



Binary XML with BiM



■ Generic

- Works for any XML language

- Adopted by international standards
- Evaluated on many XML languages

MPEG-7, TV-Anytime, ...

SVG, SMIL, XHTML, NewsML, ...

■ High Compression

- Average compression ratio

85%

- Up to 1/200 (SOAP requests)

- High compression ratio of the structure

- 98% - Highly structured documents for no cost !


■ High parsing speed


- Parsing done at the binary level


x 10

- Up to 30 times



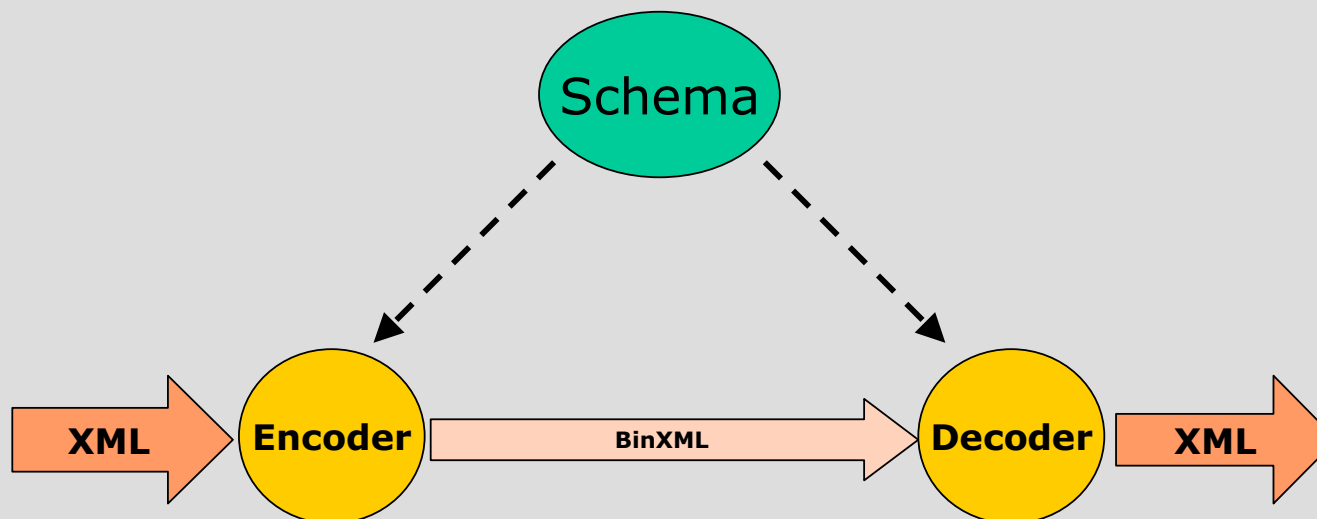
- **Pre-parsed, pre-validated** 
 - Validation and parsing done at encoding
 - ➔ Very fast decoding, PSVI reconstruction

- **Customizable** 
 - Dedicated codecs can be plugged in
 - ➔ Compression improvements, encryption, authentication

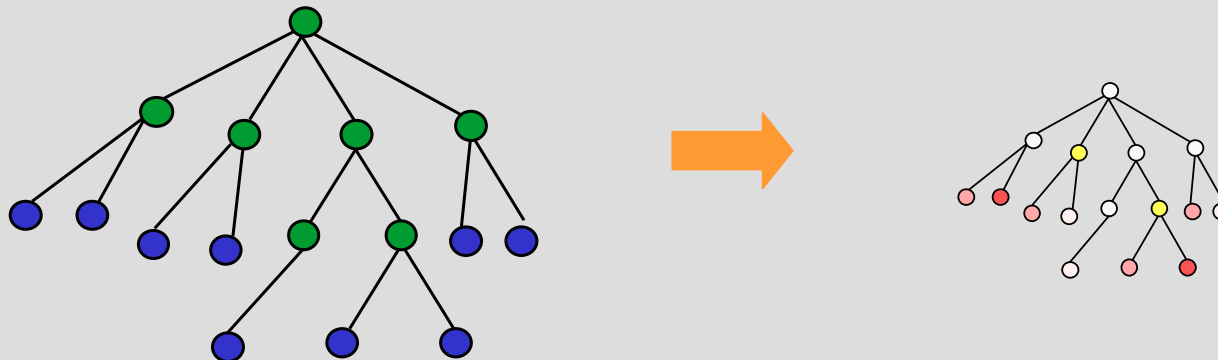
- **Partial decoding** 
 - The parsing of some document parts can be skipped
 - ➔ Improve filtering speed, large document parsing



- Automatically generates the syntax of the binary format
 - No need to develop both encoder and decoder (always in phase)
 - No need to design a specific binary format
 - Easy management of XML language evolution
- ➔ Dramatically reduces development cost



- **Standard compression methods**
 - For structure >>
 - Automaton based : simple, efficient, scalable
 - ➔ Enables validation
 - For data >>
 - Compression : Statistical, Quantization, Dictionary, ...
 - Encoding scheme: IEEE-754, UTF-8, UTF-16, ...



- **Extensibility**



- Decoder can be upgraded with compact schema information
 - ➔ Manage several XML languages, accept brand new ones

- **Forward Compatibility**



- Language versions is signalled and can be skipped
 - ➔ Manage heterogeneous park of very low end decoders

- **Flexible transmission**



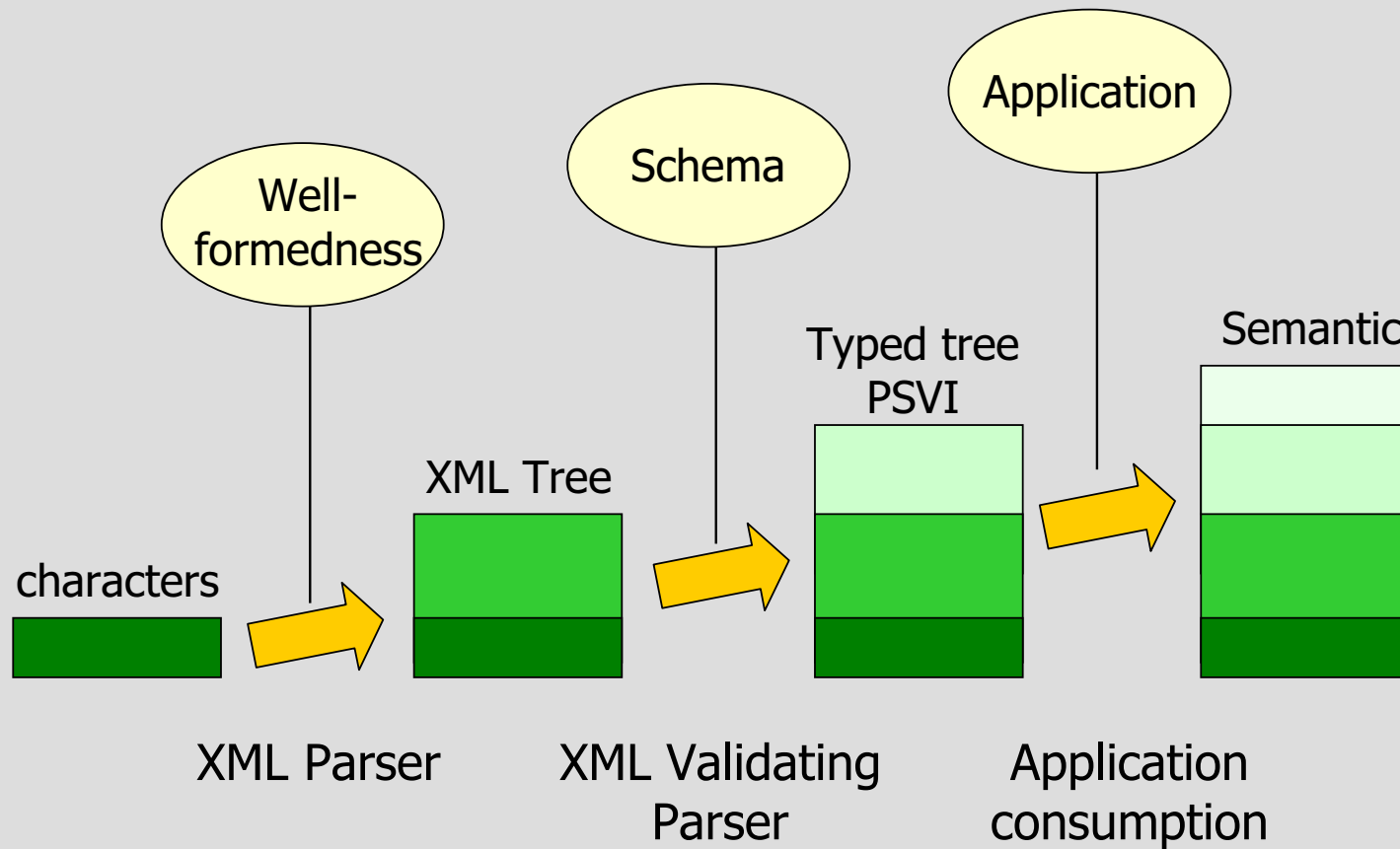
- Parts of the document can be sent independently
 - ➔ Progressive transfer, document updates, XML streaming

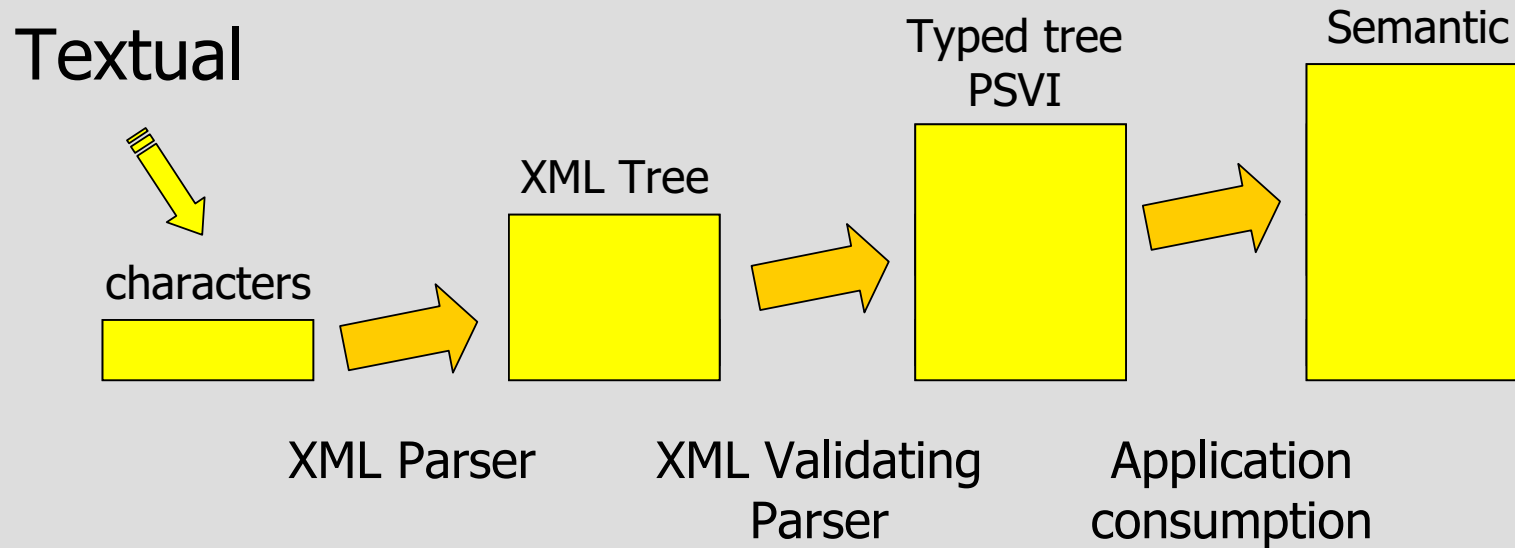


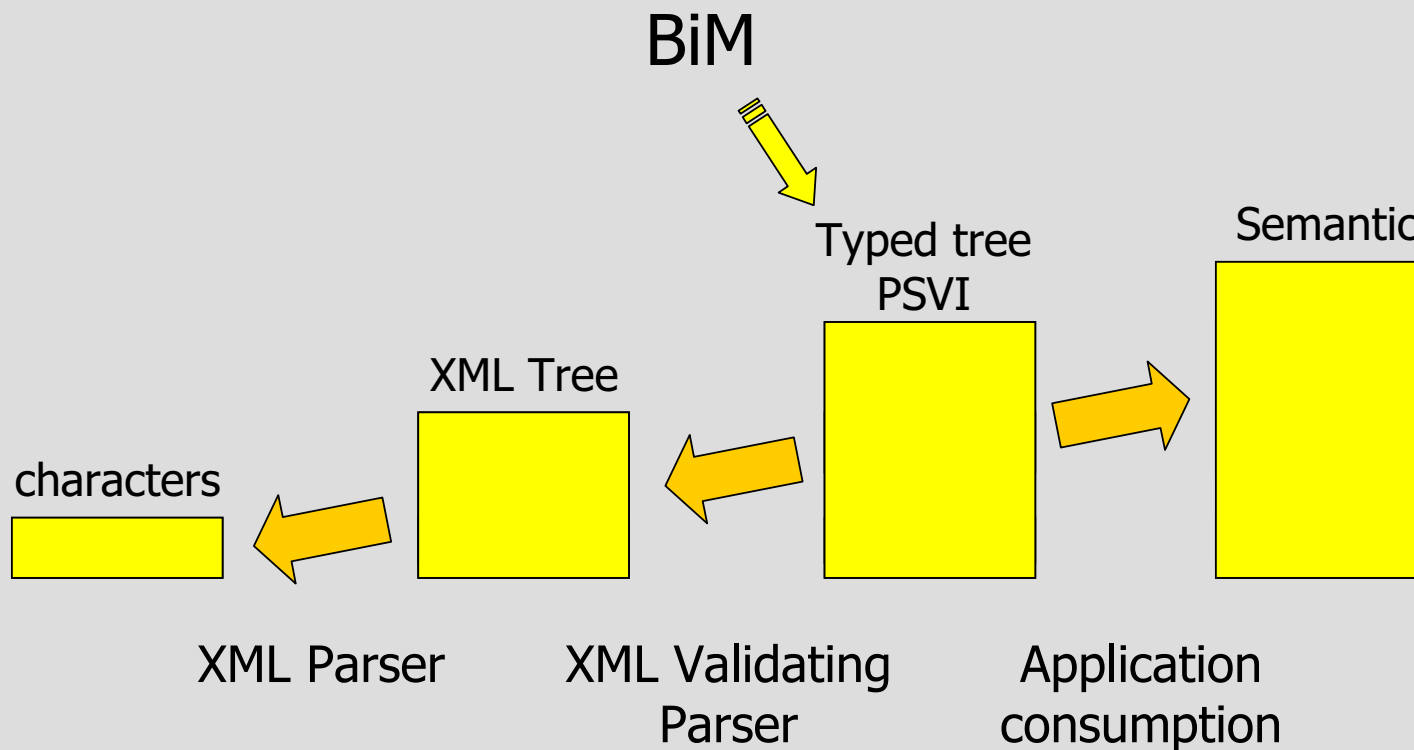


Pre-parsed binary format









- Pre-parsed format
 - No need for string matching, comparison
 - Identified elements & attributes
- Pre-validated format
 - Validation is a by-product of encoding
 - Resolved entities, default values, types, normalized data

Decoding ↔ Validating parsing
and
Direct access to the data

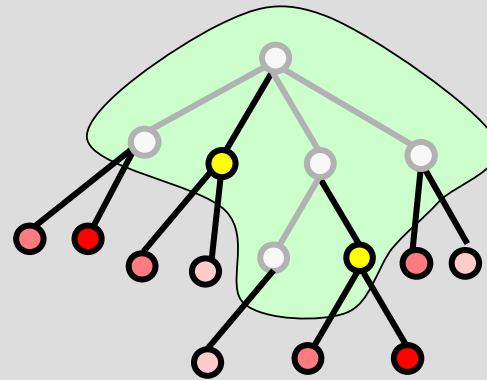




Flexible and Customizable

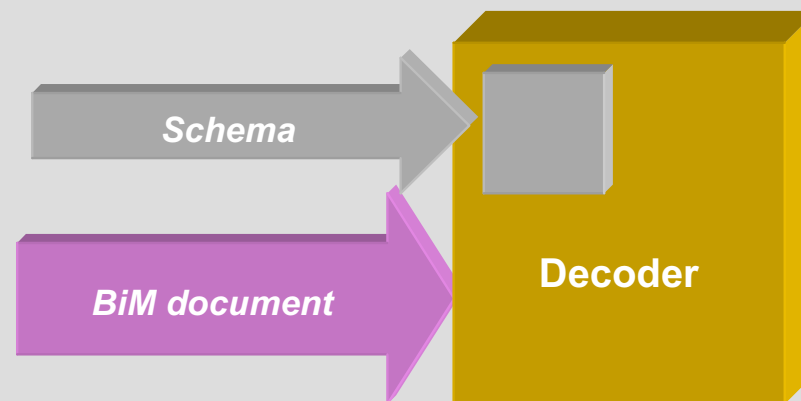


- Encoding framework can be extended to receive
 - New compression methods
 - Checksum / error correction code
 - Authentication / encryption

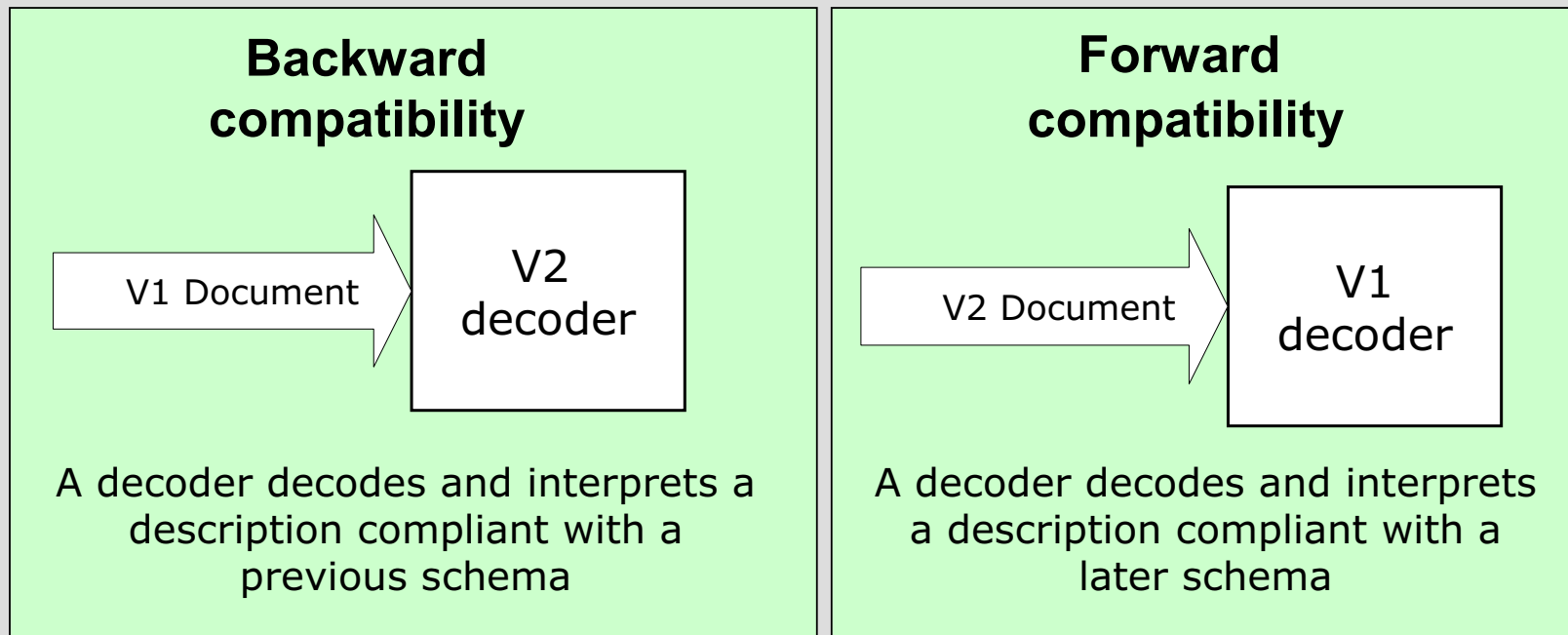


- A type mapping method is used
 - To associate a codec to a certain type
 - To trigger the codec when appropriate

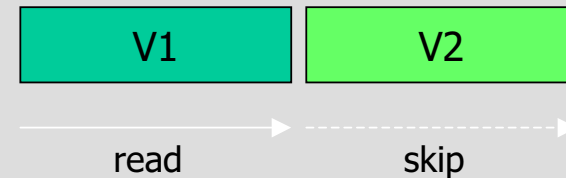
- A decoder can be upgraded to accept new XML languages
 - Schema is transmitted a compact form



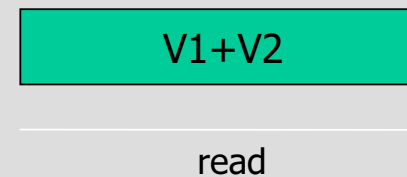
i.e. compatibility between different versions of
an XML language



- If compatibility is required, the encoder
 - Separate **V1 parts** from **V2 parts** of a description
 - Add schema identifiers
 - Add the coding length



- If compatibility is not required,
 - The description is coded as V2.

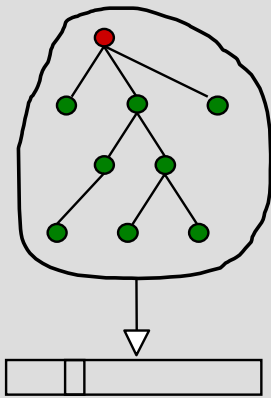


- Interest: management of
 - an heterogeneous stock of specialized decoders
 - transition periods between one version and an other
 - Transparent private extensions to a standard

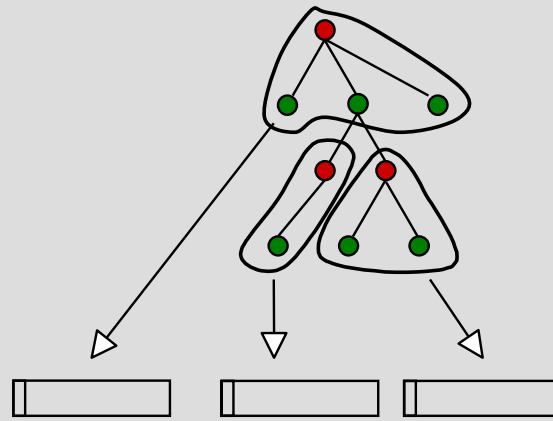


- Large documents usually contain several kind of information of different importance
- Some document parts are relevant at a specific moment in time
- Relevant parts are not always sorted in document order
- An application can start working without having acquired the entire document

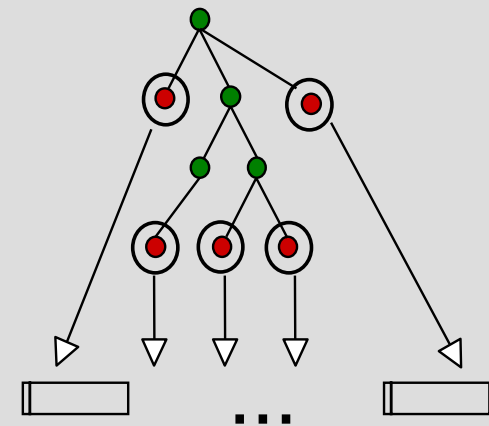




The entire document in one chunk



Subparts of the document in different chunks



Each leaf in one chunk





Compression Techniques

1. FSA based compression for structure
2. Specific Codecs
 - Zlib Codec
 - SVG Codecs





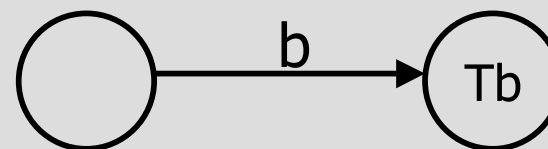
Finite State Automaton Based Compression



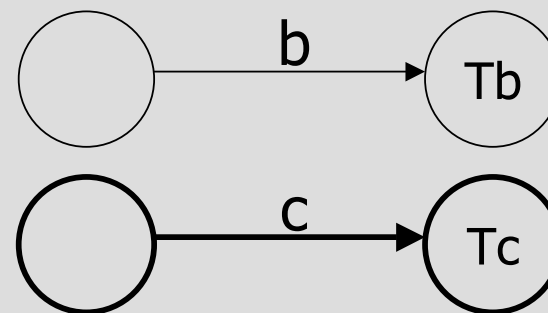
```
<sequence>  
  <element name='a' type='Ta' />  
  <choice minOccurs='0' maxOccurs='unbounded' >  
    <element name='b' type='Tb' />  
    <element name='c' type='Tc' />  
  </choice>  
</sequence>
```



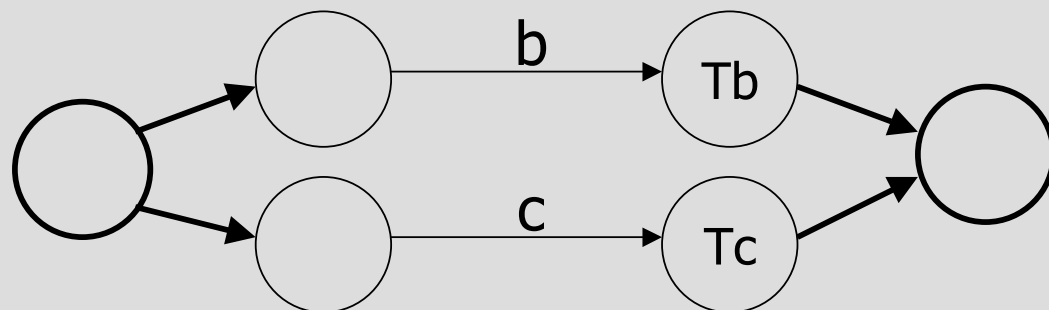
```
<sequence>  
  <element name='a' type='Ta' />  
  <choice minOccurs='0' maxOccurs='unbounded'>  
    <element name='b' type='Tb' />  
    <element name='c' type='Tc' />  
  </choice>  
</sequence>
```



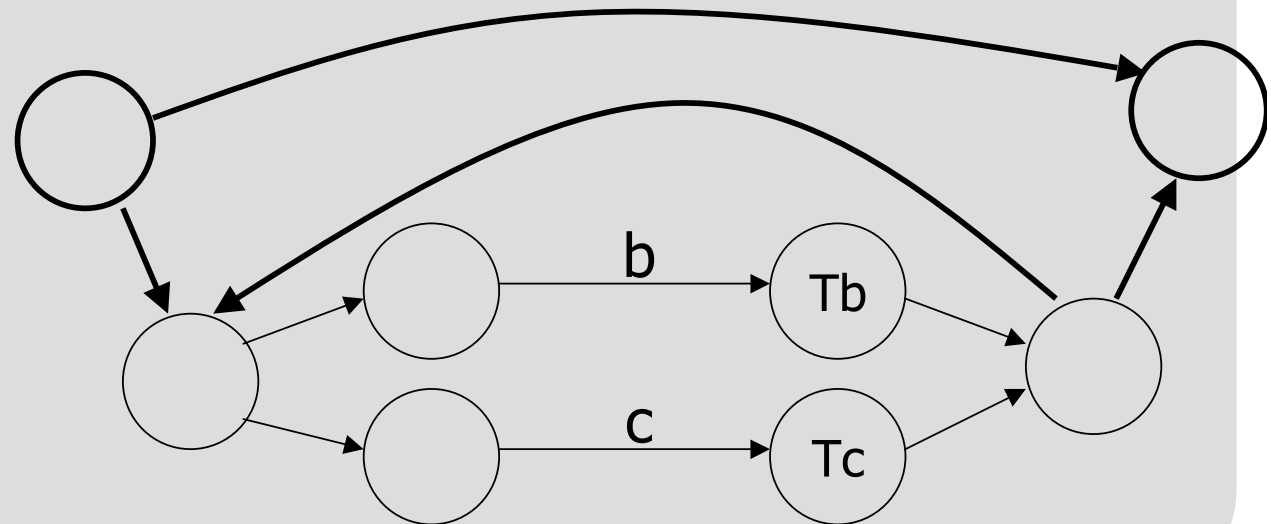
```
<sequence>  
  <element name='a' type='Ta' />  
  <choice minOccurs='0' maxOccurs='unbounded'>  
    <element name='b' type='Tb' />  
    <element name='c' type='Tc' />  
  </choice>  
</sequence>
```



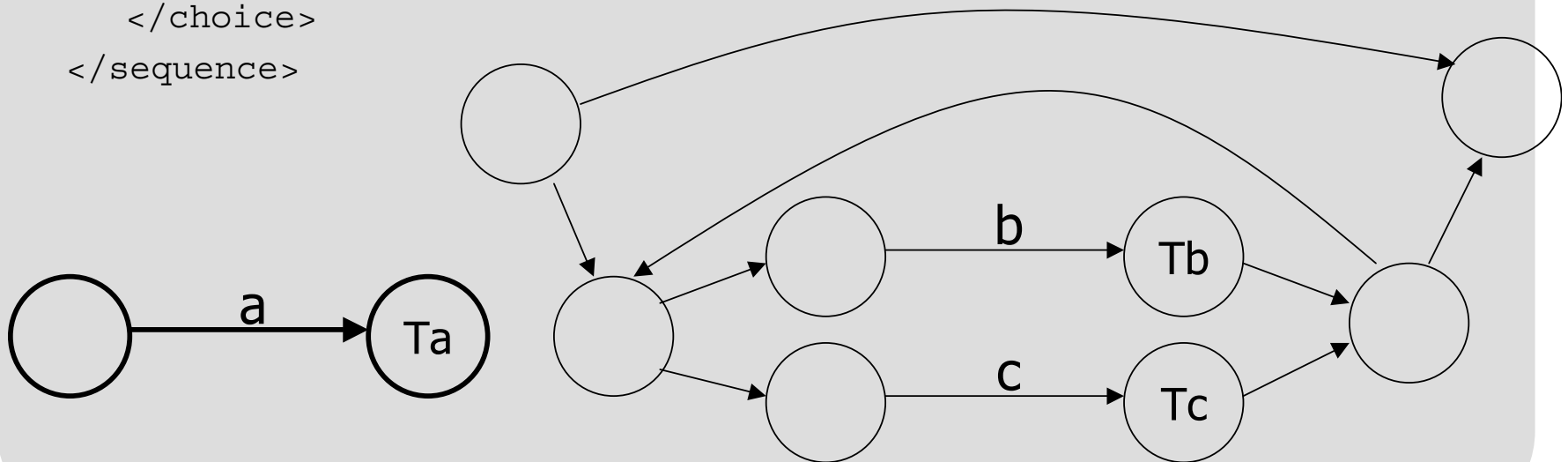
```
<sequence>  
  <element name='a' type='Ta' />  
  <choice minOccurs='0' maxOccurs='unbounded'>  
    <element name='b' type='Tb' />  
    <element name='c' type='Tc' />  
  </choice>  
</sequence>
```



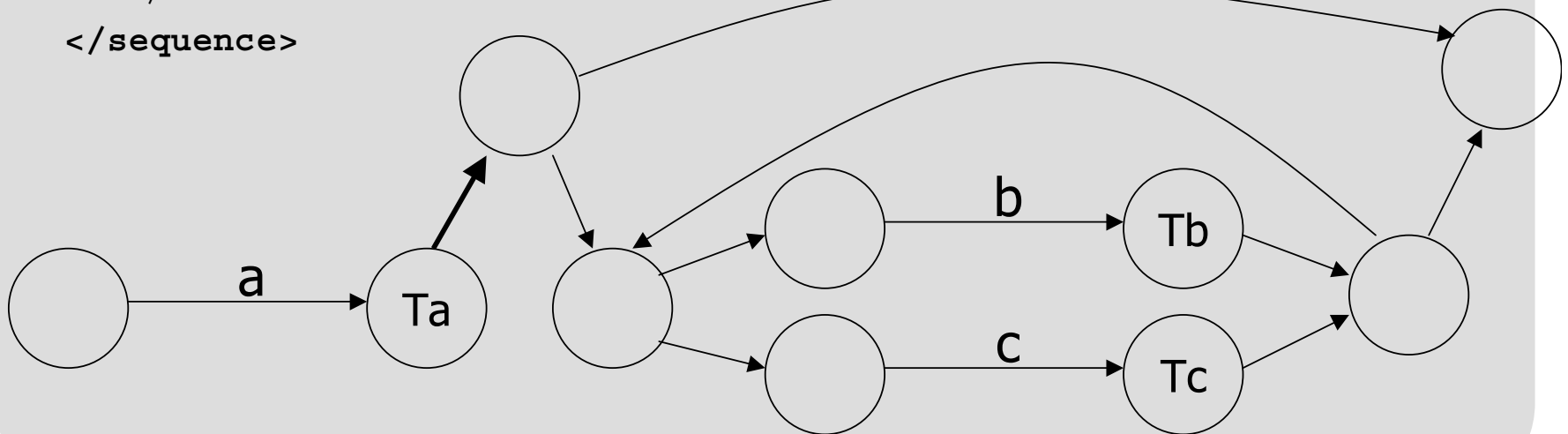
```
<sequence>  
  <element name='a' type='Ta' />  
  <choice minOccurs='0' maxOccurs='unbounded'>  
    <element name='b' type='Tb' />  
    <element name='c' type='Tc' />  
  </choice>  
</sequence>
```



```
<sequence>  
  <element name='a' type='Ta' />  
  <choice minOccurs='0' maxOccurs='unbounded'>  
    <element name='b' type='Tb' />  
    <element name='c' type='Tc' />  
  </choice>  
</sequence>
```



```
<sequence>  
  <element name='a' type='Ta' />  
  <choice minOccurs='0' maxOccurs='unbounded'>  
    <element name='b' type='Tb' />  
    <element name='c' type='Tc' />  
  </choice>  
</sequence>
```

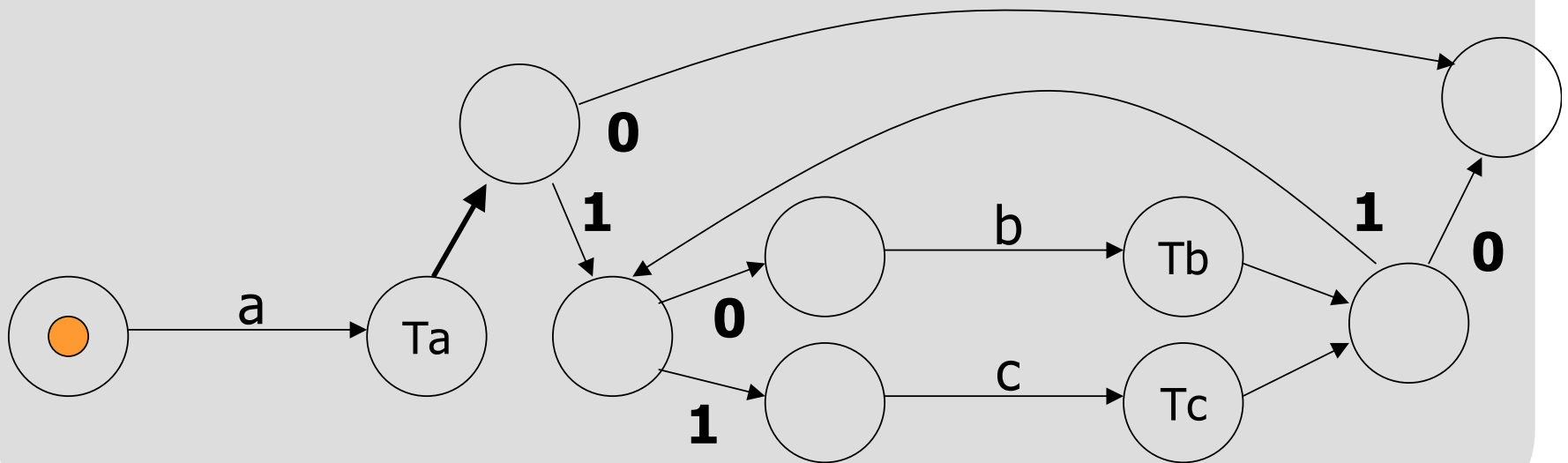




FSA Decoders - Use

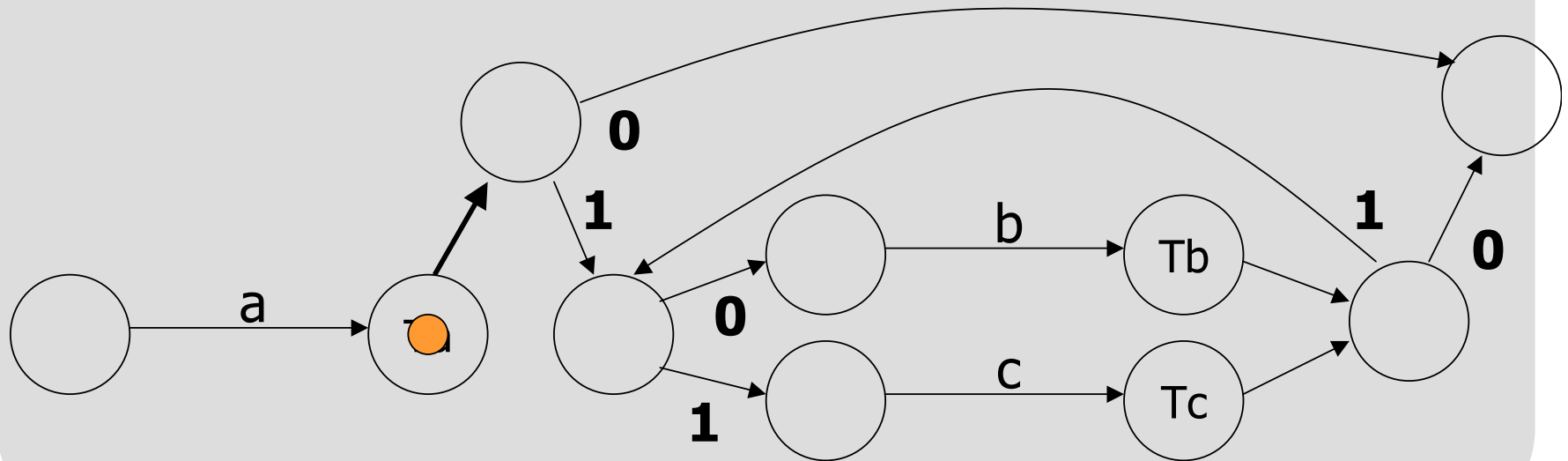


1 0 1 0 1 1 1 1 0 0



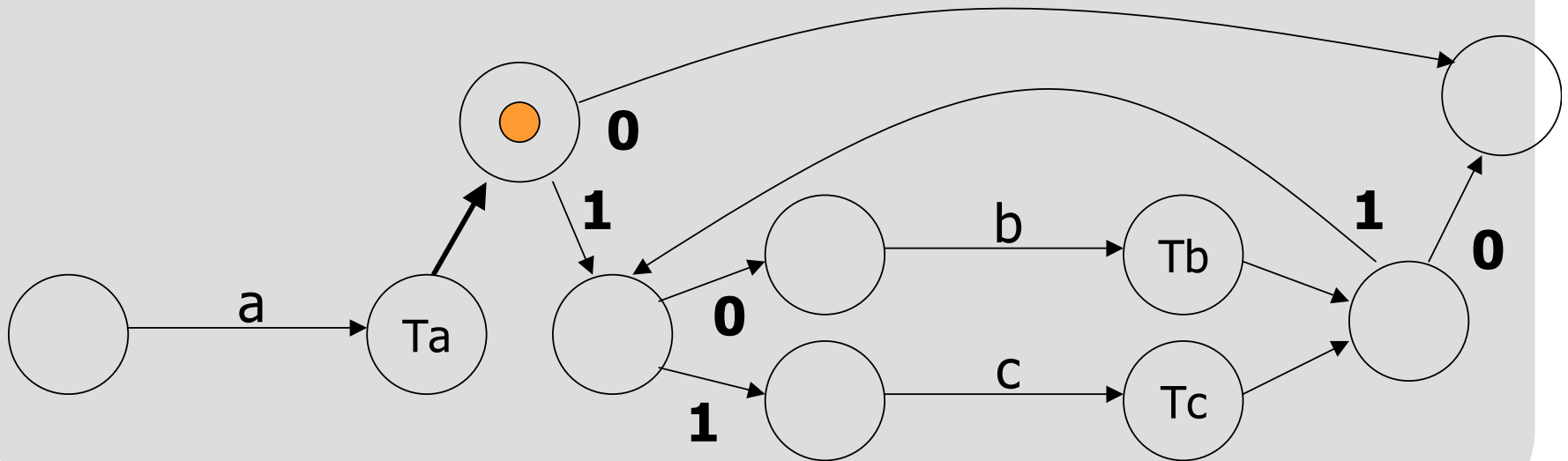
1 0 1 0 1 1 1 1 0 0

↑
a



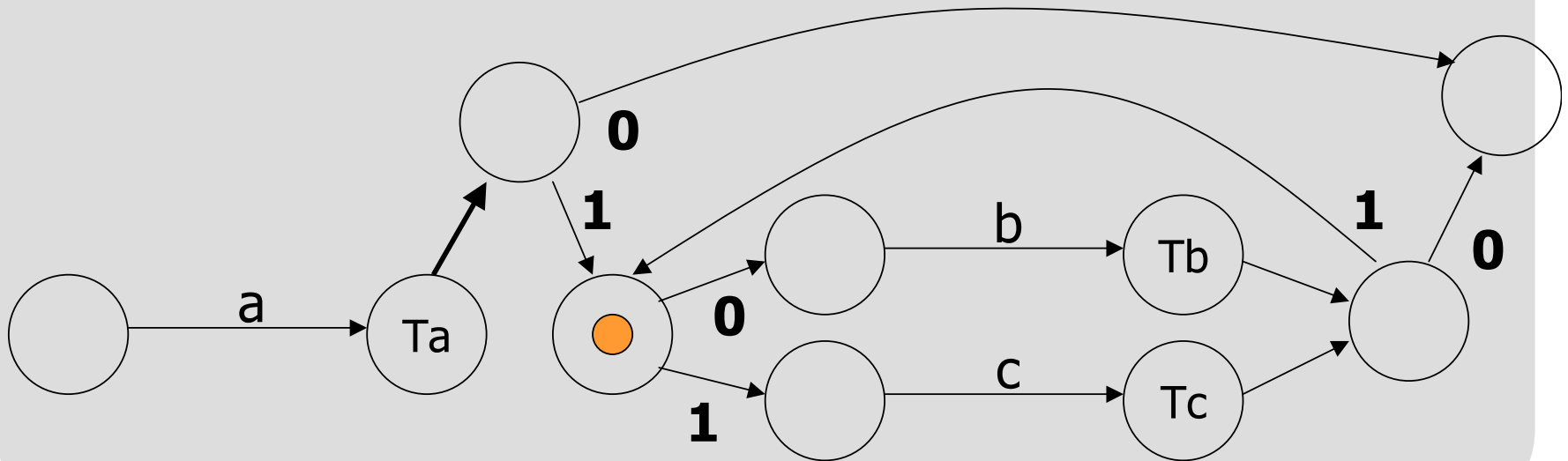
1 0 1 0 1 1 1 1 0 0

↑
a



1 0 1 0 1 1 1 1 0 0

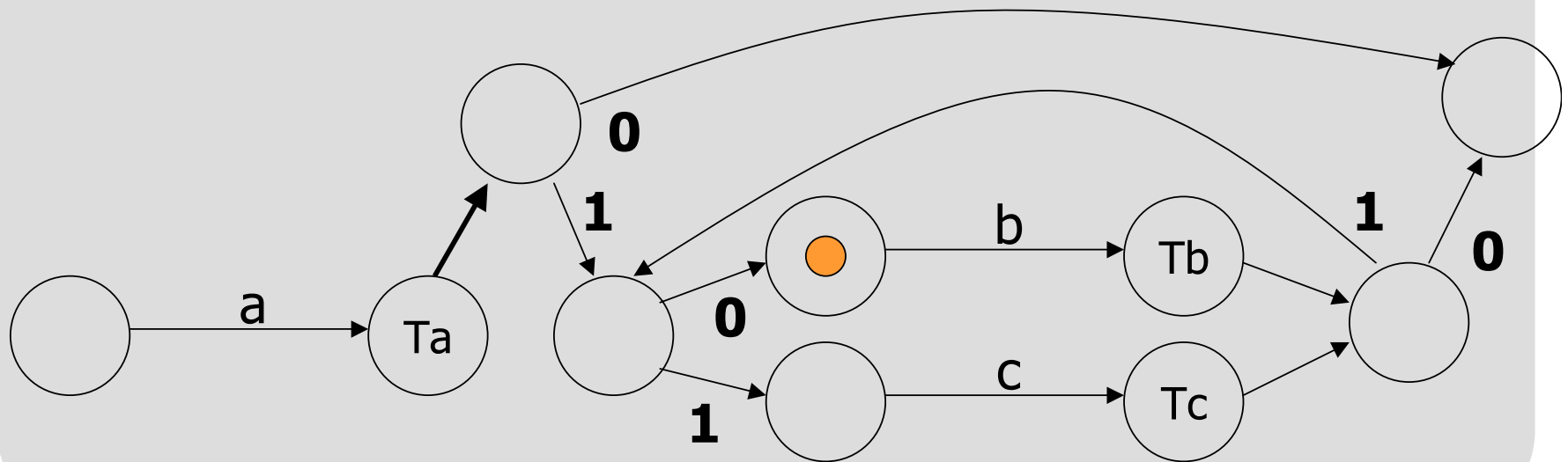
↑
a



1 0 1 0 1 1 1 1 0 0

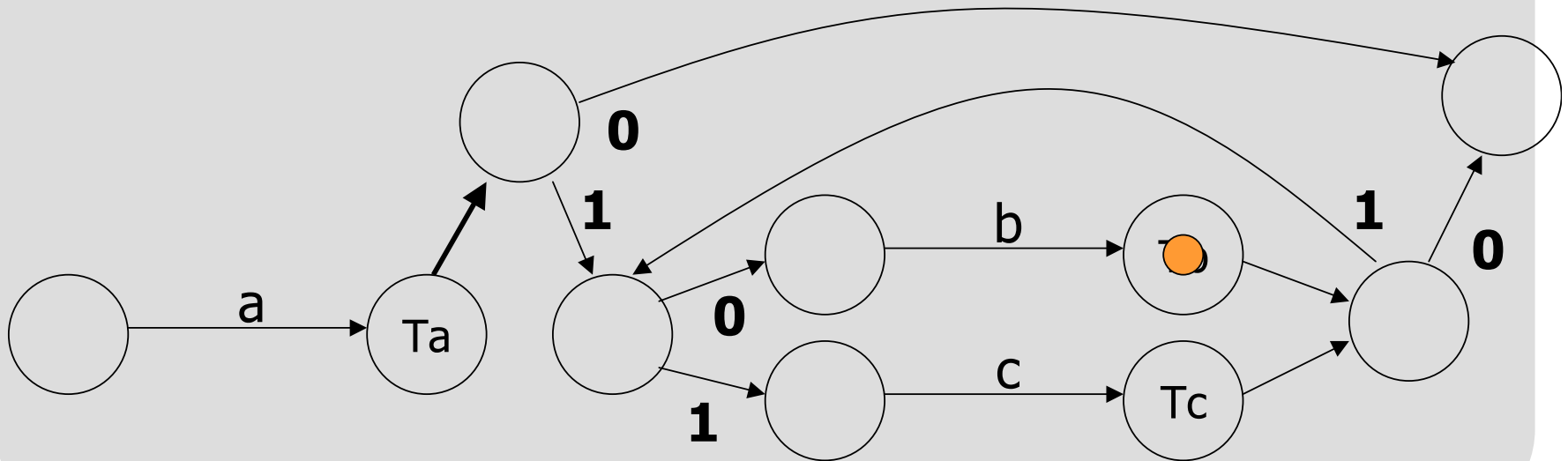


a



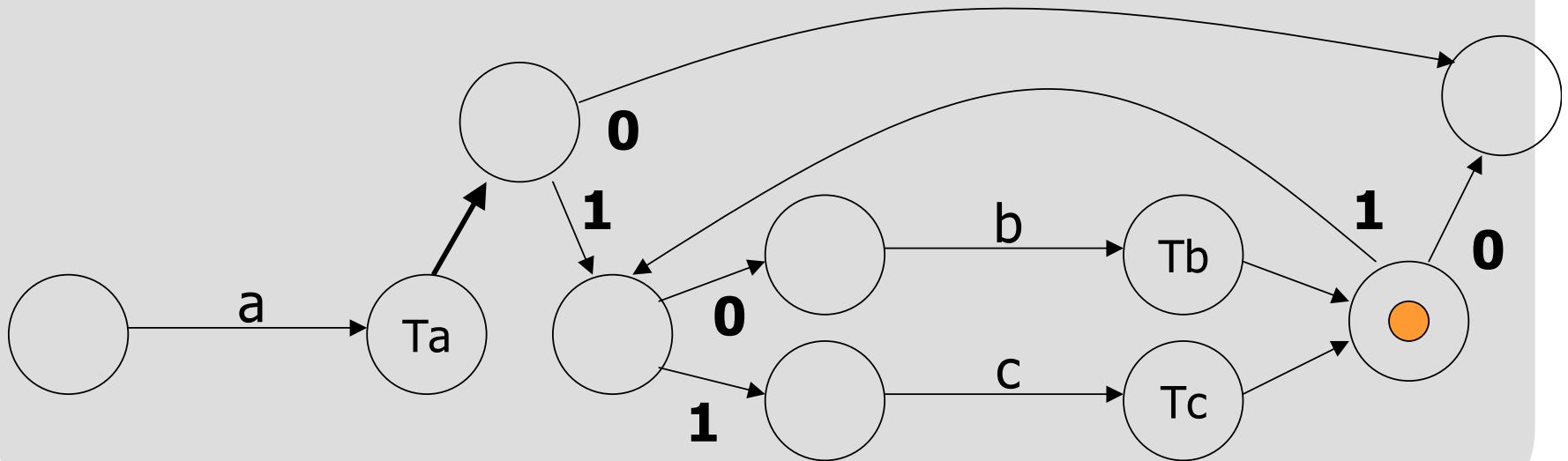
1 0 1 0 1 1 1 1 0 0

a b ↑



1 0 1 0 1 1 1 1 0 0

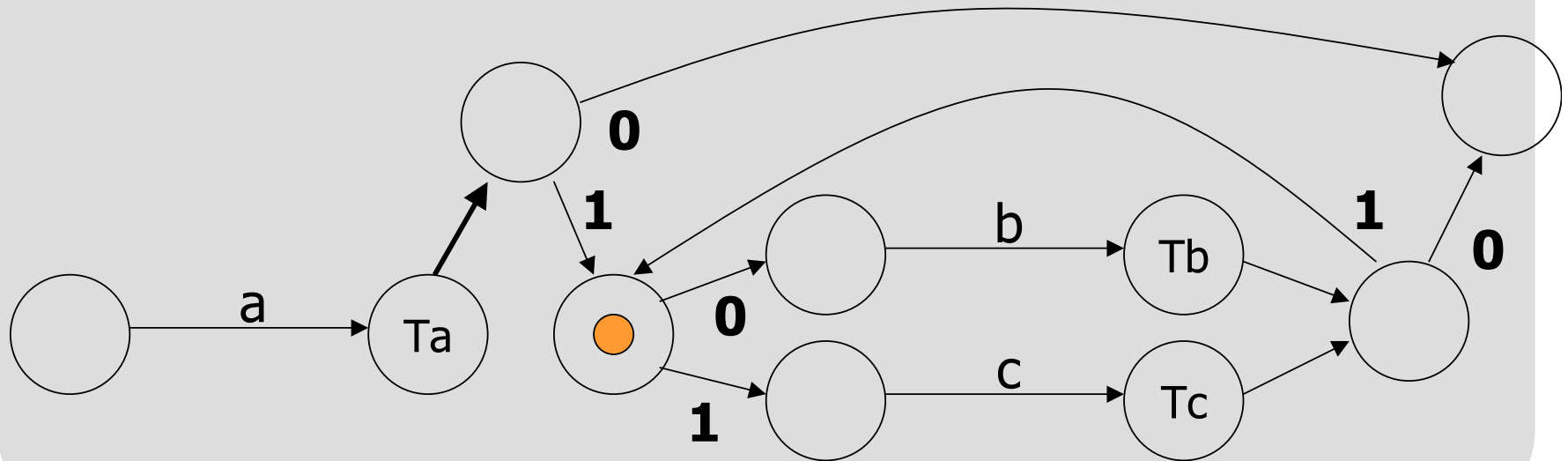
a b 



1 0 1 0 1 1 1 1 0 0



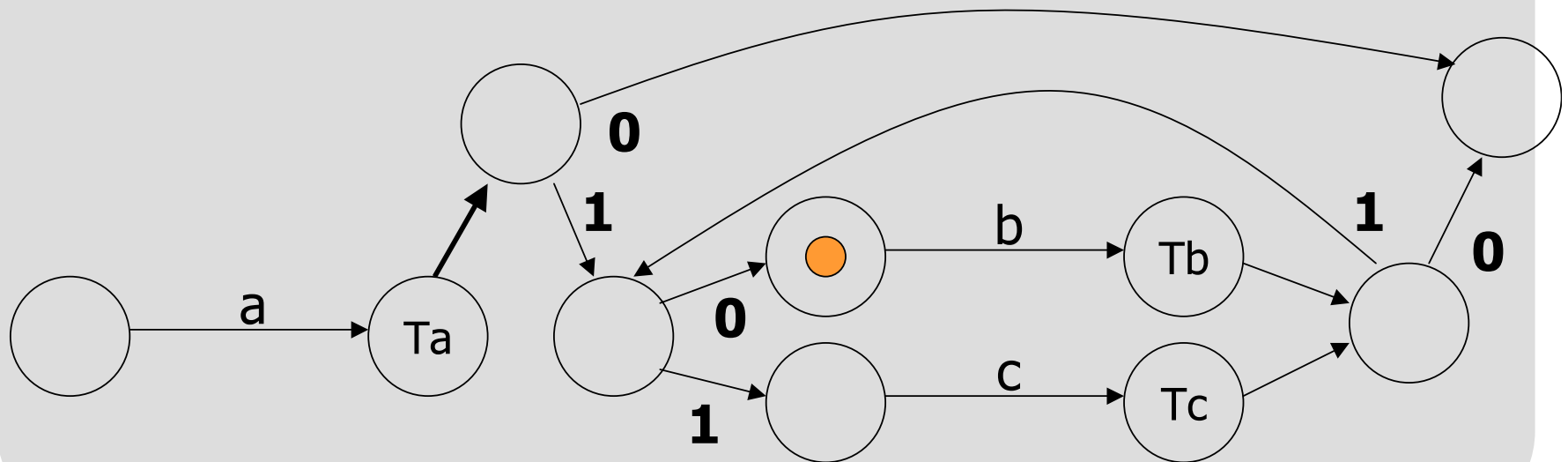
a b



1 0 1 0 1 1 1 1 0 0



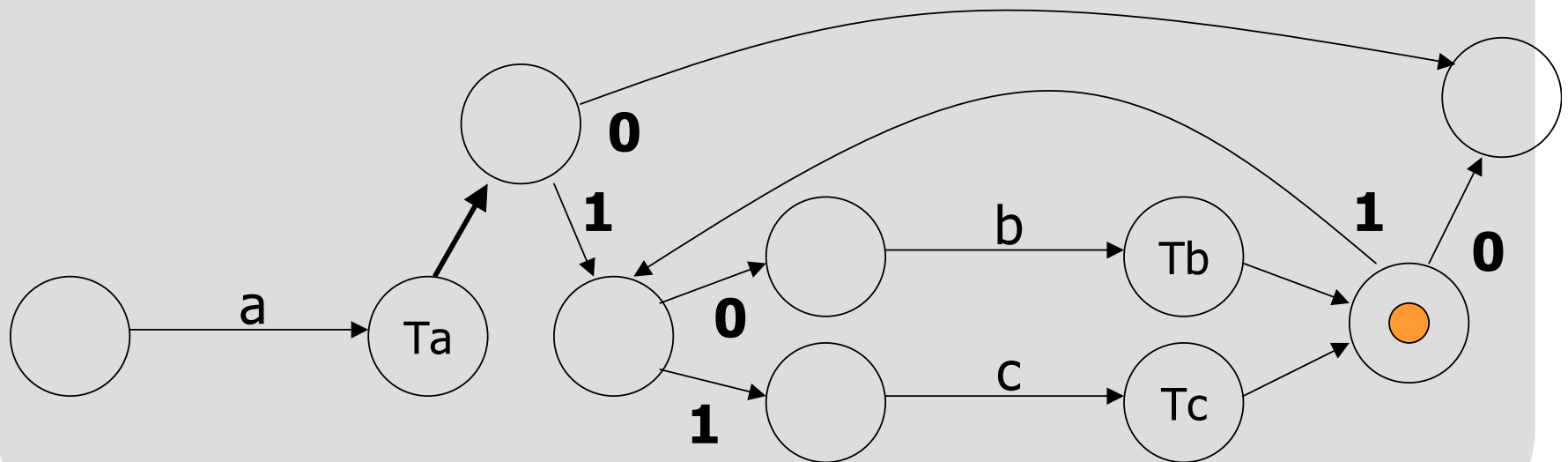
a b



1 0 1 0 1 1 1 1 0 0



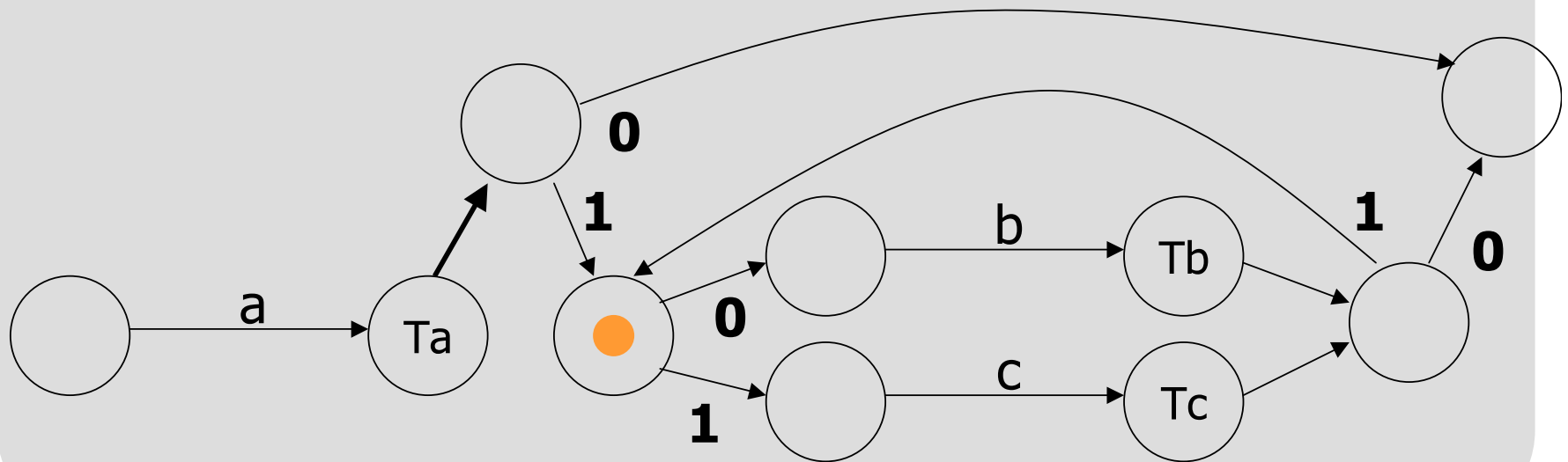
a b b



1 0 1 0 1 1 1 1 0 0



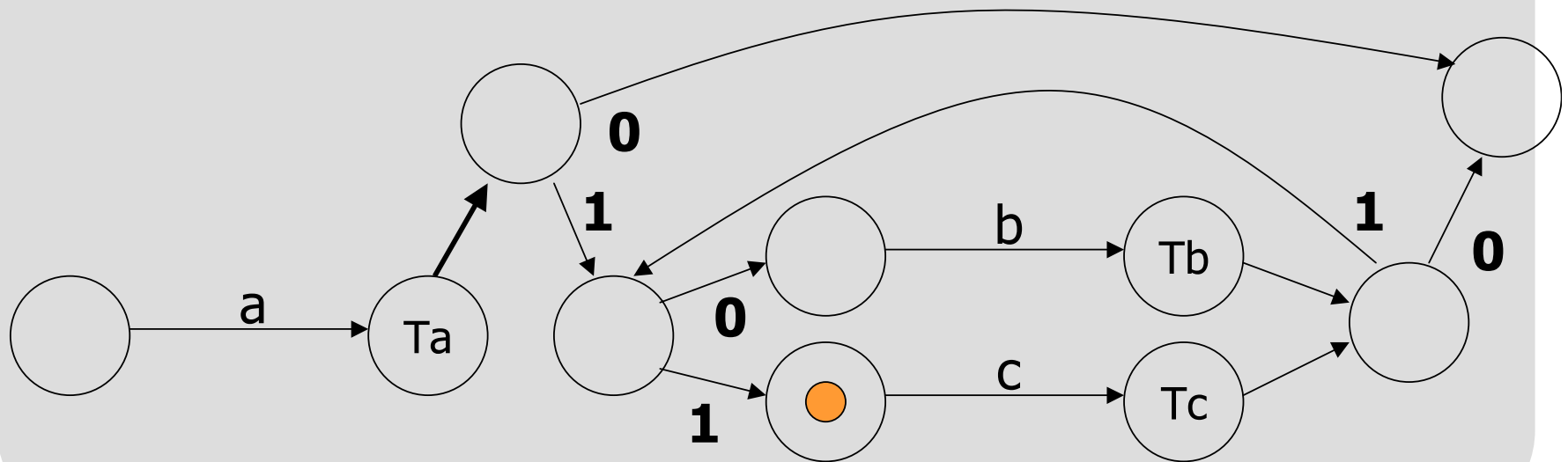
a b b



1 0 1 0 1 1 1 1 0 0



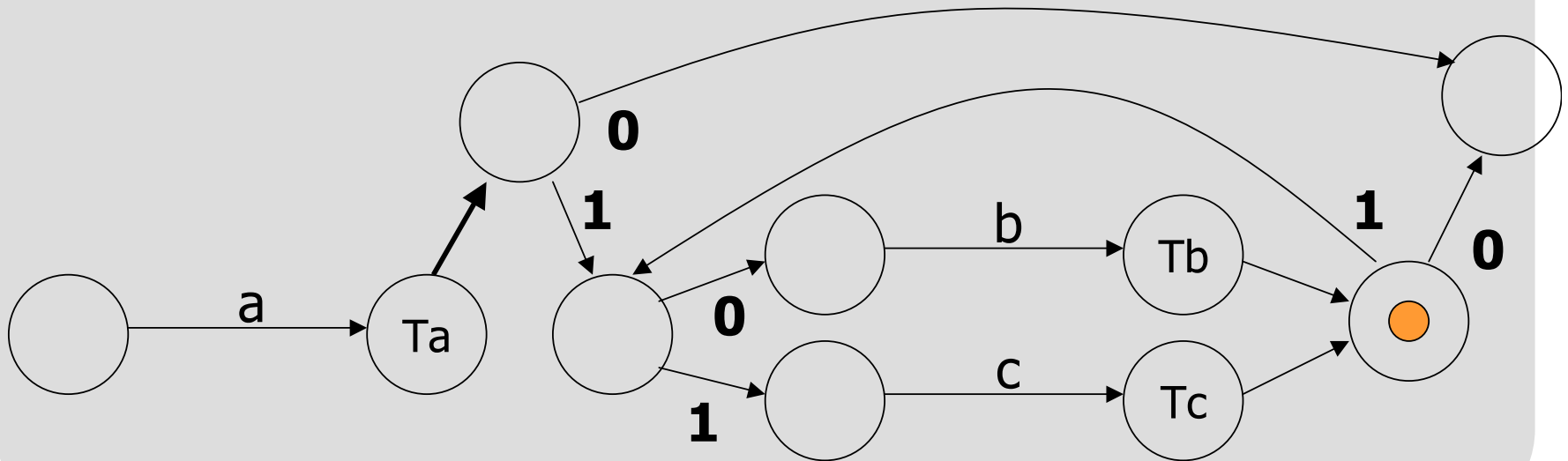
a b b



1 0 1 0 1 1 1 1 0 0

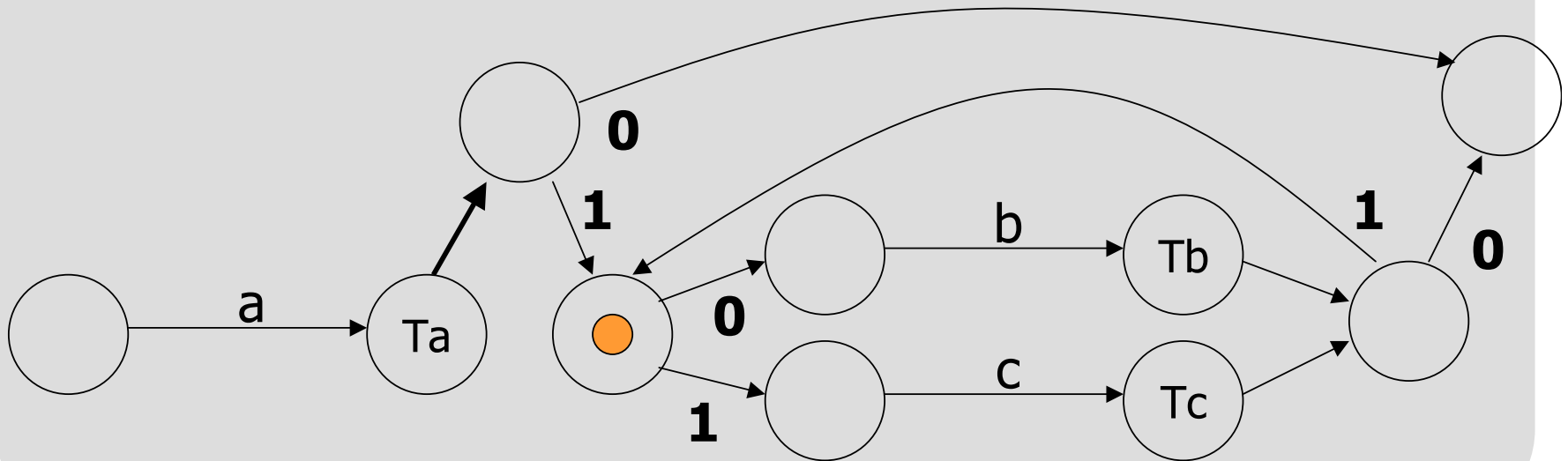


a b b c



1 0 1 0 1 1 1 1 0 0

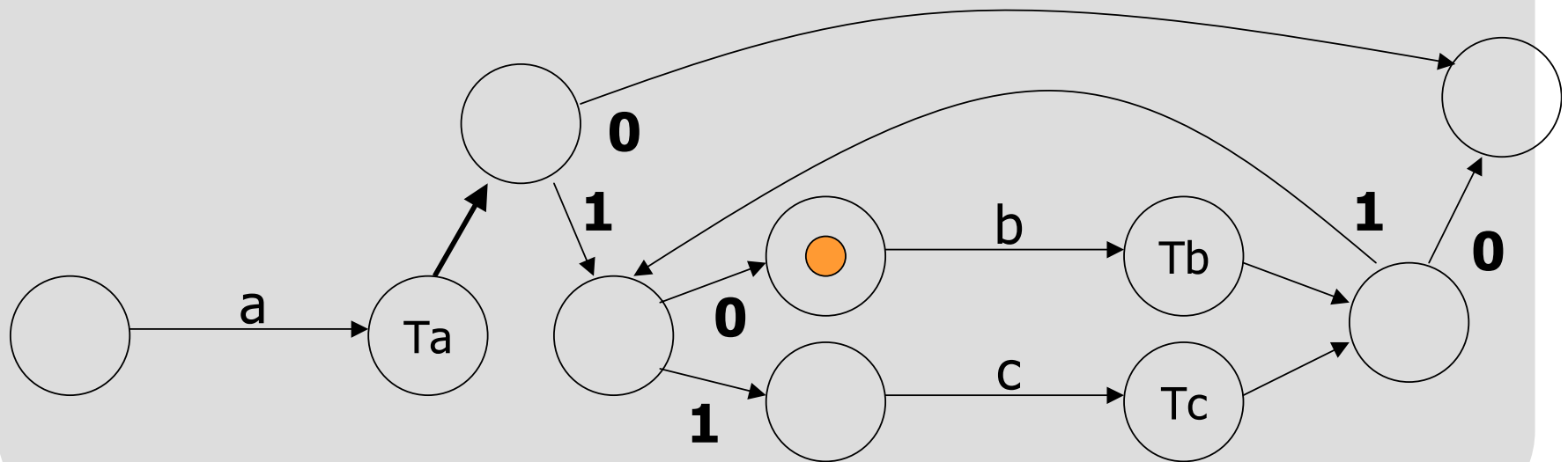
a b b c



1 0 1 0 1 1 1 1 0 0

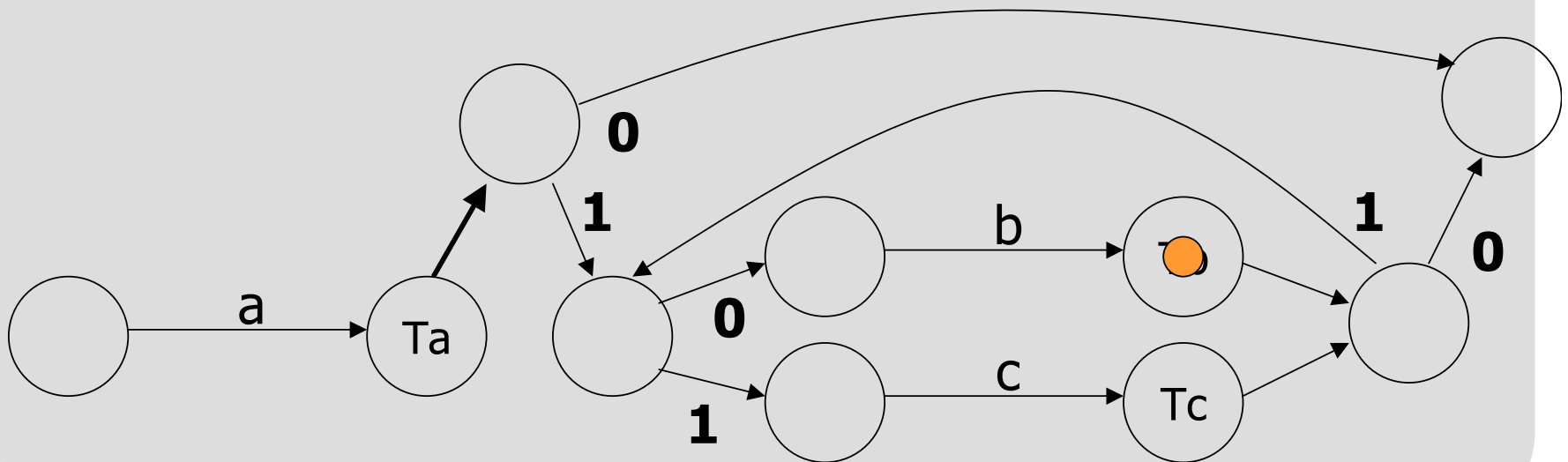


a b b c



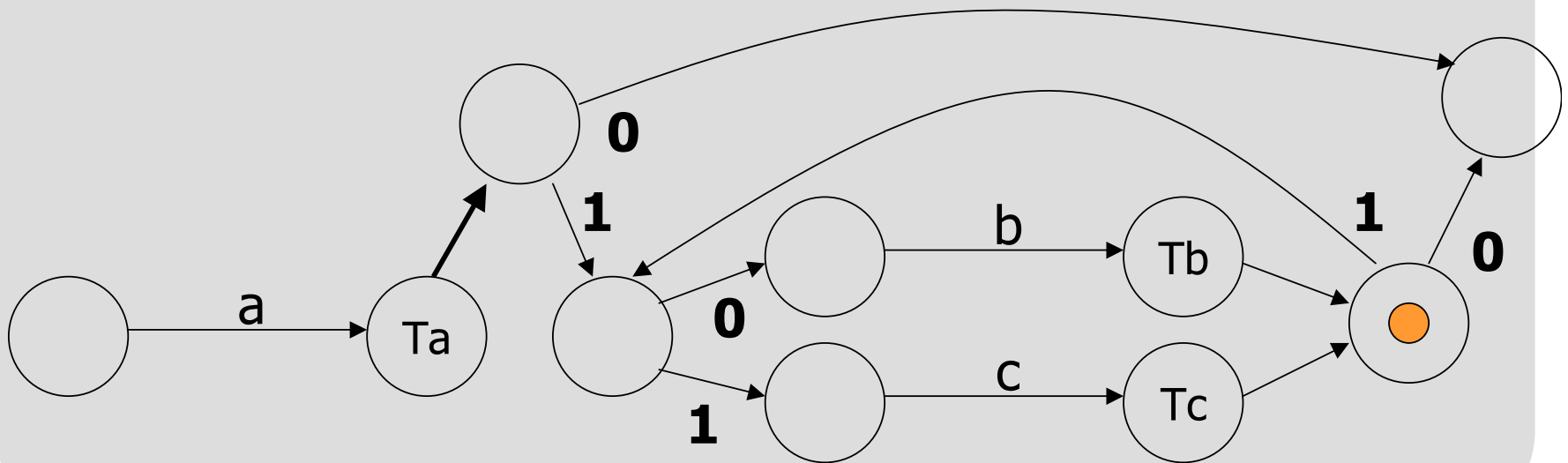
1 0 1 0 1 1 1 1 0 0

a b b c b



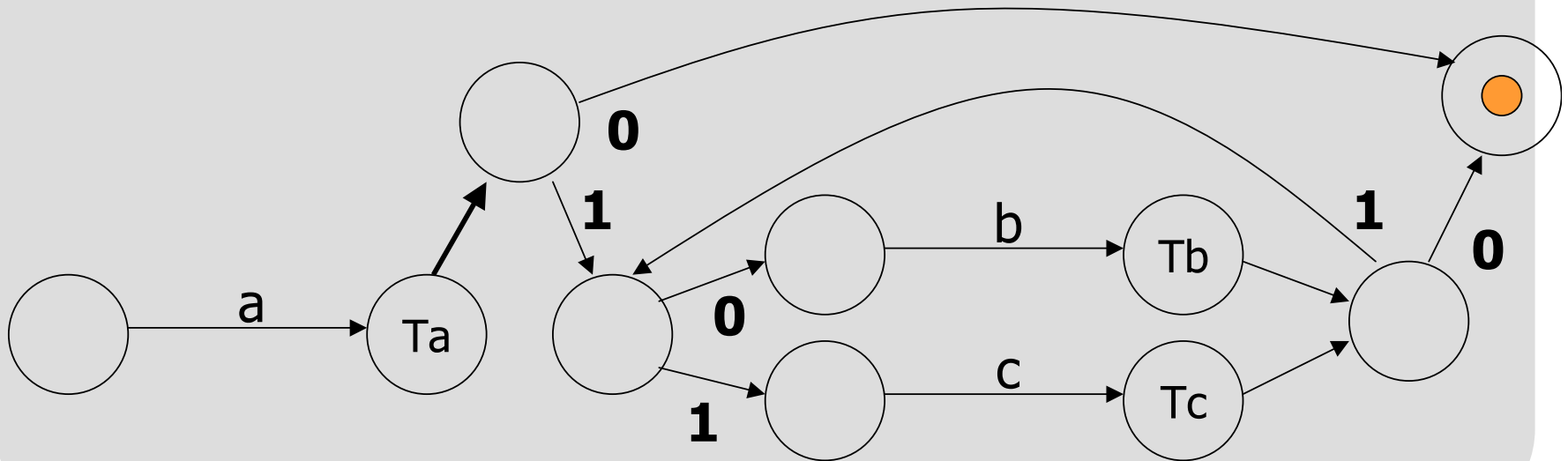
1 0 1 0 1 1 1 1 0 0

a b b c b



1 0 1 0 1 1 1 1 0 0

a b b c b



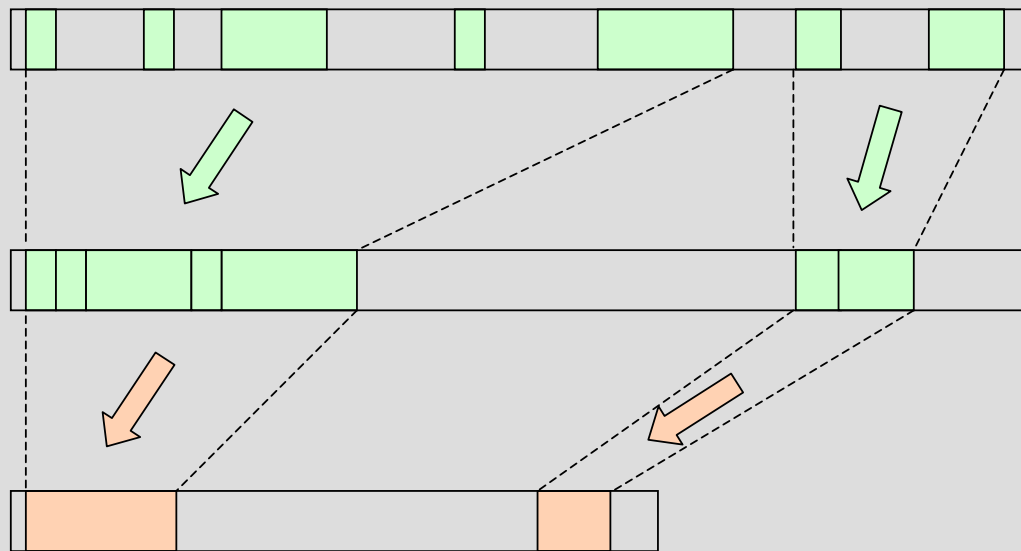


BiM Specific Codecs

1. Strings compression using the Zlib codec
2. SVG compression



- Strings are gathered in larger buffers
- Buffers are zipped



- Compression techniques used :
 - Linear quantization and relative coding
 - Contextual quantization
 - Variable bitsize coding
 - Pattern oriented coding
 - Palette





Original SVGT reference
43333 bytes

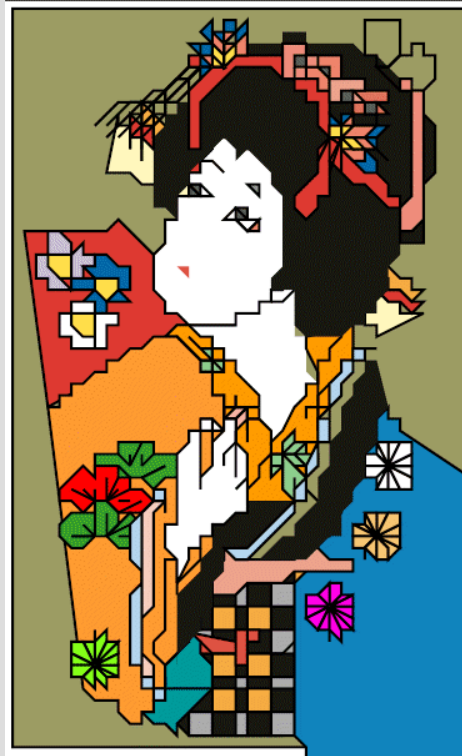


BiM 9 bits/point (lossless)
5531 bytes

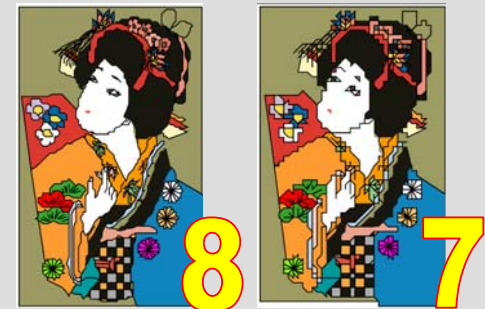
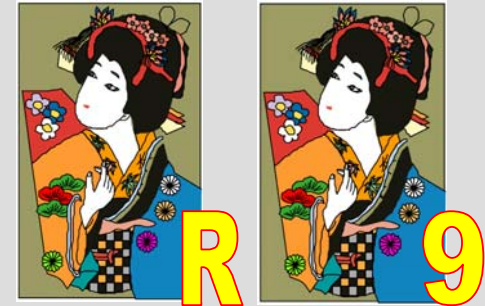
Lossy and Lossless compression



BiM 8 bits/point
4569 bytes



BiM 7 bits/point
3521 bytes



100 pixels

Zlib = **9617**



Compression (in bytes)	SVG size	Zip size	BiM size	Gain o/Zip
AroundTheSun	1935	656	246	62,5 %
Euroflag	1860	343	313	8,7 %
Flower	2630	922	388	57,9 %
Clown	32540	6628	3989	39,8 %
Fish	32080	5236	4562	12,9 %
Kimono	43333	9617	5531	42,5 %
Lion	19995	5759	3449	40,1 %





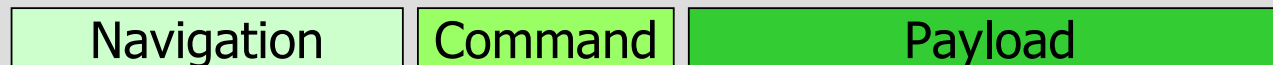
Updating Mechanisms



- Principle
 - Manage a DOM tree in the decoder
 - Modify it through a set of updates

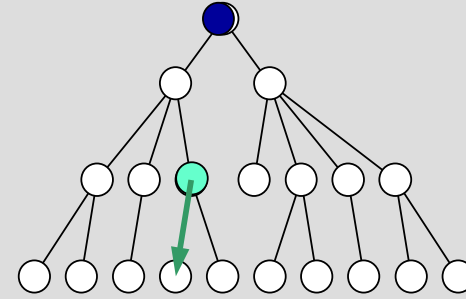
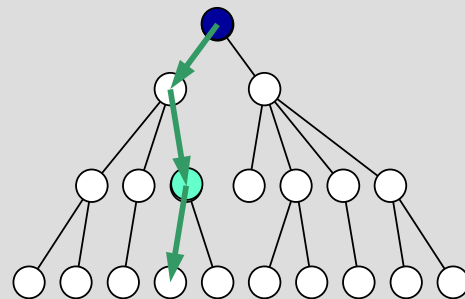
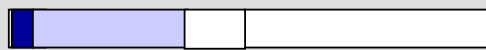
- Steps of update
 - 1** – Selection of one node in the tree (**navigation**)
 - 2** – Execution of the command: delete, update, add (**command**)
 - 3** – Decoding of the encoded subtree (**payload**)

- Update Units



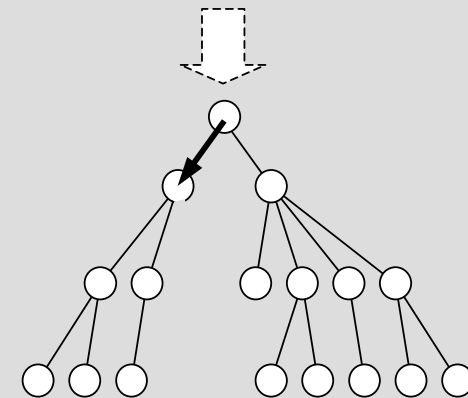
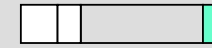
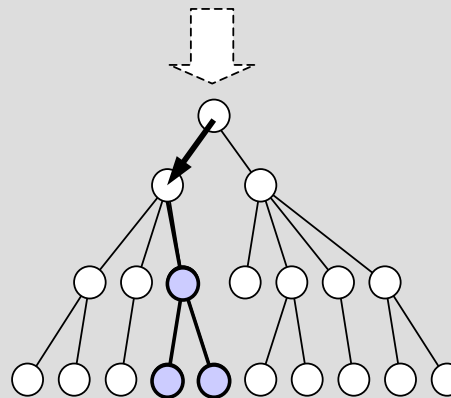
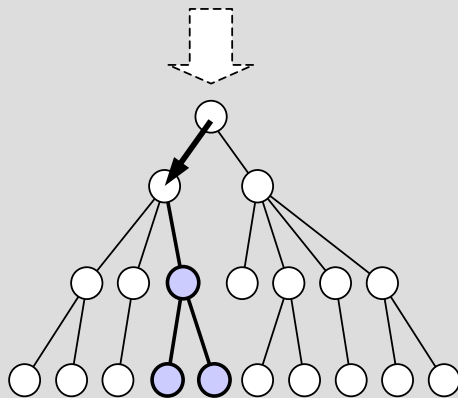
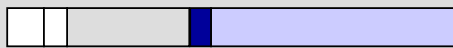
Two possibilities:

- ➡ Absolute *start from the root node*
- ➡ Relative *start from the context node*



Three possibilities:

- ➡ Add *add a subtree*
- ➡ Update *update a subtree*
- ➡ Delete *delete a subtree*



- For more information, don't hesitate to email me:
- Robin Berjon
<robin.berjon@expway.fr>

