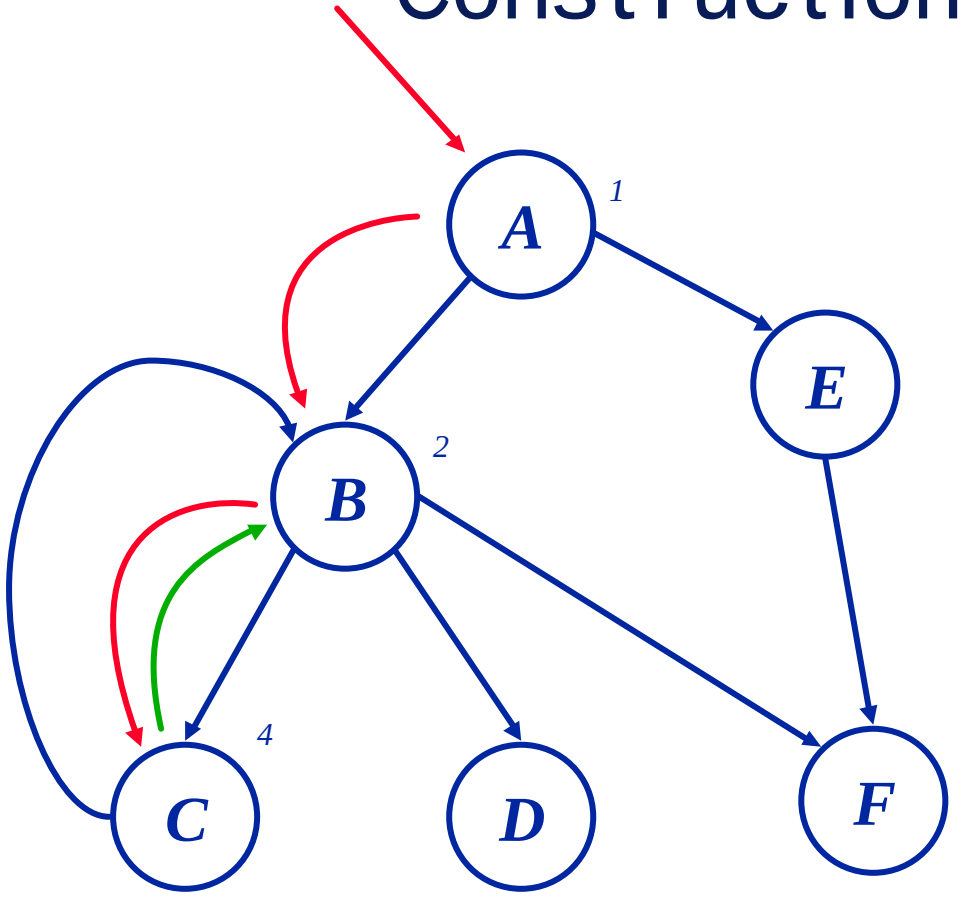
Implied constraints

4 ⟶ B ≠ 2
2 ⟶ F ≠ 8
7 suggests F = 8

resolve →

complete →

Construction - Resolution



Starting position

A = 1
B = 2 | 3
C = 4 | 5
D = 6
E = 7
F =

Current state

A = 1
B = 2 but 4 ⟶ B ≠ 2 fail
C = 4

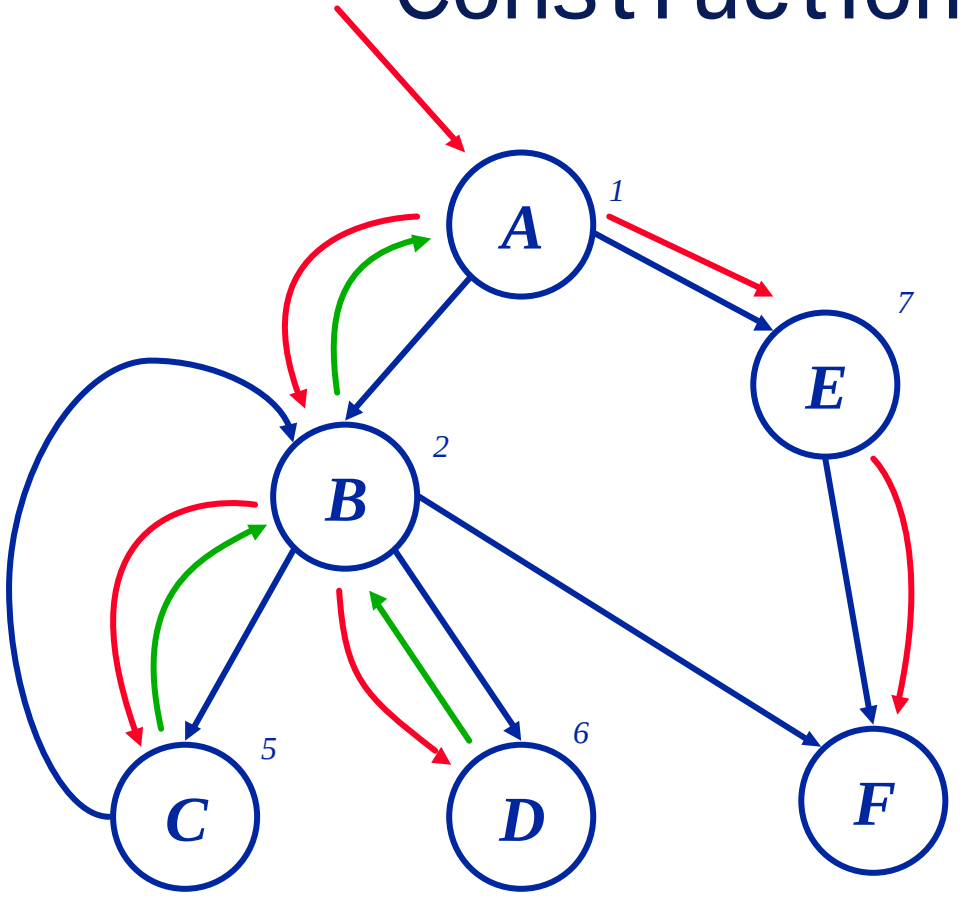
Implied constraints

4 ⟶ B ≠ 2
2 ⟶ F ≠ 8
7 suggests F = 8

resolve

complete

Construction - Resolution



Starting position

A = 1
B = 2 | 3
C = 4 | 5
D = 6
E = 7
F =

Current state

A = 1
B = 2
C = 5
D = 6
E = 7
F = 8 but 2 ⟶ F ≠ 8 fail

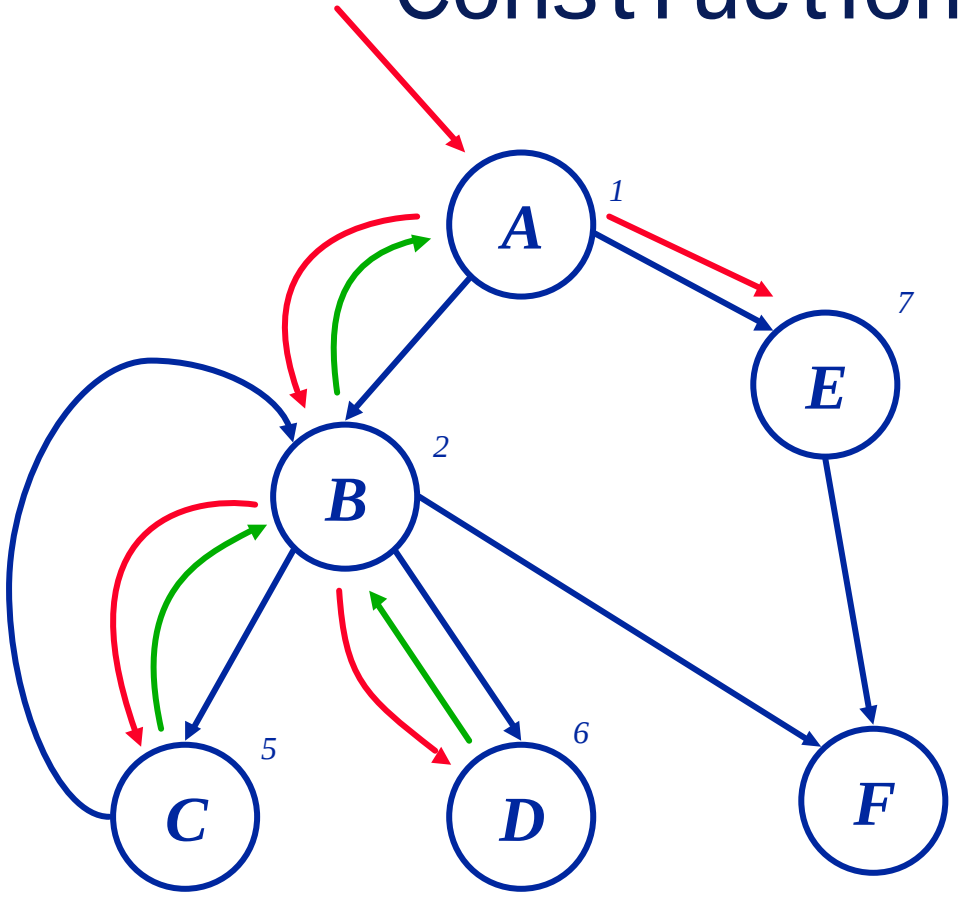
Implied constraints

4 ⟶ B ≠ 2
2 ⟶ F ≠ 8
7 suggests F = 8

resolve

complete

Construction - Resolution



Starting position

A = 1
B = 2 | 3
C = 4 | 5
D = 6
E = 7
F =

Current state

A = 1
B = 2
C = 5
D = 6
E = 7

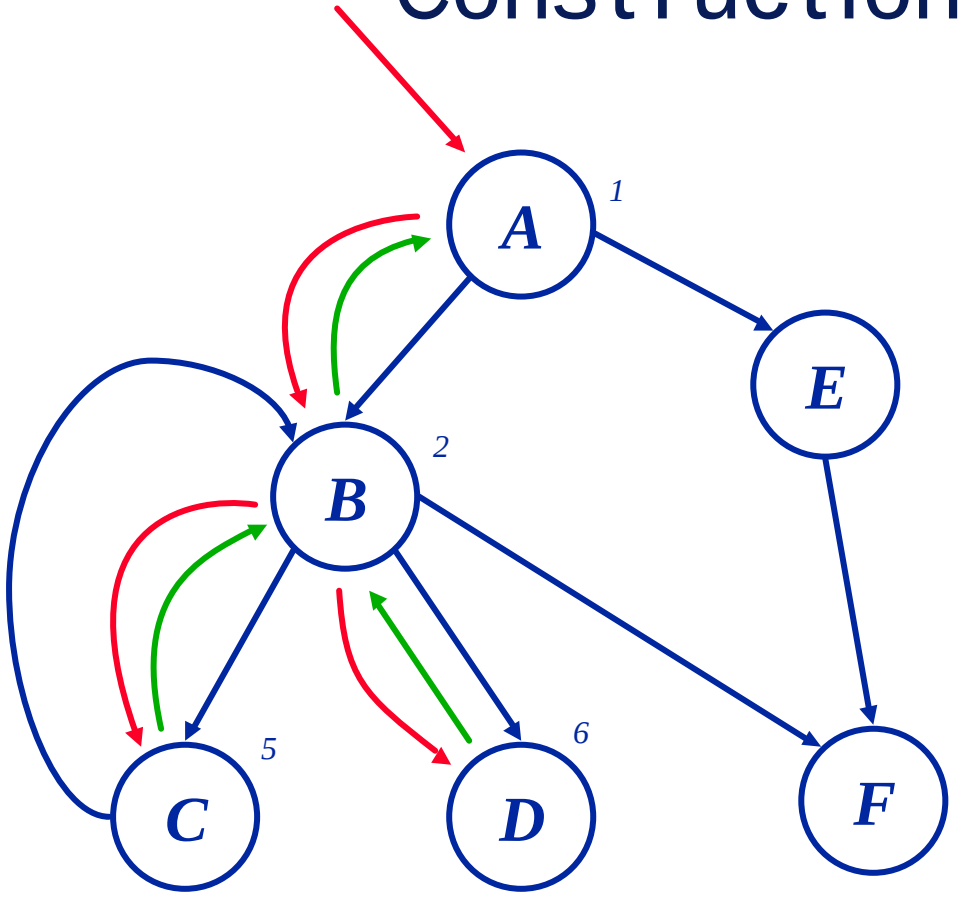
Implied constraints

4 ⟶ B ≠ 2
2 ⟶ F ≠ 8
7 suggests F = 8

resolve

complete

Construction - Resolution



Starting position

A = 1
B = 2 | 3
C = 4 | 5
D = 6
E = 7
F =

Current state

A = 1
B = 2
C = 5
D = 6

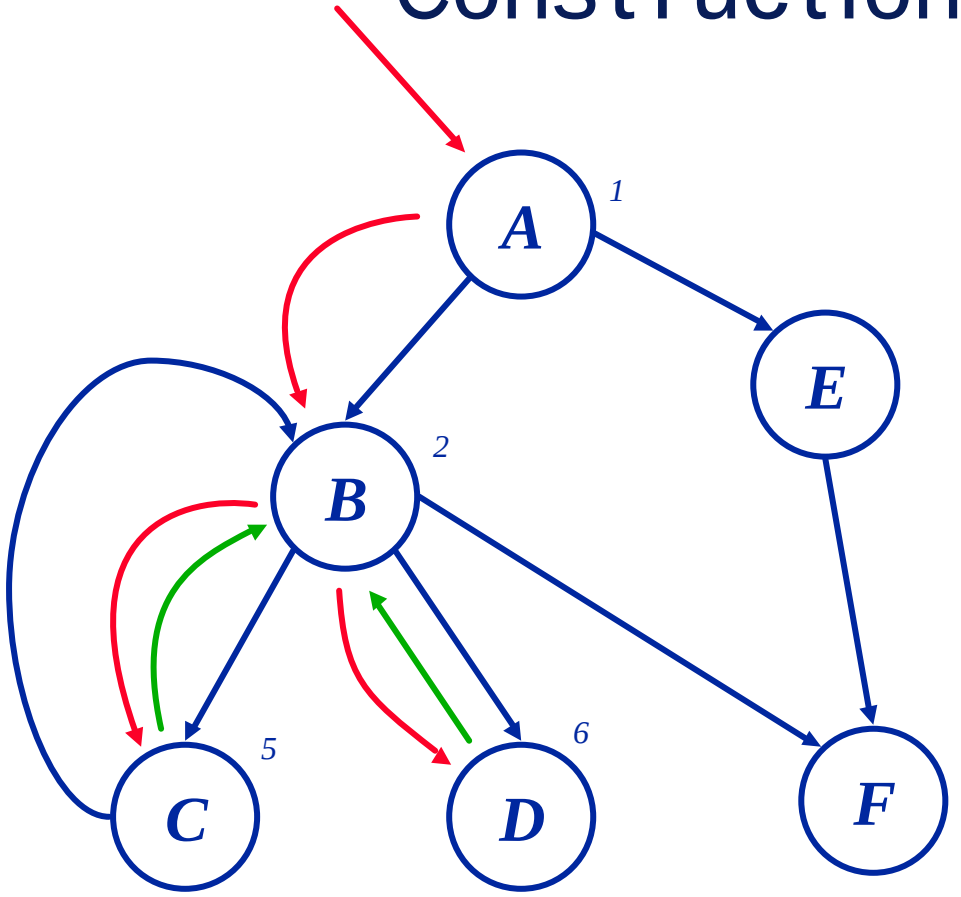
Implied constraints

4 ⟶ B ≠ 2
2 ⟶ F ≠ 8
7 suggests F = 8

resolve

complete

Construction - Resolution



Starting position

A = 1
B = 2 | 3
C = 4 | 5
D = 6
E = 7
F =

Current state

A = 1
B = 2
C = 5
D = 6

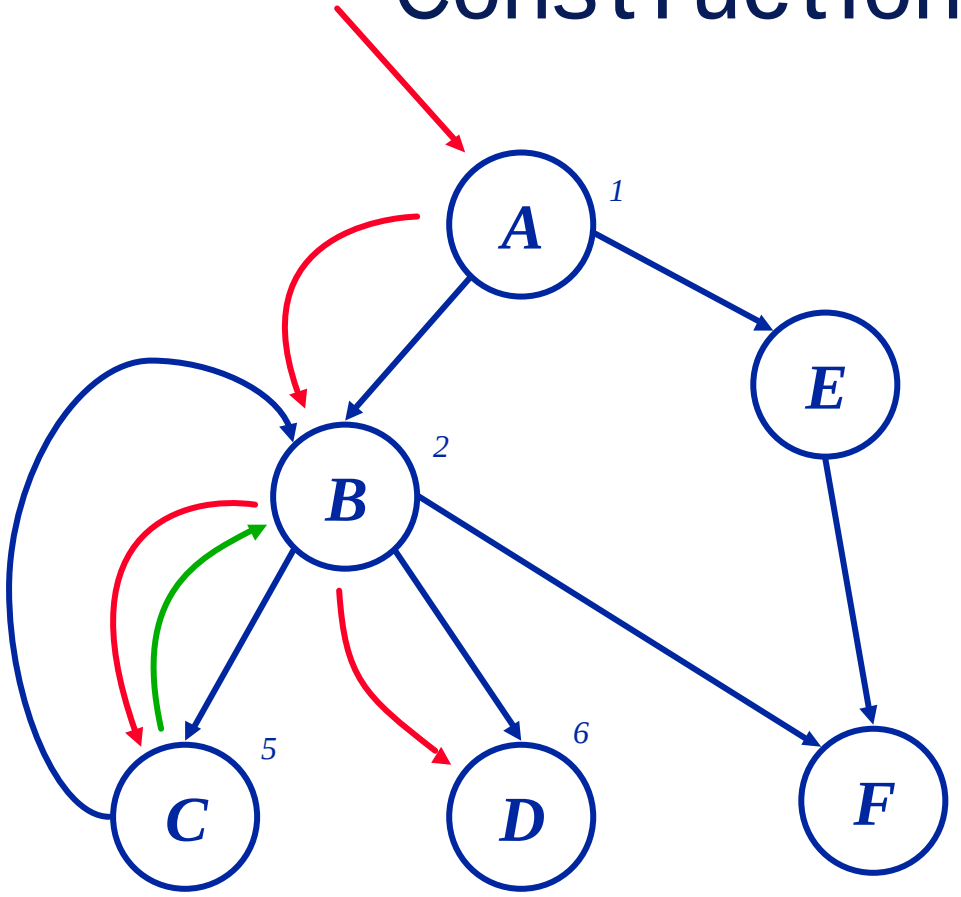
Implied constraints

4 ⟶ B ≠ 2
2 ⟶ F ≠ 8
7 suggests F = 8

resolve

complete

Construction - Resolution



Starting position

A = 1
B = 2 | 3
C = 4 | 5
D = 6
E = 7
F =

Current state

A = 1
B = 2
C = 5
D = 6

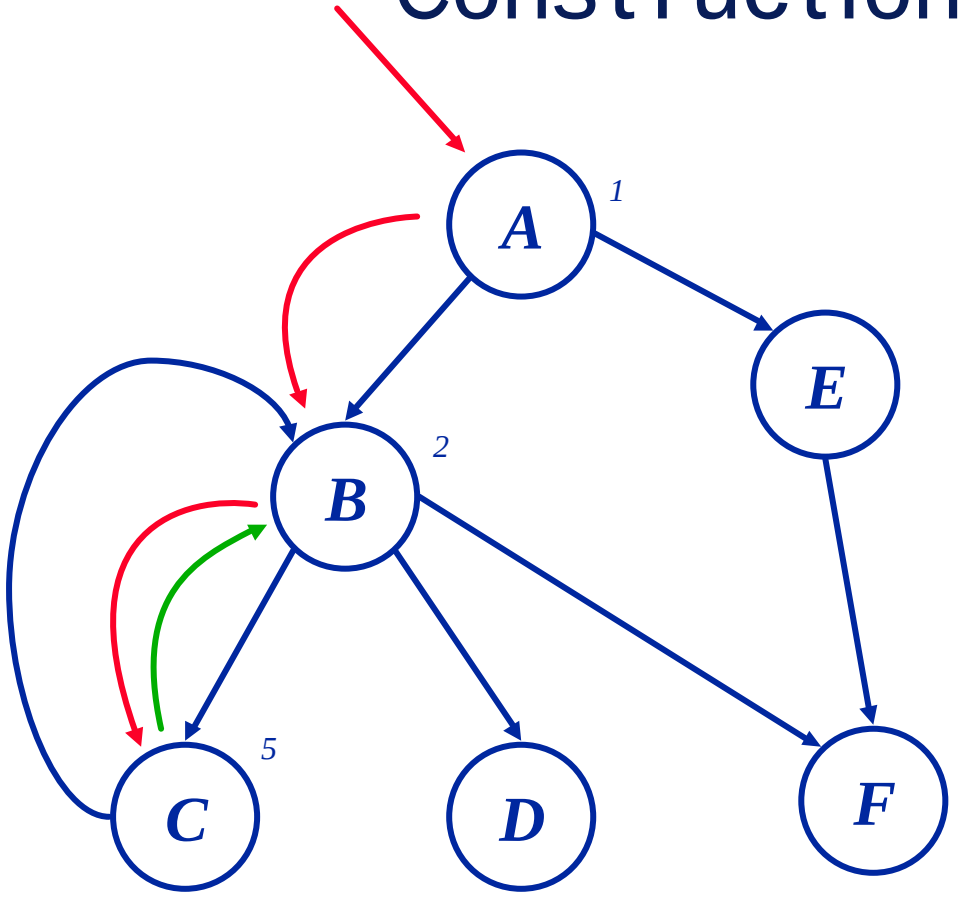
Implied constraints

4 ⟶ B ≠ 2
2 ⟶ F ≠ 8
7 suggests F = 8

resolve

complete

Construction - Resolution



Starting position

A = 1
B = 2 | 3
C = 4 | 5
D = 6
E = 7
F =

Current state

A = 1
B = 2
C = 5

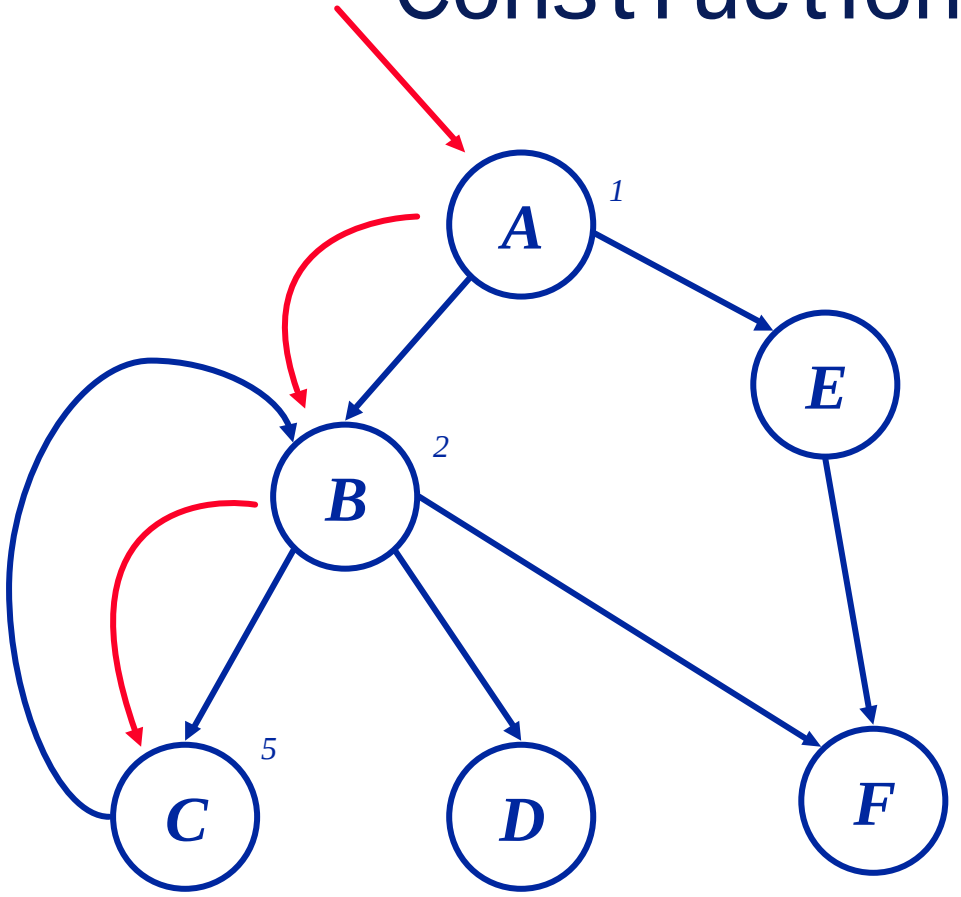
Implied constraints

4 ⟶ B ≠ 2
2 ⟶ F ≠ 8
7 suggests F = 8

resolve

complete

Construction - Resolution



Starting position

A = 1
B = 2 | 3
C = 4 | 5
D = 6
E = 7
F =

Current state

A = 1
B = 2
C = 5

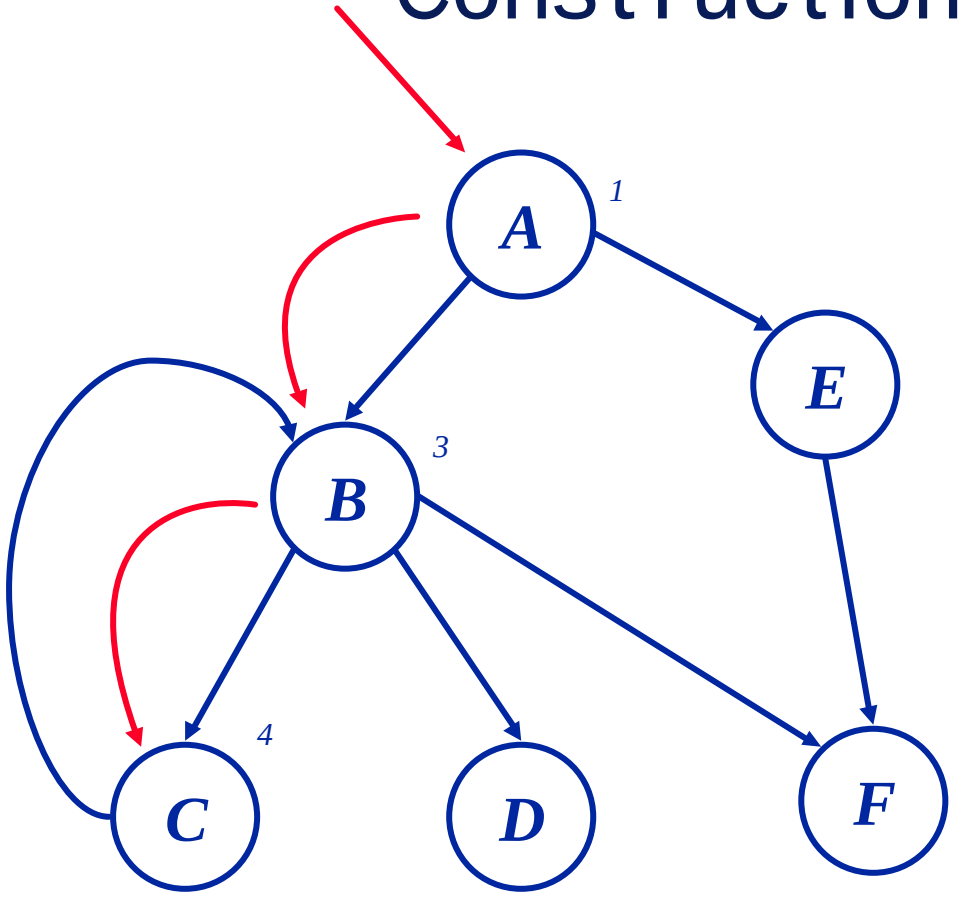
Implied constraints

4 ⟶ B ≠ 2
2 ⟶ F ≠ 8
7 suggests F = 8

resolve →

complete →

Construction - Resolution



Starting position

A = 1
B = 2 | 3
C = 4 | 5
D = 6
E = 7
F =

Current state

A = 1
B = 3
C = 4

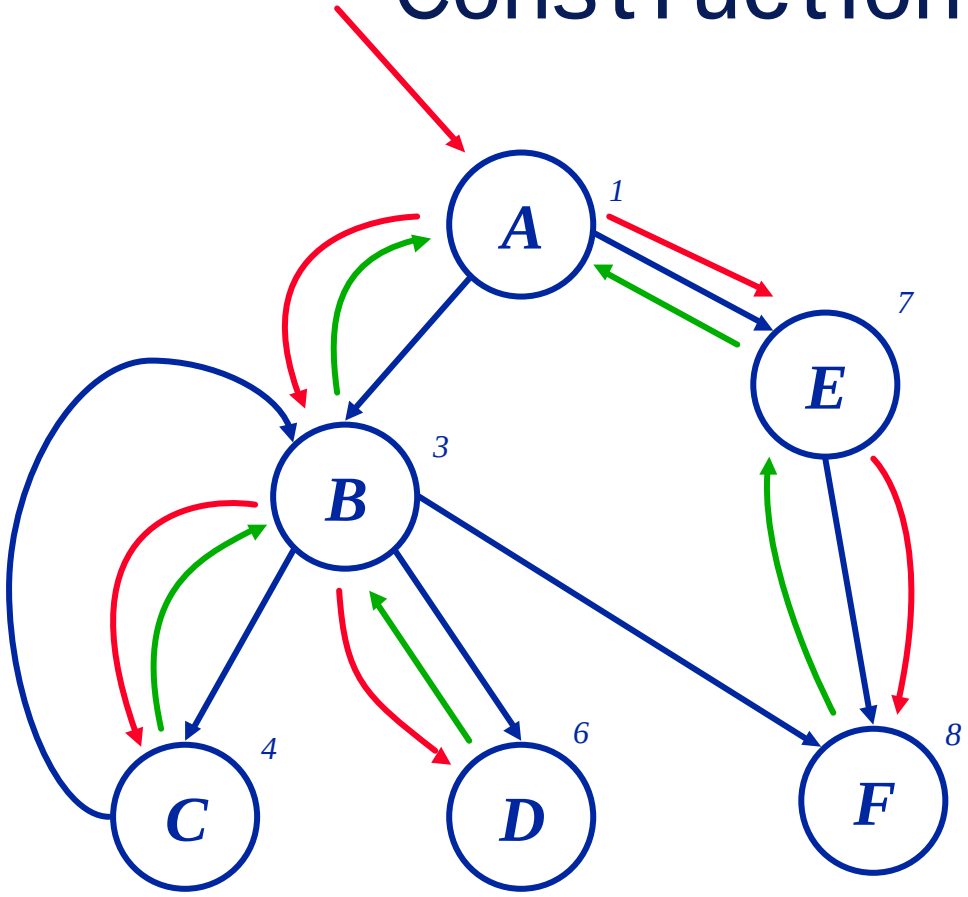
Implied constraints

4 ⟶ B ≠ 2
2 ⟶ F ≠ 8
7 suggests F = 8

resolve

complete

Construction - Resolution



Starting position

A = 1
B = 2 | 3
C = 4 | 5
D = 6
E = 7
F =

Current state

A = 1
B = 3
C = 4
D = 6
E = 7
F = 8

Implied constraints

4 ⟶ B ≠ 2
2 ⟶ F ≠ 8
7 suggests F = 8

resolve

complete

Construction - Resolution

- In the case of contention
  - the current suggestion is removed, including any constraints it applied (but not suggestions).
  - The next suggestion is then tried.
  - When the node runs out of suggestions it rolls back the previous node, and tries its next suggestion.
  - Rollback continues until the suggestion works, (success), or the root node runs out of suggestions (failure).

Construction - Resolution

- If contention reaches the root, and the root runs out of suggestions the construction fails and the Blueprint throws an exception.
- In this case the programmer may switch on debugging to trace the various decisions that are being made, to try and find the conflicts.

Recording a Blueprint

- Once resolution is complete the Blueprint may be recorded to an OutputStream.
- Potentially a database or cache of Blueprints could be maintained which could be used to speed up resolution.

Instantiation

- Having resolved which classes are to be used each “Resolving class” is instantiated and replaced by the instance.
  - Instantiation is equivalent to the null constructor
    - Resolvable classes must implement “createUninitialised()” which returns an instance.
    - c.f. - Java Beans - must implement null constructor

Initialisation

- Once all resolving classes are instantiated the tree is walked leaf-root initialising instances.
- At each node
  - the instance must implement initialise(Blueprint bp);
  - it may return three states, “complete”, “progressed”, “no progress”.
- This is repeated until all nodes are satisfied they have completed or stalemate occurs.
  - Stalemate occurs when no nodes are progressed

Interface - creating (1)

```
public    Blueprint()
public    Blueprint(File file)
public    Blueprint(Object o, Constraints con)

public void merge(String name, Blueprint bp)

public void suggest(String name, Object value)
public void suggest(String name, int i)
public void suggest(String name, Class cls)

public void link(String from, String to) {}
```

Interface - creating (2)

```
public void constrain(String name, Constraints c)
                                    throws ConstraintContentionException
public void constrain(String name, Class c)
                                    throws ConstraintContentionException


public void require(String name, Object value)
                                    throws ConstraintContentionException
public void require(String name, int i)
                                    throws ConstraintContentionException
public void require(String name, Class cls)
                                    throws ConstraintContentionException
```

Interface - shorthand

- The Blueprint records the last name set, and uses it as a default for the next call, e.g.
  - suggest(“serial”,SerialLayer.class);
  - suggest(“.serializer”,Serializer.class);
- Is exactly equivalent to
  - suggest(“serial”,SerialLayer.class);
  - suggest(“serial.serializer”,Serializer.class);
- Either may be used. Note that in the first the order of calls matters, but typo’s are less likely

Interface - reading

```
public Object get(String name)
public int getInteger(String name)   throws ClassCastException

public Enumeration suggestions(String name)
public Enumeration constraints(String name)

public Blueprint getSubBlueprint(String name)
```

Interface - construction

```
public void resolve()
public void instantiate()
public void initialise()
public void construct()
```

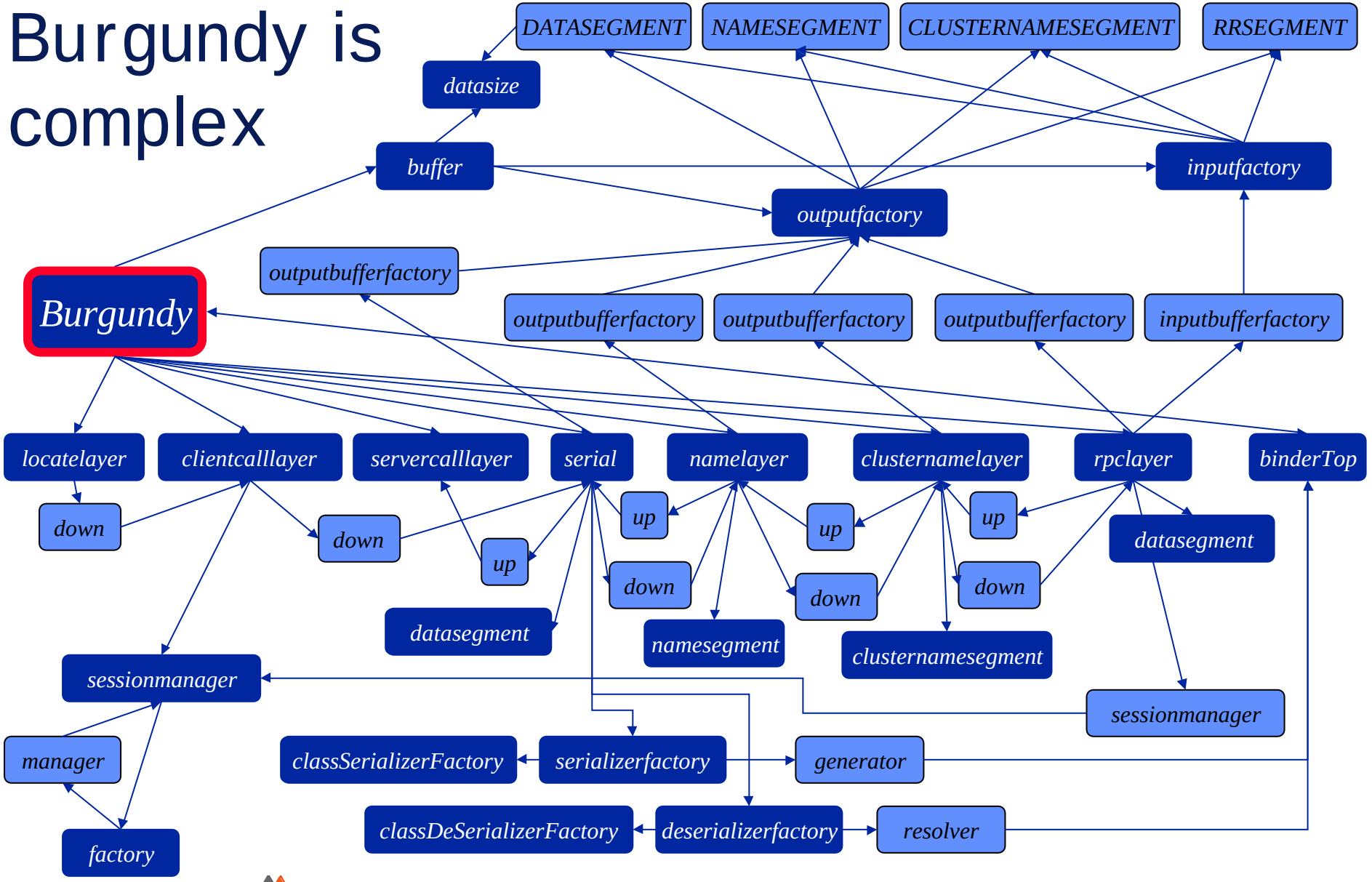
Interface - miscellaneous

```
public void addChangeListener(ChangeListener cl)
public void removeChangeListener(ChangeListener cl)

public void toString()
public void write(OutputStream os)

public Object copy()
public Enumeration propertyNames()
public Enumeration children()
```

Possible Further Work

- Beanification
  - components should be beans
  - use “bean properties” via introspection to determine what the children of a node should be, allows better error detection
  - build a visual tool for creating Blueprints
- More flexible constraint classes
  - boolean combinations etc.

Burgundy is complex



Blueprint

That's all folks.

Peter Bagnall