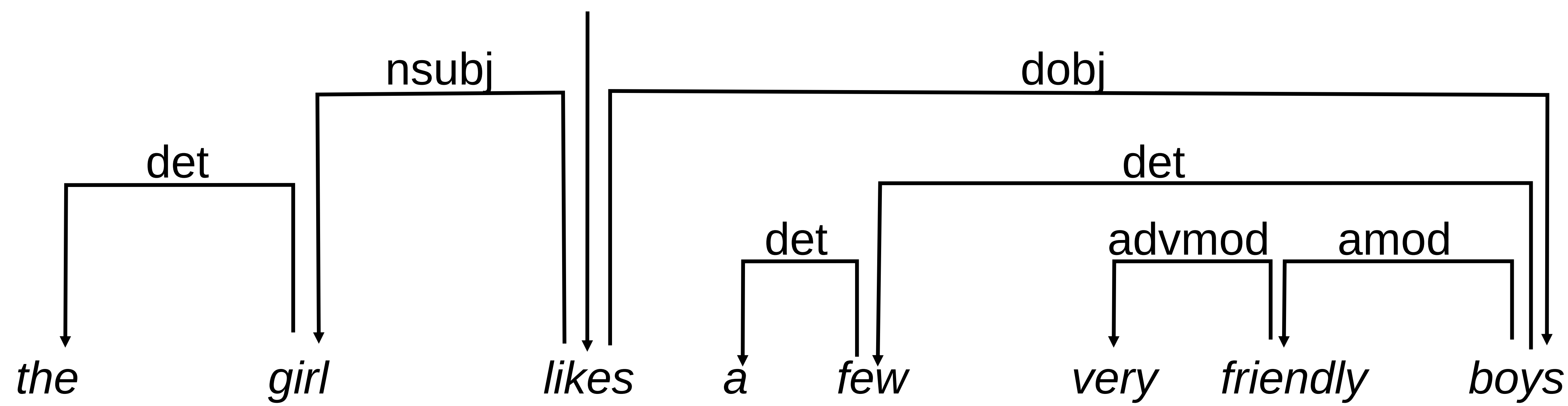
Natural Language Processing

Lecture 8: Dependency Parsing

10/8/2020

COMS W4705
Yassine Benajiba

Dependency Structure



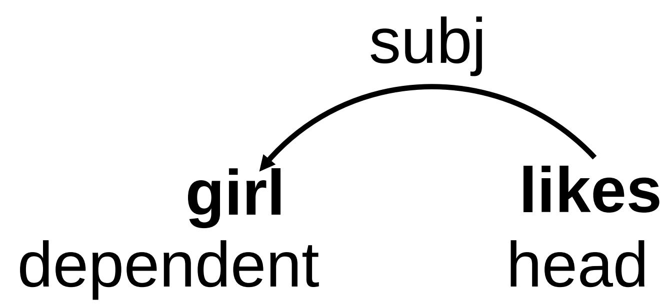
- The edges can be labeled with **grammatical relations** between words (typed dependencies):
  - Arguments (Subject, Object, Indirect Object, Prepositional Object)
  - Adjunct (Temporal, Locative, Causal, Manner…) / Modifier
  - Function words

Dependency Structure

- Long history in linguistics (Starting with Panini's Grammar of Sanskrit, 4th century BCE).
  - Modern dependency grammar originates with Tesniere and Mel'čuk.
- Different from phrase structure (but related via the concept of constituency and heads)
  - Focus is on grammatical relationships between words (Subject, Object, …)
- Tighter connection to natural language semantics.

Dependency Relations

- Each dependency relation consists of a head and a dependent.



- Represent individual edges as subj(*likes-02, girl-01)*

- or as a triple (*likes, nsubj, girl)*

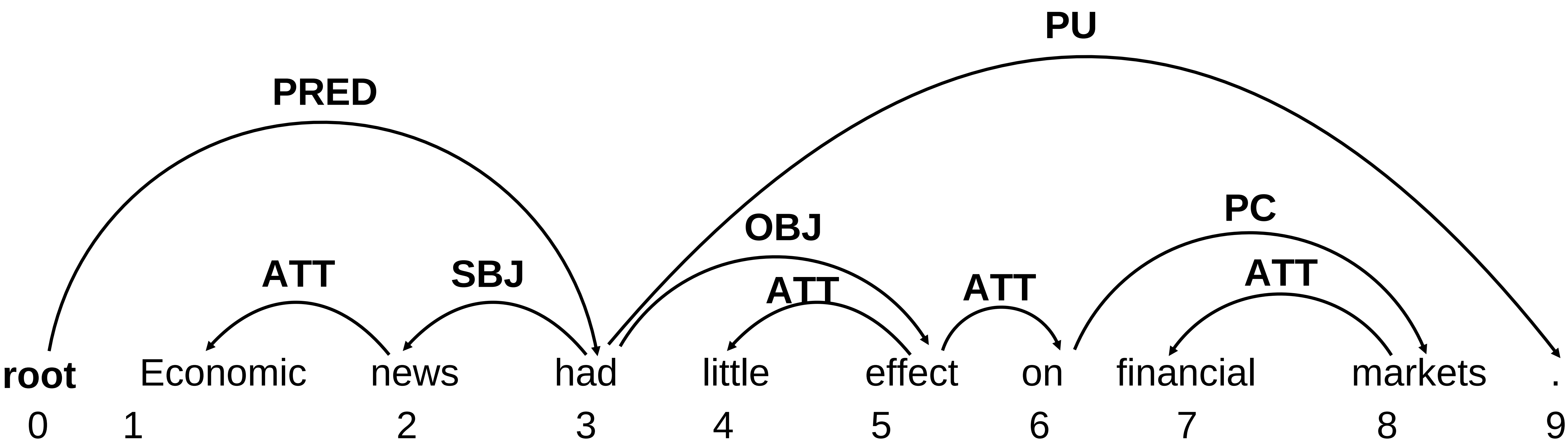
- And the entire sentence structure as a set of edges:

root(likes-2), subj(likes-2, girl-1), det(the-0, girl-1), obj(likes-2, boys-7), det(boys-7, few-4), det(few-4, a-3), amod(boys-7, friendly-6), advmod(friendly-6, very-5)

Heads and Dependents

- How do we identify the the grammatical relation between head H and Dependent D (in a particularly constituent C)?
  - H determines the syntactic category of C and can often replace C.
  - H determines the semantic category of C; D gives semantic specification.
  - H is obligatory; D may be optional.
  - H selects D and determines whether D is obligatory or optional.
  - The form of D depends on H (agreement or government).
  - The linear position of D is specified with reference to H.

Another Example

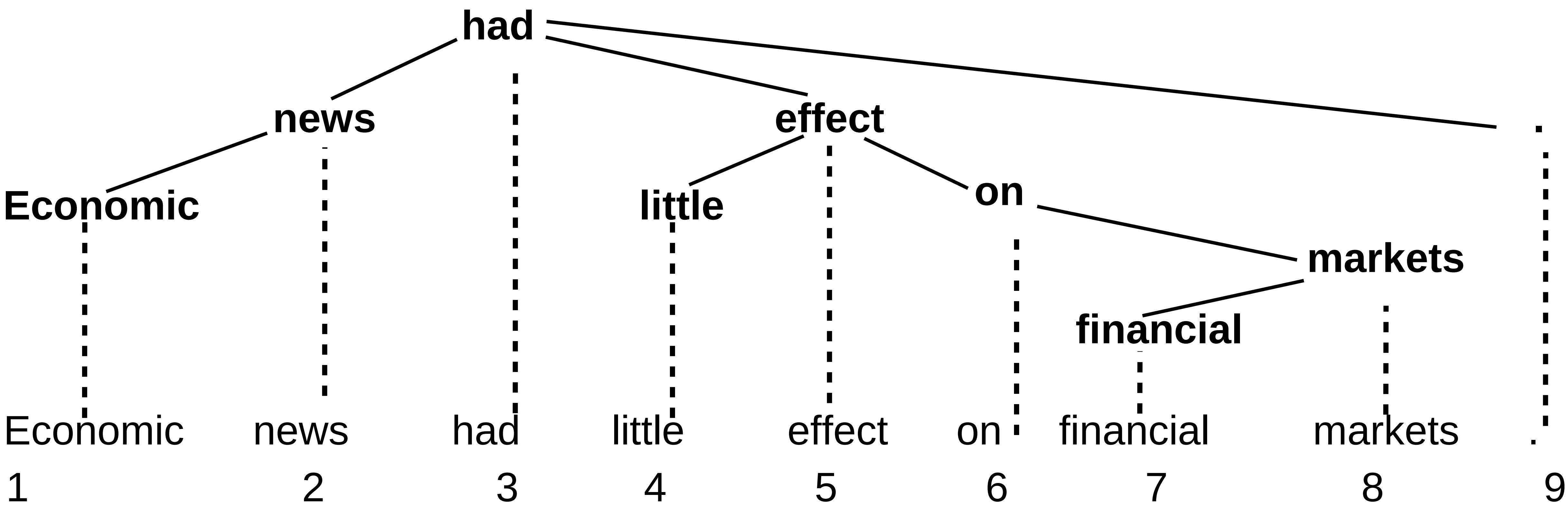


Dependency structure G = (V_S, A)

set of nodes V_S= {**root**, Economic, news, had, little, effect, on, financial, markets, . }

set of edges/ arcs A = {(root, PRED, had), (had, SBJ, news), (had, OBJ, effect),(had, PU, .),
(news,ATT,Economic),(effect,ATT,little),(effect,ATT,on), (on,PC,markets),
(markets, ATT, financial)}

Another Example



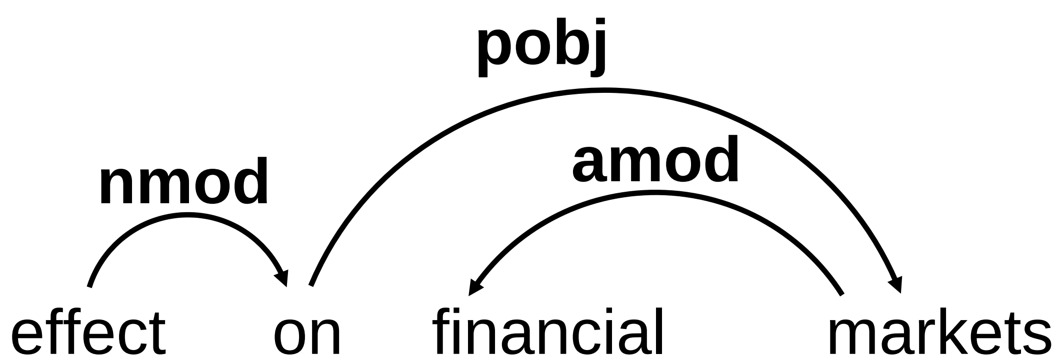
G = (V, A)

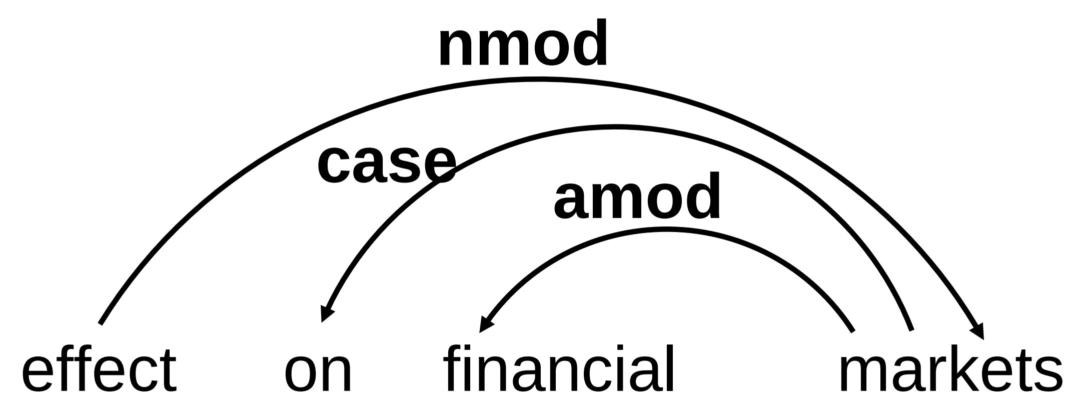
V = {**root**, Economic, news, had, little, effect, on, financial, markets, . }

A = {(root, PRED, had), (had, SBJ, news), (had, OBJ, effect),(had, PU, .),
(news,ATT,Economic),(effect,ATT,little),(effect,ATT,on), (on,PC,markets),
(markets, ATT, financial)}

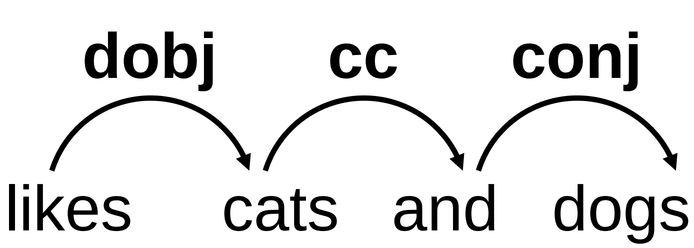
Different Dependency Representations

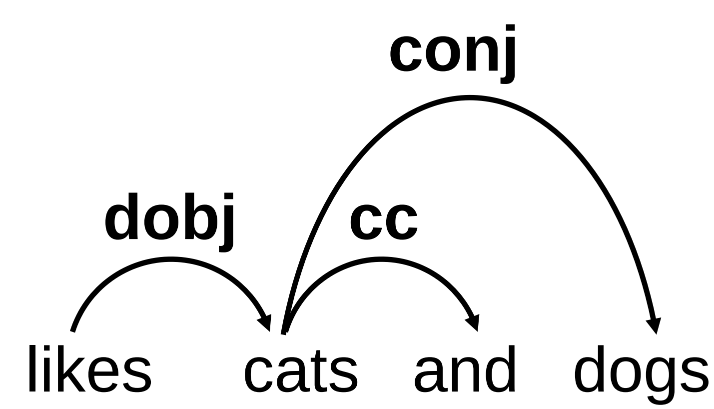
- How to deal with prepositions?

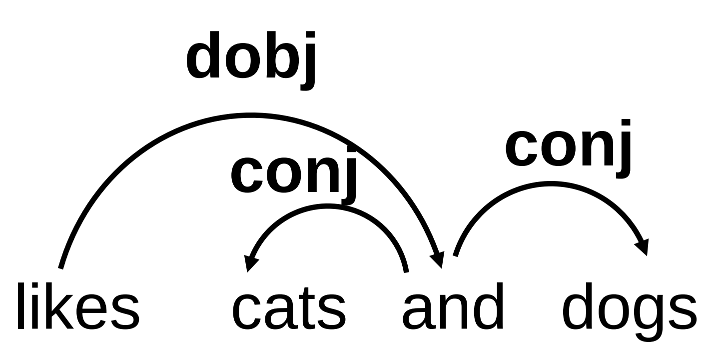




- How to deal with conjunctions?







Inventory of Relations

"Universal Dependencies" (Marneffe *et al.* 2014)

	Nominals	Clauses	Modifier words	Function Words
Core arguments	nsubj obj iobj	csubj ccomp xcomp		
Non-core dependents	obl vocative expl dislocated	advcl	advmod* discourse	aux cop mark
Nominal dependents	nmod appos nummod	acl	amod	det clf case
Coordination	**MWE**	**Loose**	**Special**	**Other**
conj cc	fixed flat compound	list parataxis	orphan goeswith reparandum	punct root dep

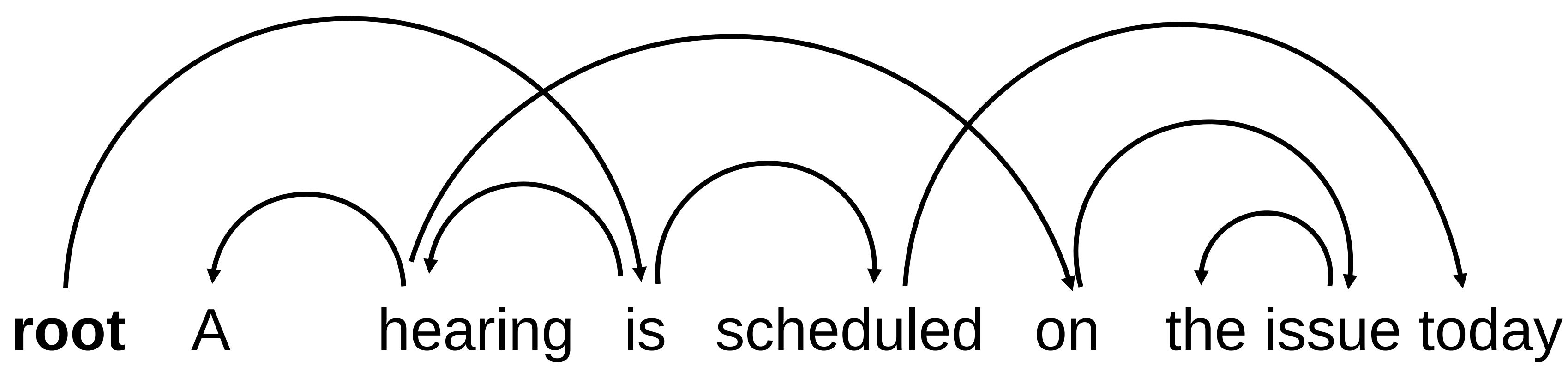
Source: http://universaldependencies.org/u/dep/

Dependency Trees

- Dependency structure is typically assumed to be a tree.
  - Root node 0 must not have a parent.
  - All other nodes must have exactly one parent.
  - The graph needs to be connected.
  - Nodes must not form a cycle.

Projectivity

- Words in a sentence appear in a linear order.
- If dependency edges cross, the dependency structure is non-projective.

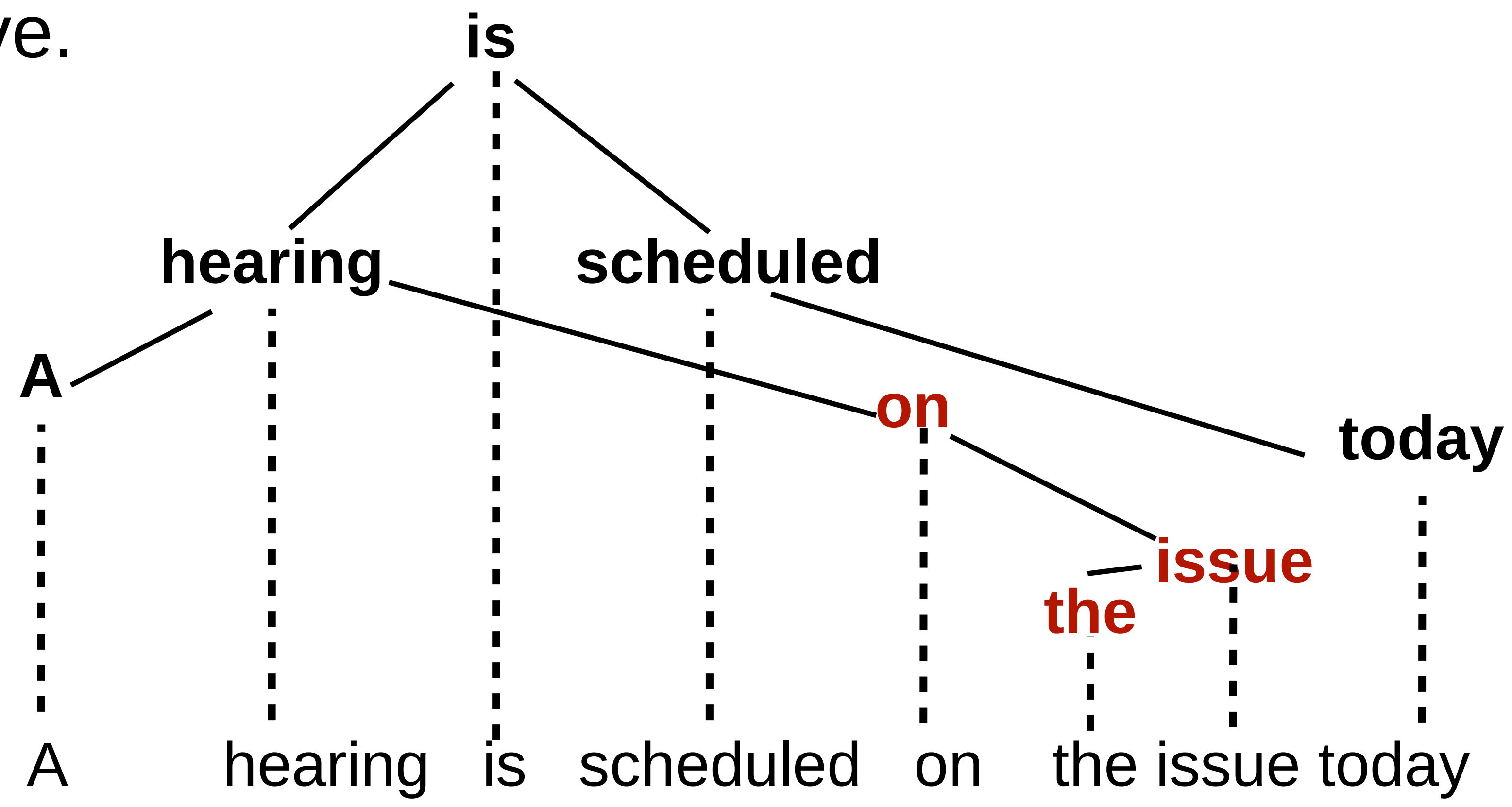


- Non-projective structures appear more frequently in some languages than others (Hungarian, German, ...)
- Some approaches to dependency parsing cannot handle non-projectivity.

Projectivity

- Words in a sentence stand in a linear order.

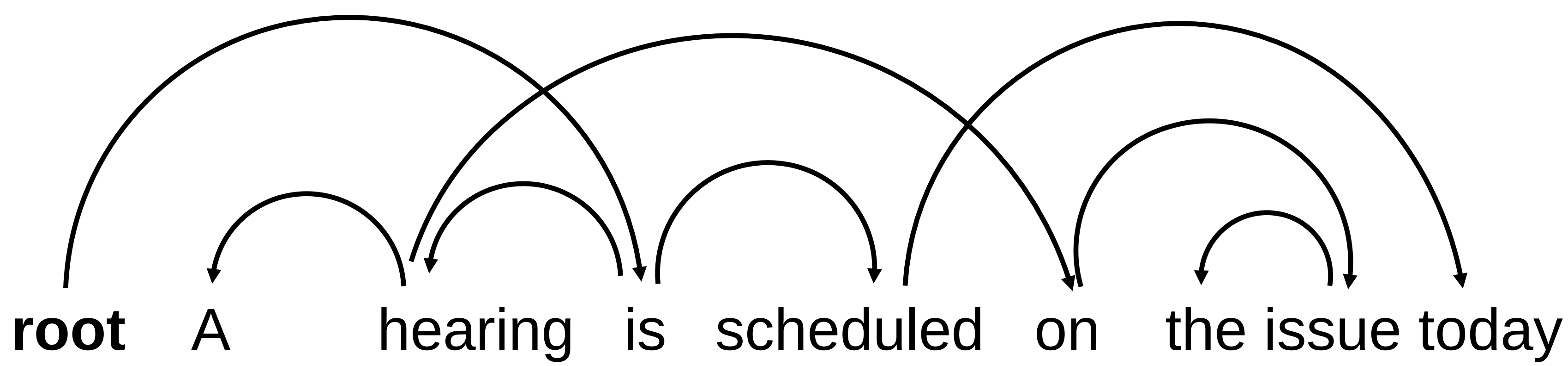
- If dependency edges cross, the dependency structure is non-projective.



- Non-projective structures appear more frequently in some languages than others (Hungarian, German, ...)

- Some approaches to dependency parsing cannot handle non-projectivity.

Projectivity



An edge (i, r, j) in a dependency tree is projective if there is a directed path from i to k for all $i < k < j$ (if $i < j$)

or all $j < k < i$ ($j < i$).

Dependency Parsing

- Input:
  - a set of nodes $V_s = \{w_0, w_1, ..., w_m\}$ corresponding to the input sentence $s = w_1, ..., w_m$ (0 is the special **root** node)
  - an inventory of labels $R = \{PRED, SBJ, OBJ, ATT, ... \}$
- Goal: Find a set of labeled, directed edges between the nodes, such that the resulting graph forms a **correct dependency tree** over V_s.

structural constraints

Dependency Parsing

- What information could we use?
  - bi-lexical affinities
    - *financial markets, meeting... scheduled*
  - dependency distance (prefer close words?)
  - Intervening words
    - *had little effect*, *little gave effect*
  - subcategorization/valency of heads.

Subcategorization/Valency

- Verbs may take a different number of arguments of different syntactic types in different positions:
  - *The baby slept.* ** The baby slept the house.*
  - *He pretended to sleep.* **He pretended the cat.*
  - *Godzilla destroyed the city.* **Godzilla destroyed.*
  - *Jenny gave the book to Carl.* **Jenny gave the book.*
  - *...* examples for *ask*, *promise*, *bet, load,...*

Dependency Parsing

- As with other NLP problems, we can think of dependency parsing as a kind of search problem:
  - Step 1: Define the space of possible analyses for a sentence
  - Step 2: Select the best analysis from this search space.
- Need to define the search space, search algorithm, and a way to determine the "best" parse.

Dependency Parsing

- Approaches to Dependency Parsing:
  - Grammar-based
  - Data-based
    - Dynamic Programming (e.g. Eisner 1996,)
    - Graph Algorithms (e.g. McDonald 2005, MST Parser)
    - **Transition-based (e.g. Nivre 2003, MaltParser)**
    - Constraint satisfaction (Karlsson 1990)

Transition-Based Dependency Parsing

- Defines the search space using parser states (configurations) and operations on these states (transitions).

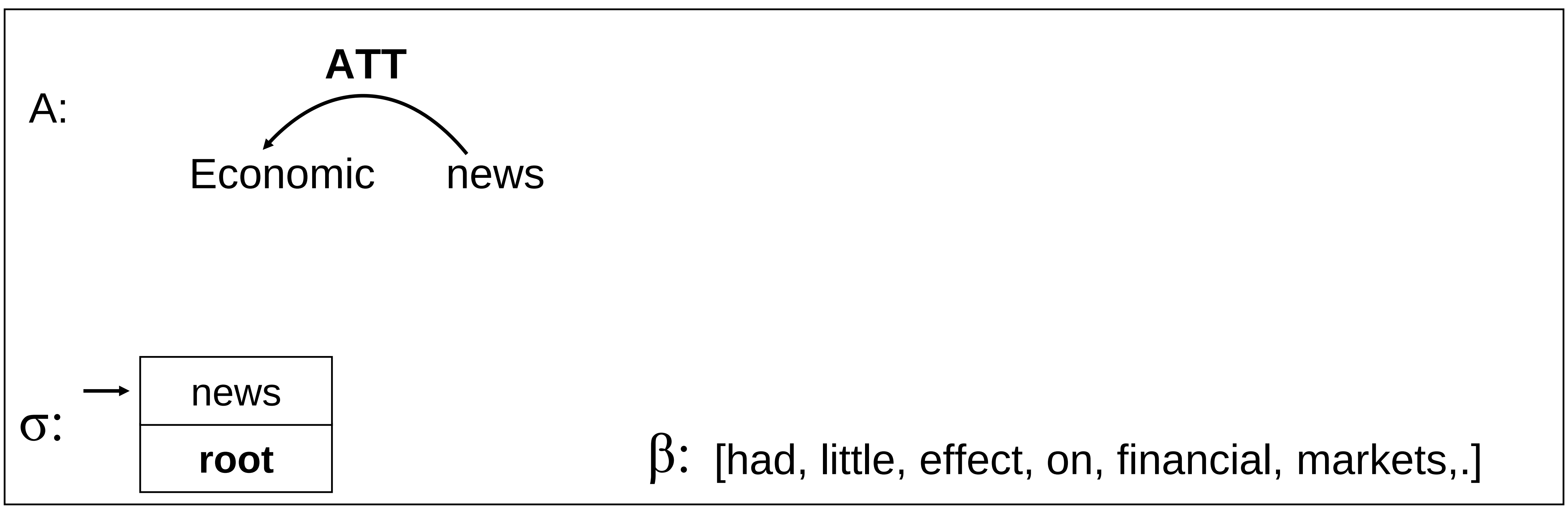
- Start with an initial configuration and find a sequence of transitions to the terminal state.

- Uses a greedy approach to find the best sequence of transitions.

    - Uses a discriminative model (classifier) to select the next transition.

Transition-Based Parsing - States

- A parser state (configuration) is a triple $c = (\sigma, \beta, A)$
  - σ - is a **stack** of words $w_i \in V_S$
  - β - is a **buffer** of words $w_i \in V_S$
  - A - is a set of dependency arcs (w_i, r, w_j)



([**root**, news]$_\sigma$, [had, little, effect, on, financial, markets,.]$_\beta$, { (news,ATT,Economic) }$_A$

Transition-Based Parsing - initial and terminal state

c_0: $([w_0]_\sigma, [w_1, w_2, \ldots, w_m]_\beta, \{\}_A) \xrightarrow{t_0} c_1 \xrightarrow{t_1} c_2 \xrightarrow{t_2} \ldots \longrightarrow c_{n-1} \xrightarrow{t_{n-1}}$ c_T: $([\sigma, [\ ]_\beta, A)$

initial state

transitions

terminal state
(for any σ and A)

- Start with initial state c_0.
- Apply sequence of transitions, $t_0, \ldots, t_{n-1}$.
- Once a terminal state C_T is reached, return final parse A from state C_T.

Transition-Based Parsing - Transitions ("Arc-Standard")

- **Shift**:
  Move next word from the buffer to the stack

  $$(\sigma, w_i \mid \beta, A) \quad \Rightarrow \quad (\sigma \mid w_i, \beta, A)$$

- **Left-Arc (for relation r):**
  Build an edge from the next word on the buffer to the top word on the stack.

  $$(\sigma \mid w_i, w_j \mid \beta, A) \quad \Rightarrow \quad (\sigma, w_j \mid \beta, A \cup \{w_j, r, w_i\})$$

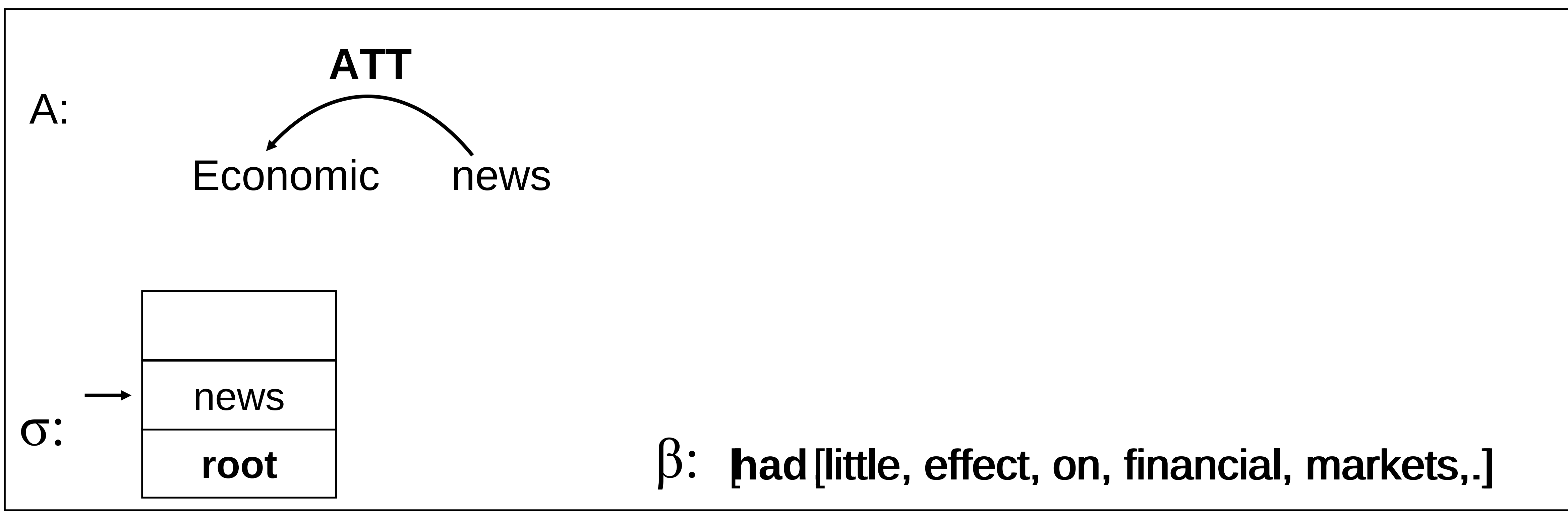
- **Right-Arc (for relation r)**
  Build an edge from the top word on the stack to the next word on the buffer.

  $$(\sigma \mid w_i, w_j \mid \beta, A) \quad \Rightarrow \quad (\sigma, w_i \mid \beta, A \cup \{w_i, r, w_j\})$$

Transition-Based Parsing - Transitions

- **Shift**
  Move next word from the buffer to the stack

$$(\sigma, w_i \mid \beta, A) \quad \Rightarrow \quad (\sigma \mid w_i, \beta, A)$$

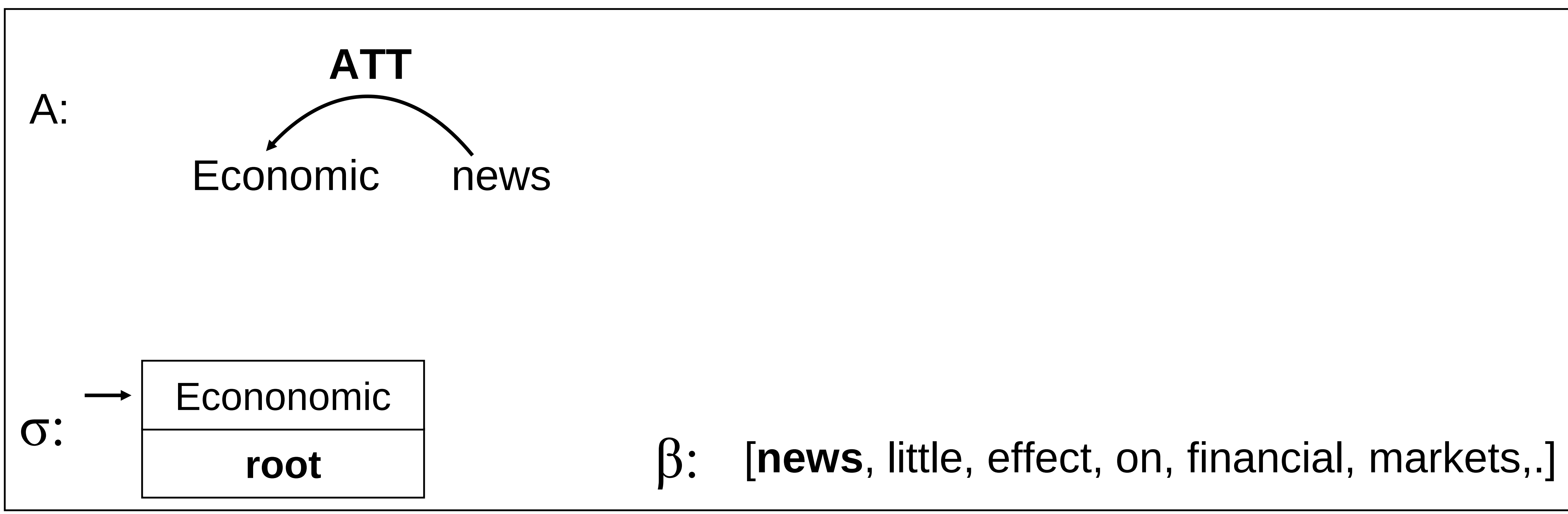


Transition-Based Parsing - Transitions

- **Arc-left$_r$**
  Build an edge from the next word on the buffer to the top word on the stack.

$$(\sigma \mid w_i, w_j \mid \beta, A) \quad => \quad (\sigma, w_j \mid \beta, A \cup \{w_j, r, w_i\})$$

Not allowed if i=0 (root may not have a parent)

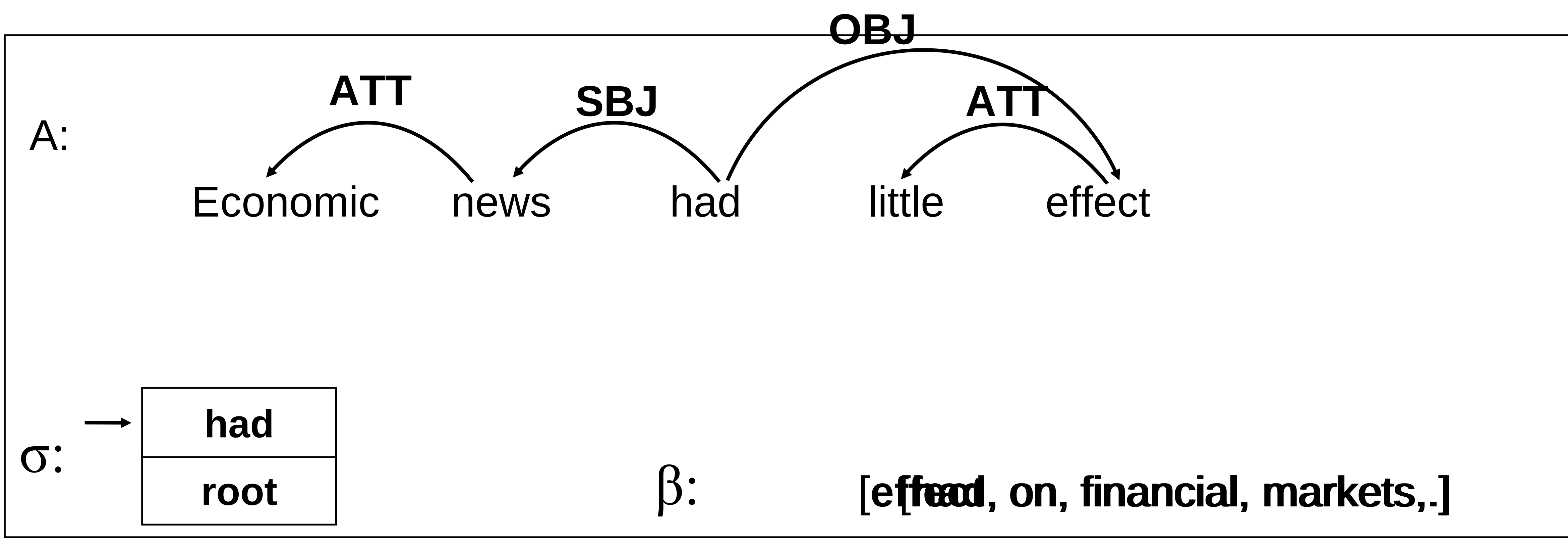


note: w_j remains in the buffer

Transition-Based Parsing - Transitions

- **Arc-right$_r$**
  Build an edge from the next word on the buffer to the top word on the stack.

$$(\sigma \mid w_i, w_j \mid \beta, A) \quad => \quad (\sigma, w_i \mid \beta, A \cup \{w_i, r, w_j\})$$

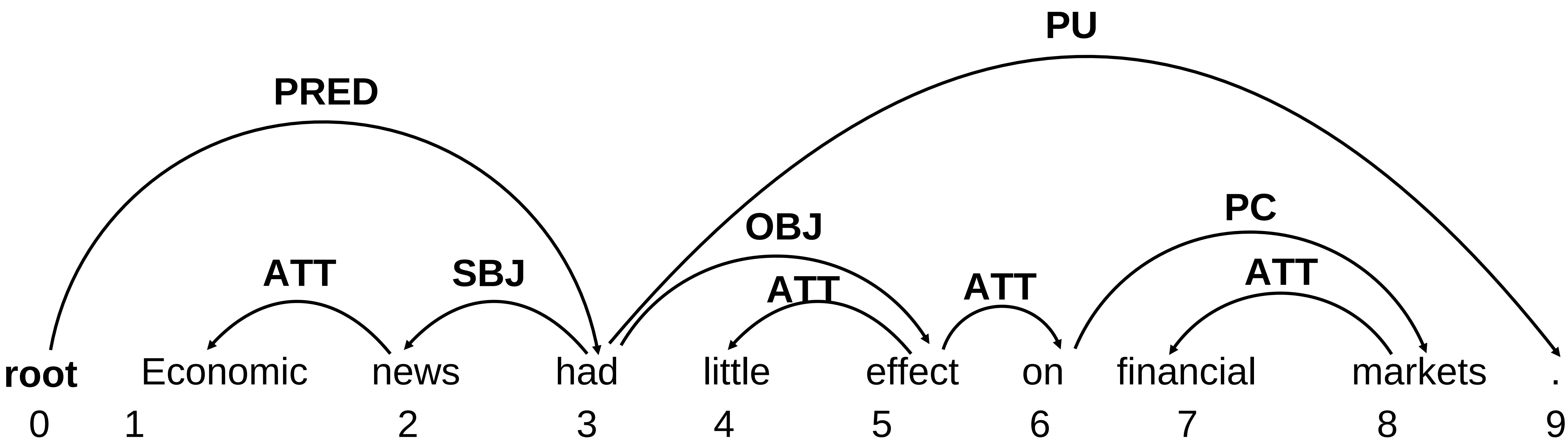


note: w_i is moved from the top of the stack back to the buffer!

Transition-Based Parsing - Some Observations

- Does the transition system contain dead ends? (states from which a terminal state cannot be reached)? No**!**
- What is the role of the buffer?
  - Contains words that can become dependents of a right-arc. Keep unseen words.
- What is the role of the stack?
  - Keep track of nodes that can become dependents of a left-arc.
- Once a word disappears from the buffer and the stack it cannot be part of any further edge!

Another Example



G = (V_S, A)

V_S= {**root**, Economic, news, had, little, effect, on, financial, markets, . }

A = {(root, PRED, had), (had, SBJ, news), (had, OBJ, effect),(had, PU, .),
(news,ATT,Economic),(effect,ATT,little),(effect,ATT,on), (on,PC,markets),
(markets, ATT, financial)}

Transition-Based Parsing - Complete Example

initial state

next transition: shift (these are all predicted by discriminative ML classifier)

A:

σ: **root** β: [Economic, news, had, little, effect, on, financial, markets,.]

Transition-Based Parsing - Oracle Example

next-transition: Left-Arc$_{\textbf{ATT}}$

A:

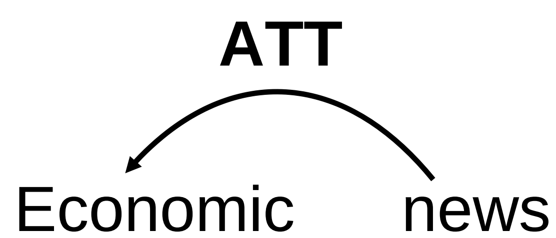
σ:
Economic
root

β: [news, had, little, effect, on, financial, markets,.]

Transition-Based Parsing - Oracle Example

next transition: shift

A:

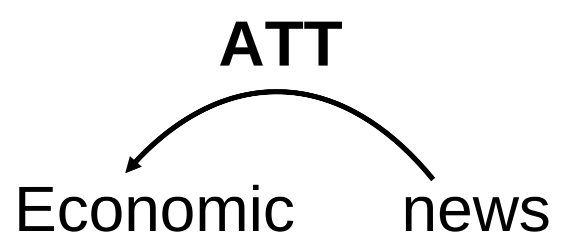


σ: **root** β: [news, had, little, effect, on, financial, markets,.]

Transition-Based Parsing - Oracle Example

next transition: Left-Arc$_{\mathbf{SBJ}}$

A:



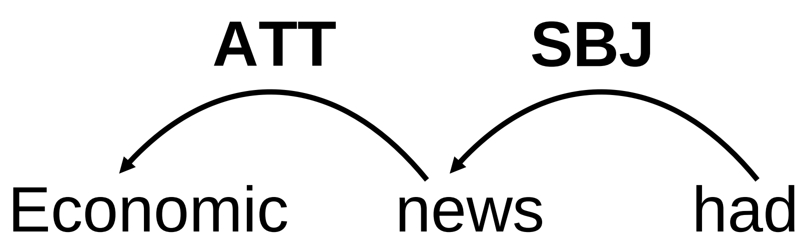
σ:
news
root

β: [had, little, effect, on, financial, markets,.]

Transition-Based Parsing - Oracle Example

next transition: shift

A:

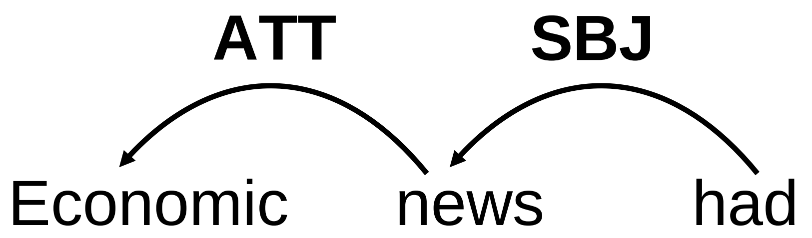


σ: **root** β: [had, little, effect, on, financial, markets,.]

Transition-Based Parsing - Oracle Example

next transition: shift

A:



had
root

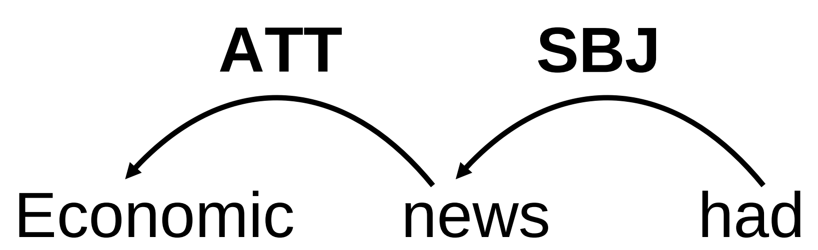
σ:

β: [little, effect, on, financial, markets,.]

Transition-Based Parsing - Oracle Example

next transition: Left-Arc$_{\mathbf{SBJ}}$

A:



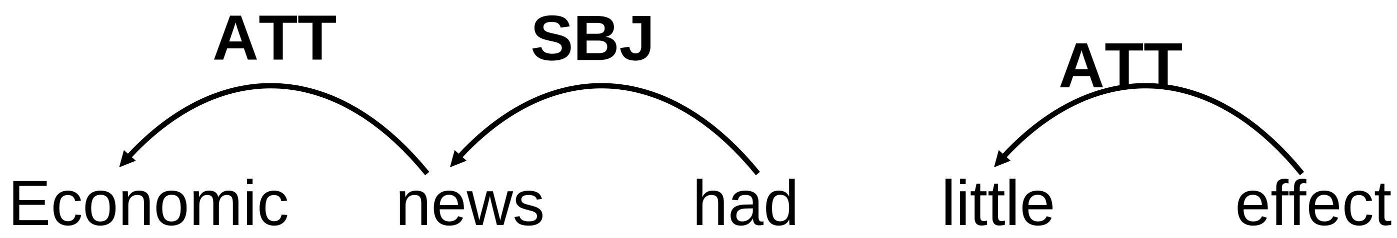
σ:
little
had
root

β: [effect, on, financial, markets,.]

Transition-Based Parsing - Oracle Example

next transition: shift

A:



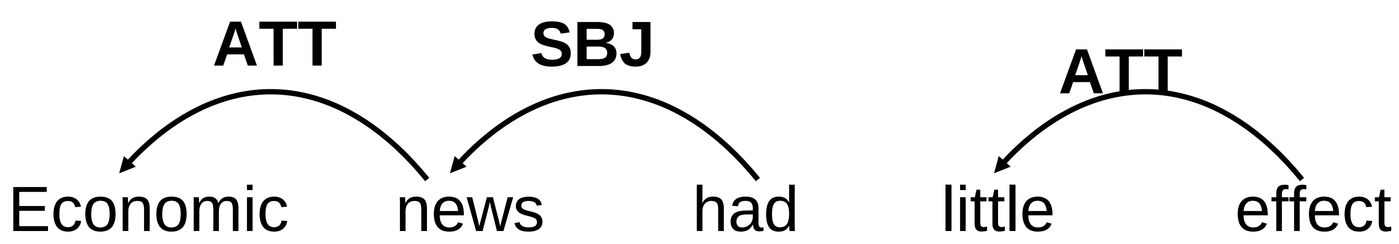
σ:
had
root

β: [effect, on, financial, markets,.]

Transition-Based Parsing - Oracle Example

next transition: shift

A:



σ:

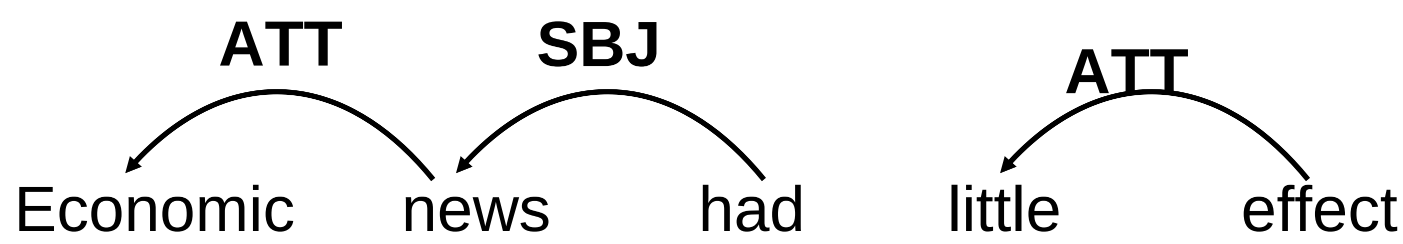
effect
had
root

β: [on, financial, markets,.]

Transition-Based Parsing - Oracle Example

next transition: shift

A:



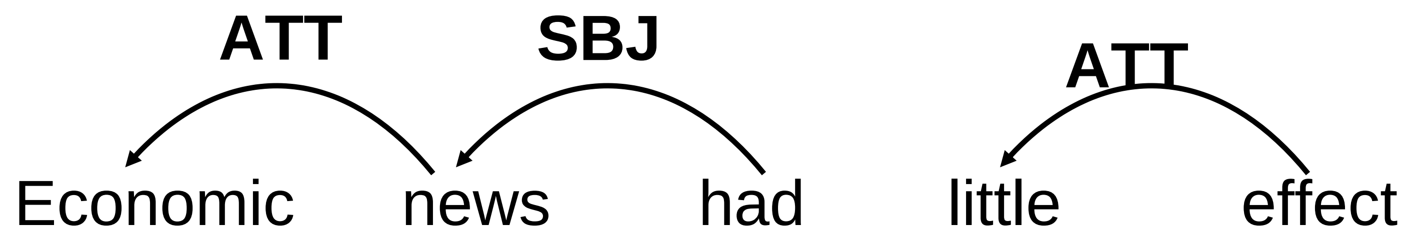
σ:
on
effect
had
root

β: [financial, markets,.]

Transition-Based Parsing - Oracle Example

next transition: Left-Arc$_{\text{ATT}}$

A:



σ:

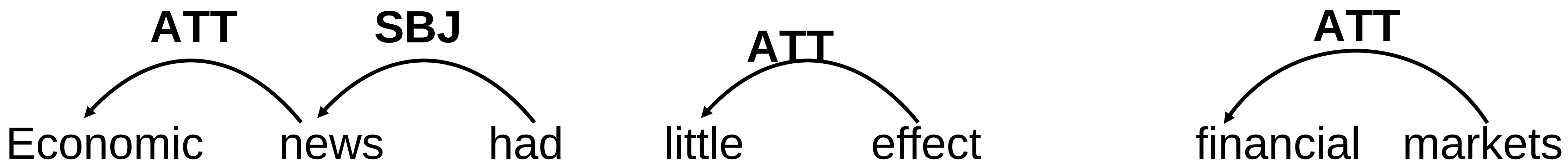
financial
on
effect
had
root

β: [markets,.]

Transition-Based Parsing - Oracle Example

next transition: Right-Arc$_{PC}$

A:



on
effect
had
root

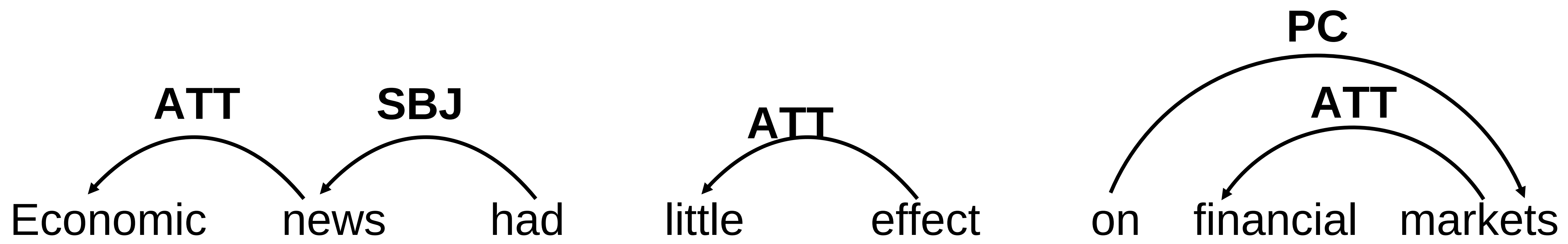
σ:

β: [markets,.]

Transition-Based Parsing - Oracle Example

next transition: Right-Arc$_{\mathrm{OBJ}}$

A:



σ:

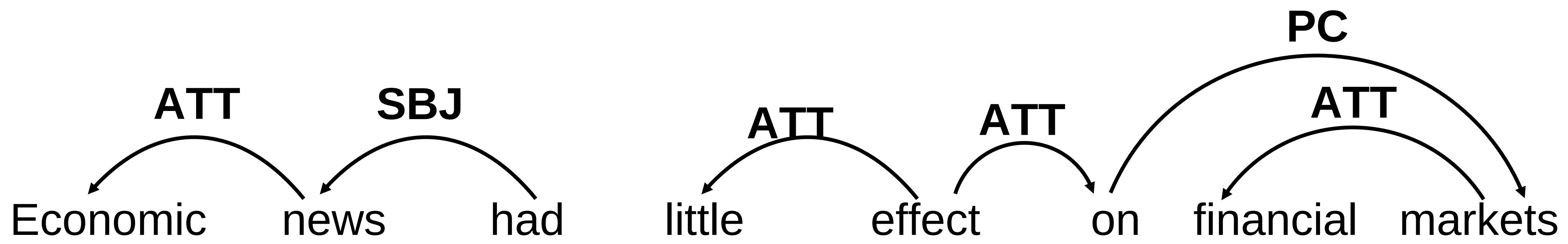
effect
had
root

β: [on,.]

Transition-Based Parsing - Oracle Example

next transition: Right-Arc$_{\mathbf{OBJ}}$

A:



σ:

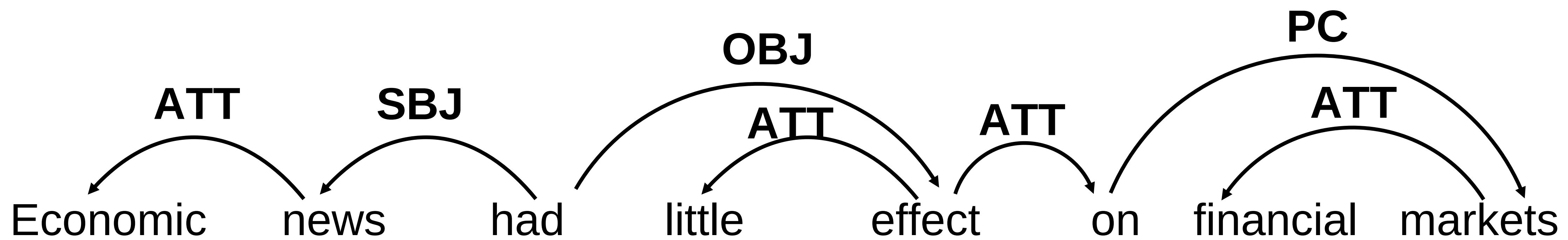
had
root

β: [effect,.]

Transition-Based Parsing - Oracle Example

next transition: shift

A:

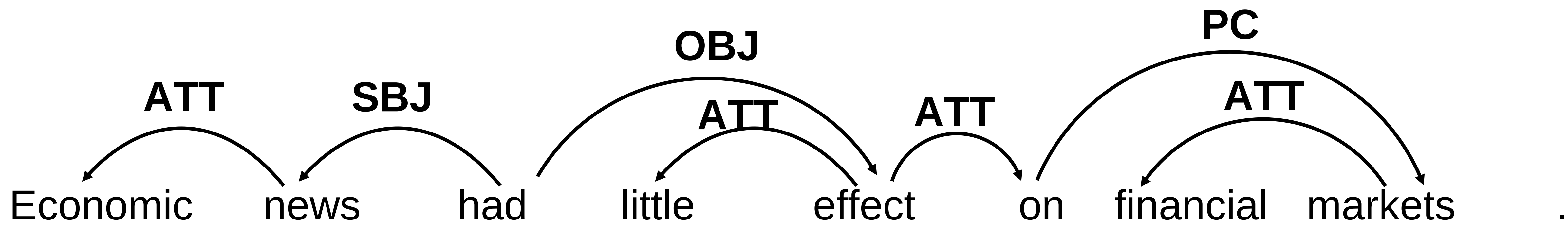


σ: **root** β: [had,.]

Transition-Based Parsing - Oracle Example

next transition: Right-Arc$_{PU}$

A:



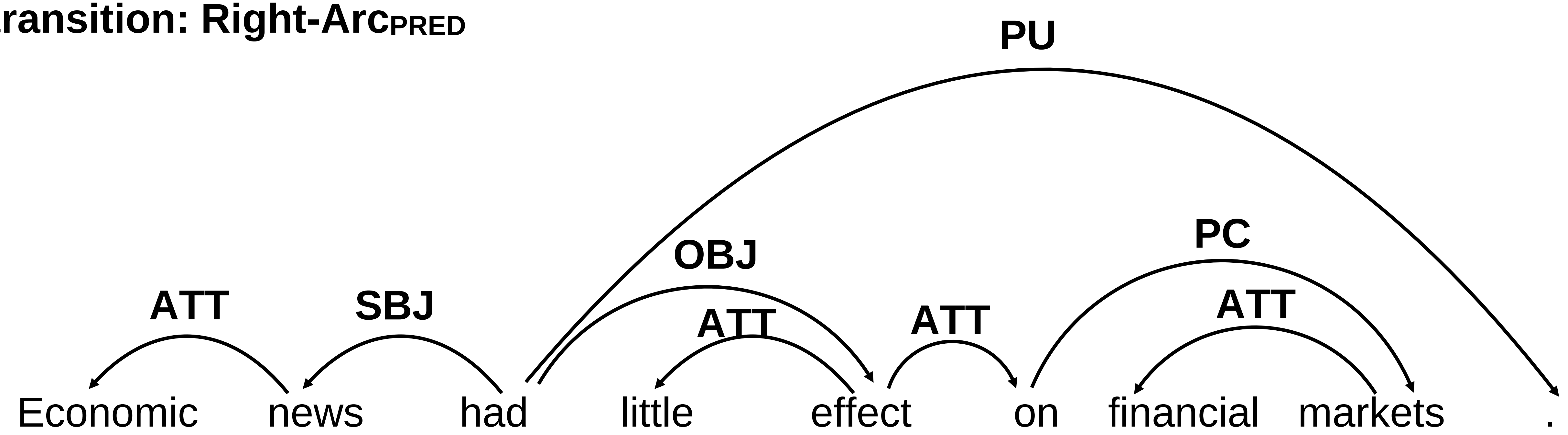
σ: | had | root |

β: [.]

Transition-Based Parsing - Oracle Example

next transition: Right-Arc$_{\text{PRED}}$

A:

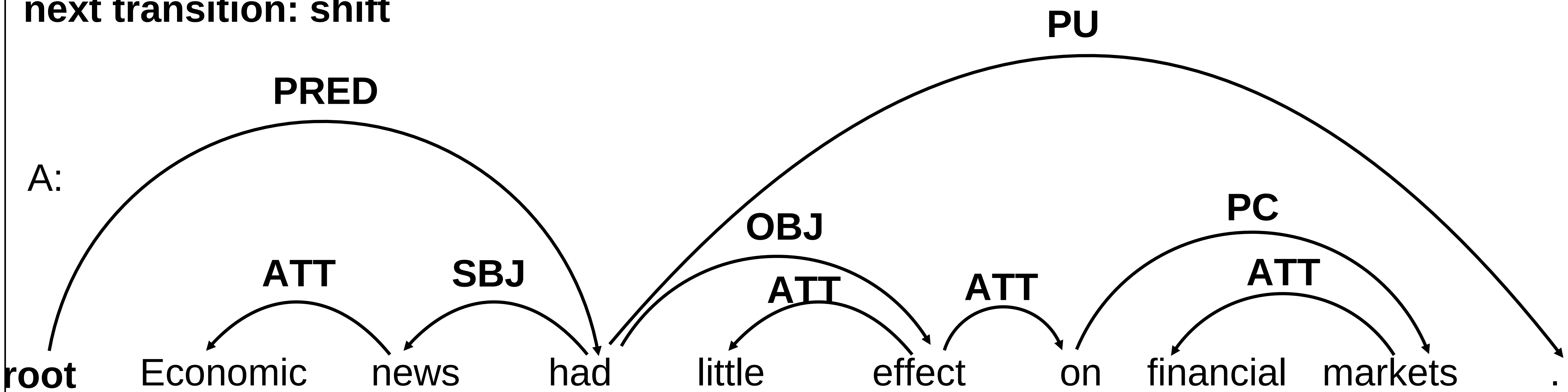


σ: **root** β: [had]

Transition-Based Parsing - Oracle Example

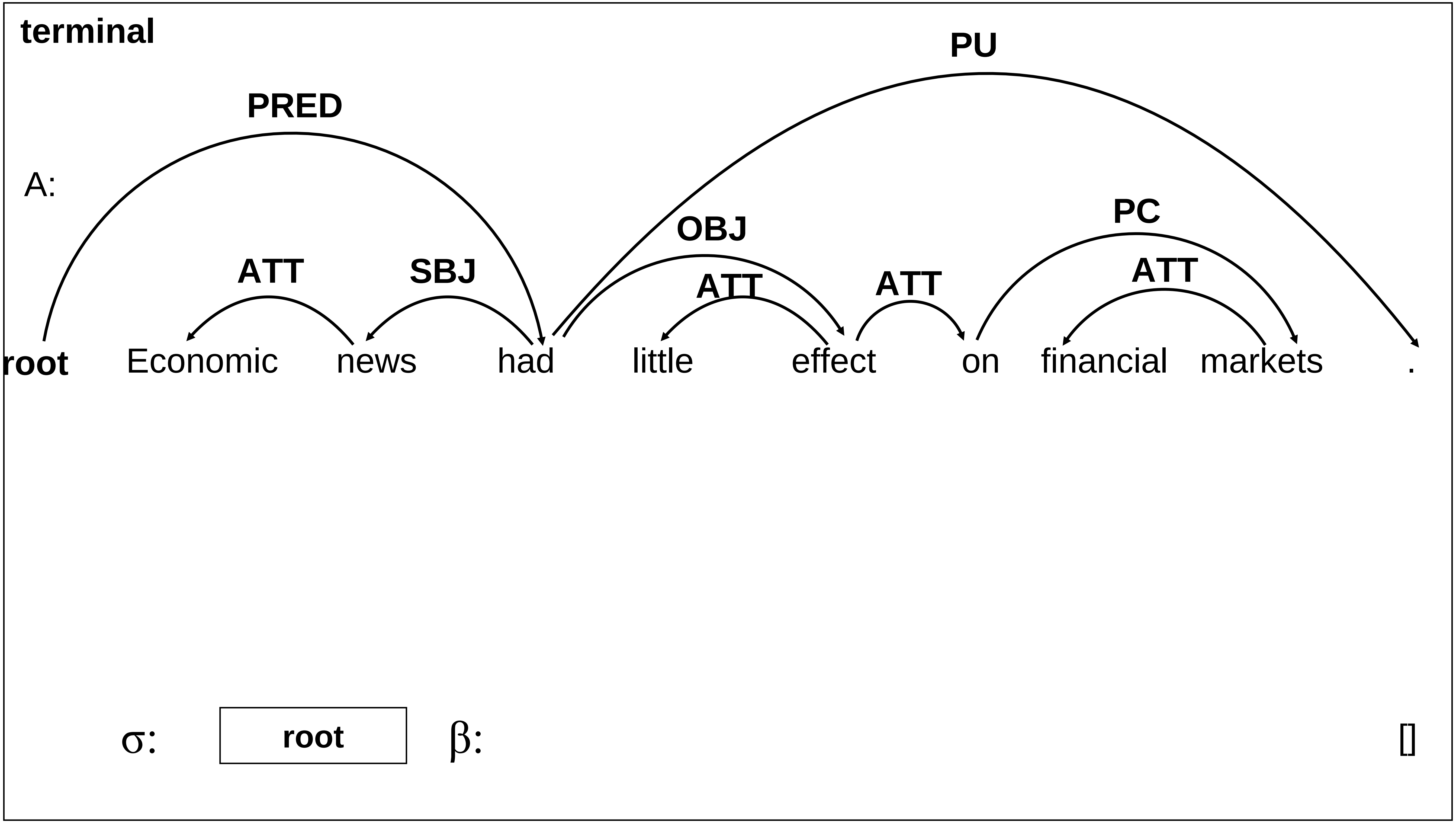
next transition: shift

A:



σ: β: [root]

Transition-Based Parsing - Oracle Example



Properties of the Transition System

- The time required to parse $w_1,\ldots, w_m$ with an oracle is O(m). Why?
- Bottom-up approach: A node must *collect* all its children before its parent. Why?
- Can only produce projective trees. Why?
- This algorithm is complete (all projective trees over $w_1,\ldots, w_m$ can be produced by some sequence of transitions)
- Soundness: All terminal structures are projective forests (but not necessarily trees)

Deciding the Next Transition

- Instead of the unrealistic oracle, predict the next transition (and relation label) using a discriminative classifier.
  - Could use perceptron, log linear model, SVM, Neural Network, ...
  - This is a greedy approach (could use beam-search too).
  - If the classifier takes O(1), the runtime for parsing is still O(m) for m words.
- Questions:
  - What features should the classifier use?

    Local features from each state (buffer, stack, partial dependency structure)
    ... but ideally want to model entire history of transitions leading to the state.
  - How to train the model?

Extracting Features

- Need to define a feature function that maps states to feature vectors.

- Each feature consists of:

  1. an address in the state description:
     (identifies a specific word in the configuration, for example "top of stack").

  2. an attribute of the word in that address:
     (for example POS, word form, lemma, word embedding, ...)

Example Features

Table 3.2: Typical feature model for transition-based parsing with rows representing address functions, columns representing attribute functions, and cells with + representing features.

	Attributes				
Address	**FORM**	**LEMMA**	**POSTAG**	**FEATS**	**DEPREL**
STK[0]	+	+	+	+	
STK[1]			+		
LDEP(STK[0])					+
RDEP(STK[0])					+
BUF[0]	+	+	+	+	
BUF[1]	+		+		
BUF[2]			+		
BUF[3]			+		
LDEP(BUF[0])					+
RDEP(BUF[0])					+

Source: S. Kübler, R. McDonald, J. Nivre (2009):"Dependency Parsing", Morgan & Claypool

Training the Model

- Training data: Manually annotated (dependency) treebank
  - *Prague Dependency Treebank*
    English/Czech parallel data, dependencies for full PTB WSJ.
  - *Universal Dependencies Treebank*
    Treebanks for more than 80 languages (varying in size)
    (http://universaldependencies.org/)
- **Problem: We have not actually seen the transition sequence, only the dependency trees!**
- Idea: Construct oracle transition sequences from the dependency tree. Train the model on these transitions.

Constructing Oracle Transitions

- Start with initial state ($[\mathbf{w_0}]_\sigma$, $[w_1, w_2, \ldots, w_m]_\beta$, $\{\}_A$).
- Then predict the next transition using the annotated dependency tree A_d

$$o(c = (\sigma, \beta, A)) = \begin{cases} \textsc{Left-Arc}_r & \text{if } (\beta[0], r, \sigma[0]) \in A_d \\ \textsc{Right-Arc}_r & \text{if } (\sigma[0], r, \beta[0]) \in A_d \text{ and, for all } w, r', \\ & \text{if } (\beta[0], r', w) \in A_d \text{ then } (\beta[0], r', w) \in A \\ \textsc{Shift}_r & \text{otherwise} \end{cases}$$

"Arc-Standard" Transitions

- **Shift**:
  Move next word from the buffer to the stack
  $$(\sigma, w_i \mid \beta, A) \quad \Rightarrow \quad (\sigma \mid w_i, \beta, A)$$

- **Left-Arc (for relation r):**
  Build an edge from the next word on the buffer to the top word on the stack.
  $$(\sigma \mid w_i, w_j \mid \beta, A) \quad \Rightarrow \quad (\sigma, w_j \mid \beta, A \cup \{w_j, r, w_i\})$$

- **Right-Arc (for relation r)**
  Build an edge from the top word on the stack to the next word on the buffer.

  $$(\sigma \mid w_i, w_j \mid \beta, A) \quad \Rightarrow \quad (\sigma, w_i \mid \beta, A \cup \{w_i, r, w_j\})$$

"Arc-Eager" Transitions

- **Shift**:
  Move next word from the buffer to the stack
  $$(\sigma, w_i \mid \beta, A) \quad \Rightarrow \quad (\sigma \mid w_i, \beta, A)$$

- **Left-Arc (for relation r):**
  Build an edge from the next word on the buffer to the top word on the stack.
  $$(\sigma \mid w_i, w_j \mid \beta, A) \quad \Rightarrow \quad (\sigma, w_j \mid \beta, A \cup \{(w_j, r, w_i)\})$$
  Precondition: $(w_j, *, w_i)$ is not yet in A.
- **Right-Arc (for relation r)**
  Build an edge from the top word on the stack to the next word on the buffer.
  $$(\sigma \mid w_i, w_j \mid \beta, A) \quad \Rightarrow \quad (\sigma \mid w_i \mid w_j, \beta, A \cup \{w_i, r, w_j\})$$

- **Reduce**
  Remove a completed node from the stack.
  $$(\sigma \mid w_i, \beta, A) \quad \Rightarrow \quad (\sigma, \beta, A)$$

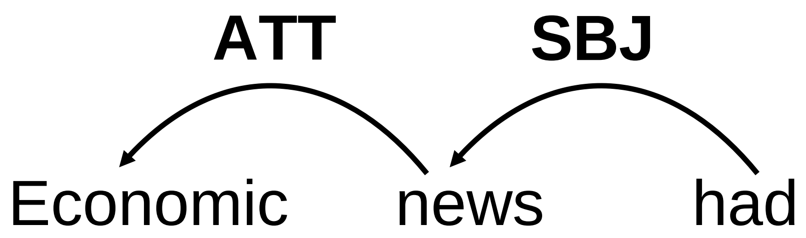
  Precondition: there is some $(*, *, w_i)$ in A.

Arc-Eager Example

next transition: RightArc$_{\mathbf{pred}}$

Can immediately attach *had* to root.

A:

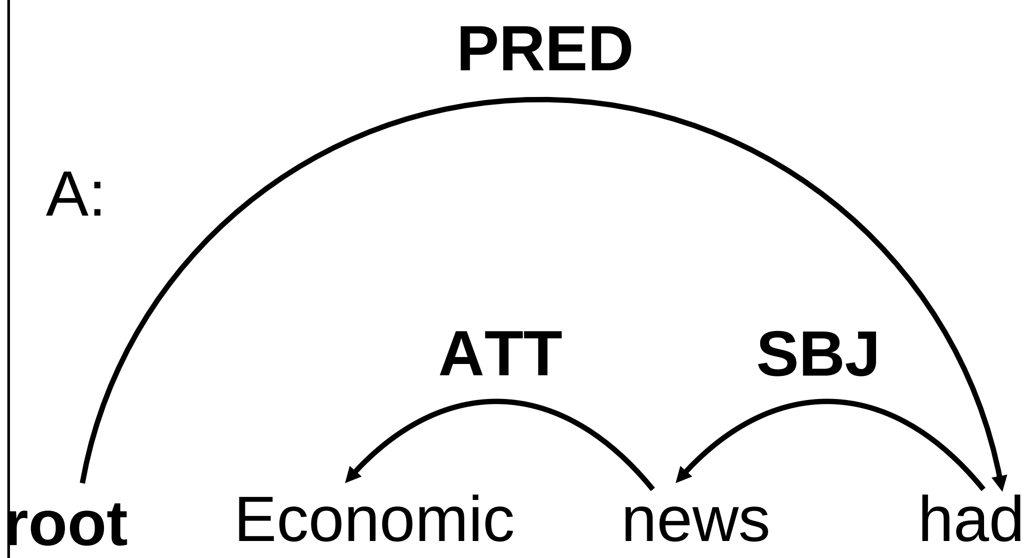


σ: **root** β: [had, little, effect, on, financial, markets,.]

Arc-Eager Example

next transition: shift

A:



σ:

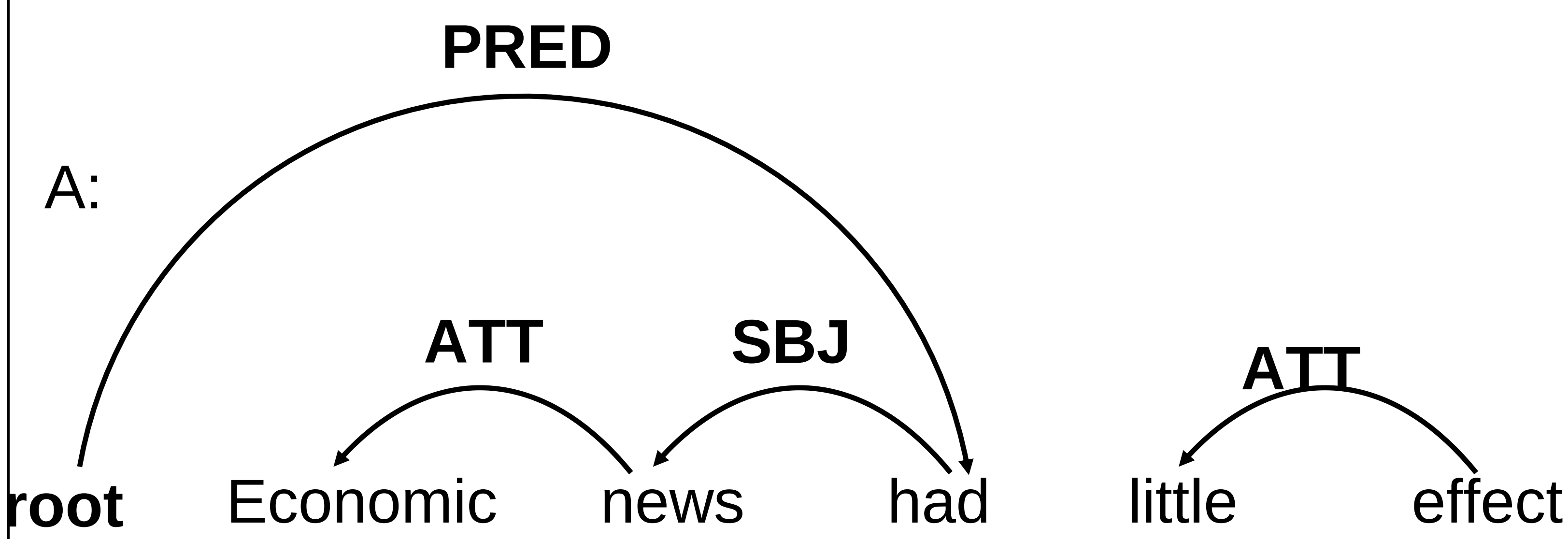
had
root

β: [little, effect, on, financial, markets,.]

Arc-Eager Example

next transition: LeftArc$_{\text{ATT}}$

A:



σ:

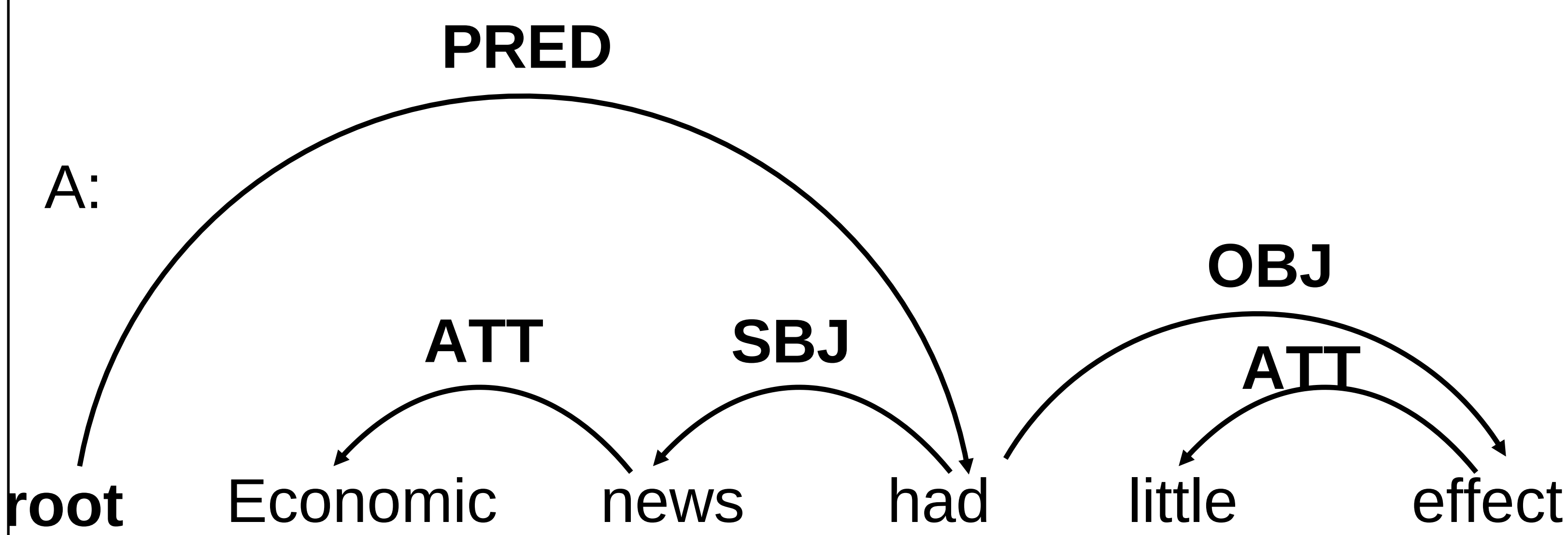
little
had
root

β: [effect, on, financial, markets,.]

Arc-Eager Example

next transition: RightArc$_{\mathbf{OBJ}}$

A:



σ:

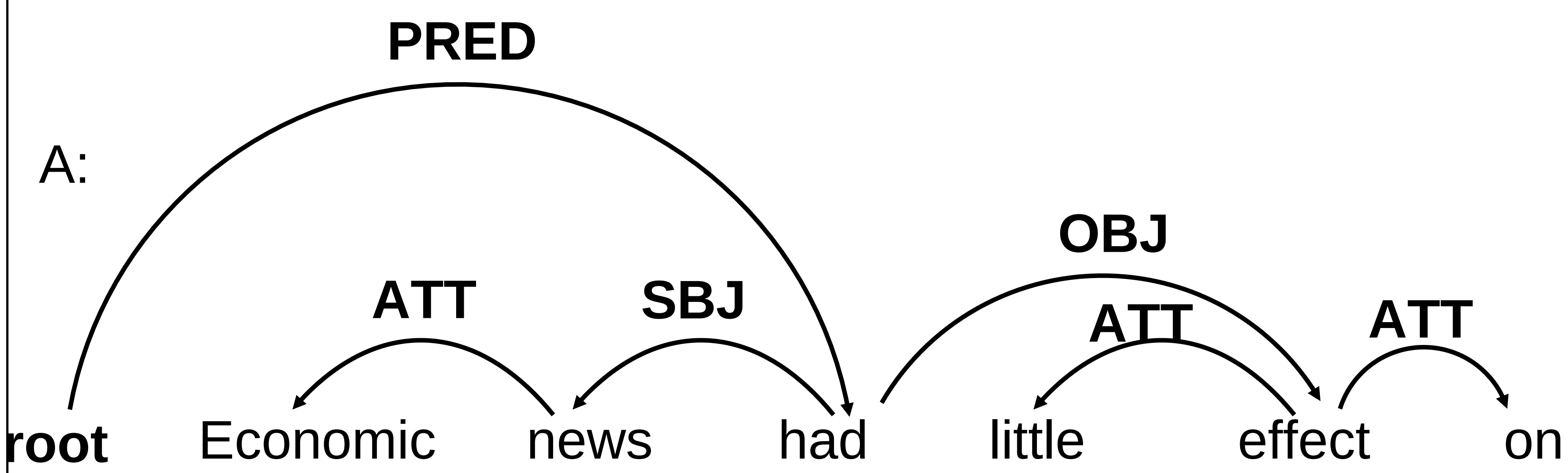
effect
little
had
root

β: [on, financial, markets,.]

Arc-Eager Example

next transition: shift

A:



σ:

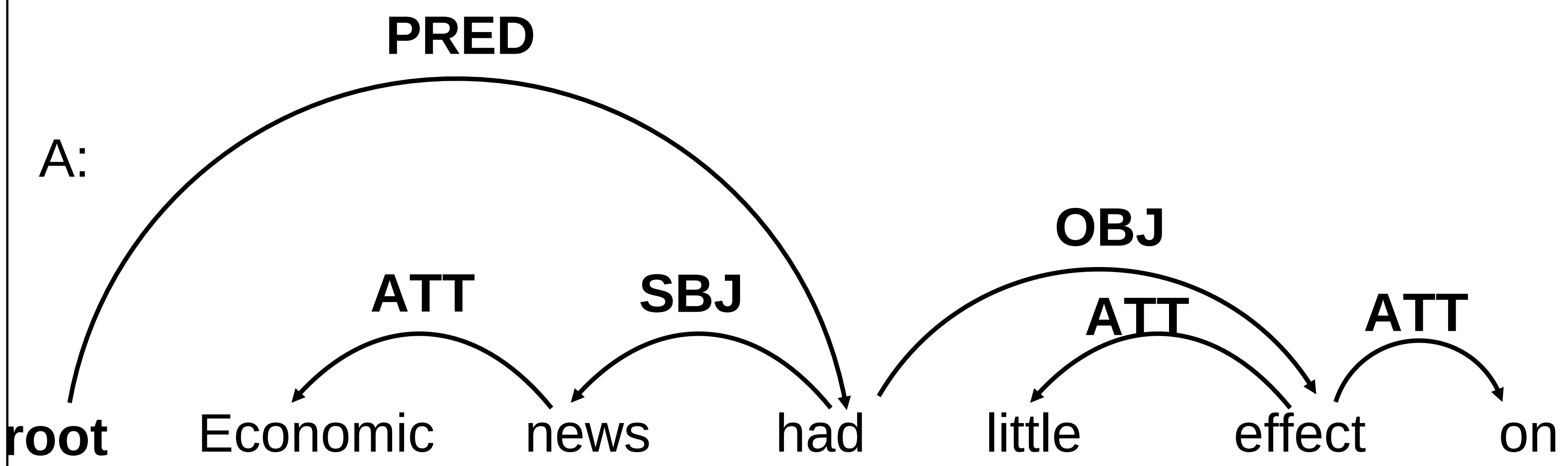
on
effect
little
had
root

β: [financial, markets,.]

Arc-Eager Example

next transition: LeftArc$_{ATT}$

A:



σ:

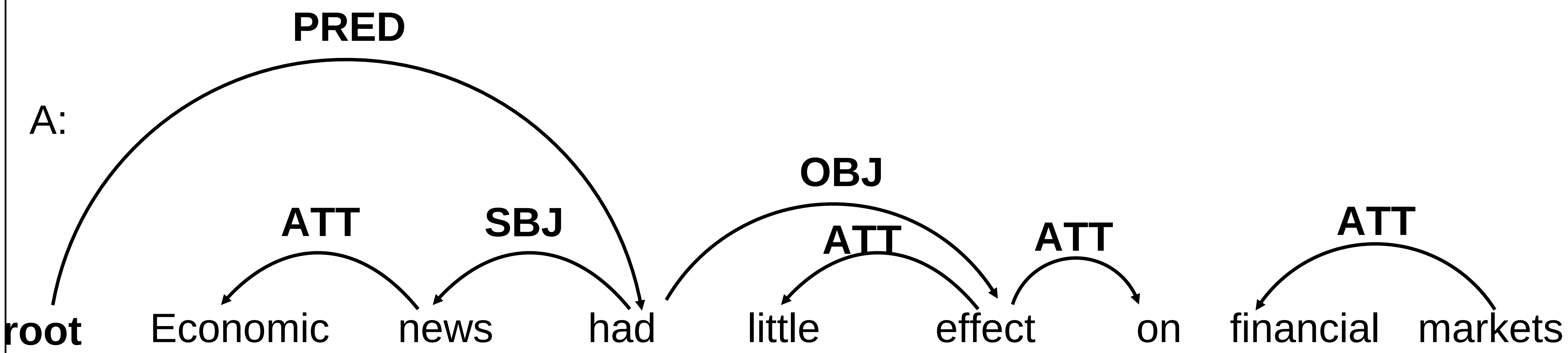
financial
on
effect
little
had
root

β: [markets,.]

Arc-Eager Example

next transition: RightArc$_{PC}$

A:



σ:

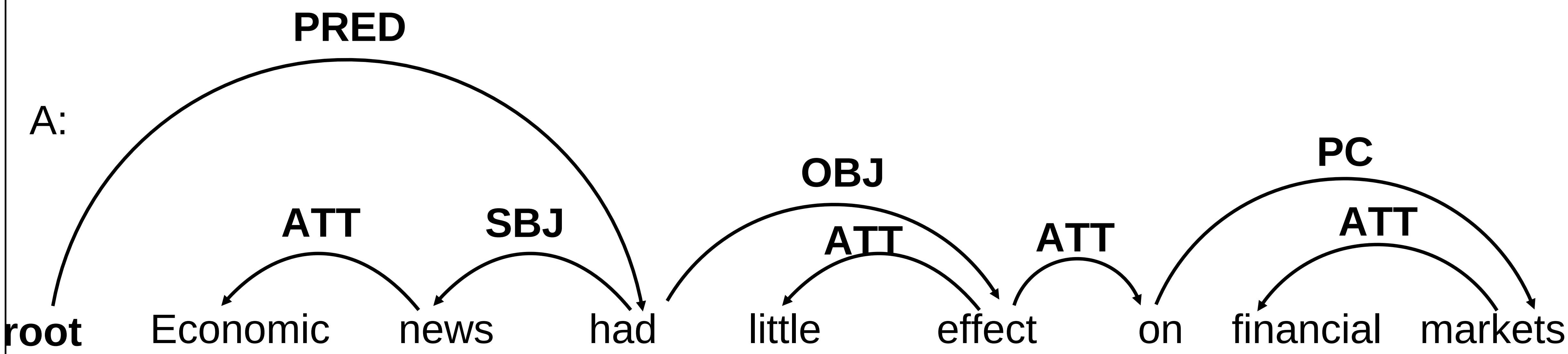
on
effect
little
had
root

β: [markets,.]

Arc-Eager Example

next transition: Reduce

A:



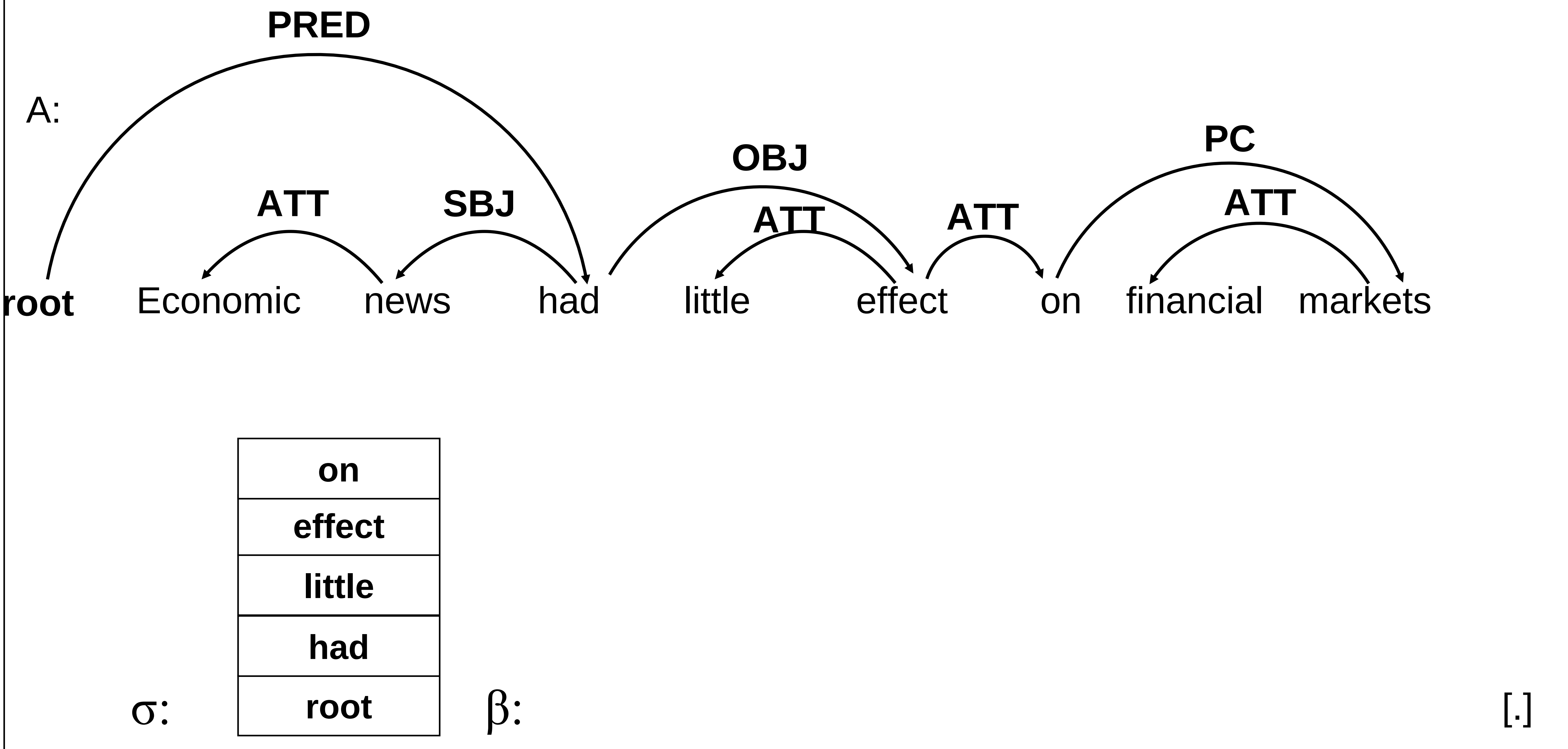
σ:

markets
on
effect
little
had
root

β: [.]

Arc-Eager Example

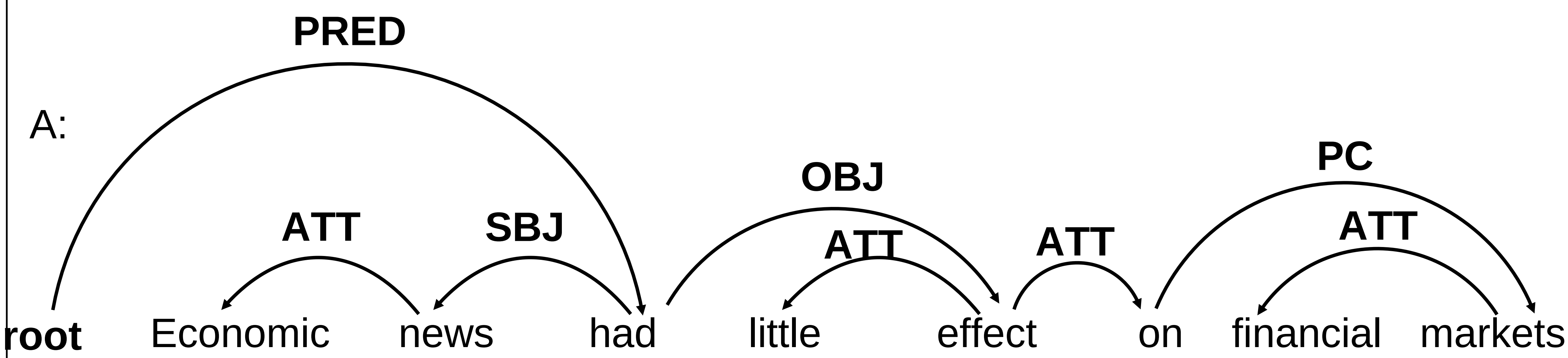
next transition: Reduce



Arc-Eager Example

next transition: Reduce

A:



σ:

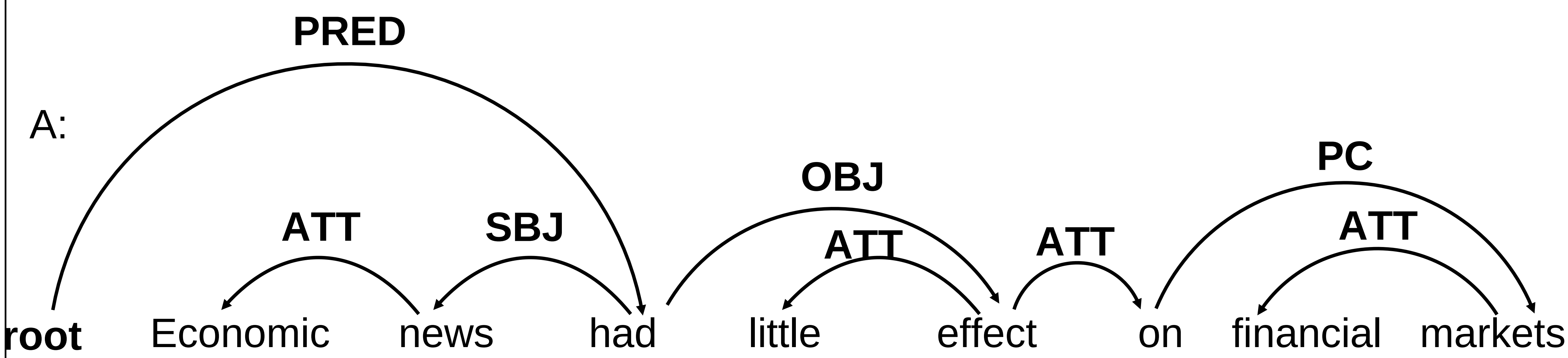
effect
little
had
root

β:

[.]

Arc-Eager Example

next transition: Reduce



σ:

little
had
root

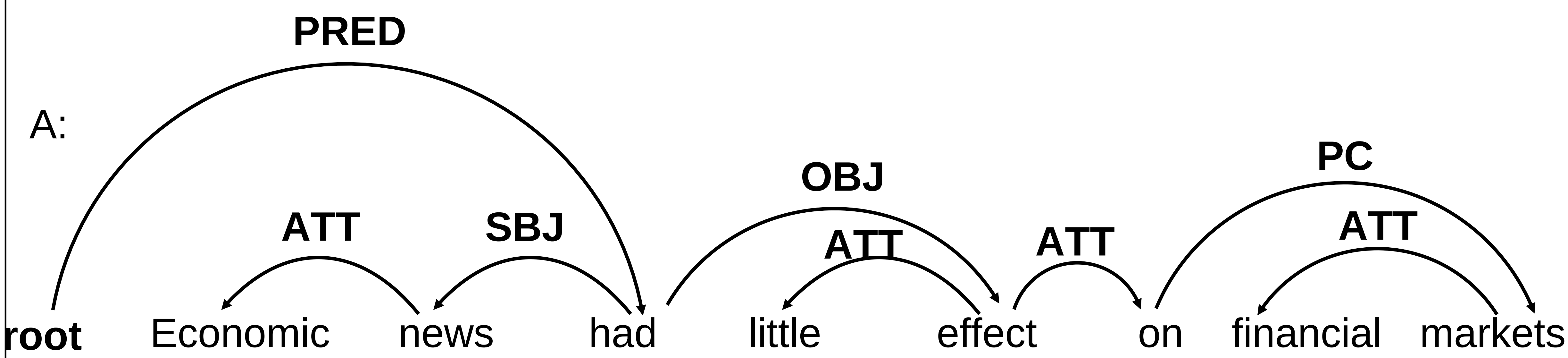
β:

[.]

Arc-Eager Example

next transition: Reduce

A:



σ:

had
root

β: [.]

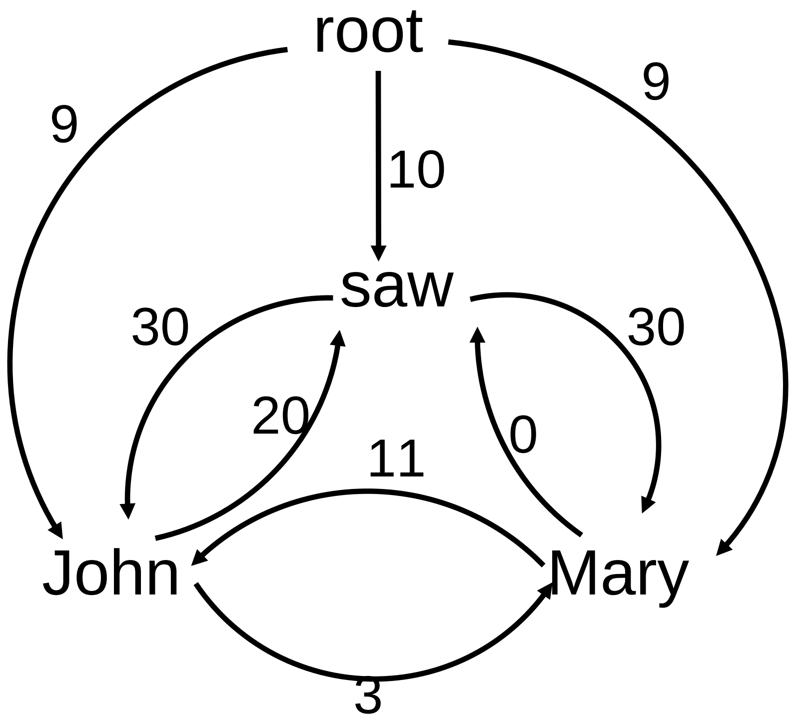
Graph-Based Approach

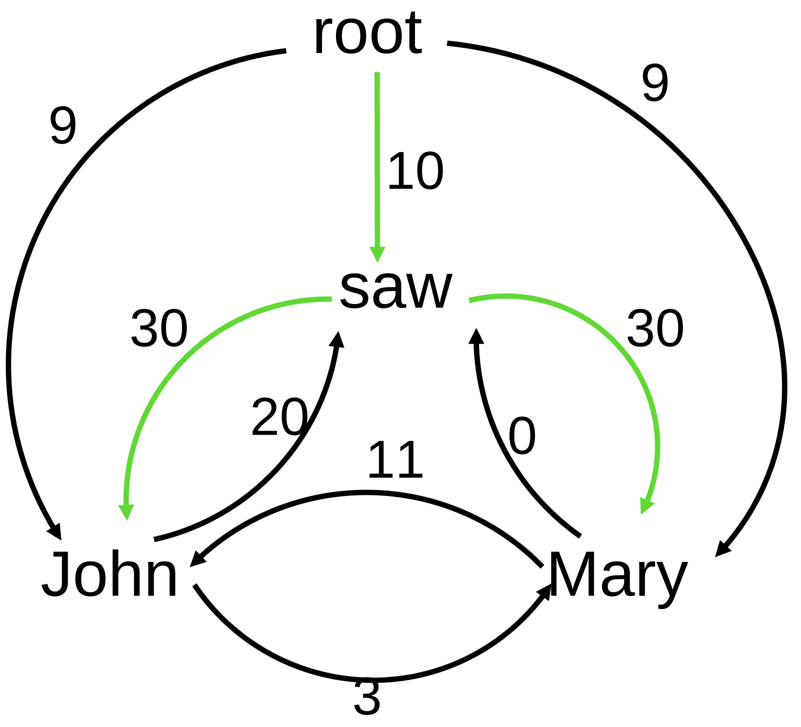
- Transition Based Parsing can only produce projective dependency structures? Why?
- Graph-based approaches do not have this restriction.
- Basic idea:
  - Each word is a vertex. Start with a completely connected graph.
  - Use standard graph algorithms to compute a Maximum Spanning Tree:
    - Need a model that assigns a score to each edge ("edge-factored model").

$$score(G) = \sum_{(w_i, r, w_j) \in A} \lambda(w_i, r, w_j)$$

R. McDonald, K. Crammer, and F. Pereira (2005)

MST Example





total score: 70

Computing the MST

- For undirected graphs, there are two common algorithms:
  - Kruskal's and Prim's, both run in O(E log V)
- For dependency parsing we deal with directed graphs, so these algorithms are not guaranteed to find a tree.
  - Instead use Chu–Liu-Edmonds' algorithm, which runs in O(EV) (naive implementation) or O(E log V) (with optimizations).