NAPA VALLEY 2008

CIKM

ACM 17th CONFERENCE ON INFORMATION and KNOWLEDGE MANAGEMENT

Tutorial T5

Evolution of Rule-based Information Extraction: From Grammars to Algebra

Rajasekar Krishnamurthy, Sriram Raghavan, and Huaiyu Zhu

IBM Almaden Research Center

Lots of Text, Many Applications!

- **Free-text, semi-structured, streaming …**
  - Web pages, emails, news articles, call-center records, business reports, spreadsheets, research papers, blogs, wikis, tags, instant messages, …
- **High-impact applications**
  - Business intelligence, personal information management, enterprise search, Web communities, Web search and advertising, scientific data management, e-government, medical records management, …
- **Growing rapidly**
  - Just look at your inbox!

(Adapted from SIGMOD '06 tutorial
by Ramakrishnan, Doan, and Vaithyanathan)

Information Extraction (IE)

- **Distill structured data from unstructured and semi-structured text**
- **Exploit the extracted data in your applications**

Annotations

For years, Microsoft Corporation CEO Bill Gates was against open source. But today he appears to have changed his mind. "We can be open source. We love the concept of shared source," said Bill Veghte, a Microsoft VP. "That's a super-important shift for us in terms of code access.“

Richard Stallman, founder of the Free Software Foundation, countered saying…

Select Name
From PEOPLE
Where Organization = ‘Microsoft’

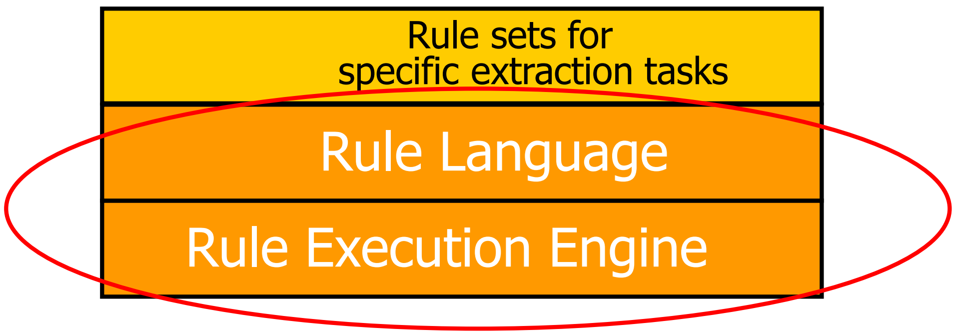
Name	Title	Organization
Bill Gates	CEO	Microsoft
Bill Veghte	VP	Microsoft
Richard Stallman	Founder	Free Soft..

Bill Gates
Bill Veghte

(from Cohen’s IE tutorial, 2003)

IE Techniques

Rule-based Approaches



Focus of this tutorial

Learning-based approaches

- Naive Bayes
- AUTOSLOG [Riloff-1993] and AUTOSLOG-TS
- LIEP [Huffman95], CRYSTAL [Soderland98], RAPIER [Cali et. al. 97]
- SRV [Freitag-98]
- WHISK [Soderland99]
- Hidden Markov Models [Leek, 1997]
- Maximum Entropy Markov Models [McCallum et al, 2000]
- Conditional Random Fields [Lafferty et al, 2000]
- Semi-supervised approaches that learn to gather more training data – DIPRE [Brin98], Snowball [Agichtein00],

This Tutorial in a Nutshell

Emerging Applications
Opinion mining, Community management, Customer care, Business Intelligence, Enterprise Search, …

⇨

Complex extraction tasks on a large scale
➔
Expressive high performance IE engines

Rule-based IE

Classical grammar-based approaches

- Based on the formalism of **cascading grammars & finite-state automata**
- Designed with classical entity extraction tasks in mind
  - Simple entity extraction (e.g., people names, company names, ..)
  - Link/Relationship extraction between such entities

Approaches based on declarative queries

- Infusion of **"database ideas"**
  - Extraction rules as database queries
  - Performance optimization
  - Alternate execution plans

Roadmap

- **Part 1 [Sriram Raghavan]**
  - Grammar-based extraction systems
  - Newer motivating applications
  - Limitations of grammar-based extraction
- **Part 2 [Huaiyu Zhu]**
  - Extended grammar-based solutions
  - Modern declarative approaches
- **Part 3 [Rajasekar Krishnamurthy]**
  - SystemT in-depth
  - Research directions

SystemT Demo &
Install of Development Environment

PART 1

Historical Perspective

- **Information Extraction**
  - Active research topic across many different research communities
  - Originally NLP & IR communities but more recently machine learning, Web, databases, ……
- **Strongly influenced by two competitions**
  - Message Understanding Conference (MUC)
  - Automatic Content Extraction (ACE)
- **MUC (Message Understanding Conference) – 1987 to 1997**
  - Competition-style conferences organized by DARPA
  - Shared data sets and performance metrics
    - News articles, Radio transcripts, Military telegraphic messages
  - Several IE systems were built during this period
    - FRUMP [DeJong82], CIRCUS [Riloff93], FASTUS [Appelt96], LaSIE/GATE, TextPro, PROTEUS, OSMX [Embley05]

Classical IE Tasks

- **Entity extraction**
  - Person names, Locations, Organization names, ….
  - Recently expanded to include newer entity types such as disease names, protein names, paper titles, journal names, etc.
    - E.g., ACE competition lists more than 100 different specific types
- **Relationship/Link extraction**
  - relationships between entities
    - e.g., person worksFor company, company1 acquired company2, …..
- **Entity resolution**
  - matching multiple mentions of the same entity, within and across documents

Finite-state Grammars

- **Common formalism underlying most of these IE systems**
  - Input text viewed as a sequence of tokens
  - Rules expressed as regular expression patterns over the lexical features of these tokens
- **Several levels of processing → Cascading Grammars**
  - A typical IE task was decomposed into
    - Low-level tokenization (e.g., word segmentation)
    - Morphological and Lexical processing (e.g., POS tagging, word sense tagging)
    - Syntactic analysis (e.g., shallow parsing)
    - Domain analysis (e.g., task-specific grammar rules)
  - Typically, at higher levels of the grammar, larger segments of text are analyzed and annotated

Example Cascading Grammar

- Set of simple grammar rules for person name recognition

Pre-processing step outside of the grammar.

Level 0

Tokenize(Document Text)	→ Sequence of <Token>

Level 1

⟨Token⟩[~ "Mr. | Mrs. | Dr. | …"]	→ ⟨Salutation⟩
⟨Token⟩[~ "Ph.D | MBA | …"]	→ ⟨Qualification⟩
⟨Token⟩[~ "[A-Z][a-z]*"]	→ ⟨CapsWord⟩
⟨Token⟩[~ "Michael | Richard | Smith| …"]	→ ⟨PersonDict⟩

Level 2

⟨PersonDict⟩ ⟨PersonDict⟩	→ ⟨Person⟩	Richard Smith
⟨Salutation⟩ ⟨CapsWord⟩ ⟨CapsWord⟩	→ ⟨Person⟩	Dr. Laura Haas
⟨CapsWord⟩ ⟨CapsWord⟩ ⟨Token⟩[~","]**?** ⟨Qualification⟩	→ ⟨Person⟩	Laura Haas, Ph.D

Common Pattern Specification Language (CPSL)

- **Motivation**
  - Each IE system had its own rule formalism tied to a particular implementation
  - CPSL attempted to separate rule specification and matching semantics from the implementation

- **CPSL 101**
  - A common language to specify and represent finite-state transducers
  - Each transducer accepts a sequence of annotations and outputs a sequence of annotations
  - CPSL interpreter maintains a cursor at the "current" position in text
  - All possible grammar rules are matched at current position
  - Longest match is chosen
    - Rule priority is used to break ties amongst longest matches
    - Output annotation(s) is produced corresponding to this match
    - Cursor moves to the next position past this match

CPSL

- **Most widely adopted "standard" for grammar-based IE systems**
- **Several known implementations**
  - TextPro: reference implementation of CPSL by Doug Appelt
  - JAPE (Java Annotation Pattern Engine)
    - Part of the GATE NLP framework
    - Under active commercial use by several companies

The modern face of IE

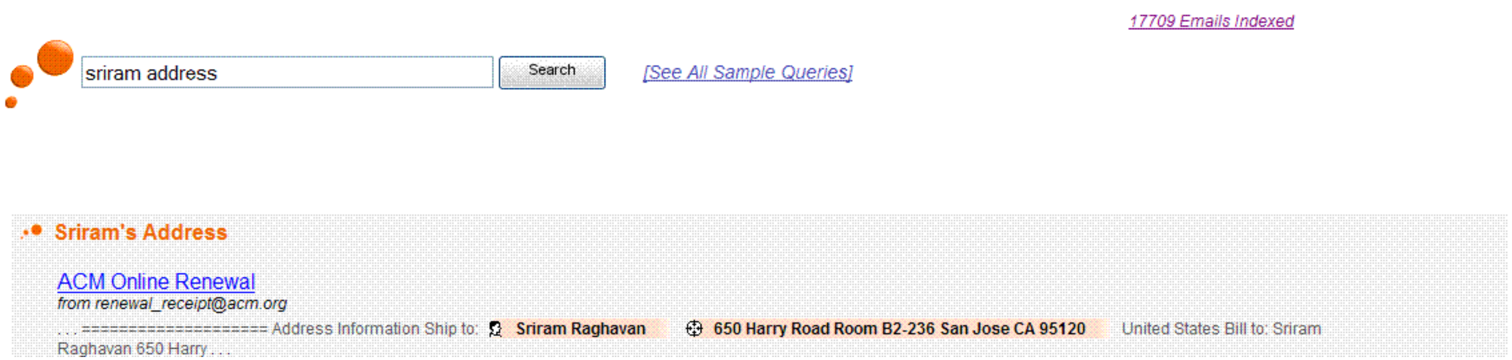
- **Emerging applications within and outside the enterprise**
  - Enterprise Search, Personal Information Management, Business Intelligence, Community Information Management, Customer Care, …..
- **New challenges for IE**
  - Noisy heterogeneous text collections
    - Emails, blogs, customer call records, etc., as opposed to homogenous well-written text such as news reports
  - Complex IE tasks
    - Reviews, Opinions, Sentiments, etc., as opposed to just entities & relationships
- **Demands on the IE engine**
  - Expressivity (as we deal with more complex tasks)
  - Performance (as we deal with larger and larger text collections)

Running Examples

- **Noisy text collection**
  - From personal email, extracting
    - Example 1: Person names
    - Example 2: Person's phone relationships
    - Example 3: Signature blocks
- **Complex extraction task**
    - Example 4: Extracting informal reviews of musical bands from blogs

IBM OmniFind Personal Email Search (IOPES)

- **Exploit IE to enable high-precision semantic search over email**
- Extraction of **entities** (persons, phone numbers, locations, etc.,), **relationships** (person ↔ phone number, person ↔ address, etc.), and **complex entities** (like conference schedules, driving directions, signature blocks, etc.)



Example 1 – Person Names

- **Simple Rules**
  - ⟨Token⟩[~ "[A-Z][a-z]*"] → ⟨CapsWord⟩
  - ⟨Token⟩[~ "Michael | Richard | Smith| …"] → ⟨PersonDict⟩
  - ⟨PersonDict⟩ ⟨PersonDict⟩ → ⟨Person⟩
  - ⟨Salutation⟩ ⟨CapsWord⟩ → ⟨Person⟩
  - ⟨Salutation⟩ ⟨CapsWord⟩ ⟨CapsWord⟩ → ⟨Person⟩
- **Example**
  - A piece of text "… Dr. John Smith …" results in three matches:
    - John Smith
    - Dr. John
    - Dr. John Smith
- **Problem**
  - Multiple overlapping matches: we want only Dr. John Smith
  - Classical grammar-based systems depend on **rule priority**
    - Implicit (e.g., longest match from a given point)
    - Explicit (anticipate rule interactions and set priorities appropriately)

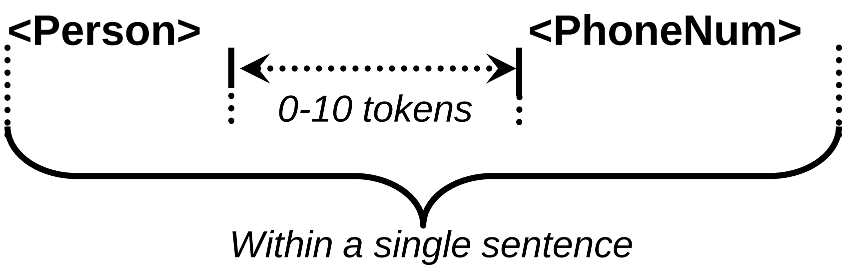
Example 1 – Person Names

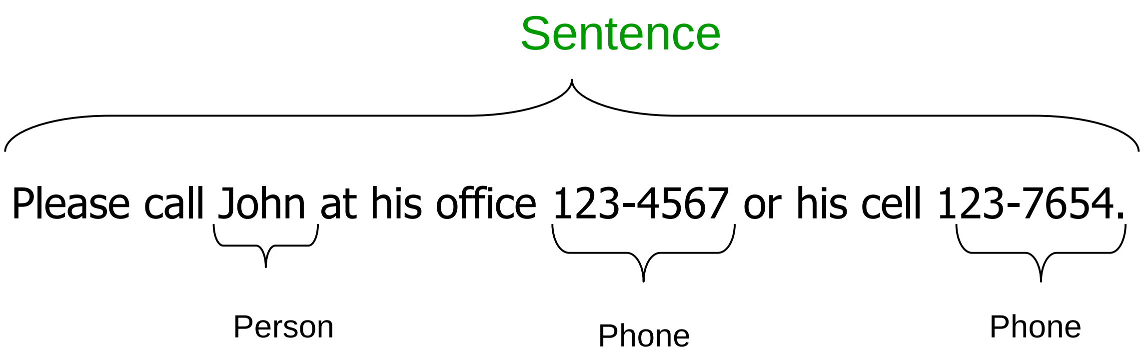
- When text is noisy and heterogeneous → names appear in numerous different ways
  - *Mr. Dabrowski received a Bachelor degree…*
  - *Dr. Jean L. Rouleau Dean of Medicine University…*
  - *…met Peter and Katie Lawton who have…*
  - *…lives in Riverdale, NY, with his wife Marie-Jeanne. He has two married sons, James and Michael.*
  - *The Honorable Carol Boyd Hallett - Of Counsel…*
  - *Kimberly Purdy Lloyd received a Bachelor of Science degree from the University of Texas…*
  - *……attendees Ida White, Bridget McBean, Volker Hauck ….*
  - ……..many more……

Example 1 – Person Names

- To cover all of these possibilities, a good high-quality person name extractor for emails requires **numerous rules**
  - E.g., over 100 rules for the Person name annotator in an email search application
- When using grammars,
  - Reasoning about the interactions between this many rules to set appropriate priorities becomes **unmanageable!!**
- Better approach
  - Allow rules to match independently
  - Use the concept of **consolidation** to address overlapping matches (details in Parts 2 & 3)

Example 2 – Person's Phone





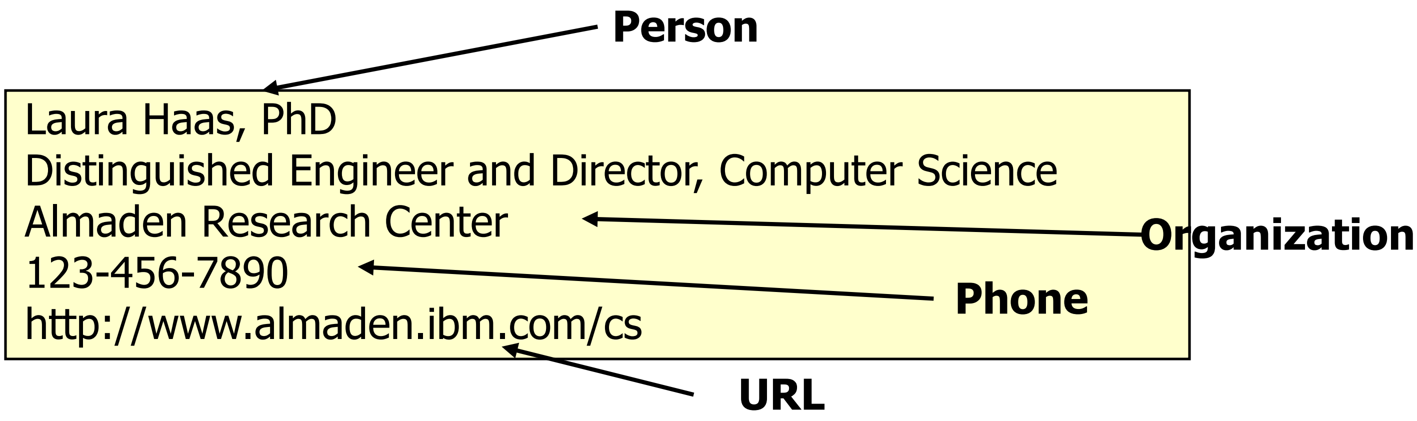
Desired Output

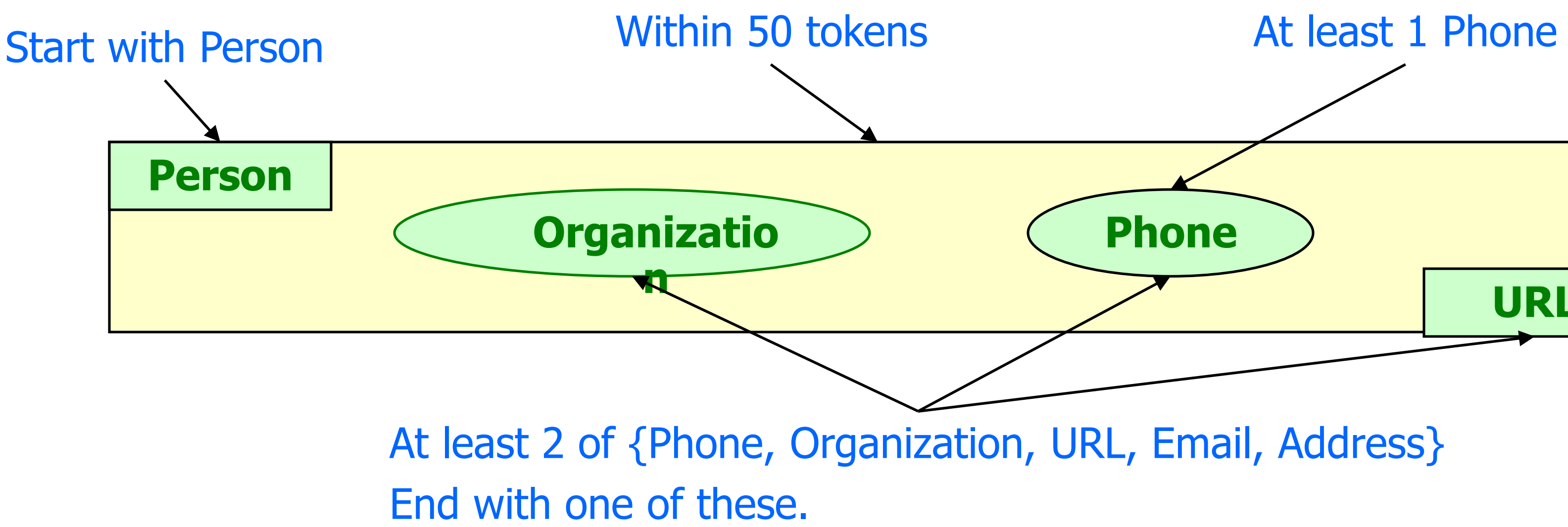
(John, 123-4567)

(John, 123-7654).

- Example illustrates two problems with classical grammar-based systems:
  - Do not support overlapping output annotations
  - Do not support span-based predicates (to express the condition that the span of text matched by the rule must be fully contained within the span of a sentence)

Example 3 - Signature Block Extraction





Example 3 - Signature Block Extraction

- **First approximation**
  - Macro: ⟨**Contact** ⟩ = ⟨**Phone**⟩|⟨**Organization**⟩|⟨**URL**⟩
  - Rule: ⟨**Person**⟩ (.{,25} ⟨**Contact**⟩){2,} → ⟨**Signature**⟩
  - Problems:
    - Cannot guarantee at least one phone → false positives
    - Cannot express the restriction that total token count must be < 50 → false positives and false negatives

- **Second approximation**
  - Rule: (⟨**Person**⟩ .{,25} ⟨**Phone**⟩ (.{,25} ⟨**Contact**⟩)+) |

    (⟨**Person**⟩.{,25} (⟨**Contact**⟩.{,25})+ ⟨**Phone**⟩ (.{,25} ⟨**Contact**⟩)*)
  - Problems:
    - Rule becomes combinatorially more complex as the number of count constraints increases
    - Still cannot express restriction on total token count

Example 3 - Signature Block Extraction

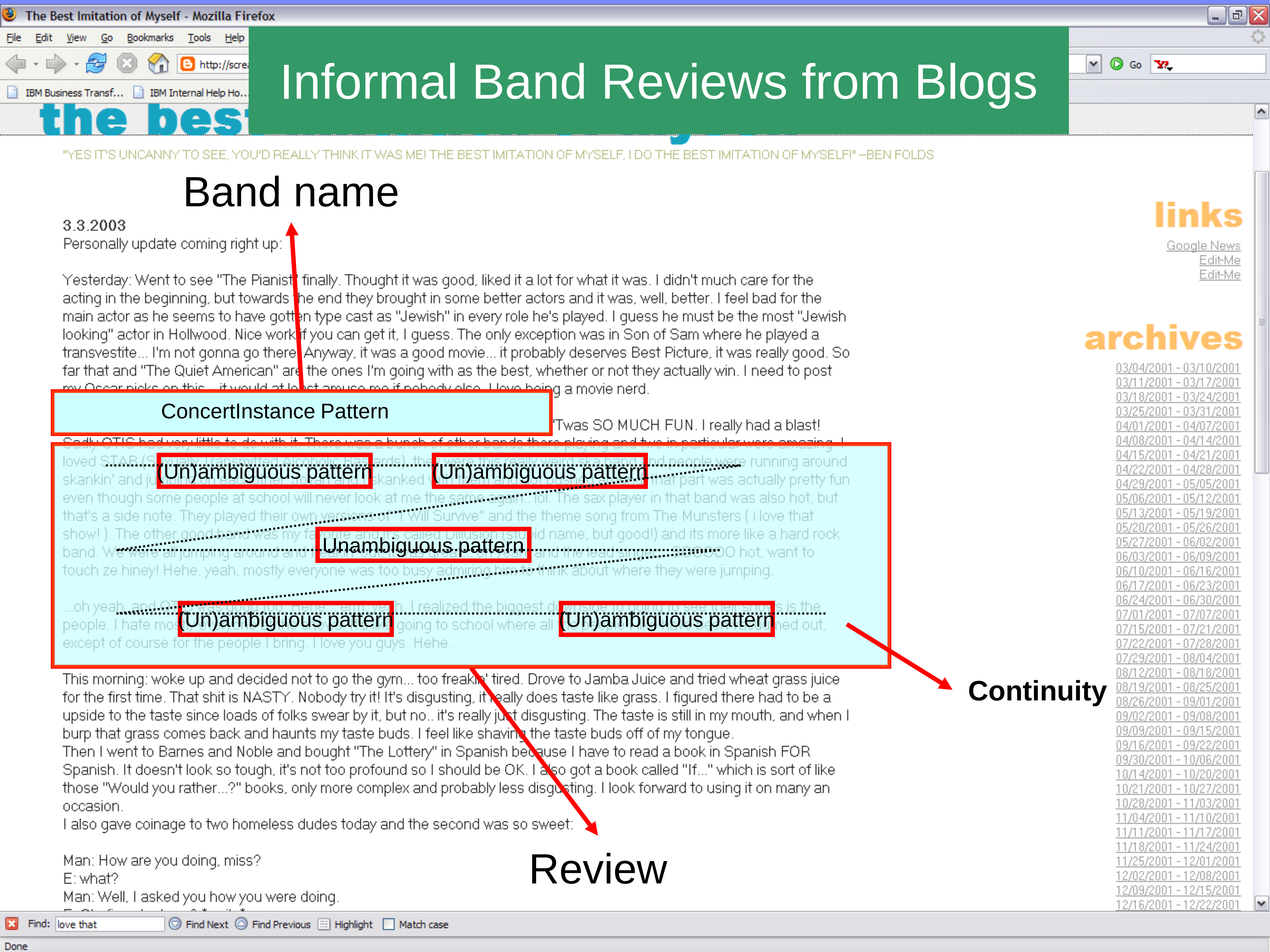
- **Signature Block extraction rule had the following**
  - Start and end annotations
  - Maximum length of matching region
  - Minimum count of one kind of annotation
  - Minimum count of several kinds of annotations
- **Using grammars**
  - Unable to faithfully represent these conditions
  - Even approximations involve combinatorial blow up in the number of rules
- **Takeaway**
  - Grammars lack support for **window-based counts**

Example 4 - Band review

- **Extract informal reviews of band performances posted on blogs**
- **Example**

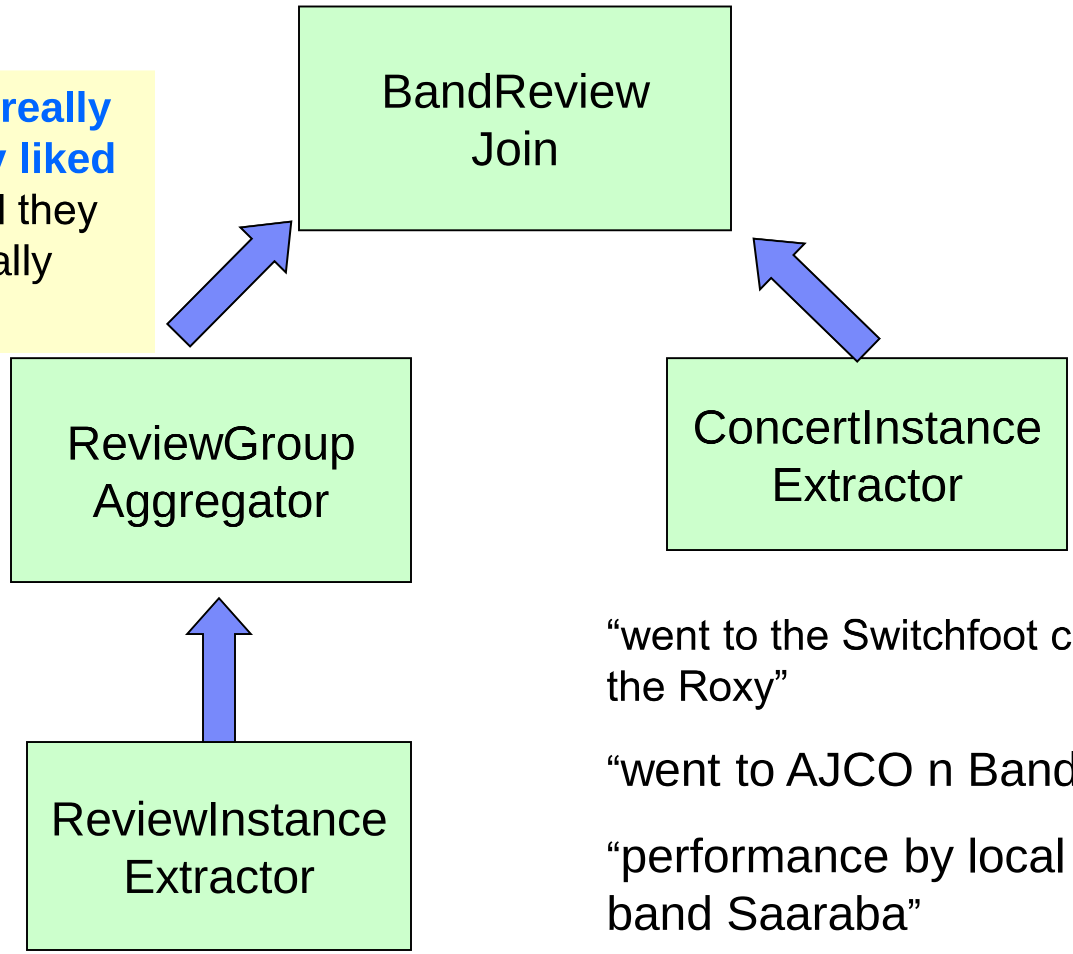
went to the Switchfoot concert at the Roxy. It was pretty fun,… The lead singer/guitarist was really good, and even though there was another guitarist (an Asian guy), he ended up playing most ofthe guitar parts, which was really impressive. The biggest surprise though is that I actually liked the opening bands. …I especially liked the first band

Informal Band Reviews from Blogs



went to the Switchfoot concert at the Roxy. It was pretty fun,… **The lead singer/guitarist was really good**, and even though there was another guitarist (an Asian guy), he ended up playing most of the guitar parts, which was really impressive. The biggest surprise though is that I actually **liked the opening bands**. …I especially **liked the first band**

“Lead singer/guitarist was really good, and even … **I actually liked the opening bands**. … Well they were none of those. I especially **liked the first band”**



“lead singer/guitarist was really good”

“Liked the opening bands”

“Liked the first band”

“Kurt Ralske played guitar”

“put on a great show”

“went to the Switchfoot concert at the Roxy”

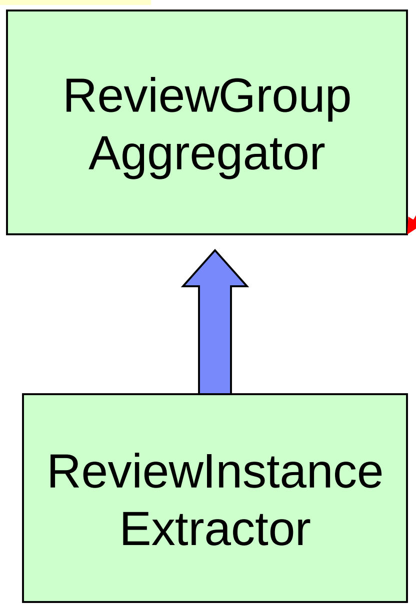
“went to AJCO n Band concert”

“performance by local funk band Saaraba”

Band Review: Window-based Count Problem

"**Lead singer/guitarist was really good**, and even … **I actually liked the opening bands**. … Well they were none of those. I especially **liked the first band**"

Computation of ReviewGroup requires the same kind of window-based counts that we saw in Signature and was hard to do with grammars



"lead singer/guitarist was really good"

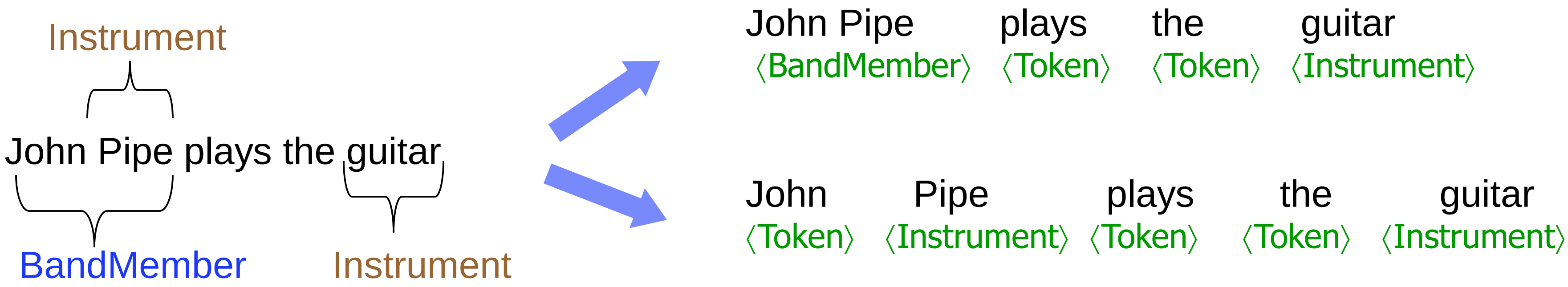
"Liked the opening bands"

"Liked the first band"

"Kurt Ralske played guitar"

"put on a great show"

Band Review – Sequencing Input Annotations

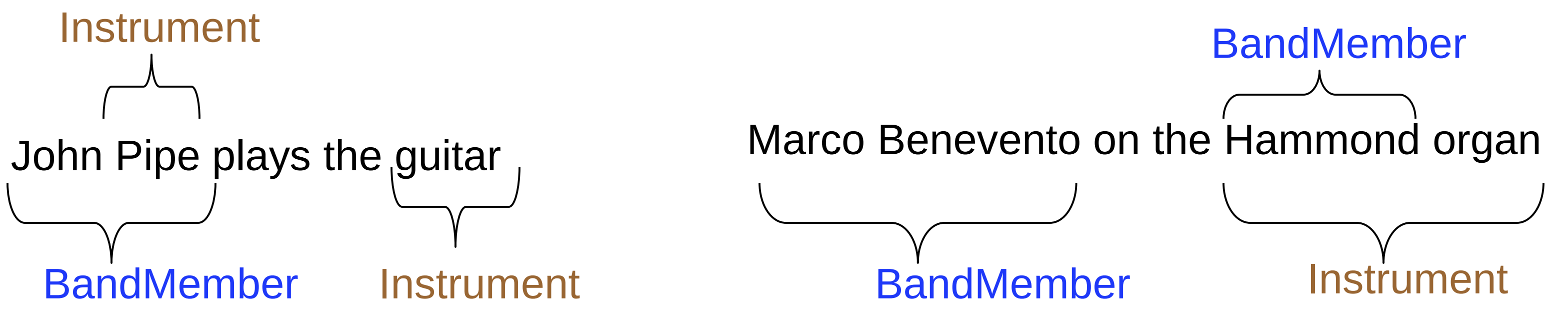


Which of the two should we pick?

- **Problem**
  – Grammars do not permit **overlapping annotations on input**
  – A potential lattice of annotations must be serialized into a token stream before being fed as input
- **Typical approaches adopted (each has issues)**
  – Pre-specified disambiguation rules (e.g., pick the annotation that starts earlier)
  – Manually provide tie-breaking rules (e.g., annotation type A trumps annotation type B)
  – Let the implementation make an internal non-deterministic choice

Sequencing Problems Continued..

⟨BandMember⟩ ⟨Token⟩{0,5} ⟨Instrument⟩



Case (A) **Case (B)**

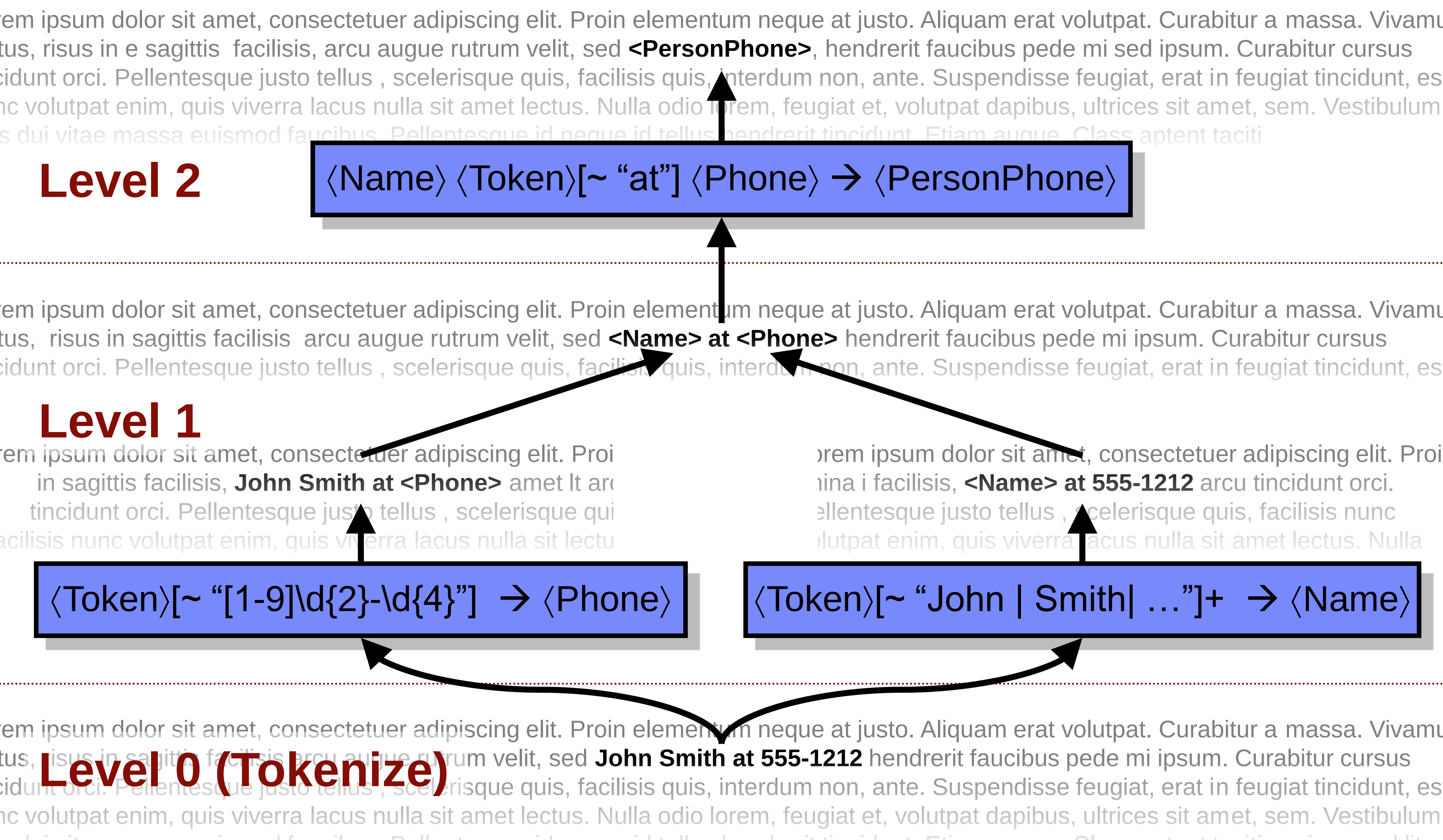
– If we pick Instrument over BandMember, we miss case (A). Other way round, we miss case (B).

Over 4.5M blog entries, our experiments showed that a choice one way or another would change the number of annotations by +/- 25%.

Summary: Limitations of Classical Grammar-based Extraction

- **Expressivity problems**
  - Consolidation (Person)
  - OutputOverlap (Person's Phone)
  - SpanPredicate (Person's Phone)
  - WindowCount (Signature & BandReview)
  - InputOverlap (BandReview)
- **Performance problems**

Cascading Grammars By Example



Performance: Existing Solutions

- **Performance issues**
  - Complete pass through tokens for each rule
  - Many of these passes are wasted work
- **Dominant approach: Make each pass go faster**
  - Faster finite state machines
- **Doesn’t solve root problem!**

Using a finely tuned grammar-based extraction system, processing 4.5M blogs for reviews took over 7hrs.

Can we do better??

PART 2

Roadmap

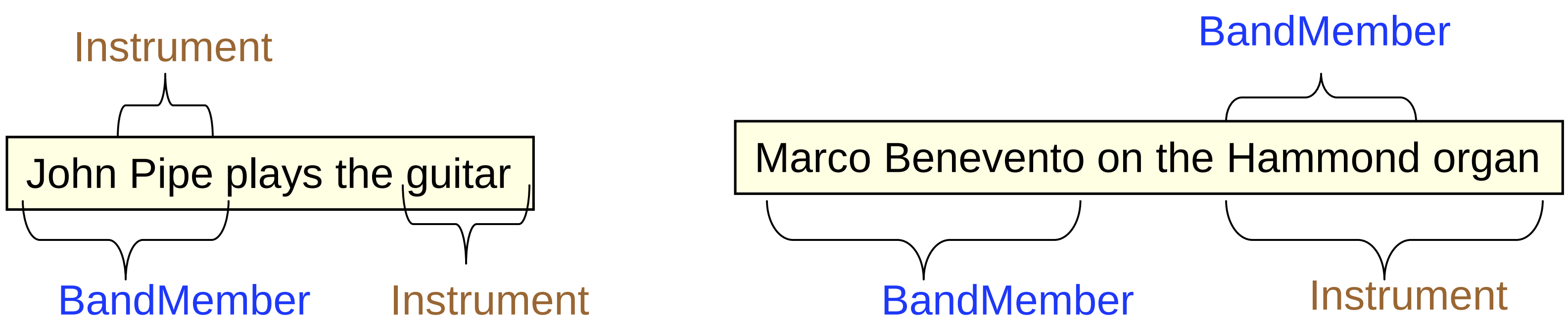
- Part 1 [Sriram Raghavan]
  - Grammar-based extraction systems
  - Newer motivating applications
  - Limitations of grammar-based extraction
- **Part 2 [Huaiyu Zhu]**
  - Extended grammar-based solutions
  - Modern declarative approaches
- Part 3 [Rajasekar Krishnamurthy]
  - SystemT in-depth
  - Research directions

Overcoming Limitations of Classical Grammar

- **Extended grammar based systems**
  - **AFst** (Annotation-Based Finite State Transducer)
    - Developed at IBM Watson Research Center.
  - **JAPE** (Java Annotation Patterns Engine)
    - Developed at University of Sheffield
- **Systems based on declarative queries**
  - **CIMPLE** (declarative IE with Datalog)
    - Developed at University of Wisconsin
  - **SystemT** (declarative IE using an extraction algebra)
    - Developed at IBM Almaden Research Center

AFst enhancements

- **Overcomes InputOverlap problem**
  - Input is a **lattice of annotations** as opposed to a sequence → multiple annotations may cover overlapping regions of text



Case (A) **Case (B)**

⟨BandMember⟩ ⟨Token⟩{0,5} ⟨Instrument⟩ → (John Pipe, guitar)
(Marco Benevento, Hammond organ).

AFst enhancements

- **Partially overcomes SpanPredicate problem**
  - *Boundary Annotations*
    - Restrict scope of a rule to be within span of specified annotation type
    - Example
      - produce rule matches that are always contained within *Sentence* annotations
  - *Honor Annotations*
    - Do not apply a rule if the match overlaps with the span of other specified annotation type.
    - Example
      - produce Year annotations but only if the tokens are not covered by *StreetAddress*
- **However,**
  - These span predicates are built-in extensions

JAPE (Java Annotation Patterns Engine)

- An implementation of CPSL with extensions
- Partially address the **OutputOverlap** problem
  - Support for getting multiple overlapping outputs from different rules
  - Several *control styles* for a grammar
    - All, Brill, Appelt, First, Once
- **However,**
  - a single rule cannot produce multiple overlapping matches starting from the same position.
  - so the following problem remains

Please call John at his office 123-4567 or his cell 123-7654.

Person (John) — Phone (123-4567) — Phone (123-7654)

Cannot get both (John, 123-4567) and (John, 123-7654).

JAPE

- **Partially overcomes SpanPredicate problem**
  - Contextual operators: contains, within
    - {A contains B} is equivalent to {B within A}.
    - Example
      - {PersonPhone within Sentence}
- **However,**
  - This is a built-in operator. It does not allow arbitrary span predicates.

Solutions: Roadmap

- Extended grammar-based systems
- Extraction systems based on declarative queries
  - **CIMPLE**
  - SystemT

CIMPLE (declarative IE with Datalog)

- **Overview**
  - Lowest level extraction through user defined predicates
    - Procedural code (Perl, Java, C++, …)
  - Higher level extraction workflow expressed using **Xlog**
    - A Datalog-based language with text related notions such as span, containment, document, etc.

- **Advantages of using Xlog**
  - Cleaner organization than custom code.
  - Allow application of query optimization techniques.

Datalog with Embedded Procedural Predicates

Talks

"Feedback in IR"
Relevance feedback is important ...

"Personalized Search"
Customizing rankings with relevance feedback ...

docs
d_1
d_2
d_3

title	abstract
"Feedback in IR"	"Relevance feedback is important..."
"Personalized Search"	"Customizing rankings with relevance feedback..."

titles(d,t) :- docs(d), extractTitle(d,t). → perl module

abstracts(d,a) :- docs(d), extractAbstract(d,a). → C++ module

talks(d,t,a) :- titles(d,t), abstracts(d,a),
immBefore(t,a), → perl module
contains(a,"relevance feedback"). → perl module

(Adapted from VLDB '07: Shen et. al.)

Person's Phone Example in Xlog

- personsphone(p,t,d) :- docs(d),
  extractPerson(d,p),
  extractPhone(d,t),
  distTokens(p,t) < 10

- Procedural predicates (*p-predicates)*
  - Two *p-predicates* corresponding to extractPerson and extractPhone
- Procedural functions (*p-functions*)
  - A *p-function* corresponding to distTokens

Solutions: Roadmap

- Extended grammar-based systems
- Extraction systems based on declarative queries
  - CIMPLE
  - **SystemT**

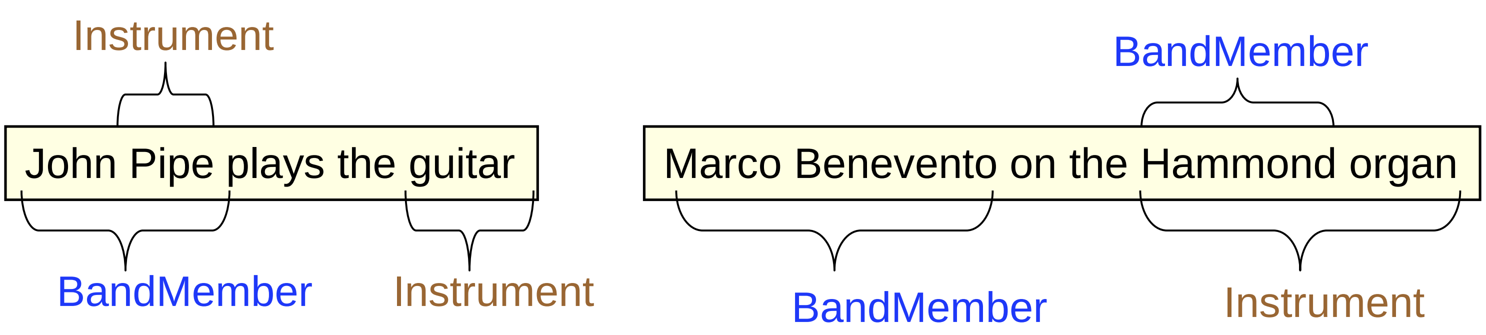
SystemT

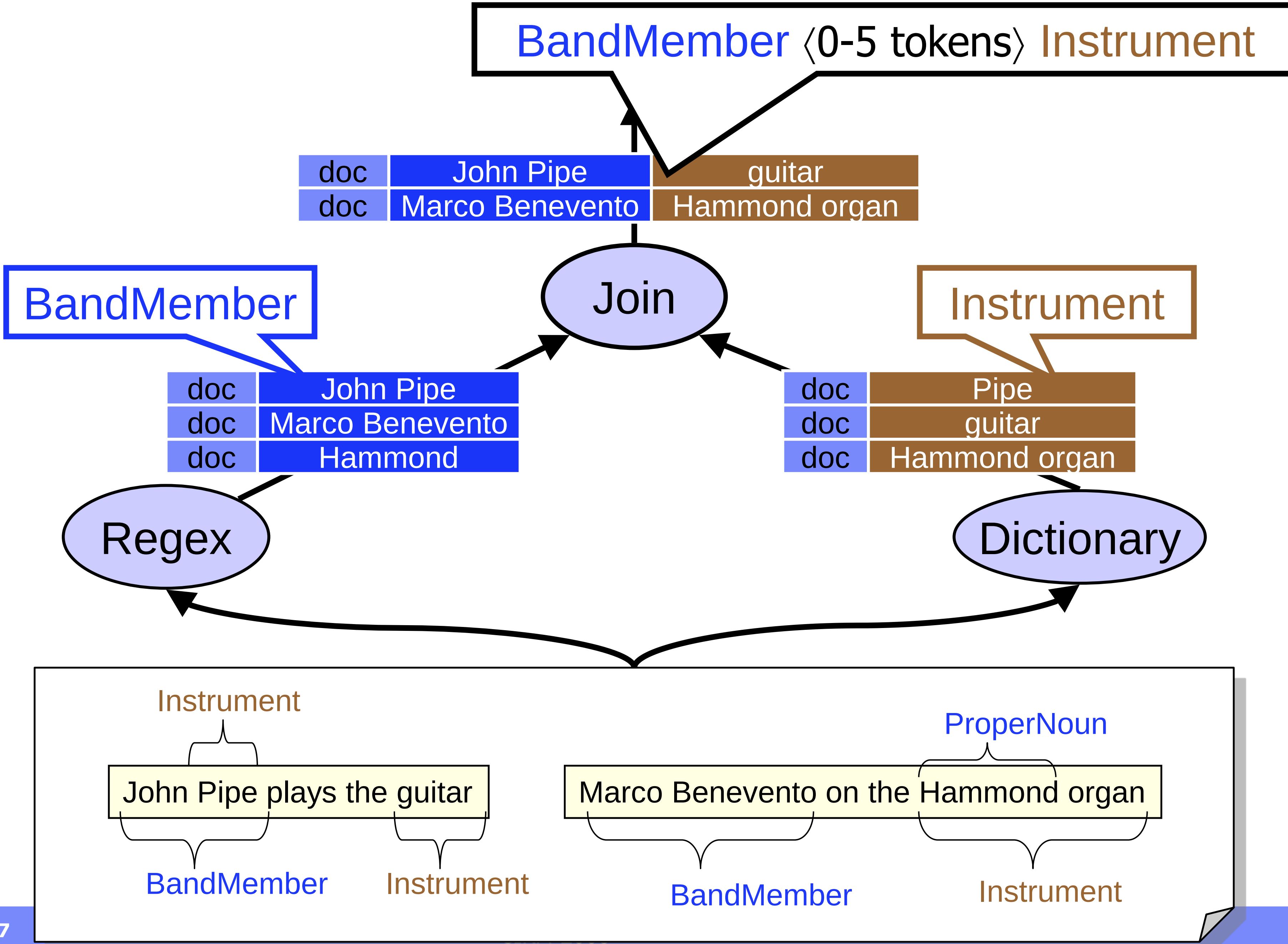
- **Each operator in the algebra…**
  - …operates on tuples of annotations
  - …produces tuples of annotations
- **Rich set of operators:**
  - Operators from relational algebra: select, project, join, …
  - Text related operators/predicates: regex, dictionary, span-based, …
- **Evaluation is restricted to within each document**
  - Algebra expression is defined over
    - text of the current document
    - existing annotations over the current document
  - Output is attached to the same document

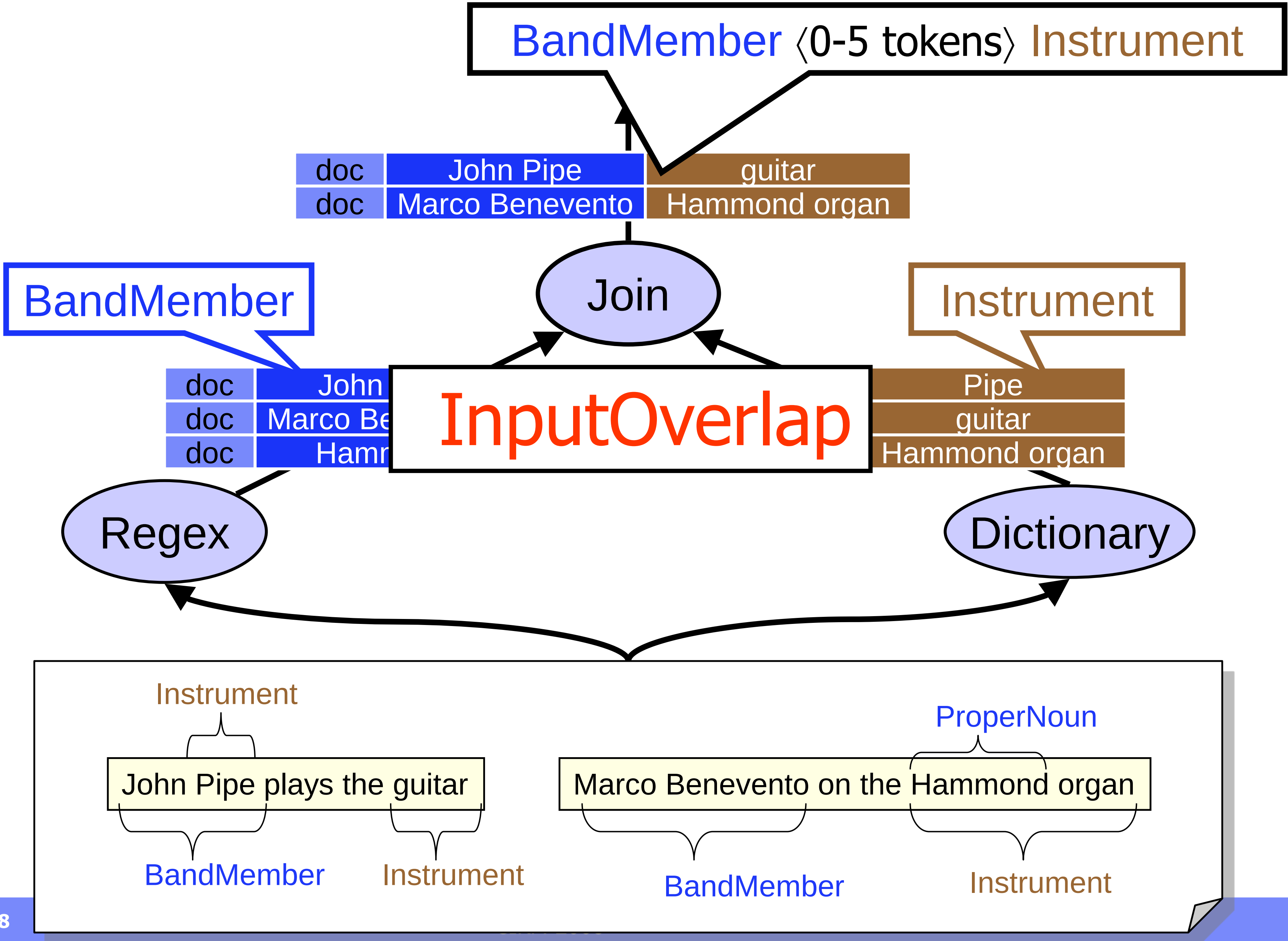
BandReviewInstance: InputOverlap

BandMember <0-5 tokens> Instrument

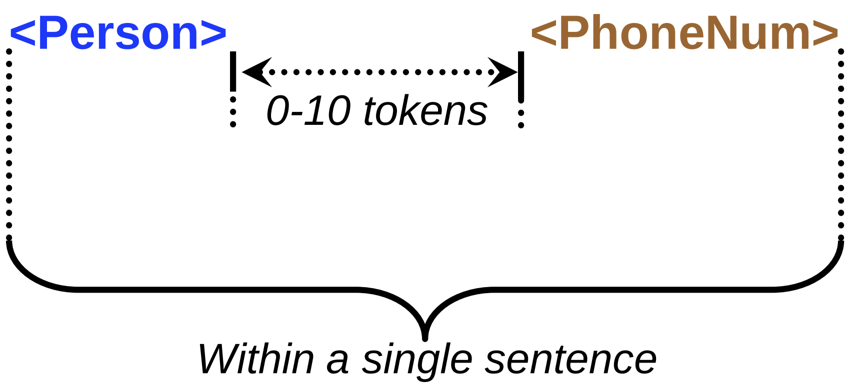
InputOverlap




BandMember ⟨0-5 tokens⟩ Instrument
doc | John Pipe | guitar
doc | Marco Benevento | Hammond organ
Join
BandMember
doc | John Pipe
doc | Marco Benevento
doc | Hammond
Instrument
doc | Pipe
doc | guitar
doc | Hammond organ
Regex
Dictionary
Instrument
John Pipe plays the guitar
BandMember
Instrument
ProperNoun
Marco Benevento on the Hammond organ
BandMember
Instrument

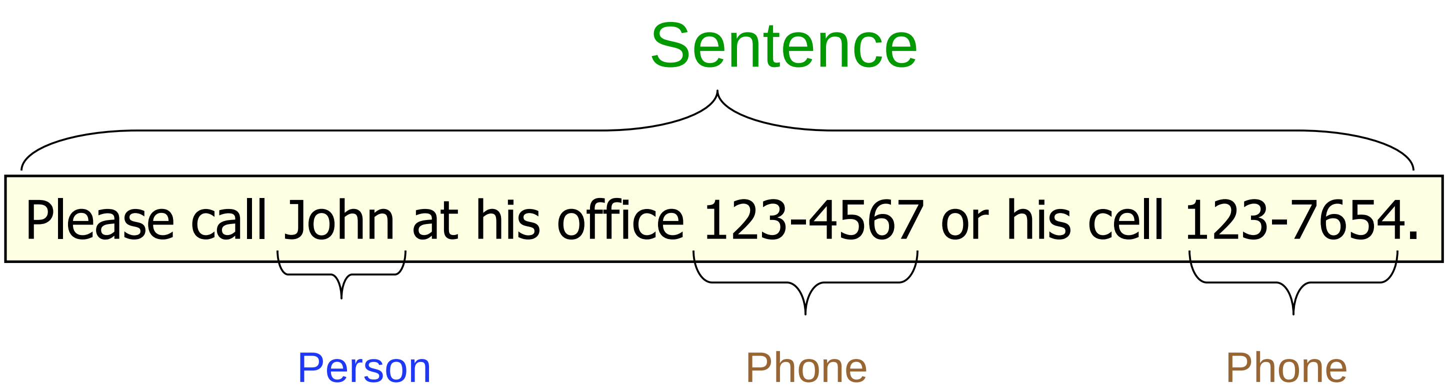

BandMember ⟨0-5 tokens⟩ Instrument
doc | John Pipe | guitar
doc | Marco Benevento | Hammond organ
Join
BandMember
Instrument
doc | John
doc | Marco Be
doc | Hamr
InputOverlap
Pipe
guitar
Hammond organ
Regex
Dictionary
Instrument
ProperNoun
John Pipe plays the guitar
Marco Benevento on the Hammond organ
BandMember
Instrument
BandMember
Instrument

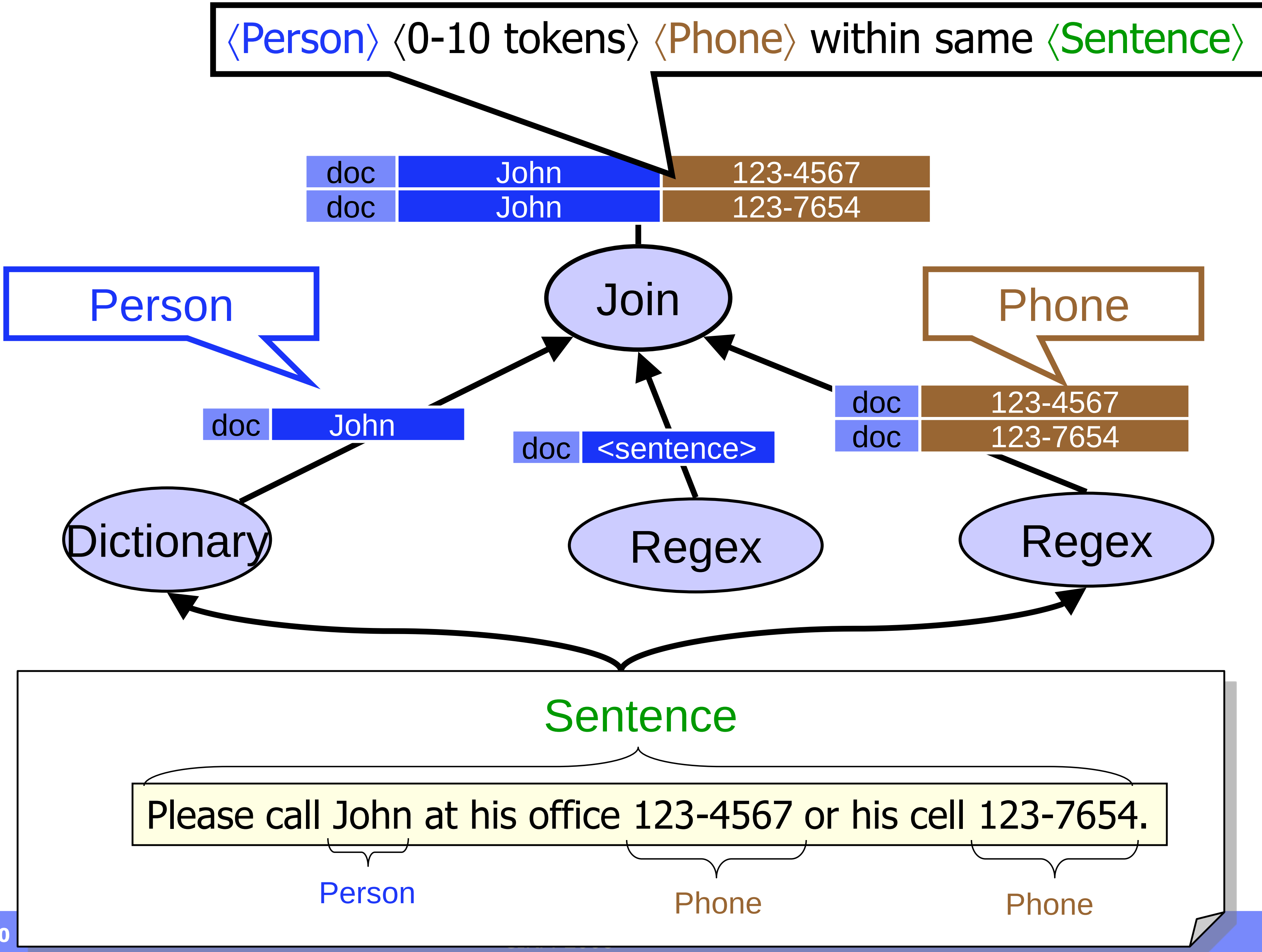
PersonsPhone: SpanPredicate, OutputOverlap

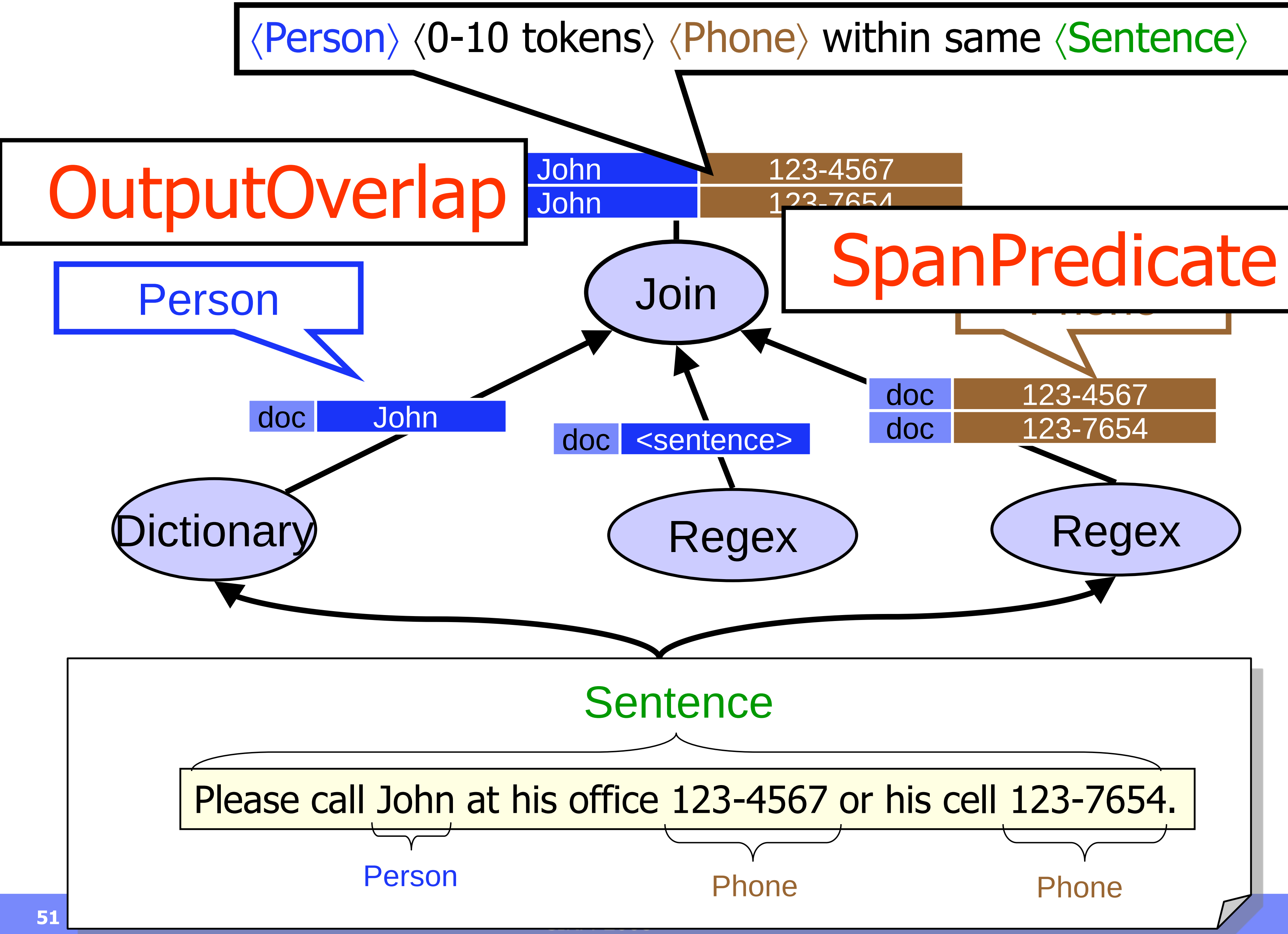


SpanPredicate

OutputOverlap

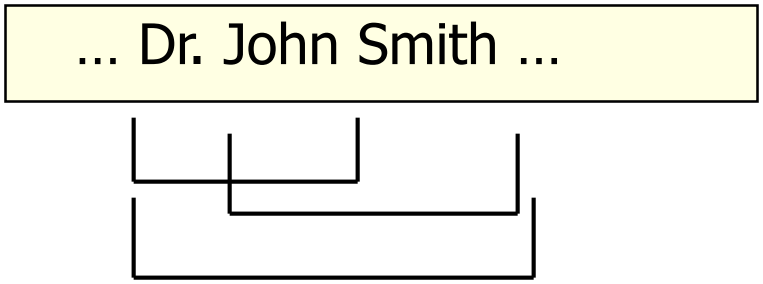



⟨Person⟩ ⟨0-10 tokens⟩ ⟨Phone⟩ within same ⟨Sentence⟩
doc John 123-4567
doc John 123-7654
Join
Person
Phone
doc John
doc <sentence>
doc 123-4567
doc 123-7654
Dictionary
Regex
Regex
Sentence
Please call John at his office 123-4567 or his cell 123-7654.
Person
Phone
Phone


⟨Person⟩ ⟨0-10 tokens⟩ ⟨Phone⟩ within same ⟨Sentence⟩
OutputOverlap
John 123-4567
John 123-7654
SpanPredicate
Join
Person
doc John
doc <sentence>
doc 123-4567
doc 123-7654
Dictionary
Regex
Regex
Sentence
Please call John at his office 123-4567 or his cell 123-7654.
Person
Phone
Phone

Person: Consolidation

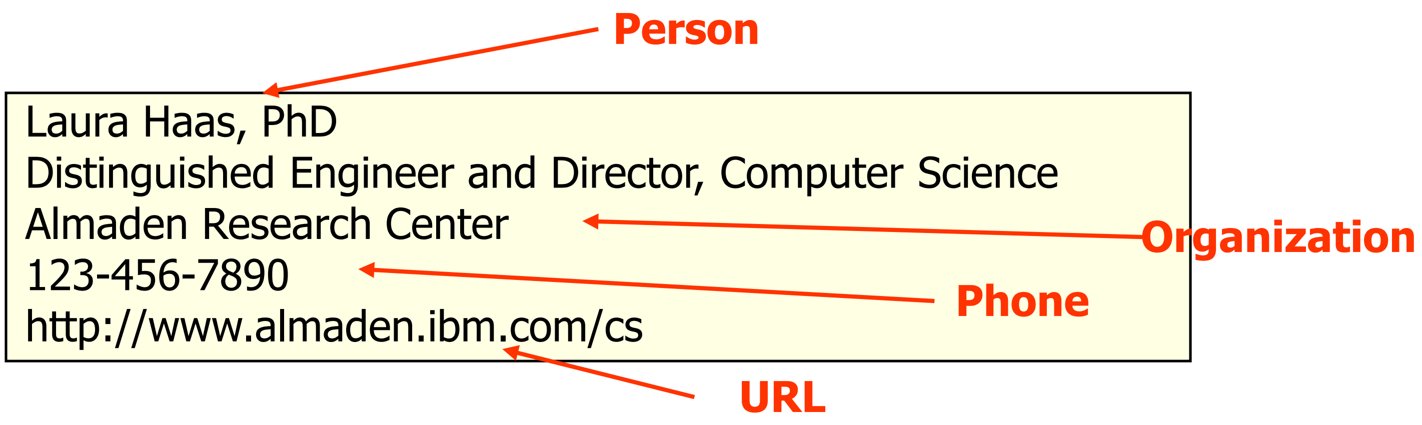
⟨PersonDict⟩ ⟨PersonDict⟩ → ⟨Person⟩
⟨Salutation⟩ ⟨CapsWord⟩ → ⟨Person⟩
⟨Salutation⟩ ⟨CapsWord⟩ ⟨CapsWord⟩ → ⟨Person⟩

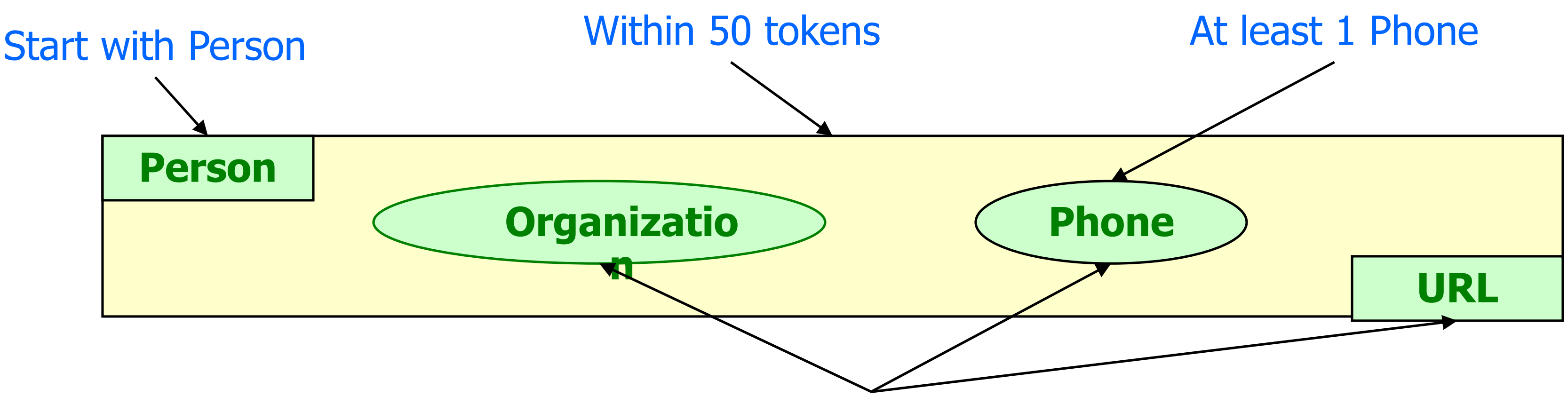


Desired Output: Dr. John Smith

- **Classical grammar world**
  - Anticipate all possible rule interactions and control through rule priority
  - Becomes unmanageable as number of rules run into the hundreds
- **SystemT approach**
  - Only need to think about possible overlap scenarios
  - Use appropriate consolidation operators
    - Some out-of-the-box, others can be added easily

Signature: WindowCount





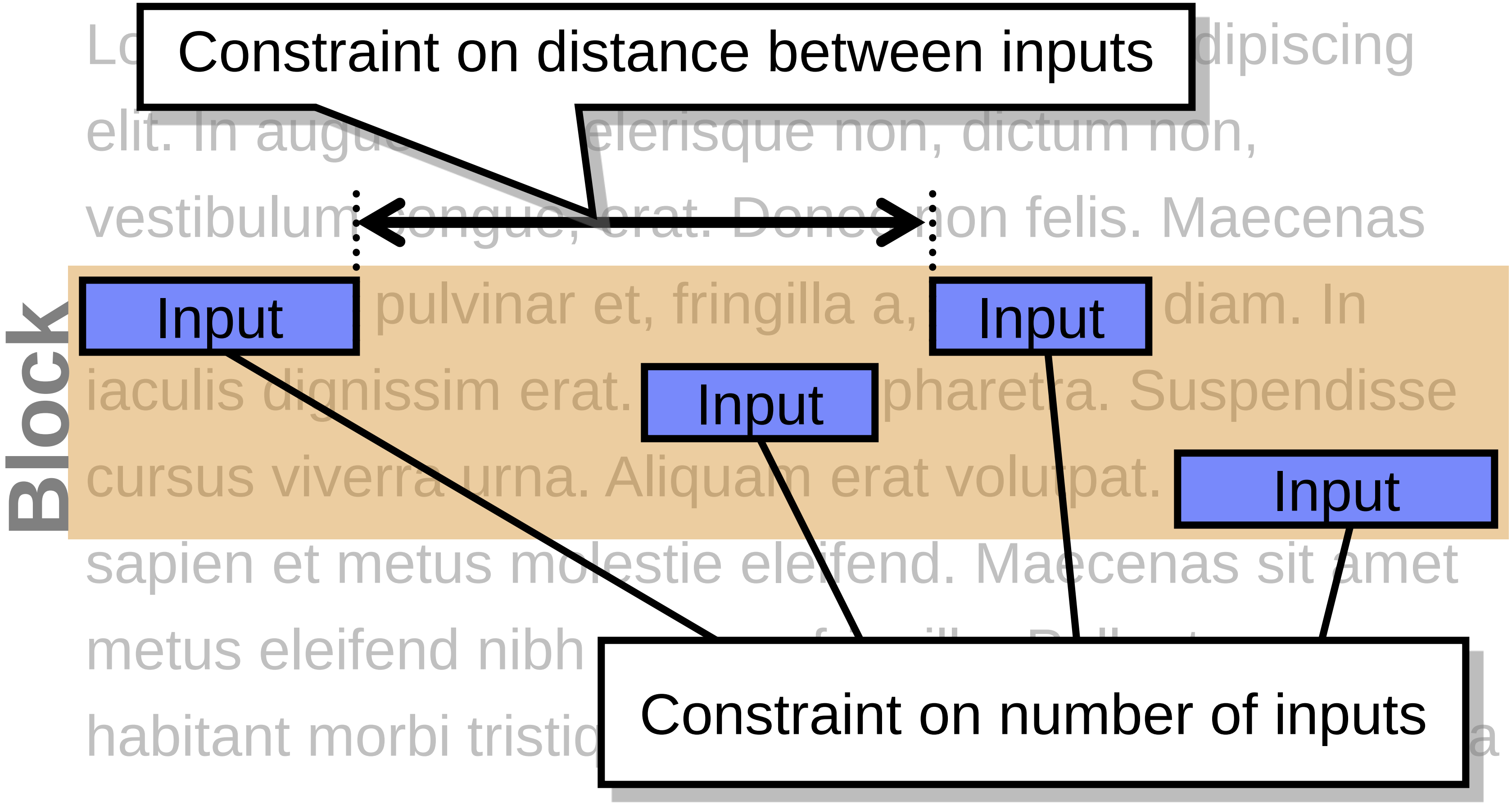
At least 2 of {Phone, Organization, URL, Email, Address}

End with one of these.

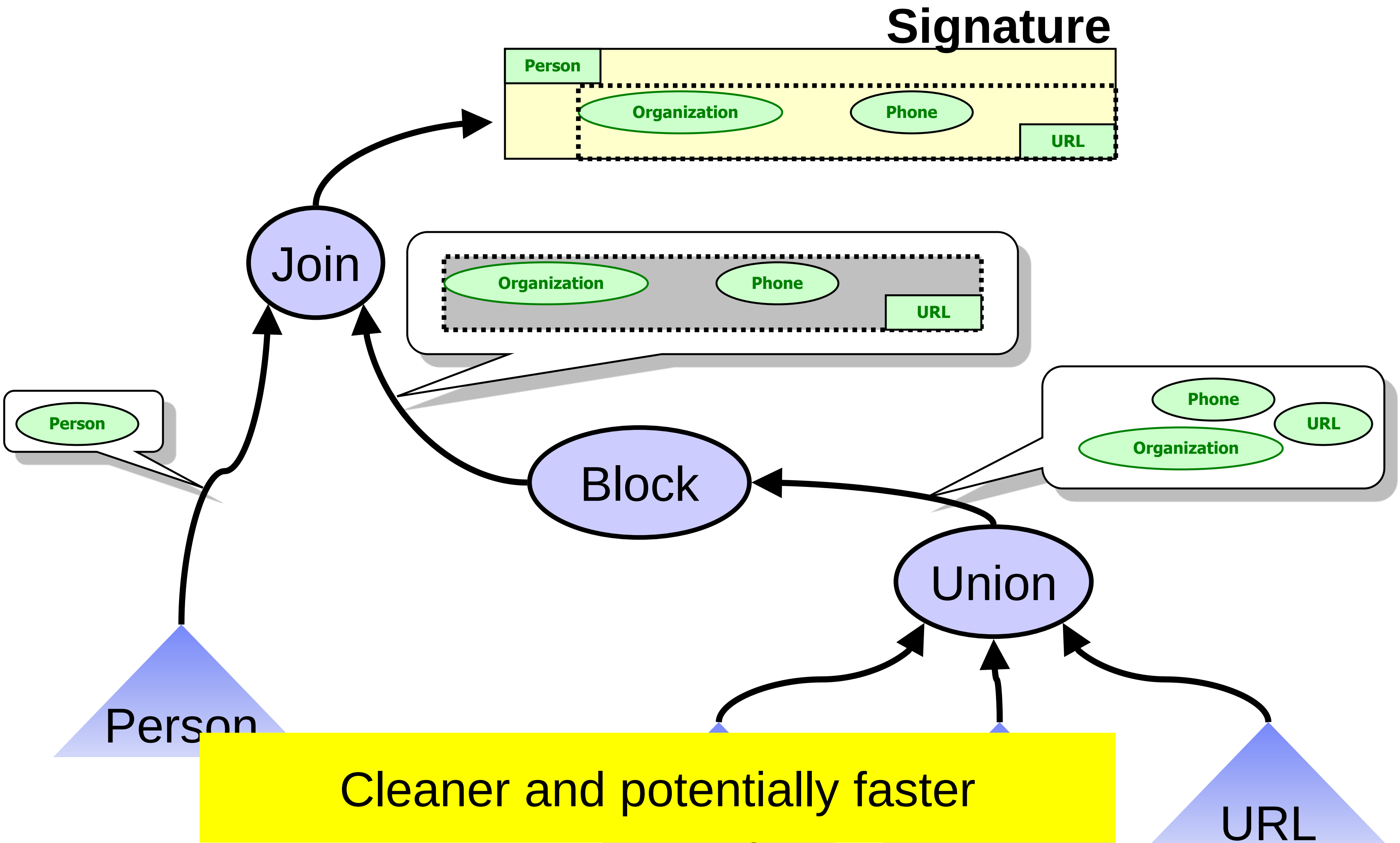
Block Operator (β)

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In augue mi, scelerisque non, dictum non, vestibulum congue, erat. Donec non felis. Maecenas Input pulvinar et, fringilla a, Input diam. In iaculis dignissim erat. Input pharetra. Suspendisse cursus viverra urna. Aliquam erat volutpat. Input sapien et metus molestie eleifend. Maecenas sit amet metus eleifend nibh semper fringilla. Pellentesque habitant morbi tristique senectus et netus et malesuada

Block Operator (β)



Back to signature

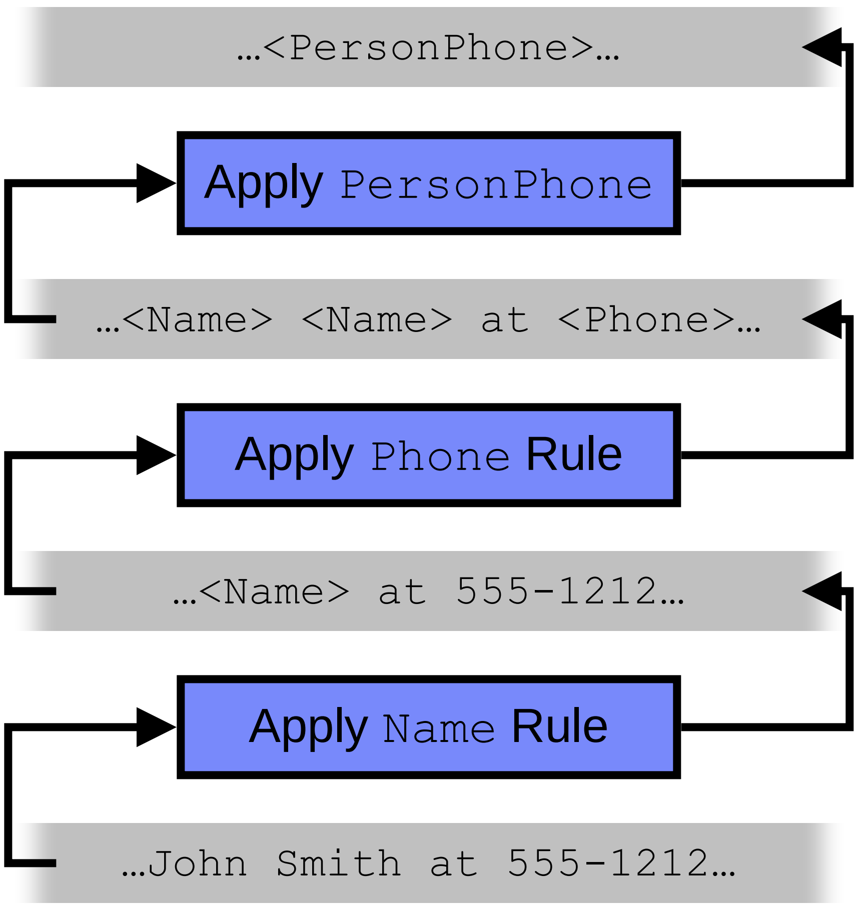


Cleaner and potentially faster

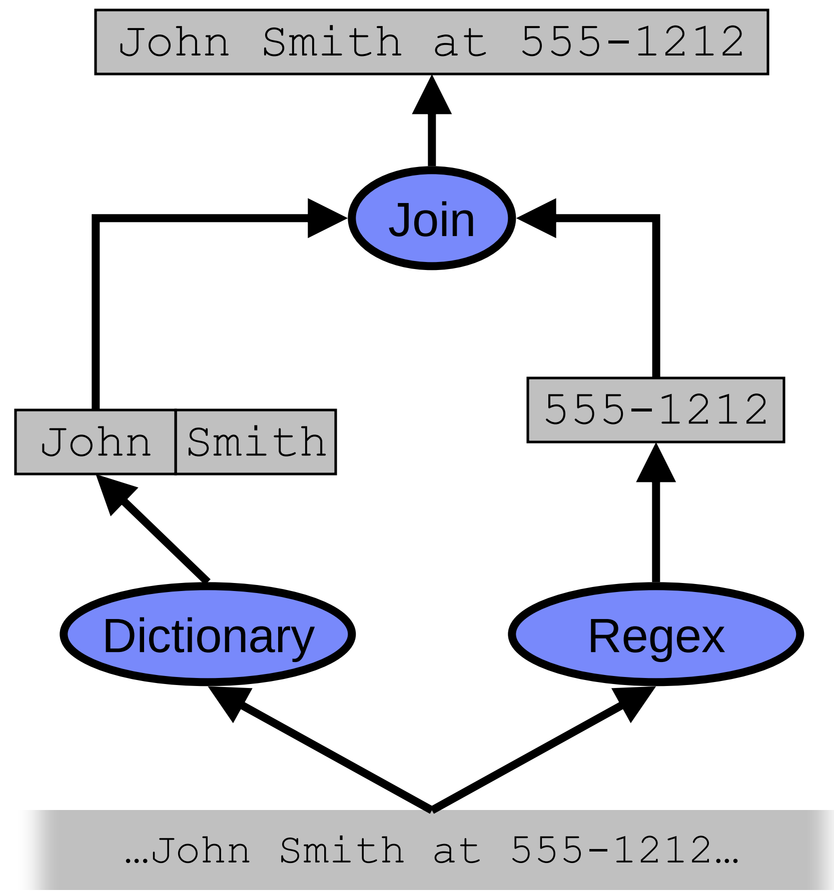
Solutions: Roadmap

- **We have seen how expressivity problems are addressed**
- **On to performance problems**

PersonPhone: Performance

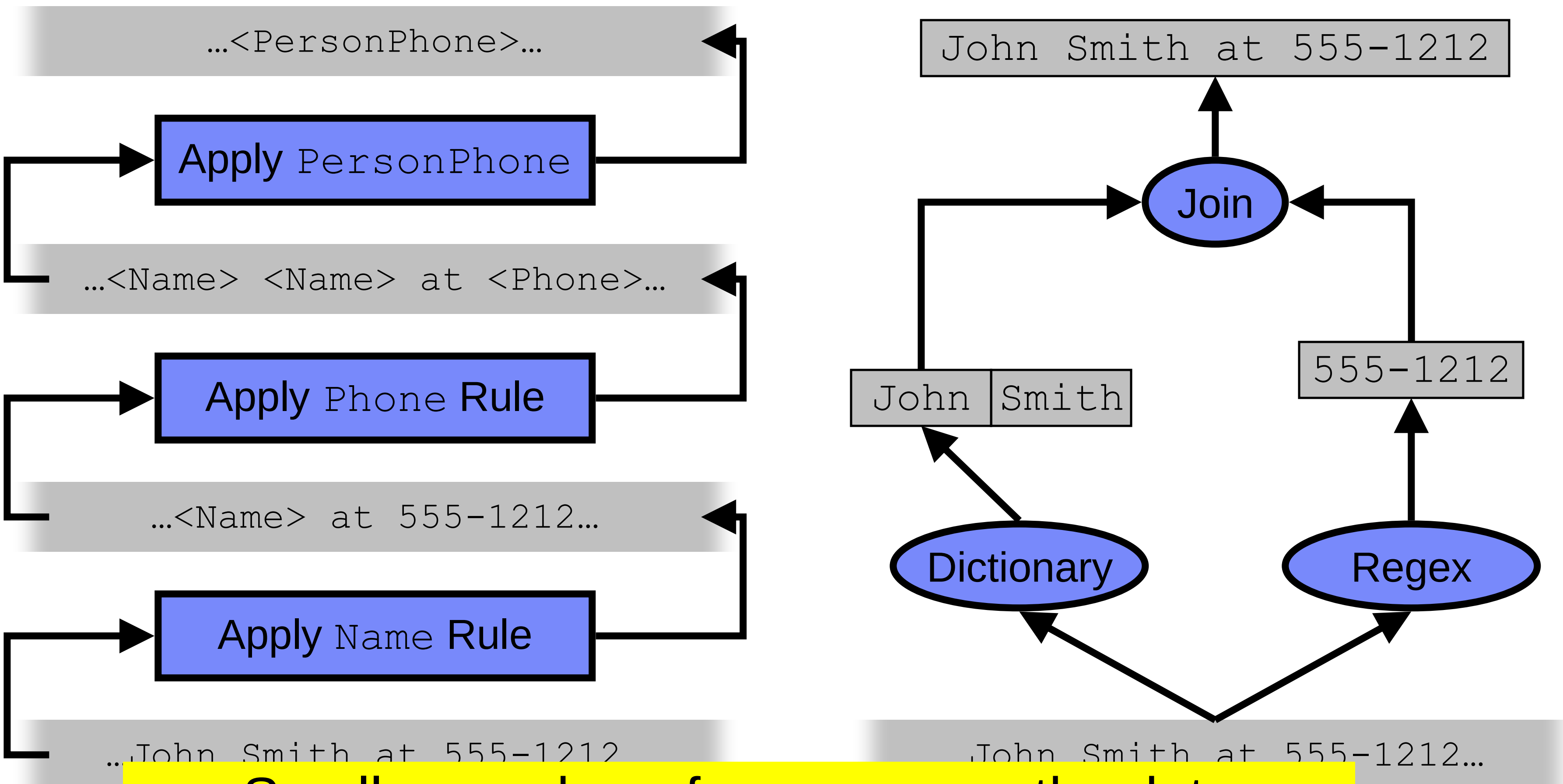


Grammar



Algebra

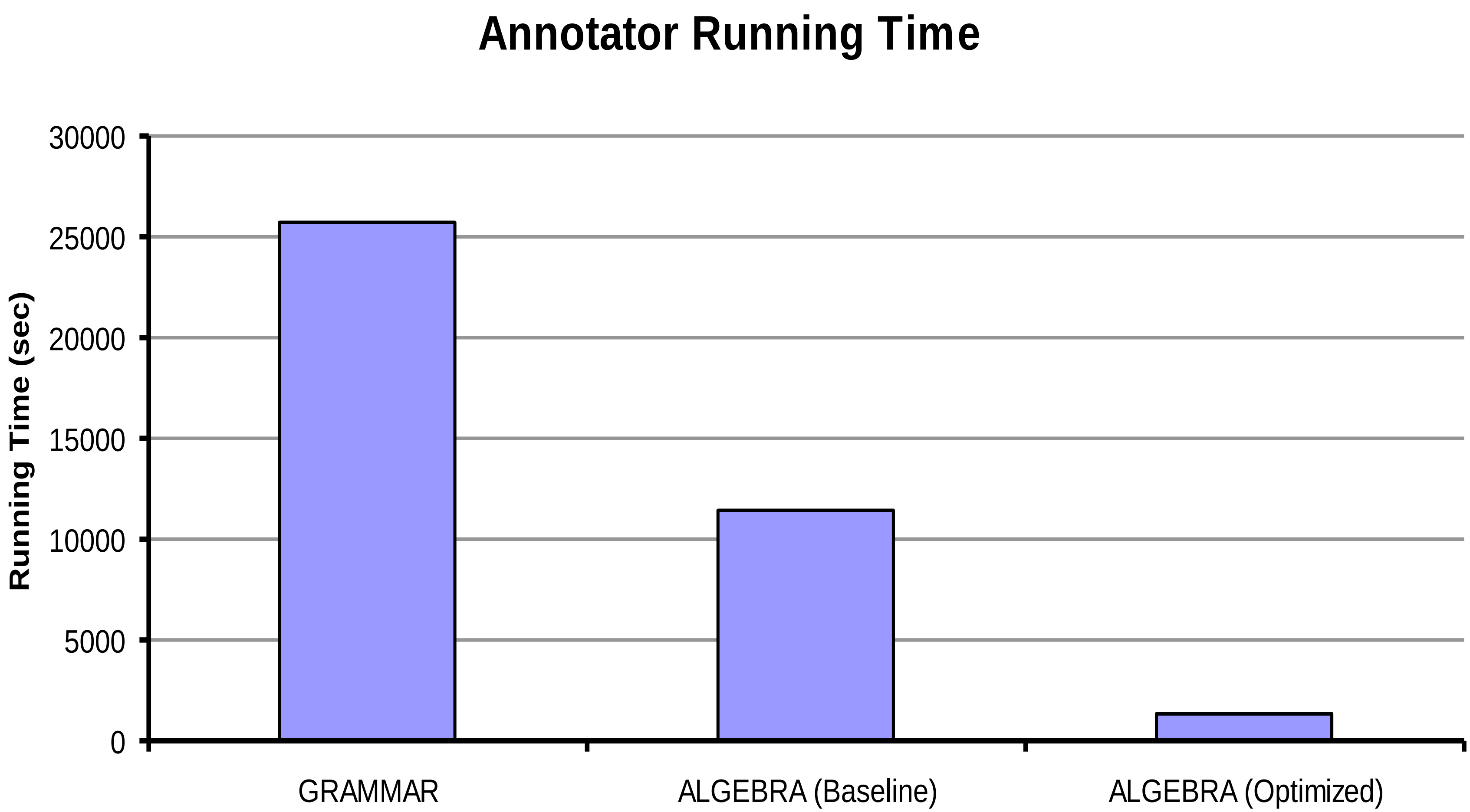
PersonPhone: Performance



Smaller number of passes over the data

Many other optimizations possible.

Experimental Results



PART 3

Roadmap

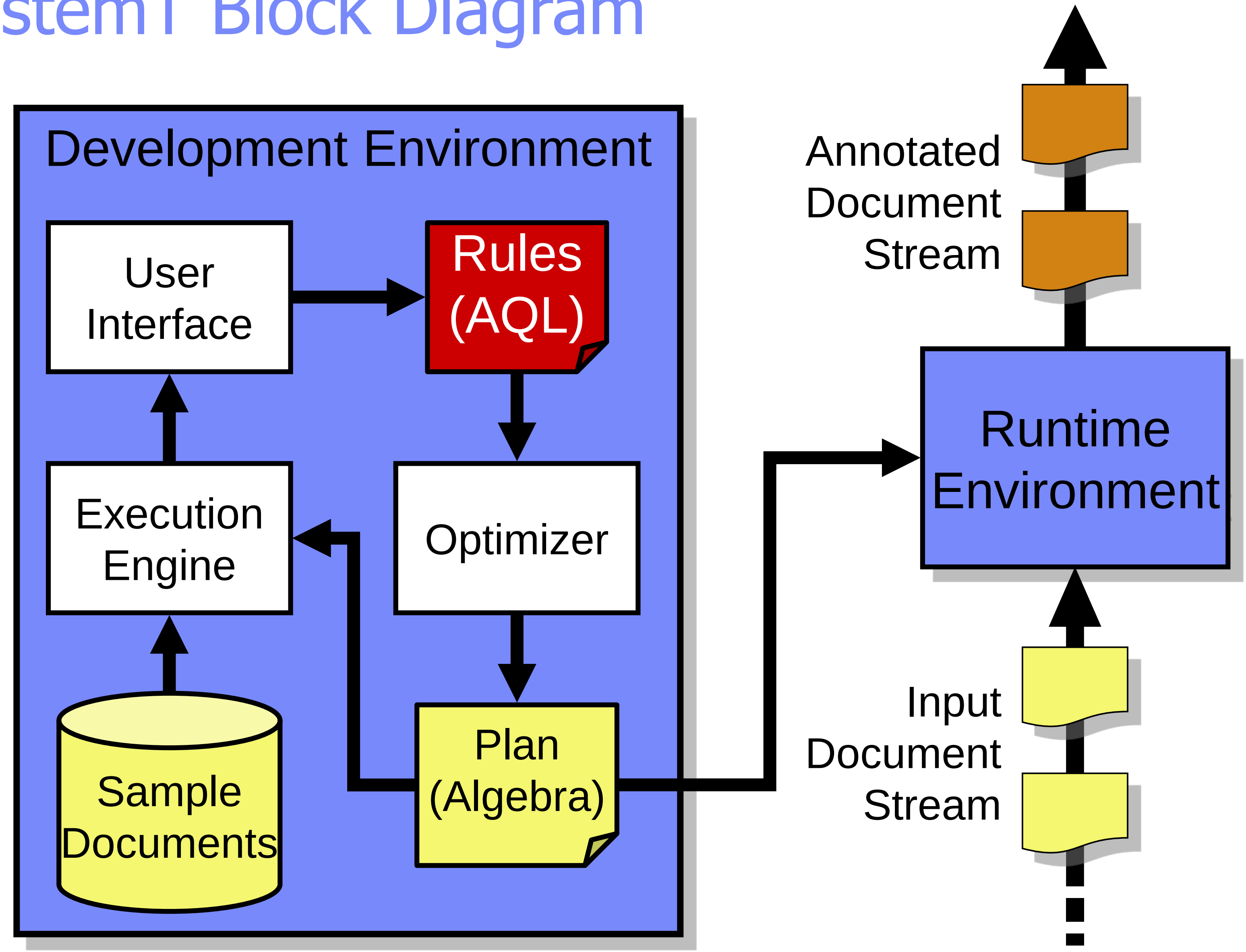
- Part 1 [Sriram Raghavan]
  - Grammar-based extraction systems
  - Newer motivating applications
  - Limitations of grammar-based extraction
- Part 2 [Huaiyu Zhu]
  - Extended grammar-based solutions
  - Modern declarative approaches
- **Part 3 [Rajasekar Krishnamurthy]**
  - SystemT in-depth
  - Research directions

Overview

System T

- **Next-generation information extraction system**
- **Makes developing annotators like developing other enterprise software**
  - AQL rule language
    - Declarative language for building annotators
  - Development environment
    - Provides support for building complex annotators
  - Runtime environment
    - Deploy to corporate PCs or server farms

SystemT Block Diagram



SystemT in-depth: Roadmap

- **Data Model and Algebra**
- **Annotation Query Language (AQL)**
- **Optimization**

Data Model

Document
text: STRING

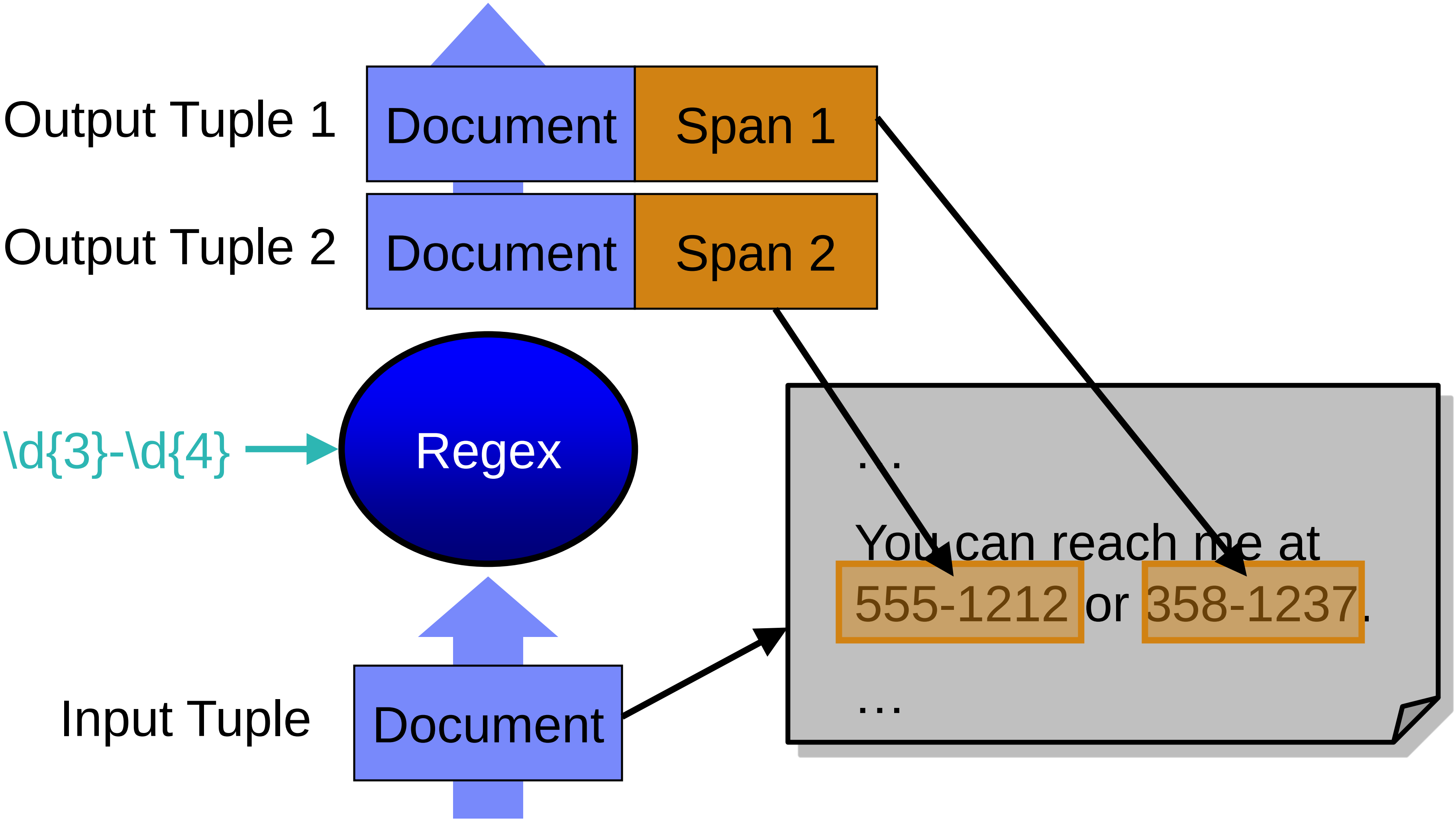
Annotation		
begin: INT	*end*: INT	*doc*: DOC

- Document consists of a text attribute
- Annotations are represented by a type called Span, which consists of begin, end and document attribute

Algebra for Intra-document IE

- **Each Operator in the algebra**
  - operates on one or more tuples of annotations
  - produces tuples of annotations
- **"Document at a time" execution model**
  - Algebra expression is defined over
    - the current document
    - annotations defined over current document
  - Algebra expression is evaluated over each document in the corpus individually

Example: Regular Expression Extraction Operator



Operators in the Algebra

Three main classes of operators

- **Relational operators**
  - Selection, Cross product, Join, Union, …
- **Span extraction operators**
  - Regular expression, Dictionary
- **Span aggregation operators**
  - Consolidation, Block

Recall ReviewInstance pattern from before

- **Sample snippets**
  - Kurt Ralske played guitar
  - John Pipe plays the guitar
  - Marco Benevento on the Hammond organ

<BandMember> Regular Expression Match

<within 5 tokens>

<Instrument> Dictionary Match

<[A-Z]\w+(\s[A-Z]\w+)?>

<d_1|d_2|…d_n>

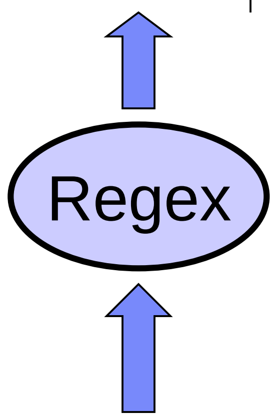
Span Extraction operators

- **Standard Regular Expression Matcher**
  - identifies all non-overlapping matches when given regular expression is evaluated from left-to-right over the input text
- **Dictionary Matcher**
  - finds all occurrences in the input text for each word/phrase in given dictionary
- **Token-bound Regular Expression Matcher**
  - identifies the longest match (of length within given bound) when given regular expression is evaluated from the beginning of every token in the input text

Dictionary and Token-bound Regular Expression Matcher may return matches with overlapping spans

BandMember (Regular expression)

1	Kurt … guitar	Kurt Ralske
2	John …. guitar	John Pipe
3	Marco … organ	Marco Benevento
4	Marco … organ	Hammond

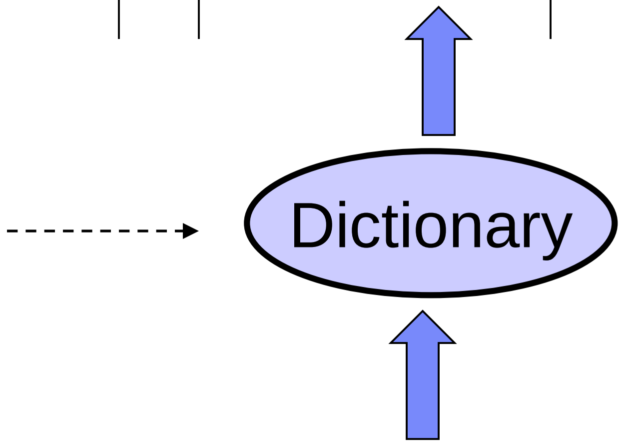


[A-Z]\w+(\s[A-Z]\w+)?

1	Kurt Ralske played guitar
2	John Pipe plays the guitar
3	Marco Benevento on the Hammond organ

Instrument (Dictionary)

1	Kurt … guitar	guitar
2	John …. guitar	guitar
3	Marco … organ	Hammond organ



Instrument Dictionary

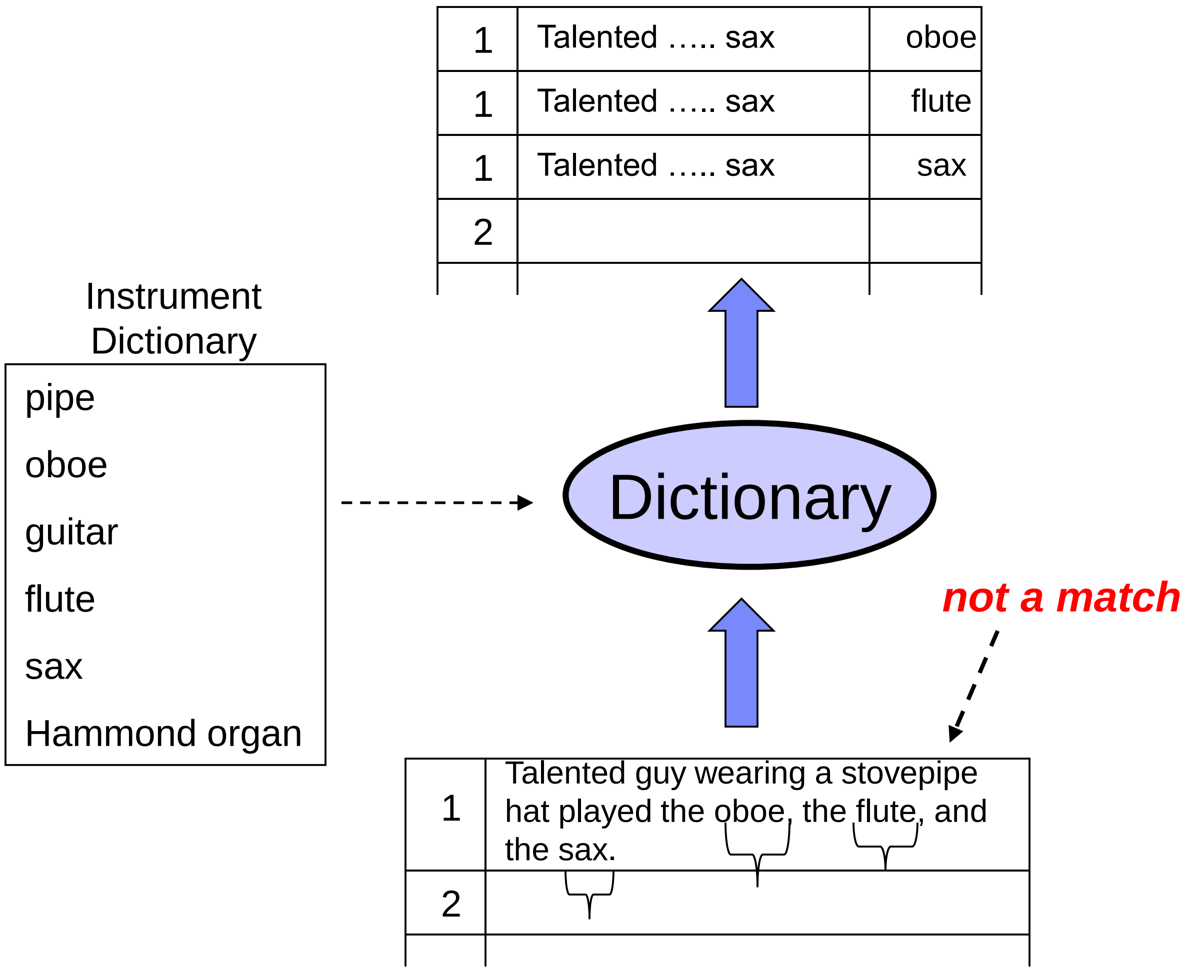
- pipe
- oboe
- guitar
- flute
- sax
- Hammond organ

1	Kurt Ralske played guitar
2	John Pipe plays the guitar
3	Marco Benevento on the Hammond organ

Is the Dictionary operator redundant?

- **It may seem that a dictionary can be written as a regular expression**
  - (pipe | oboe | …| hammond organ)
- **However,**
  - Matches in the dictionary are expected only at token boundaries
  - Disjunctions in regular expressions are short-circuited
  - Dictionary operator returns all matches whereas regular expression operator returns non-overlapping matches
  - Performance could be a problem as regular expressions are not tuned to handle very large disjunctions

Dictionary matches only at token boundaries



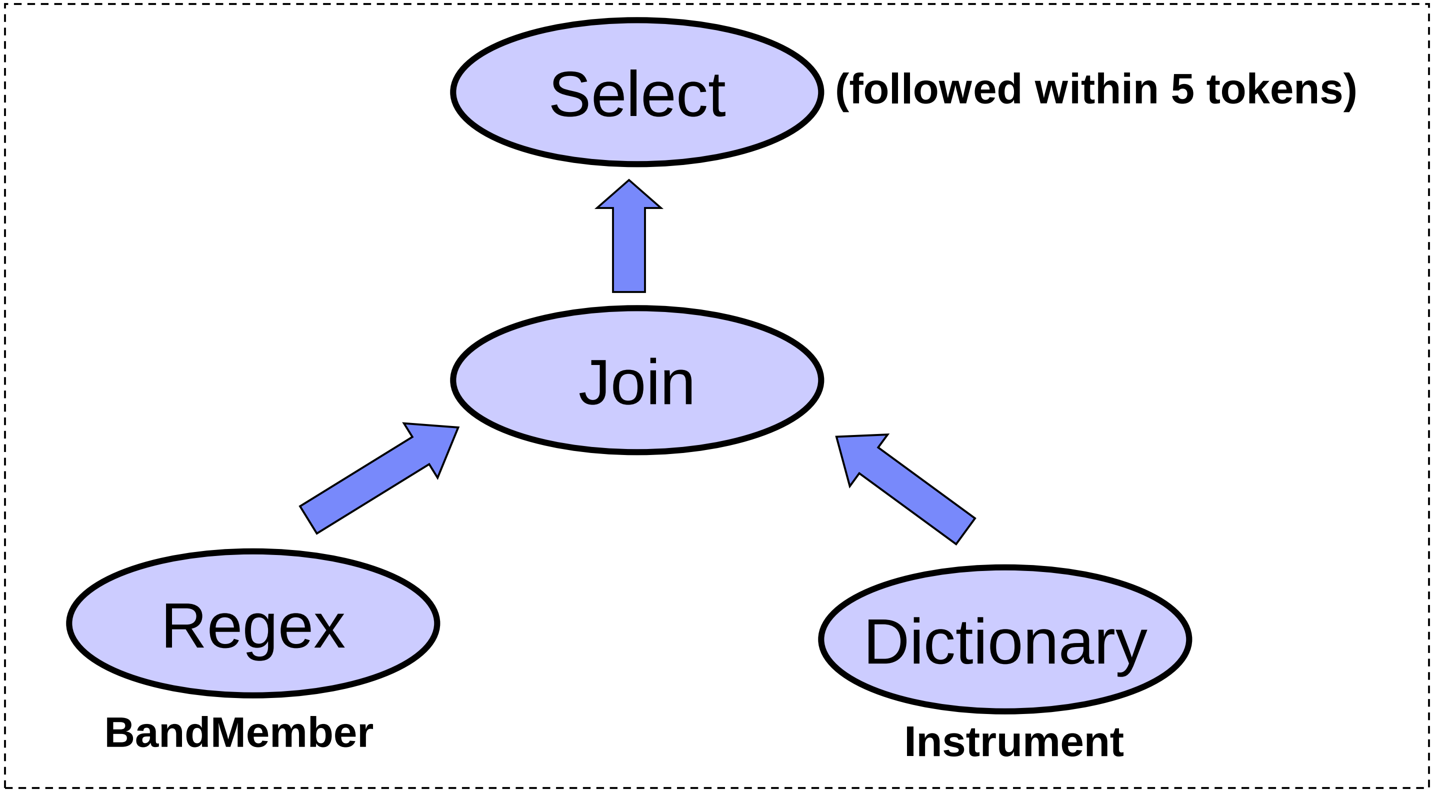
Problem with Disjunctions in regular expressions

- For the text "The talented guy played the pipe organ" the two regular expressions
  - (pipe | pipe organ)
  - (pipe organ | pipe)

  will return different results due to the short-circuiting semantics of regular expressions.
- Rewriting dictionaries as regular expressions is non-trivial if entries in the dictionary can match overlapping regions of text

An example ReviewInstance Rule

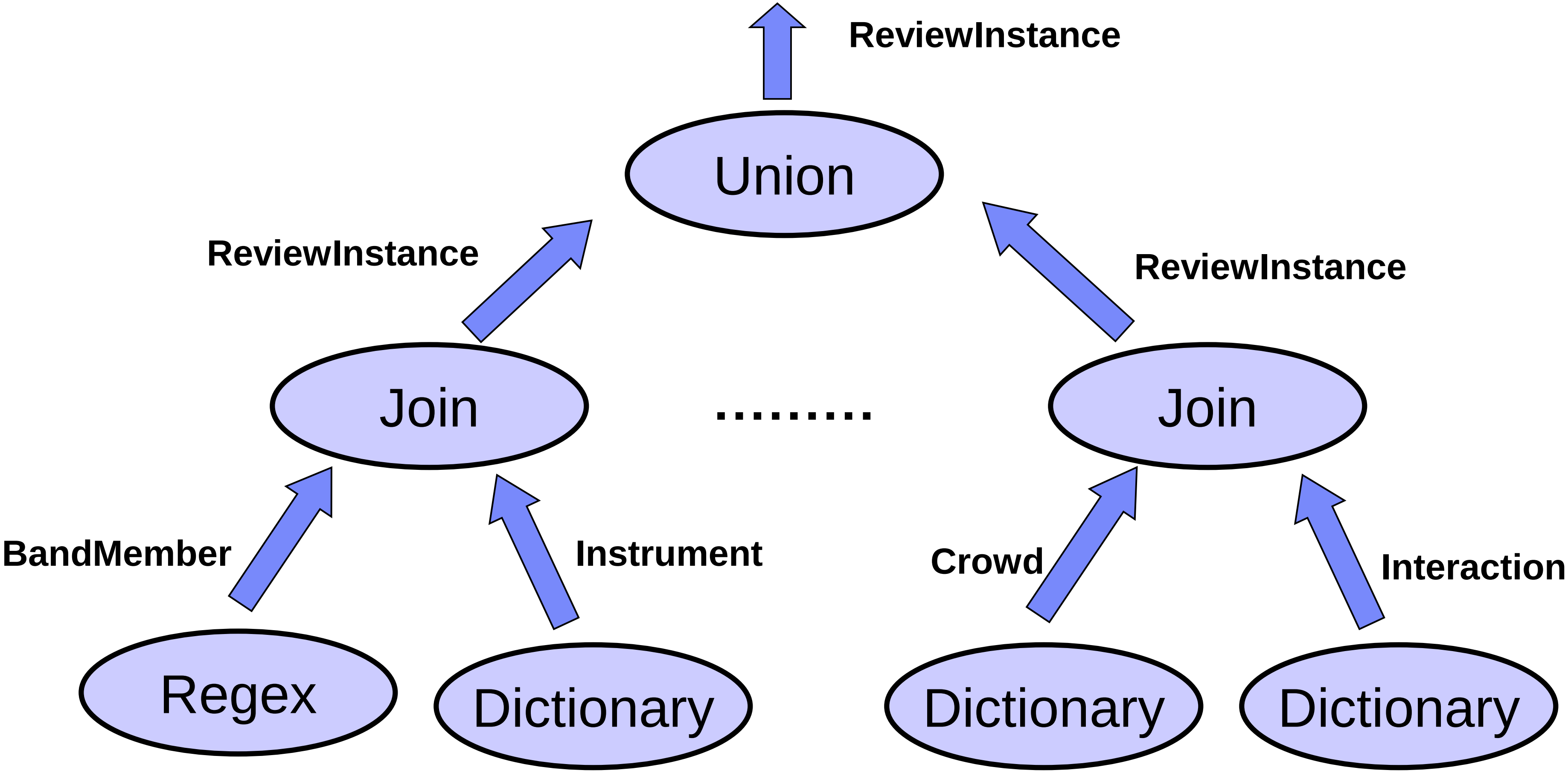
<BandMember> Regular Expression Match **<within 5 tokens>** **<Instrument>** Dictionary Match



Span predicates

Predicate	Explanation
s_1 ☸$_d$ s_2	s_1 and s_2 do not overlap, s_1 precedes s_2 and there are at most d characters between the end of s_1 and the beginning of s_2
s_1 ℯ𝓇 s_2	The spans overlap
s_1 ♉ s_2	s_1 is strictly contained within s_2
$s_1 = s_2$	Spans are identical

Putting multiple ReviewInstance rules together



Outline of the BandReview Annotator

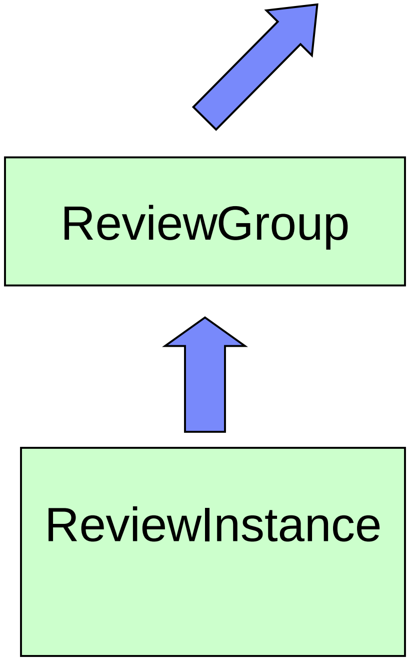
"**Lead singer/guitarist was really good**, and even … **I actually liked the opening bands**. … Well they were none of those. I especially **liked the first band**"

"lead singer/guitarist was really good"

"Liked the opening bands"

"Liked the first band"

"Kurt Ralske played guitar"



Span Aggregation Operators

- Support aggregation over a set of input spans
- Two such operators in SystemT
  - Block operator
  - Consolidation operator

Block Operator

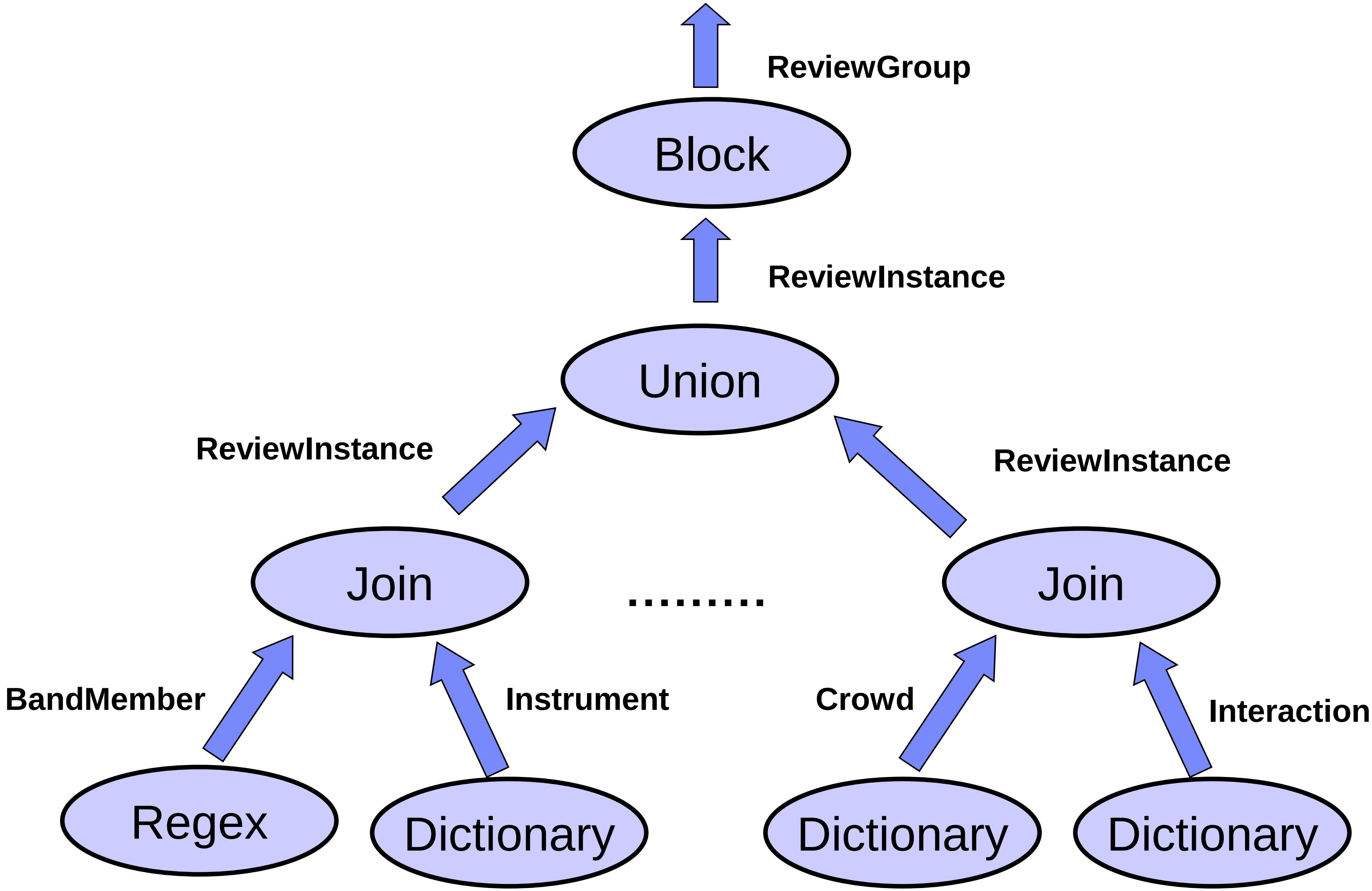
- Identify regions of text where the input appears frequently
- **Input :**
  - Input annotations I
  - Separation constraint d
  - Length constraint l
- **Output :**
  - All Spans s in the text where
    - s contains at least l non-overlapping annotations from I
    - Successive annotations in s are at most d distance apart

"**Lead singer/guitarist was really good**, and even … **I actually liked the opening bands**. … Well they were none of those. I especially **liked the first band**"

Block Operator

- **Block(l >= 2, d <= 50)** over the text below will return 3 results
  - Lead singer … first band
  - Lead singer … opening bands
  - I actually … first band
- Note how all possible matches to the operator definition are returned

"**Lead singer/guitarist was really good**, and even … **I actually liked the opening bands**. … Well they were none of those. I especially **liked the first band**"


ReviewGroup
Block
ReviewInstance
Union
ReviewInstance
ReviewInstance
Join
....
Join
BandMember
Instrument
Crowd
Interaction
Regex
Dictionary
Dictionary
Dictionary

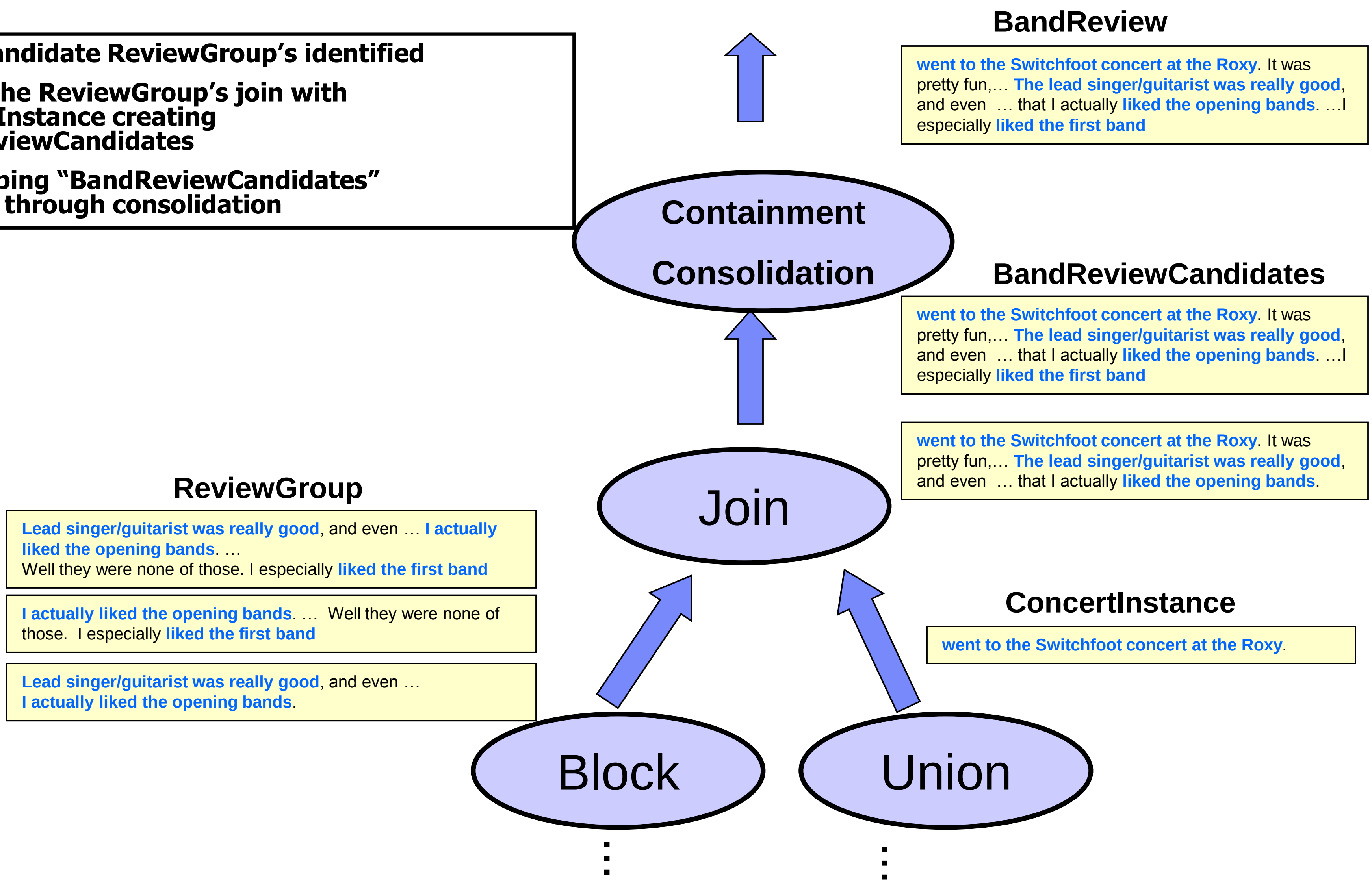
Consolidation operator

- **To handle overlapping matches produced by**
  - Multiple extraction patterns specified for the same concept
    - E.g., multiple rules for ReviewInstance may identify different portions of the same text
  - Other operators in the algebra such as Block, Join
- **Containment Consolidation**
  - Output only those spans in the input that are not contained within another
- **LeftToRight Consolidation**
  - Emulates the overlap handling policy used in standard regular expression engines

Three candidate ReviewGroup's identified

Two of the ReviewGroup's join with ConcertInstance creating BandReviewCandidates

Overlapping "BandReviewCandidates" handled through consolidation



Flexibility to generate and retain overlapping annotations at the lower levels of extraction. Use consolidation to discard "duplicates" at higher levels.

SystemT Algebra Summary

- **Current algebra has three main classes of operators**
  - Relational operators
    - Selection, Cross product, Join, Union, …
  - Span extraction operators
    - Regular expression, Dictionary
  - Span aggregation operators
    - Consolidation, Block
- **What is not supported currently**
  - Set valued attributes
    - will be added soon
  - Regular expressions over annotations
    - limited support : added as required
    - Block is an example

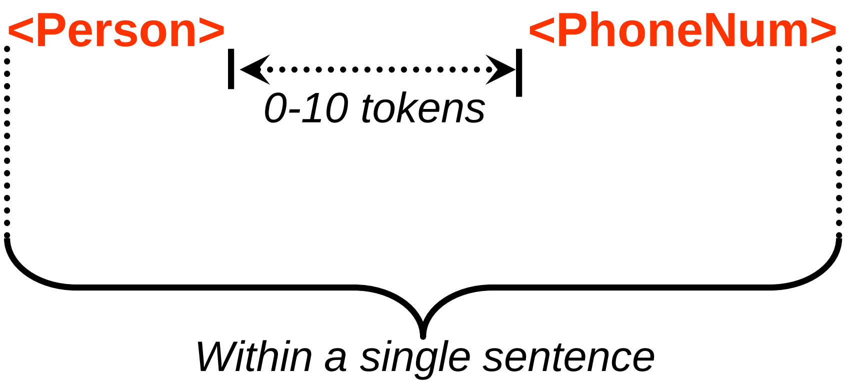
SystemT in-depth: Roadmap

- **Data Model and Algebra**
- **Annotation Query Language (AQL)**
- **Optimization**

AQL

- **Declarative language for defining annotators**
  - Compiles into our algebra
- **Main features**
  - Separates semantics from performance
  - Familiar syntax
  - Full expressive power of algebra

AQL By Example : PersonsPhone




```
create view PersonPhone as
select P.name as person, N.number as phone
from Person P, PhoneNumber N, Sentence S
where
    FollowsTok(P.name. N.number, 0, 10)
    and Contains(S.sentence, P.name)
    and Contains(S.sentence, N.number);
```

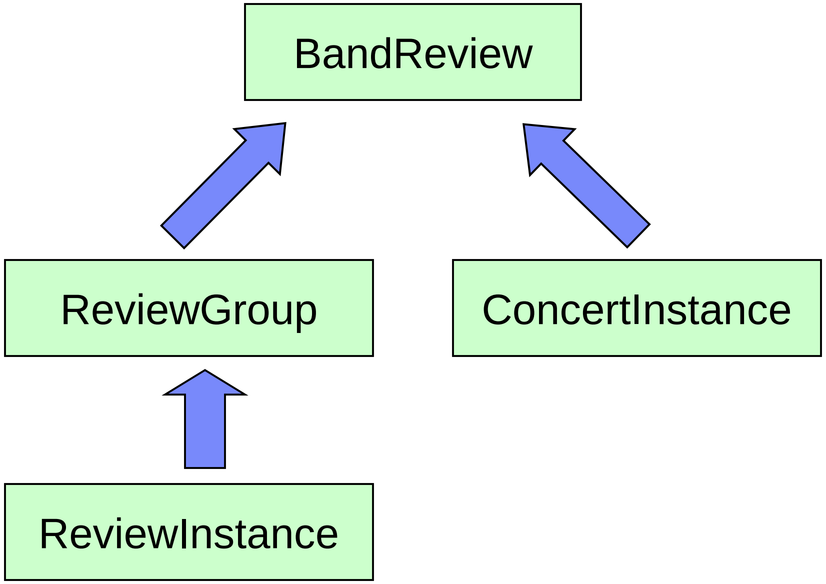
AQL By Example : ConcertReview


```
-- Define a dictionary of instrument names
create dictionary Instrument as ( ' flute ' , ' guitar ' , ... );

-- Use a regular expression to find names of band members
create view BandMember as
extract regex /[A−Z]\w+(\s+[A−Z]\w+)/
on 1 to 3 tokens of D.text
as name
from Document D;

-- A single ReviewInstance rule . Finds instances of
-- BandMember followed within 30 characters by an
-- instrument name.
create view ReviewInstance as
select CombineSpans(B.name, I.inst) as instance
from BandMember B,
    (extract dictionary ' Instrument' on D.text as inst
     from Document D) I
where FollowsTok(B.name, I . inst , 0, 5)
consolidate on CombineSpans(B.name, I.inst);
```

AQL By Example : BandReview




```
create view ReviewGroup as
 extract blocks
                  with length between 3 and 10
                  and separation between 0 and 100 characters
                  on I . instance as instblock
from ReviewInstance I;

create view BandReview as
select  CI . instance as concert ,
         CombineSpans(CI.instance, RG. instblock ) as review
from             ConcertInstance CI, ReviewGroup RG
where            Follows (CI. instance , RG. instblock , 0, 30)
consolidate on CombineSpans(CI.instance, RG.instblock )
using 'ContainedWithin';
```

ReviewGroup :	Block of 3 to 10 Review instances,
	Successive instances occur within 100 characters
BandReview :	ConcertInstance and ReviewGroup within 30 characters
	Handle overlapping bandreviews by removing any match completely contained within another match

AQL Demo : Simplified Phone Annotator

DEMO

- **Iteration 1 :** Identify 10 digit phone numbers

```
create view USPhone as
extract
  regex  /\(\d{3}\)[\- ]?\d{3}[\-\. ]?\d{4}/
  on D.text
  as match
from Document D;
```

Identifies correct instances such as

- Phone: (202) 466-9176
- please call the GISB office at (713) 356-0060

Also identifies incorrect instances

- Fax : (202) 331-4717

AQL Demo : Simplified Phone Annotator

DEMO

- **Iteration 2 :** Predicate to remove fax numbers

```
create view USPhone as
extract
  regex  /\(\d{3}\)[\- ]?\d{3}[\-\. ]?\d{4}/
  on D.text
  as match
from Document
-- phrase fax does not appear in the left context
having   Not(ContainsRegex( /[Ff][Aa][Xx][^\r\n]+$/ ,LeftContext(match,20)));
```

AQL Demo: Simplified Person Annotator

DEMO

- **Iteration 1** : Start with a single rule
  - <FirstName> <LastName>
- **Iteration 2** : Add two more rules
  - Rule R1 : <FirstName> <LastName>
  - Rule R2 : <CapitalizedWord> <LastName>
  - Rule R3 : <FirstName><CapitalizedWord>
- **Iterations 3, 4 and 5** : Handle overlapping annotations
  - Consolidation
  - Subtraction

Iteration 1: <FirstName><LastName>

DEMO

```
 --Find first names, using a dictionary.
create view FirstName as
extract
  dictionary 'strictfirst.dict' on D.text as first
from Document D
having MatchesRegex( /[A-Z][a-z]*/ , first);

 --Find last names, using a dictionary.
create view LastName as
extract
  dictionary 'strictlast.dict'  on D.text as last
from Document D
having MatchesRegex( /[A-Z][a-z]*/ , last);

 --Find complete names
create view Person as
select FN.first as first, LN.last as last, CombineSpans(FN.first, LN.last) as name
from FirstName FN, LastName LN
where FollowsTok(FN.first, LN.last,0,0);
```

Results after iteration 1

DEMO

Investment Professionals

Kim Marvin
John Becker
Dino Cusumano
Paul Bamatter
Kenneth Dabrowski
Ryan Hodgson
Graham Sullivan
Eric Baroyan

Advisory Board

Medhi Ali

Erwin Billig
David Boerger

Maurice Holmes

Rule identifies person names accurately

Need more rules to improve recall

Iteration 2: Combining rules R1, R2 and R3

DEMO

```
-- Find capitalized words using a regular expression
create view CapitalizedWord as
extract
  regex /\b\p{Lu}\p{M}*(\p{L}\p{M}*){0,10}(['-][\p{Lu}\p{M}*])?(\p{L}\p{M}*){1,10}\b/
                   on D.text as word
from Document D;

-- Rule R2 <CapitalizedWord><LastName>
create view CapitalizedWordLastName as
select CombineSpans(CW.word, LN.last) as name
from CapitalizedWord CW, LastName LN
where FollowsTok(CW.word, LN.last,0,0);

-- Union results of all three rules
create view Person as
(select R.name as name from FirstNameLastName R)
union all
(select R.name as name from CapitalizedWordLastName R)
union all
(select R.name as name from FirstNameCapitalizedWord R);
```

Results after iteration 2

DEMO

Example text		Rule	Annotations
…		Rule R1	Kim Marvin John Becker …
Investment Professionals		Rule R2	Professionals Kim Kim Marvin John Becker Cusumano Paul …
Kim Marvin John Becker		Rule R3	Kim Marvin Marvin John John Becker …
Dino Cusumano			
…			

Overlapping annotations output by different rules

Use the fact that Rules R2 and R3 are weaker than Rule R1

Iteration 3 : Delete weaker matches overlapping with R1

DEMO

```
-- union Rules R2, R3
create view WeakPersons as
(select R.name as name from CapitalizedWordLastName R)
union all
(select R.name as name from FirstNameCapitalizedWord R);

-- Identify WeakPersons overlapping with R1
create view WeakPersonsToDelete as
select   WP.name as name
from     FirstNameLastName R, WeakPersons WP
where  Overlaps(R.name, WP.name);

-- WeakPersons that do not overlap with R1
create view WeakPersonsRemaining as
(select R.name as name from WeakPersons R)
minus
(select R.name as name from WeakPersonsToDelete R);

-- Union results of R1 and remaining weak persons
create view Person as
(select R.name as name from FirstNameLastName R)
union all
(select R.name as name from WeakPersonsRemaining R);
```

Results after iteration 3

DEMO

…

Investment Professionals

Kim Marvin
John Becker

Dino Cusumano

Paul Bamatter

…

Overlaps resolved

Overlapping annotations remain across Rules R2 and R3

Iteration 4 : Consolidate annotations

DEMO

```
-- Union results of R1 and remaining weak persons
create view AllPersons as
(select R.name as name from FirstNameLastName R)
union all
(select R.name as name from WeakPersonsRemaining R);

create view Person as
select R.name as name
from AllPersons R
-- consolidate overlapping matches in a left-to-right fashion
consolidate on R.name
using 'LeftToRight' ;
```

Results after iteration 4

DEMO

Investment Professionals

Kim Marvin
John Becker
Dino Cusumano
Paul Bamatter
Kenneth Dabrowski
Ryan Hodgson
Graham Sullivan
Eric Baroyan

Advisory Board

Medhi Ali

Erwin Billig

David Boerger

Maurice Holmes

LeftToRight consolidation results in some mistakes

Iteration 5 : Disallow newlines in weaker rule matches

DEMO

```
-- Union results of R1 and remaining weak persons
create view AllPersons as
(select R.name as name from FirstNameLastName R)
union all
(select R.name as name from WeakPersonsRemaining R
 -- weak matches do not span newlines
 where Not(ContainsRegex( /[\n\r]/ ,R.name)));

create view Person as
select R.name as name
from AllPersons R
-- consolidate overlapping matches in a left-to-right fashion
consolidate on R.name
using 'LeftToRight‵ ;
```

Results after iteration 5

DEMO

Investment Professionals

Kim Marvin
John Becker
Dino Cusumano
Paul Bamatter
Kenneth Dabrowski
Ryan Hodgson
Graham Sullivan
Eric Baroyan

Advisory Board

Medhi Ali

Erwin Billig

David Boerger

Maurice Holmes

AQL Summary

- **Statements**
  - Create view : Creates a new logical view
  - Extract : Extract basic features from text
    - Regex, Dictionary
  - Select : constructing complex patterns from simpler building blocks
    - Select … from … where … consolidate … order by
- **Built-in functions**
  - Predicate functions : Contains, ContainsRegex, Follows, …
  - Scalar functions : CombineSpans, LeftContext, RightContext, …
  - Table functions : Block, BlockTok

Roadmap for SystemT

- **SystemT in-depth**
  - Data Model and Algebra
  - Annotation Query Language (AQL)
  - Optimization

An Aside: Relational Query Optimization

- **Central concept in relational databases**
  - User specifies what she is looking for
  - System decides how to find it
  - Greatly reduces development and maintenance costs
- **Basic approach**
  - Enumerate many equivalent relational algebra expressions
  - Estimate the cost of each one
  - Choose the fastest

What's new in SystemT Optimization

- **Query optimization is a familiar topic in databases. What's different?**
  - Operations over sequences and spans
  - Document-at-a-time processing model
  - Costs concentrated in extraction operators (dictionary, regular expression)

Main Components in SystemT Optimizer

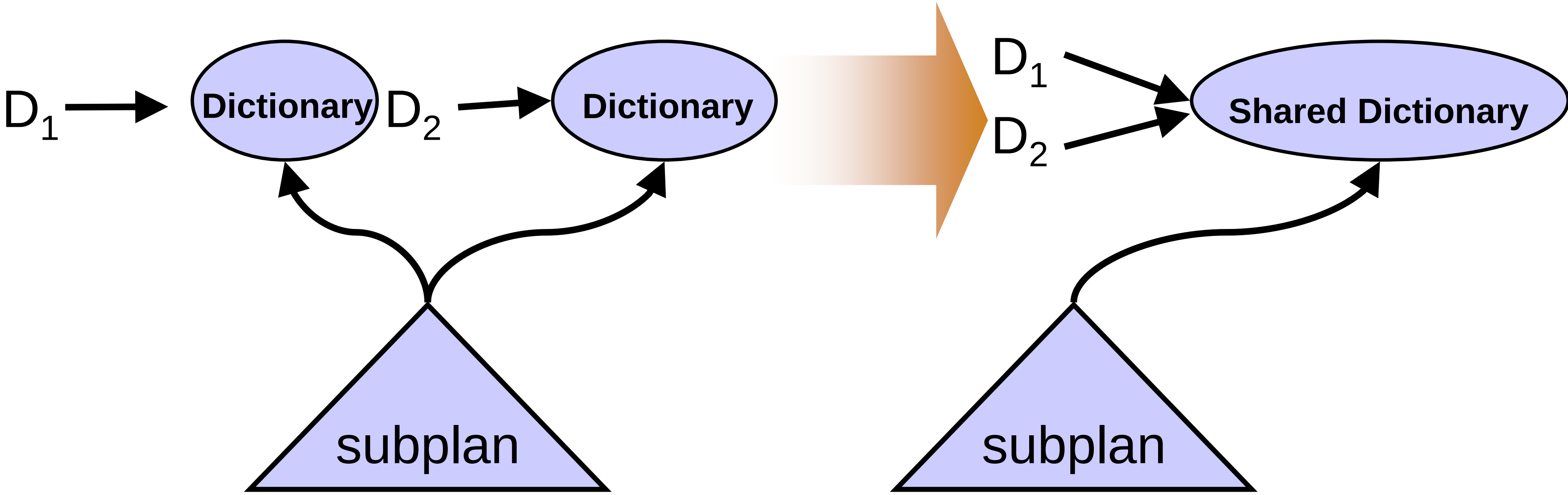
- **Rule rewriting**
  - Text specific query rewrites to reduce cost of extraction primitives
  - E.g., Regular Expression Strength Reduction, Shared Dictionary Matching
- **Cost-based optimization**
  - Choose join orders and methods to minimize cost of extraction primitives
  - Take advantage of document-at-a-time execution
  - E.g., Conditional Evaluation, Restricted Span Evaluation

Regular Expression Strength Reduction (RSR)

- **Basic idea:**
  - Build a fast engine for a restricted class of regular expressions
    - Regular expressions enumerating a fixed set of strings
    - Disallow complex syntactic constructs like lookaheads and lookbehinds
  - Use the fast engine when possible
- **Several different techniques available**
  - Some make single regexes faster
  - Others evaluate multiple regexes at once
  - Others use indexing

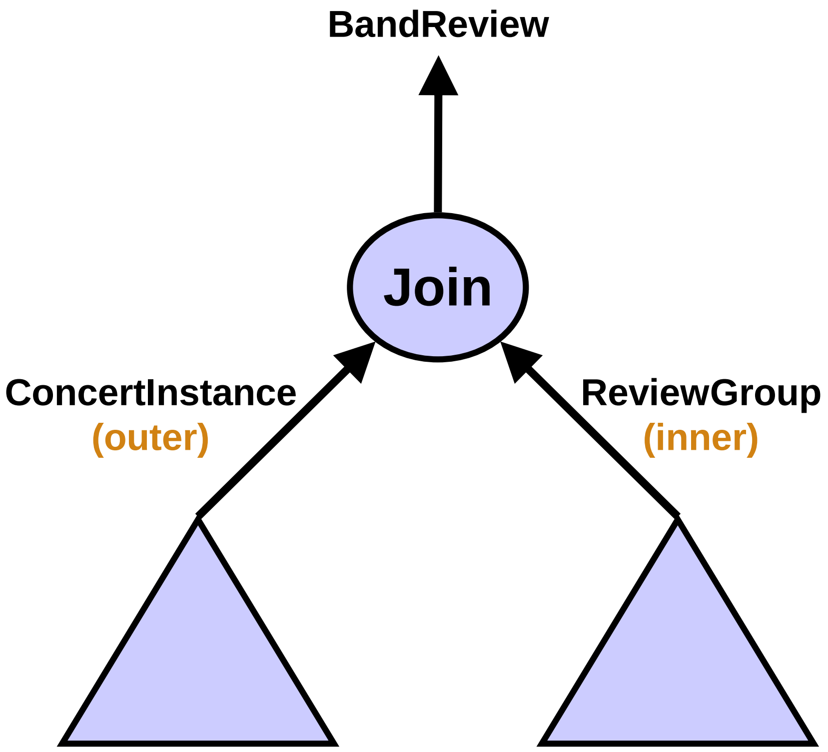
Shared Dictionary Matching (SDM)

- **Dictionary matching has 3 steps:**
  - Tokenize text
  - Hash each token
  - Generate matches based on hash table entry
- **Can share the first two steps among many dictionaries**



Conditional Evaluation (CE)

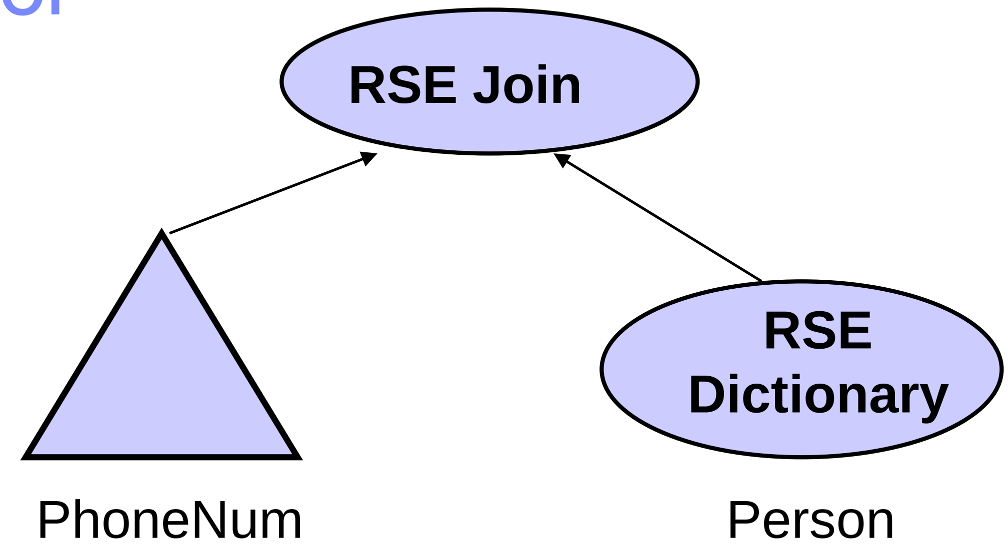
- **Leverage document-at-a-time processing**
- **Don't evaluate the inner operand of a join if the outer has no results**
- **Example: Band review**
  - Can skip one side of the top-level join

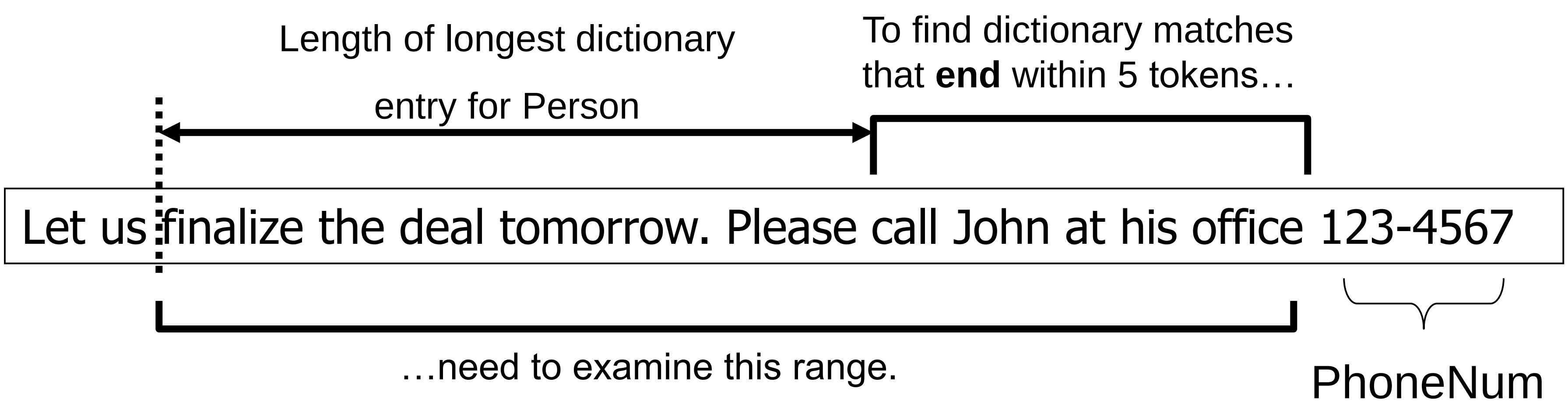


Restricted Span Evaluation (RSE)

- **Conditional evaluation at a finer granularity**
- **Only perform extraction on the portions of the document that could match the join predicate**

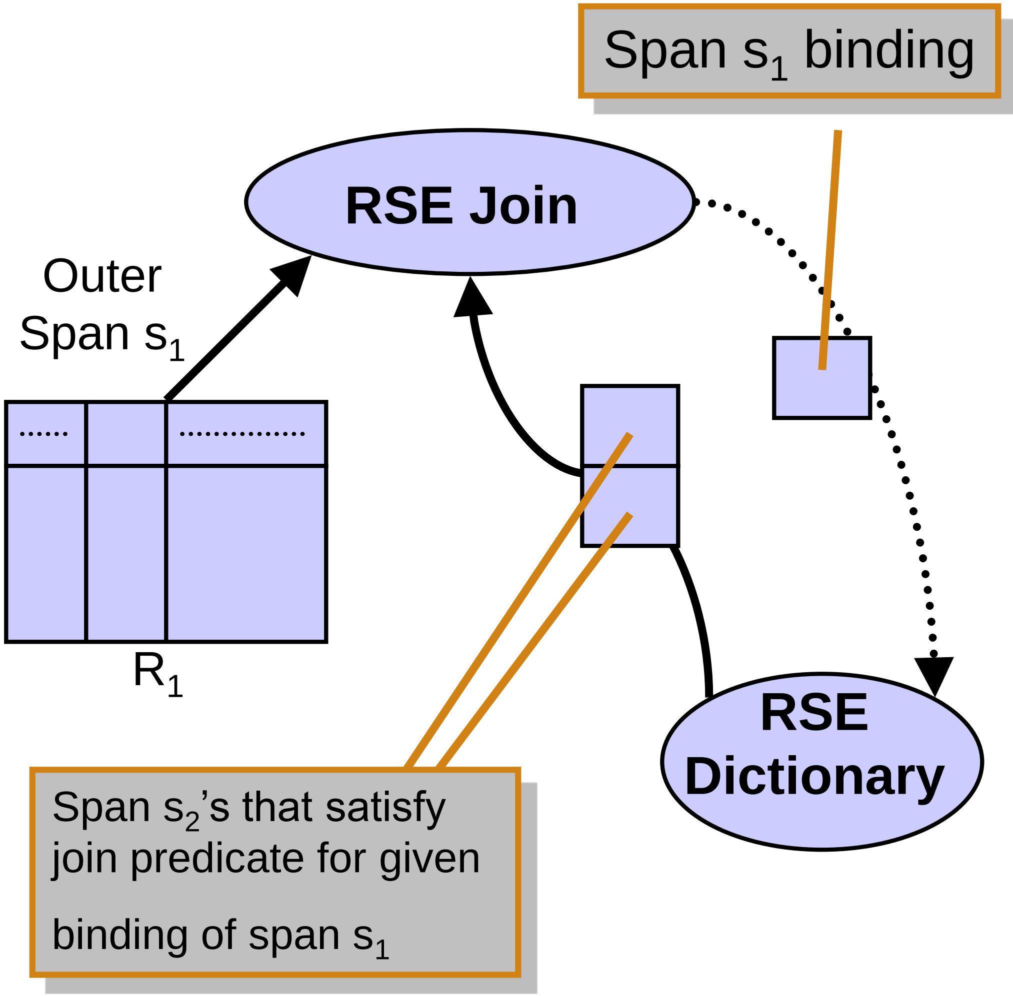
RSE Dictionary Operator





Restricted Span Evaluation (RSE)

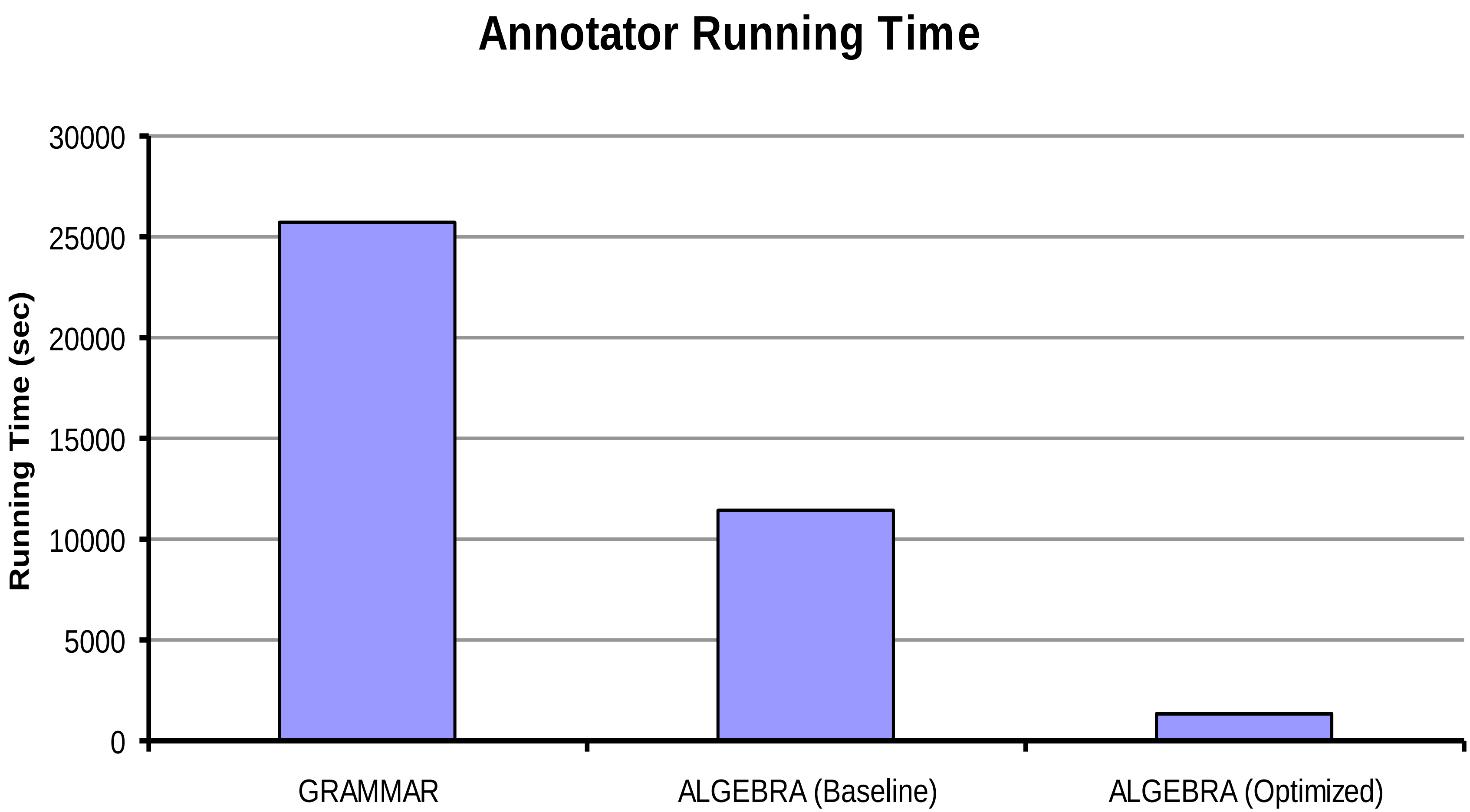
- **For each outer span, pass *join bindings* down to the inner of the join**
- **Extraction performed in the "neighborhood" of given span based on join predicate**
- **Requires special physical operators to implement this extraction:**
  - RSE Dictionary
  - RSE Regex



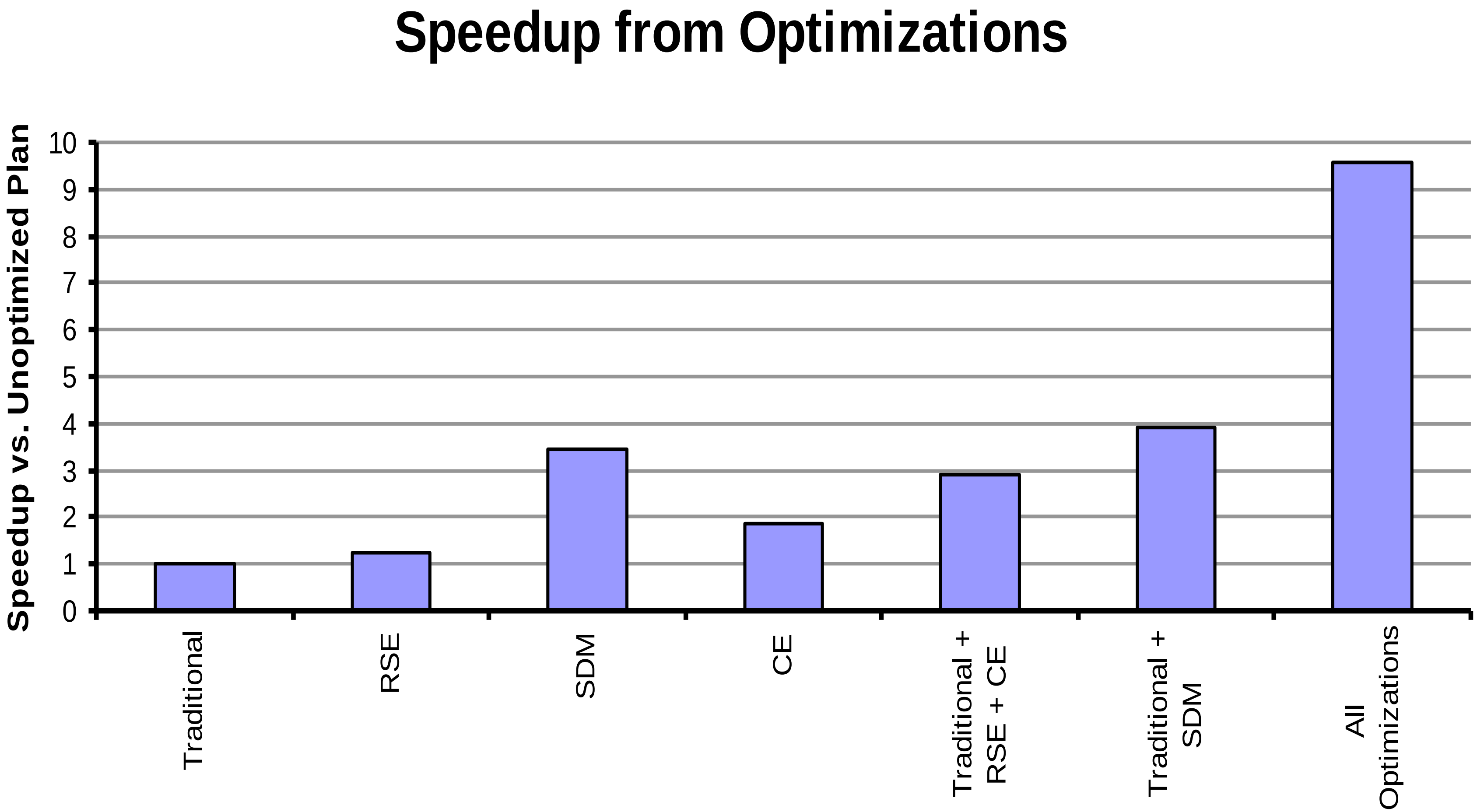
Optimization Experiments : BandReview annotator

- **BandReview annotator described earlier**
  - 40 rules over 33 dictionaries, 13 regular expressions
- **Data set:**
  - 4.5 million blogs
  - 5.1 GB data
- **3 implementations of annotator**
  - GRAMMAR
    - Our own CPSL engine
  - ALGEBRA(Baseline)
    - Translation of CPSL rules into algebra
    - First level of grammar becomes extraction operators
    - Higher levels of grammar become joins and aggregations
  - ALGEBRA(Optimized):
    - Use SDM, RSE, CE, join reordering to generate alternative plans
    - Statistics gathered from a 100-document sample

Experimental Results



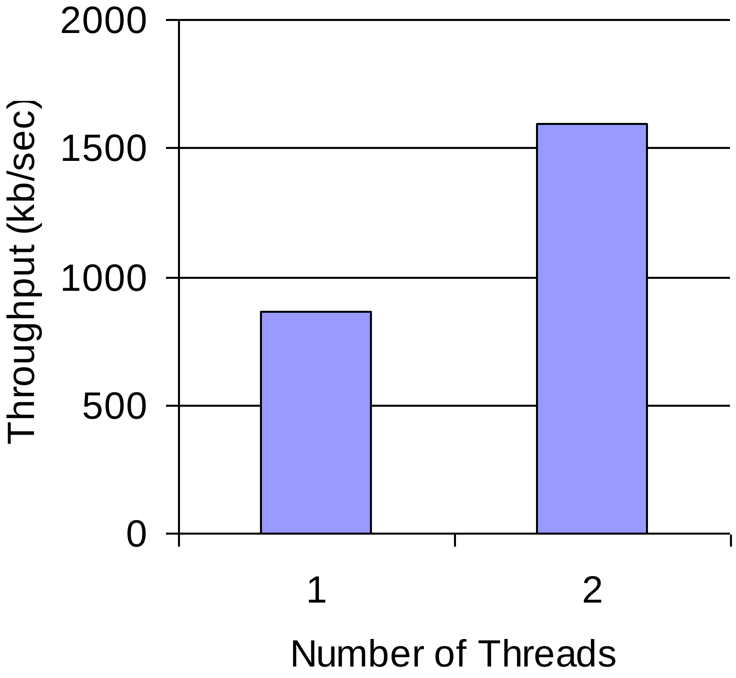
Experimental Results



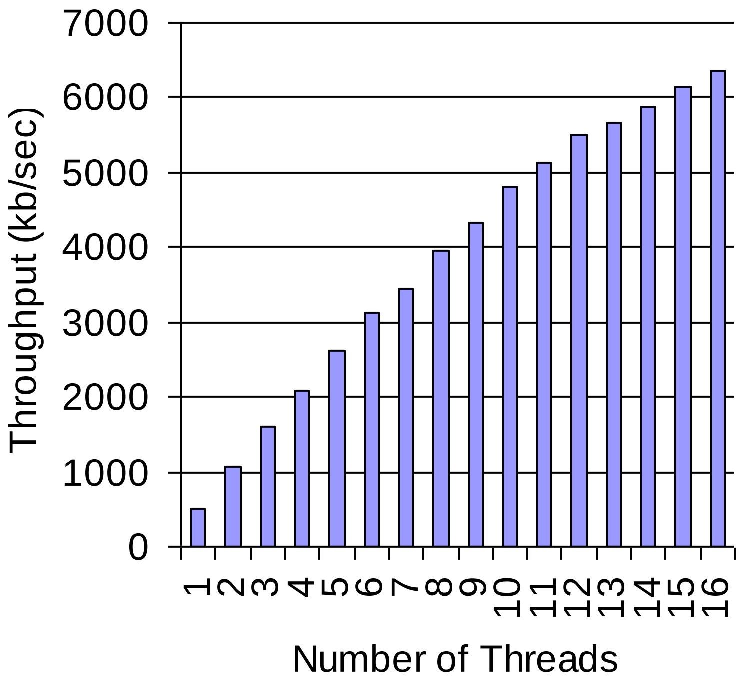
SystemT Named Entity Annotators

- **Statistics:**
  - 8 types of entities
  - 327 AQL statements
  - Throughput: 800+ kb/sec/core (on a laptop)
- **Entities extracted**
  - Person, Organization, Address, Phone Number, Email Address, Url, Date, Time

Performance of SystemT Named-Entity Annotator



Laptop (Intel Core 2 Duo 2.33 GHz)



Server (4×quad-core AMD Opteron)

Research Directions

- We have seen the advantages of a declarative approach to rule based information extraction.
- Opens up several interesting research issues
  - Theoretical questions
    - Alternative algebras for IE
    - Desiderata for IE algebras
    - Building in imprecision and uncertainty into IE algebras
  - Systems and techniques to assist in building rule-sets for specific extraction tasks
  - Performance optimization
    - Indexing techniques to speedup extraction
    - Text-specific optimization techniques
    - Cost estimation techniques

References: Systems described in this Tutorial

- AFst
  - B. Boguraev, “Annotation-based finite state processing in a large scale NLP architecture,” Recent Advances in Natural Language Processing III, 2004.
- JAPE (http://gate.ac.uk/)
  - H. Cunningham, D. Maynard, V. Valentin Tablan, “JAPE: A Java Annotation Patterns Engine,” Research Memo, Dept. of Computer Science, Univ. of Sheffield, 2000.
- CIMPLE (http://pages.cs.wisc.edu/~anhai/projects/cimple/)
  - P. DeRose, W. Shen, F. Chen, A. Doan, R. Ramakrishnan, “Building Structured Web Community Portals: A Top-Down, Compositional, and Incremental Approach,” VLDB 2007.
- SystemT (http://www.almaden.ibm.com/cs/projects/avatar/)
  - F. Reiss, S. Raghavan, R. Krishnamurthy, H. Zhu, and S. Vaithyanathan, “An Algebraic Approach to Information Extraction,” ICDE 2008.

References: Software and Data-sets

- **Data sets**
  – Linguistic Data Consortium
    http://www.ldc.upenn.edu/
  – Repository of Online Information Sources Used in Information Extraction Tasks (RISE)
    http://www.isi.edu/info-agents/RISE/
- **Natural Language Frameworks**
  – UIMA (Unstructured Information Management Architecture)
    http://www.research.ibm.com/UIMA/
  – GATE (A General Architecture for Text Engineering)
    http://gate.ac.uk/
- **Rule development environment**
  – System Text for Information Extraction (SystemT Development Environment)
    http://www.alphaworks.ibm.com/tech/systemt/
  – JAPE (part of the GATE distribution)
    http://gate.ac.uk/
- **Machine Learning Toolkits**
  – MALLET (Machine Learning for LanguageE Toolkit)
    http://mallet.cs.umass.edu/index.php/Main_Page
  – DOT.KOM IE Tools
    http://tcc.itc.it/research/textec/projects/dotkom/
  – MinorThird
    http://minorthird.sourceforge.net/

References: Related Tutorials

- D. Appelt et. al, "Introduction to Information Extraction Technology", IJCAI-99 Tutorial
- J. Cowie & W. Lehnert, “Information Extraction”, Communications of the ACM, 39:1, 1996.
- C. Cardie, “Empirical Methods in Information Extraction”, AI Magazine, 18:4, 1997.
- W. Cohen & A. McCallum, “Information Extraction from the World Wide Web”, NIPS 2002 & KDD 2003.
- E. Agichtein & S. Sarawagi, “Scalable Information Extraction and Integration”, KDD 2006.
- R. Feldman, “Information Extraction, Theory and Practice”, ICML 2006.
- A. Doan, R. Ramakrishan, & S. Vaithyanathan, “Managing Information Extraction”, SIGMOD 2006.

Upcoming SIGMOD Record Issue on IE

- Papers describing several IE systems including
  - TEXTRUNNER, WEBTABLES, GOOGLE DEEP WEB CRAWLER from Google and University of Washington
  - KYLIN from University of Washington
  - YAGO-NAGA from Max Planck Institute
  - SQoUT from Columbia University
  - Purple SOX from Yahoo!
  - SystemT from IBM Almaden
  - CIMPLE from University of Wisconsin