UNIVERSITEIT VAN AMSTERDAM

Grammars and Parsing

Software Construction 2018

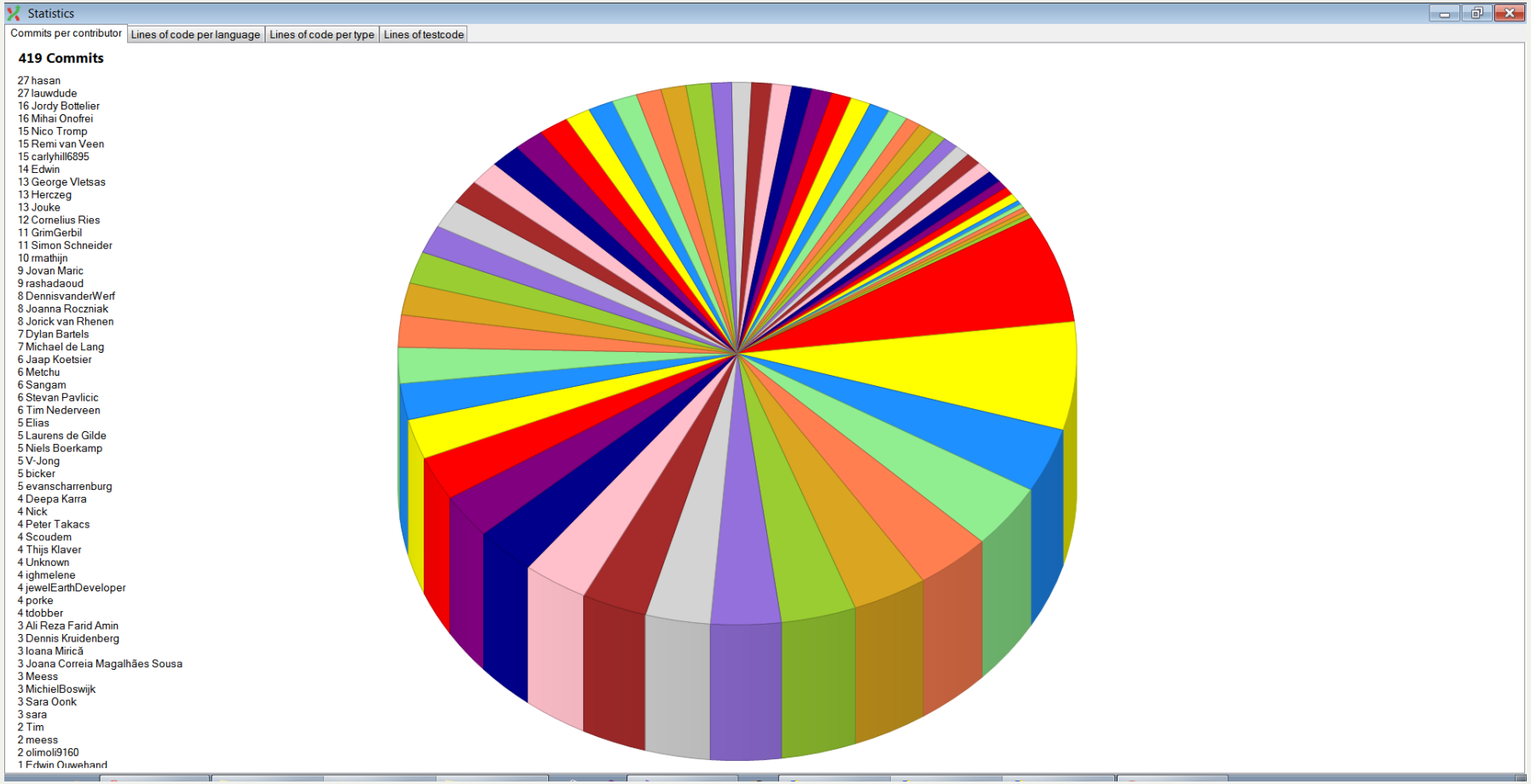
Dr. Vadim Zaytsev aka @grammarware

raincode LABS compiler experts

Test questions

- Link in the email!
  - Design patterns
  - Global variables
  - Lifecycle costs
  - Warnings

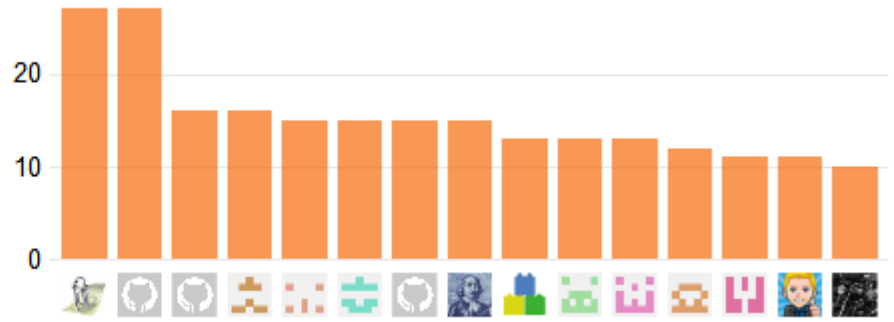
419 commits by 41 people [out of 62]

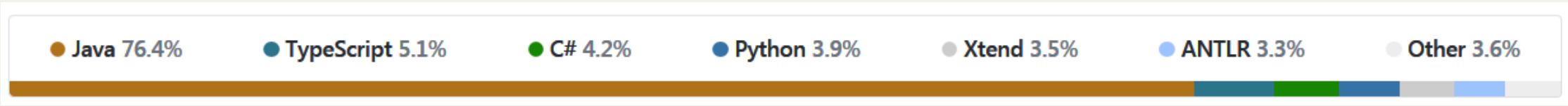


raincode LABS
compiler experts

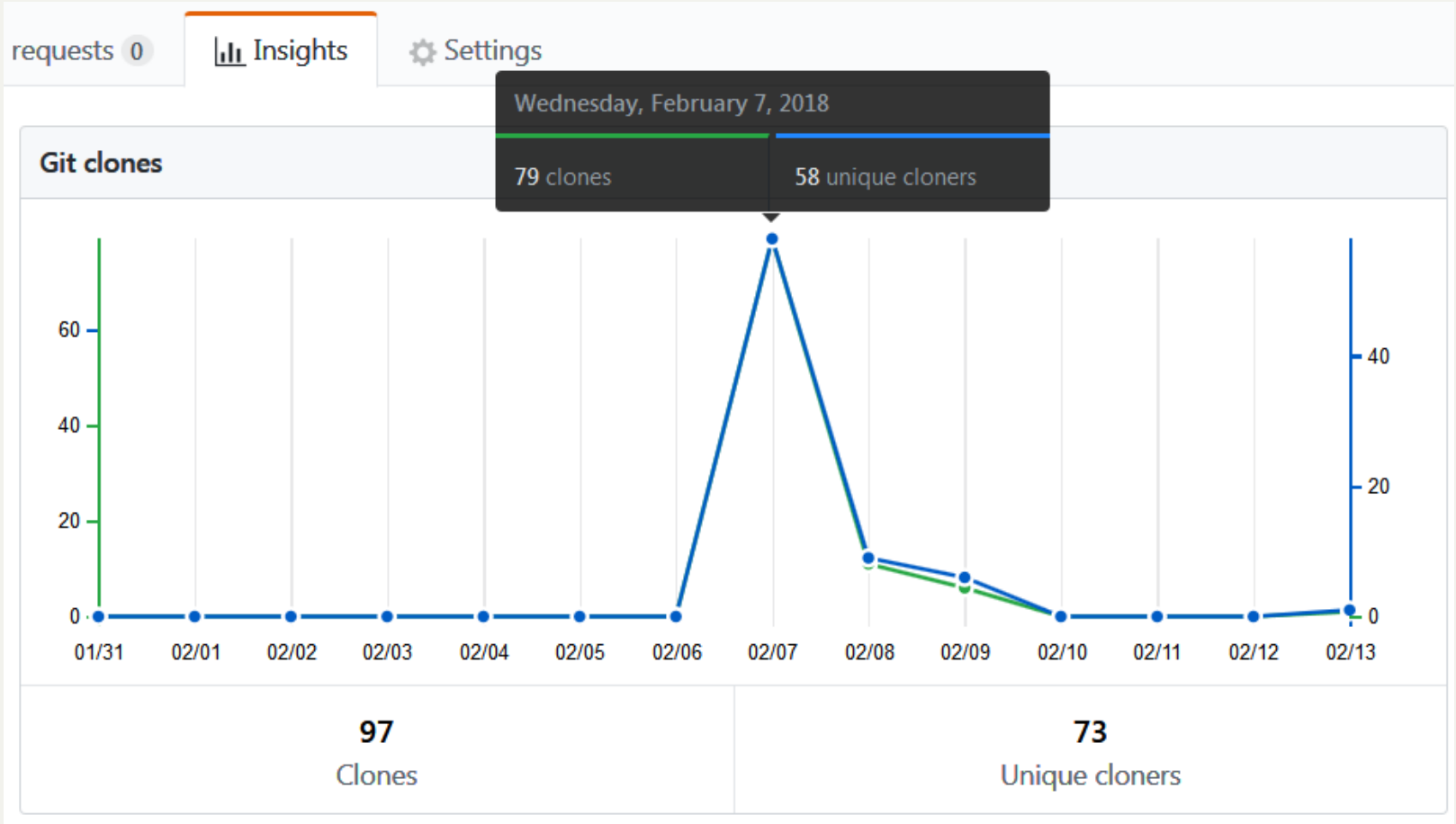
70 kLOC; mostly Java

Excluding merges, **58 authors** have pushed **361 commits** to master and **419 commits** to all branches. On master, **938 files** have changed and there have been **69,418 additions** and **0 deletions**.





79 clones on 7 Feb, 73 unique?

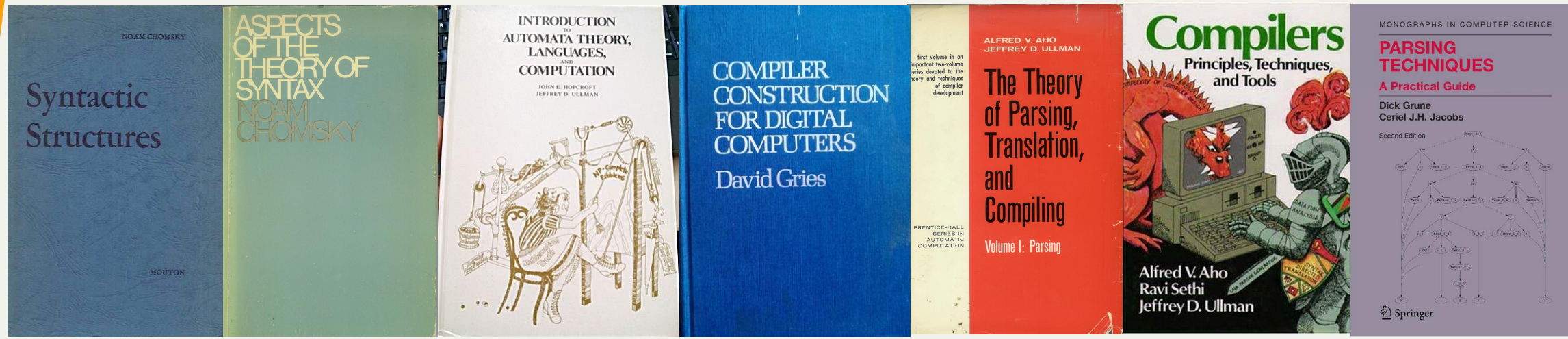


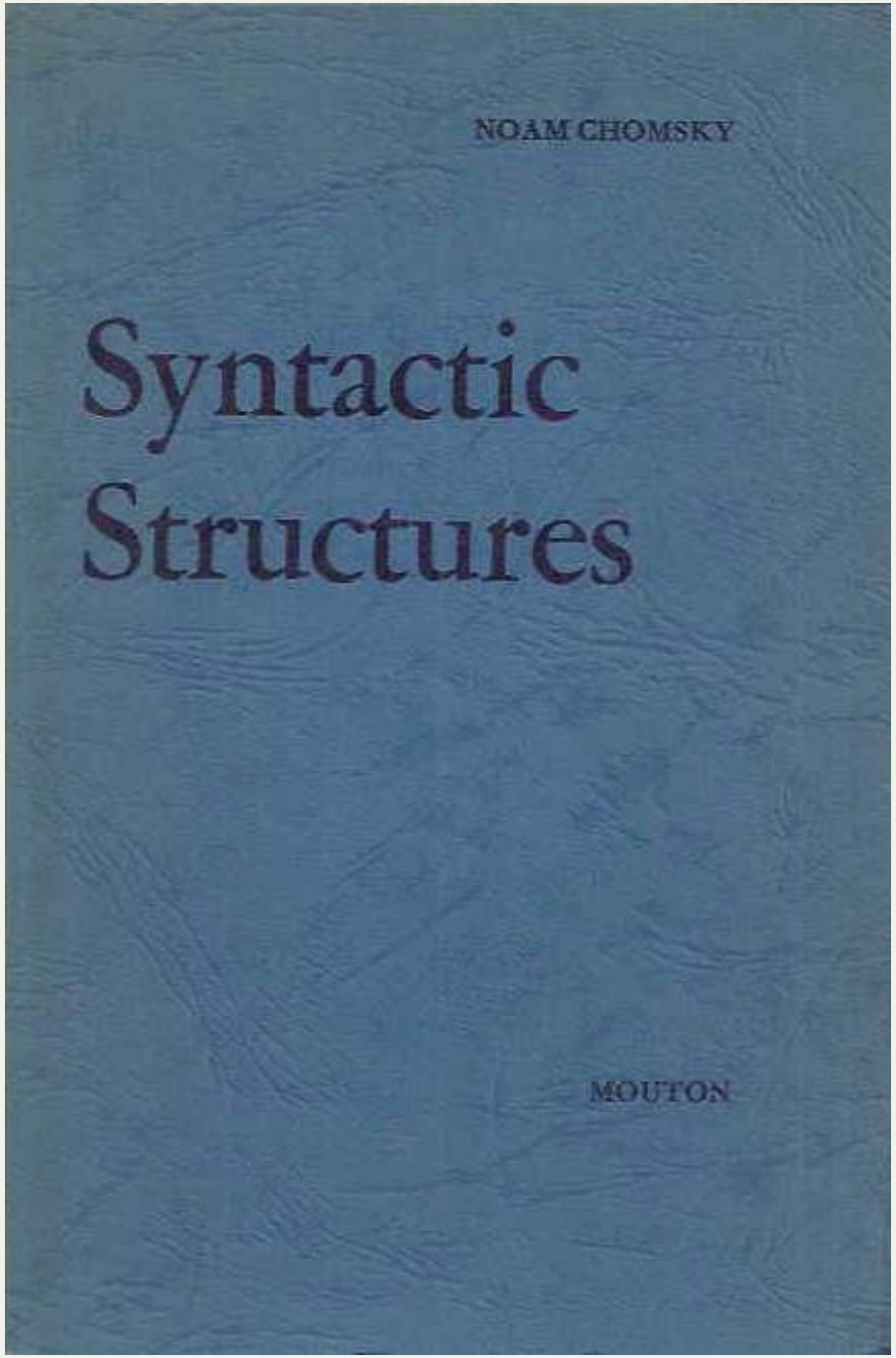
Figures

- 62 students
- 32 groups
- 6.4 lost points so far (out of 12.4)
- 20 students without commits
- 16 with bad commit messages
- 2 silently renamed their account
- 1 shadow committer (user.email not set)
- 0 force pushes so far

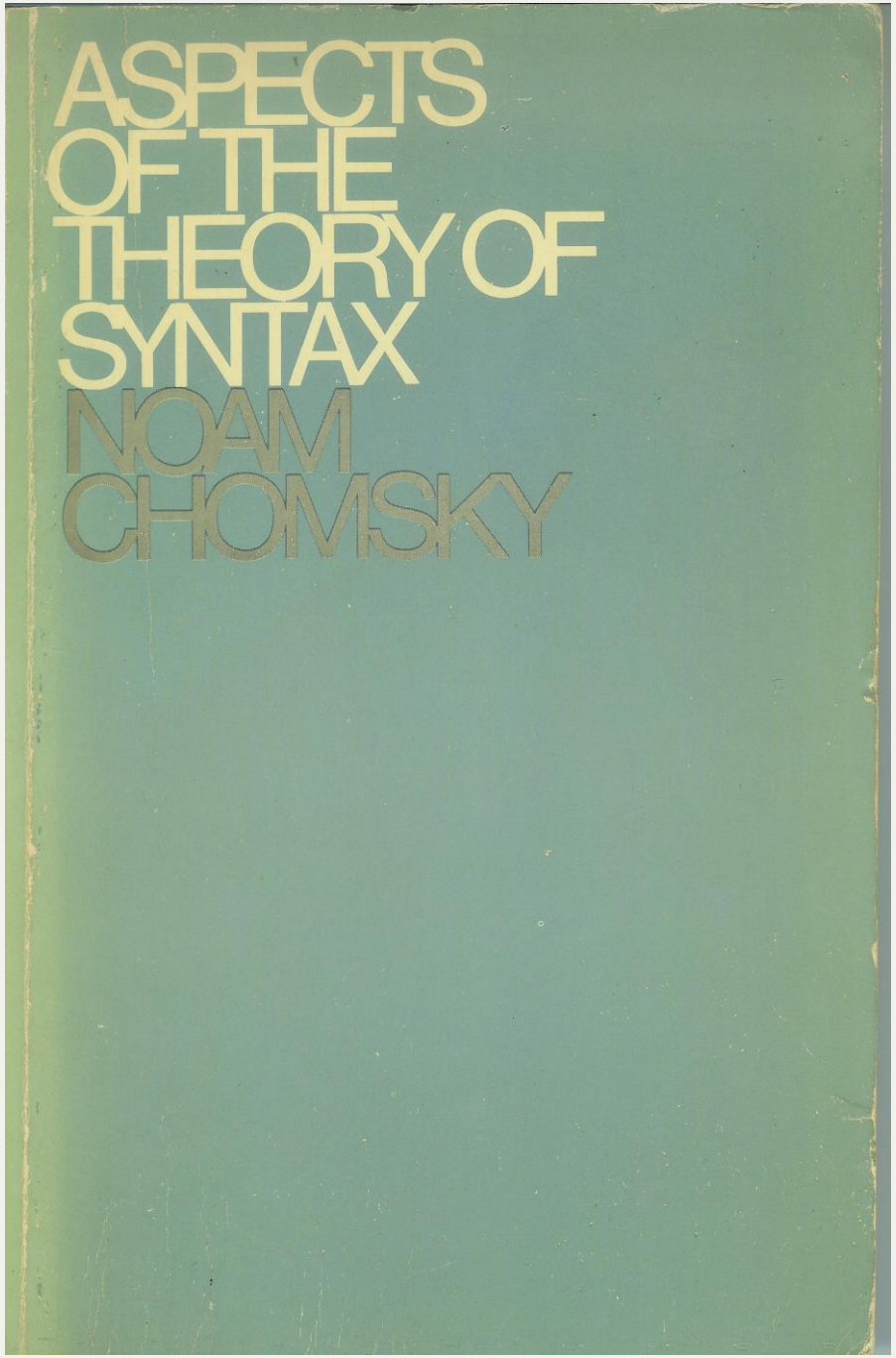
Grammars and Parsing

- among the most established areas of CS/SE

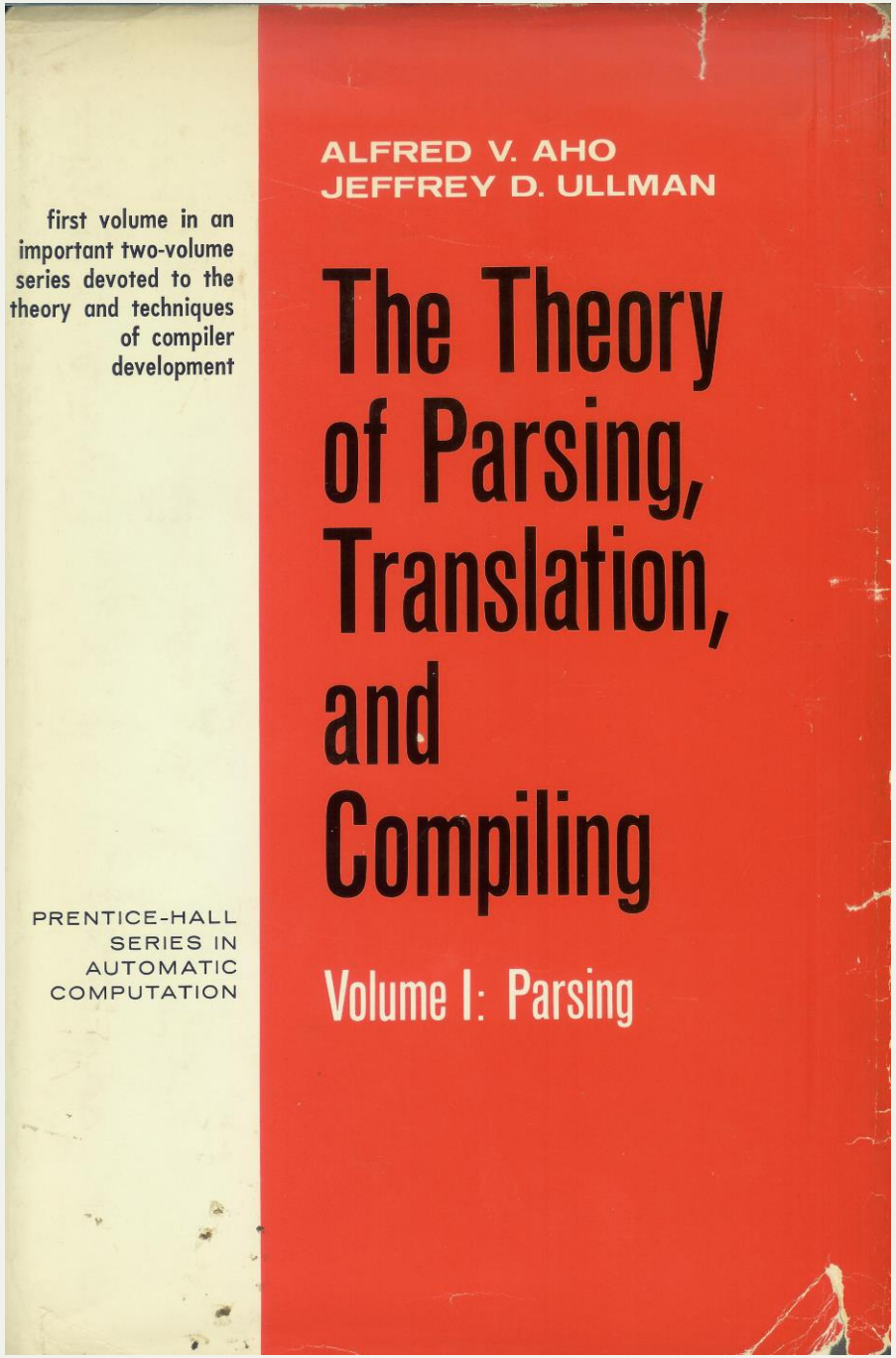




N. Chomsky, *Syntactic Structures*, 1957

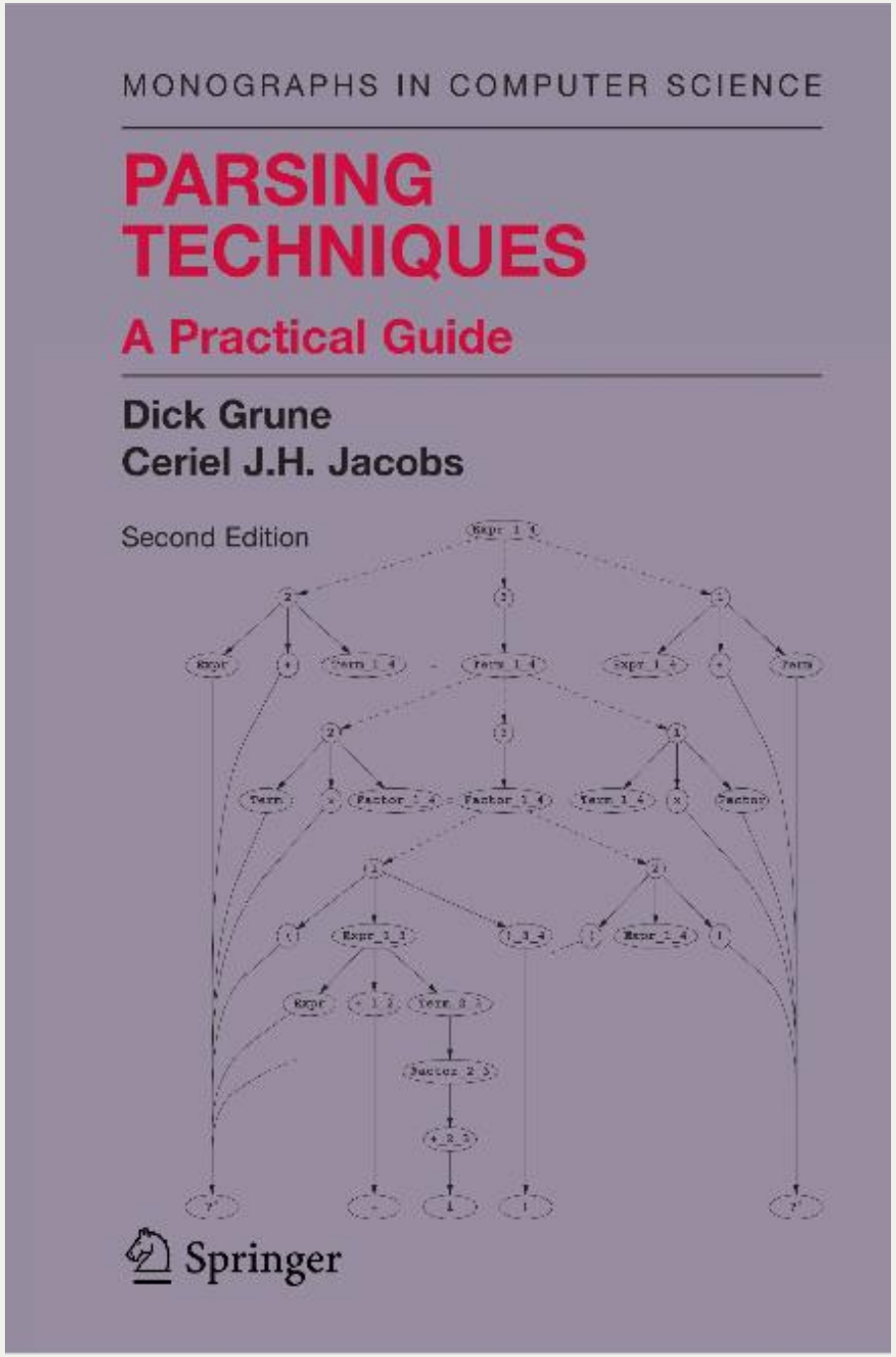


N. Chomsky, *Aspects of the Theory of Syntax*, 1965



A.V. Aho,
J.D. Ullman,
The Theory of Parsing, Translation and Compiling,
Volumes I + II,
1972

A. V. Aho,
R. Sethi,
J. D. Ullman,
Compilers: Principles, Techniques and Tools,
1986



D. Grune, C.J.H. Jacobs, *Parsing Techniques: A Practical Guide*, 2 ed, 2008

Why are grammars and parsing relevant?

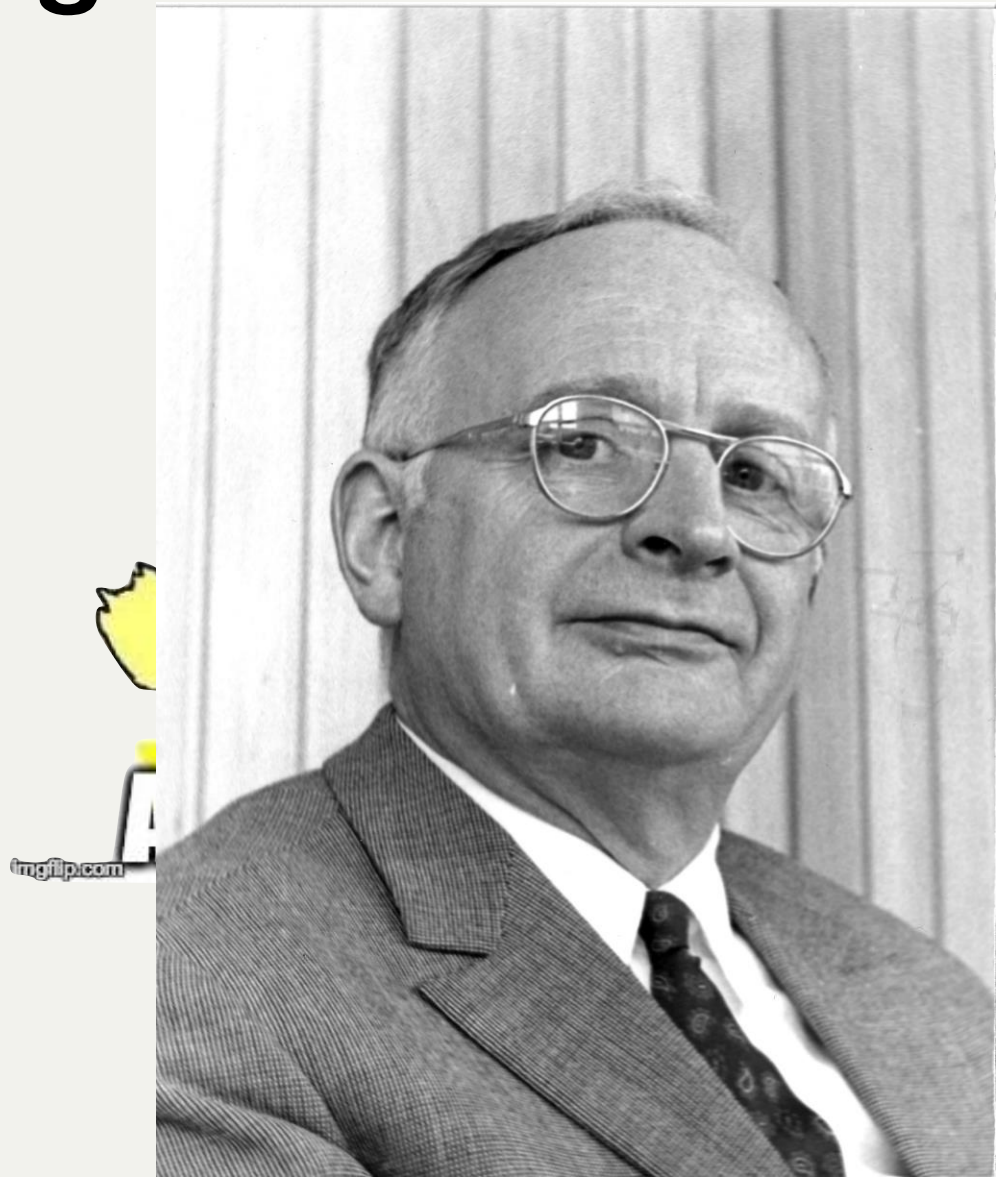
Language

- Programming languages: C, Java, C#, JavaScript
- Markup languages: HTML, XML, TeX, Markdown, wikis
- Domain-specific languages: BibTeX, CSS, SQL, QL
- Data formats: JSON, log files, protocol data, bytecode
- …
- (formally: a set of strings)

How to define a language?

Adriaan van Wijngaarden

- List all the sentences!
- Infinite languages?
- Finite recipes = grammars
- Infinite grammars?
- Two level grammars



Example

- Valid sentences/programs/instances:
  - Alice
  - Alice and Bob
  - Alice, Bob and Coen
  - Alice, Bob, Coen and Daenerys
  - ...
- How to define a recipe?

ABCD Grammar

Name → Alice

Name → Bob

Name → Coen

Name → Daenerys

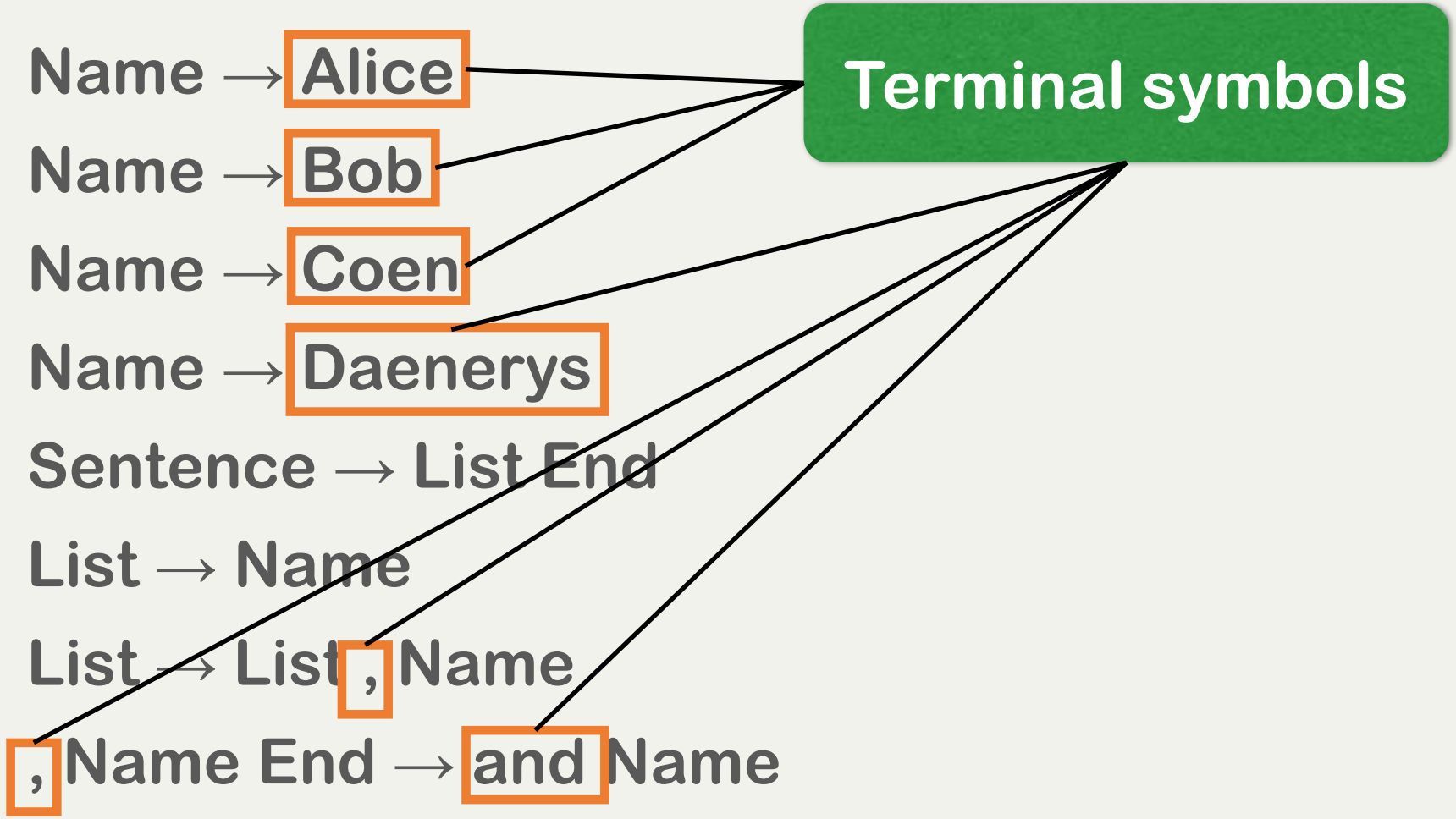
Sentence → List End

List → Name

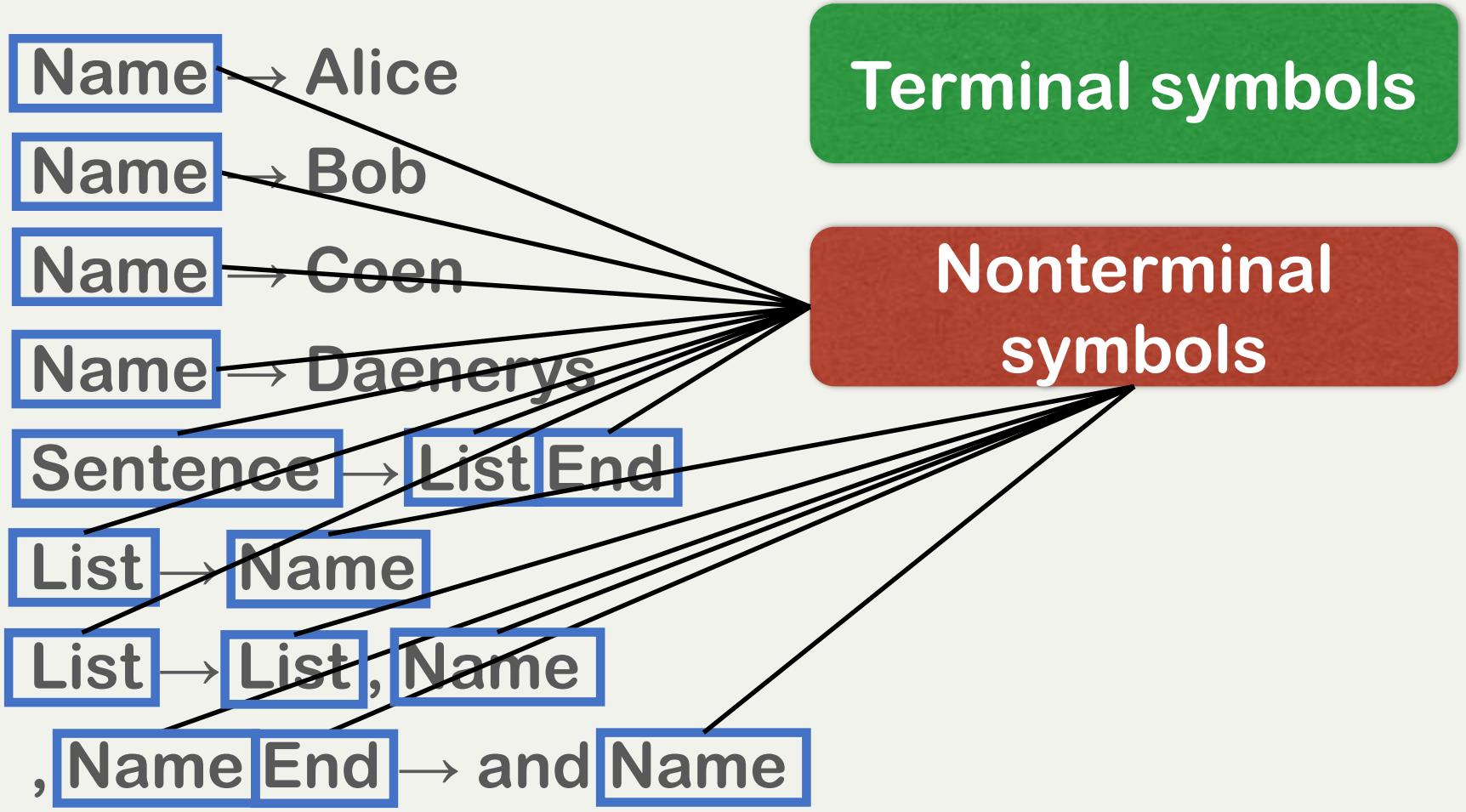
List → List , Name

, Name End → and Name

ABCD Grammar



ABCD Grammar



ABCD Grammar

Name → Alice

Name → Bob

Name → Coen

Name → Daenerys

Sentence → List End

List → Name

List → List , Name

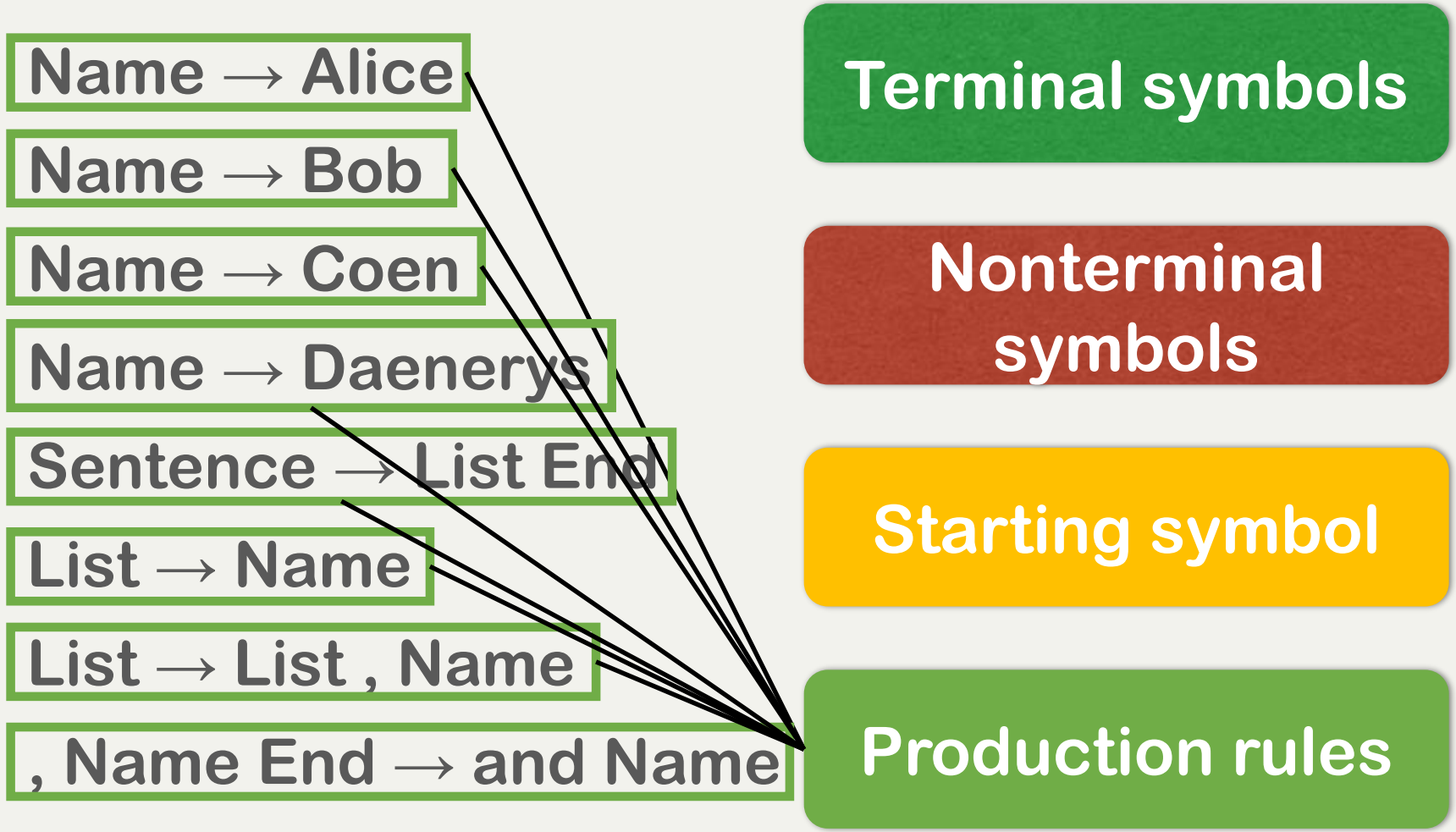
, Name End → and Name

Terminal symbols

Nonterminal symbols

Starting symbol

ABCD Grammar



Using ABCD

Name → Alice
Name → Bob
Name → Coen
Name → Daenerys
Sentence → List End
List → Name
List → List , Name
, Name End → and Name

Production

- Alice and Bob
- Sentence
  → List End
  → List , Name End
  → Name , Name End
  → Name and Name
  → Alice and Name
  → Alice and Bob
- (generative semantics)

- Alice and Bob
- Alice and Bob
  → Name and Bob
  → Name and Name
  → Name , Name End
  → List , Name End
  → List End
  → Sentence
- (analytic semantics)

raincode LABS
compiler experts

Notations

- Name → Alice | Bob | Coen | Daenerys
- Name → "Alice" | "Bob" | "Coen" | "Daenerys"
- Name → “Alice” | “Bob” | “Coen” | “Daenerys”
- ⟨Name⟩ → Alice | Bob | Coen | Daenerys
- ⟨Name⟩ → “Alice” | “Bob” | “Coen” | “Daenerys”

Notations

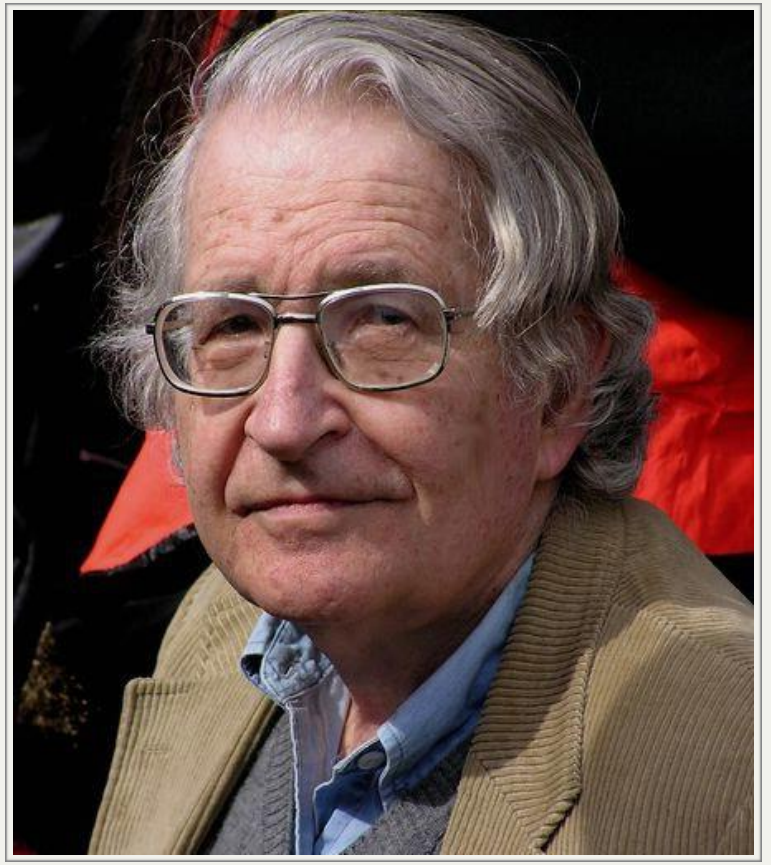
- List → List , Name
- ⟨List⟩ ::= ⟨List⟩ "," ⟨Name⟩ ;
- List "," Name -> List
- List <- List ',' Name
- define List [List] , [Name] end define
- syntax List = List "," Name;
- List -> List ',' Name : ['$1'|'$3'].

Common metaconstructs

- Optional symbols
  - A?, [A]
- Zero or more (Kleene star)
  - A*, {A}
- One or more (Kleene cross)
  - A+
- Choice (disjunction)
  - A | B, A / B
- Less common (careful!)
  - conjunction
  - negation
  - exact repetition
  - reference naming
  - priorities

Chomsky-Schützenberger hierarchy

- Type-0: **Recursively enumerable**
  - Rules: $\alpha \rightarrow \beta$ (unrestricted)
- Type-1: **Context-sensitive**
  - Rules: $\alpha A \beta \rightarrow \alpha \gamma \beta$
- Type-2: **Context-free**
  - Rules: $A \rightarrow \gamma$
- Type-3: **Regular**
  - Rules: $A \rightarrow a$ and $A \rightarrow aB$



Noam Chomsky. *On Certain Formal Properties of Grammars*, Information & Control 2(2):137–167, 1959.

CFG for ABCD

⟨Name⟩ → “Alice” | “Bob” | “Coen” | “Daenerys”

⟨Sentence⟩ → ⟨Name⟩ | ⟨List⟩ “and” ⟨Name⟩

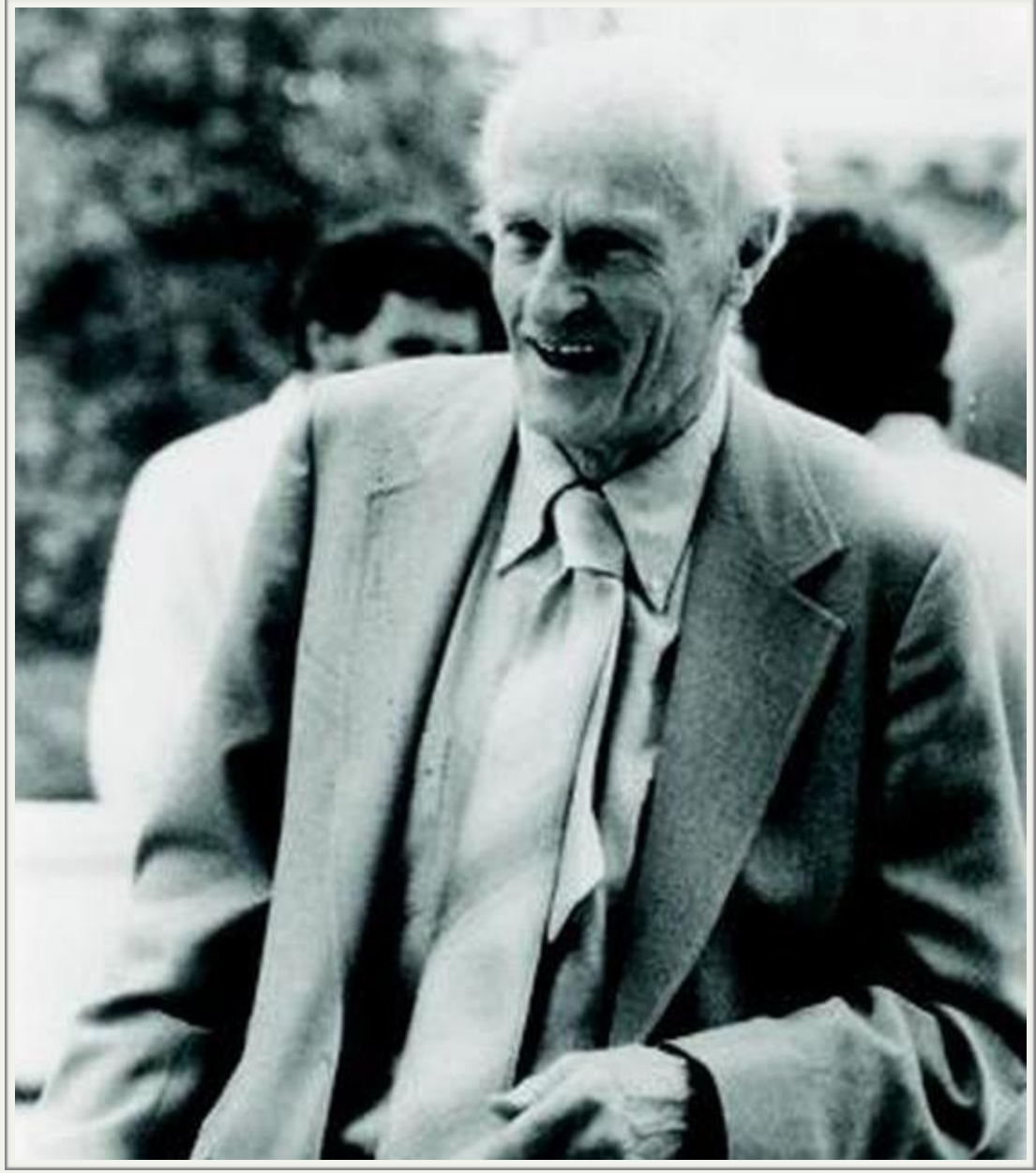
⟨List⟩ → ⟨Name⟩ “,” ⟨List⟩ | ⟨Name⟩

⟨List⟩ → ⟨Name⟩ (“,” ⟨Name⟩)*

⟨List⟩ → {⟨Name⟩ “,”}+

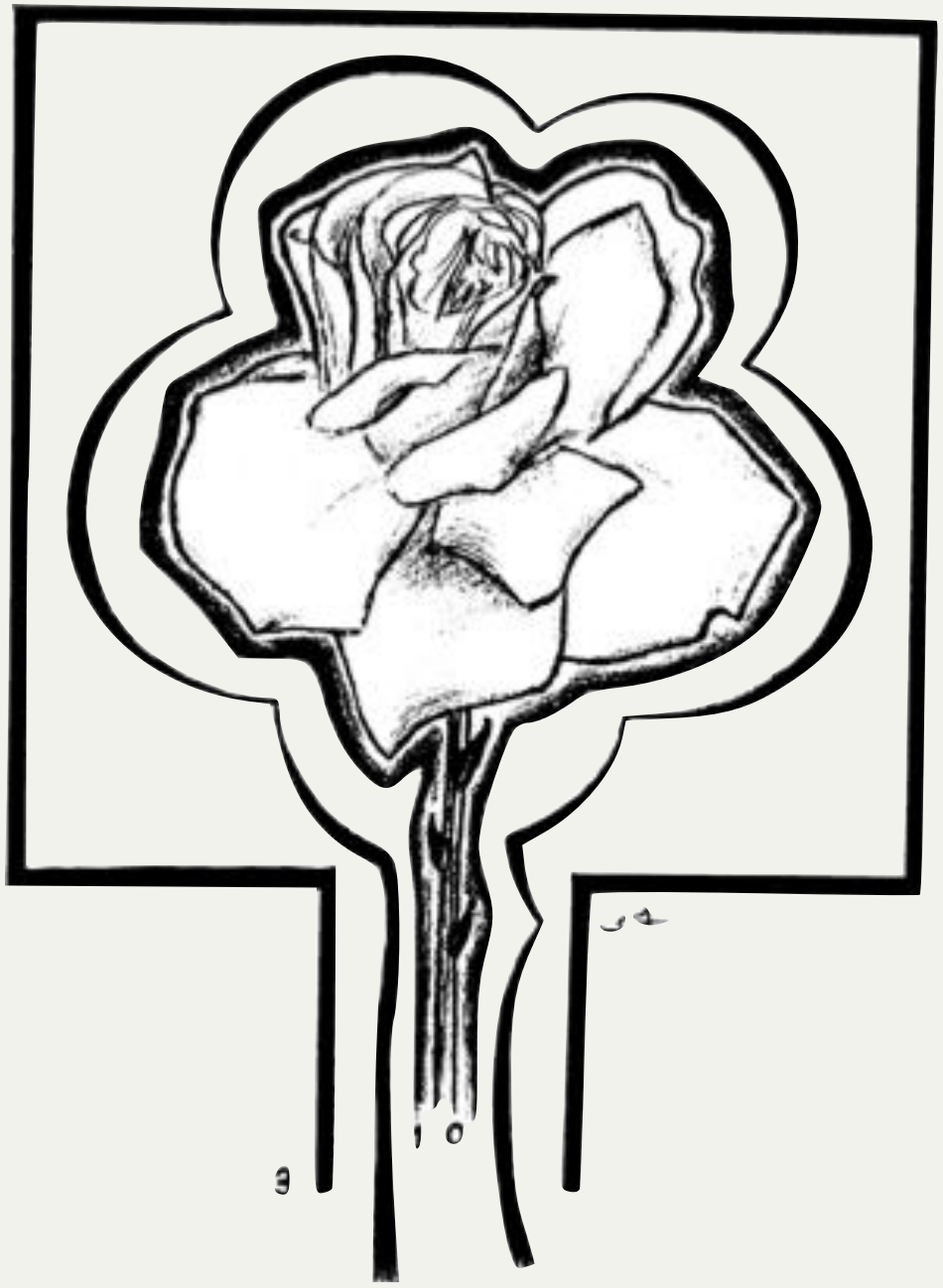
Regexp for ABCD

^\w+((, \w+)* and \w+)?$



S. C. Kleene, *Representation of Events in Nerve Nets and Finite Automata*.
In *Automata Studies*, pp. 3–42, 1956.
photo from: Konrad Jacobs, S. C. Kleene, 1978, MFO.

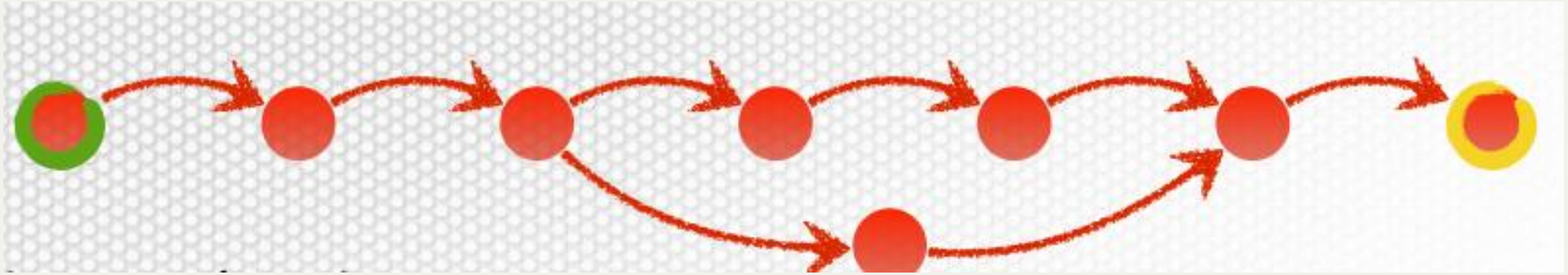
raincode LABS
compiler experts



Rose by Arwen Grune; p.58 of Grune/Jacobs' "Parsing Techniques", 2008

Finite world

- Explicitly given lists
- Acyclic automata

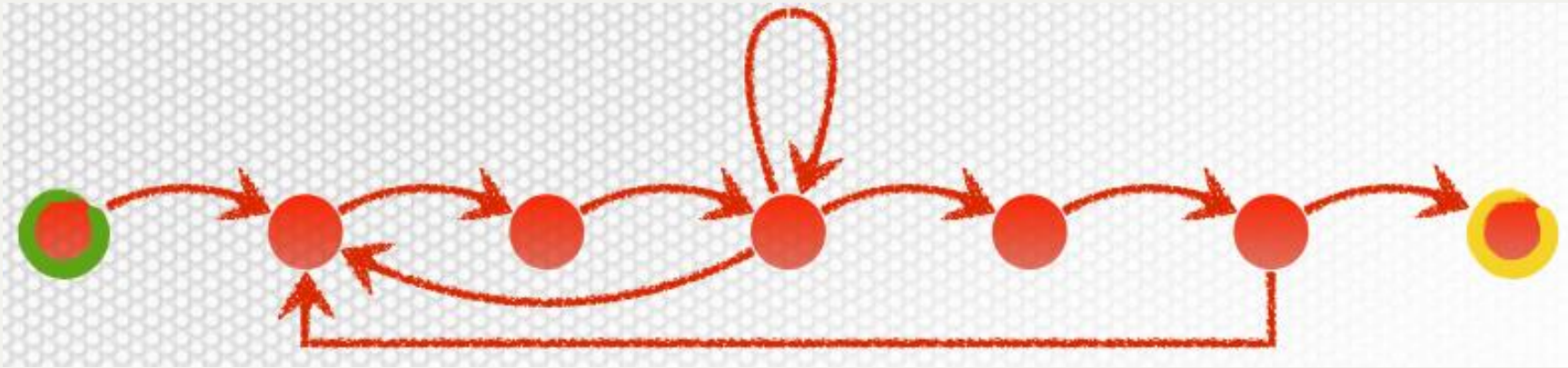


- Finite choice grammars (non-recursive, non-iterating)

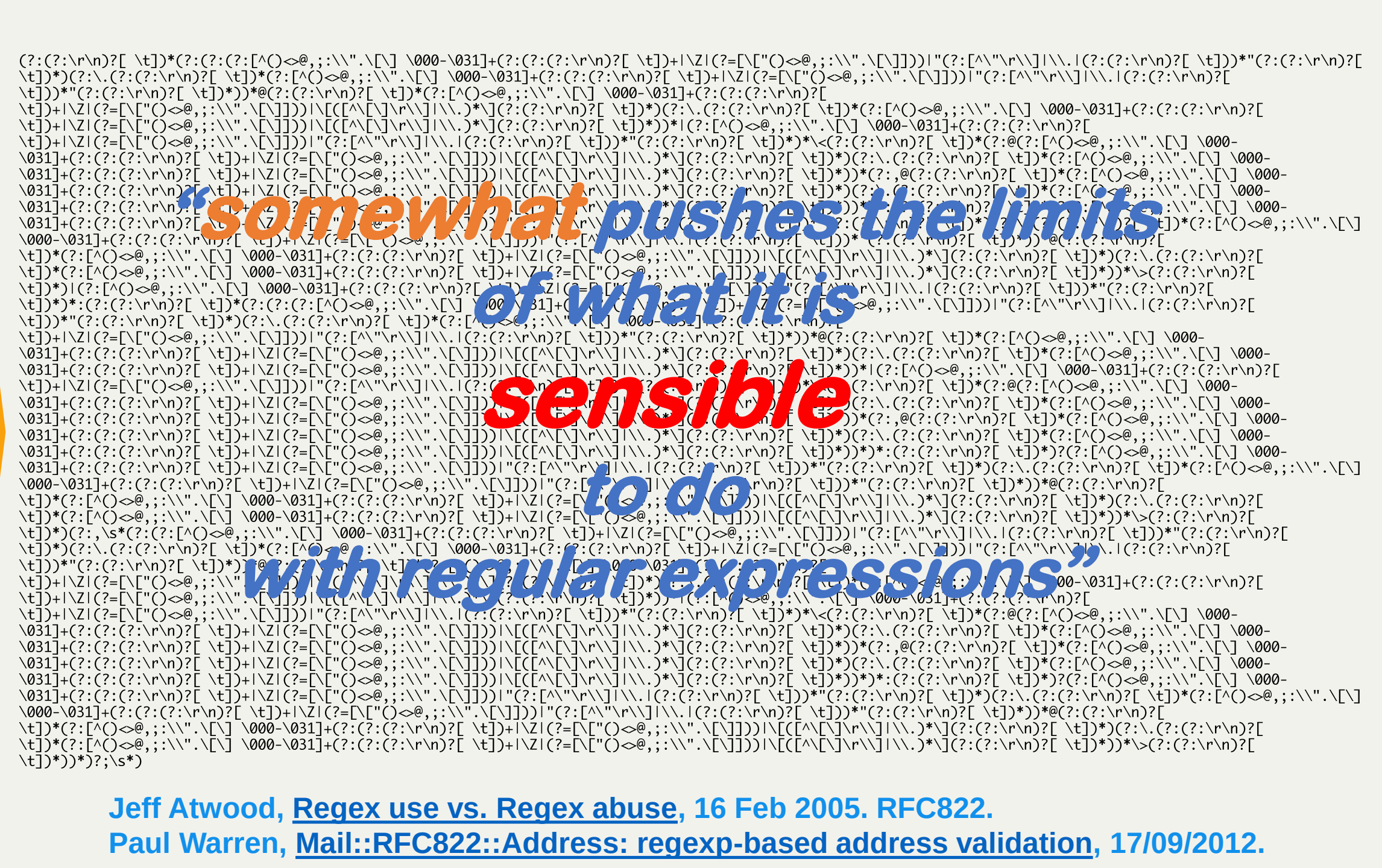
- i.e., users, keywords, postcodes

Regular world

- Regular expressions
- Finite automata
- Grammars:
  - A → a
  - A → aB



- i.e., substring search, substring replace, counting

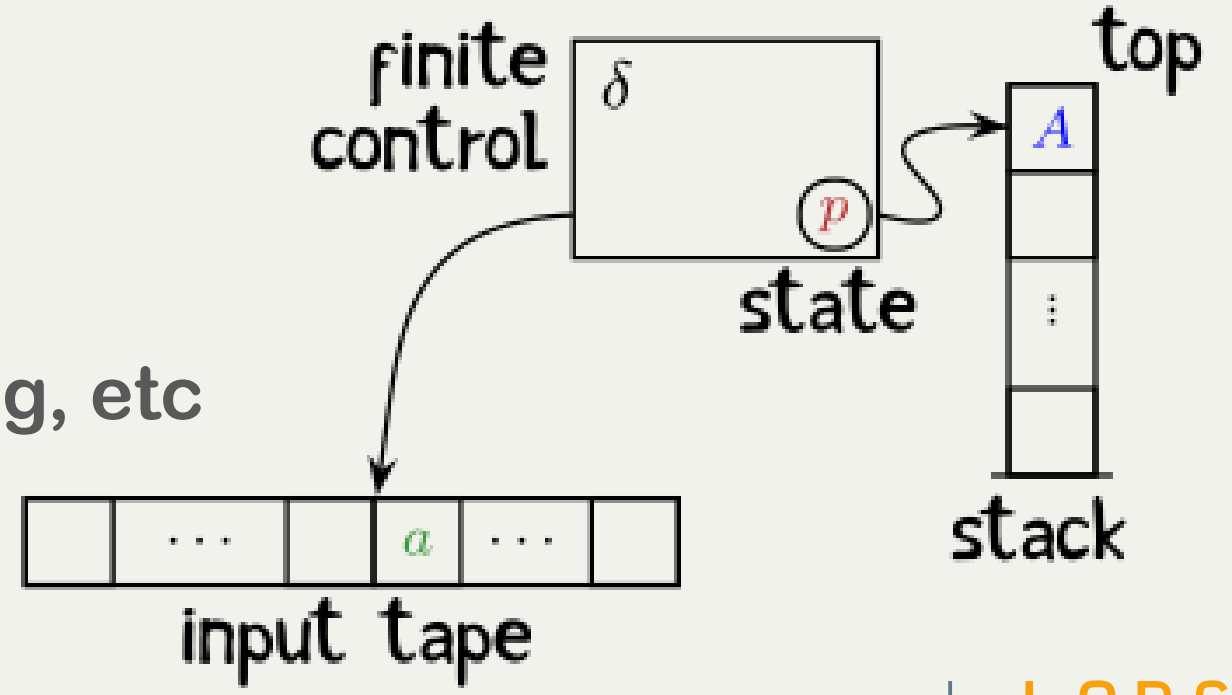


Jeff Atwood, Regex use vs. Regex abuse, 16 Feb 2005. RFC822.

Paul Warren, Mail::RFC822::Address: regexp-based address validation, 17/09/2012.

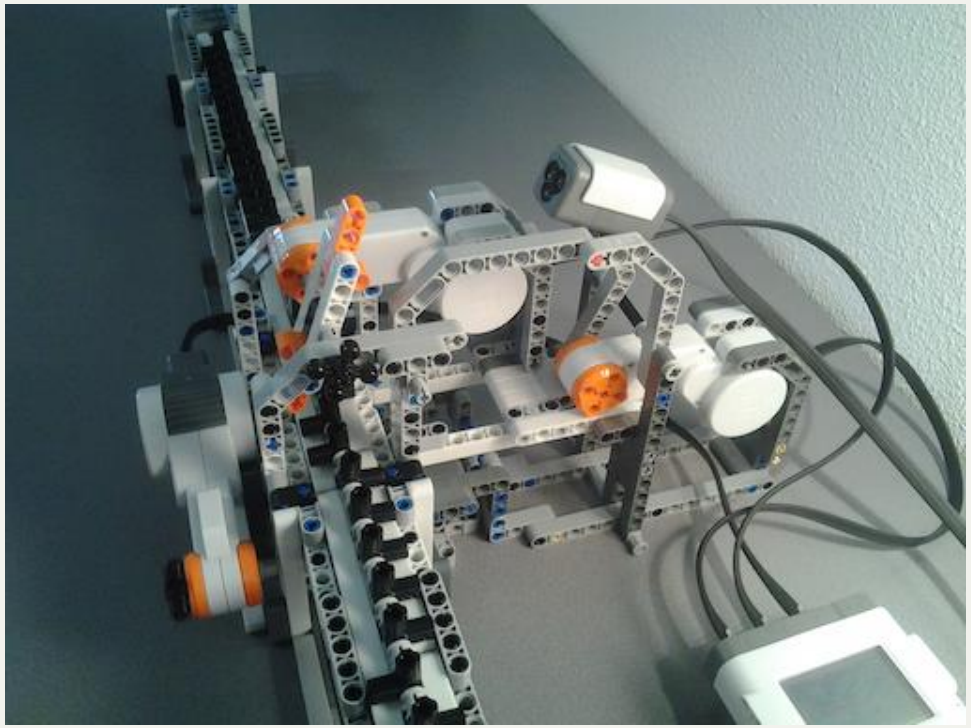
Context-free world

- Grammarware and software languages
- Nondeterministic pushdown automata
- Grammars:
  - $A \rightarrow \gamma$
    - $A \rightarrow BC$
    - $A \rightarrow a$
    - $A \rightarrow \varepsilon$
- i.e., parsing, pretty-printing, etc



Context-sensitive world

- Computer
- Linear-bounded automata
- Grammars:
  - $\alpha A \beta \rightarrow \alpha \gamma \beta$
- i.e., anything practical



~~http://www.legoturingmachine.org~~ https://vimeo.com/44202270

Unrestricted world

- Imaginary machines
- Turing machine, λ-calculus, semi-Thue rewriting systems, Lindenmayer systems, Markov algorithm, …

parsing is impossible

- Grammars:
  - $\alpha \rightarrow \beta$
- i.e., almost anything

recognising is impossible

In practice…

- Regular grammars are used for lexical analysis
  - keywords
  - constants
  - comments (if not nested)
- Context-free grammars: for structured/nested constructs
  - class declaration
  - if statement
  - …
- Everything else: annotations + hacking

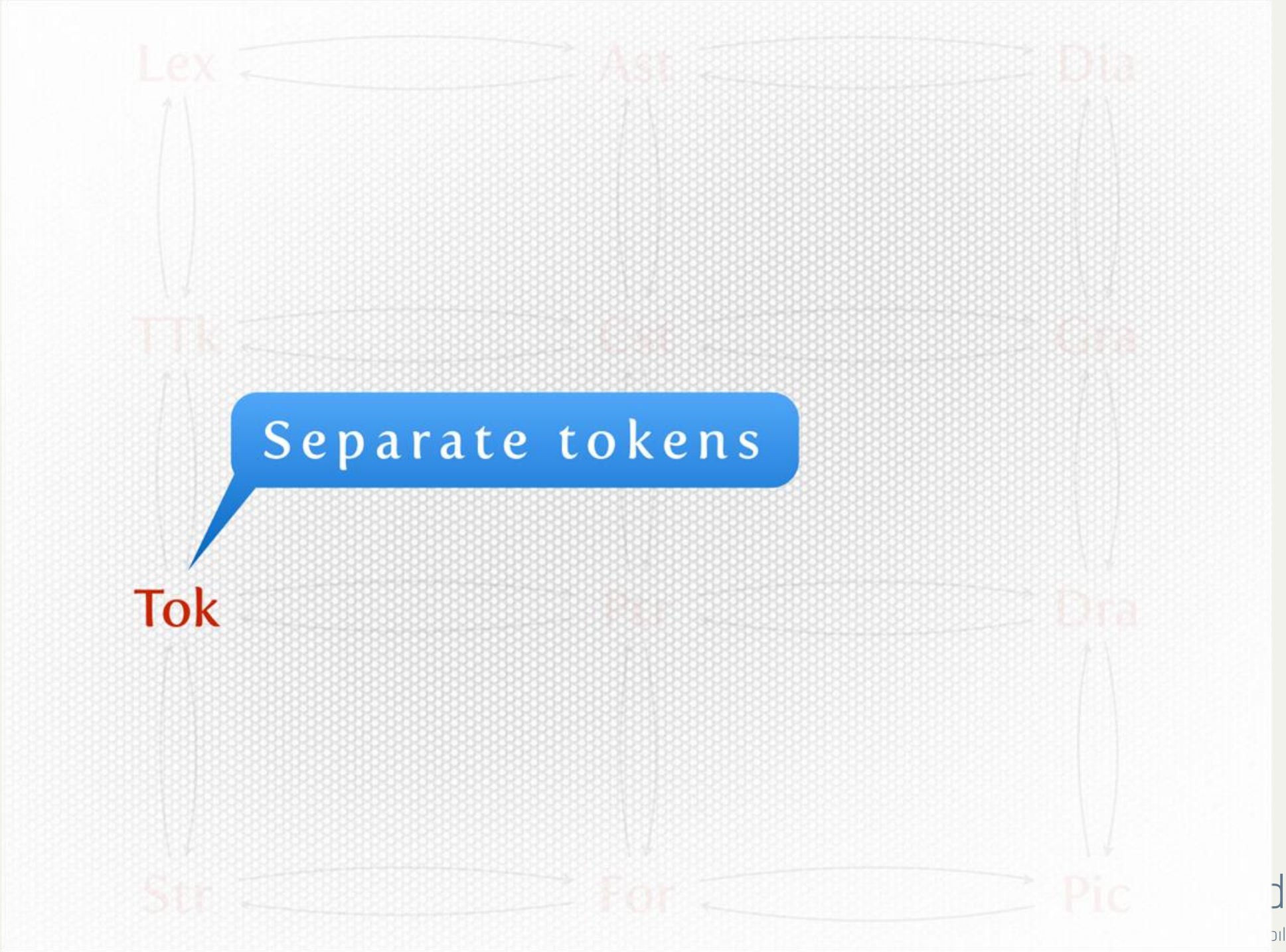
Time for a break

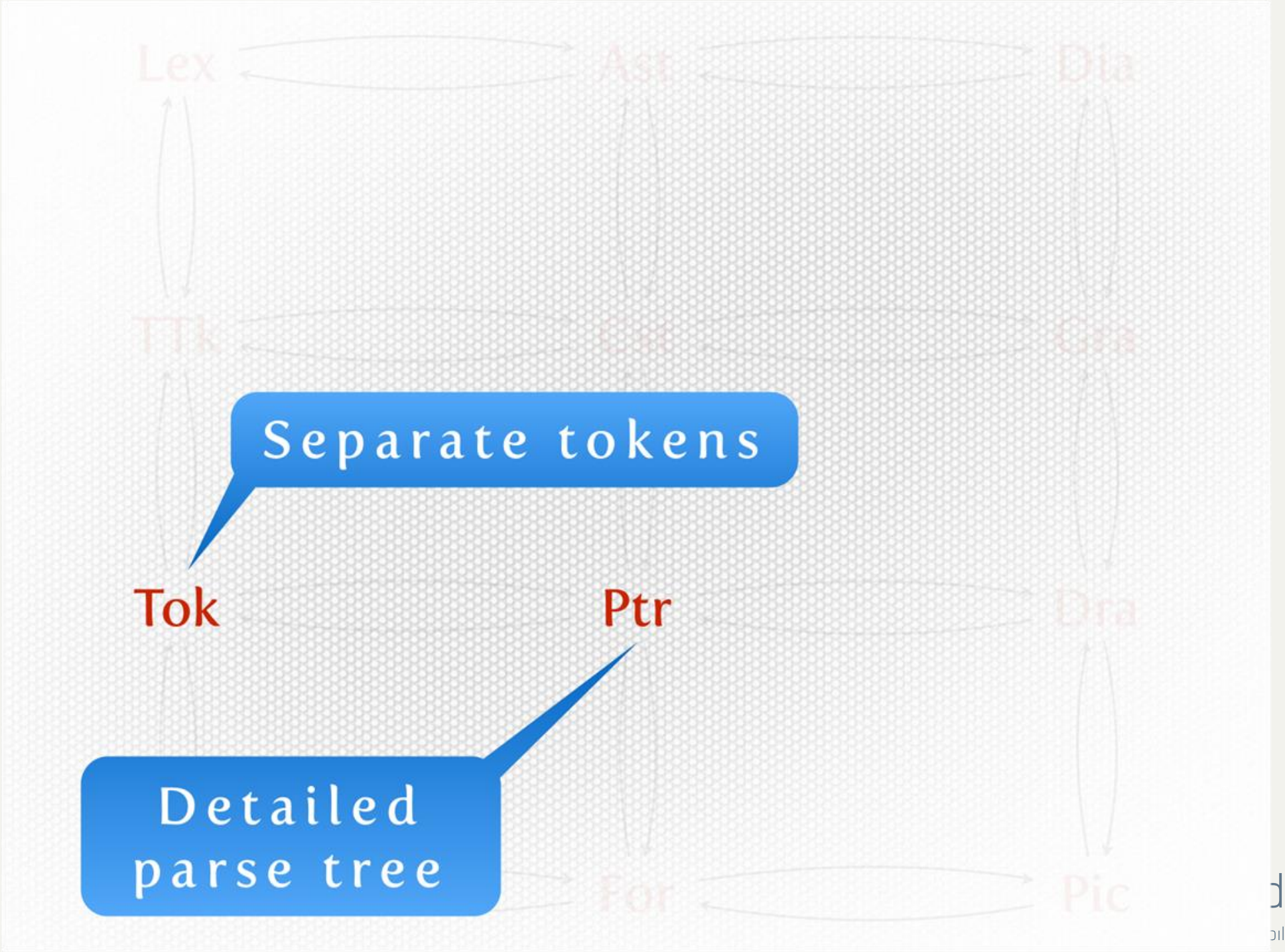
raincode LABS
compiler experts

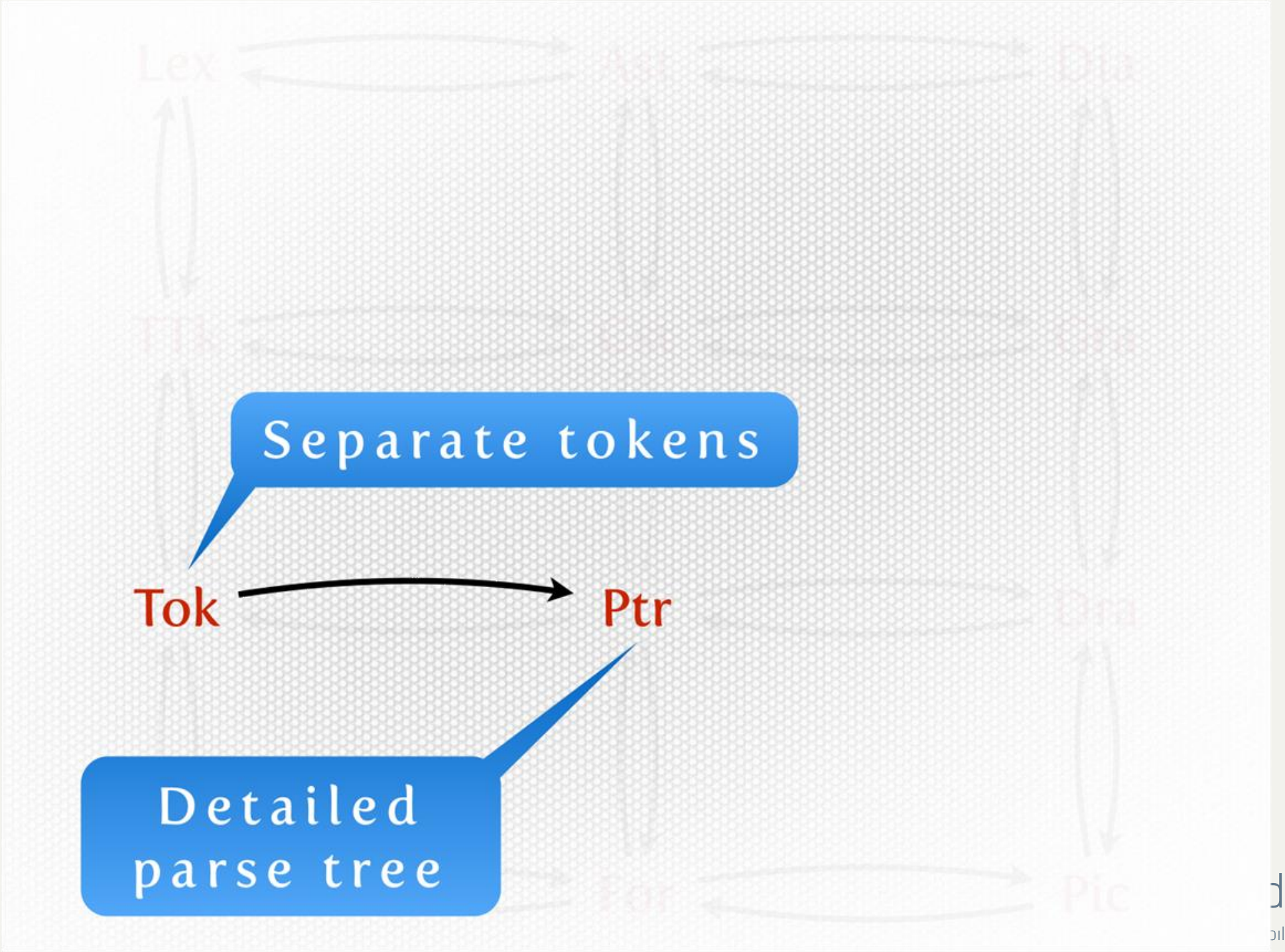
Parsing *in a broad sense*

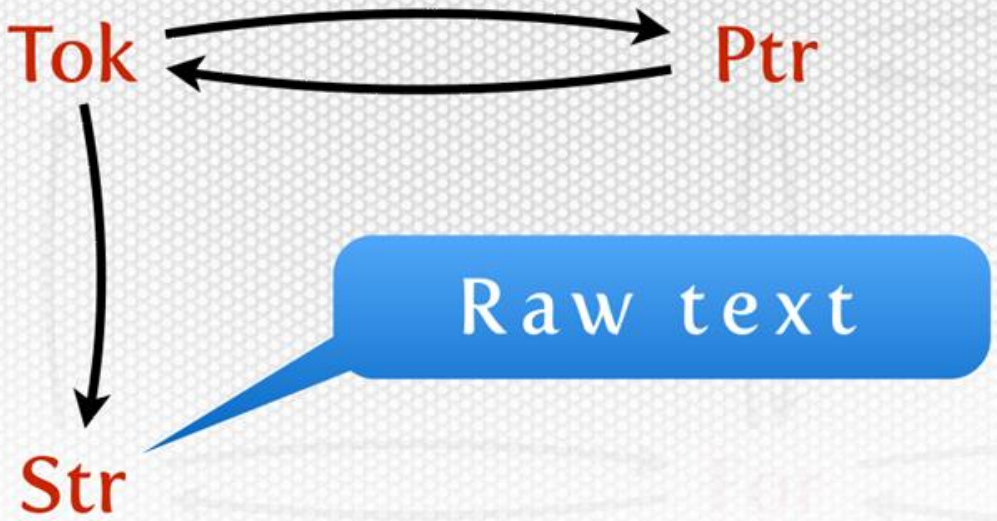
- Parsing in recognising structure
  - text → tree
  - tree → graph
  - tokens → hierarchy
  - forest disambiguation
  - image recognition
- Unparsing is representing structure

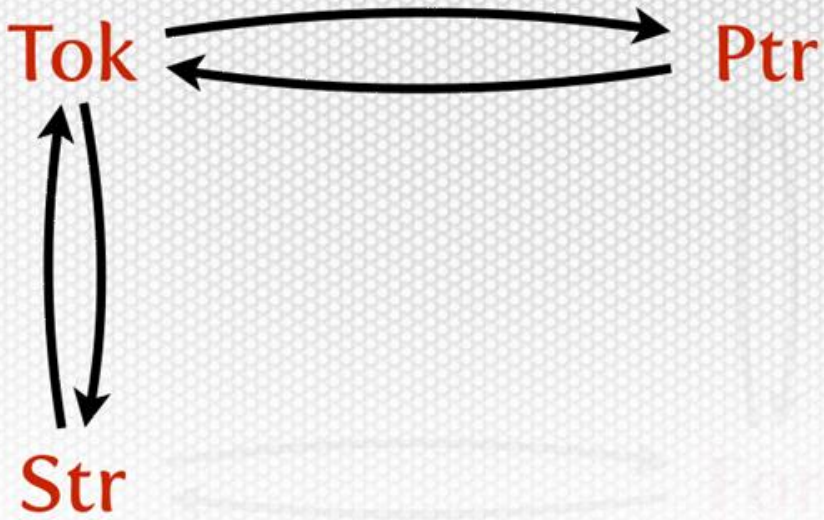
V. Zaytsev, A. H. Bagge, *Parsing in a Broad Sense*, MoDELS 2014.

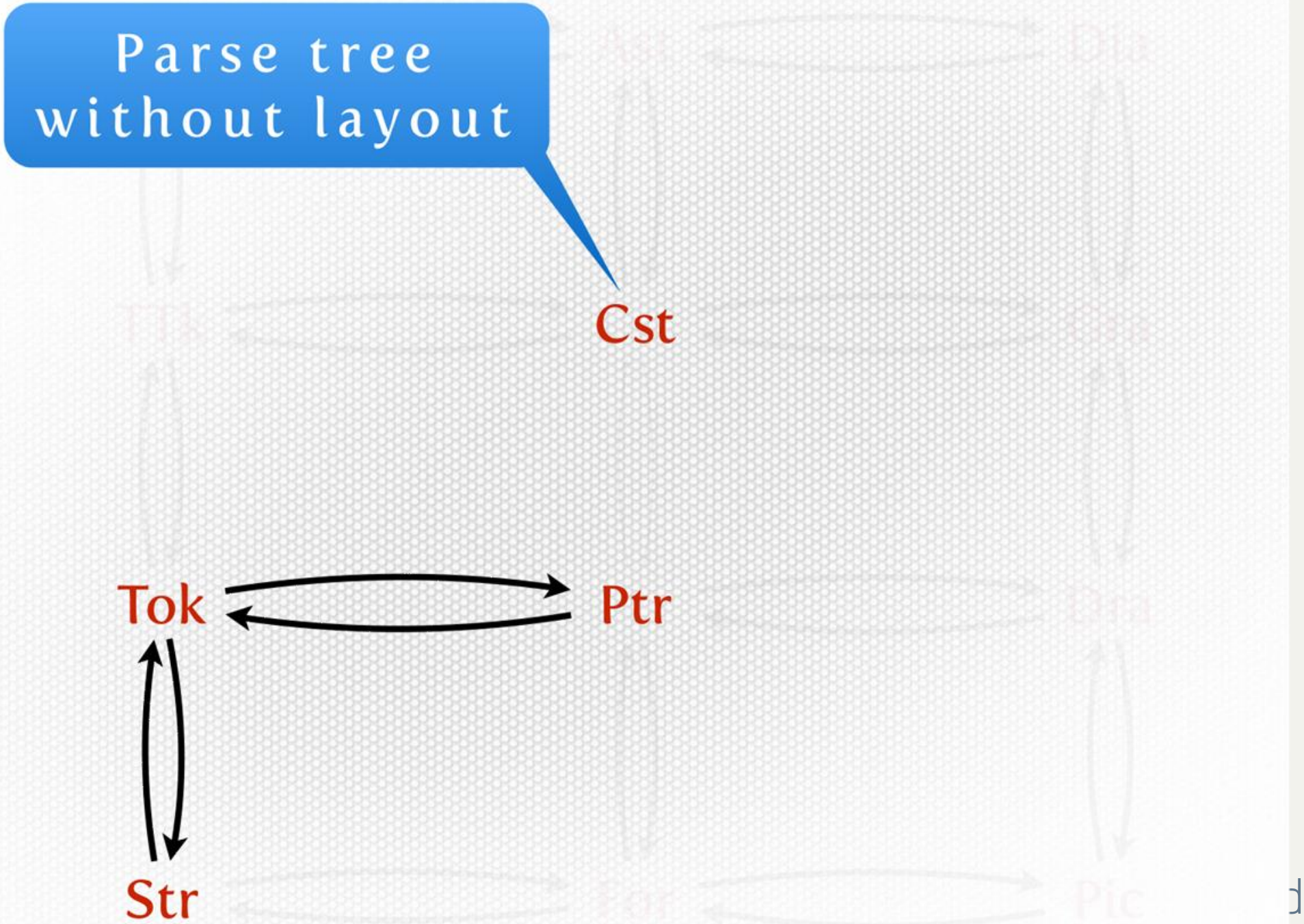


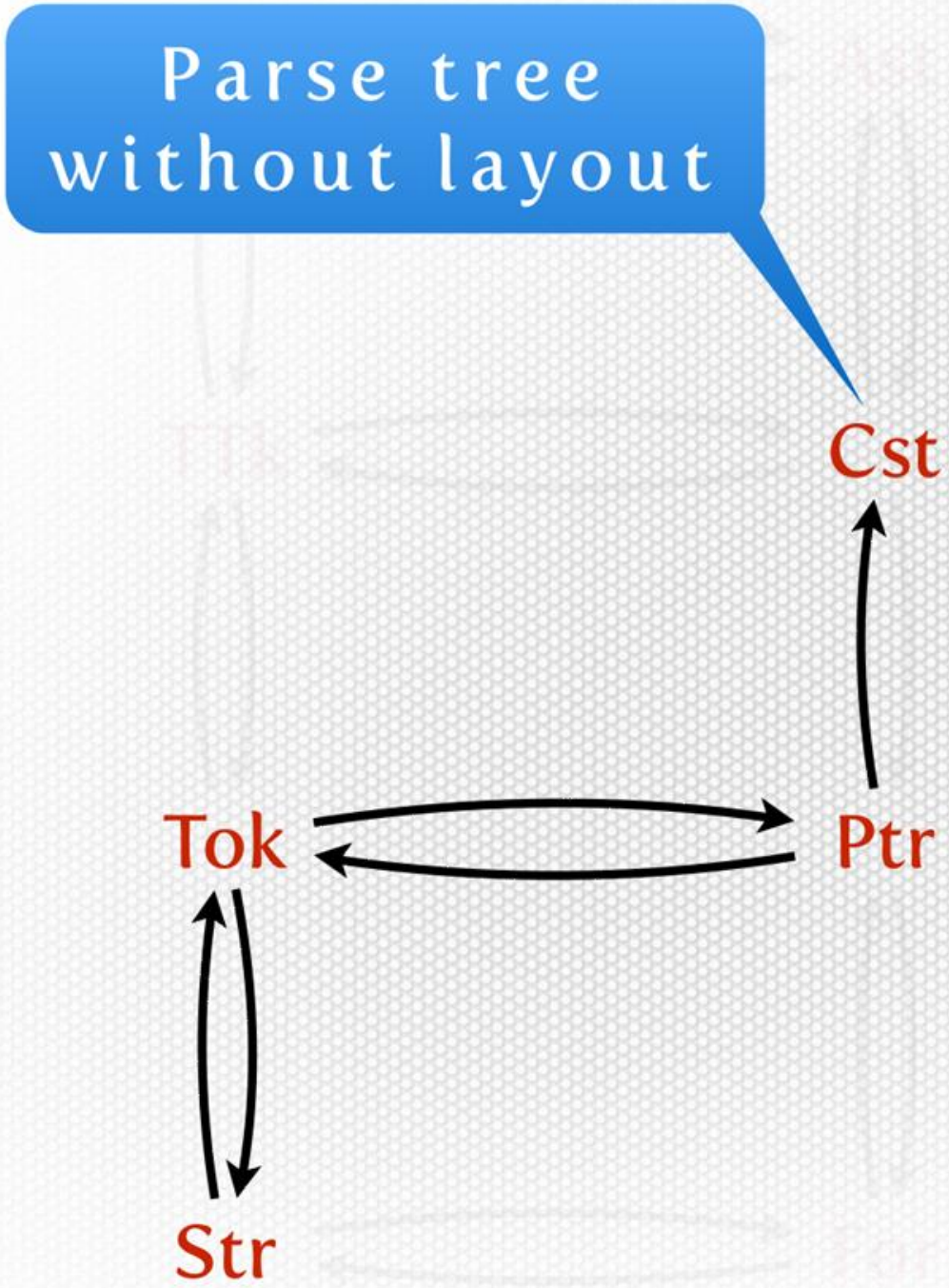


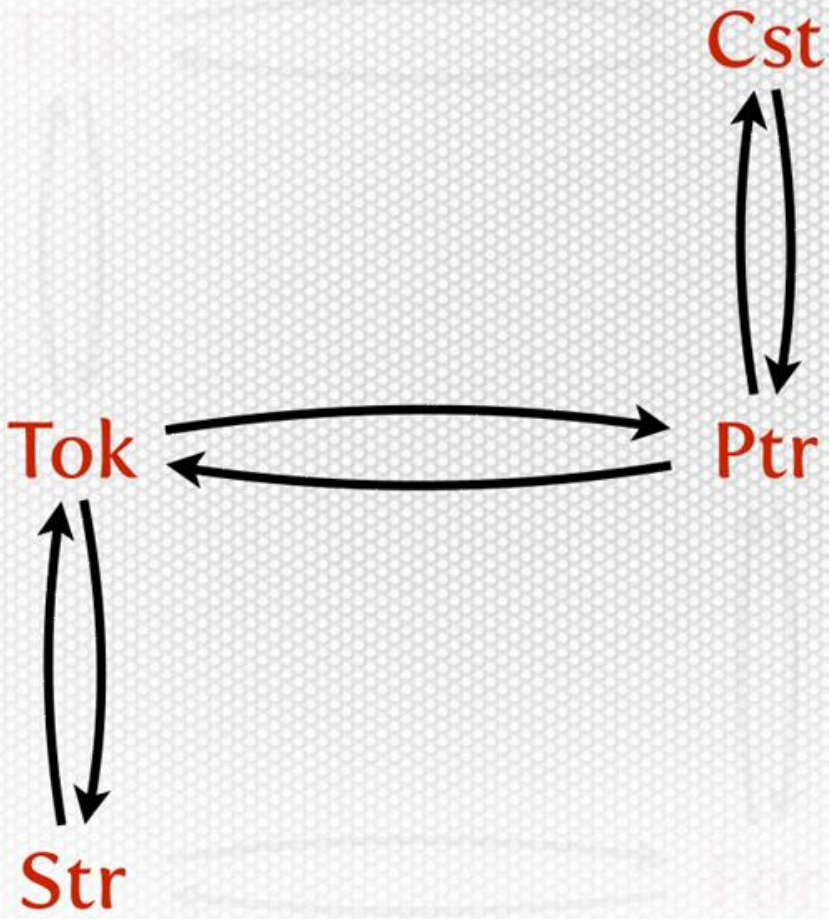


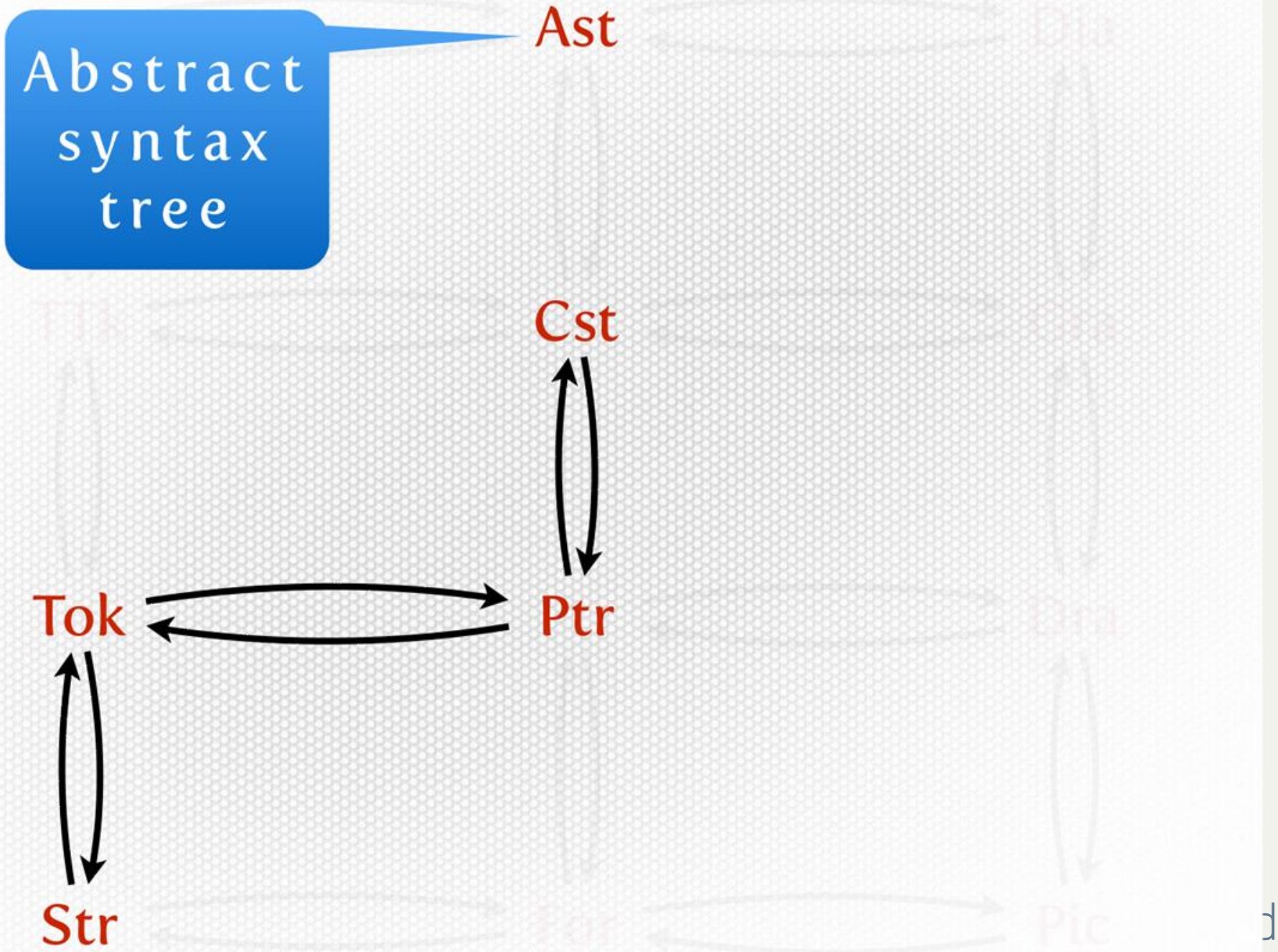


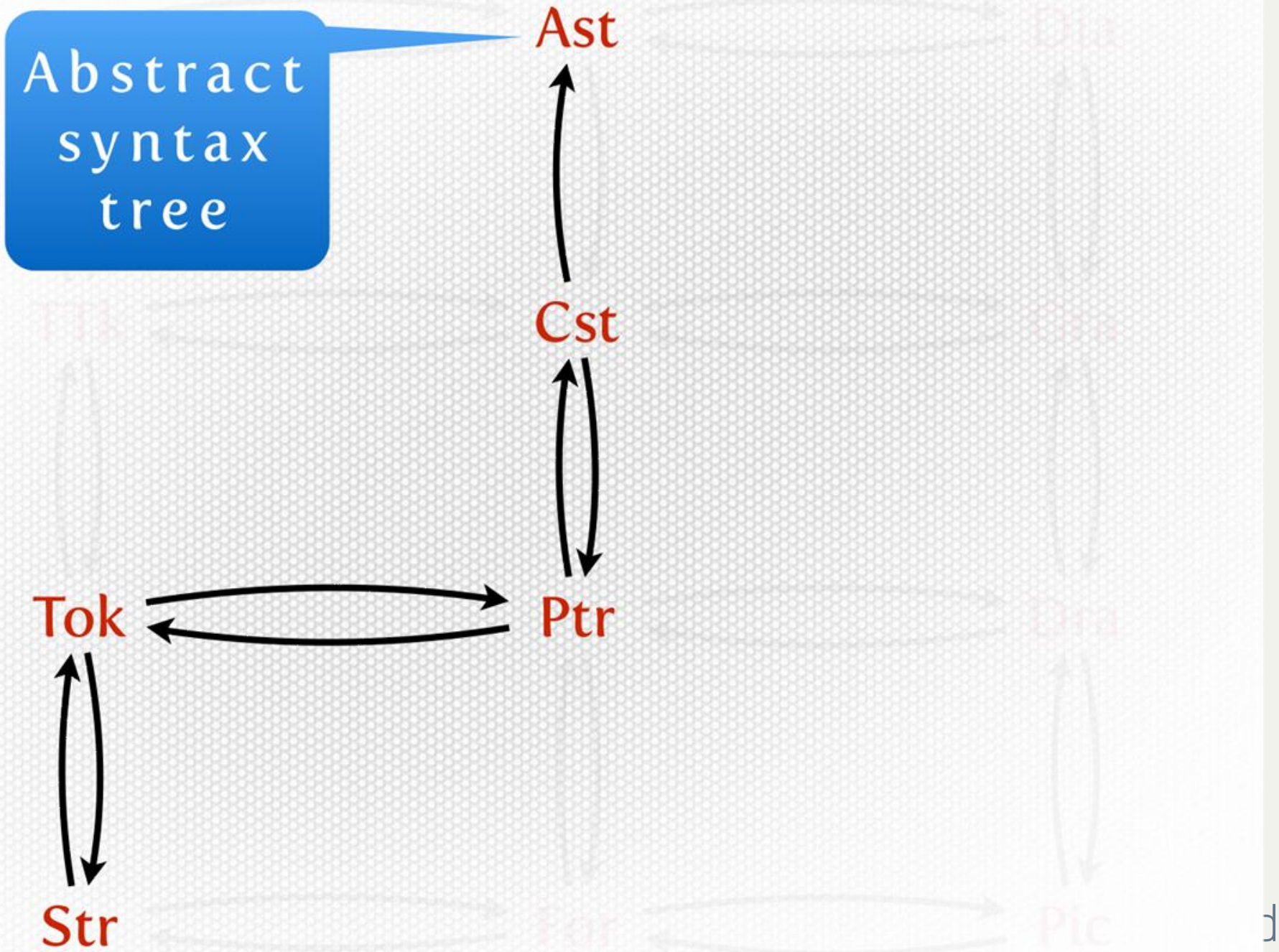


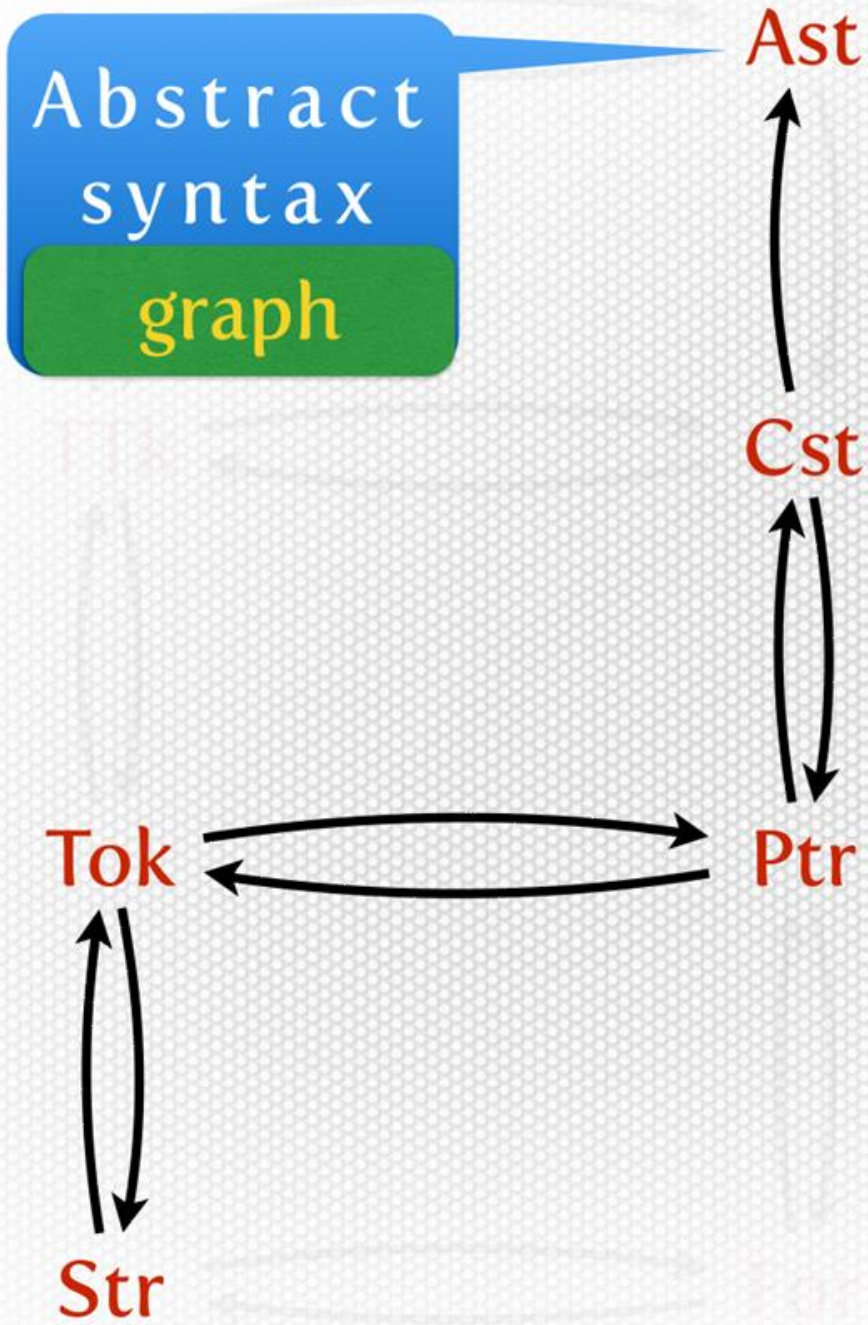


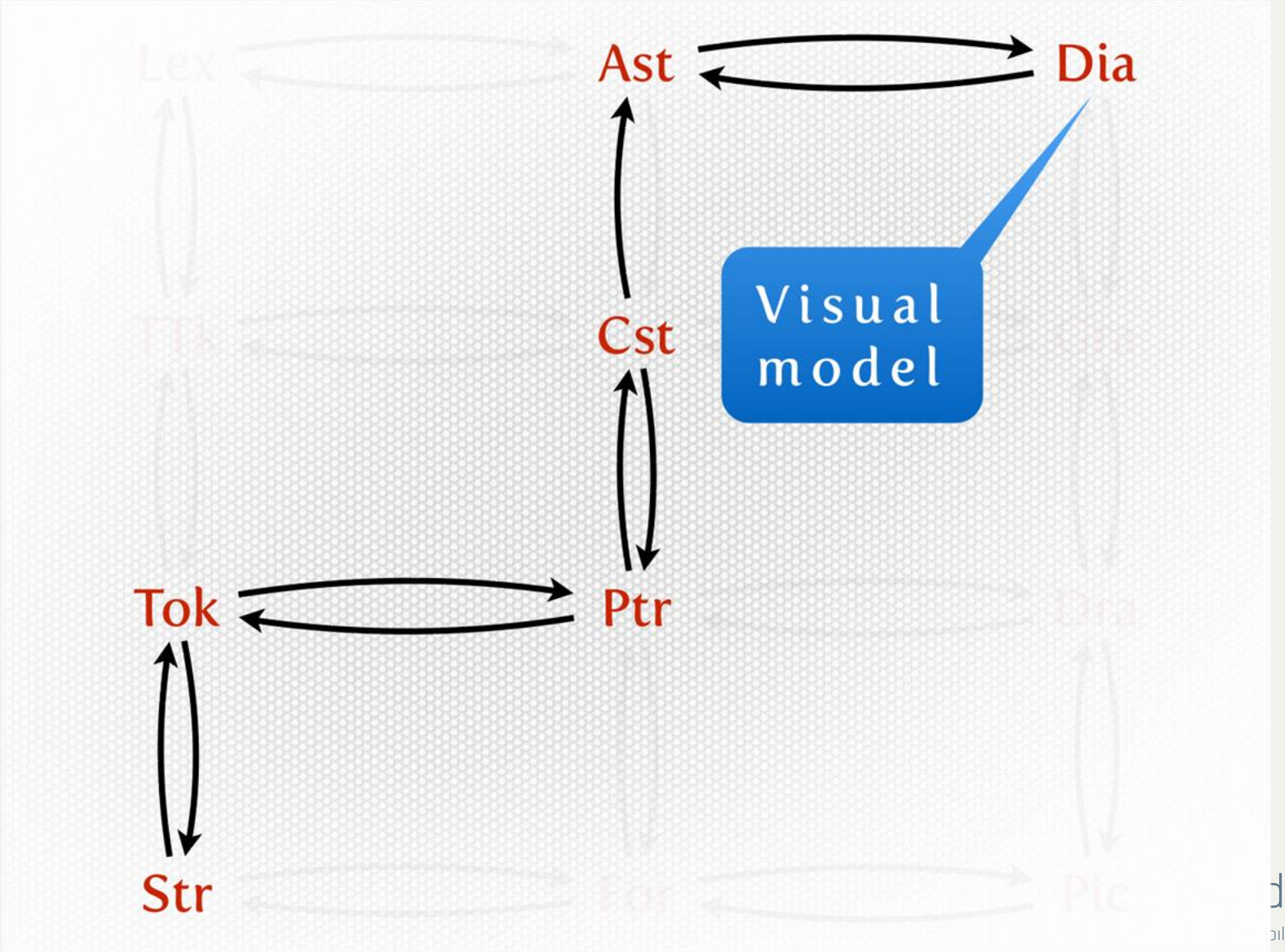


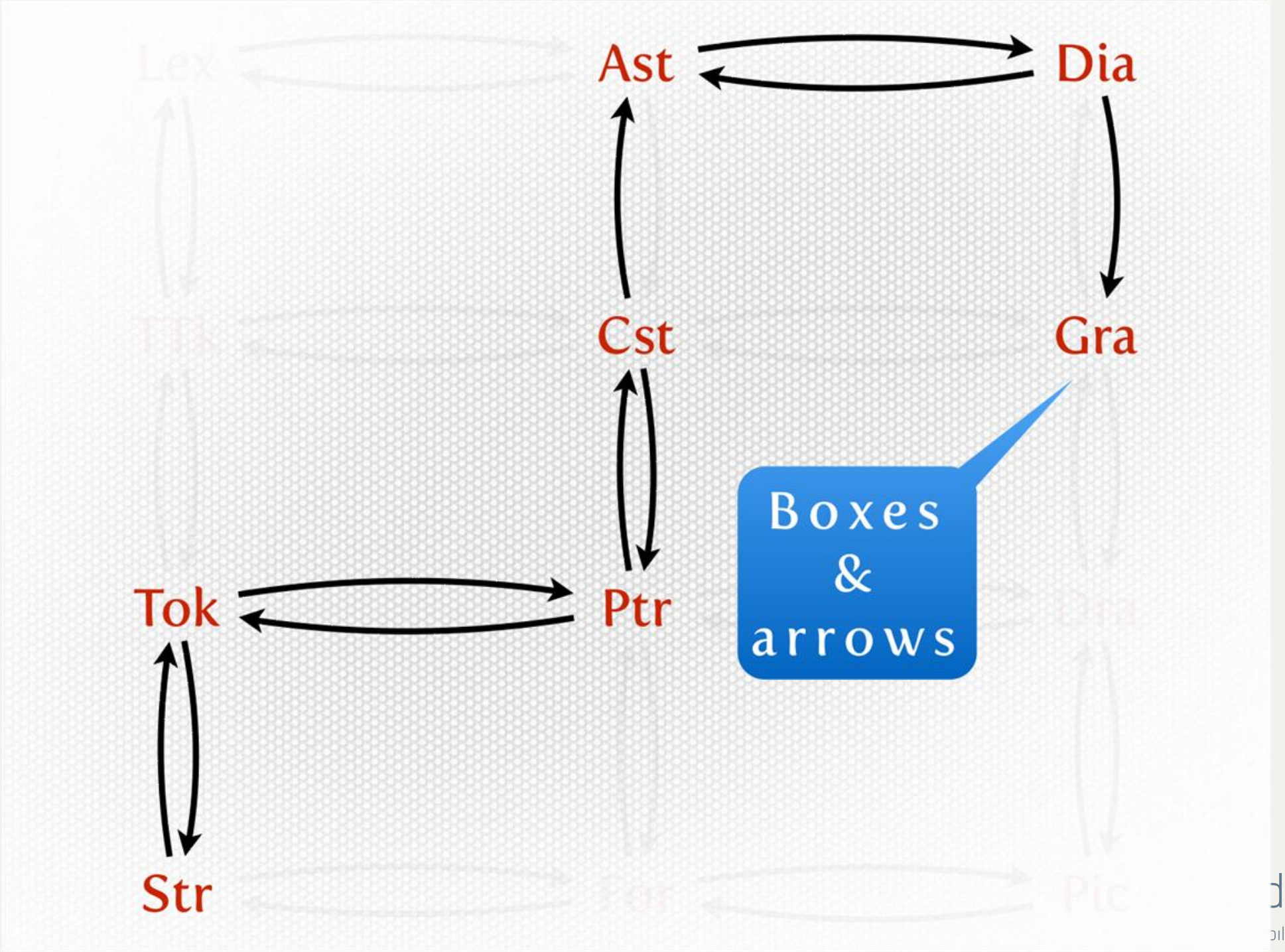


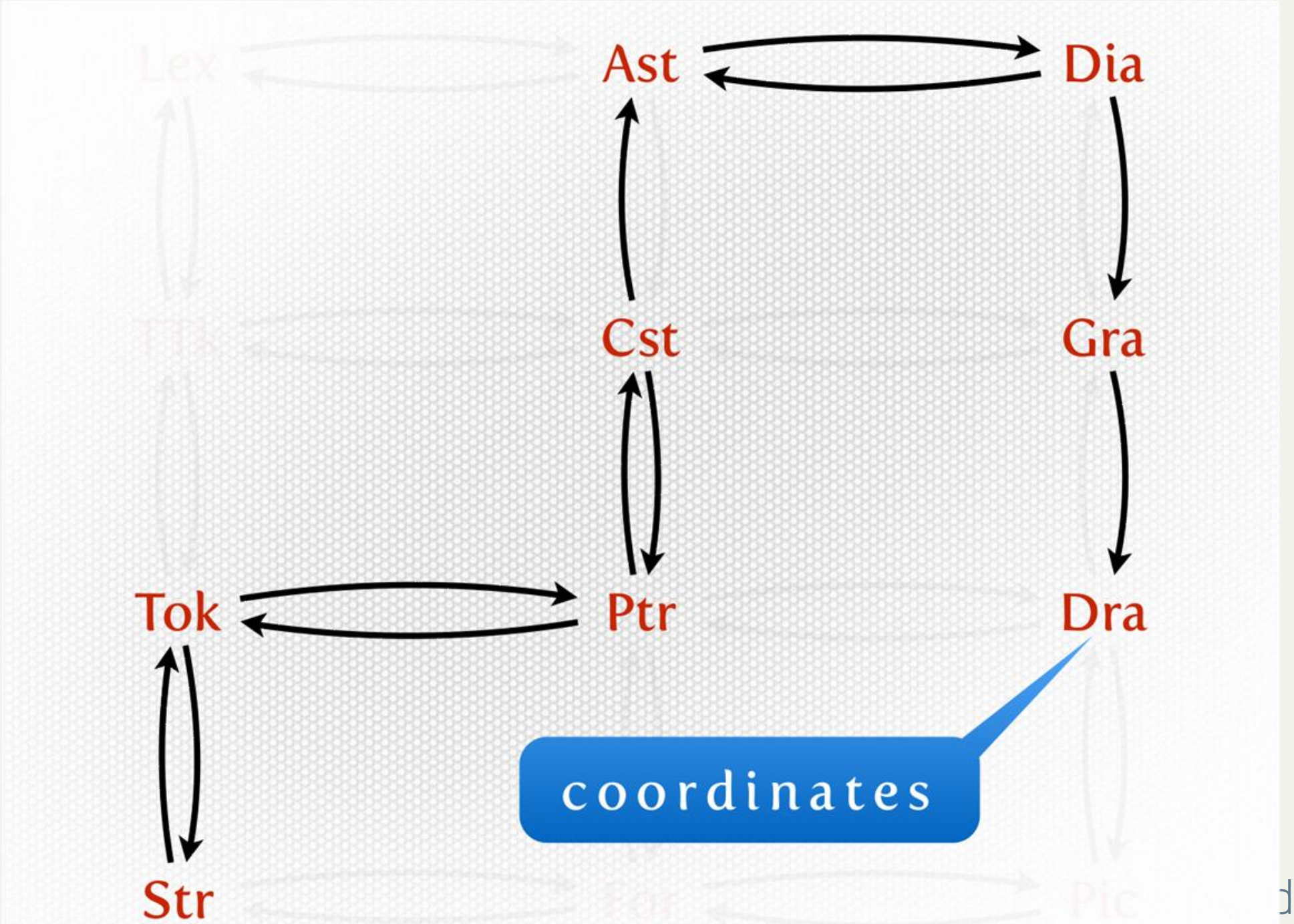


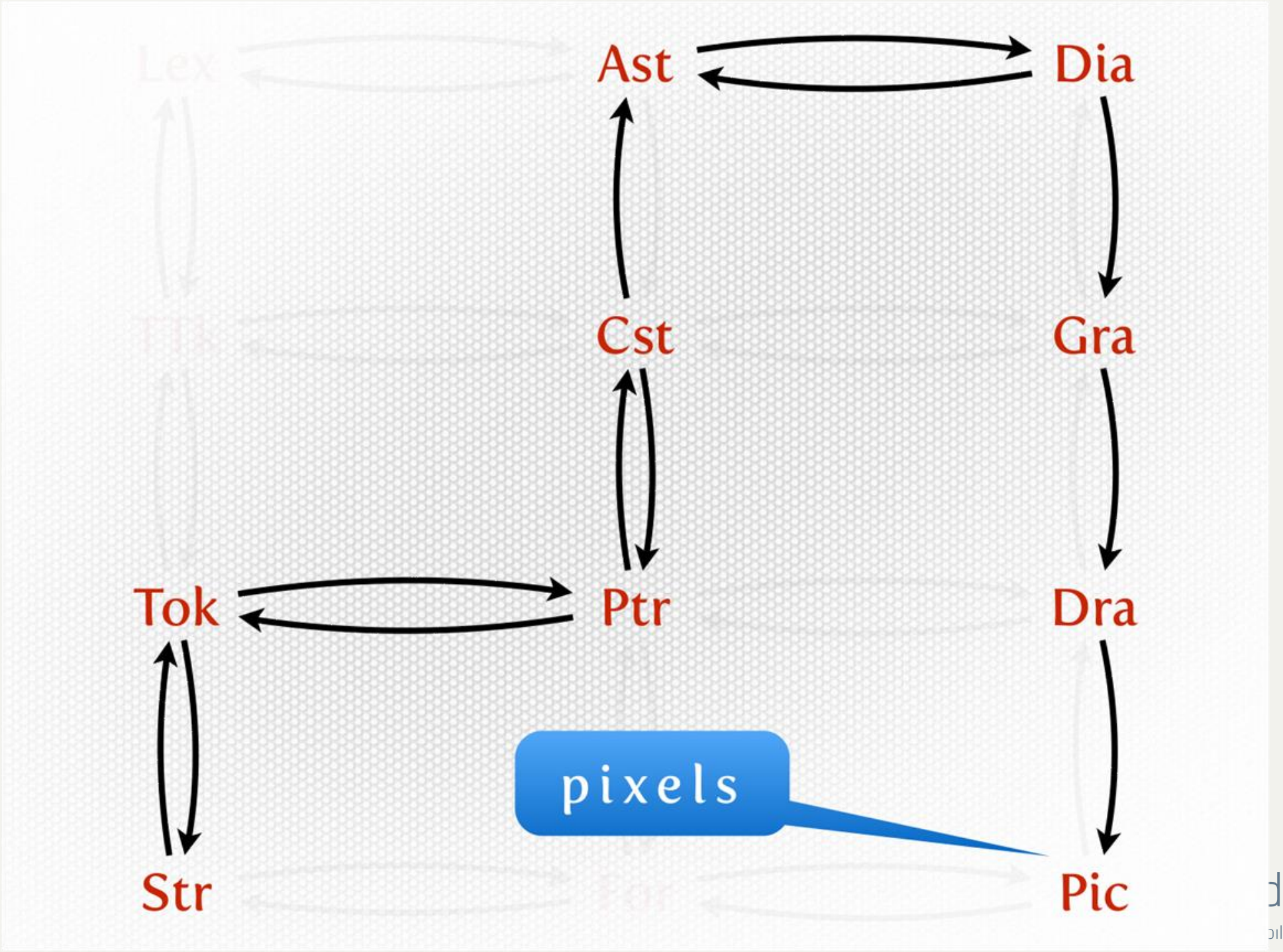


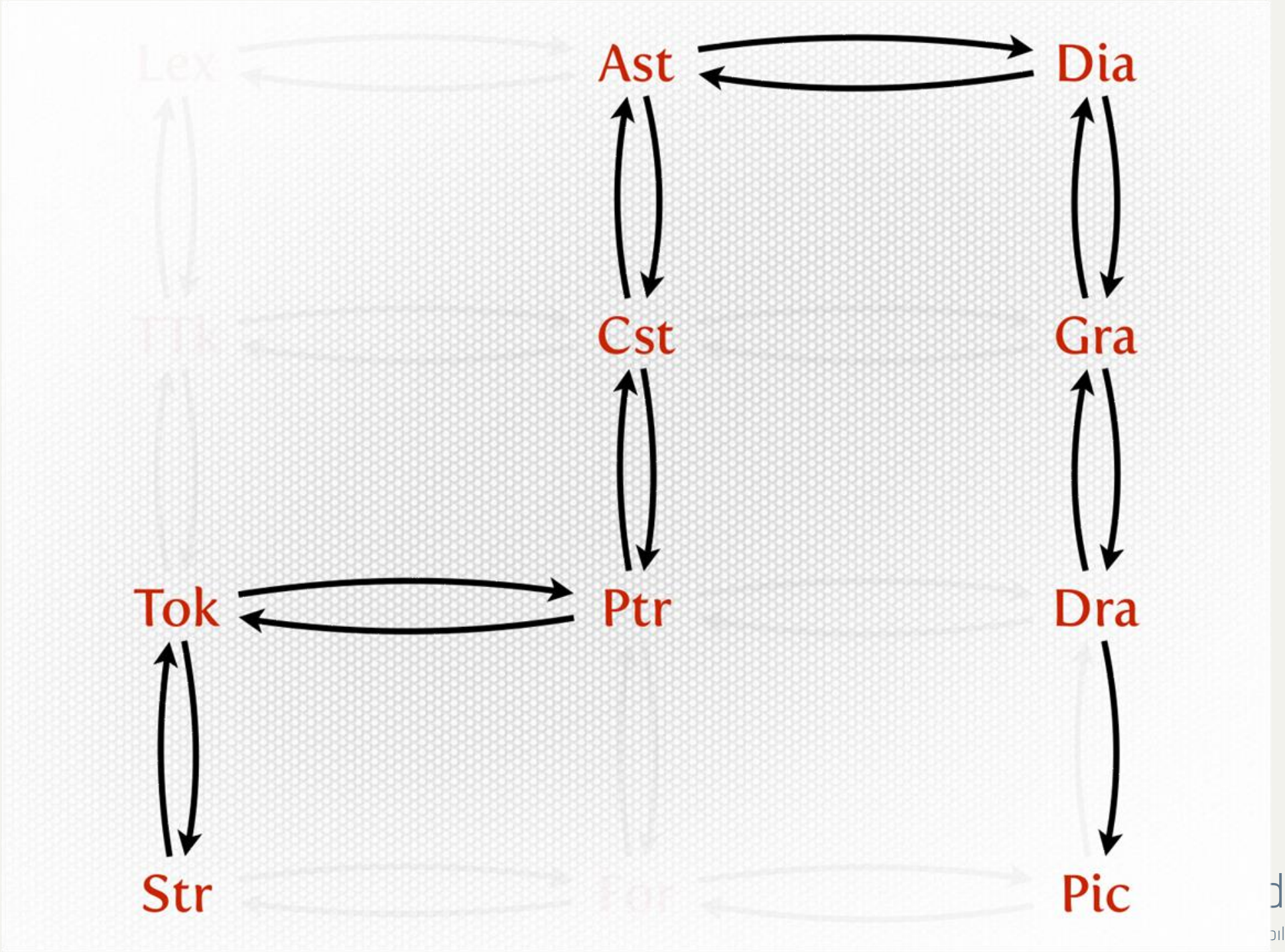


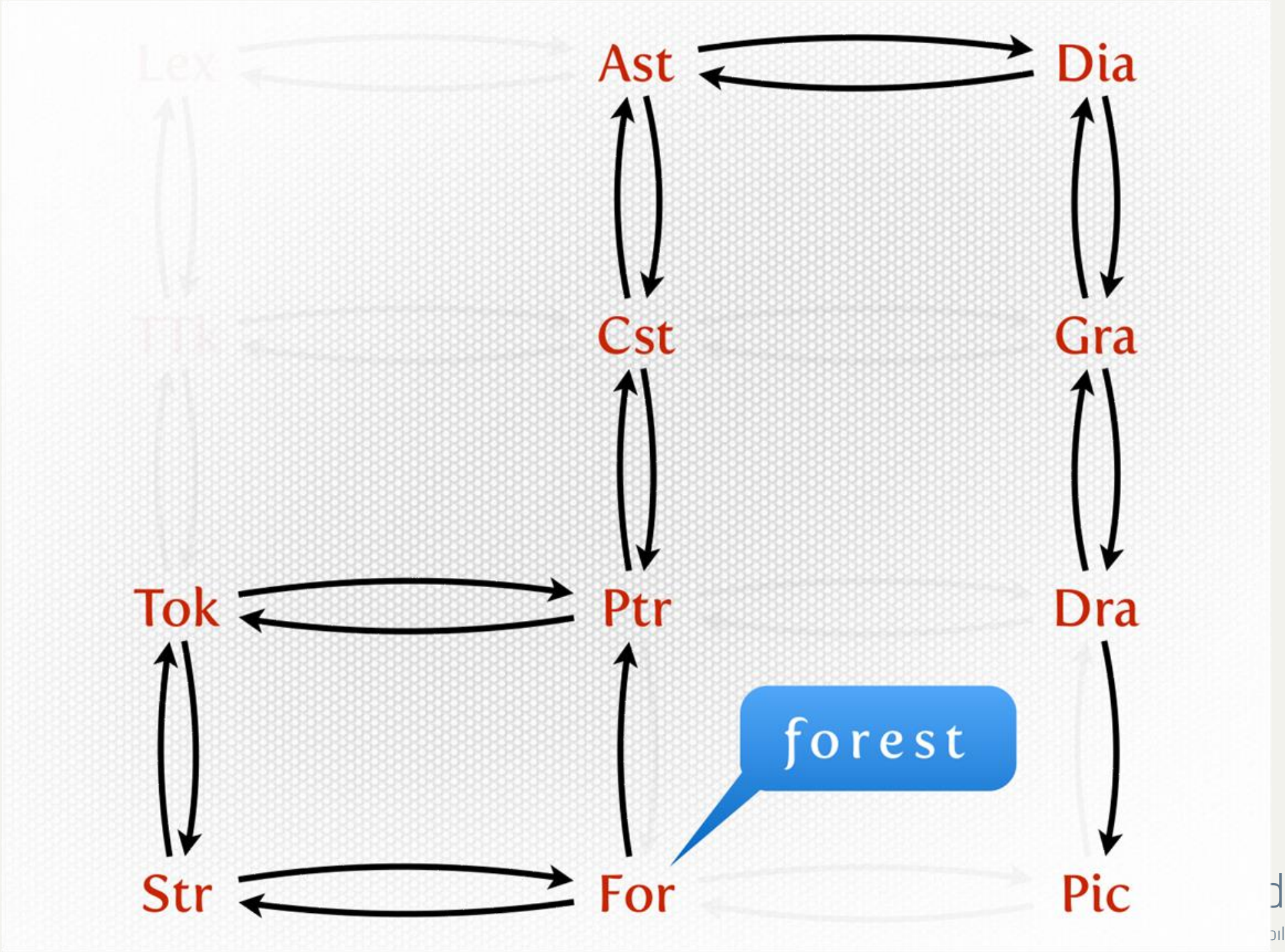


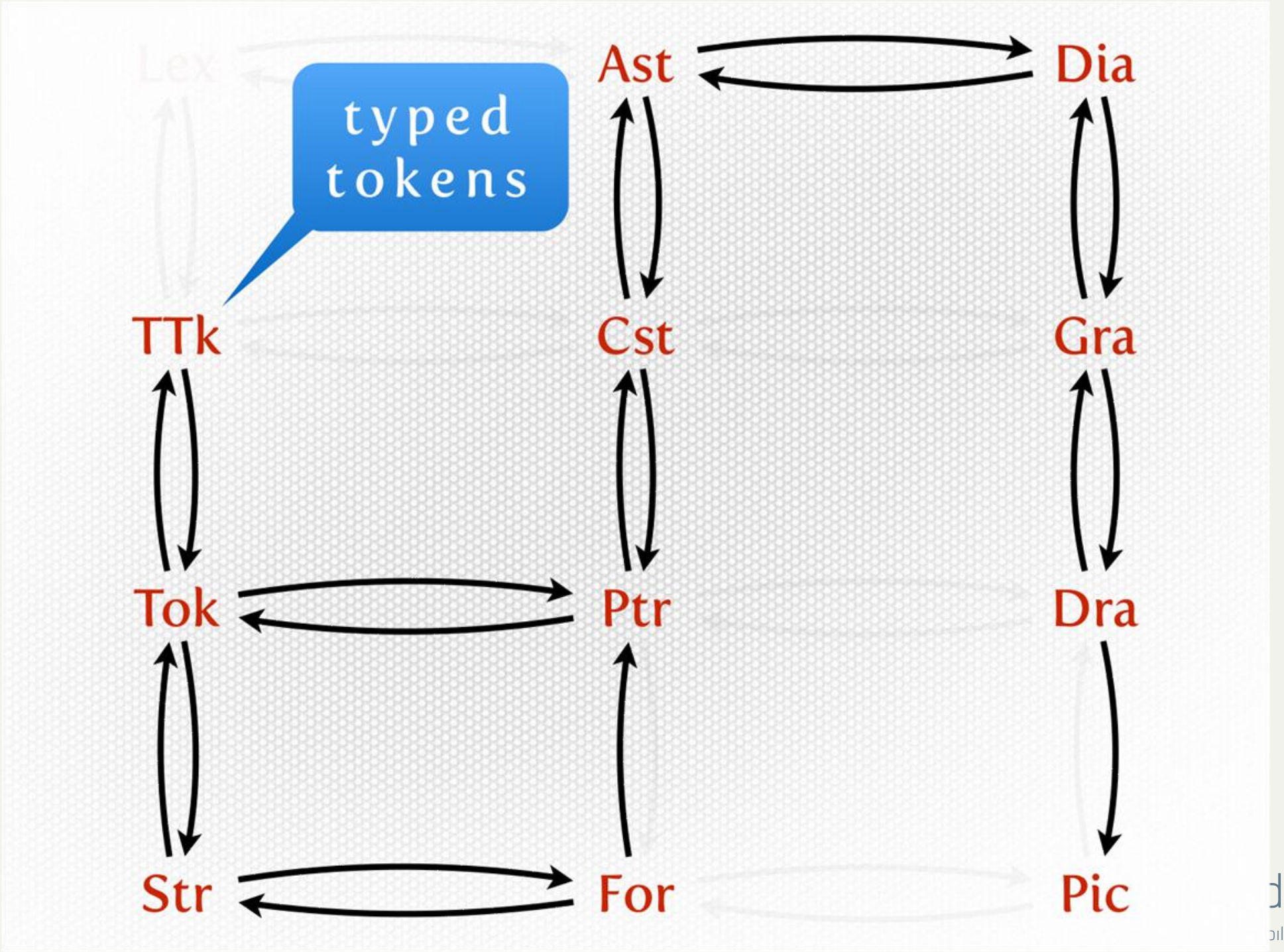


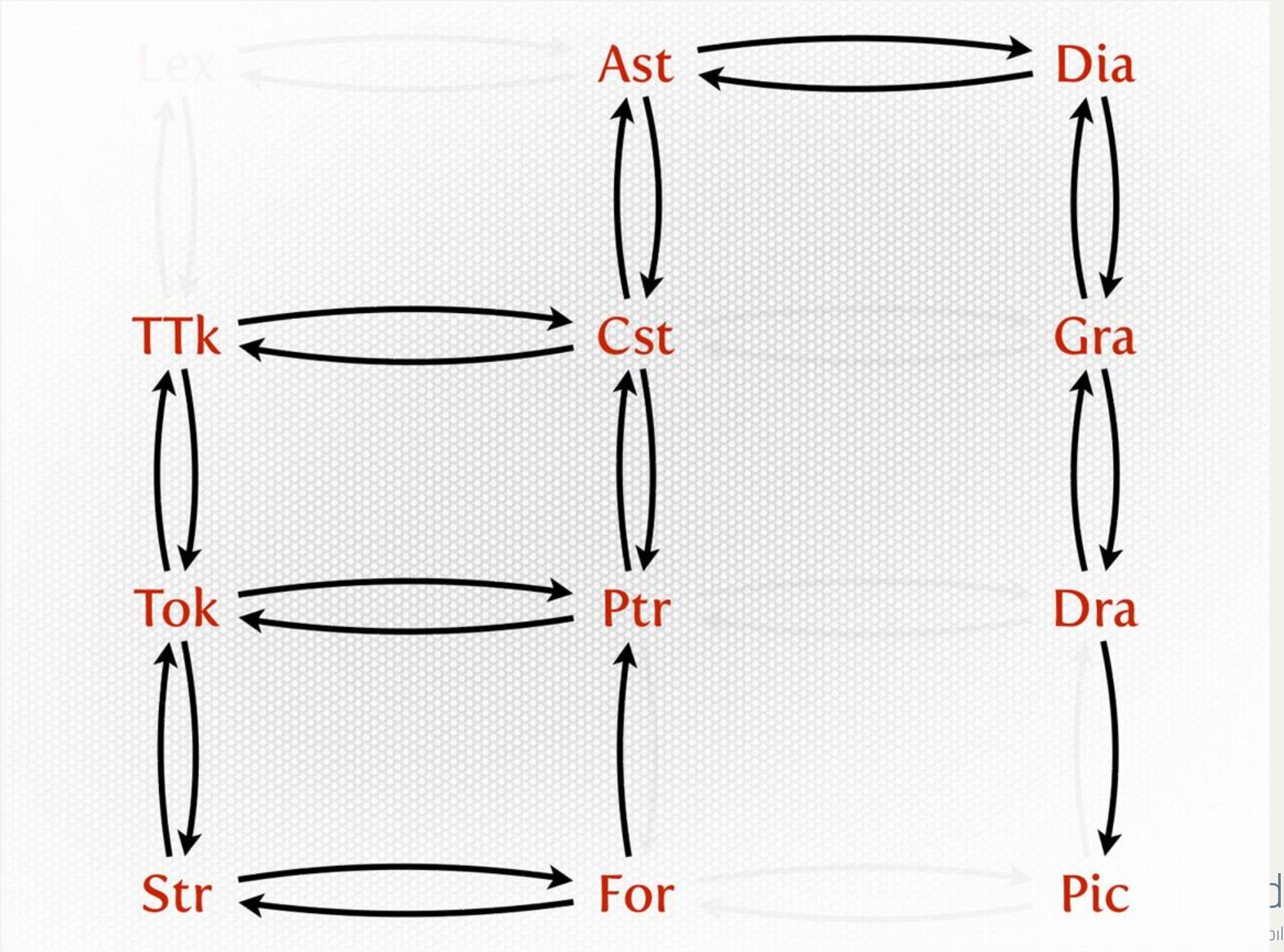


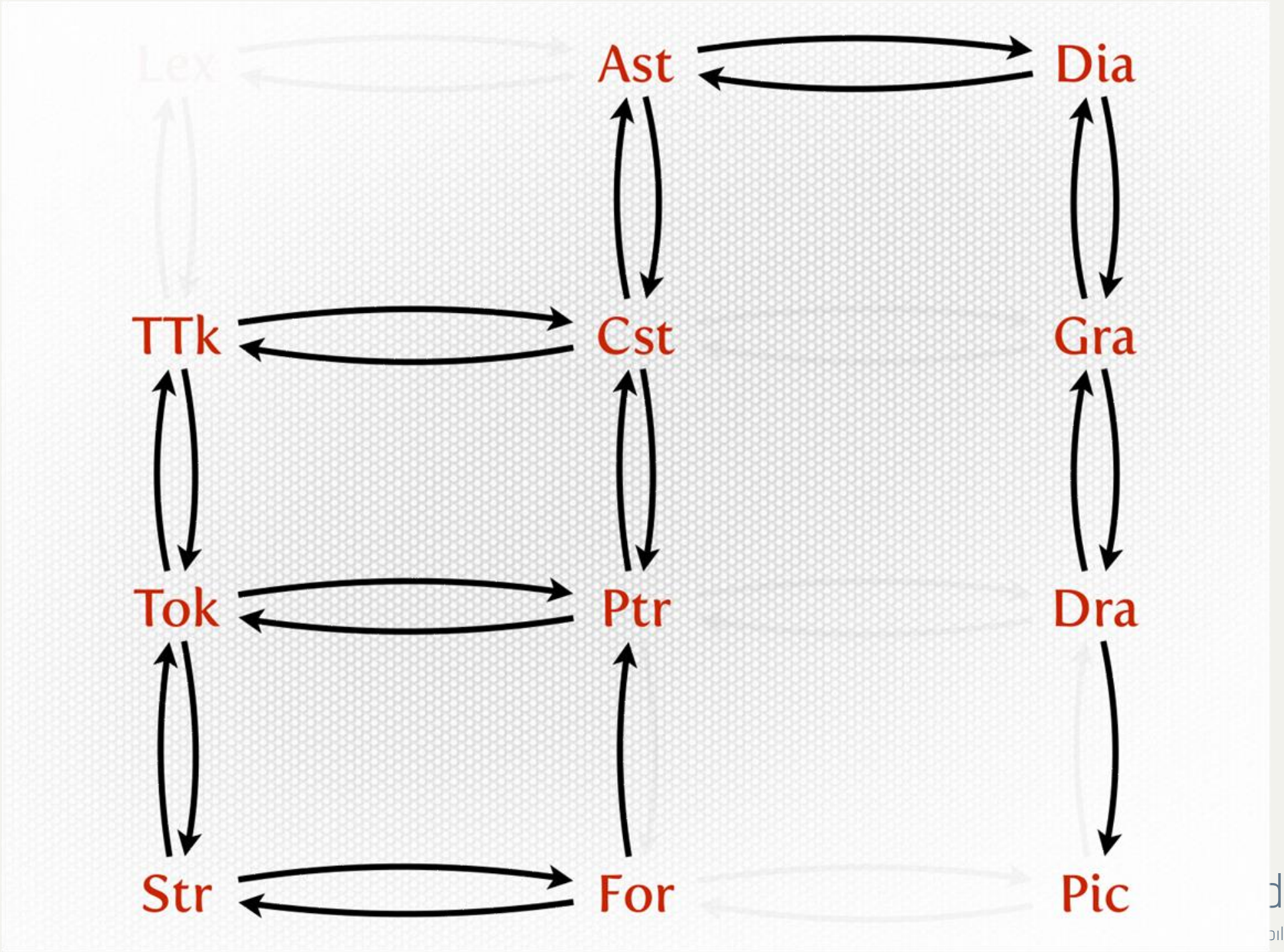


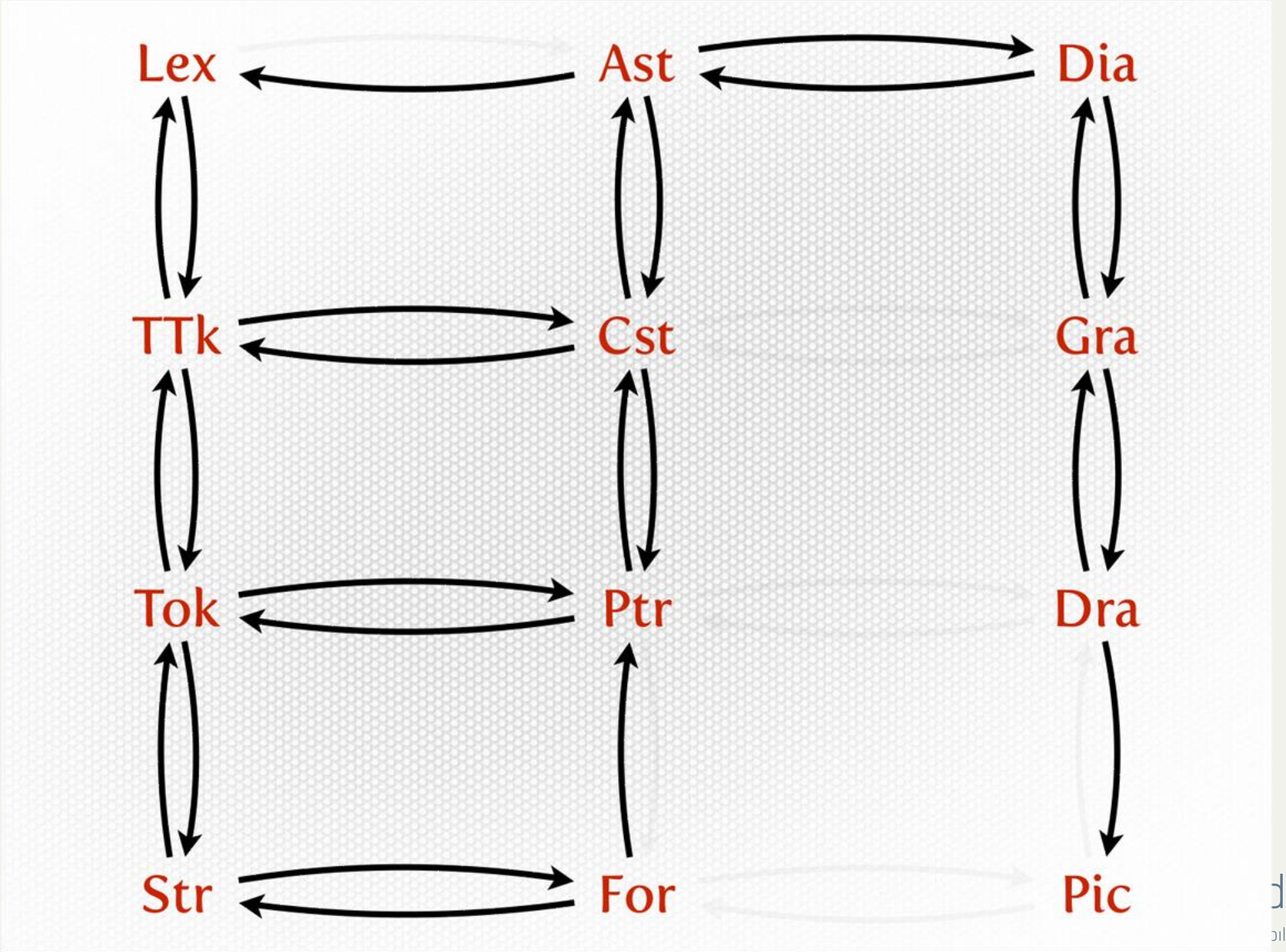


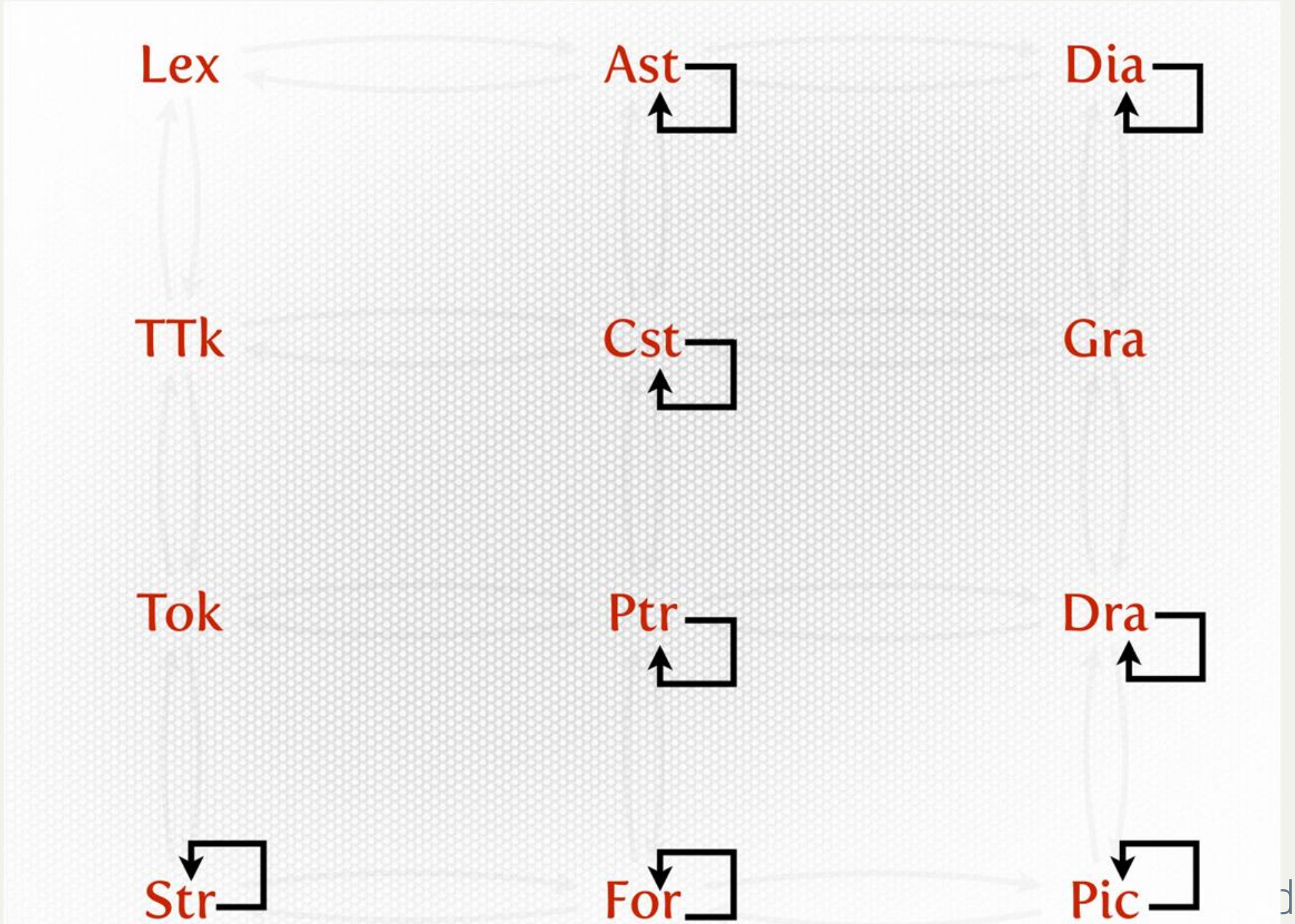


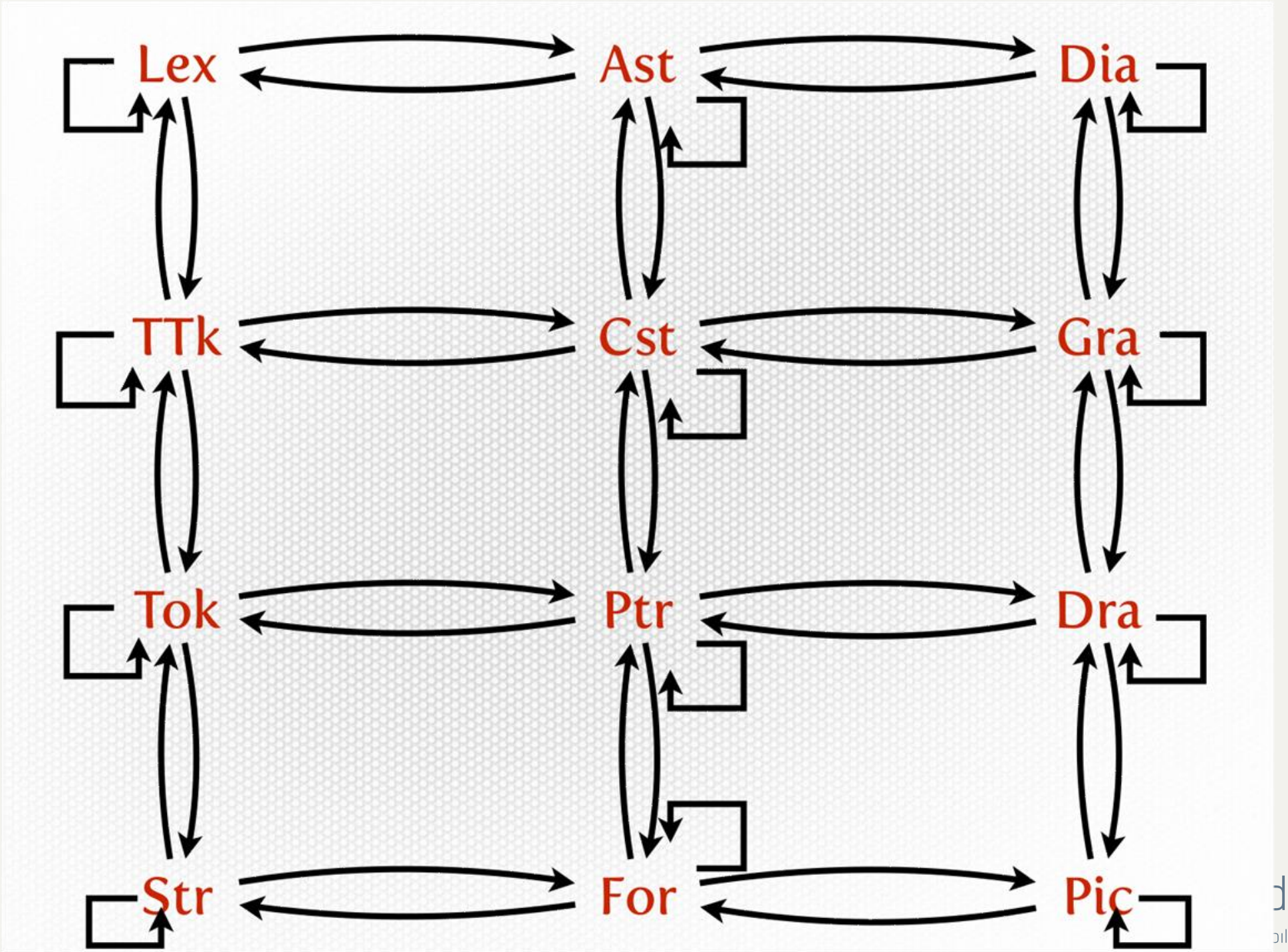

typed tokens
Ast
Dia
TTk
Cst
Gra
Tok
Ptr
Dra
Str
For
Pic

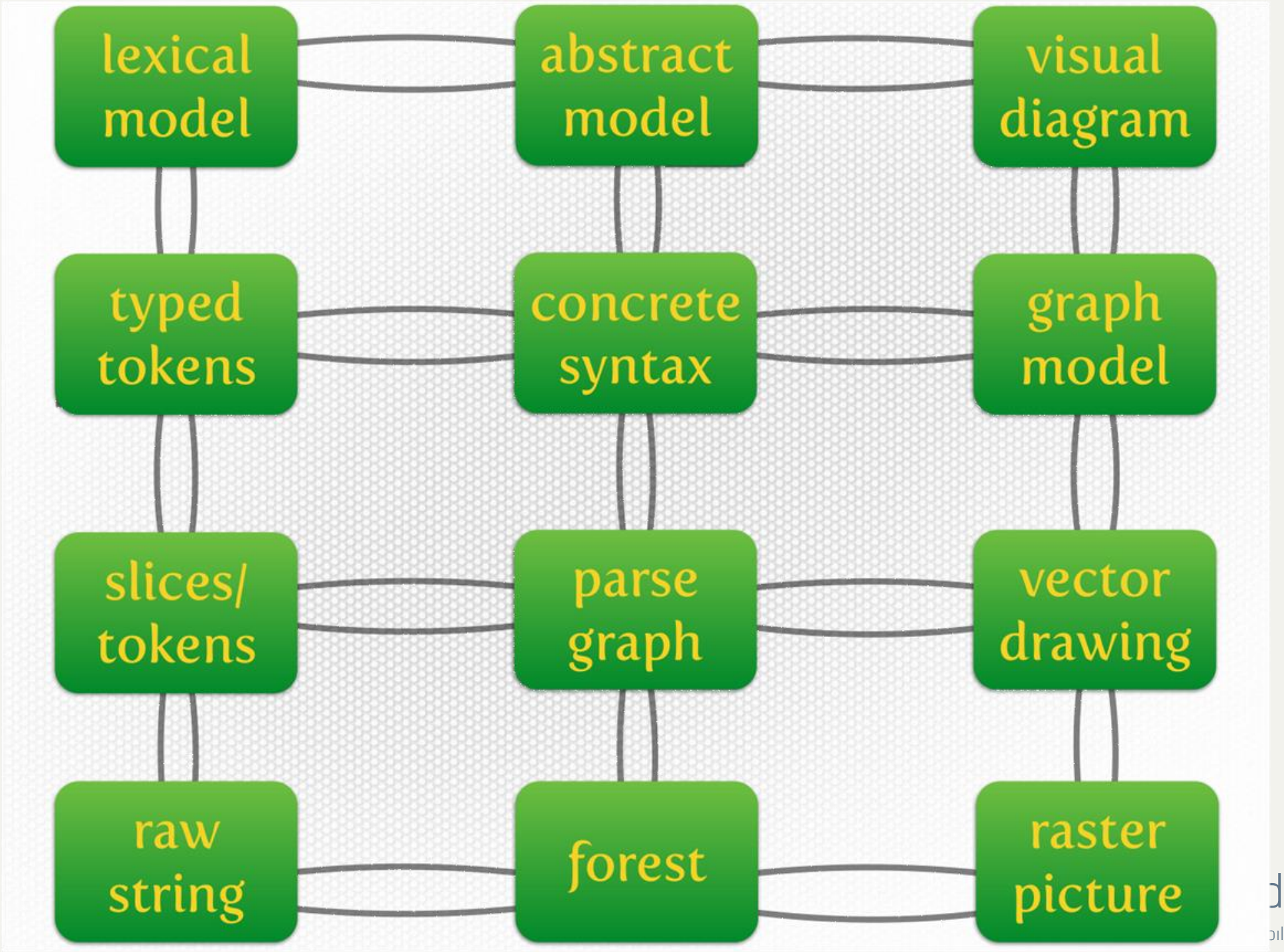












Bottom-up parsing

- Reduce the input back to the start symbol
- Recognise terminals
- Replace terminals by nonterminals
- Replace terminals and nonterminals by left-hand side of rule

- LR, LR(0), LR(1), LR(k), LALR, SLR, GLR, SGLR, CYK, …

Top-down parsing

- Imitate the production process by rederivation
- Each nonterminal is a goal
- Replace each goal by subgoals (= elements of its rule)
- Parse tree is built from top to bottom

- LL, LL(1), LL(k), LL(*), GLL, DCG, rec. descent, Packrat, Earley

How to parse with a grammar?

- Reuse an existing framework
- Write a parser manually
  - good error handling
  - possible fine-tuning
  - a lot of work (seriously, years)
- Generate with a parser generator
  - less work (maybe)
  - complex, rigid, idiosyncratic frameworks
  - difficult error handling

Parser generators: ↑

- LALR(1)
  - Beaver
  - YACC, byacc, bison, etc
  - Eli
  - Irony
  - SableCC
  - yecc
- GLR
  - bison
  - DMS
  - GDK
  - Tom
- SGLR
  - ASF+SDF Meta-Env
  - Spoofax, Stratego/XT

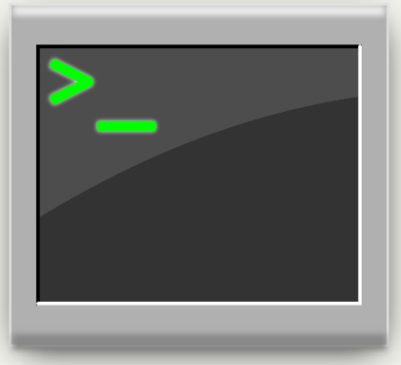
Parser generators: ↓

- LL(k)
  - JavaCC
- LL(*)
  - ANTLR
- Earley
  - Marpa
  - ModelCC
- GLL
  - Rascal — SGTDBF
  - gll-combinators (Scala)
- Packrat
  - Rats!
  - OMeta
  - PetitParser
- Others
  - TXL

Summary of parsing techniques

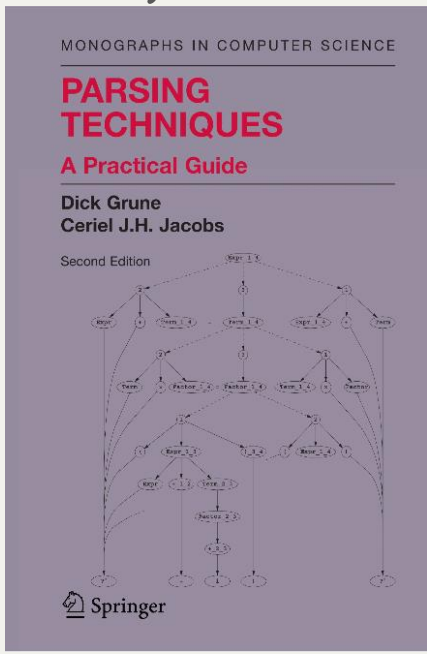
- Top-down
  - predict-match and variants/improvements
  - backtracking is good; memoisation is good
  - left recursion is generally problematic
- Bottom-up
  - shift-reduce and variants
  - deterministic CFLs in linear time
  - hard to debug

Conclusion



PARSING in a BROAD SENSE

- "Making a linear-time parser for an arbitrary given grammar is 10% hard work; the other 90% can be done by computer". [Grune/Jacobs, Parsing Techniques, 2008, p.81]
- Every notation/metatool defines a class of Gs/Ls
- Parsing should never be more complex than $O(n^3)$
- Many books/papers exist; beware of bullshit!



follow @grammarware!

raincode LABS
compiler experts