Stanford CoreNLP *

David McClosky, Mihai Surdeanu, Chris Manning and many, many others

4/22/2011

* Previously known as BaselineNLProcessor

Overview

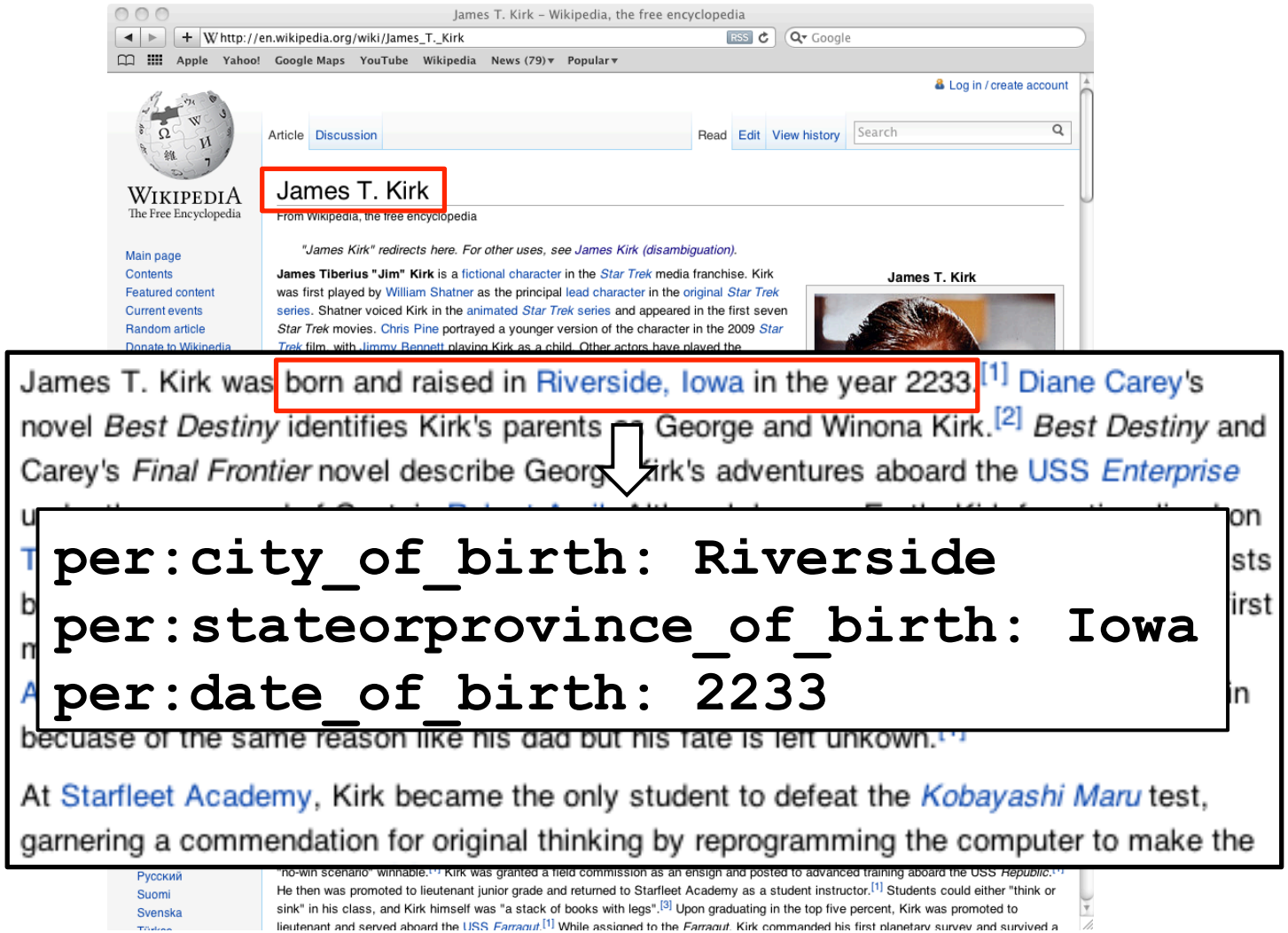
- Part I: KBP task overview
- Part II: Stanford CoreNLP
- Part III: NFL Information Extraction

Knowledge Base Population (KBP)

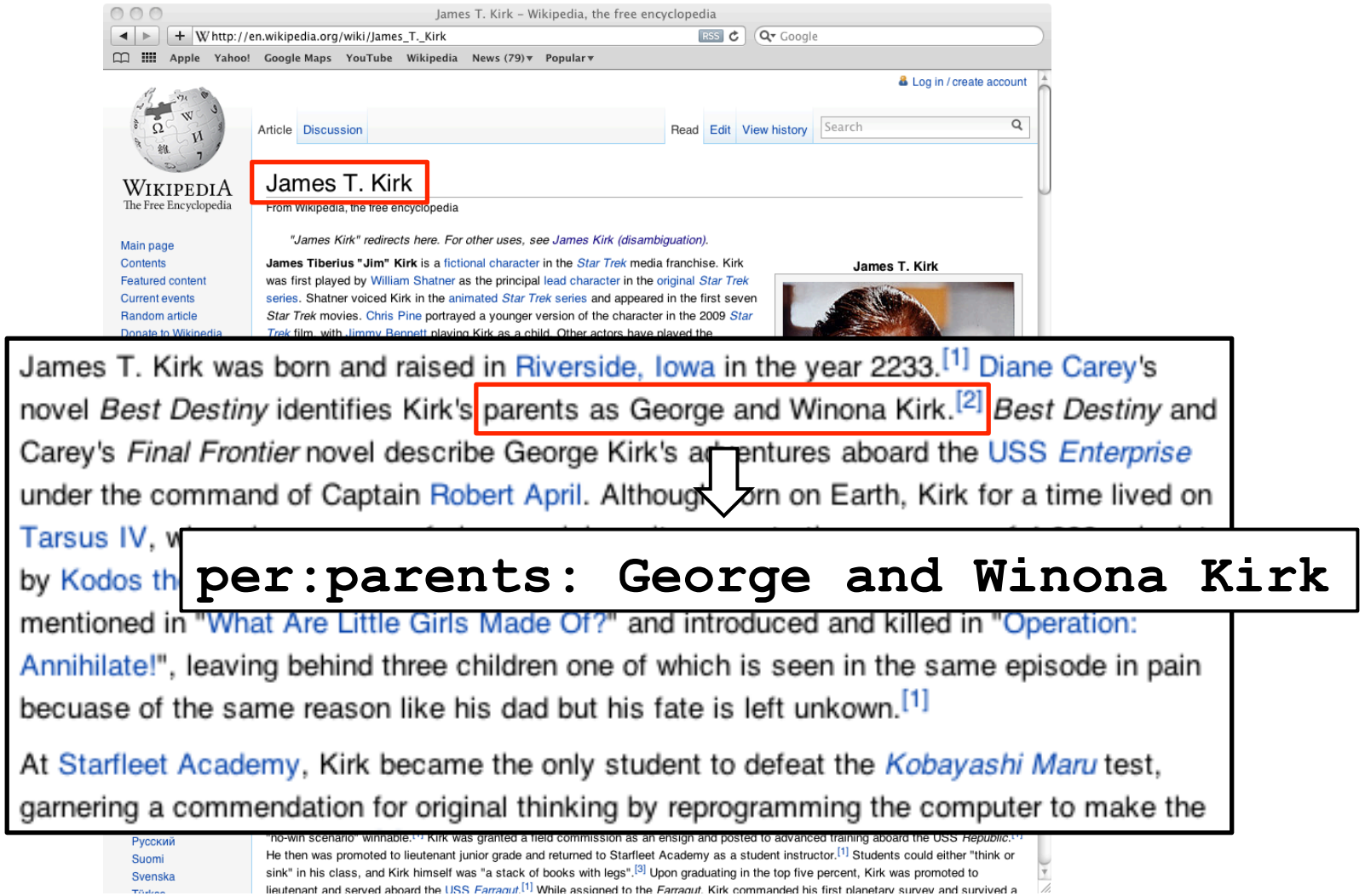
- **KBP** is a **bake-off** (shared task) held yearly
- **Task:** Given an entity, fill in values for various slots
- **Entities** can be people or organizations
- **Slots** are prespecified
- **Provenance** (textual sources) must be provided

Knowledge Base Population (KBP)

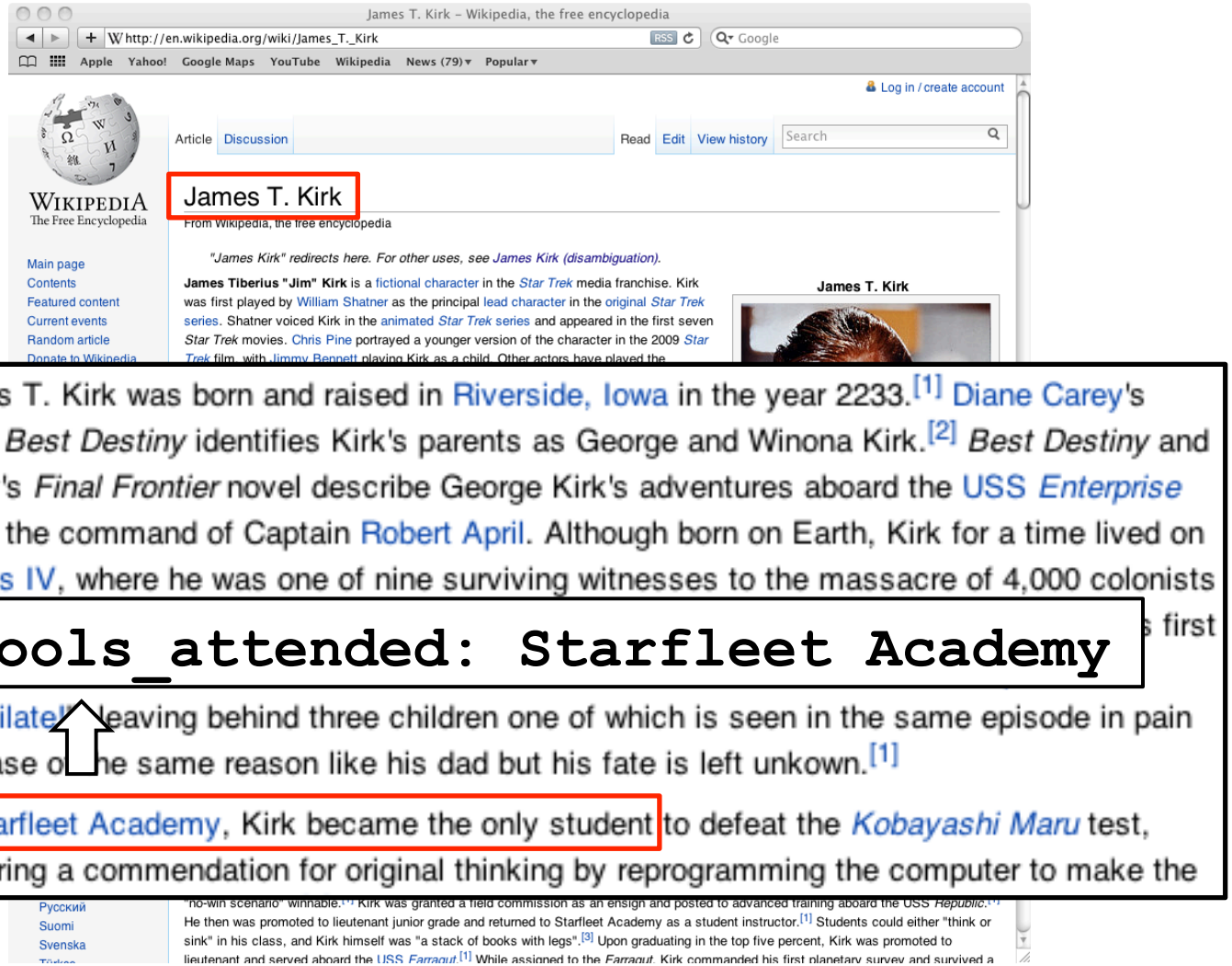
Knowledge Base Population (KBP)



Knowledge Base Population (KBP)



Knowledge Base Population (KBP)



Task Specification

- Inputs
  - Knowledge base
    - Entities, slot names and fillers
  - Source collection
    - Wikipedia, newswire text, broadcast news
  - Evaluation data from 2009, 2010
    - Queries (entities)
    - Slot names and fillers
  - Testing: Queries
- Output
  - Slot name, filler, document ID (provenance)

KBP Challenges: Slot types


```
per:parents: George and Winona Kirk
per:city_of_birth: Riverside
per:stateorprovince_of_birth: Iowa
per:date_of_birth: 2233
per:title: Captain
```

Type

List of people

City

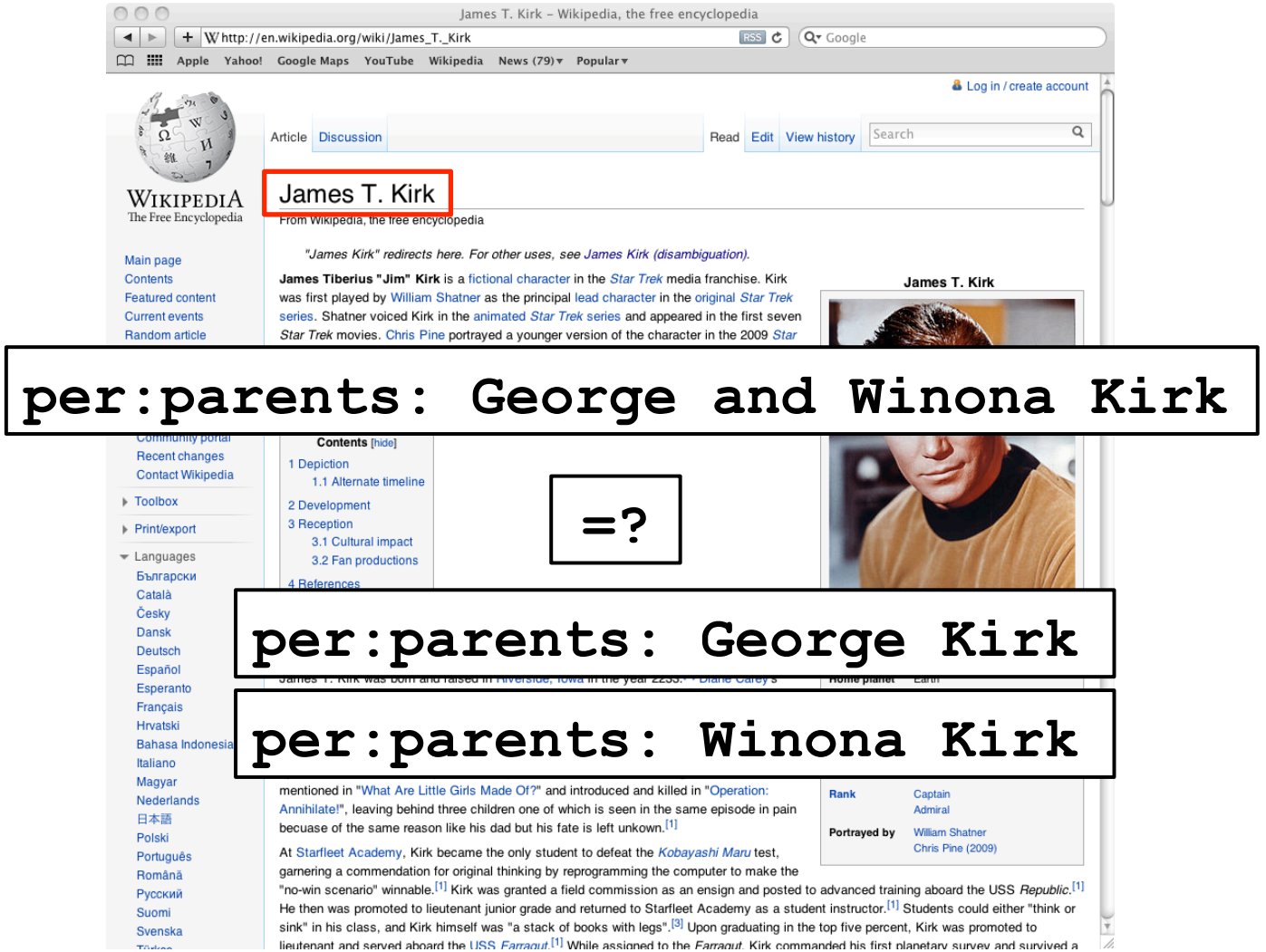
State

Date

Title?

…

KBP Challenges: List slots



KBP Challenges: Consistency

per:city_of_birth: Shi'Kahr ⊗

per:stateorprovince_of_birth: Iowa

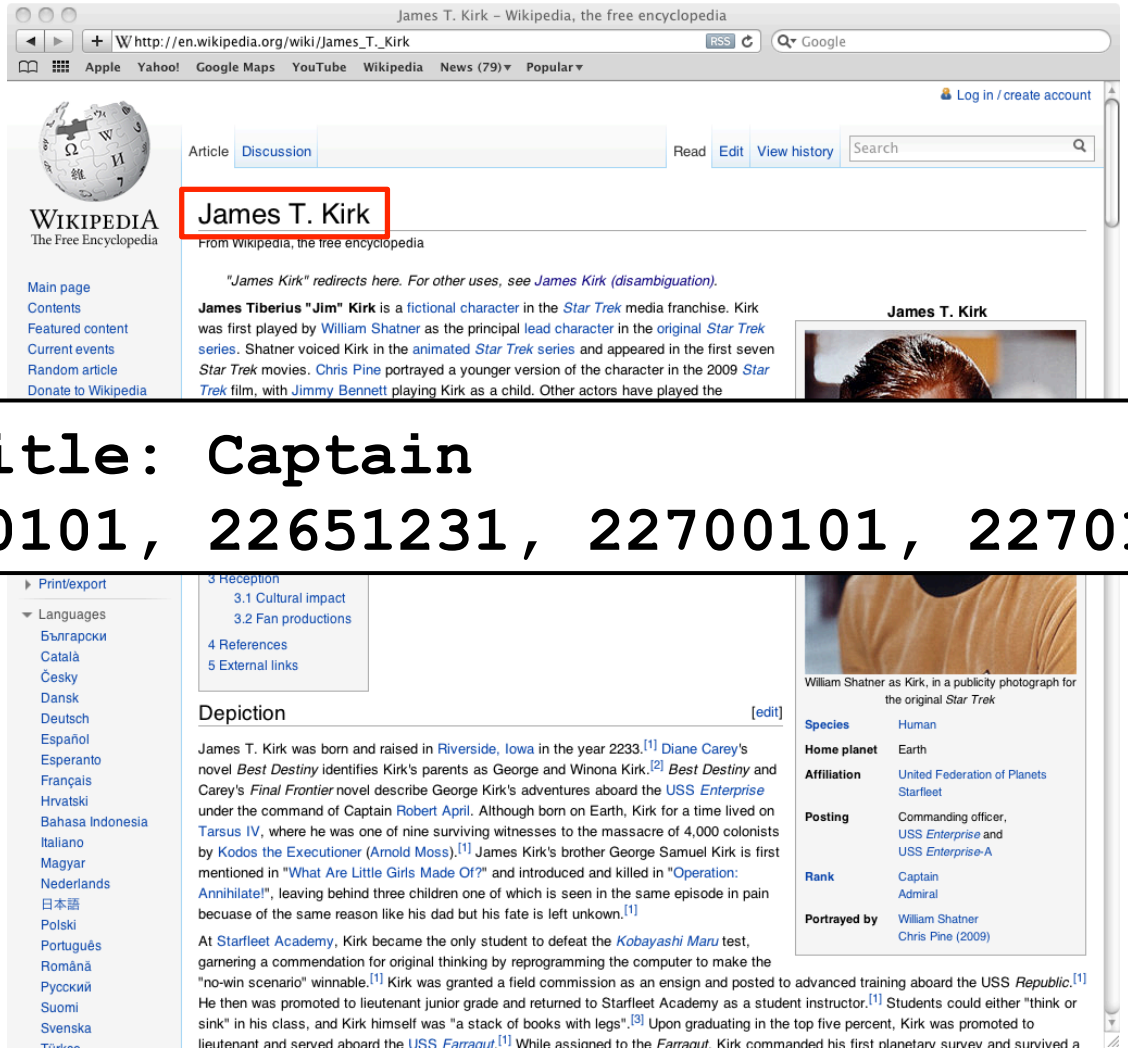
KBP Challenges: Coreference

Temporal Extensions

- Temporal slot filling:
  - **`per:spouse`**
  - **`per:title`**
  - **`org:top_employees/members`**
  - **`...`**

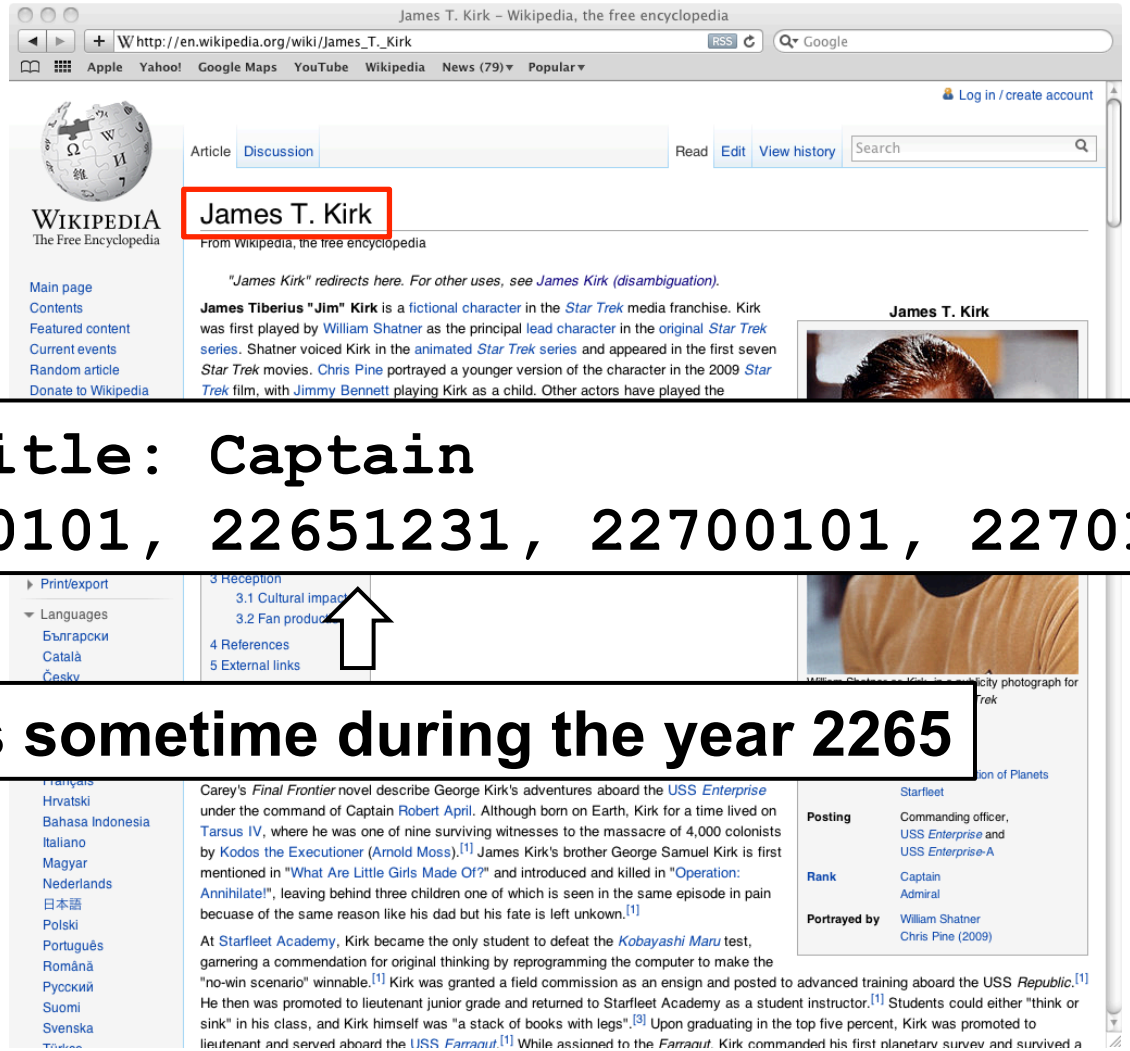
- Simple representation:
  - **`[T1, T2, T3, T4]`**
    = **`T1`** ≤ start ≤ **`T2`**, **`T3`** ≤ end ≤ **`T4`**
  - Any can be null to indicate a lack of constraint
  - Day resolution (YYYYMMDD)

Temporal Example




```
per:title: Captain
[22650101, 22651231, 22700101, 22701231]
```

Temporal Example



per:title: Captain
[22650101, 22651231, 22700101, 22701231]

Start is sometime during the year 2265

Overview

- Part I: KBP task overview
- Part II: Stanford CoreNLP
- Part III: NFL Information Extraction

Stanford CoreNLP Overview

We've been working on a whole bunch of stuff:

- Joint NLP models
- Coreference (now in Stanford CoreNLP)
- Supervised relation extraction (NFL)
- Supervised event extraction (BioNLP)
- Distantly supervised relation extraction (KBP)
- Scenario templates and graph models in IE

This section describes our NLP pipeline,
common to many of these components.

http://nlp.stanford.edu/software/corenlp.shtml

Stanford CoreNLP Outline

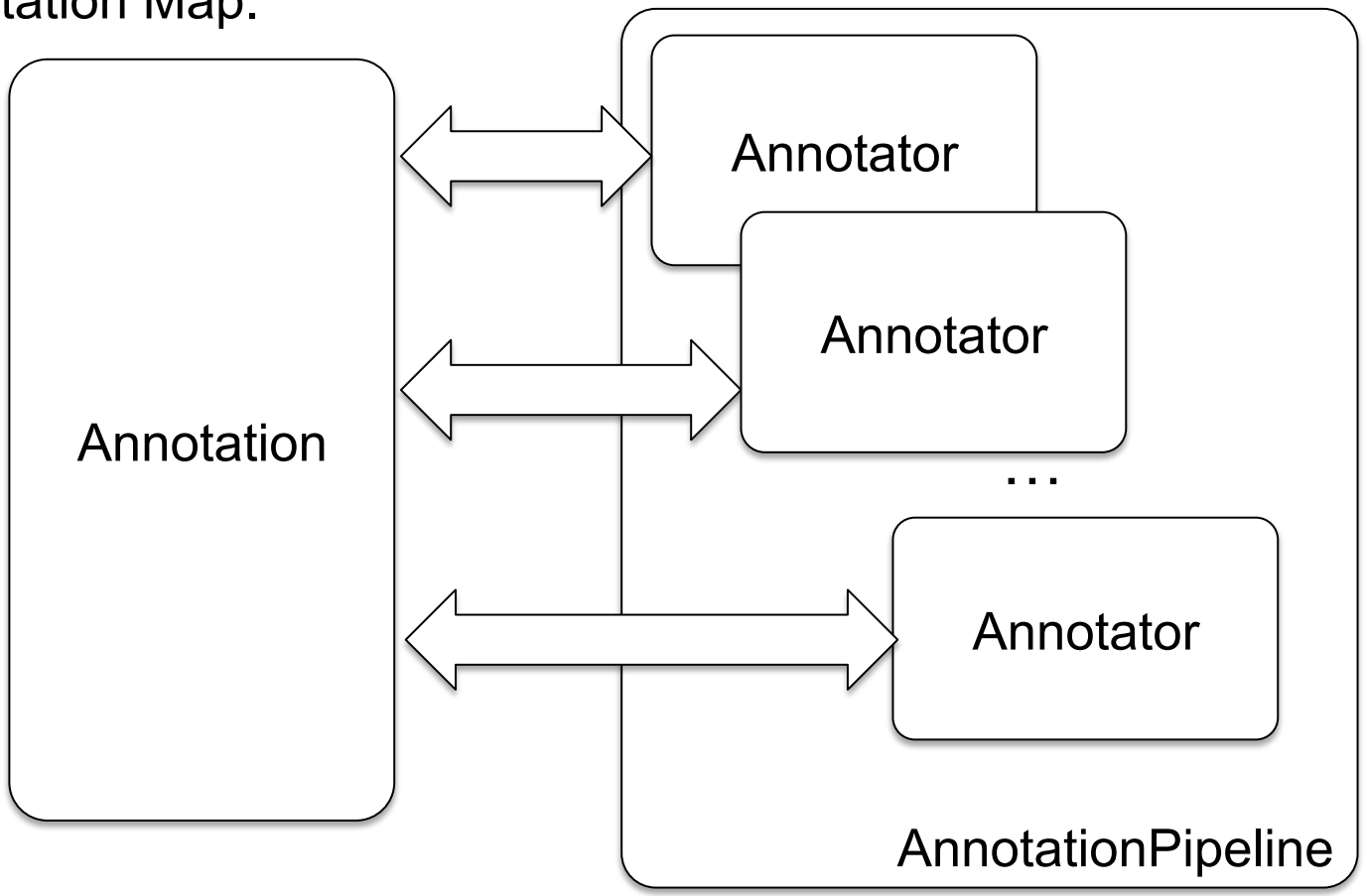
- Approach
- How to use
  - Command-line (shell, batch)
  - Java interface

Motivation

- Quickly and painlessly get linguistic annotations for a text
- Hides variations across components behind common API
- Simple Java objects passed around (no XML, UIMA, etc.)
  - But results can easily be written to XML, etc.

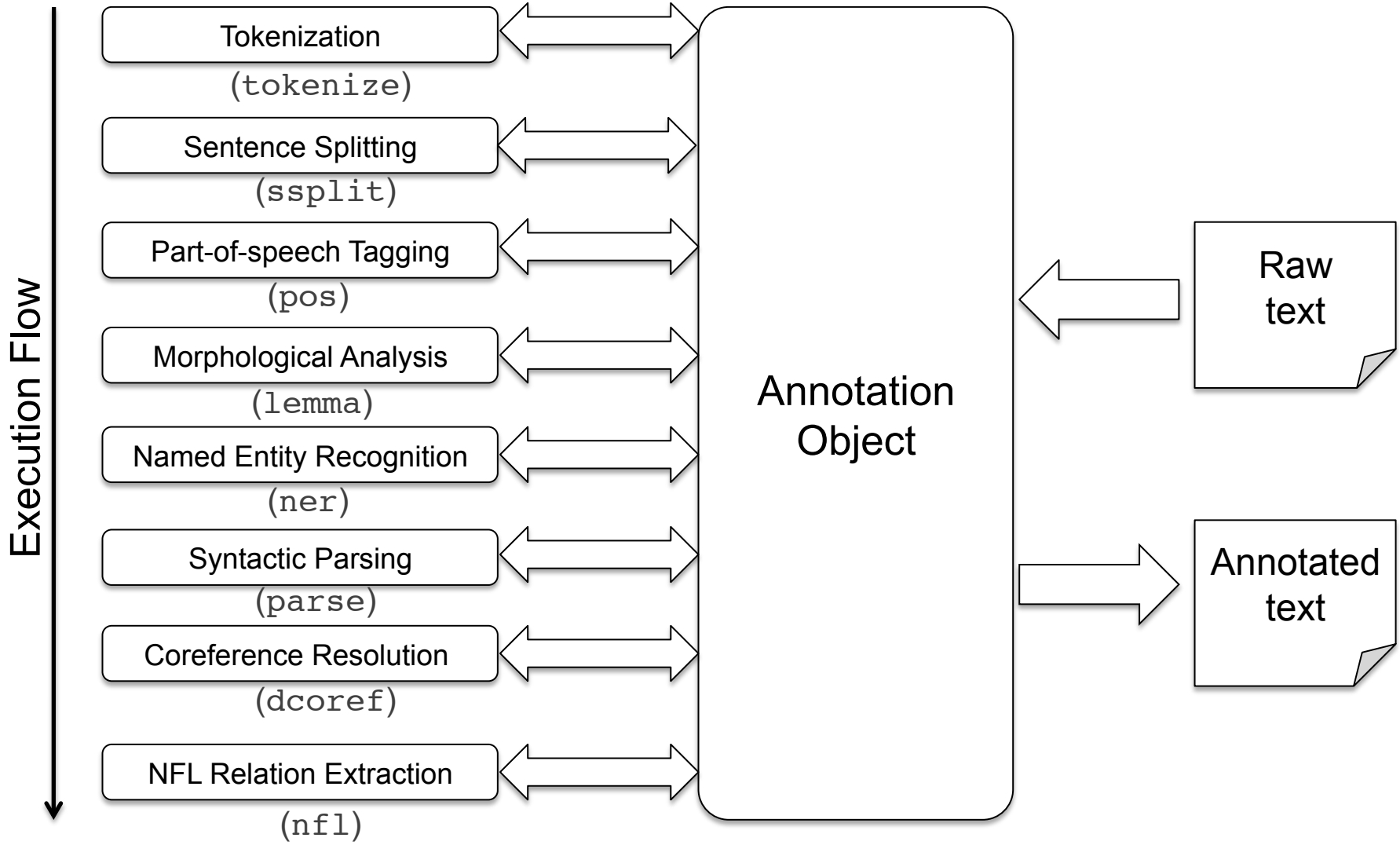
Idea

Store the input text as well as the output of each Annotator as values in an Annotation Map.



- There are dependencies between Annotators ➔ the pipeline ordering is important!

Stanford CoreNLP Pipeline



Stanford CoreNLP Outline

- Approach
- How to use
  - Command-line (shell, batch)
  - Java interface

Running as a shell

- `java -cp classes:lib/xom.jar:lib/jgrapht.jar -Xmx6g edu.stanford.nlp.pipeline.StanfordCoreNLP -props src/edu/stanford/nlp/pipeline/StanfordCoreNLP.properties`

Example sentence: `Stanford is located in California.`

```
Sentence #1 (6 tokens):
[Word=Stanford Current=Stanford Tag=NNP Lemma=Stanford NER=ORGANIZATION]
[Word=is Current=is Tag=VBZ Lemma=be NER=O]
[Word=located Current=located Tag=VBN Lemma=locate NER=O]
[Word=in Current=in Tag=IN Lemma=in NER=O]
[Word=California Current=California Tag=NNP Lemma=California NER=LOCATION]
[Word=. Current=. Tag=. Lemma=. NER=O]

(ROOT (S (NP (NNP Stanford))
         (VP (VBZ is) (VP (VBN located) (PP (IN in) (NP (NNP California)))))
         (. .)))

nsubjpass(located-3, Stanford-1)
auxpass(located-3, is-2)
prep_in(located-3, California-5)
```

Running in batch mode

- `java -cp classes:lib/xom.jar:lib/jgrapht.jar -Xmx6g edu.stanford.nlp.pipeline.StanfordCoreNLP -props src/edu/stanford/nlp/pipeline/StanfordCoreNLP.properties -file input.txt`

- `java -cp classes:lib/xom.jar:lib/jgrapht.jar -Xmx6g edu.stanford.nlp.pipeline.StanfordCoreNLP -props src/edu/stanford/nlp/pipeline/StanfordCoreNLP.properties -file inputdirectory –extension .xml`

- `java -cp classes:lib/xom.jar:lib/jgrapht.jar -Xmx6g edu.stanford.nlp.pipeline.StanfordCoreNLP -props src/edu/stanford/nlp/pipeline/StanfordCoreNLP.properties -file inputdirectory –outputDirectory somewhereElse -outputExtension .annotated –replaceExtension true -noClobber`

Running in Java

```
Annotator pipeline =
  new StanfordCoreNLP(properties);

Annotation annotation =
  new Annotation(text);

pipeline.annotate(annotation);
```

Annotators in CoreNLP

- `tokenize` – split text into tokens, PTB-style
- `cleanxml` – remove specific XML tags
- `truecase` – restore case (e.g. if all lowercase, etc.)
- `ssplit` – sentence splitter
- `pos` – add POS tags to tokens
- `lemma` – add lemmas to tokens

More annotators in CoreNLP

- `ner` – add named entity tags to tokens
- `regexner` – add rule-based NER tags from regular expressions
- `parse` – add parse trees (Stanford Parser)
- `berkeleyparse`, `charniakparse` – Add parse trees from other parsers as well

- `dcoref` – add coreference links
- `nfl` (Machine Reading distribution only) – add NFL entity and relation extraction mentions
- `time` – add temporal annotations (coming later!)

Interpreting the Output

```
List<CoreMap> sentences = annotation.get(SentencesAnnotation.class);
for (CoreMap sentence : sentences) {
    // traversing the words in the current sentence
    for (CoreLabel token: sentences.get(i).get(TokensAnnotation.class)) {
        String word = token.get(TextAnnotation.class);
        String pos = token.get(PartOfSpeechAnnotation.class);
        String ne = token.get(NamedEntityTagAnnotation.class);
    }
    // this is the parse tree of the current sentence
    Tree tree = sentence.get(TreeAnnotation.class);
}

// this is the coreference link graph
List<Pair<IntTuple, IntTuple>> graph = annotation.get(CorefGraphAnnotation.class);
```

Interpreting the Output

hash map with class objects as keys and custom value types

```
List<CoreMap> sentences = annotation.get(SentencesAnnotation.class);
for (CoreMap sentence : sentences) {
    // traversing the words in the current sentence
    for (CoreLabel token: sentences.get(i).get(TokensAnnotation.class)) {
        String word = token.get(TextAnnotation.class);
        String pos = token.get(PartOfSpeechAnnotation.class);
        String ne = token.get(NamedEntityTagAnnotation.class);
    }
    // this is the parse tree of the current sentence
    Tree tree = sentence.get(TreeAnnotation.class);
}

// this is the coreference link graph
List<Pair<IntTuple, IntTuple>> graph = annotation.get(CorefGraphAnnotation.class);
```

Interpreting the Output

hash map with class objects as keys and custom value types

`CoreMap` with additional properties (`HasWord`, `HasTag`, etc.)

```
List<CoreMap> sentences = annotation.get(SentencesAnnotation.class);
for (CoreMap sentence : sentences) {
    // traversing the words in the
    for (CoreLabel token: sentences.get(i).get(TokensAnnotation.class)) {
        String word = token.get(TextAnnotation.class);
        String pos = token.get(PartOfSpeechAnnotation.class);
        String ne = token.get(NamedEntityTagAnnotation.class);
    }
    // this is the parse tree of the current sentence
    Tree tree = sentence.get(TreeAnnotation.class);
}

// this is the coreference link graph
List<Pair<IntTuple, IntTuple>> graph = annotation.get(CorefGraphAnnotation.class);
```

Interpreting the Output

hash map with class objects as keys and custom value types

```
List<CoreMap> sentences = annotation.get(SentencesAnnotation.class);
for (CoreMap sentence : sentences) {
    // traversing the words in the
    for (CoreLabel token: sentences.get(i).get(TokensAnnotation.class)) {
        String word = token.get(TextAnnotation.class);
        String pos = token.get(PartOfSpeechAnnotation.class);
        String ne = token.get(NamedEntityTagAnnotation.class);
    }
    // this is the parse tree of the current sentence
    Tree tree = sentence.get(TreeAnnotation.class);
}

// this is the coreference link graph
List<Pair<IntTuple, IntTuple>> graph = annotation.get(CorefGraphAnnotation.class);
```

`CoreMap` with additional properties (`HasWord`, `HasTag`, etc.)

uniquely identify a word by <sentence position, token position> (both offsets start at 0) [***note:*** *annotation will change soon…*]

Creating Your Own Annotator

```
/** Simple annotator that recognizes locations stored in a gazetteer */
public class GazetteerLocationAnnotator implements Annotator {
    // this is the only method that must be implemented by an annotator
    public void annotate(Annotation annotation) {
        // traverse all sentences in this document (assumes that text already tokenized)
        for (CoreMap sentence : annotation.get(SentencesAnnotation.class)) {
            // loop over all tokens in sentence
            List<CoreLabel> tokens = sentence.get(TokensAnnotation.class);
            for (int start = 0; start < tokens.size(); start++) {
                // assumes that the gazetteer returns the token index
                // after the match or -1 otherwise
                int end = Gazetteer.isLocation(tokens, start);
                if (end > start) {
                    for (int i = start; i < end; i ++) {
                        tokens.get(i).set(NamedEntityTagAnnotation.class, "LOCATION");
                    }
                }
            }
        }
    }
}
```

Overview

- Part I: KBP task overview
- Part II: Stanford CoreNLP
- Part III: NFL Information Extraction

NFL Information Extraction

- Add "`nfl`" to the "`annotators`" property.
- Construct and call the same way.
- Interpreting output:

```
List<CoreMap> sentences = annotation.get(SentencesAnnotation.class);
for (CoreMap sentence : sentences) {
    List<EntityMention> entities =
         sentence.get(MachineReadingAnnotations.EntityMentionsAnnotation.class);
    List<RelationMention> relations =
         sentence.get(MachineReadingAnnotations.RelationMentionsAnnotation.class);
}
```

Span and type

Entities and type

Useful methods

- `ExtractionObject:`
  - `CoreMap getSentence()`
  - `Span getExtent()`
  - `String getType()`
  - `Counter<String> getTypeProbabilities()`
- `EntityMention:`
  - `int getSyntacticHeadTokenPosition()`
  - `CoreLabel getSyntacticHeadToken()`
- `RelationMention:`
  - `List<ExtractionObject> getArgs()`
  - `List<String> getArgNames()`

NFL System Overview

- Named entity recognition
- Relation extraction

(Throughout: Lessons from adapting our IE system to NFL domain)

NFL System Overview

- Named entity recognition
- Relation extraction

(Throughout: Lessons from adapting our IE system to NFL domain)

CRF entity extractor

- Use NER system to classify each token as one of the NFL entity types or "O" (other)
- Contiguous tokens of the same type are combined into `EntityMentions`
- Marginal probabilities on each token form the results of `getTypeProbabilities()`

Rule-based System

- Extended the NFL team gazetteer with **`NFLGame`** entities extracted from Dekang Lin's distributional similarity dictionary:
  - seeds: win, loss, game
  - added: victory, triumph, shutout, defeat, lead, match, rout, strikeout...
- If a word sequence (partially) matches a gazetteer entry and it includes the head of a NP ➔ gazetteer label
- If the generic NER labels a sequence as **`DATE`** ➔ Date
- If the generic NER labels a sequence as **`NUMBER`** and it is a valid score and not followed by measurement unit ➔ **`Score`**

Goal: maximize recall!

CRF + MTurk

- Harvested 1400+ sentences on NFL games from **`sports.yahoo.com`**
  - "It was the third quarter of the Philadelphia Eagles' 38-10 rout of the Carolina Panthers on Sunday and both franchises suddenly had big worries about their veteran quarterbacks."
- Tagged corpus with rule-based NER, which maximized recall
- Generated MTurk HITs from this data, using all possible relations between the identified NEs
  - "Is it true that the Philadelphia Eagles scored 38 points in this game?" ➔ **yes**
  - "Is it true that the Philadelphia Eagles scored 10 points in this game?" ➔ **no**
- Averaged annotations from four annotators for each HIT

CRF + MTurk: Analysis

- MTurk helped only up to a point…
- Why?
  - There was a bug in the rule-based NER used to generate candidates
  - Turkers could not identify subtle mistakes, hence errors propagated in the final MTurk corpus
    - "… the victory game against Dallas"
    - Is "victory" the best word to describe the game? ➔ **yes**
    - Is "game" the best word to describe the game? ➔ **yes**

Adapting Entity Extraction to NFL

- Gazetteer used for team names, “game” entities
  - “Packers” should match “Green Bay Packers”
  - …but “Bay” shouldn’t.
- Tokenizer wasn’t splitting scores (“37-7”)
- Head finder needed adjustments
  - Heads of entities are critical features for both extraction tasks

NFL System Overview

- Named entity recognition
- Relation extraction

(Throughout: Lessons from adapting our IE system to NFL domain)

Relation Extraction Approach

- Logistic regression classifier
  - Positive datums: annotated relations in the corpus
  - Negative datums: all other possible combinations between existing entities
  - Example:
    - "It was the third quarter of the Philadelphia Eagles' 38-10 rout of the Carolina Panthers on Sunday and both franchises suddenly had big worries about their veteran quarterbacks."
    - Positive: **teamScoringAll**("*Philadelphia Eagles*", *38*)
    - Negative: **teamScoringAll** ("*Philadelphia Eagles*", *10*)
  - Features:
    - info on the entities in the relation
    - syntactic path between entities (both dependencies and constituents)
    - surface path between entities
    - entities between the relation elements

Basic NFL Inference

- Relation classifier is one-against-many
  - Can only predict one relation per pair of entities
- NFL domain often violates this!
  - **gameWinner**(*team, game*) ➔ **teamInGame**(*team, game*)
- System also doesn't understand domain semantics, e.g.:
  - Games have exactly one winner and one loser.
  - Teams with higher scores win.
- Simple logical rules fill in some of these cases.

Extraction results

System	Entity Mentions	Relation Mentions
Baseline	73.7	49.7
+ gazetteer features	74.0	50.2
+ rule-based model for `NFLTeam`	75.5	53.2
+ improved head finding	**76.1**	57.9
+ basic inference	**76.1**	**59.5**

Summary

- Work on TAC-KBP and MR-KBP
- Use Stanford CoreNLP! ☺

  `http://nlp.stanford.edu/software/corenlp.shtml`

- NFL system builds on top of CoreNLP

Thank you!

Questions?