Modularizing Natural Language Processing

AAAI 2020 Feb 8, 2020

Petuum

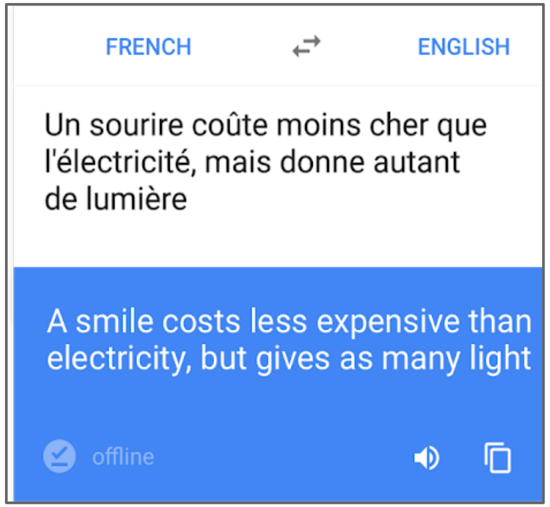
Carnegie Mellon University
School of Computer Science

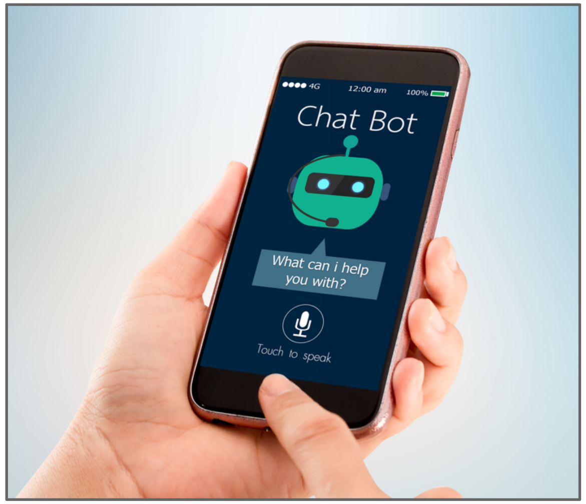

Hector Liu

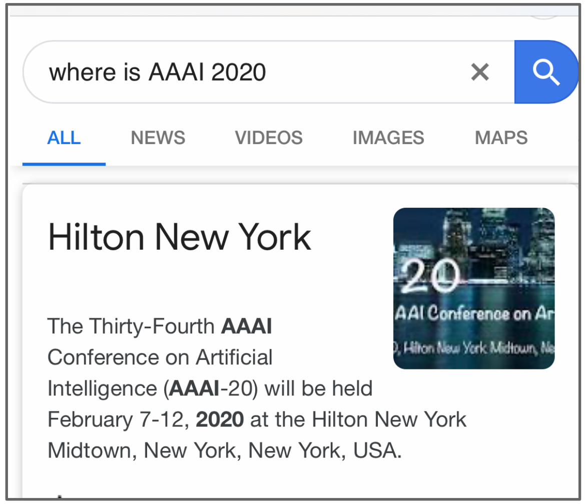

Zhiting Hu


Eric Xing

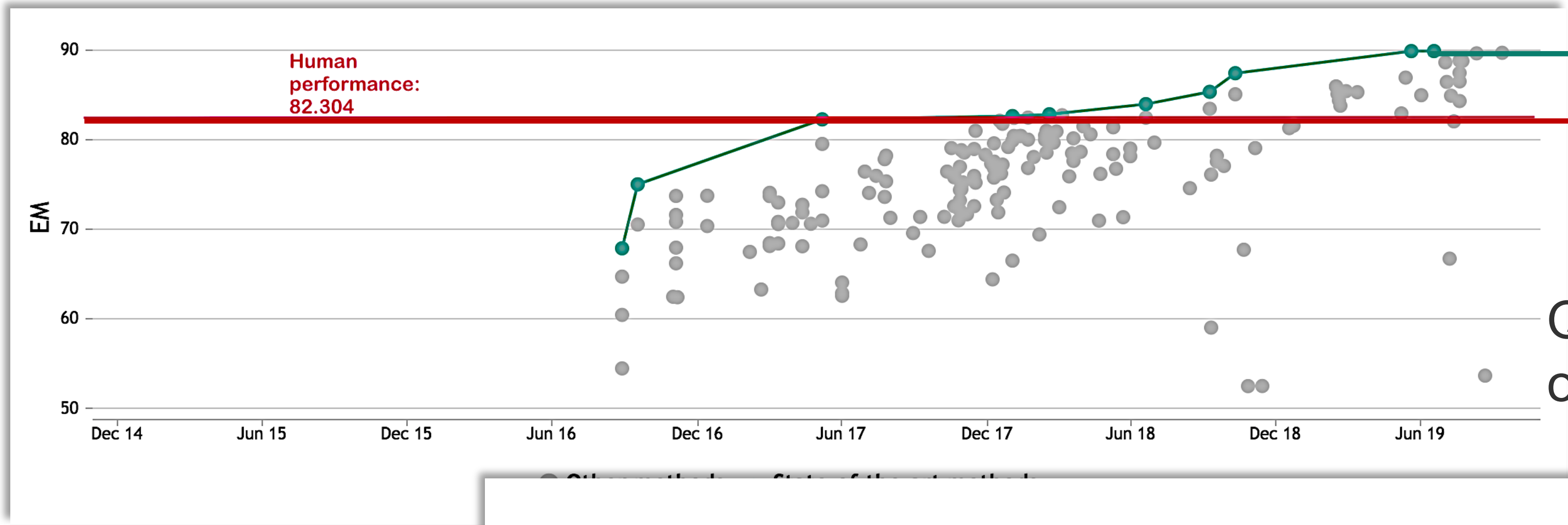
NLP Applications







Inspirational Success on Benchmarks



Model

Human

Question Answering on SQuAD

Microsoft Achieves Human Parity on Chinese-English Machine Translation

LIKE DISCUSS

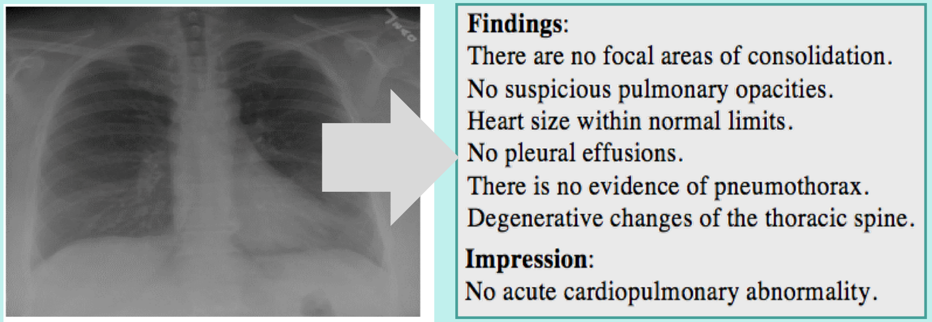
MAR 15, 2018 • 2 MIN READ

NLP in Real-world Context

The Healthcare Industry

Building a ready-to-use AI solution for this is

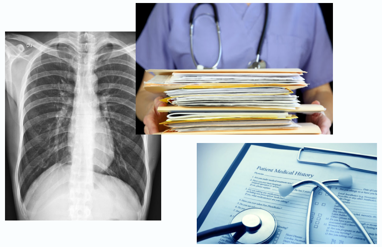
Extremely complex



Task: Automatic Medical Report Generation

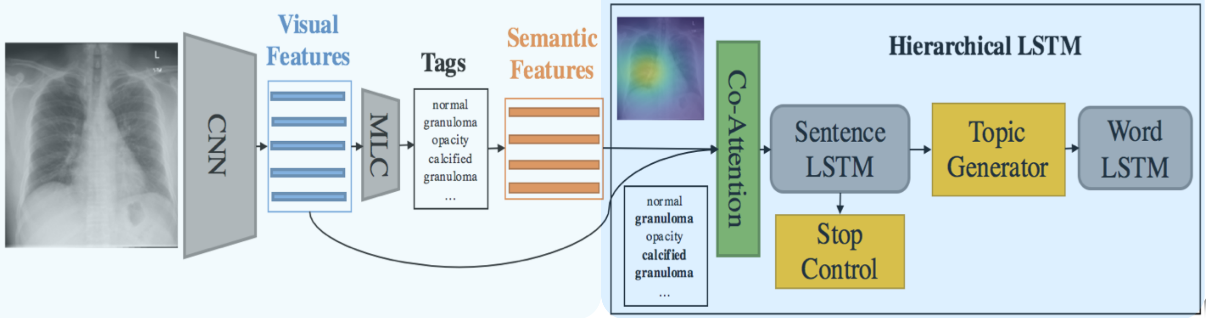
Requires inter-operation between diverse components

Raw Data Cleansing

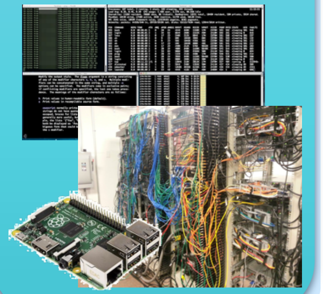


Data Enrichment

Model/Algorithm



System/Infra

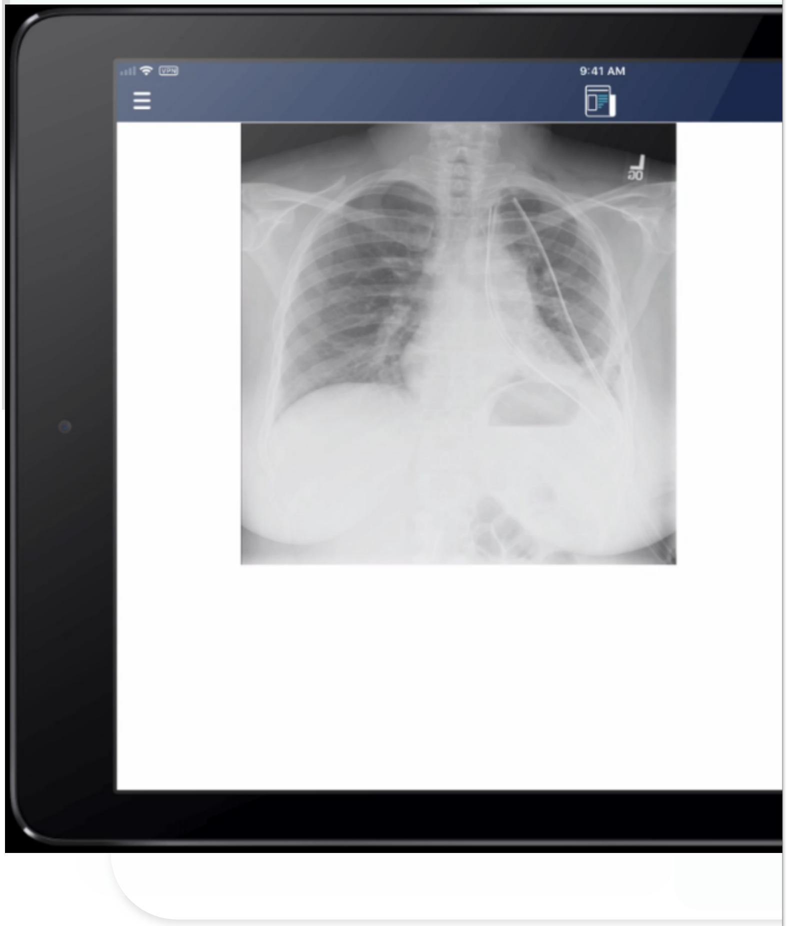


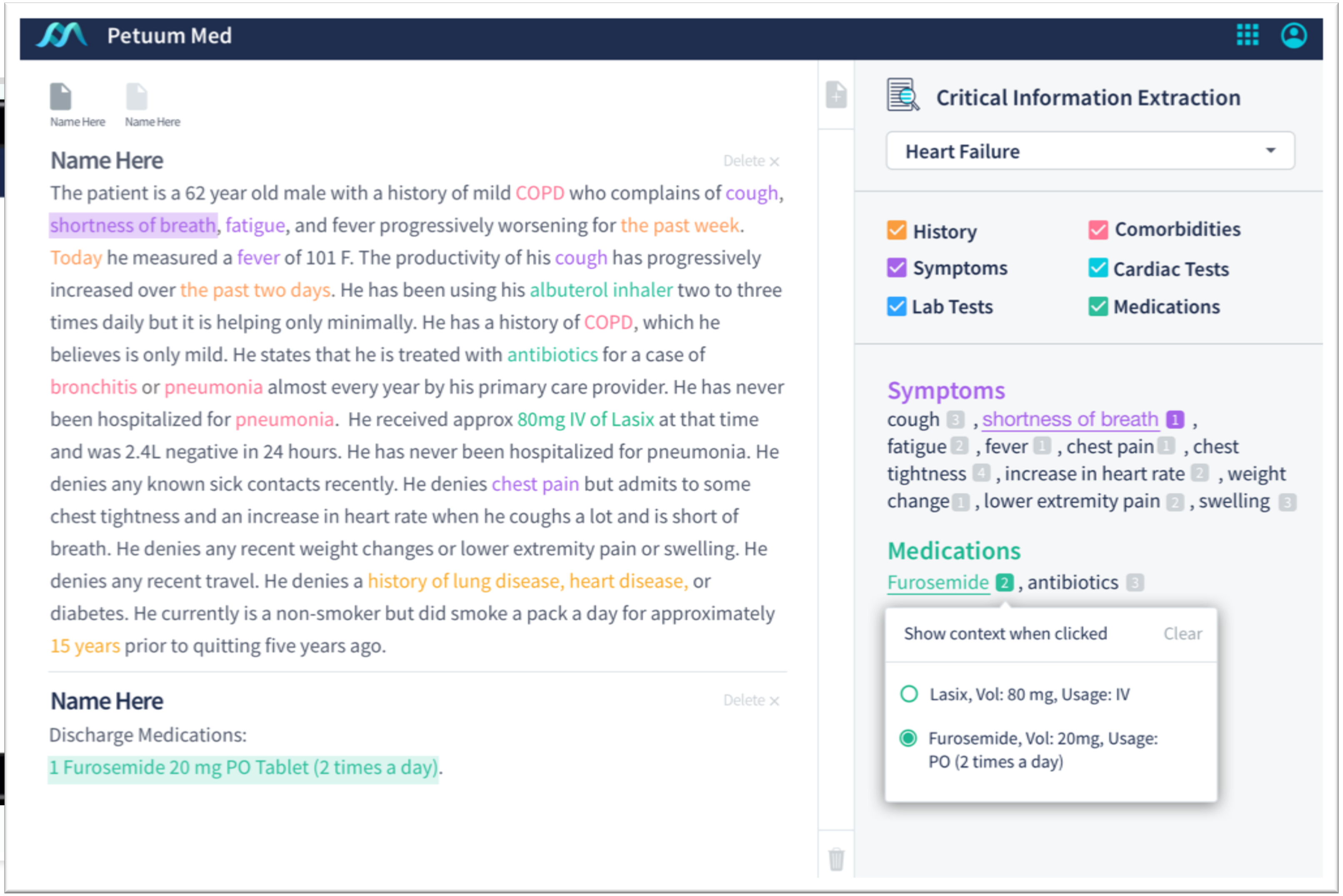
Petuum

Carnegie Mellon University
School of Computer Science

Building a ready-to-use AI solution for this is

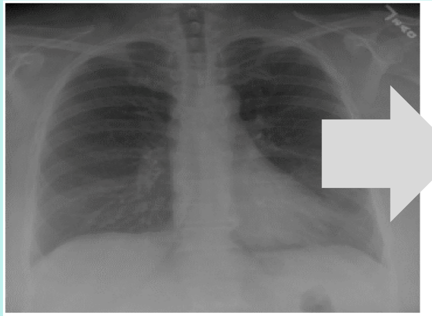
User interface for Doctors





Building a ready-to-use AI solution for this is

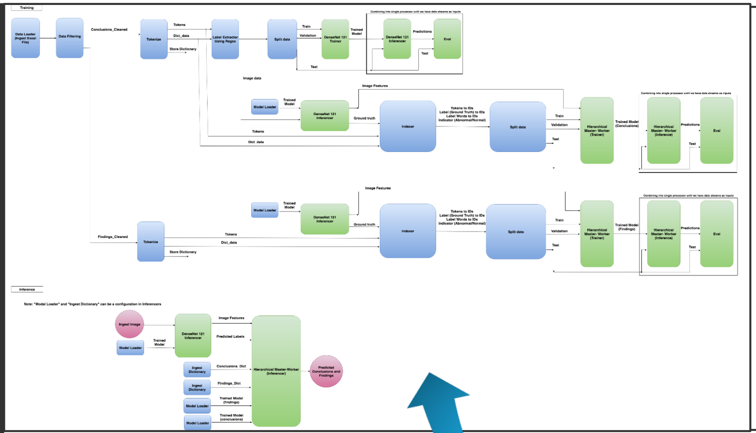
Extremely complex

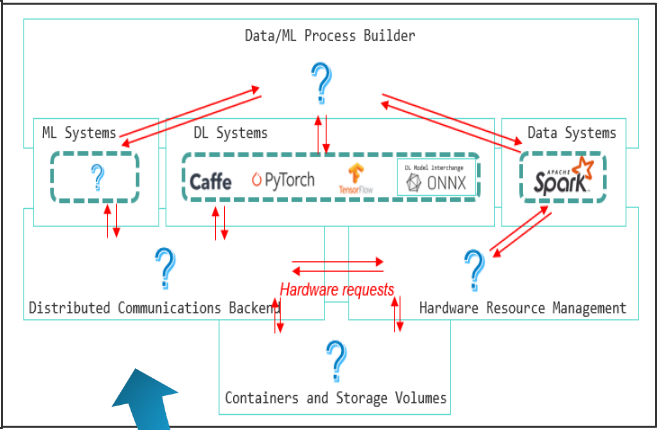


Findings:
There are no focal areas of consolidation.
No suspicious pulmonary opacities.
Heart size within normal limits.
No pleural effusions.
There is no evidence of pneumothorax.
Degenerative changes of the thoracic spine.

Impression:
No acute cardiopulmonary abnormality.

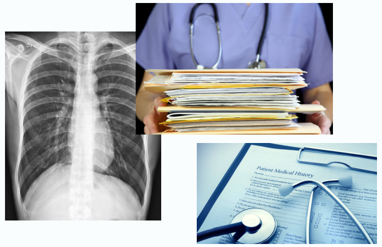
Task: Automatic Medical Report Generation





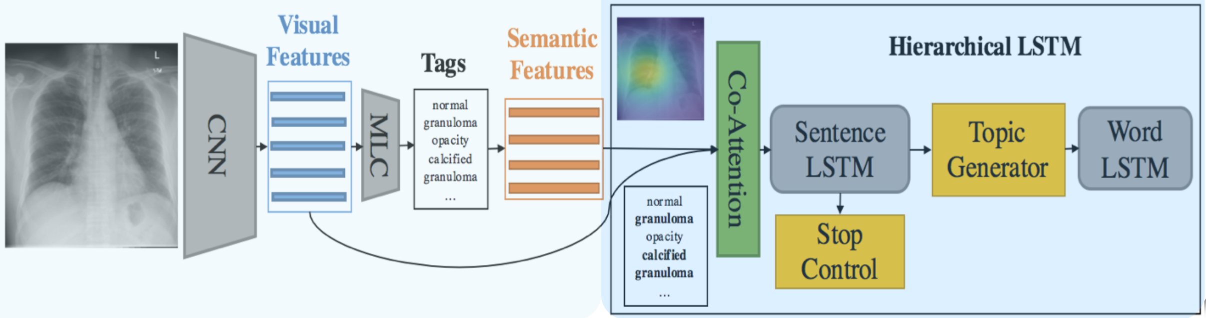
Requires inter-operation between diverse components

Raw Data Cleansing

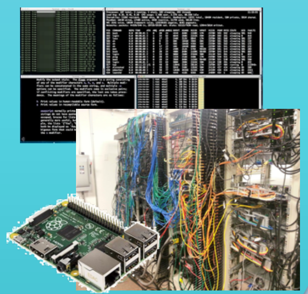


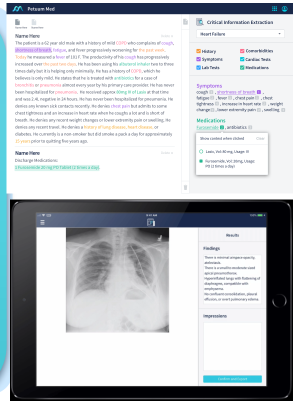
Data Enrichment

Model/Algorithm



System/Infra



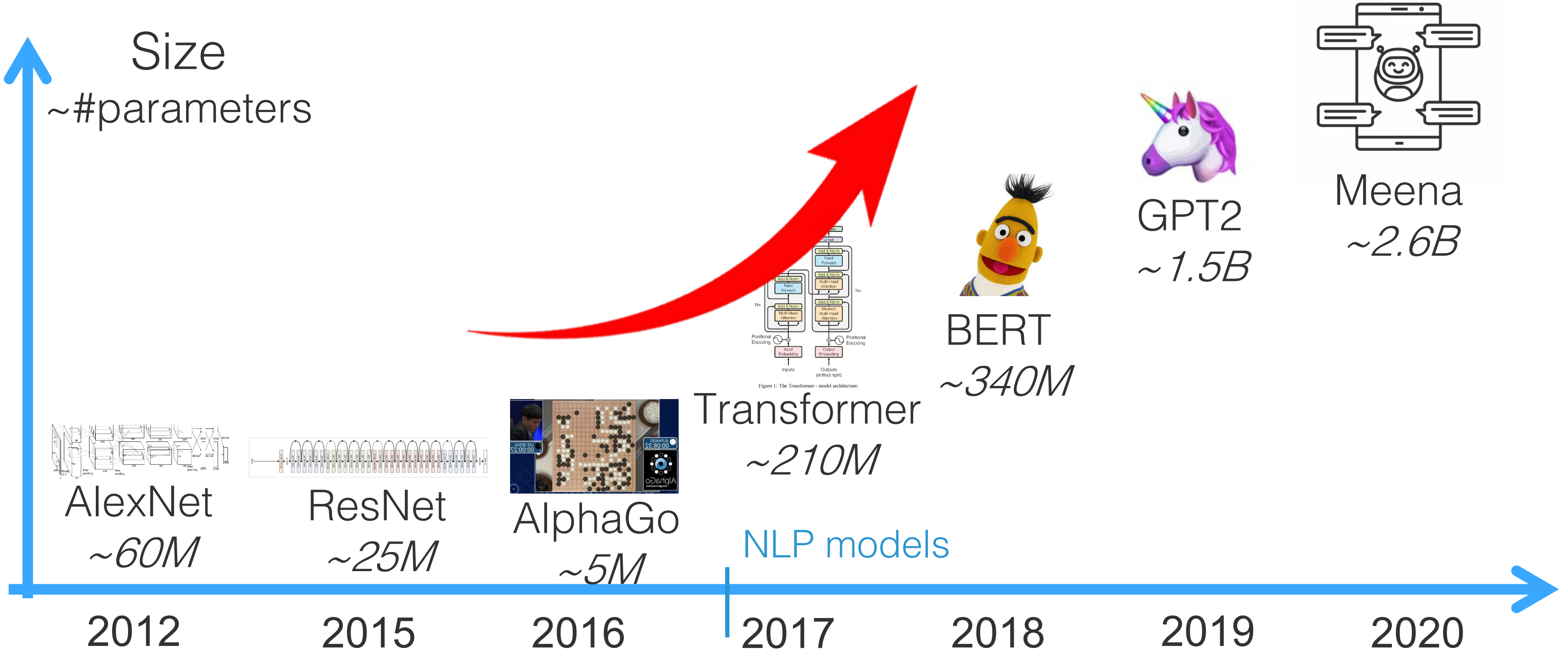


Petuum

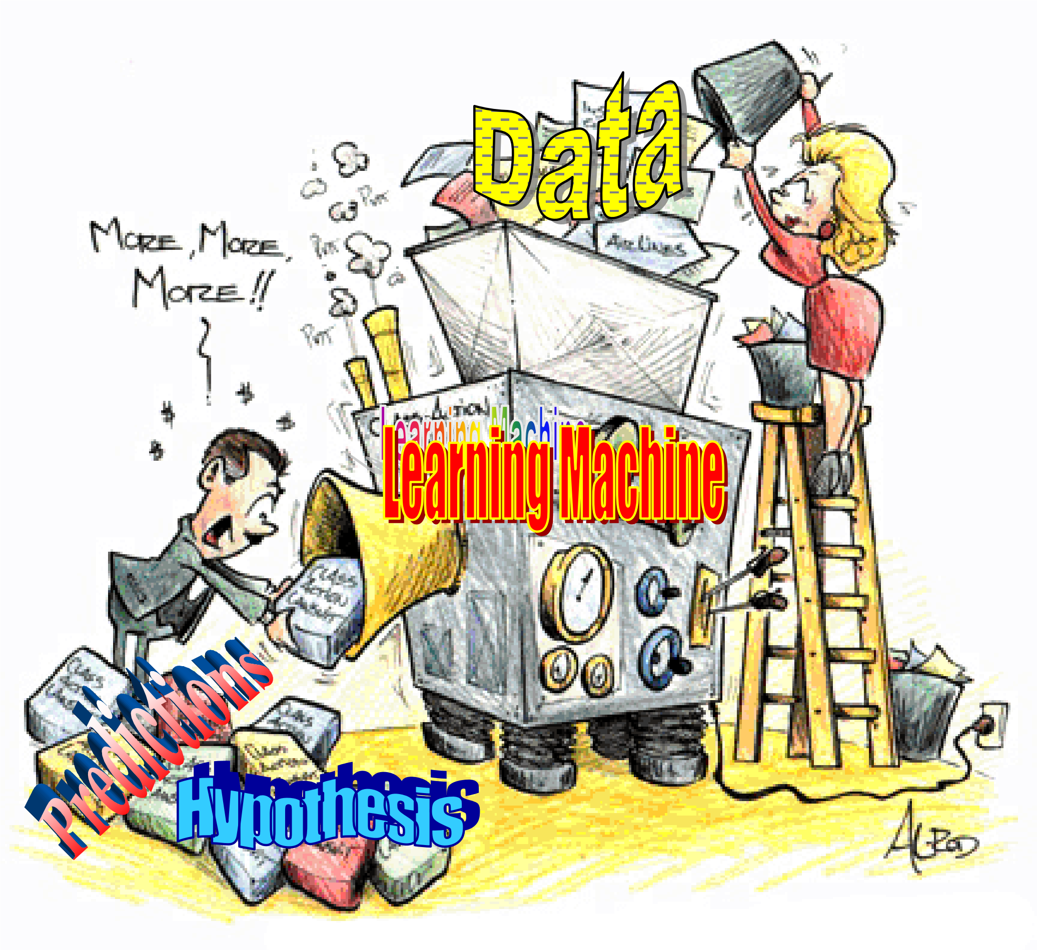
Carnegie Mellon University
School of Computer Science

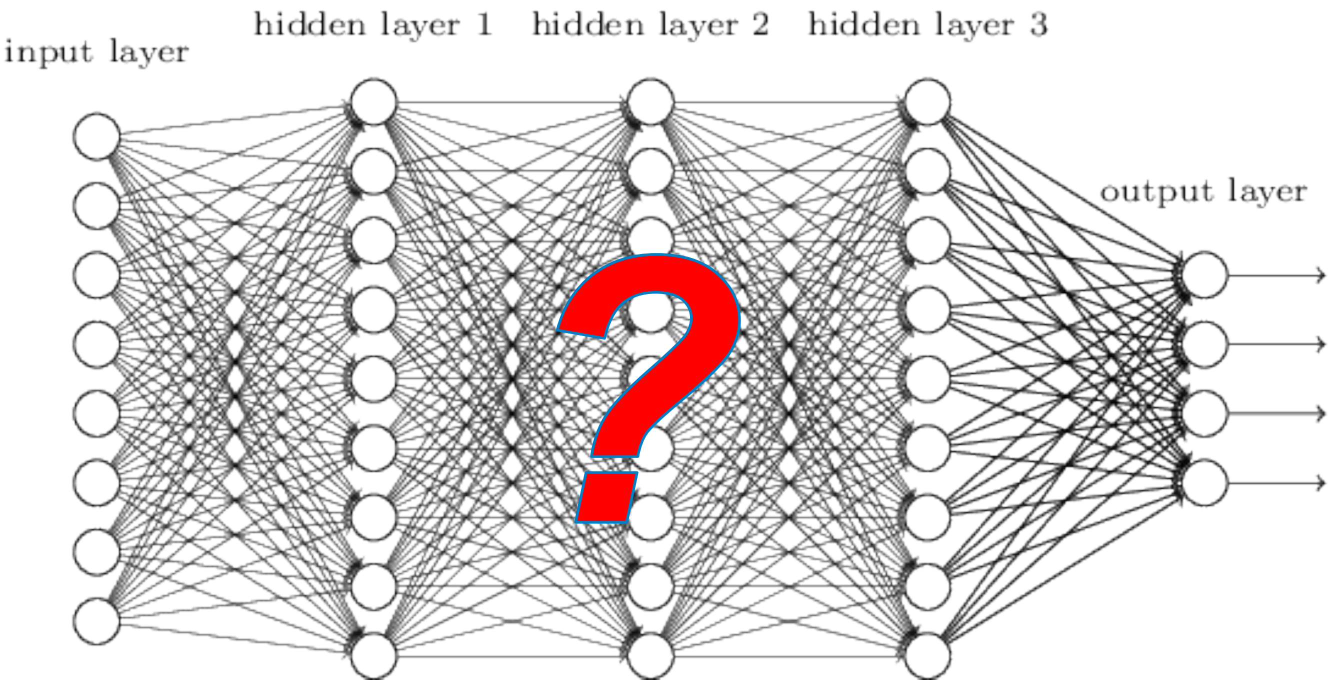
Single Models with Increasing Size and Performance

- Increasingly large black-box neural networks
- Good, even super-human performance on some tasks

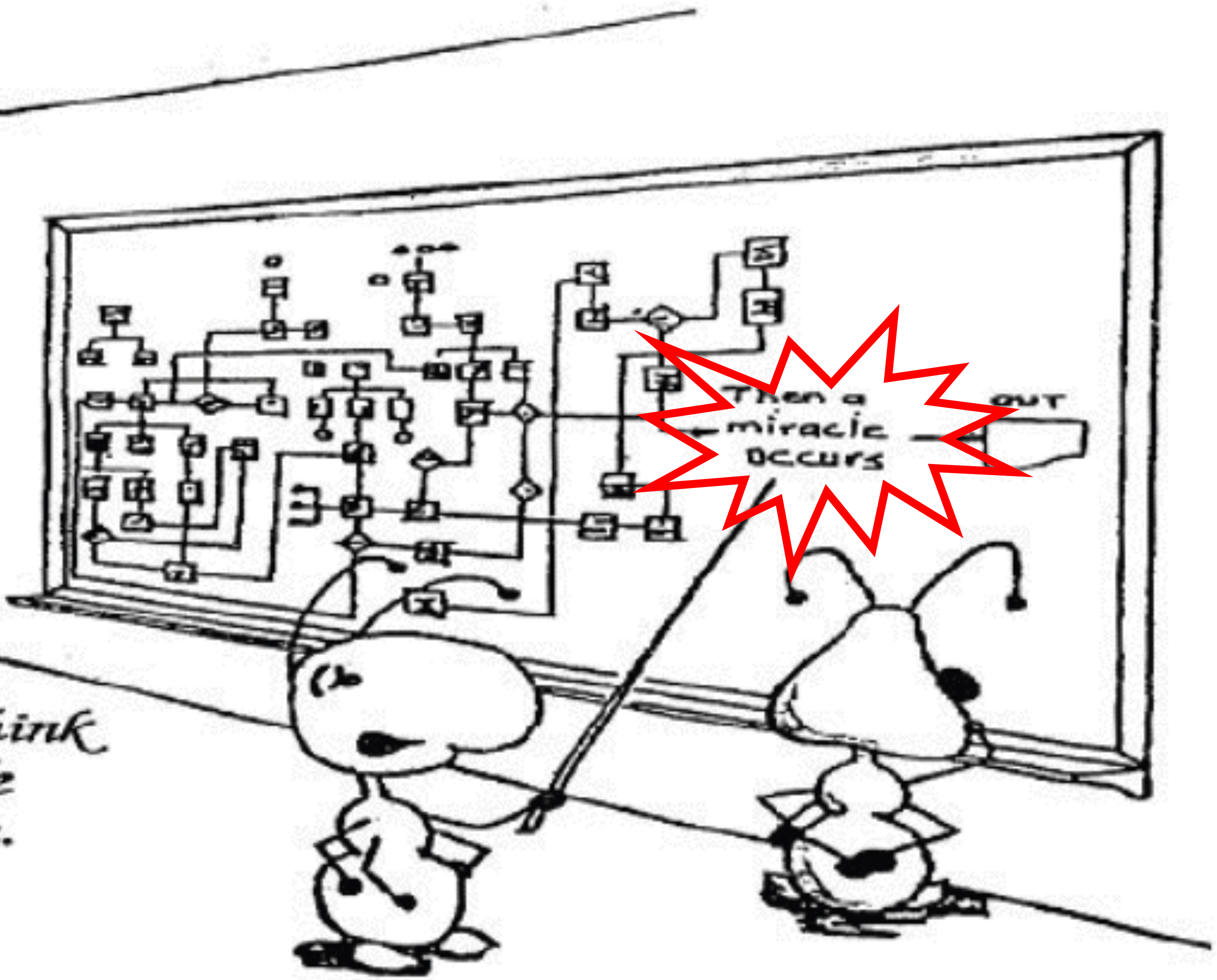


Single Giant Models Enough?





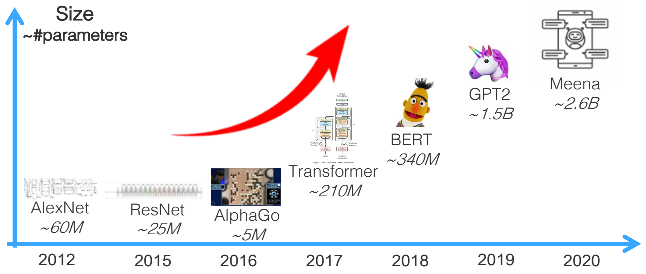
Difficulties of Single Giant Models

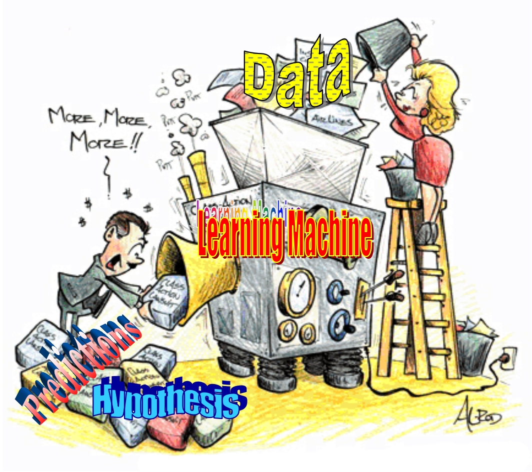


Good work -- but I think we might need a little more detail right here.

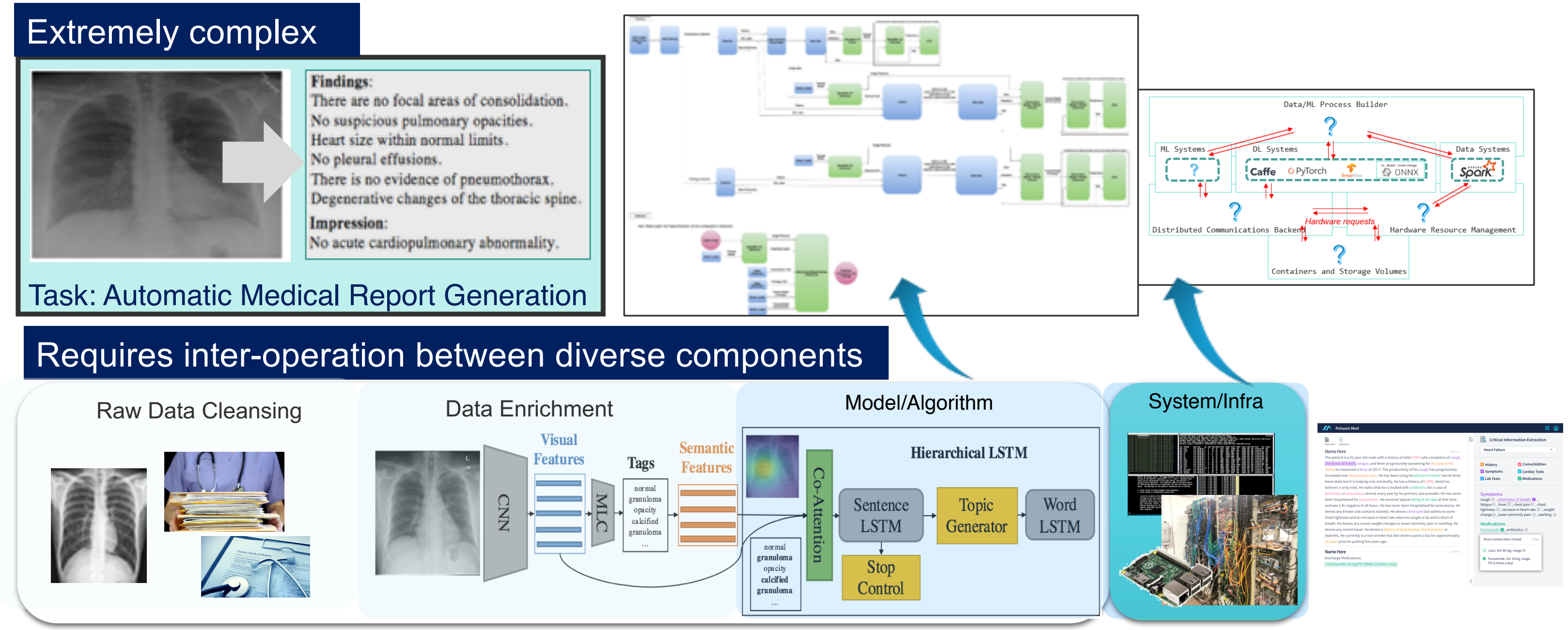
- Explainability
- Debugging
- Maintenance
- Upgrade
- Scalability
- …

Far from Solving Real Complex Problems





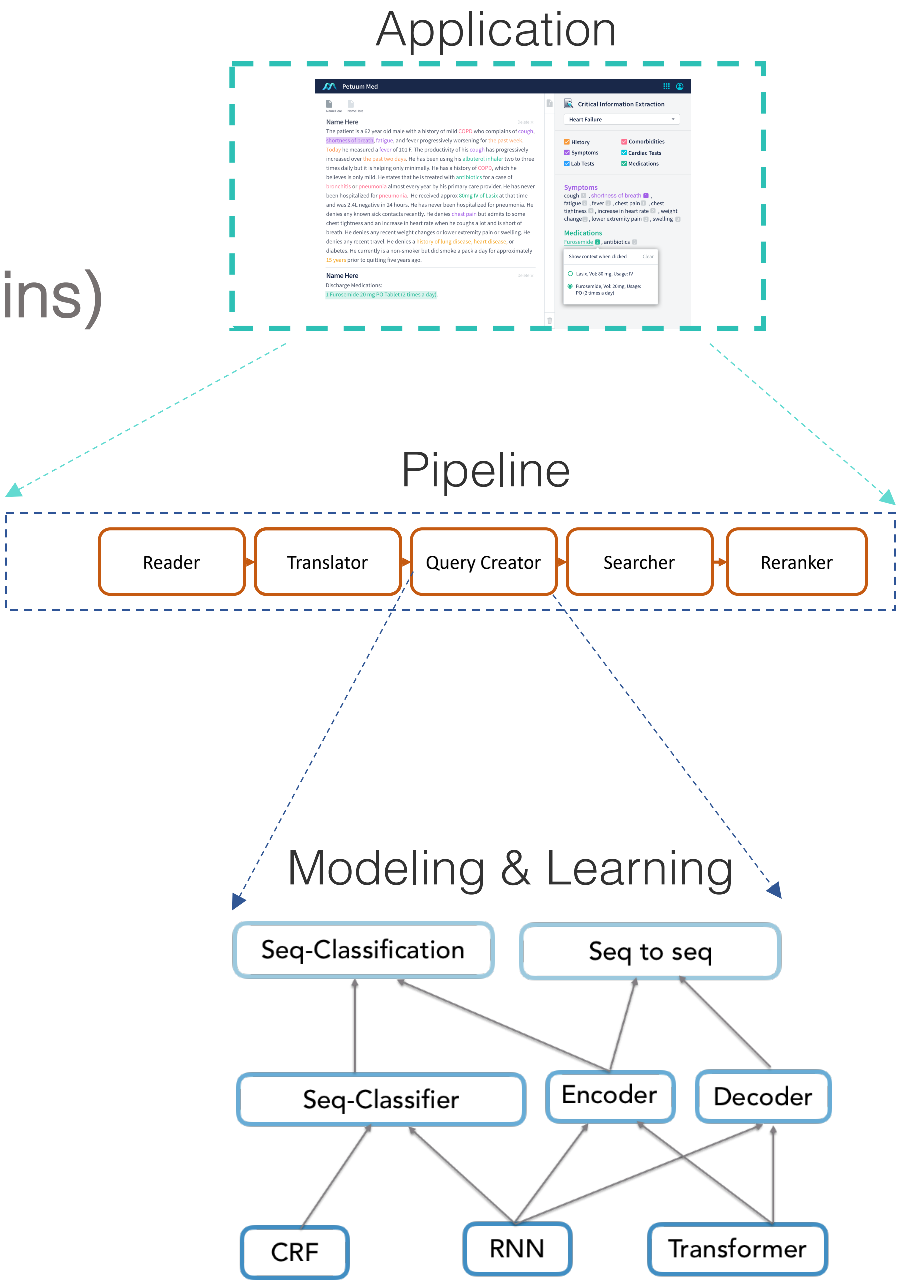
V.S.



Modularization & Standardization

Agenda

- Natural Language Processing Overview (10mins)
- Modularizing NLP Pipeline (40mins)
  - Complexity of NLP pipeline
  - A standardized view of NLP pipeline
  - A standardized implementation of NLP pipeline
- Modularizing NLP Model & Learning (30mins)
  - Composable ML
- QA (10mins)



Agenda

- Natural Language Processing Overview (10mins)
- Modularizing NLP Pipeline (40mins)
  - Complexity of NLP pipeline
  - A standardized view of NLP pipeline
  - A standardized implementation of NLP pipeline
- Modularizing NLP Model & Learning (30mins)
  - Composable ML
- QA (10mins)

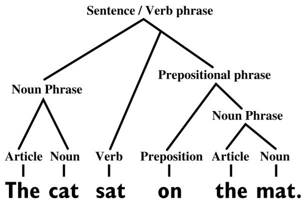
What's in NLP?

- Informally, NLP is about interacting with or processing human languages with computers
- Broadly, NLP contains/relates to many fields
  - Speech Processing
  - Information Retrieval
  - Information Extraction
  - Text Analysis
  - Language Generation
  - Speech Synthesize

Text Processing

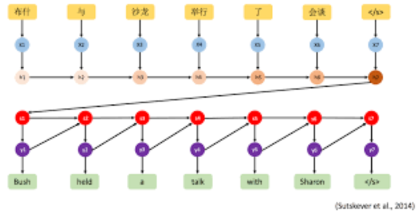
We'll talk about these today.

Categorization of Text Processing



Language Understanding

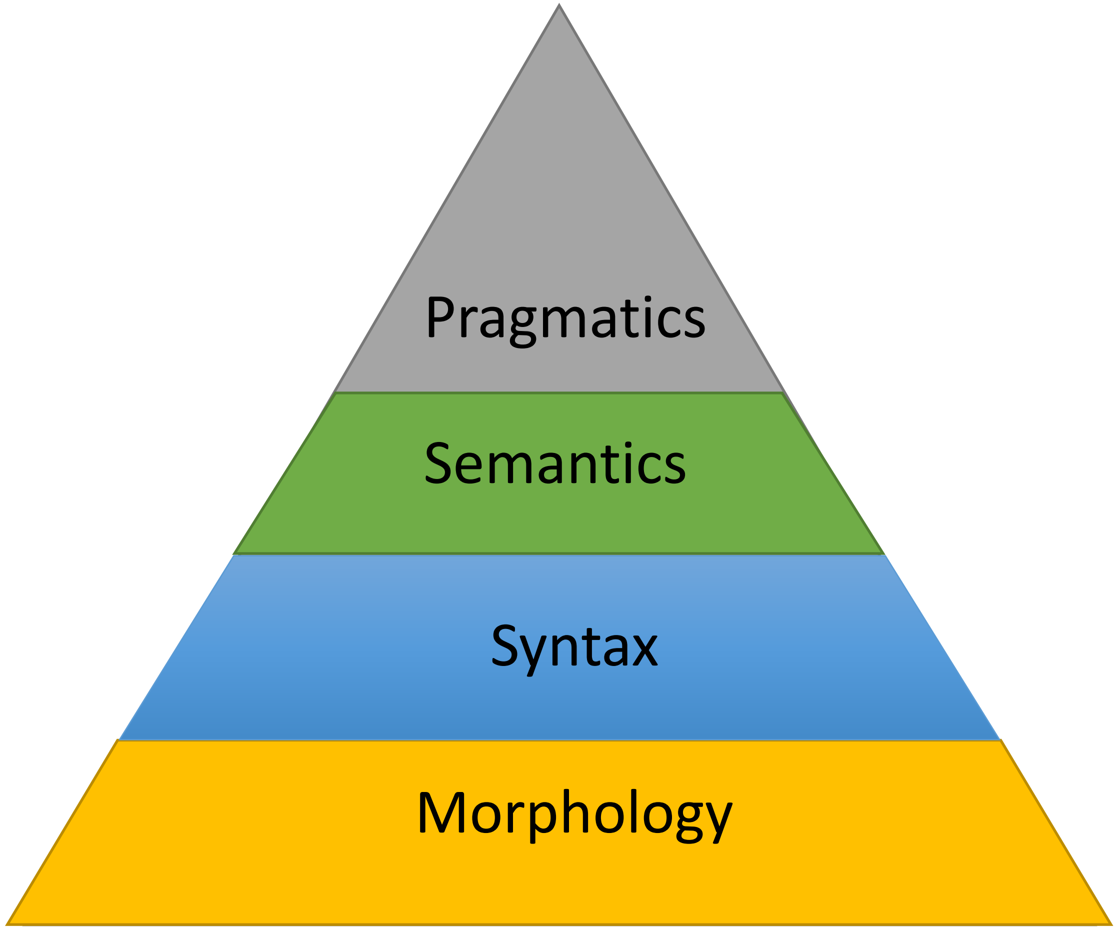
Text Retrieval



Language Generation

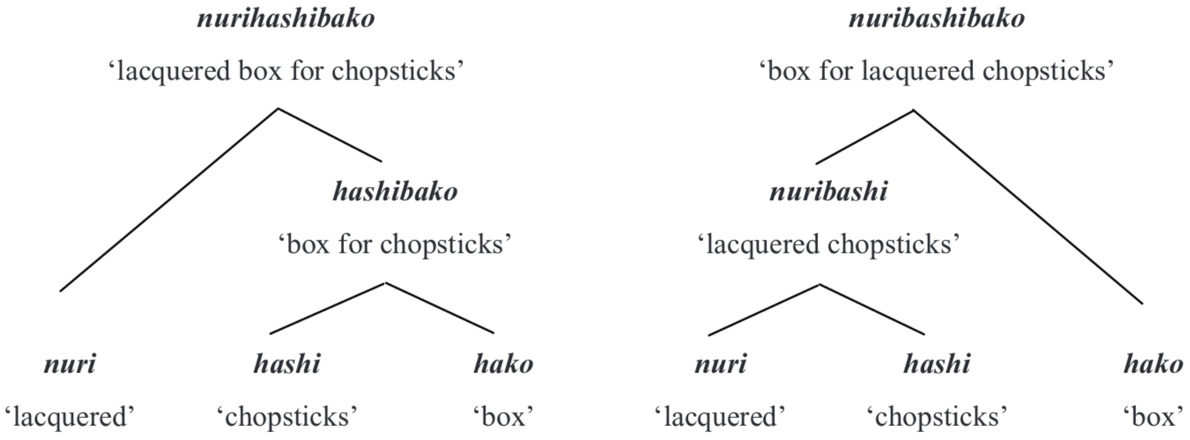
Many different levels of tasks

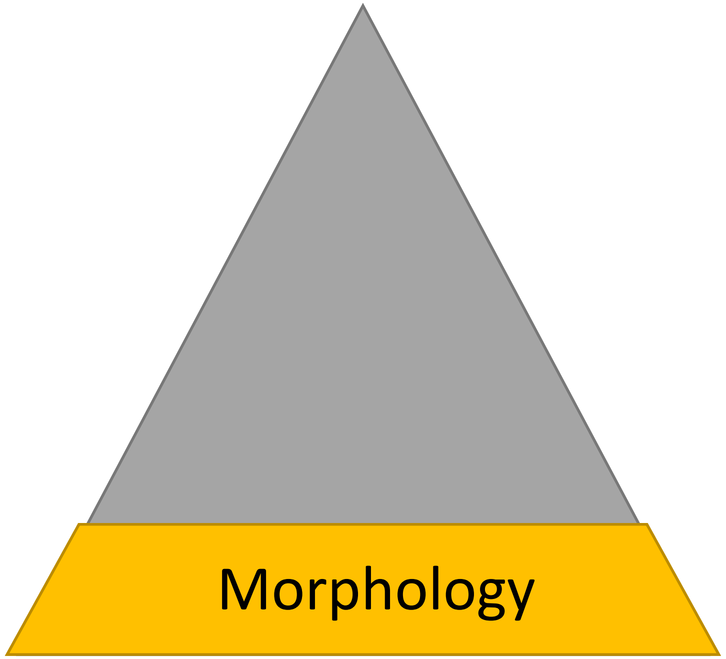
Language Understanding Pyramid



Morphology

Morphology Analysis





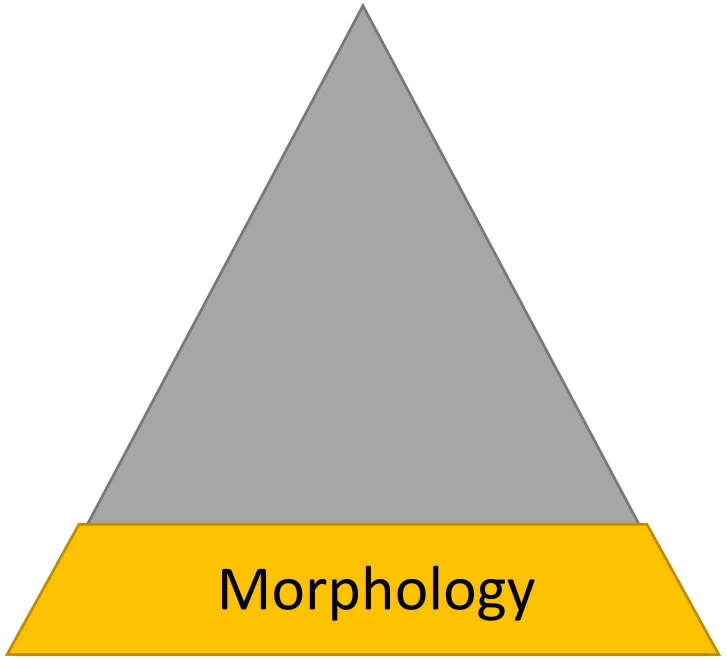
Morphology

Stemming

adjustable → adjust_
formality → formaliti
formaliti → formal
airliner → airlin_

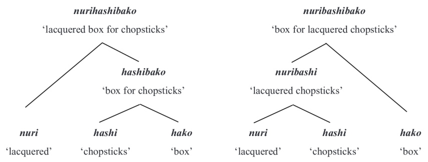
Lemmatize

was → (to) be
better → good
meeting → meeting

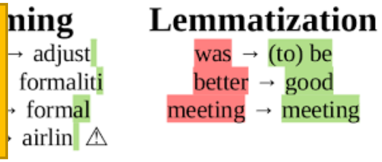


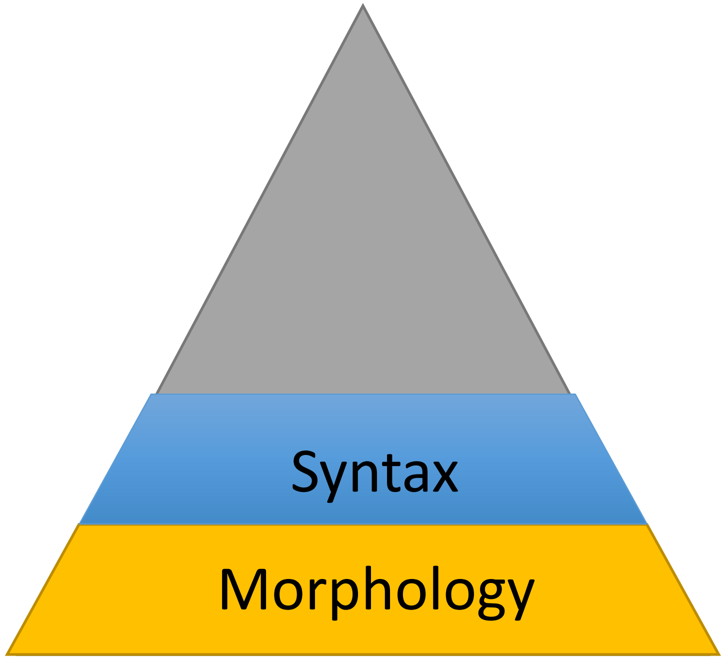
Language Understanding Pyramid

Morphology



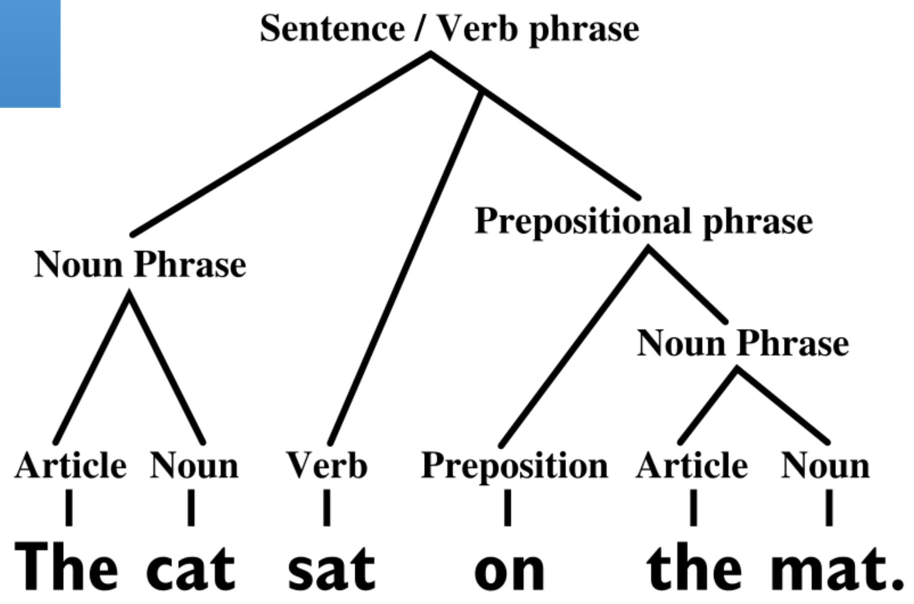
Lemmatize

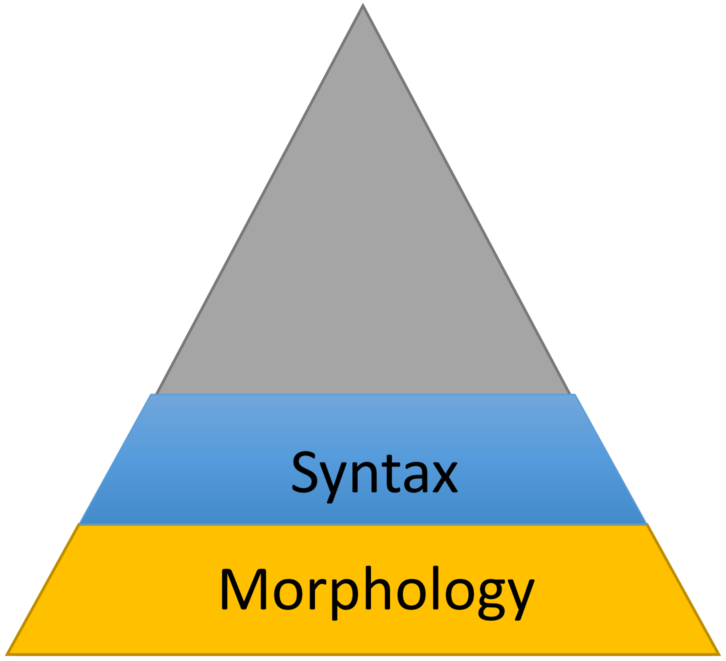




Syntax

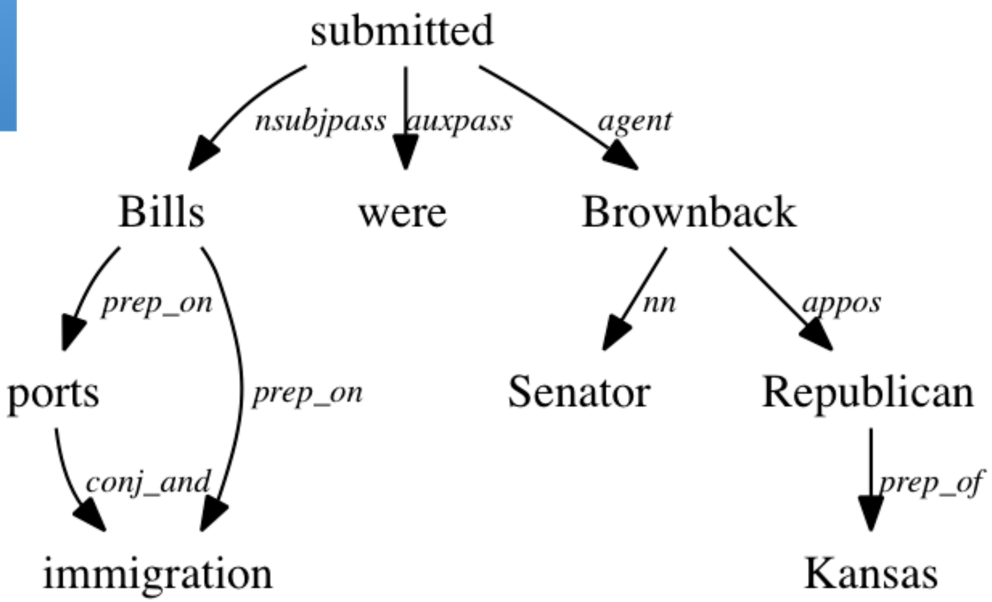
Constituent Parse

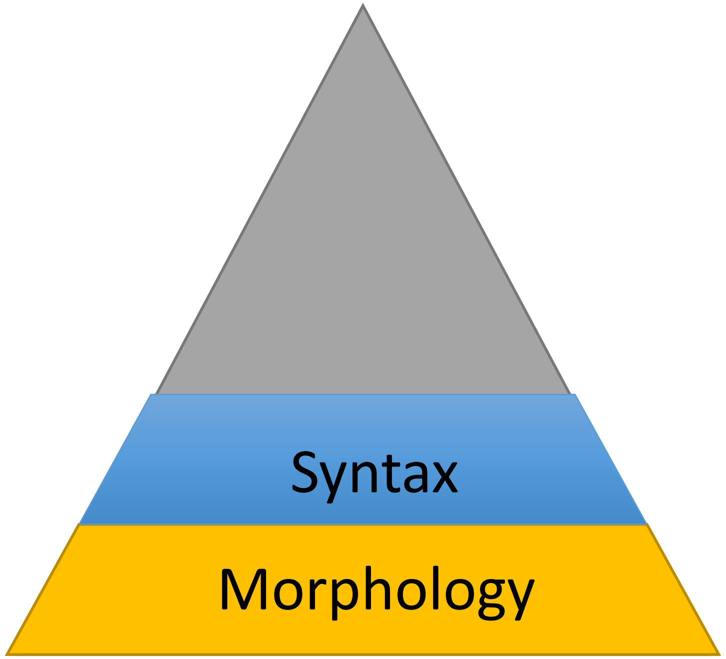




Syntax

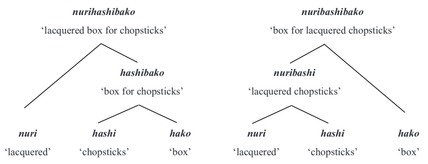
Dependency Parse



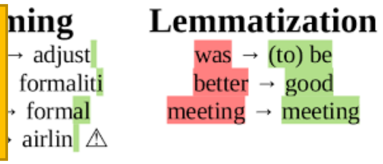


Language Understanding Pyramid

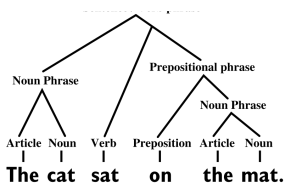
Morphology



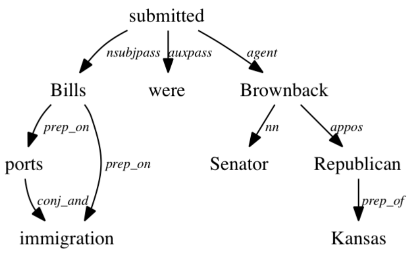
Lemmatize

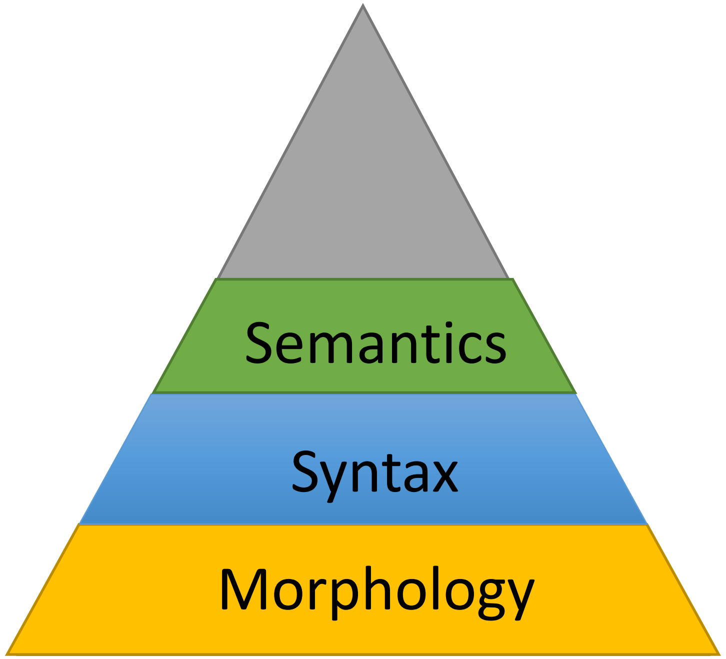


Constituent Parse



Dependency Parse



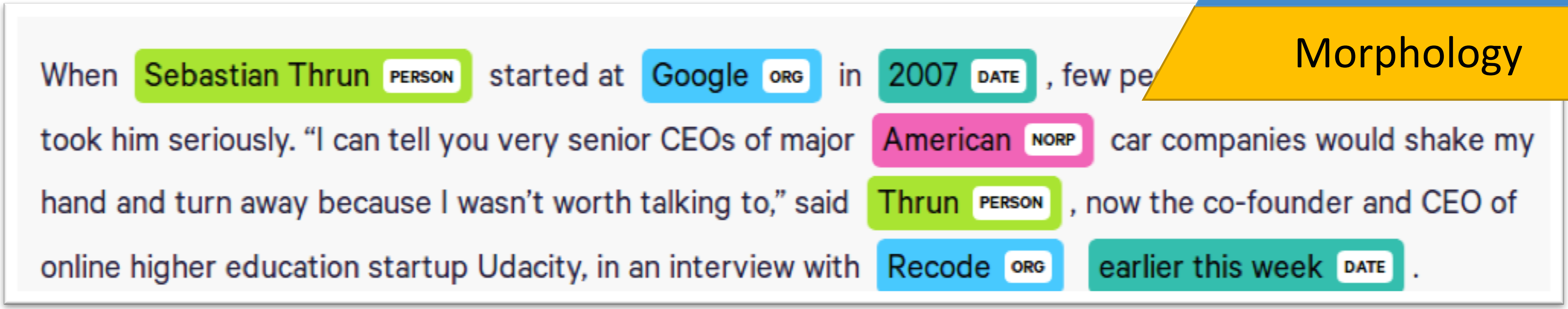


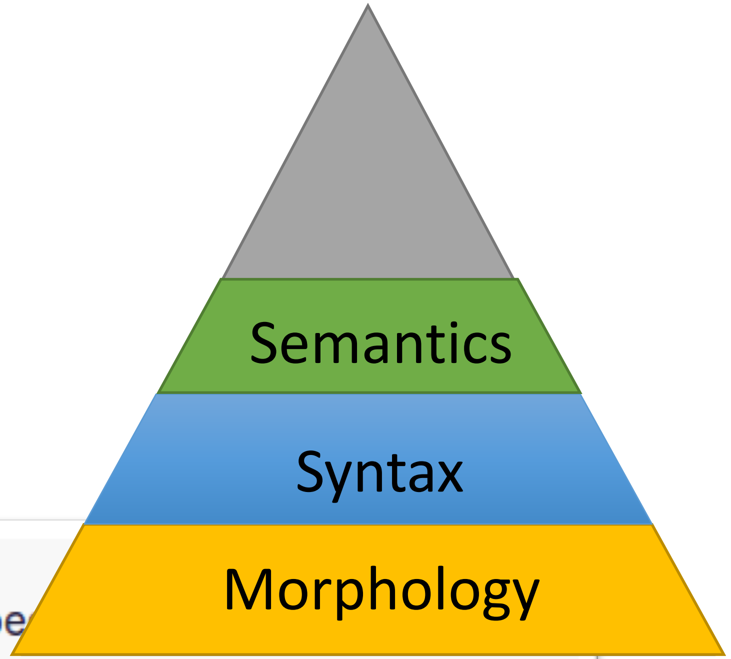
Petuum

Carnegie Mellon University
School of Computer Science

Semantic

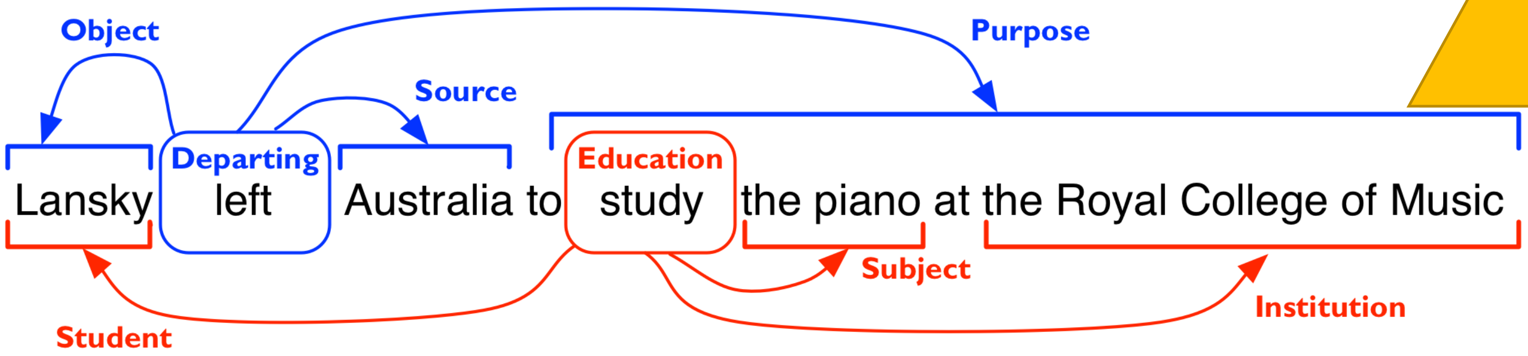
Named Entity Recognition

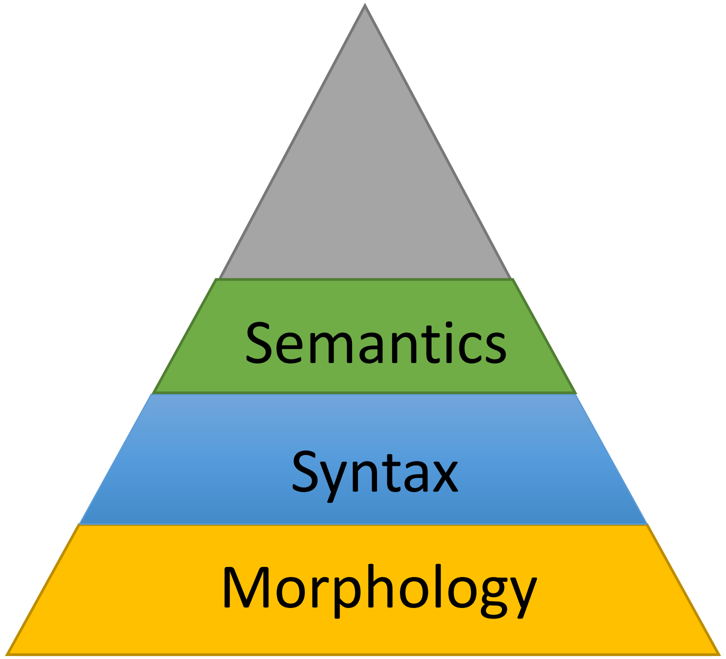




Semantic

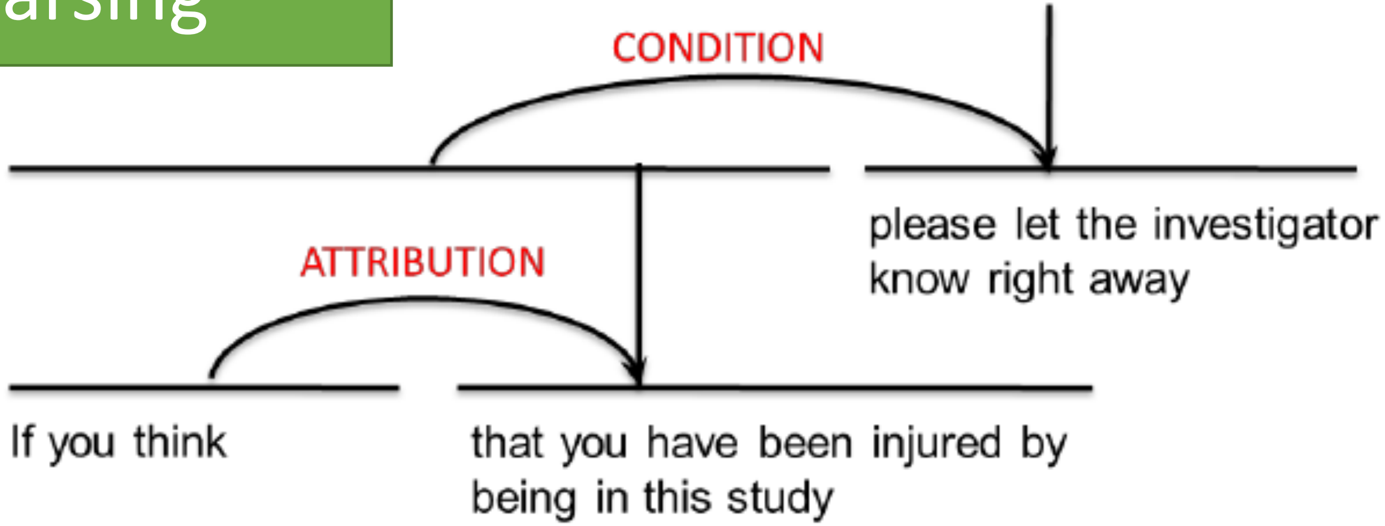
Semantic Roles

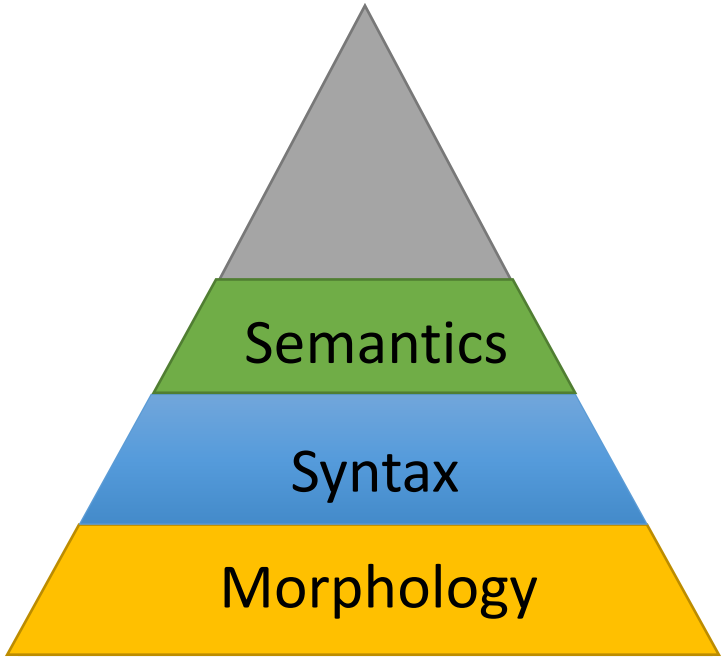




Semantic

Discourse Parsing

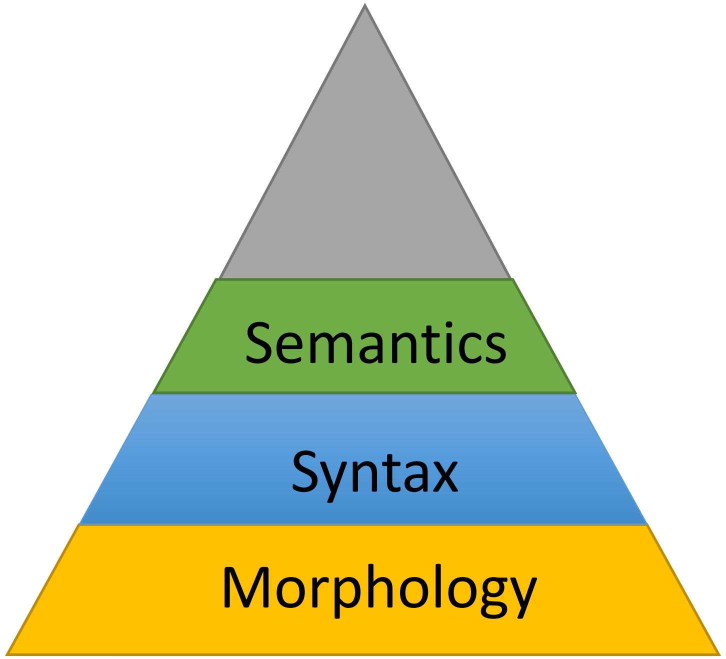




Semantic

Coreference

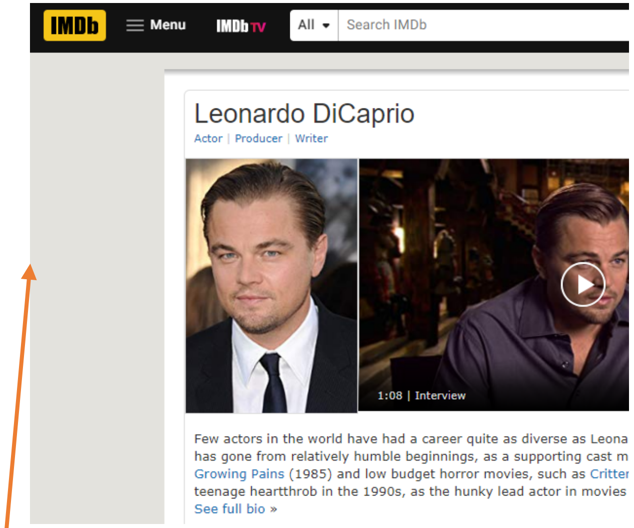
"I voted for Nader because he was most aligned with my values," she said.

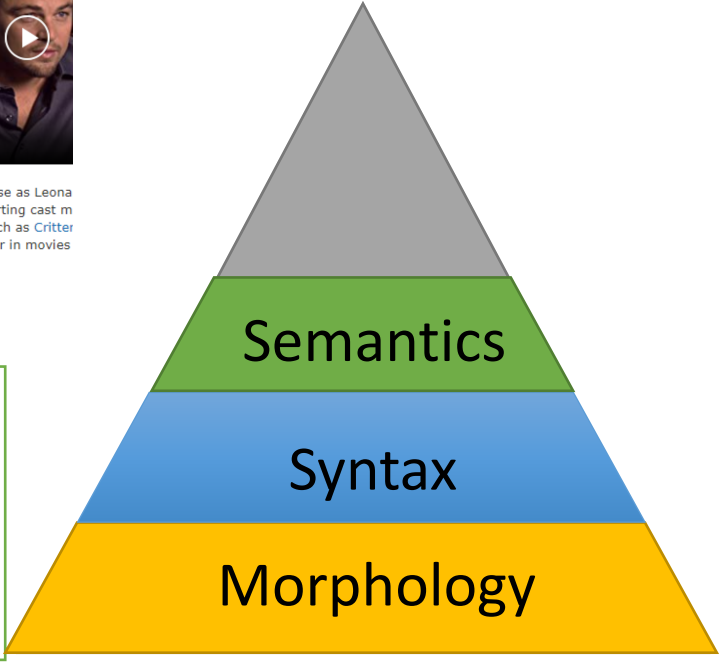


Semantic

Entity Linking

Kate Winslet and Leonardo Dicaprio have definitely created a timeless classic.



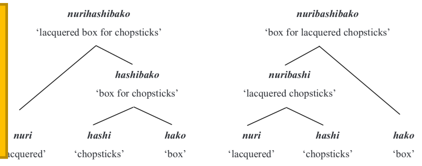


Petuum

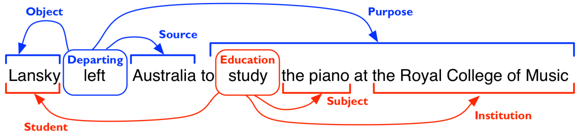
Carnegie Mellon University
School of Computer Science

Language Understanding Pyramid

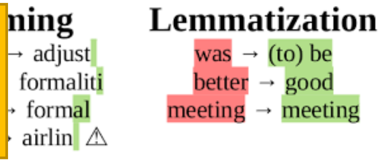
Morphology



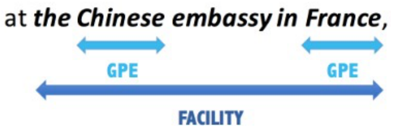
Semantic Parsing



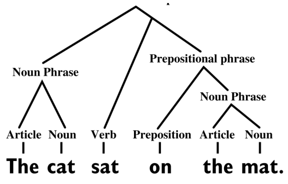
Lemmatize



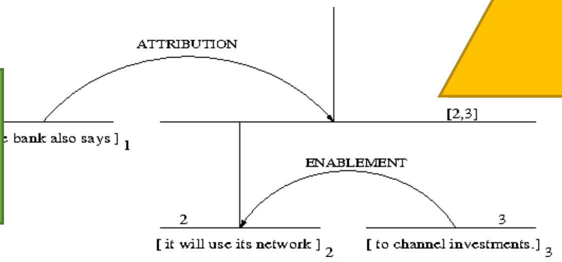
NER



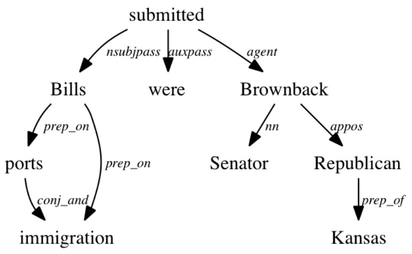
Constituent Parse



Discourse Parsing

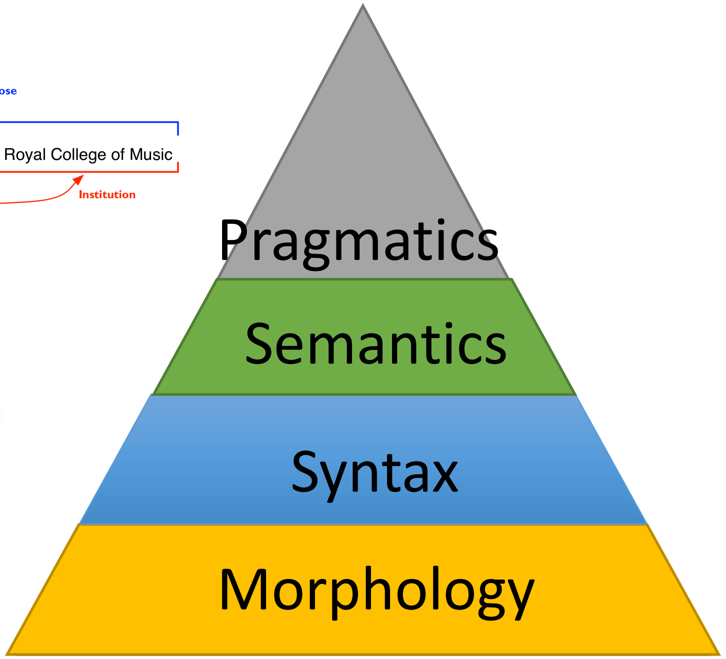


Dependency Parse



Coreference

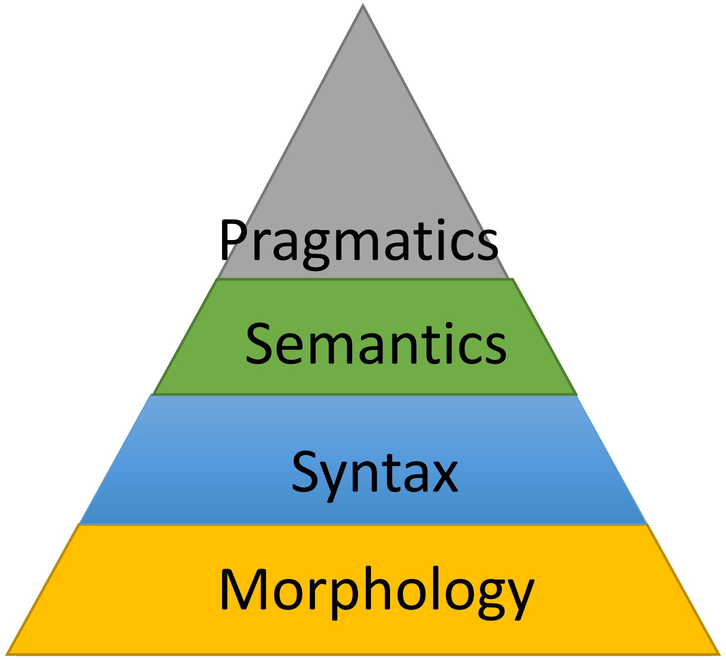
"I voted for Nader because he was most aligned with my values," she said.



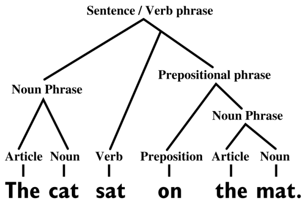
Entity Linking

Pragmatics


CECILIA,
CAN YOU STOP BY MY OFFICE? WE NEED TO TALK.
- PROF. JONES
"WE NEED TO TALK"
WHAT IT CAN MEAN:
A) YOU'RE FIRED.
B) I FORGOT WHAT YOUR PROJECT IS ABOUT. CAN YOU REMIND ME?
C) I HAVE A TRIVIAL REQUEST THAT I'M TOO EMBARRASSED TO WRITE IN AN EMAIL.
JORGE CHAM © 2013
OH, SO THAT'S HOW YOU CHANGE THE LOGO!
I AM BOTH RELIEVED AND ANNOYED, PROF. JONES!
WWW.PHDCOMICS.COM

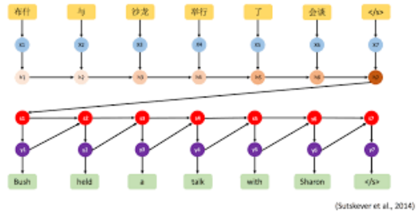

Pragmatics
Semantics
Syntax
Morphology

Categorization of Text Processing



Language Understanding

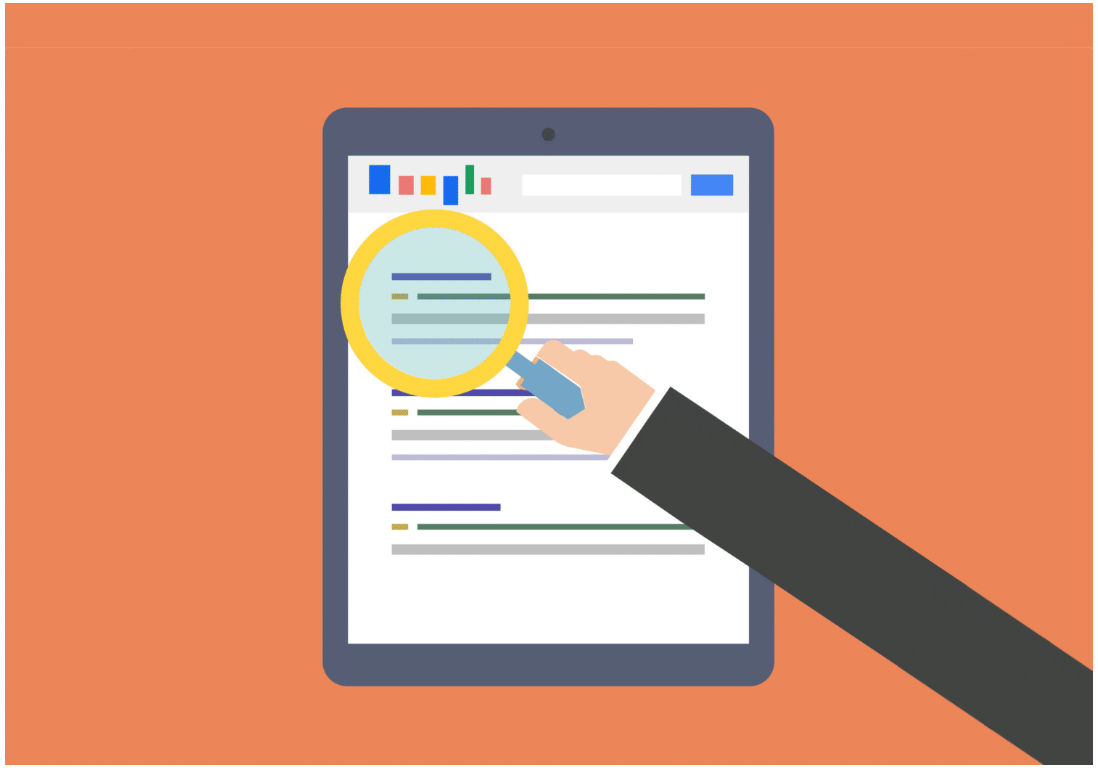
Text Retrieval



Language Generation

Information Retrieval

- Retrieve relevant documents based on user query
- Some IR subs-tasks:
  - The Search Step: quickly get relevant documents as a rank list based on an efficient Index.
  - The Reranking Step: fine-tune the rank list to create better ranking



Text Generation

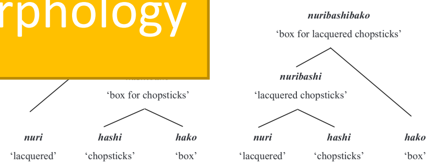
- Machine Translation
- Summarization
- Dialogue Response Generation

Agenda

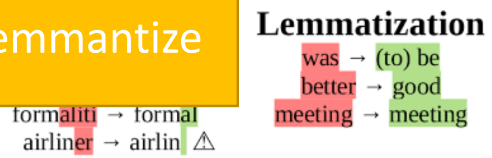
- Natural Language Processing Overview (10mins)
- Modularizing NLP Pipeline (40mins)
    - Complexity of NLP pipeline
    - A standardized view of NLP pipeline
    - A standardized implementation of NLP pipeline
- Modularizing NLP Model & Learning (30mins)
    - Composable ML
- QA (10mins)

NLP tasks are Complex

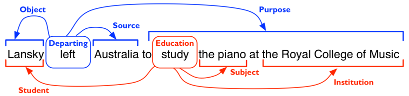
Morphology



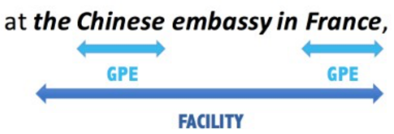
Lemmantize



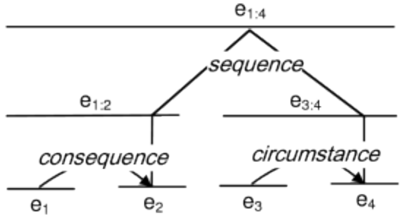
Semantic Parsing



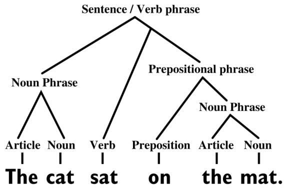
NER



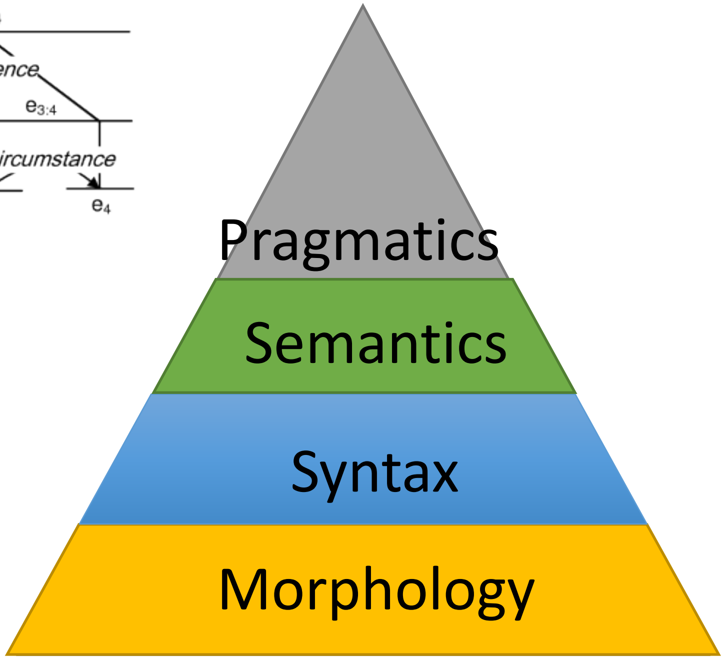
Discourse Parsing



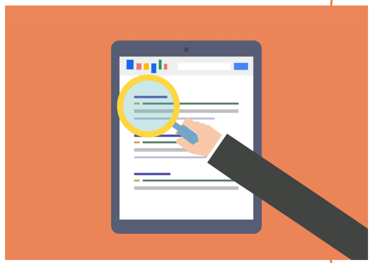
Syntax Parse



Many Different Tasks



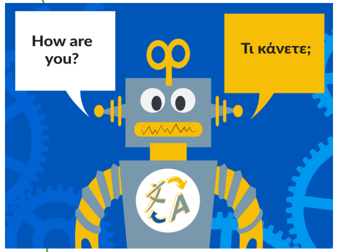
Different Levels



Retrieval

Understand

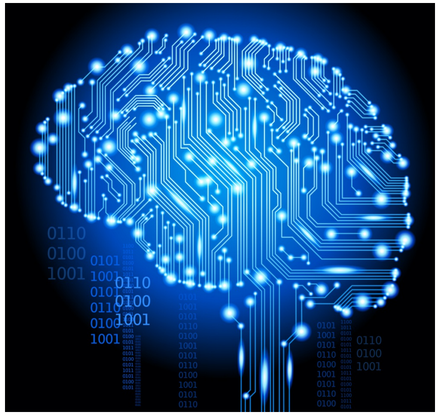
Generation

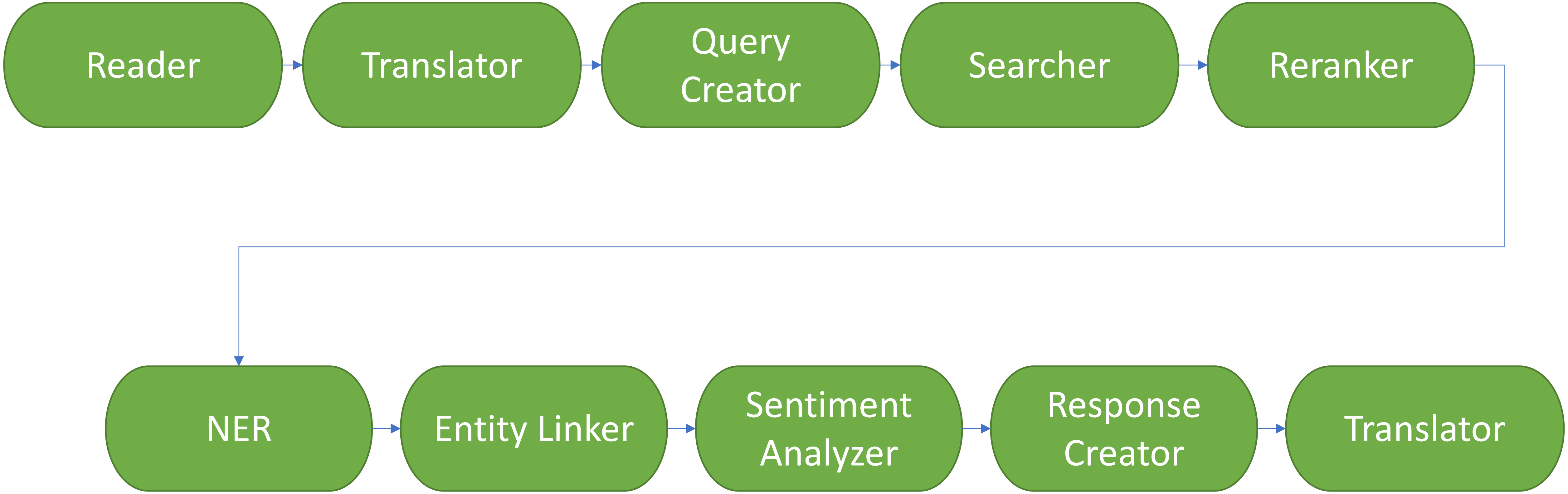


Multiple Stages

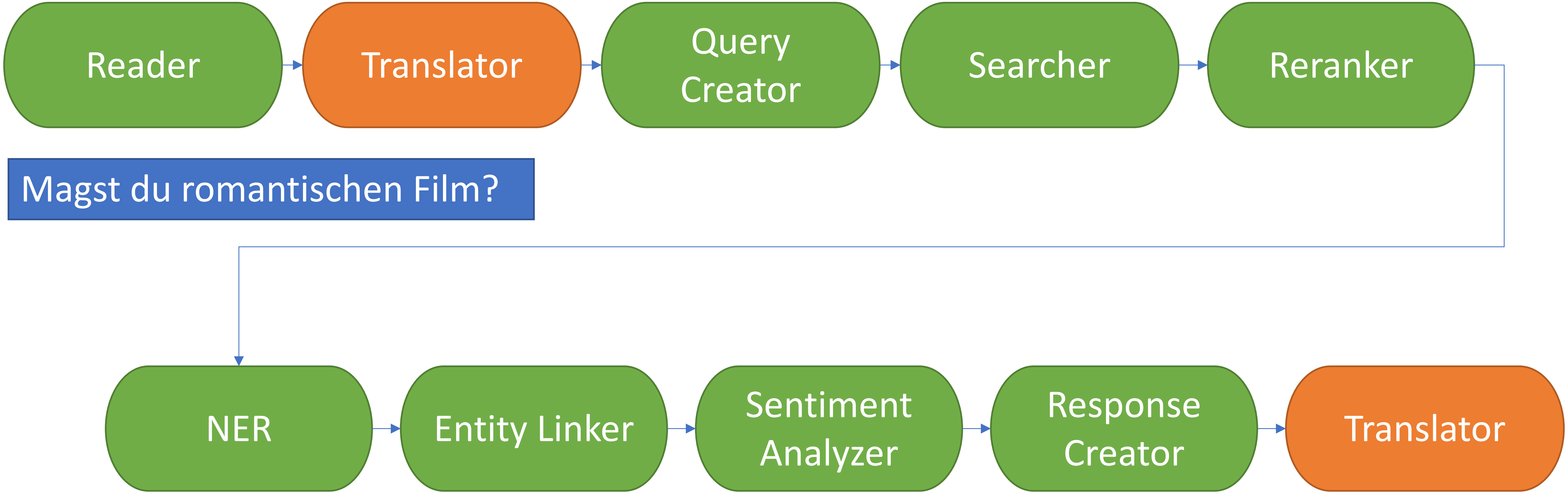
Real World Applications are Complex

- A user **speaks German** but would like to find good **romantic movies.**
- We have a **corpus of English movie reviews.**
- What should we do?

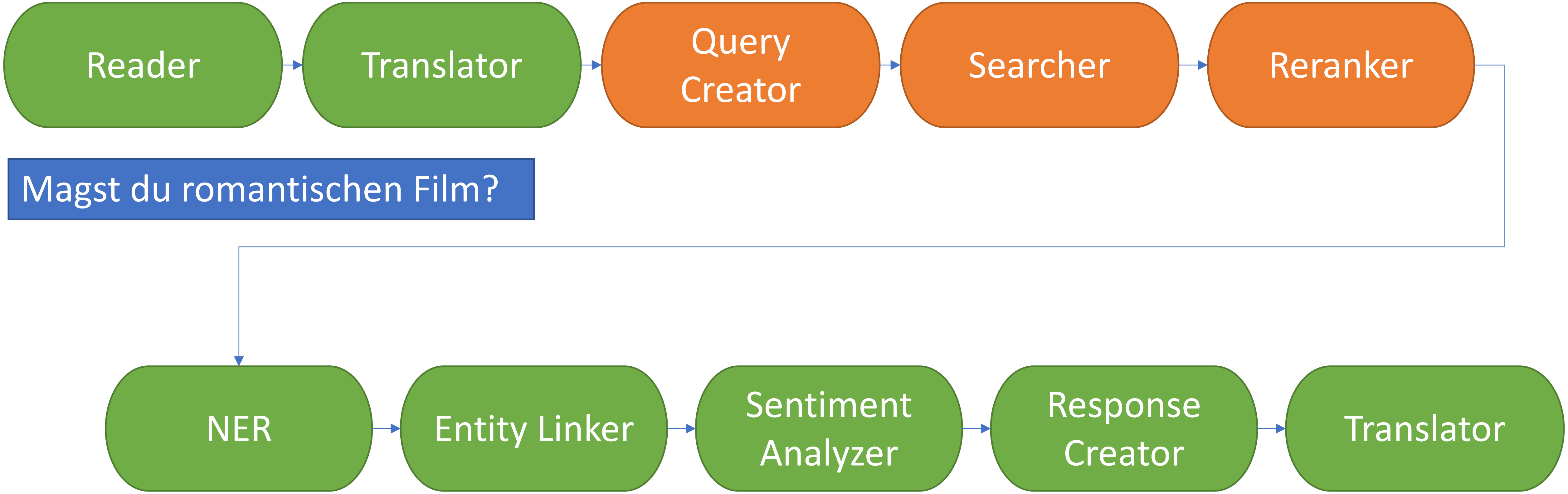




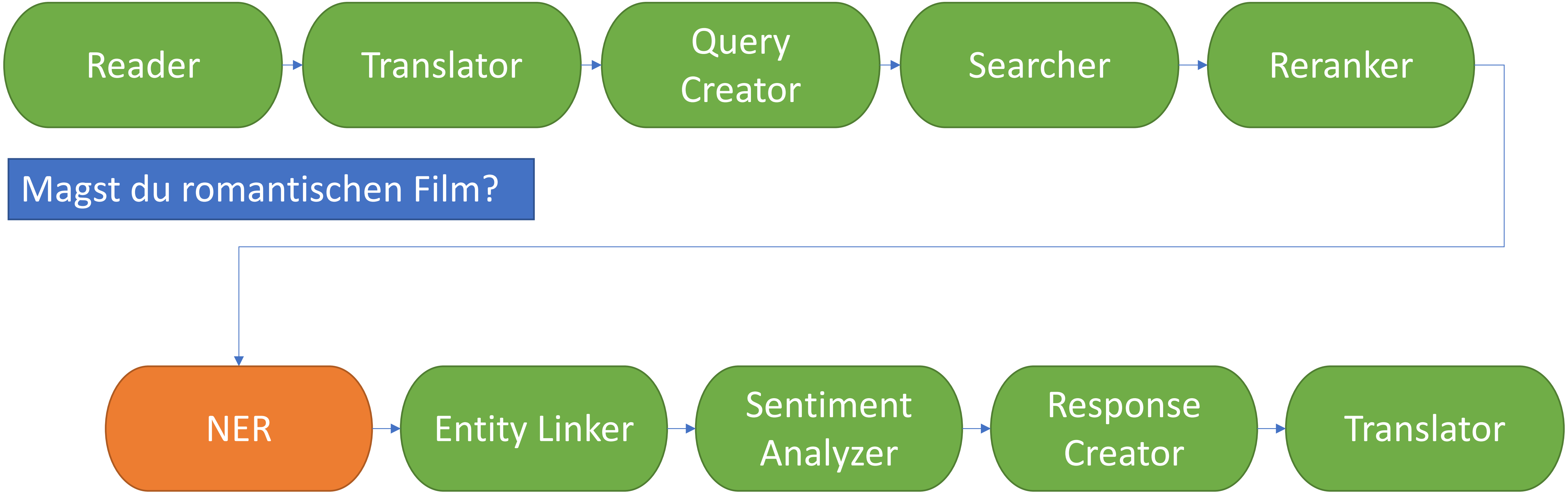
Here is one possible solution pipeline



Translate the input to/from English.

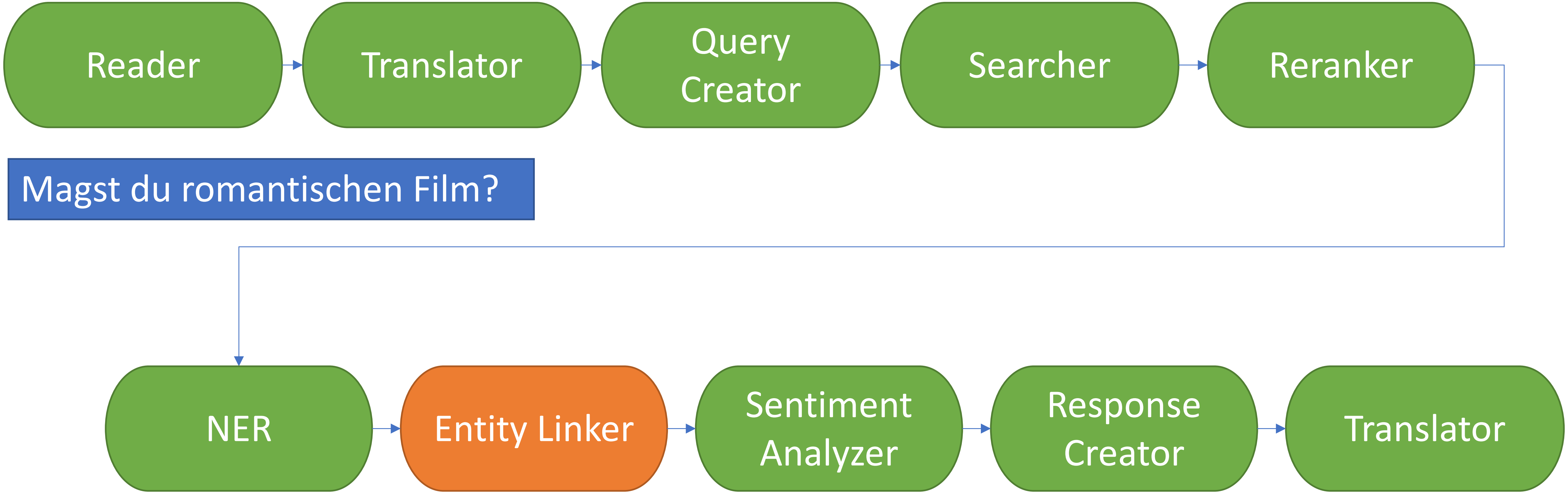


Create appropriate queries and run on the Information Retrieval Engine



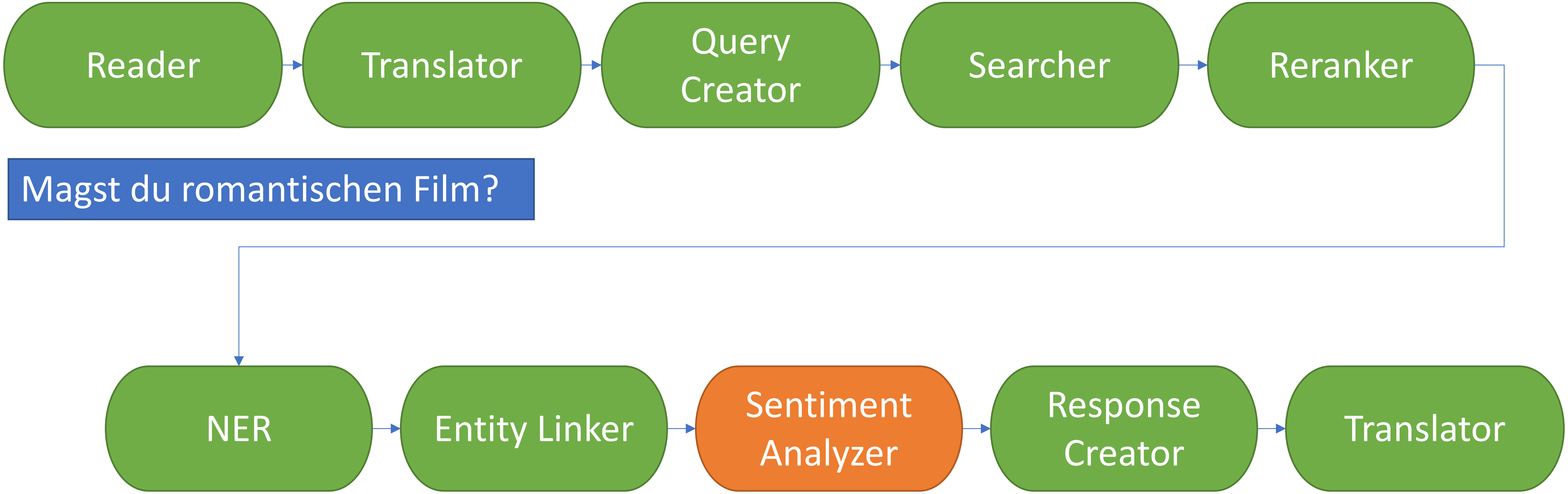
NER

Kate Winslet and Leonardo Dicaprio have definitely created a timeless classic.



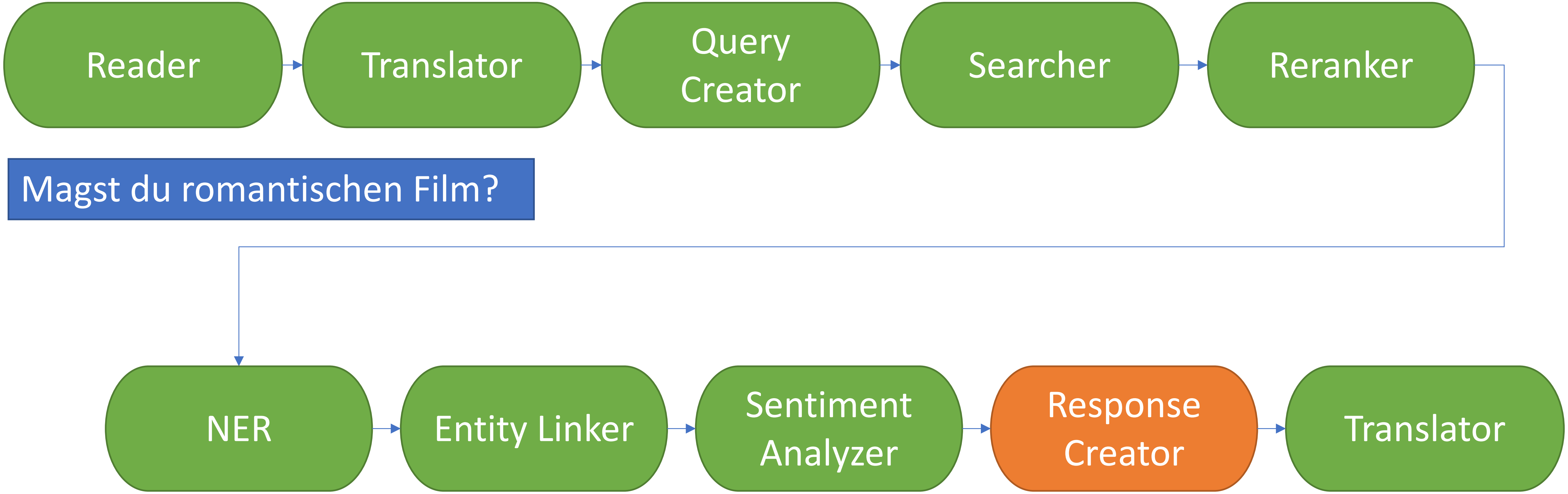
Entity Linker

Link "well-known" mentions to backend knowledge base, such as "Leonardo Dicaprio"



Sentiment Analyzer

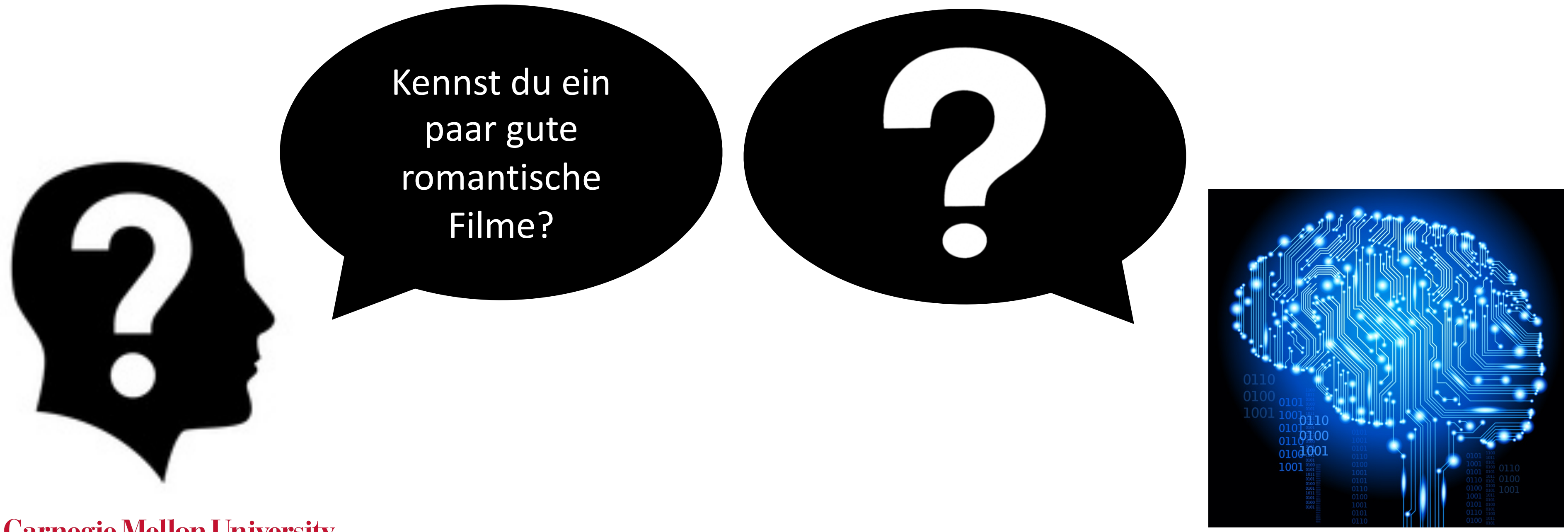
Analyzer the sentiment of the retrieved reviews to pick a positive one.



Now with all the above, can we construct an answer?

Inter-Operation Across the Pipeline

- Every Step in the Pipeline produces useful information
- Can we easily access and utilize these for the final goal?
- Let’s review how we normally build such a pipeline



How to build complex NLP pipelines?

Decompose the pipeline into steps

Solve each step

Use an existing implement	Build your own implement

Connect the steps

How to build complex NLP pipelines?

Decompose the pipeline into steps

↓

Solve each step

Pick an existing implement	Build your own implement

↓

Connect the steps

AllenNLP

NLTK

DKPro

spaCy

NMT

How to build complex NLP pipelines?

Decompose the pipeline into steps

Solve each step

Pick an existing implement | Build your own implement

Connect the steps

PYTORCH

scikit learn

TensorFlow

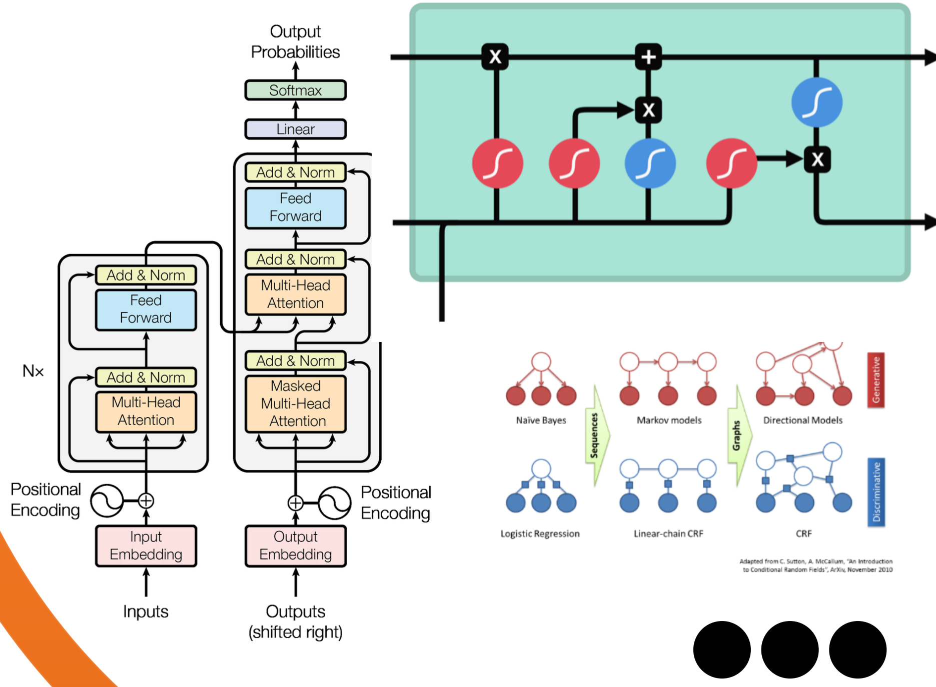
How to build complex NLP pipelines?

Decompose the pipeline into steps

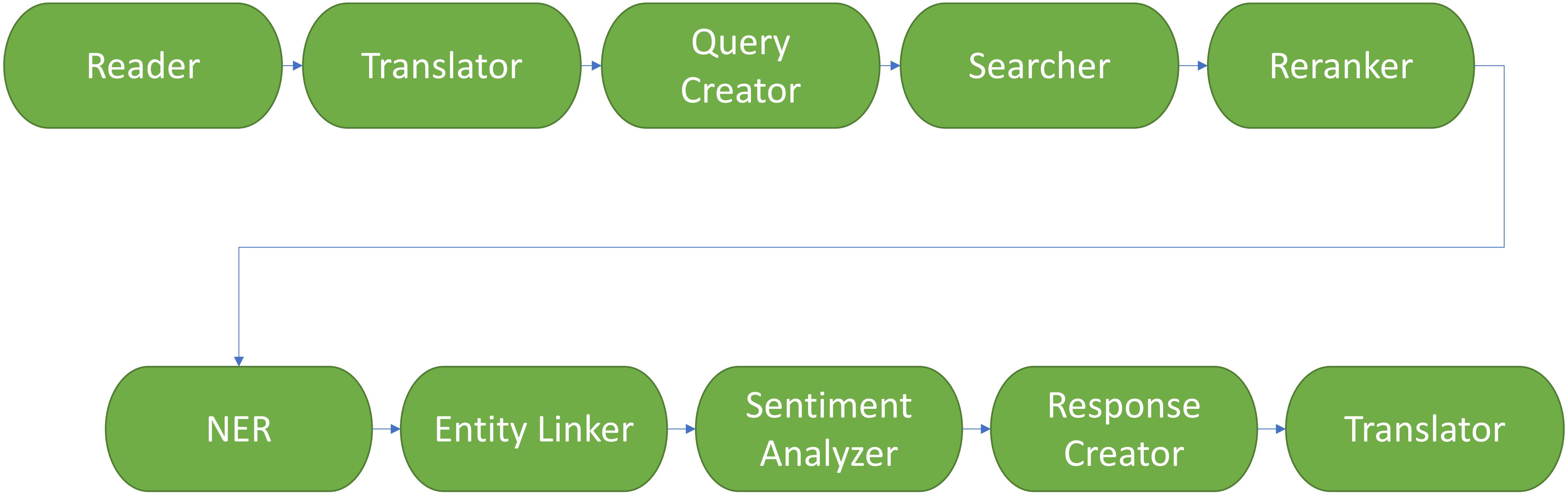
Solve each step

Pick an existing implement	Build your own implement

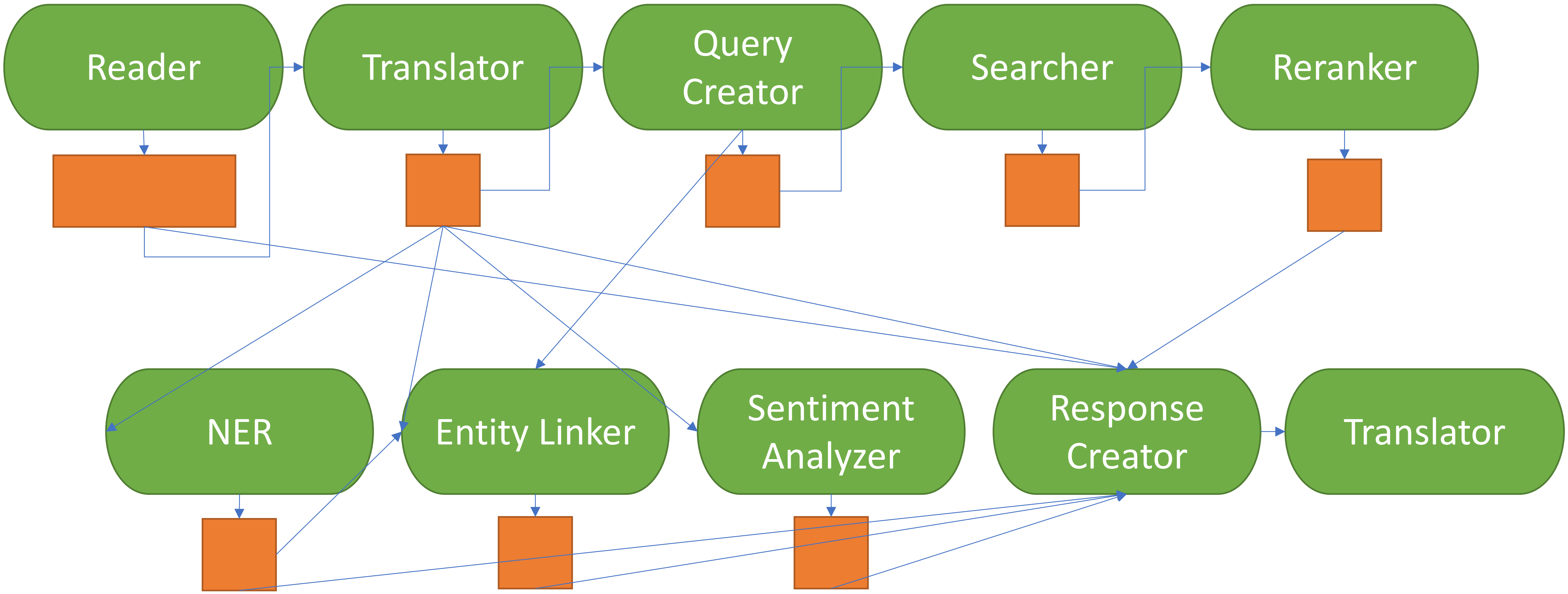
Connect the steps



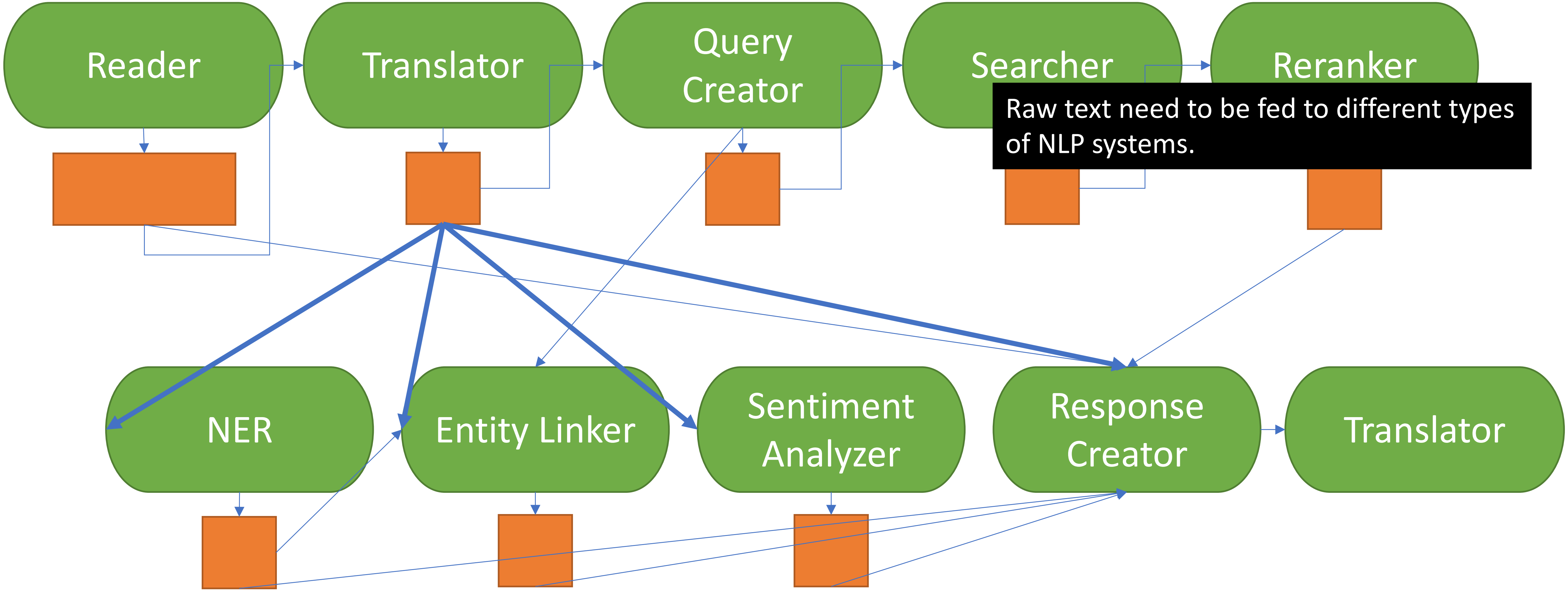
So here is the “expected” pipeline



What it might look like

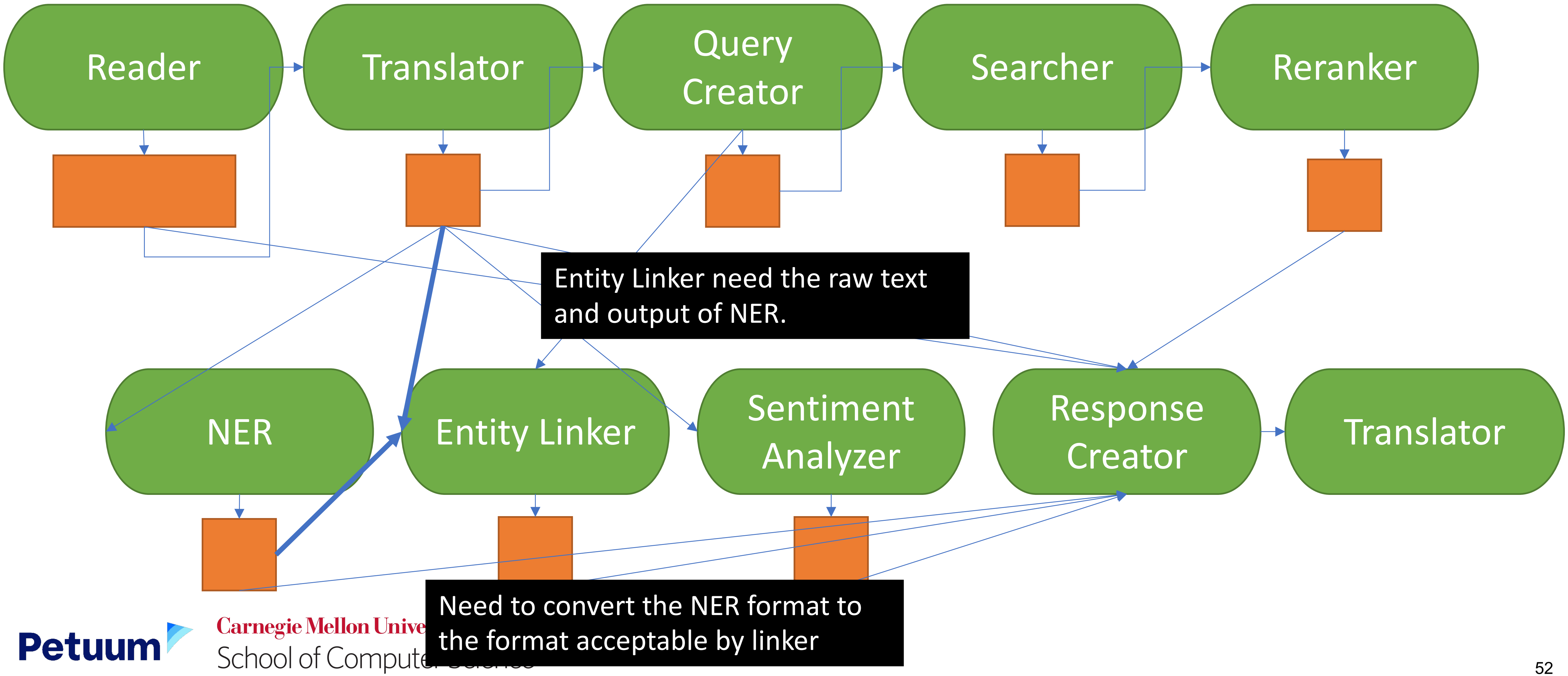


What it might look like

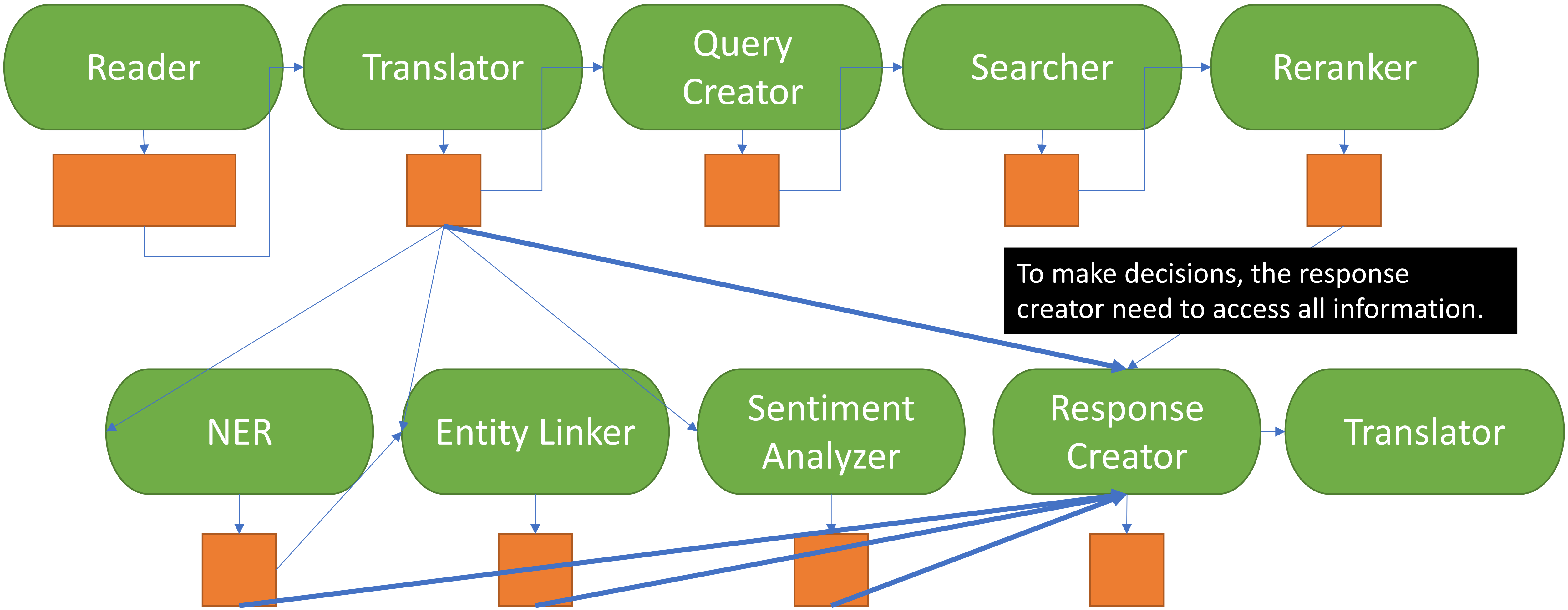


Raw text need to be fed to different types of NLP systems.

What it might look like



What it might actually look like



What it might look like

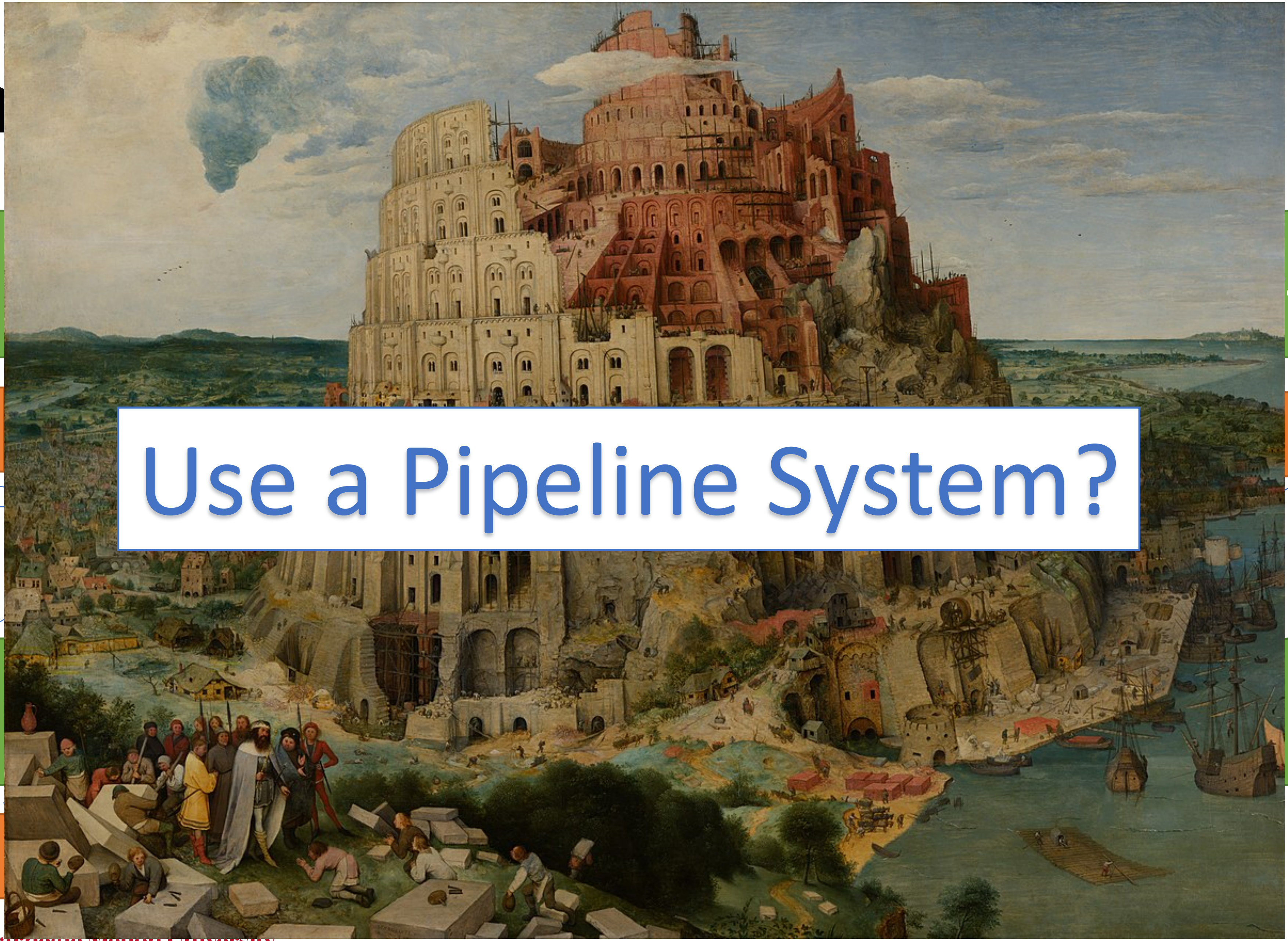
Wha

Reader

anker

N

Translator



Use a Pipeline System?

Examples of Some Pipeline Systems

- Illinois Curator
    - Support many different NLP tasks
    - Especially strong in different flavors of SRL
    - Allow composing many different tasks

Curator Demo check status? about?

Enter some text in the box below. Then, check the boxes next to the types of annotation you want Curator to provide. Click "process" to send your text to be annotated with those resources. When the selected annotation services have finished, the outputs of those services will be displayed on this page.

You can replace this placeholder text with the sentences you'd like the Curator to annotate.

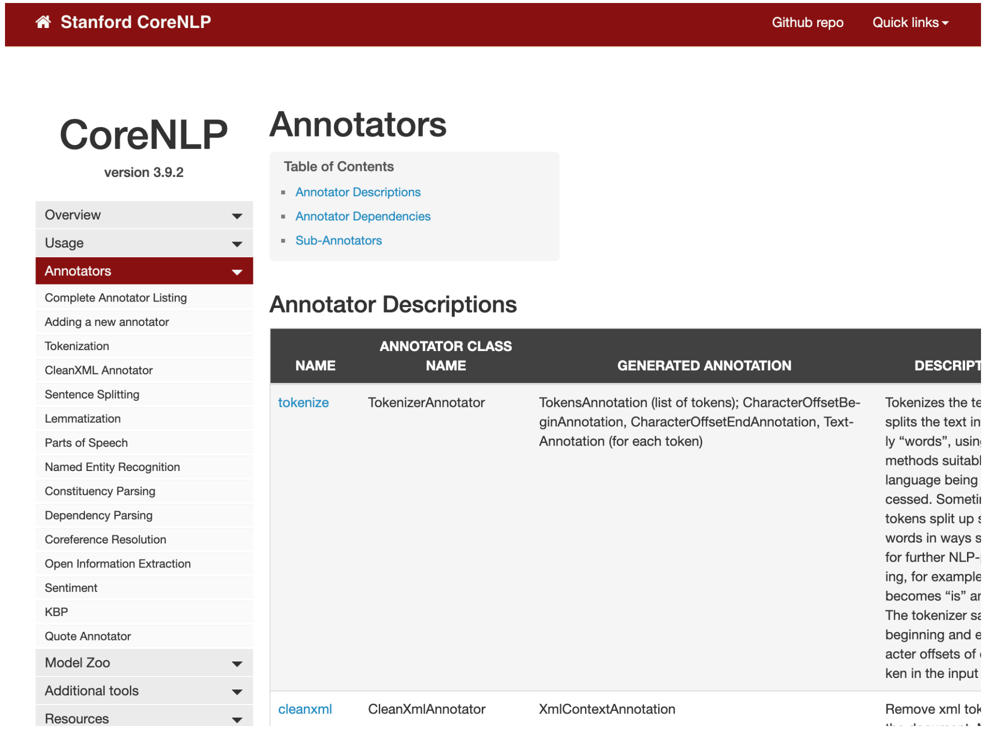
process

Select the annotation services you'd like to be applied to your text:

- [x] Sentences -- Illinois sentence-level segmenter
- [x] Tokens -- Illinois token-level segmenter
- [ ] Part-of-Speech -- Illinois Part-of-Speech tagger
- [ ] Shallow Parse chunks -- Illinois Chunker (a.k.a. Shallow Parser)
- [ ] Named Entities -- Illinois Named Entity Recognizer 2014 CoNLL (PER/LOC/ORG/MISC)
- [ ] Named Entities -- Illinois Extended Named Entity Recognizer 2014 Ontonotes (18 types)
- [ ] Quantities -- Illinois Quantity Recognizer
- [ ] Semantic Roles (verbs) -- older SRL

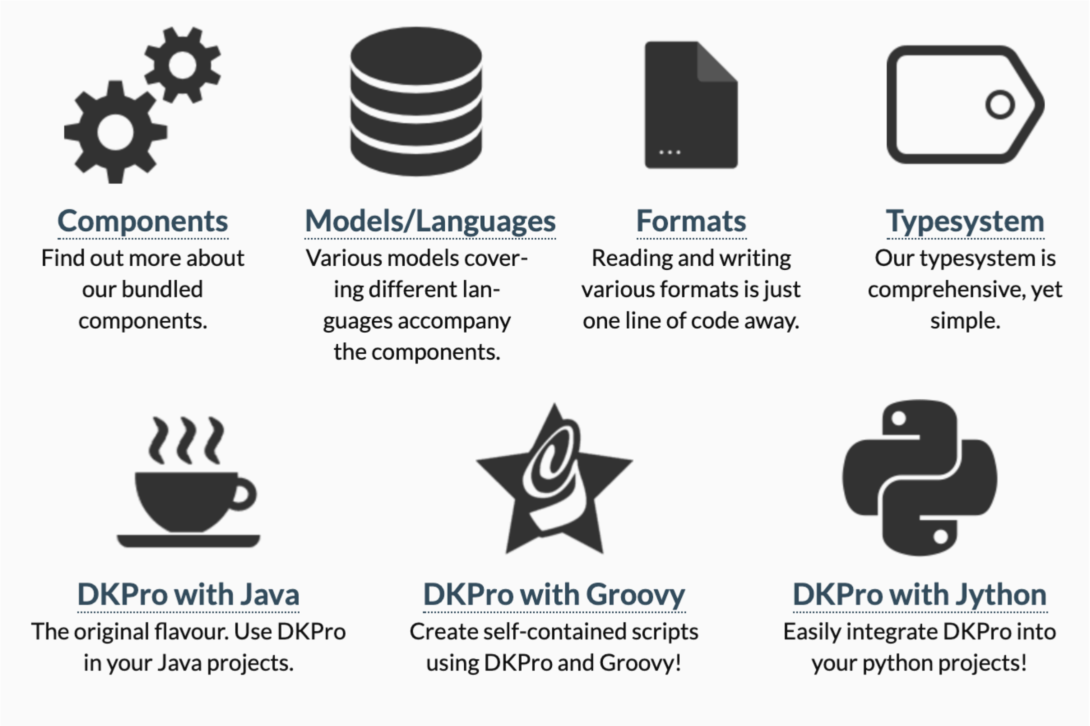
Examples of Some Pipeline Systems

- Stanford
  - Features in a good coverage in core NLP tasks
  - Provide utilities on these tasks
  - Strong dependency between tasks (e.g. parsing depends on tokenization)



Examples of Some Pipeline Systems

- DKPro Core
  - Support large number of tools
  - Tooks are very loosely coupled
  - Use a universal data format based on Typesystem
  - Easy pipeline composing
  - A transparent data flow



What's Provided Now?

The current approach

- ✓ Useful NLP Tools
- ✓ Pre-trained Models
- ✓ Pipeline Systems

Decompose the pipeline into steps

↓

Solve each steps

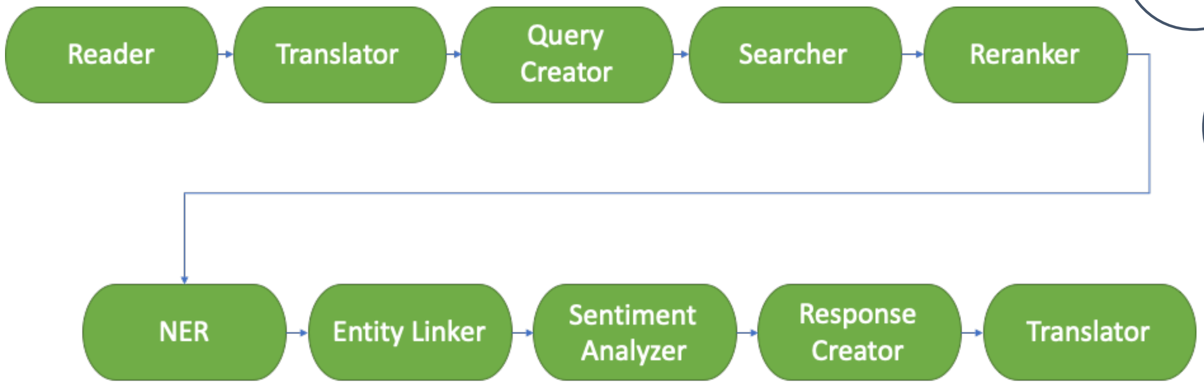
Pick an existing implement	Build your own implement

↓

Connect the steps

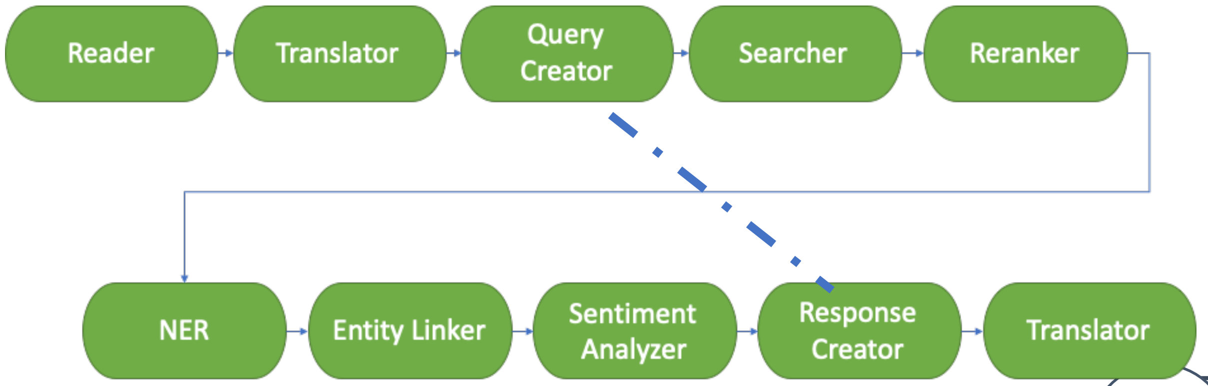
What do we want to achieve?

- Full Spectrum Pipeline
- Inter-Operation Across Tasks
- Easy Pipeline Customization
- Flexible Model Building
- Versatile Data Support



- **Full Spectrum Pipeline**
- Inter-Operation Across Tasks
- Easy Pipeline Customization
- Flexible Model Building
- Versatile Data Support

Can we use one pipeline to support all steps above?



- Downstream tasks can use the results from the previous tasks
- E.g. Using sentiment analysis result in the answer selection step

- Full Spectrum Pipeline
- **Inter-Operation Across Tasks**
- Easy Pipeline Customization
- Flexible Model Building
- Versatile Data Support

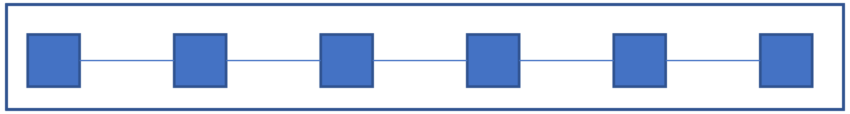
Can we freely change the operations in a pipeline?

For example:

- Plug-in a tokenizer.
- Replace the translator.
- Remove lemmatization.

- Full Spectrum Pipeline
- Inter-Operation Across Tasks
- **Easy Pipeline Customization**
- Flexible Model Building
- Versatile Data Support

Can we directly reuse a model for a new task quickly?



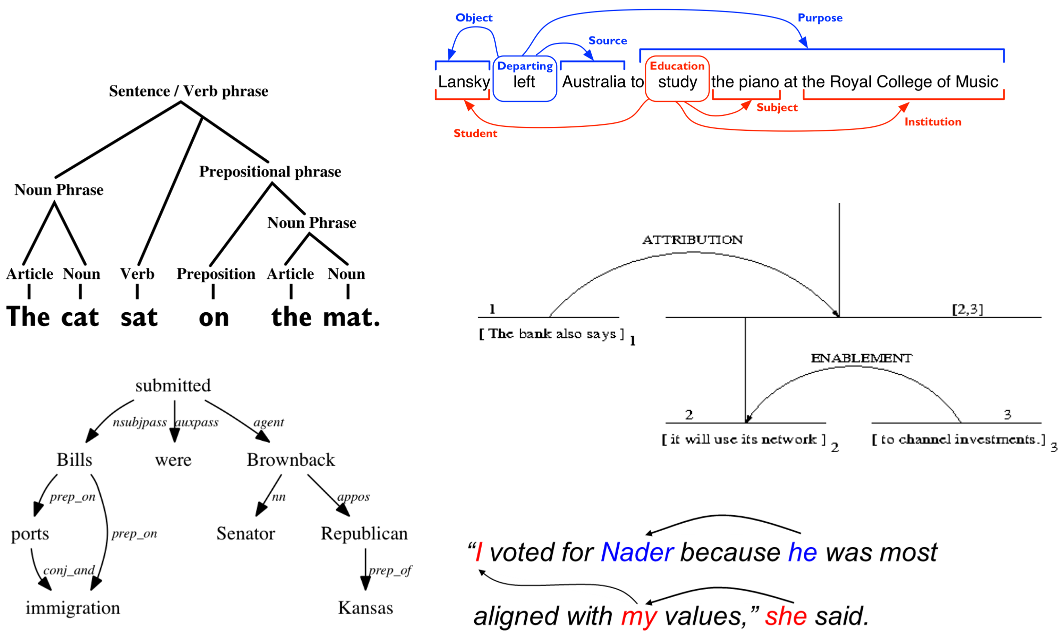
LSTM-CRF

Tom and Bill work at the same company.
PER O PER O O O O O

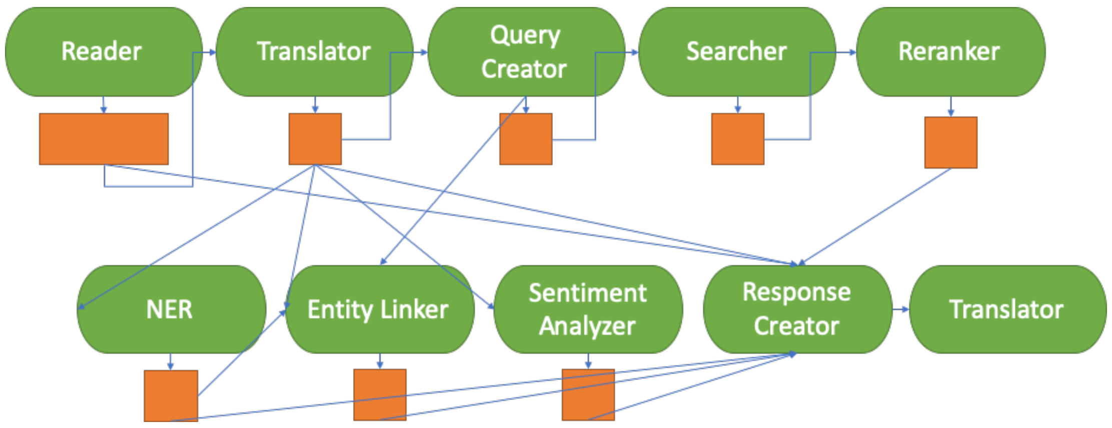
He is going to the bookshop.
PR V V IN DT N

- Full Spectrum Pipeline
- Inter-Operation Across Tasks
- Easy Pipeline Customization
- **Flexible Model Building**
- Versatile Data Support

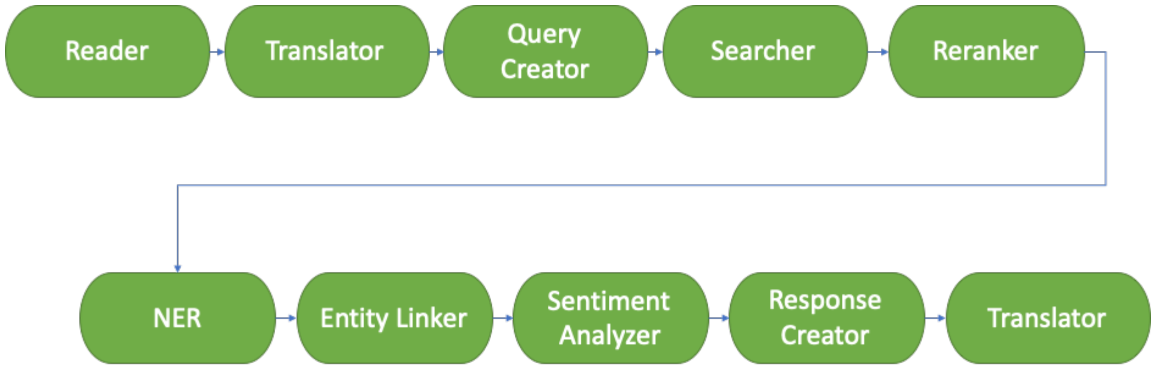
E.g. Different NLP data formats of different tasks



- Full Spectrum Pipeline
- Inter-Operation Across Tasks
- Easy Pipeline Customization
- Flexible Model Building
- Versatile Data Support



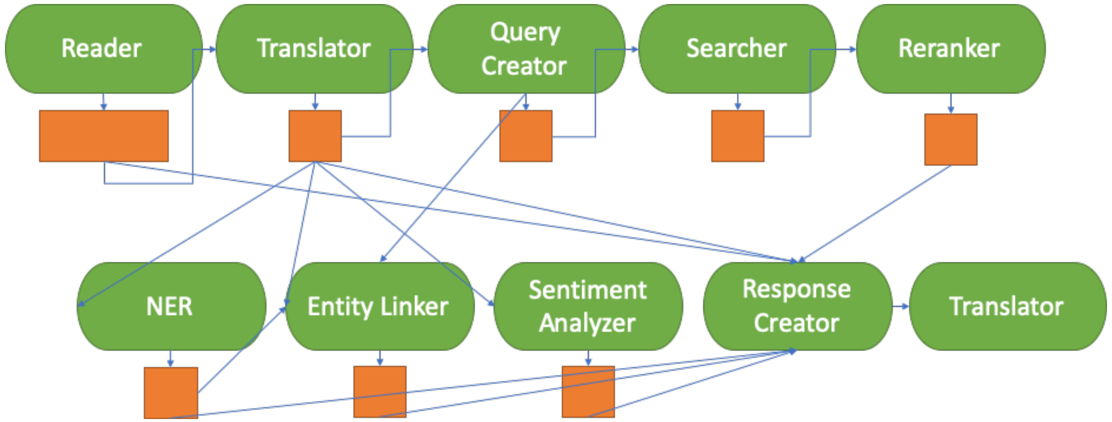
How?



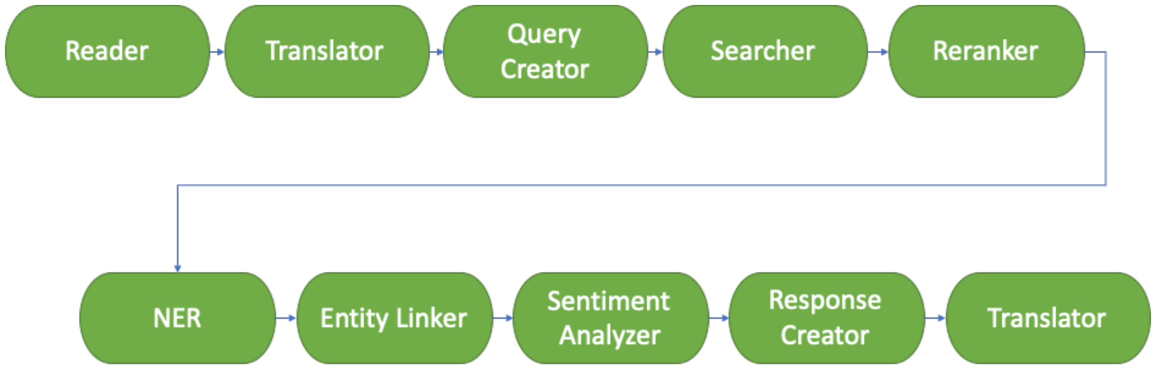
- Full Spectrum Pipeline
- Inter-Operation Across Tasks
- Easy Pipeline Customization
- Flexible Model Building
- Versatile Data Support

Petuum

Carnegie Mellon University
School of Computer Science

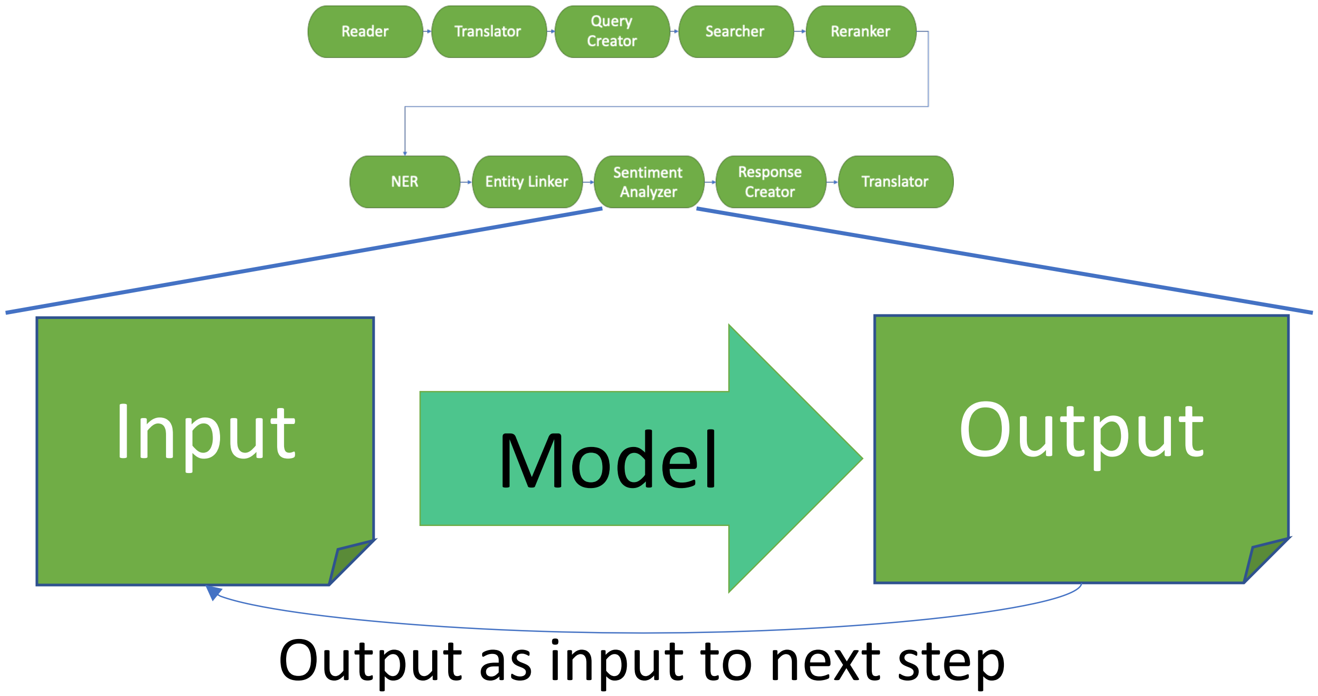


Standardize NLP

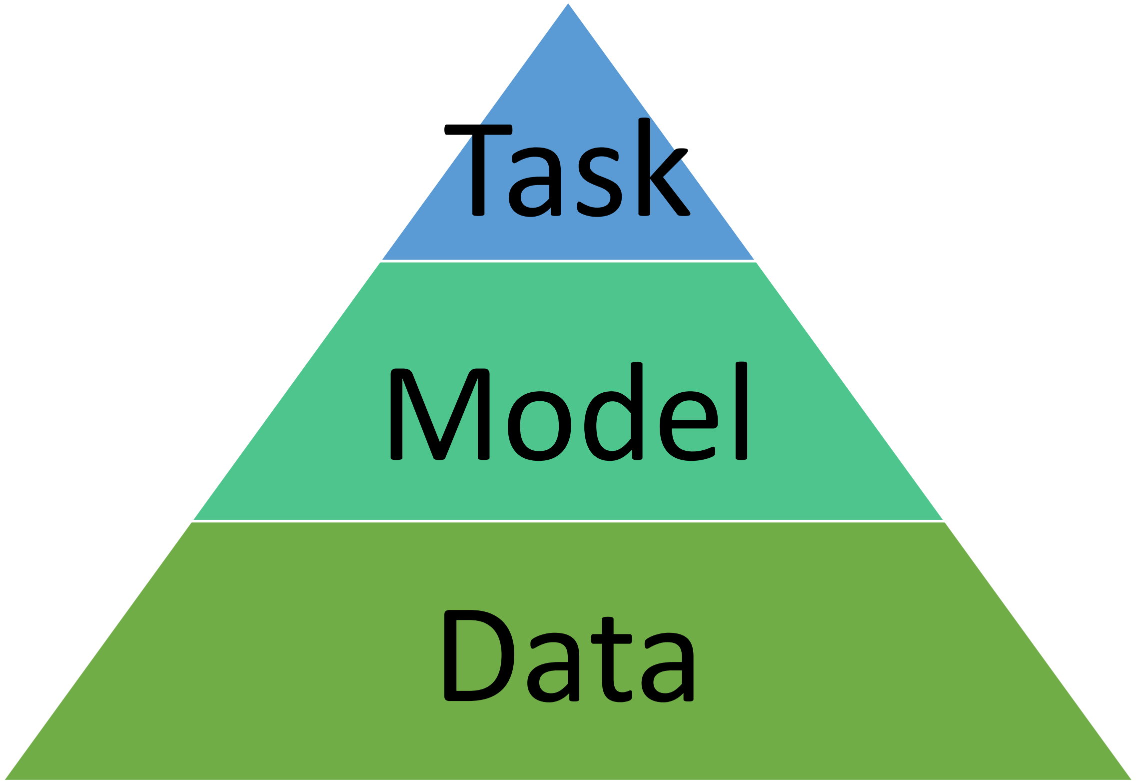


- Full Spectrum Pipeline
- Inter-Operation Across Tasks
- Easy Pipeline Customization
- Flexible Model Building
- Versatile Data Support

Standardize NLP Interfaces



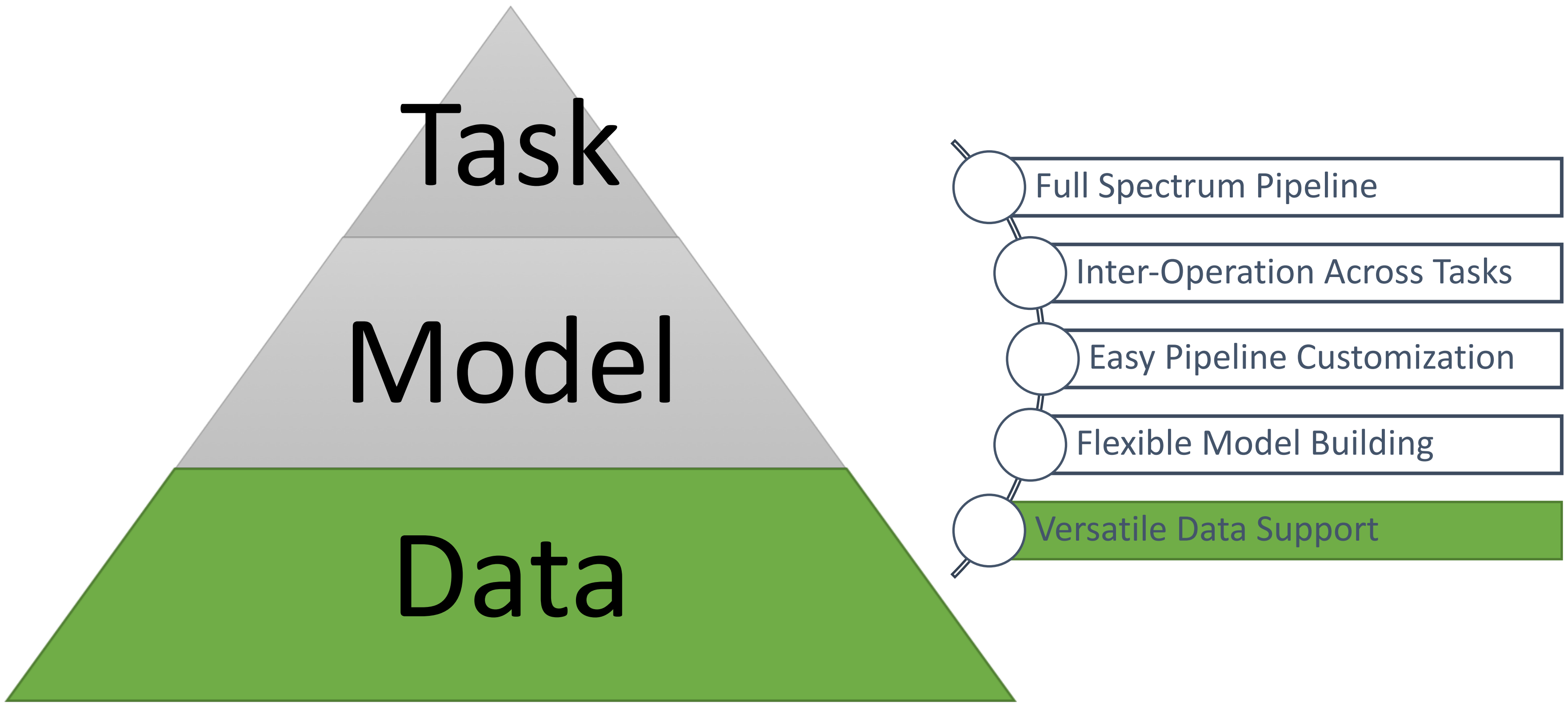
Operationalized View

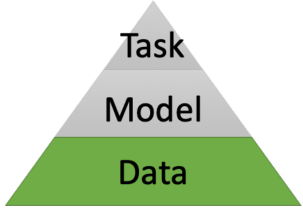


Conceptual View

Standardize interfaces between the 3 levels

Considerations of Standardizing Data Representation

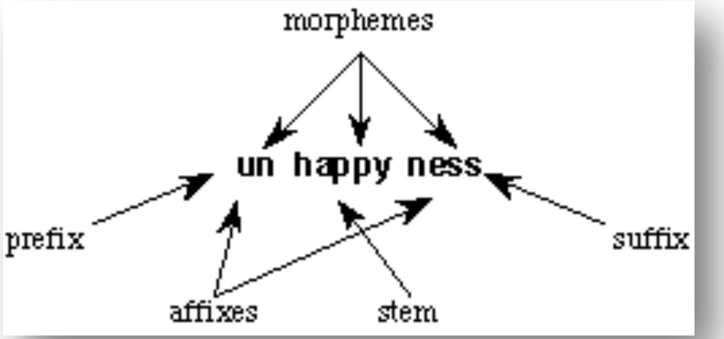


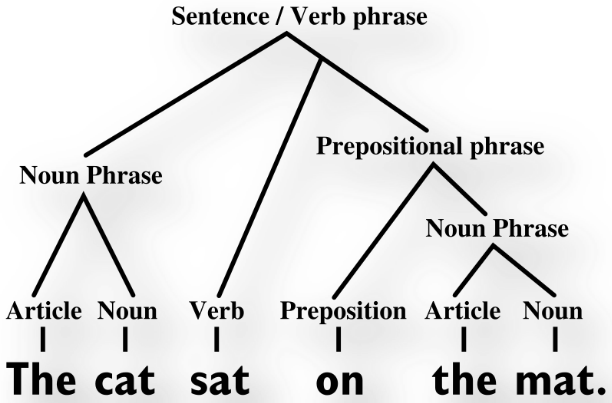


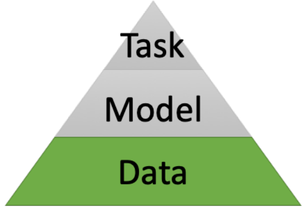
Data Granularity

- NLP data have different granularity:
  - **Character** level (Chinese Radical)
  - **Token** level (Morphology)
  - **Sentence** level (Syntax Parsing)
  - **Document** level (Entity Coreference)
  - **Corpus** level (Information Retrieval)

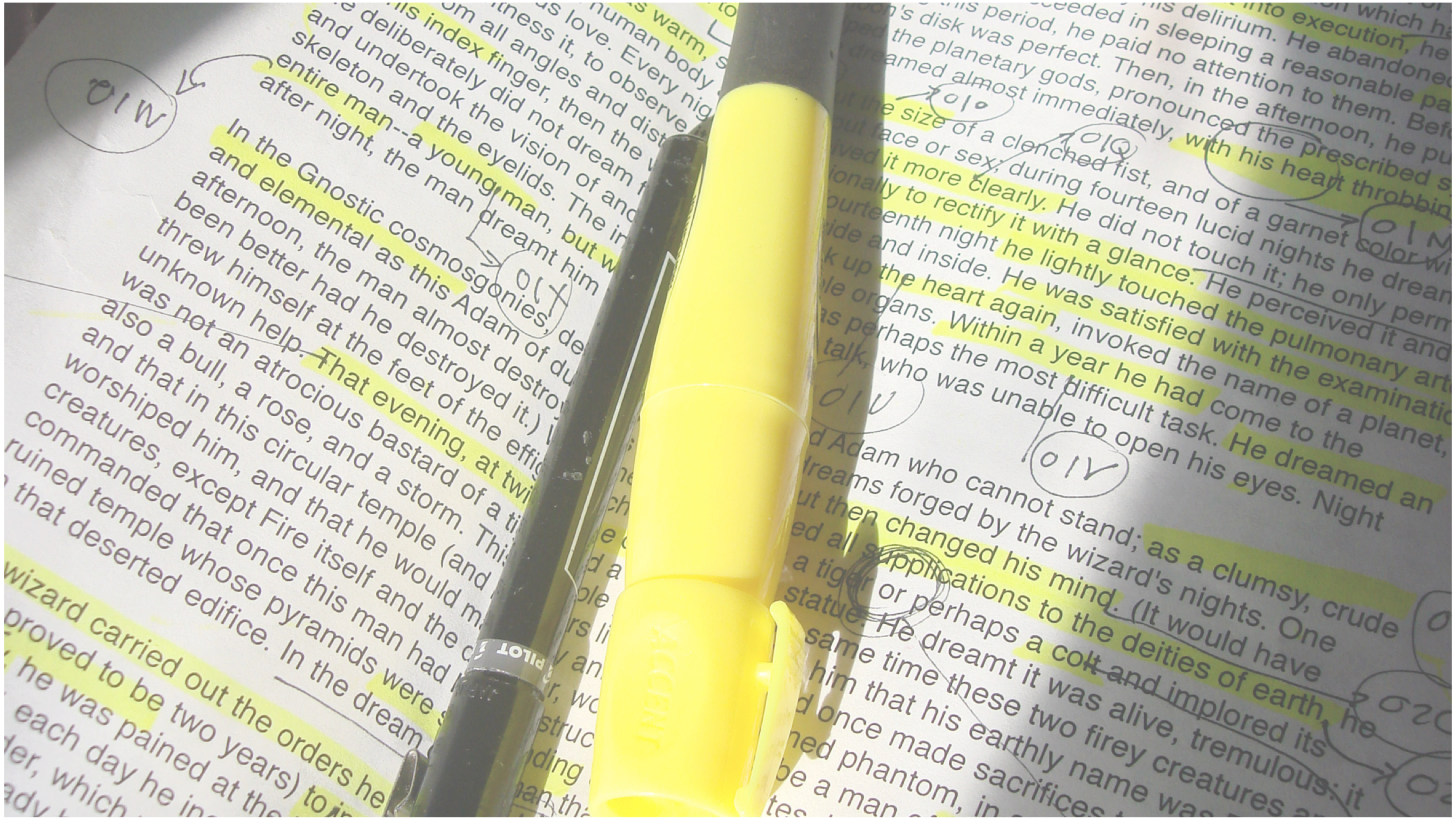
- Representation should handle flexible granularities

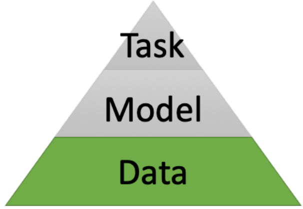






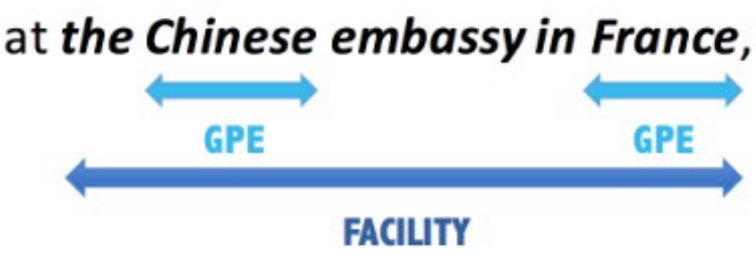
Data Structures

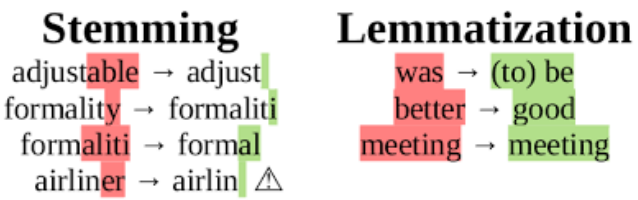


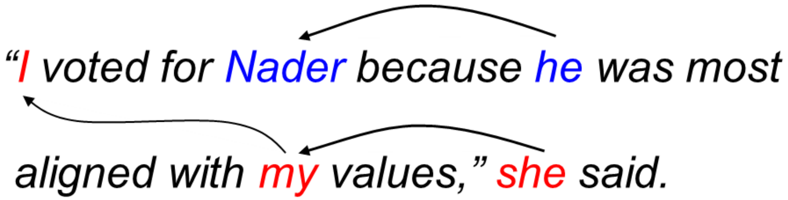


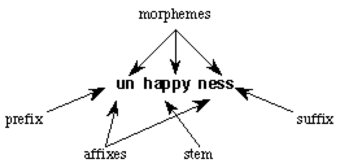
Data Structures

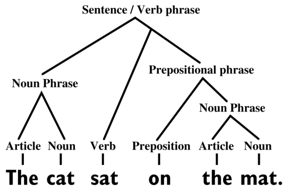
- Let's again look at the tasks, any patterns?

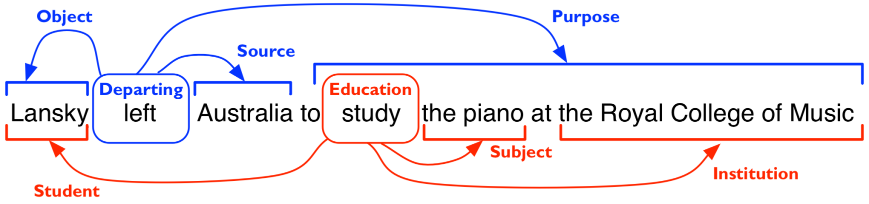


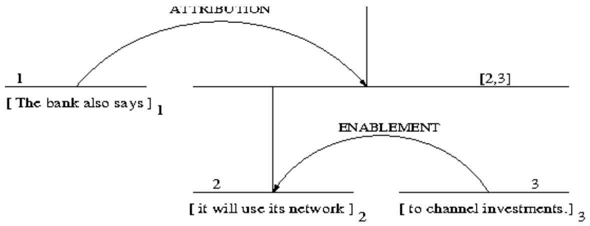


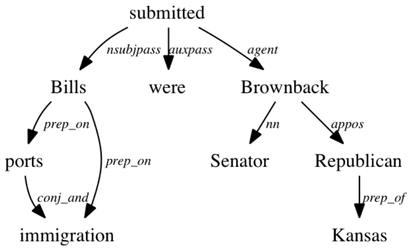






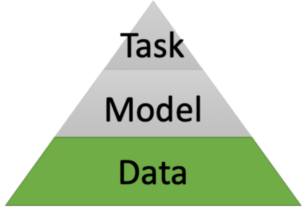






Petuum

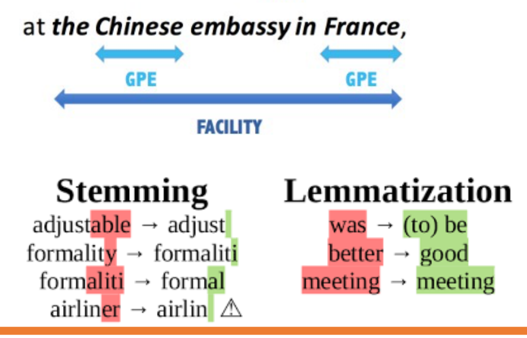
Carnegie Mellon University
School of Computer Science



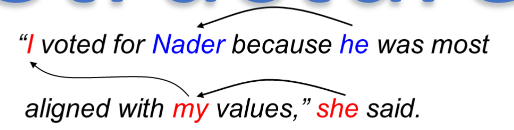
Data Structures

- Let’s again look at the tasks, any patterns?

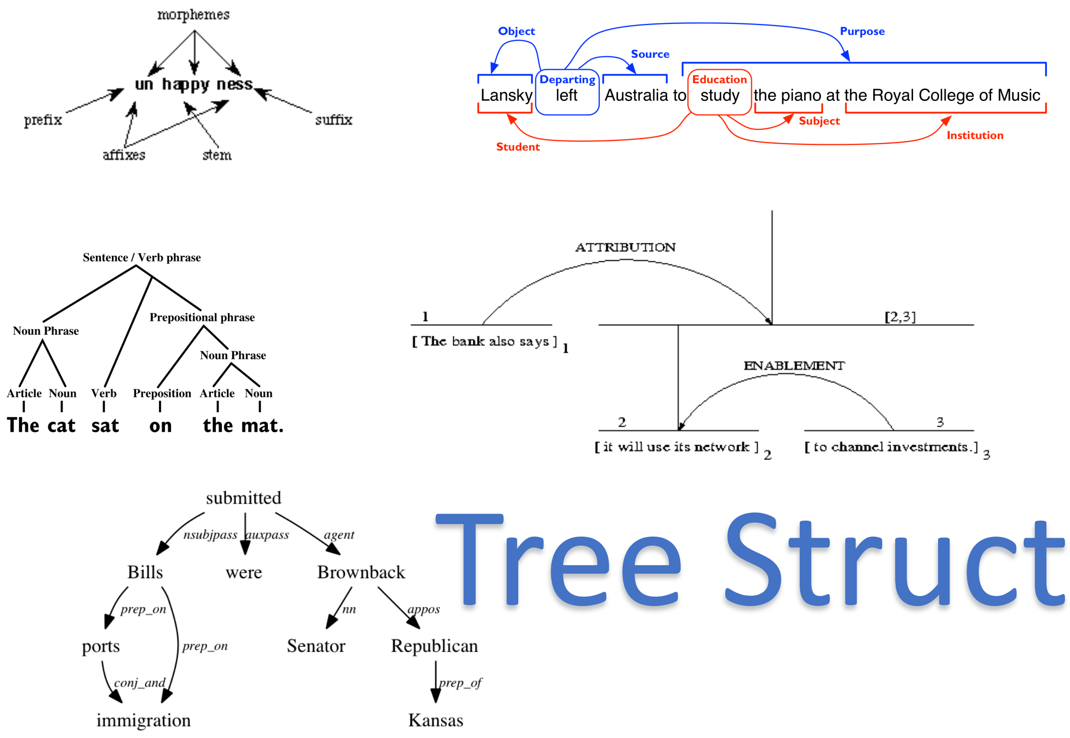
Sequence Structure

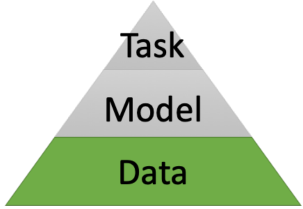


Cluster Structure



Tree Structure

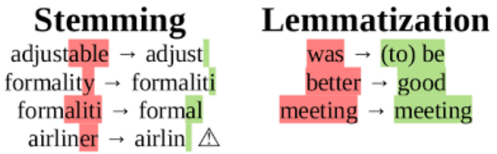


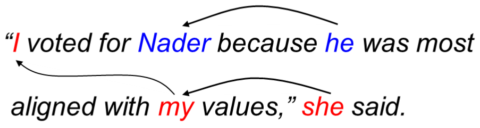


Data Structures

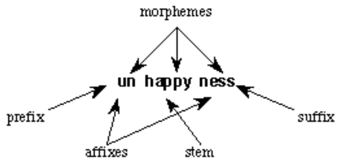
- Let’s again look at the tasks, any patterns?

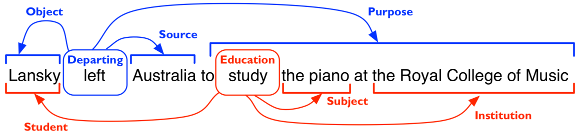
Spans

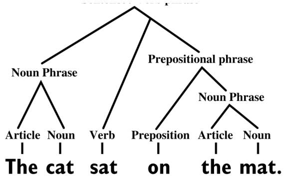


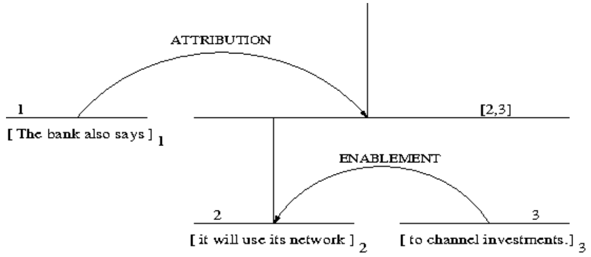


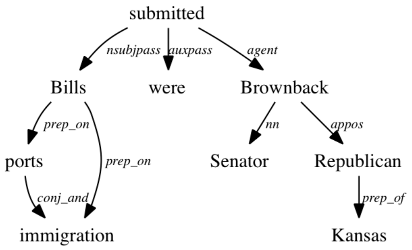
Groups



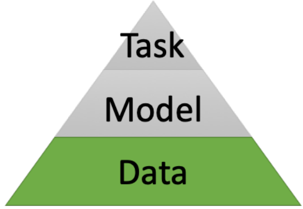








Links



Spans

additional fields, e.g., embeddings, ner tag

Span = (type, begin, end, …)

E1 = (EntityMention, 25, 35)

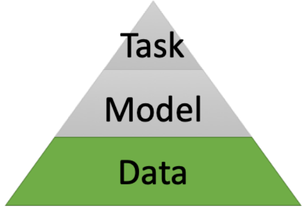
P2 = (Predicate, 42, 46)

EntityMention

Predicate

EntityMention

Last year, it was Rams quarterback Jared Goff, who failed to spot a wide-open Brandin Cooks when the NFC Champions ran in the second half a play that had sprung Cooks free in the first half.



Links

additional fields, e.g., dependency type

Link = (Type, Parent, Child, …)

S1= (SemanticRoleLink, P1, E1)

EntityMention

Predicate

EntityMention

Last year, it was Rams quarterback Jared Goff, who failed to spot a wide-open Brandin Cooks when the NFC Champions ran in the second half a play that had sprung Cooks free in the first half.

Groups

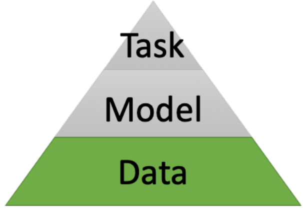
Task
Model
Data

additional fields, e.g., group type, score

Group = (Type, Member List, …)

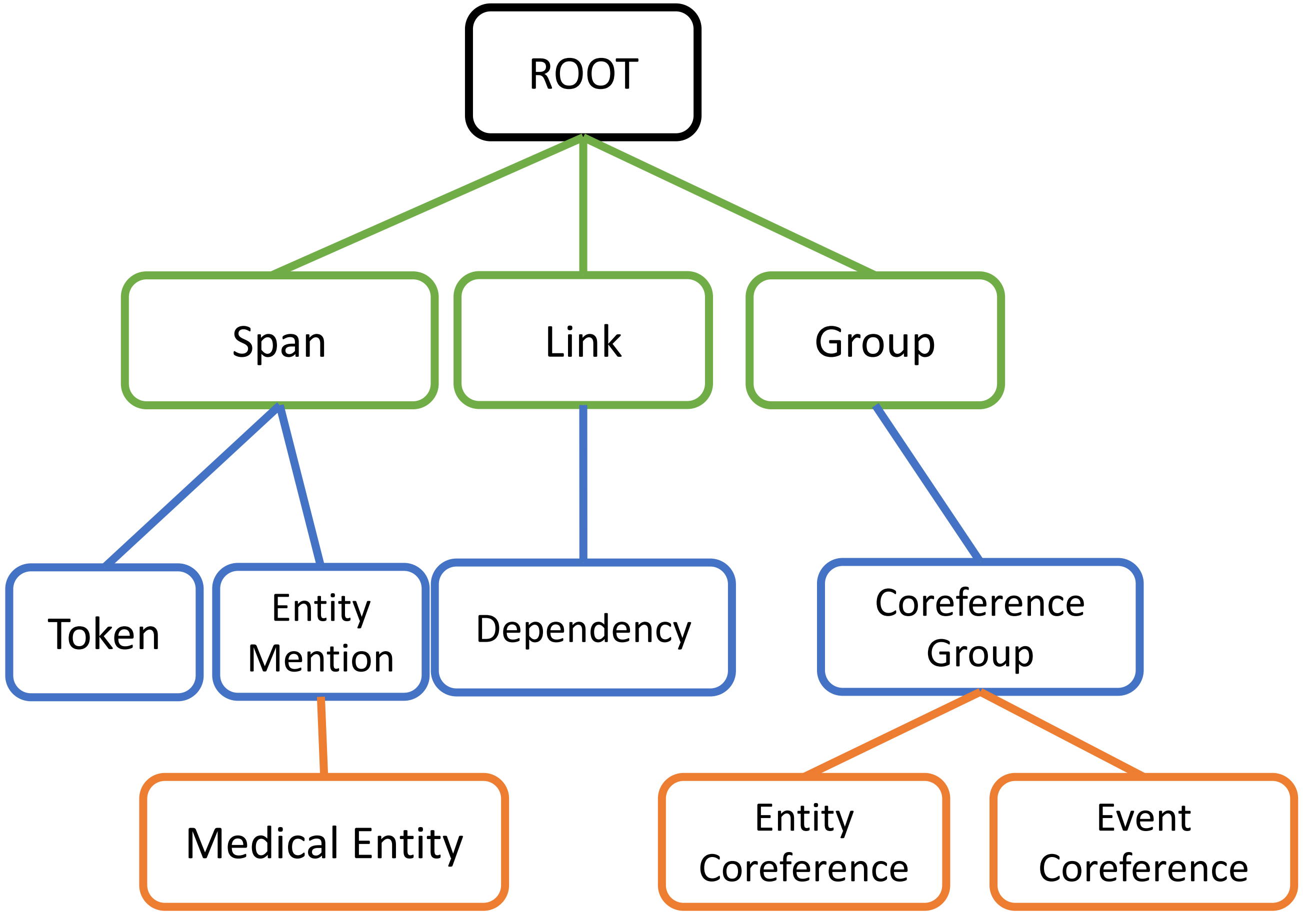
G1= (EntityCoreferenceGroup, Member=[E2,E3])

Last year, it was Rams quarterback Jared Goff, who failed to spot a wide-open Brandin Cooks when the NFC Champions ran in the second half a play that had sprung Cooks free in the first half.

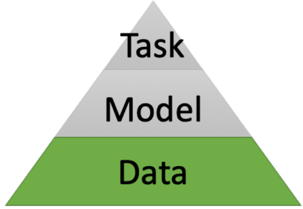


Ontology of Data Structures

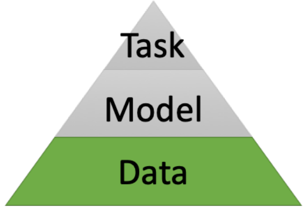
- Types of data structures can be organized as an ontology tree.
- The ontology can be customized for new domains.



Practical Data Considerations

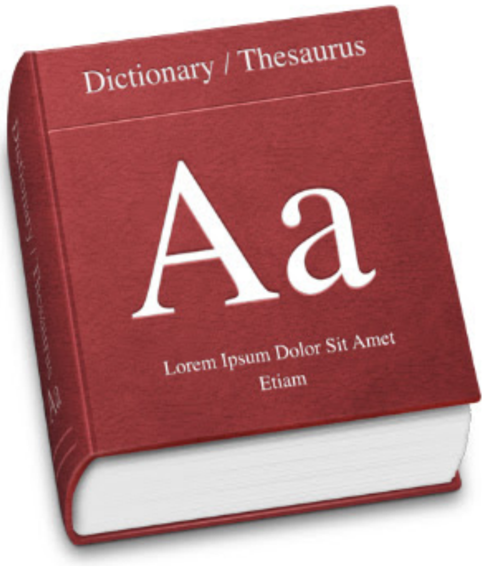


- Serialization (*S*) and Deserilization (*D*)
  - Loseless: *D*(*S*(data)) = data
  - Can be passed around (e.g. networks channels)
- Readable, Interpretable
- Meta Data
  - Keep source information
  - Automatic record creation time, creator, etc.

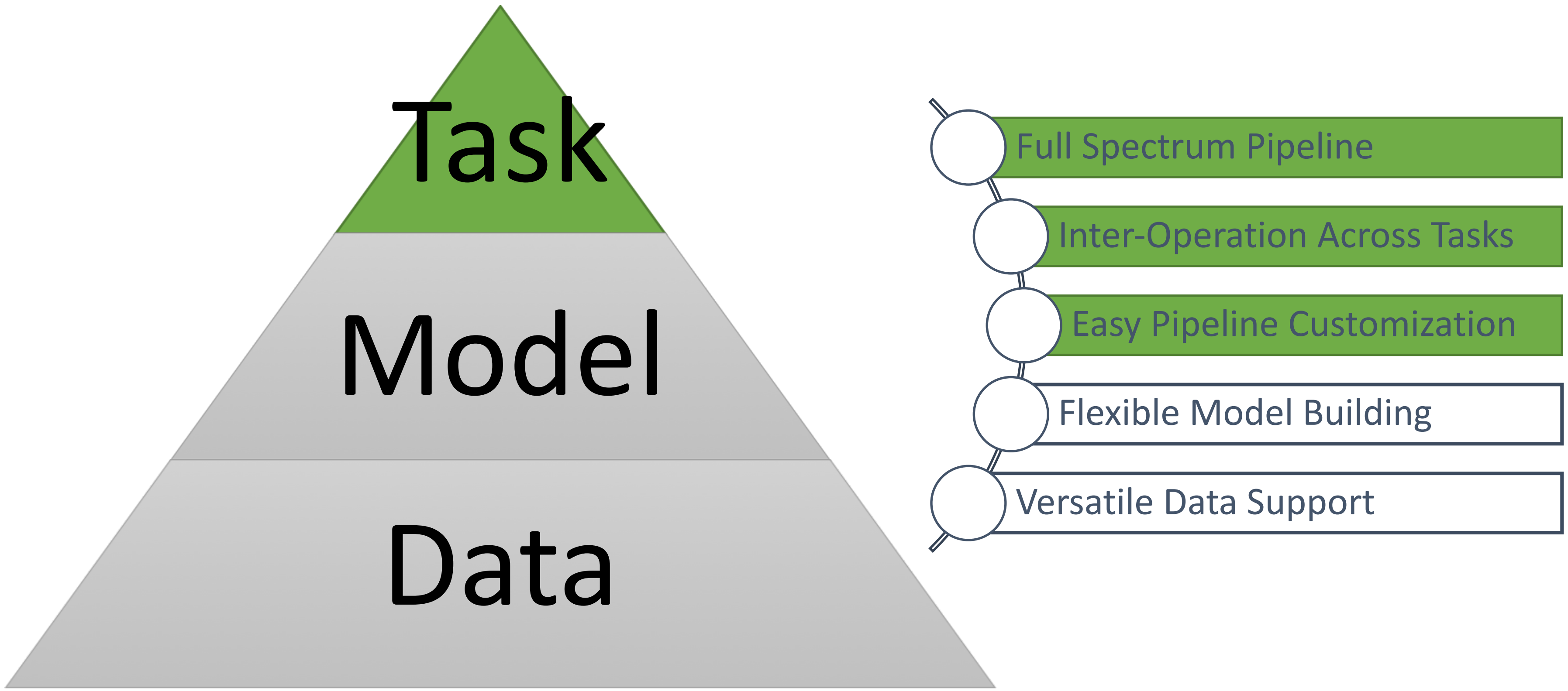


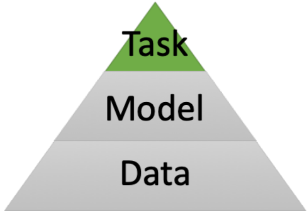
Global Data

- Vocabulary
- Embeddings (e.g. Word2Vec, Bert)



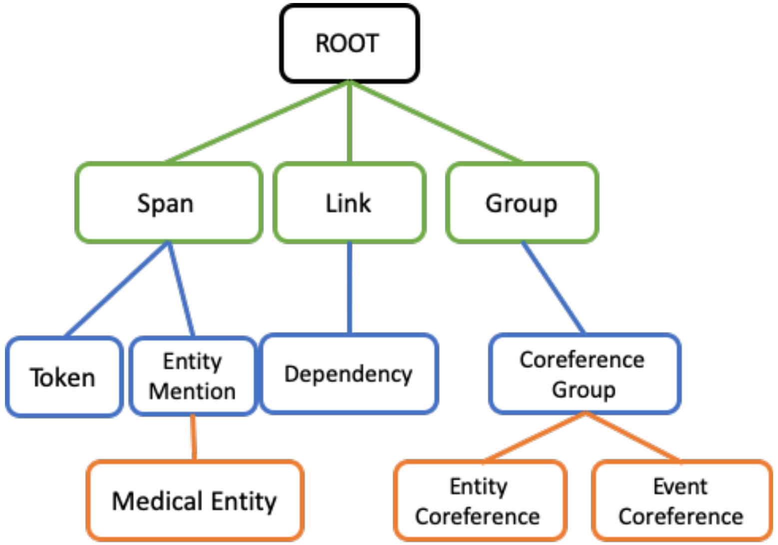
How about Tasks?





Data as Interface

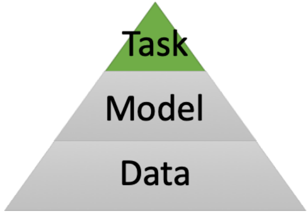
- A standard interface can be defined given the data representation.
- This helps task modularization.

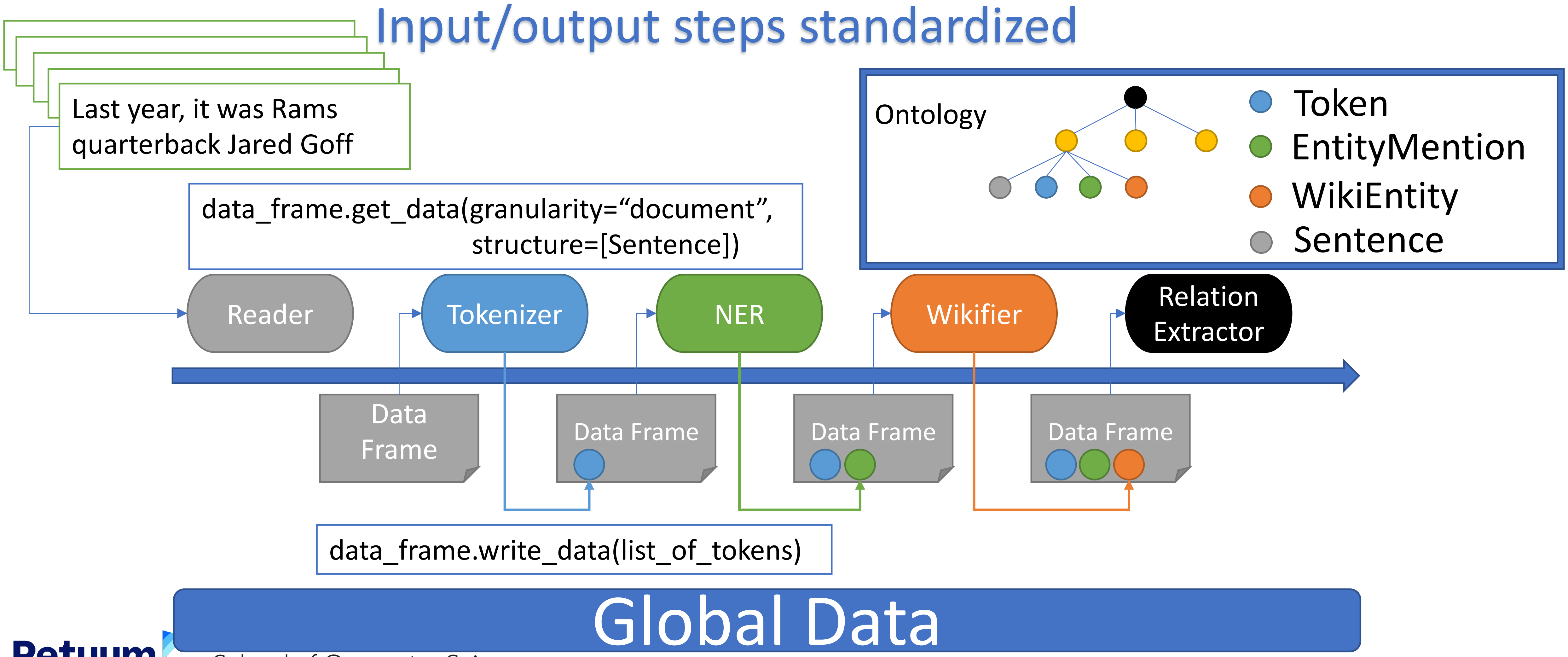


Accessing data using the granularity and structure

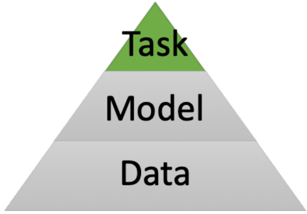
get_data(**granularity**="sentence",
structure=[PredicateMention, EntityMention])

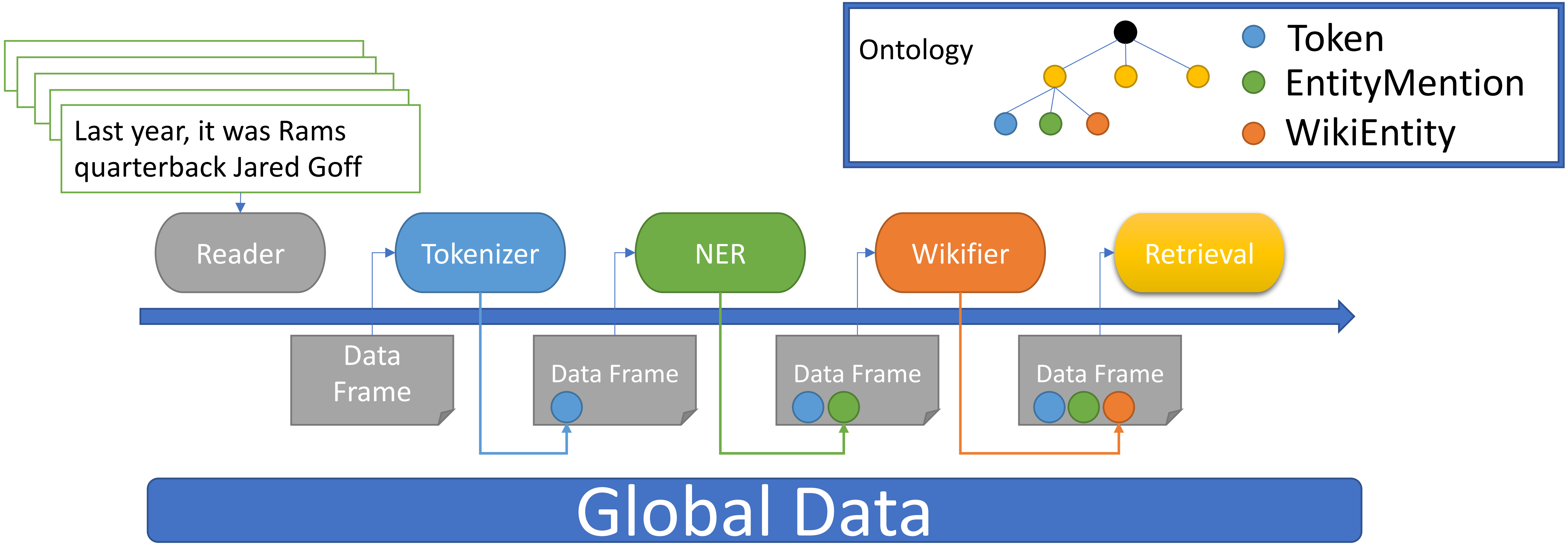
- With Standardized Data and Interfaces:
  - Tasks can already be organized in a modular way





- Full Spectrum Pipeline Customization: We can insert/replace an Information Retrieval into the pipeline
- Inter Operation: Complex queries can be built using the NER and Wikifier information

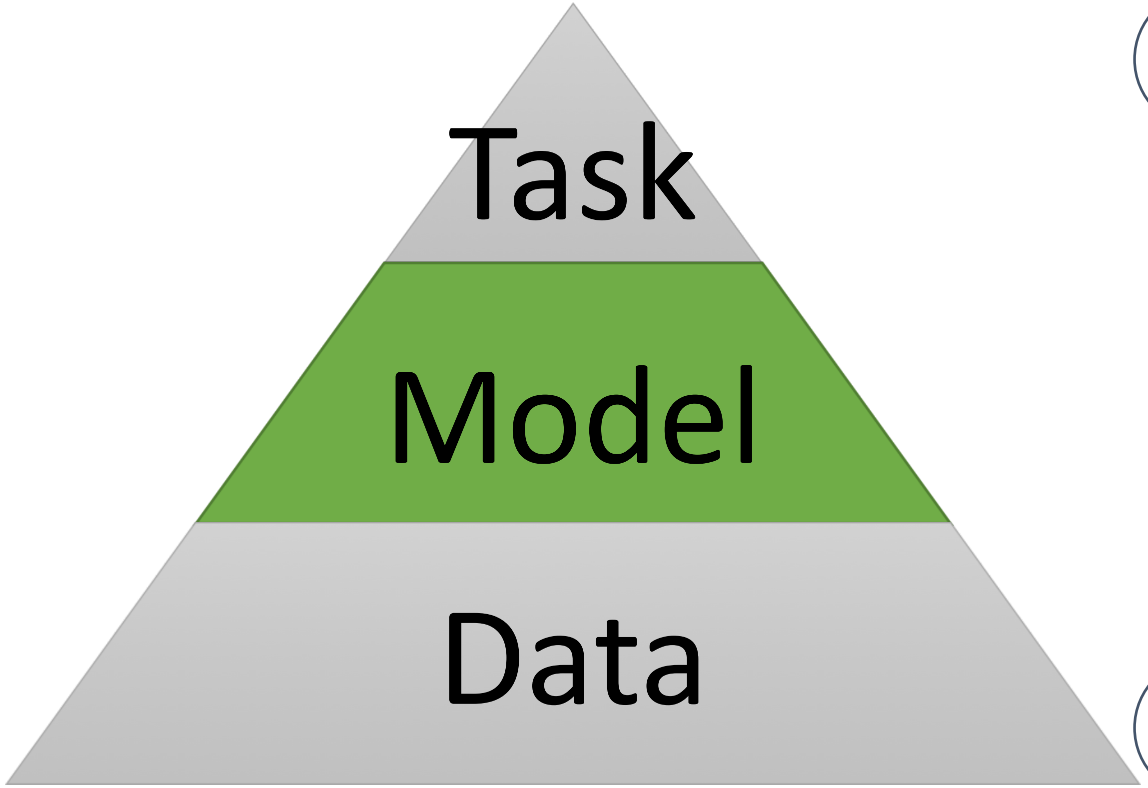




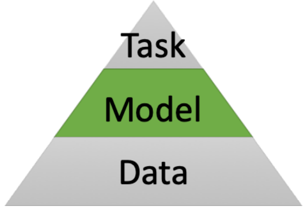
Petuum

Carnegie Mellon University
School of Computer Science

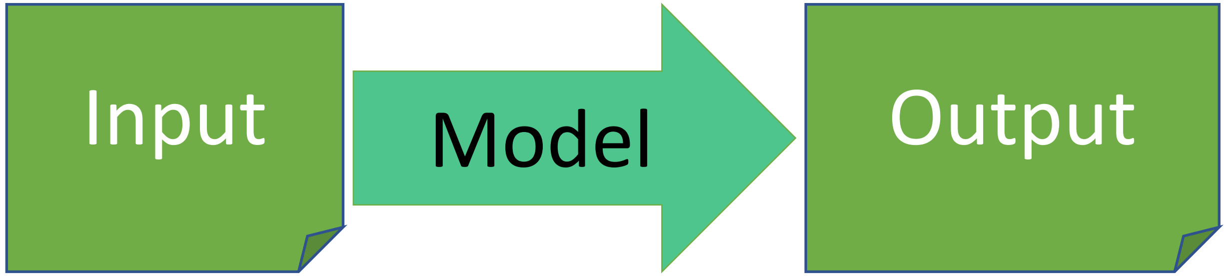
How to Standardize Model?



- Full Spectrum Pipeline
- Inter-Operation Across Tasks
- Easy Pipeline Customization
- Flexible Model Building
- Versatile Data Support

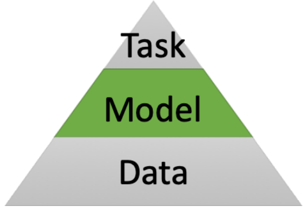


He is **going** to the book shop.



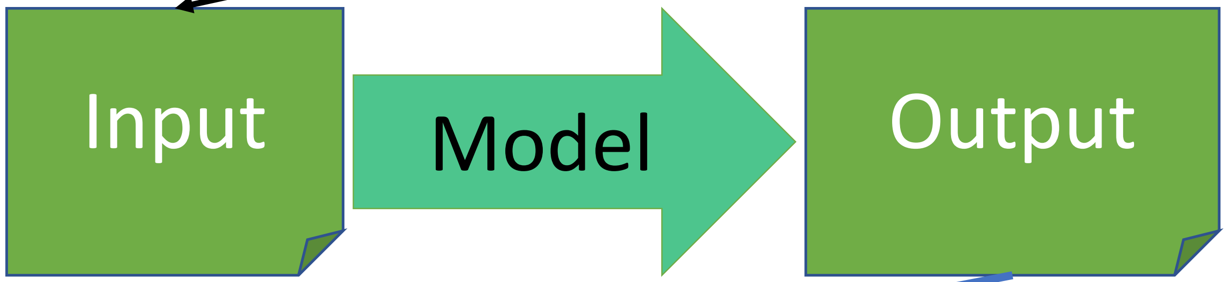
ARG0 ARGM-LOC

He is **going** to the book shop.



He is **going** to the book shop.

```
data_frame.get_data(granularity=sentence,
                    structure=[PredicateMention, EntityMention])
```



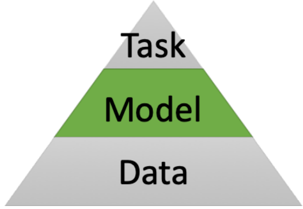
We already have a standard interface between the data and the model

```
data_frame.write_data(granularity=sentence, structure=PredicateLink)
```

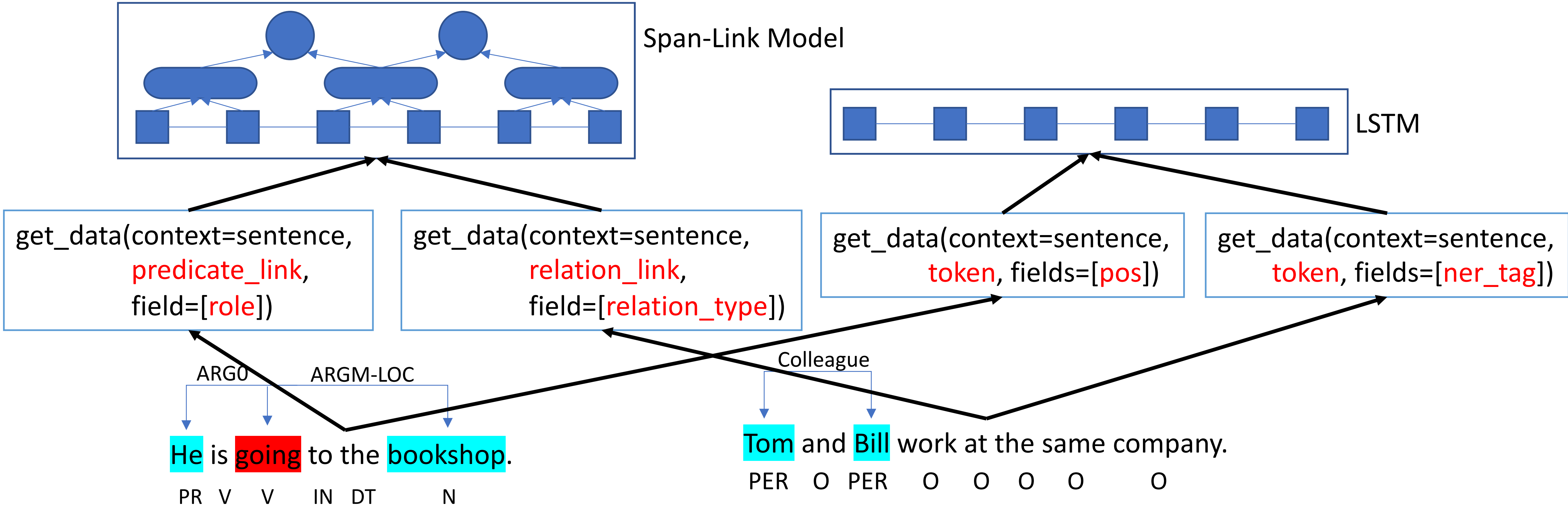
ARG0 ARGM-LOC

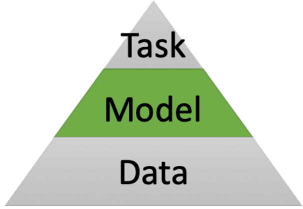
He is **going** to the book shop.

Model-Data Interface



By changing a few parameters, we can fit to different models.



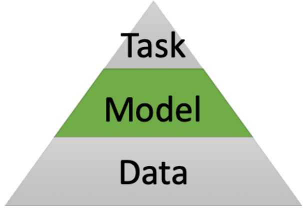


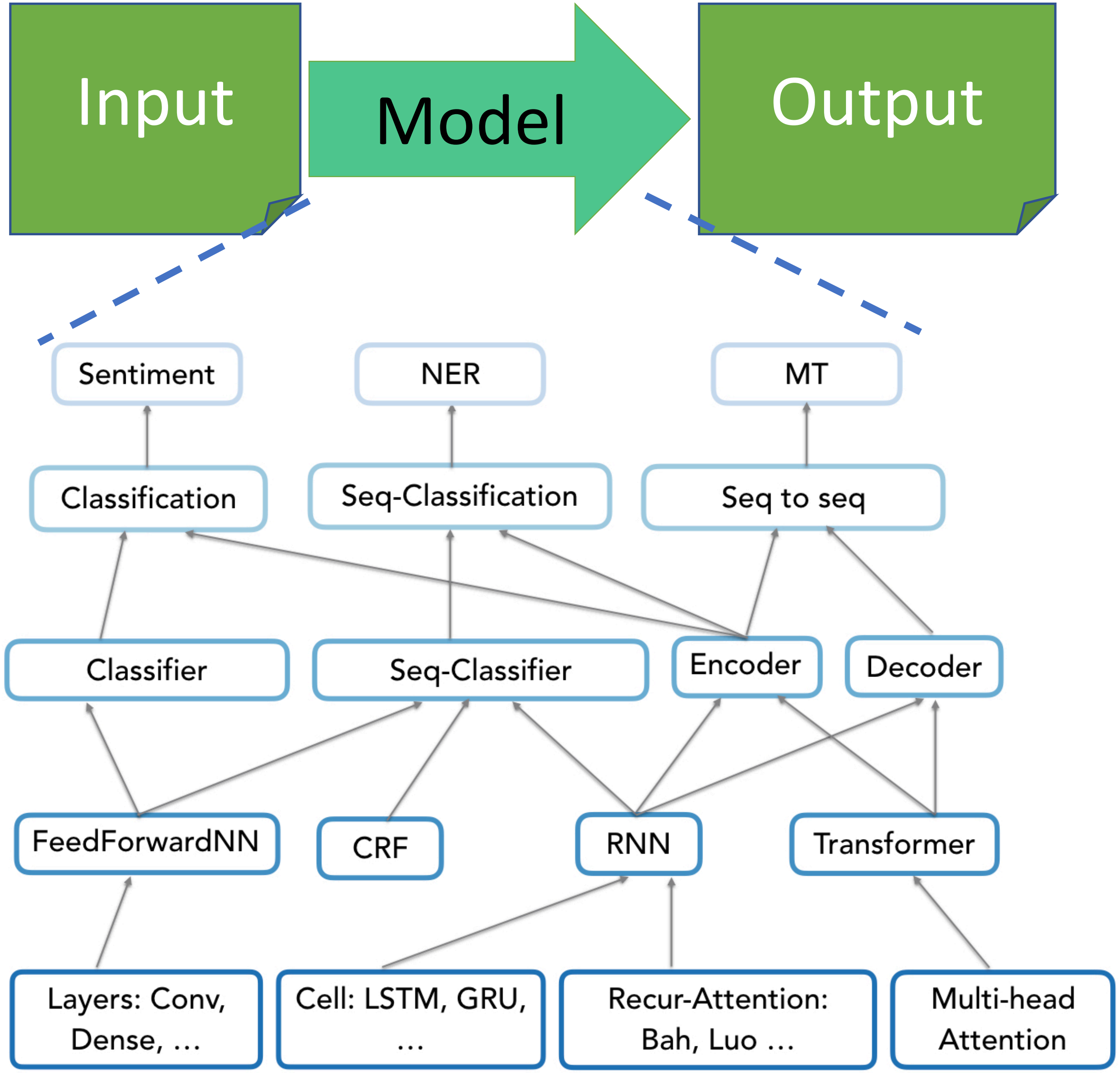
Other Model Support

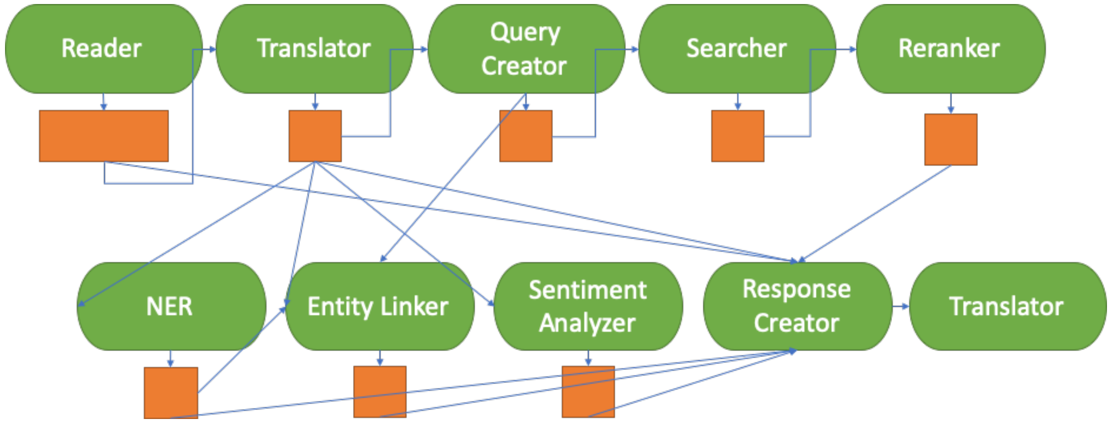
- Feature Extraction
  - E.g. Embedding support
- Training Loop Support
  - Data Iterator
  - Batcher
- Learning Library Interface

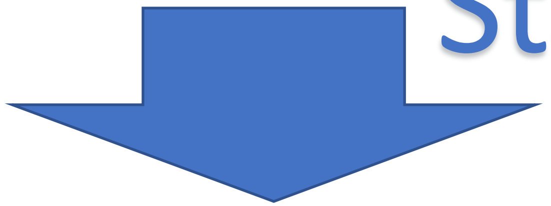
```
vector = sentence.embedding(type='BERT')
```

```
batcher = FixSizeBatcher(num_instance=8)

for data in get_data(context=sentence, relation_link,
                     field=[relation_type]):
    batcher.add(data)

    if batcher.ready():
        batcher.yield()
```

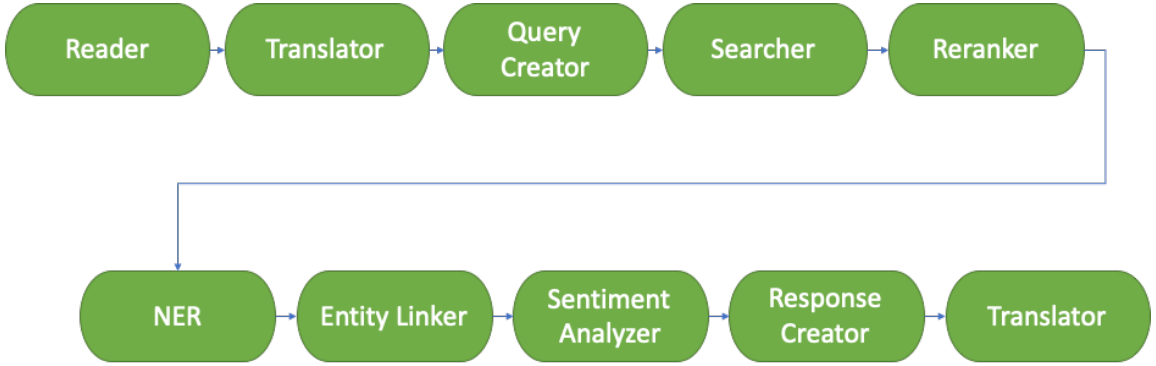

Task
Model
Data


Input
Model
Output
Sentiment
NER
MT
Classification
Seq-Classification
Seq to seq
Classifier
Seq-Classifier
Encoder
Decoder
FeedForwardNN
CRF
RNN
Transformer
Layers: Conv, Dense, …
Cell: LSTM, GRU, …
Recur-Attention: Bah, Luo …
Multi-head Attention





Standardize NLP



- Full Spectrum Pipeline
- Inter-Operation Across Tasks
- Easy Pipeline Customization
- Flexible Model Building
- Advanced Data Support

Case Study: Modularization with *Forte*

Forte: a flexible and Powerful NLP Pipeline FOR TExt.

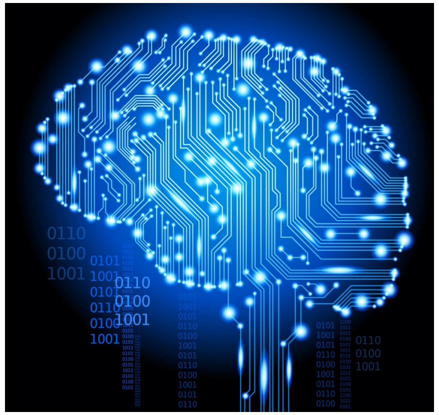
Forte

https://github.com/asyml/forte

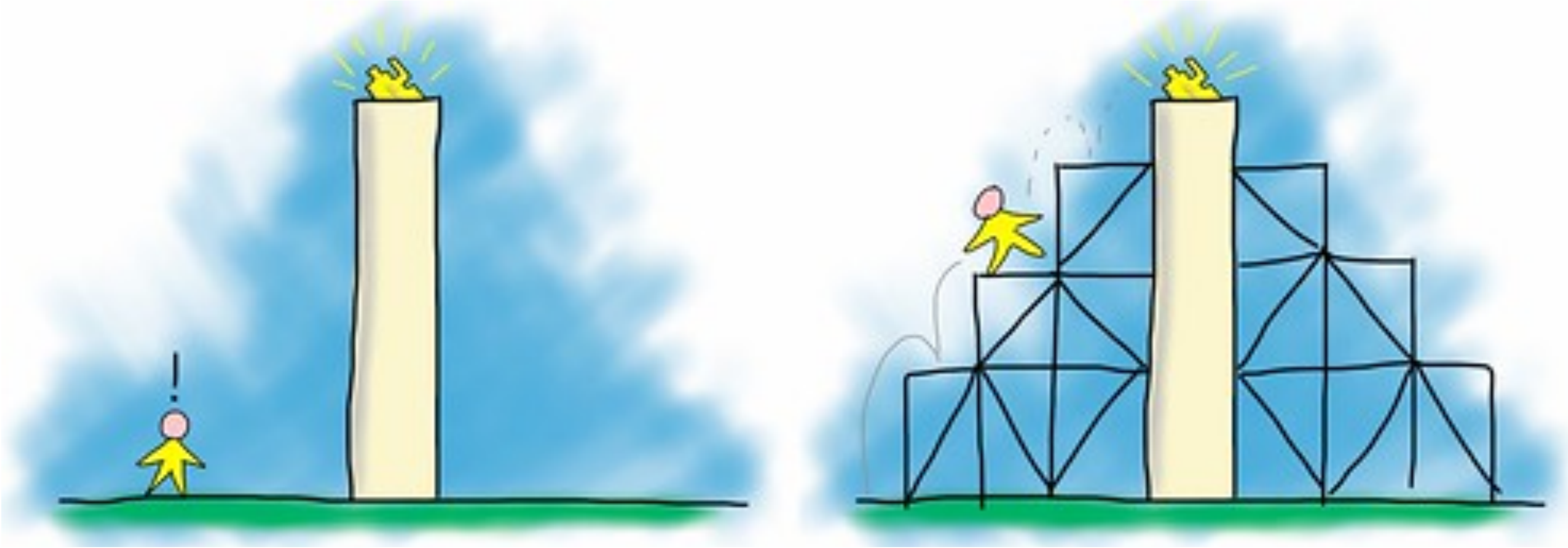
- We built **Forte**, a system that validates these ideas:
  - Universal Data Flow
  - Pipeline Construction
  - Full Spectrum: Information Retrieval, Text Analysis, Generation
  - Abstract Data, Model, Task Interfaces
- Other features:
  - Batching
  - Bookkeeping
  - …

Revisit the Problem

- A user **speaks German** but would like to find good **romantic movies.**
- We have a **corpus of English movie reviews.**
- What can we do?



Forte: Scaffold for the Goal



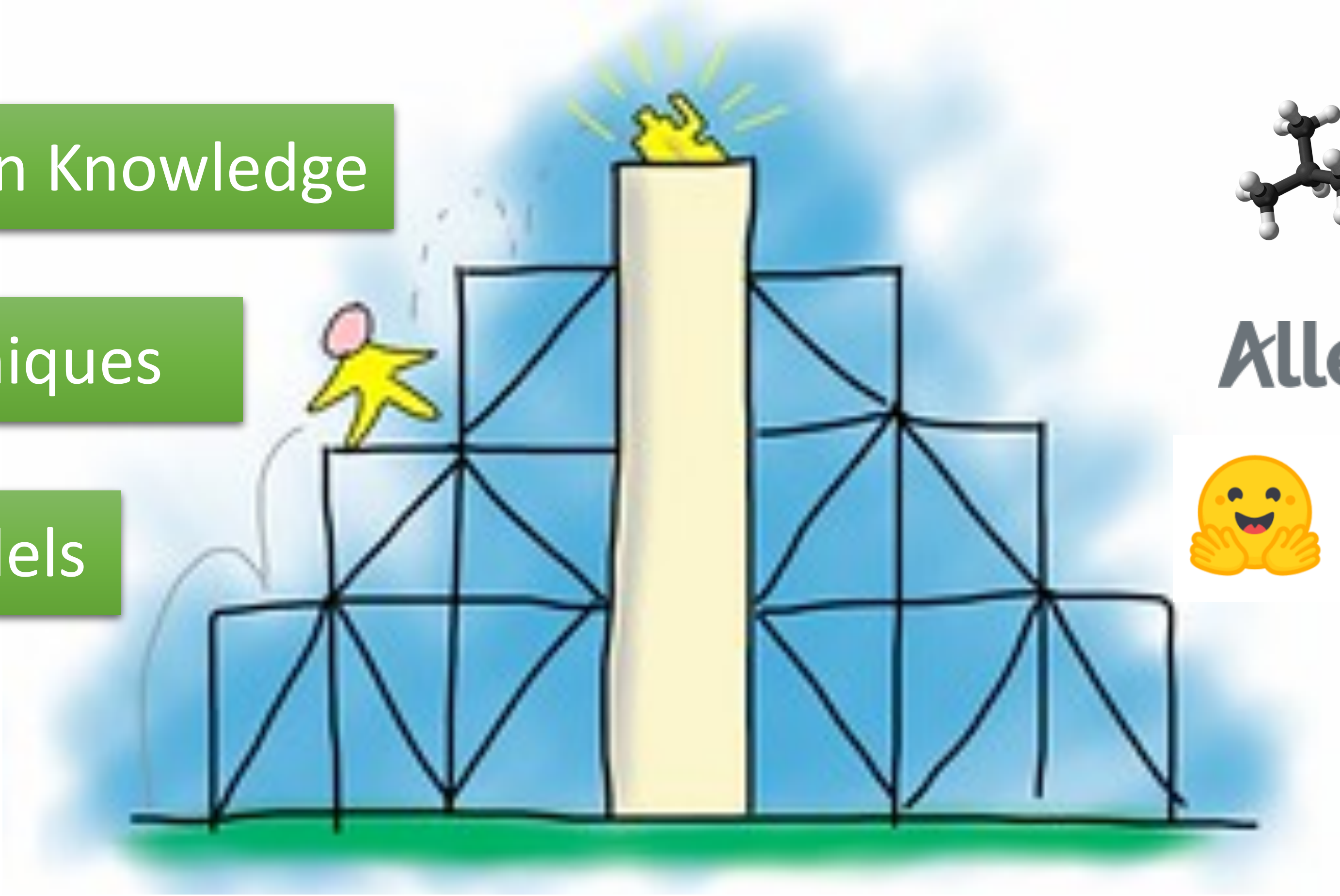
An Open, Flexible Scaffold for NLP

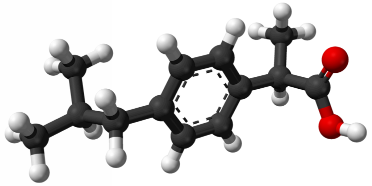
Domain Knowledge

NLP Techniques

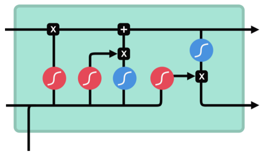
Pre-train Models

ML Library





AllenNLP



Texar

PYTORCH

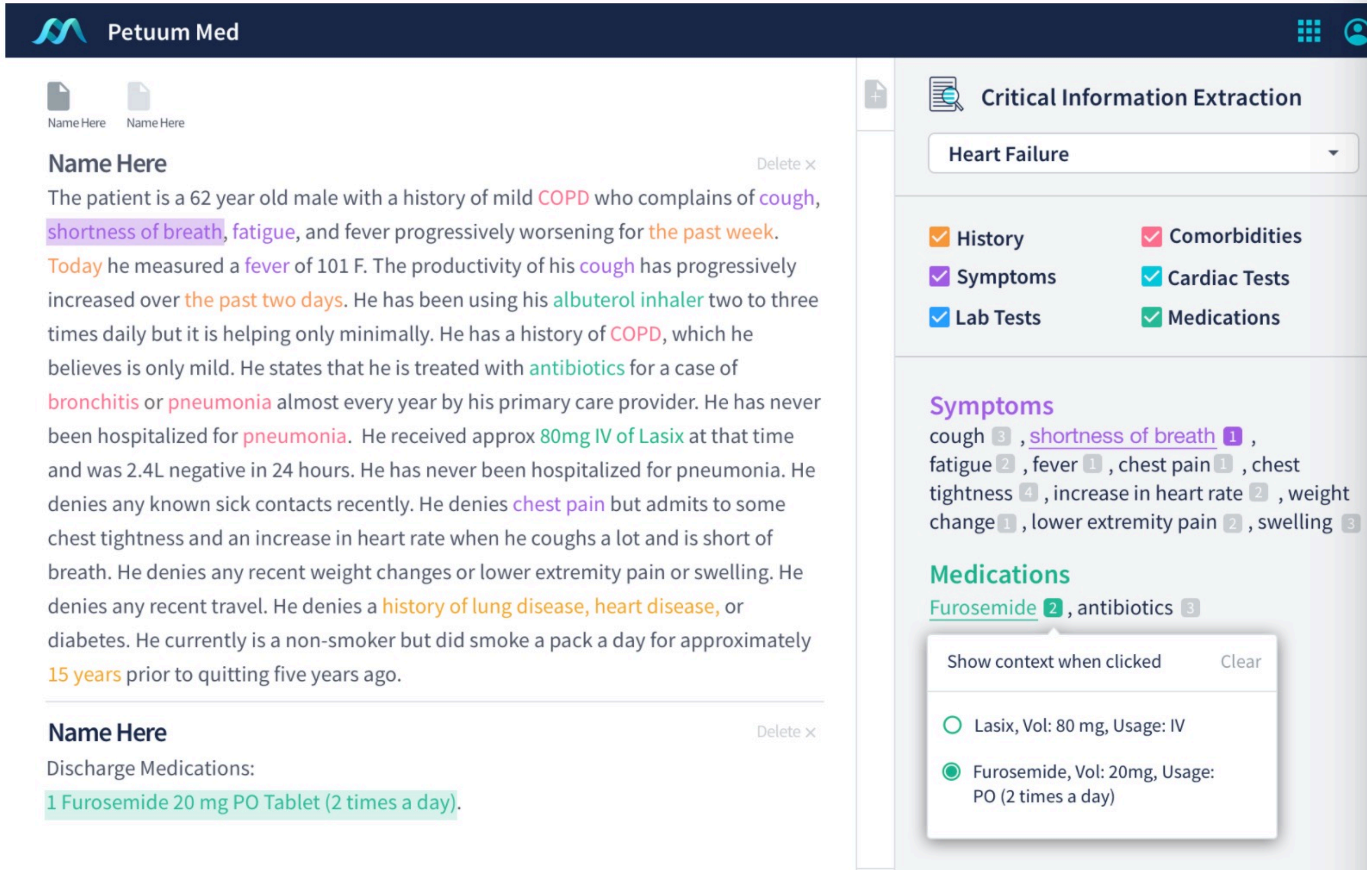
Data Representation

	Forte	spaCy	CoreNLP	DKPro	AllenNLP	Curator
Universal Data Flow	😆	😀	😆	😀	😡	😆
Extendable Ontology	😆	😡	😡	😆	😡	😡
Lossless Serialization	😆	😆	😡	😆	😡	😆
Pipeline Construction	😆	😆	😆	😆	😡	😆
Easy Processor Replacement	😆	😀	😀	😆	😡	😆
Inter Operation	😆	😀	😆	😡	😡	😆
Retrieval	😆	😡	😆	😡	😡	😡
Generation	😆	😡	😡	😡	😆	😡
Standard Model Interface	😆	😡	😡	😡	😀	😆
Deep Learning Integration	😆	😆	😀	😡	😆	😡
Integrated ML Support	😆	😀	😀	😡	😆	😆

😆 Well Supported! 😀 Some Support 😡 No Support

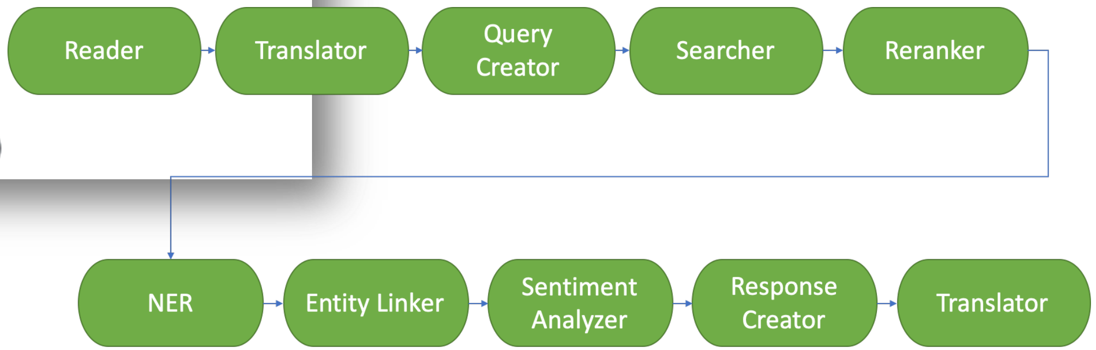
Standardization with Forte

Forte



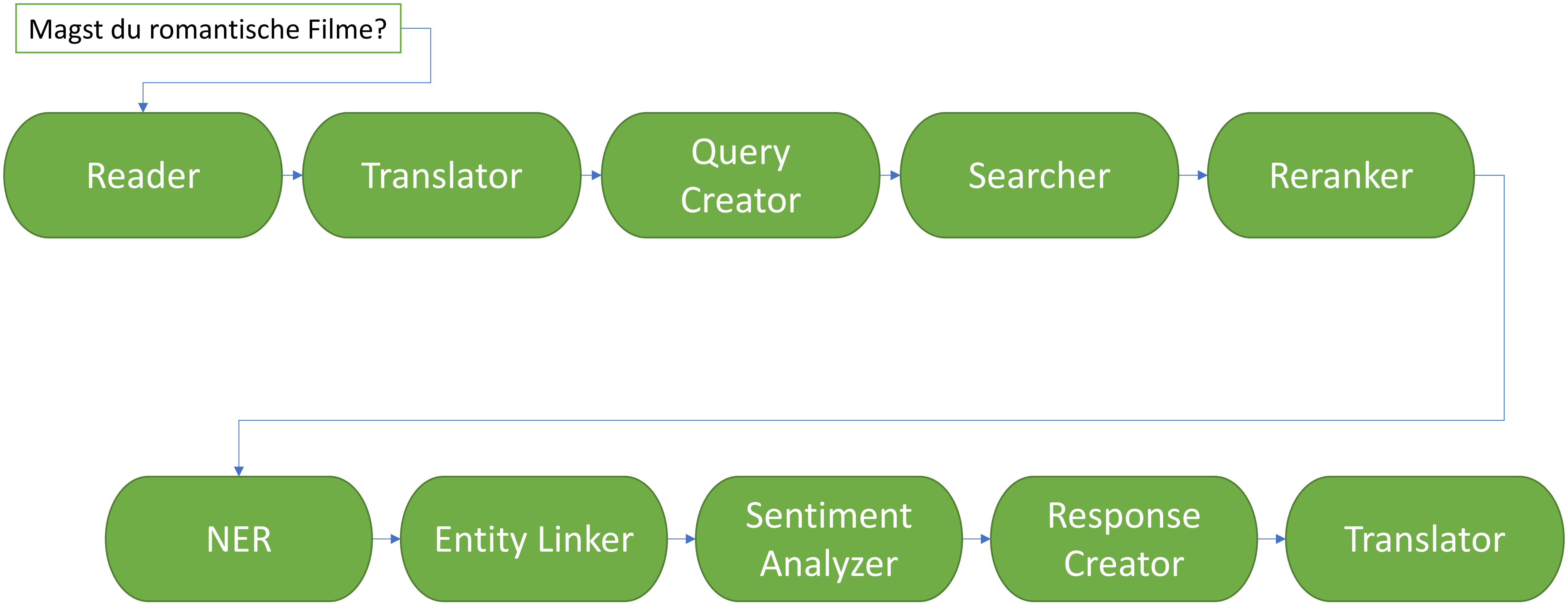
Chaining with Forte

```
query_pipeline = Pipeline(resource=resource)
query_pipeline.set_reader(
    reader=MultiPackTerminalReader(), config=config.reader)

query_pipeline.add_processor(
    processor=MicrosoftBingTranslator(), config=config.translator)
query_pipeline.add_processor(
    processor=BertBasedQueryCreator(), config=config.query_creator)
query_pipeline.add_processor(
    processor=SearchProcessor(), config=config.indexer)
query_pipeline.add_processor(
    processor=CoNLLNERPredictor(), config=config.ner,
    selector=NameMatchSelector(
        select_name=config.indexer.response_pack_name[0]))
```

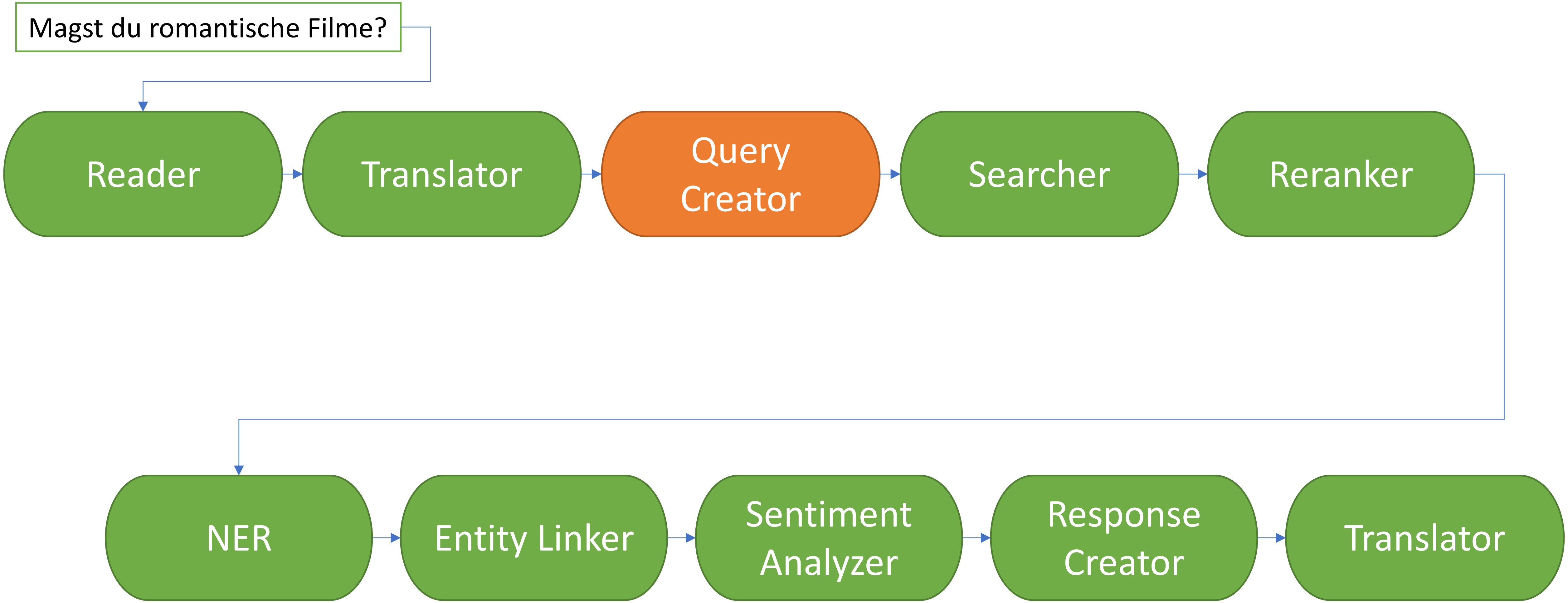


Petuum
Carnegie Mellon University
School of Computer Science

Implement Processors



Implement Processors



Query Creator

Forte

Here we create a simple text-based query.

```python
class ElasticSearchQueryCreator(QueryProcessor[MultiPack]):

    def _build_query(self, text: str) -> Dict[str, Any]:
        r"""Constructs Elasticsearch query that will be consumed by
        Elasticsearch processor.

        Args:
            text: str
              A string which will be looked up for in the corpus under field
              name `field`. `field` can be passed in a `config` during
              :meth:`ElasticSearchQueryCreator::initialize`. If `config` does
              not contain the key `field`, we will set it to "content"
        """
        size = self.config.size or 1000
        field = self.config.field or "content"
        return {"query": {"match": {field: text}}, "size": size}
```

Build Simple Query

Petuum

Carnegie Mellon University
School of Computer Science

Query Creator

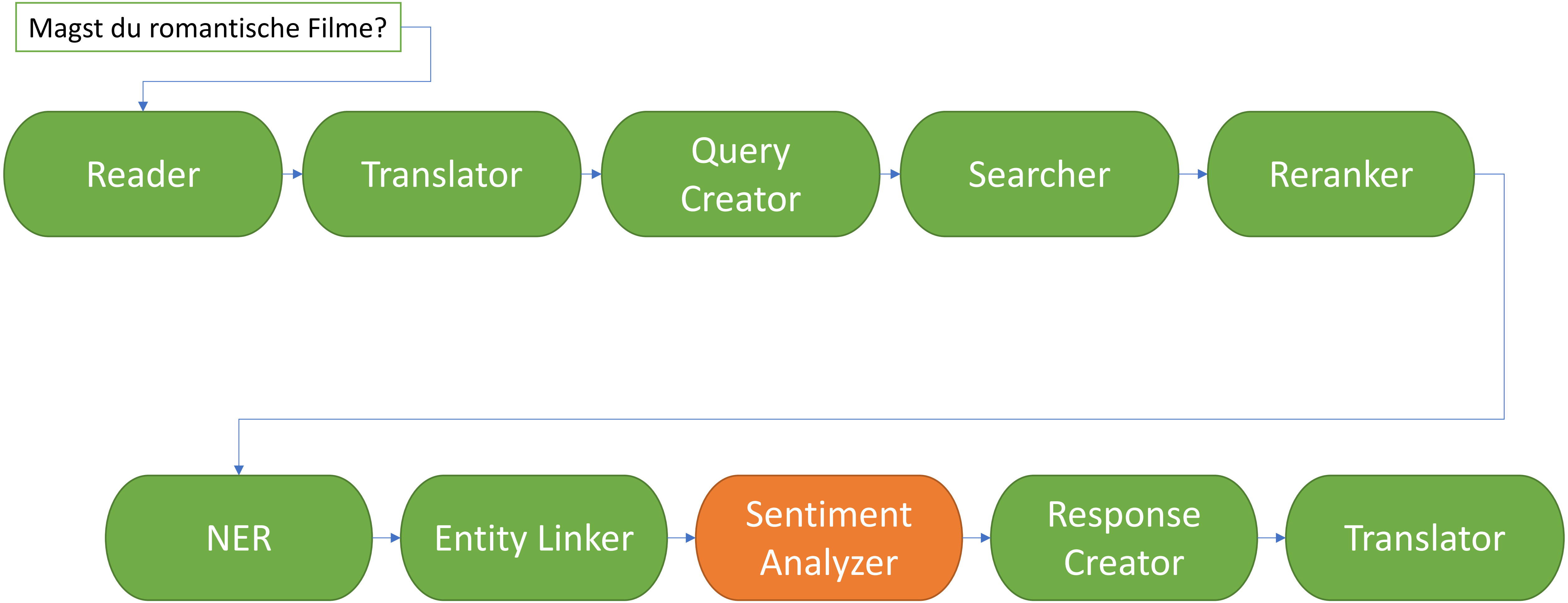
Forte

Alternatively, we can create an embedding based query.

```python
class BertBasedQueryCreator(QueryProcessor[MultiPack]):
    r"""This processor searches relevant documents for a query"""

    @torch.no_grad()
    def get_embeddings(self, inputs, sequence_length, segment_ids):
        output, _ = self.encoder(inputs=inputs,
                                 sequence_length=sequence_length,
                                 segment_ids=segment_ids)
        cls_token = output[:, 0, :]

        return cls_token

    def _build_query(self, text: str) -> np.ndarray:
        input_ids, segment_ids, input_mask = \
            self.tokenizer.encode_text(
                text_a=text, max_seq_length=self.config.max_seq_length)
        input_ids = torch.LongTensor(input_ids).unsqueeze(0).to(self.device)
        segment_ids = torch.LongTensor(segment_ids).unsqueeze(0).to(self.device)
        input_mask = torch.LongTensor(input_mask).unsqueeze(0).to(self.device)
        sequence_length = (1 - (input_mask == 0)).sum(dim=1)
        query_vector = self.get_embeddings(inputs=input_ids,
                                           sequence_length=sequence_length,
                                           segment_ids=segment_ids)
        query_vector = torch.mean(query_vector, dim=0, keepdim=True)
        query_vector = query_vector.cpu().numpy()
        return query_vector
```

Build query by using the embedding

Implement Processors



Adding Sentiment Analysis

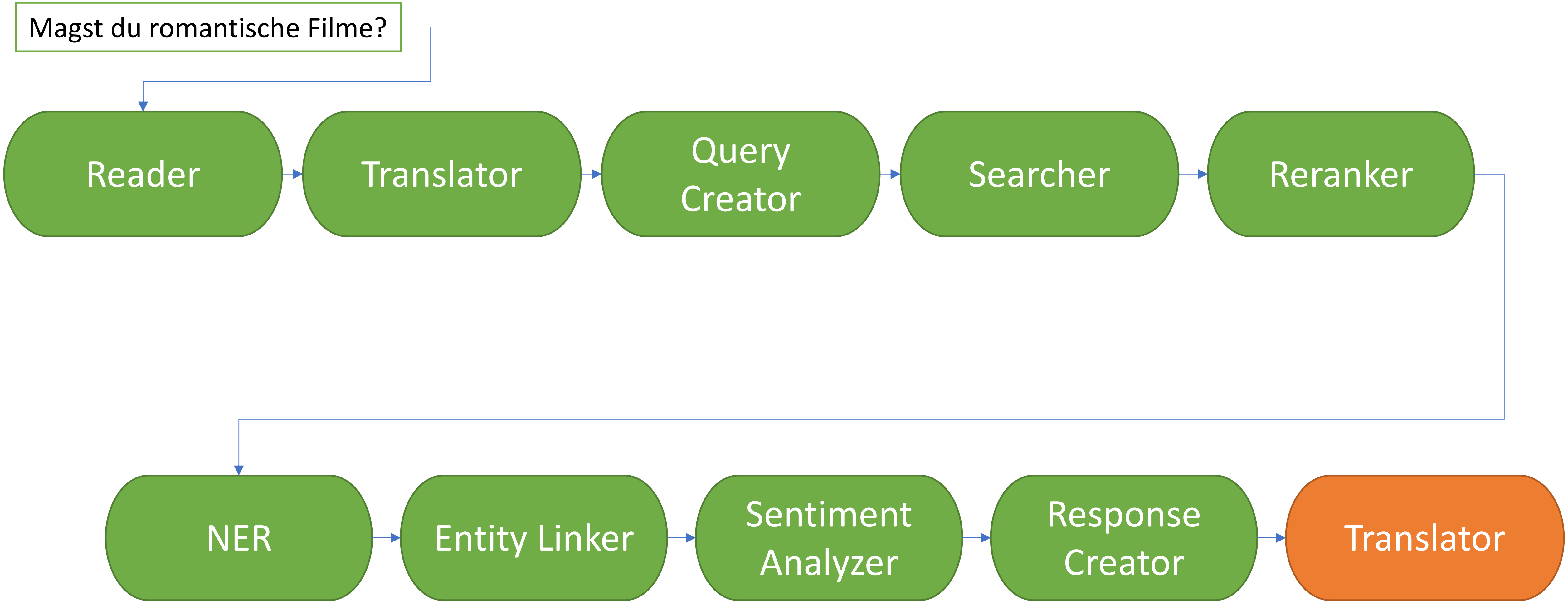
Forte

- We can easily wrap an external sentiment analysis system

```
class VaderSentimentProcessor(PackProcessor):

    def __init__(self):
        super().__init__()
        self.sentence_component = None
        self.analyzer = SentimentIntensityAnalyzer()

    def initialize(self, resource: Resources, configs: HParams):
        self.sentence_component = configs.get('sentence_component')

    def _process(self, input_pack: DataPack):
        sentence: Sentence
        for sentence in input_pack.get(entry_type=Sentence,
                                       component=self.sentence_component):
            scores = self.analyzer.polarity_scores(sentence.text)
            sentence.sentiment = scores
```

Assign the scores to each sentence.

Implement Processors



Adding a Translator

Forte

Get translation output with Bing API

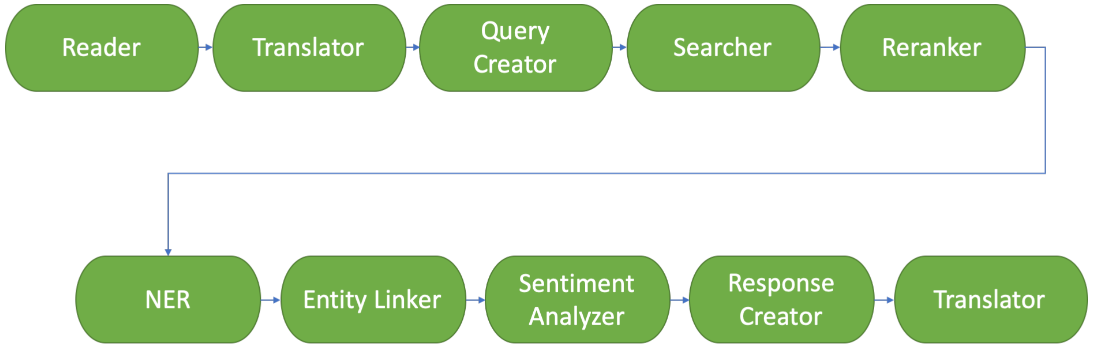
Add translated results to data

```
def _process(self, input_pack: MultiPack):
    query = input_pack.get_pack(self.in_pack_name).text
    params = '?' + urlencode(
        {'api-version': '3.0',
         'from': self.src_language,
         'to': [self.target_language]}, doseq=True)
    microsoft_constructed_url = self.microsoft_translate_url + params

    response = requests.post(
        microsoft_constructed_url, headers=self.microsoft_headers,
        json=[{"text": query}])

    if response.status_code != 200:
        raise RuntimeError(response.json()['error']['message'])

    text = response.json()[0]["translations"][0]["text"]
    pack = DataPack()

    document = Document(pack, 0, len(text))
    utterance = Utterance(pack, 0, len(text))
    pack.add_entry(document)
    pack.add_entry(utterance)

    pack.set_text(text=text)
    input_pack.update_pack({self.out_pack_name: pack})
```

Inter-operation

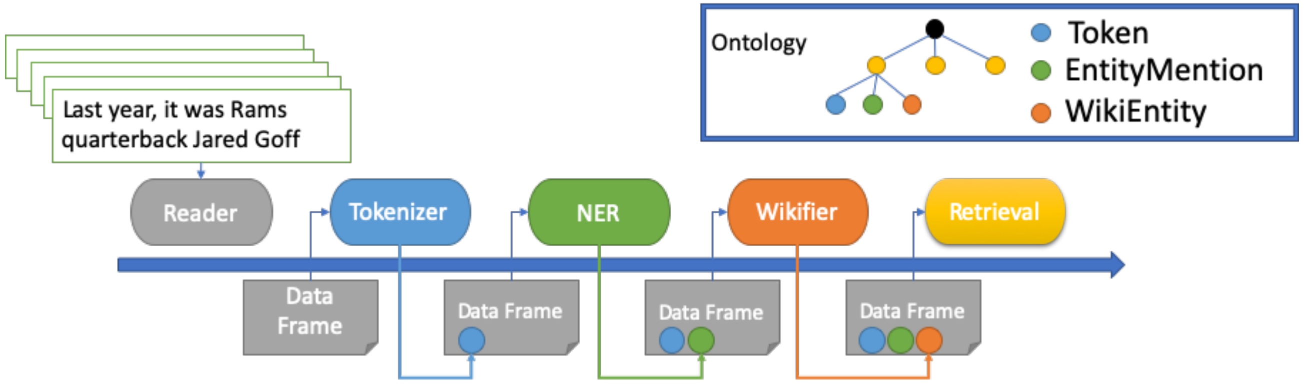
Forte

- User: Kennst du ein paar gute romantische Filme?
- Response: *Yes, Titanic. Kate Winslet and Leonardo Dicaprio have definitely created a timeless classic.*
    - Response selected by inter-operation
    - It contains the actors (NER + Entity Linking)
    - It contains the sentiment (Sentiment Analyzer)
- Informed decisions can be made with a well-designed pipeline



Some Take-home Messages

- Use standard and shared data representation
- NLP concepts can be categorized
- Enrich data, don't delete data
- Keep consistent interfaces between models and tasks
- Understand your domain

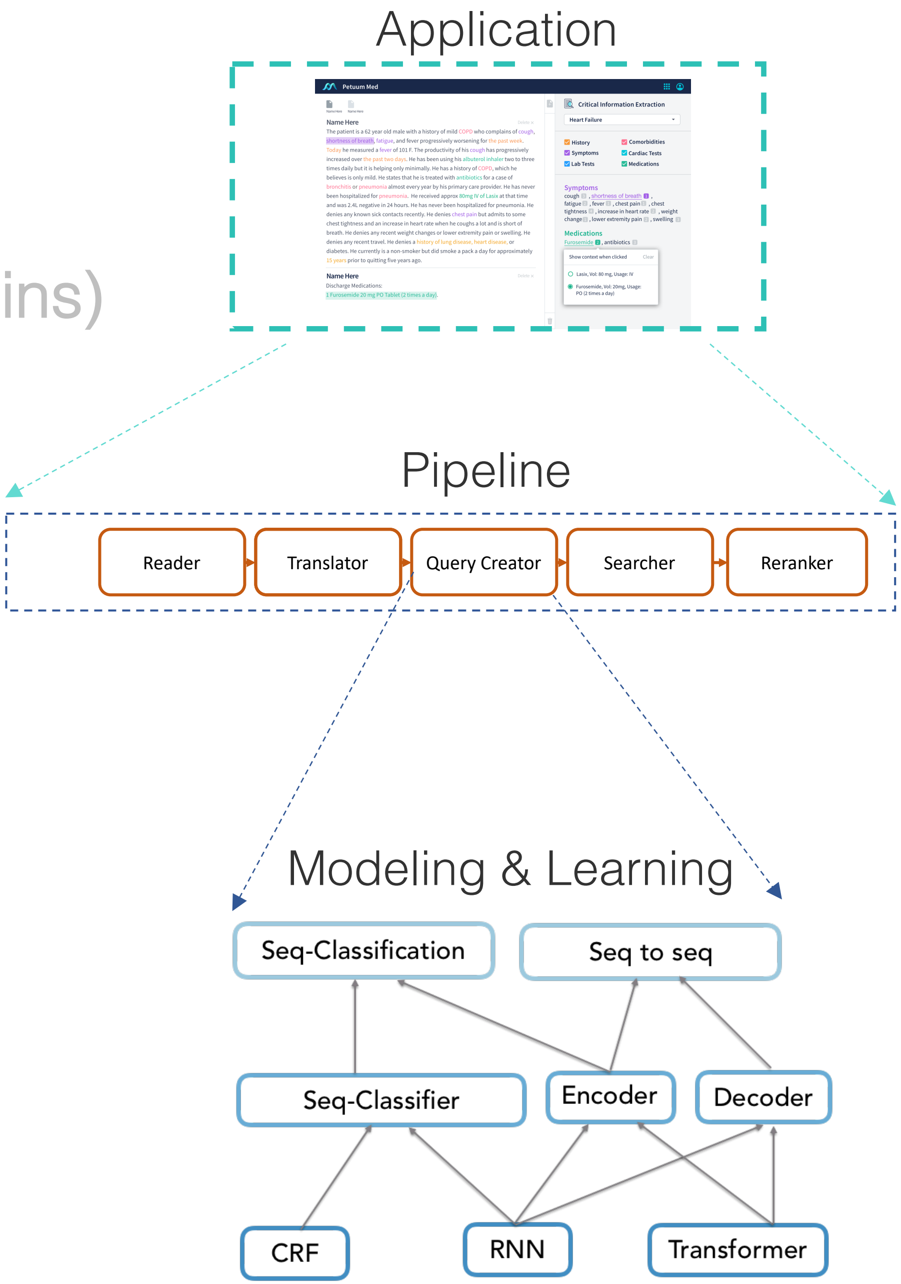


Agenda

- Natural Language Processing Overview (10mins)
- Modularizing NLP Pipeline (35mins)
    - Complexity of NLP pipeline
    - A standardized view of NLP pipeline
    - A standardized implementation of NLP pipeline
- Short break & QA (5mins)
- Modularizing NLP Model & Learning (30mins)
    - Composable ML
- QA (10mins)

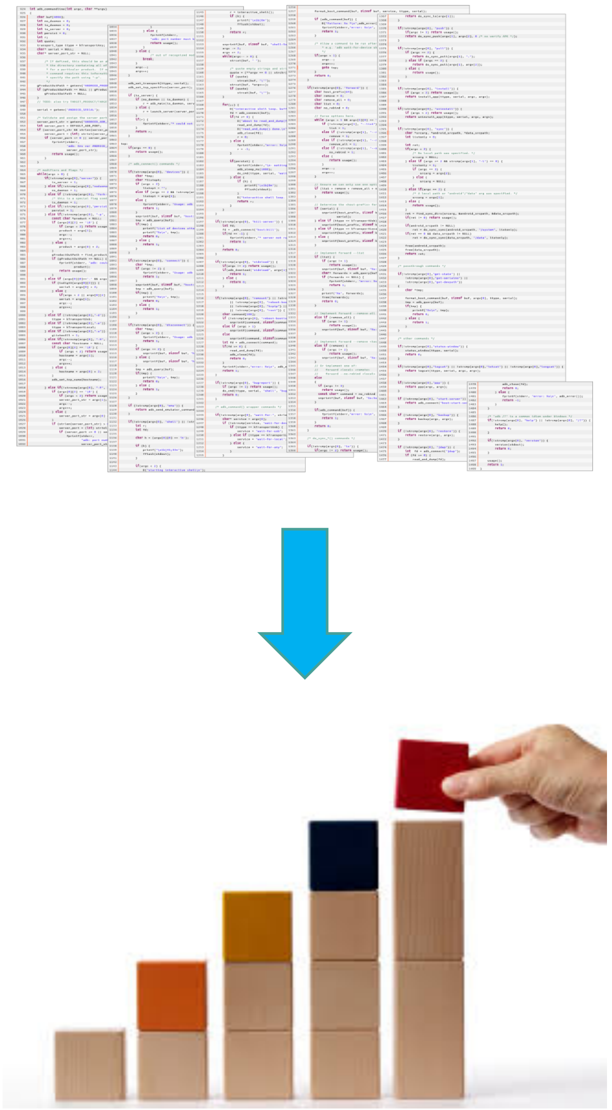
Agenda

- Natural Language Processing Overview (10mins)
- Modularizing NLP Pipeline (35mins)
    - Complexity of NLP pipeline
    - A standardized view of NLP pipeline
    - A standardized implementation of NLP pipeline
- Short break & QA (5mins)
- Modularizing NLP Model & Learning (30mins)
    - Composable ML
- QA (10mins)

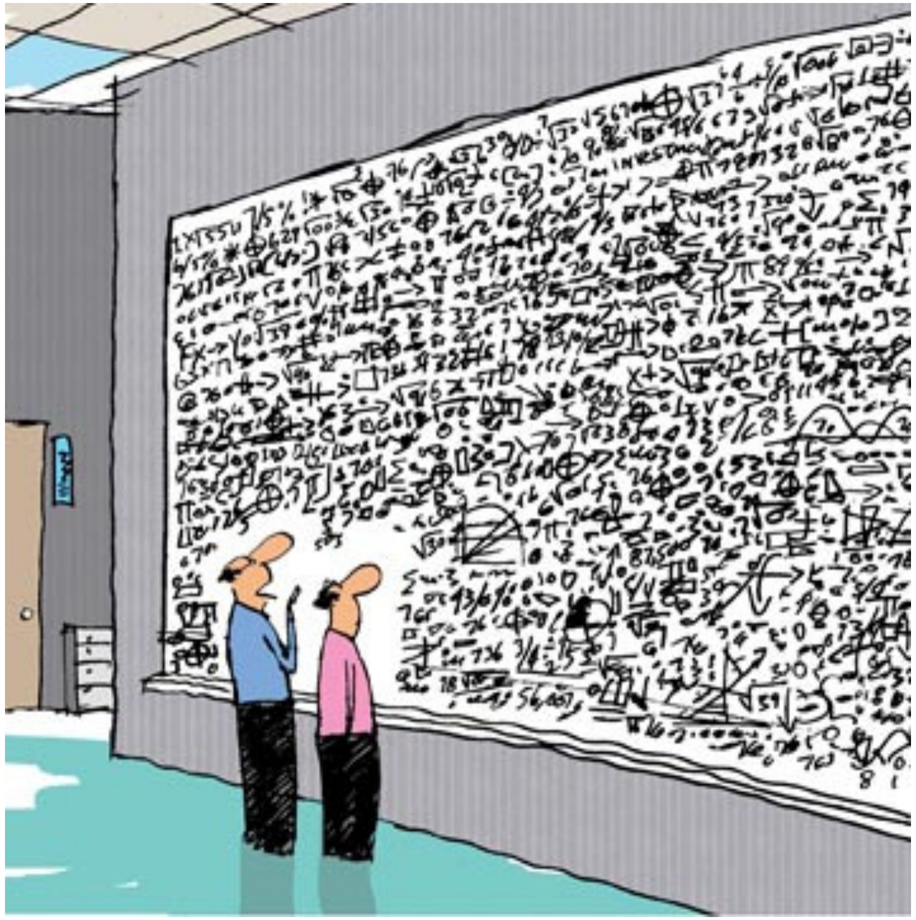


Composable ML

- Build ML models more easily, via pick-and-choose
- Stop writing same one-off code again and again
  - More reliable and easier to debug
  - Easier to onboard new developers



Decomposing Machine Learning



Model architectures
(RNNs, Transformers, Graphical, …)

Loss functions
(likelihood, reconstruction, margin, …)

Constraints
(normality, sparsity, logical, KL, sum, …)

Algorithms
MC (MCMC, Importance), Opt (gradient, IP), …

Experience
(processing, augmentation, weighting, ...)

Decomposing Machine Learning

Machine Learning:

Computational methods that enable machines to learn concepts and improve performance from experience

Decomposing Machine Learning

Machine Learning:

Computational methods that enable machines to learn concepts and improve performance from experience

$$\min_\theta \mathcal{L}(\theta, \mathcal{E}) + \Omega(\theta)$$

Decomposing Machine Learning

Machine Learning:

Computational methods that enable machines to learn concepts and improve performance from experience

$$\min_{\theta} \mathcal{L}(\theta, \mathcal{E}) + \Omega(\theta)$$

$$\boldsymbol{y} \sim p_{\theta}(\boldsymbol{y}|\boldsymbol{x})$$

model architecture/
inference procedure

Decomposing Machine Learning

Machine Learning:

Computational methods that enable machines to learn concepts and improve performance from experience

$$\min_{\theta} \mathcal{L}(\theta, \mathcal{E}) + \Omega(\theta)$$

$(\boldsymbol{x}^*, \boldsymbol{y}^*)$

model architecture/
inference procedure

experience
(e.g., data)

Decomposing Machine Learning

Machine Learning:

Computational methods that enable machines to learn concepts and improve performance from experience

$$\min_{\theta} \mathcal{L}(\theta, \mathcal{E}) + \Omega(\theta)$$

loss

model architecture/ inference procedure

experience (e.g., data)

Decomposing Machine Learning

Machine Learning:

Computational methods that enable machines to learn concepts and improve performance from experience

$$\min_\theta \mathcal{L}(\theta, \mathcal{E}) + \Omega(\theta)$$

loss

model architecture/ inference procedure

experience (e.g., data)

constraint

Decomposing Machine Learning

Machine Learning:

Computational methods that enable machines to learn concepts and improve performance from experience

$$\min_\theta \mathcal{L}(\theta, \mathcal{E}) + \Omega(\theta)$$

learning procedure

loss

model architecture/ inference procedure

experience (e.g., data)

constraint

$$\min_\theta \mathcal{L}(\theta, \mathcal{E}) + \Omega(\theta)$$

Running Example: Machine Translation

raw data

cleaning
tokenizing
vocabular
y
truncation
…

source.dat

I like this movie.

Lovely and poignant

Insanely hilarious!

…

target.dat

Ich mag diesen film.

Schön und ergreifend

Wahnsinnig witzig!

…

clean data

model

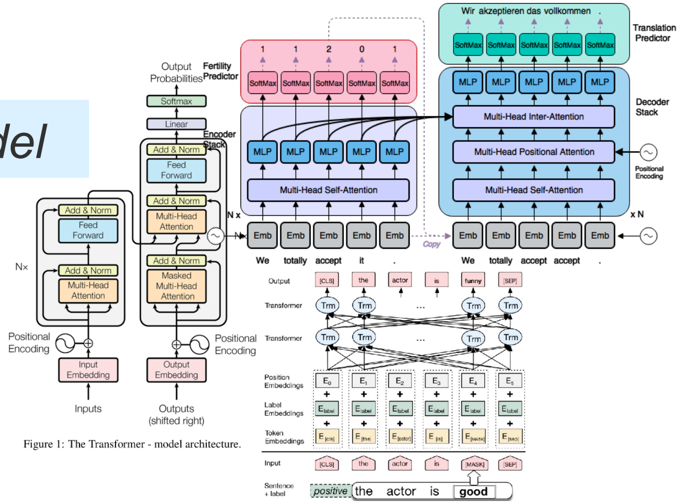


Figure 1: The Transformer - model architecture.

training

Maximum likelihood training

Reinforcement
Adversarial learning
Finetuning

evaluation
post-processing

…

ML Components

$\min_\theta \mathcal{L}(\theta, D) + \Omega(\theta)$

Constraint

Loss

Learning

Inference

Architecture

ML Components

$\min_\theta \mathcal{L}(\theta, D) + \Omega(\theta)$

Constraint

Loss

Learning

Inference

Architecture

Architecture (1): Language Model

$\min_{\theta} \mathcal{L}(\theta, D) + \Omega(\theta)$

- Calculates the probability of a sentence:
  - Sentence:

$$\boldsymbol{y} = (y_1, y_2, \dots, y_T)$$

Example:

(I, like, this, …)

Architecture (1): Language Model

$\min_{\theta} \mathcal{L}(\theta, D) + \Omega(\theta)$

- Calculates the probability of a sentence:
  - Sentence:

$$\boldsymbol{y} = (y_1, y_2, \ldots, y_T)$$

$$p_\theta(\boldsymbol{y}) = \prod_{t=1}^{T} p_\theta(y_t \mid \boldsymbol{y}_{1:t-1})$$

Example:

(I, like, this, …)

$$\cdots p_\theta\,(like \mid I)\; p_\theta(this \mid I, like) \cdots$$

Architecture (1): Language Model

$\min_\theta \mathcal{L}(\theta, D) + \Omega(\theta)$

- Calculates the probability of a sentence:
  - Sentence:

$$\boldsymbol{y} = (y_1, y_2, \dots, y_T)$$

$$p_\theta(\boldsymbol{y}) = \prod_{t=1}^{T} p_\theta(y_t \mid \boldsymbol{y}_{1:t-1})$$

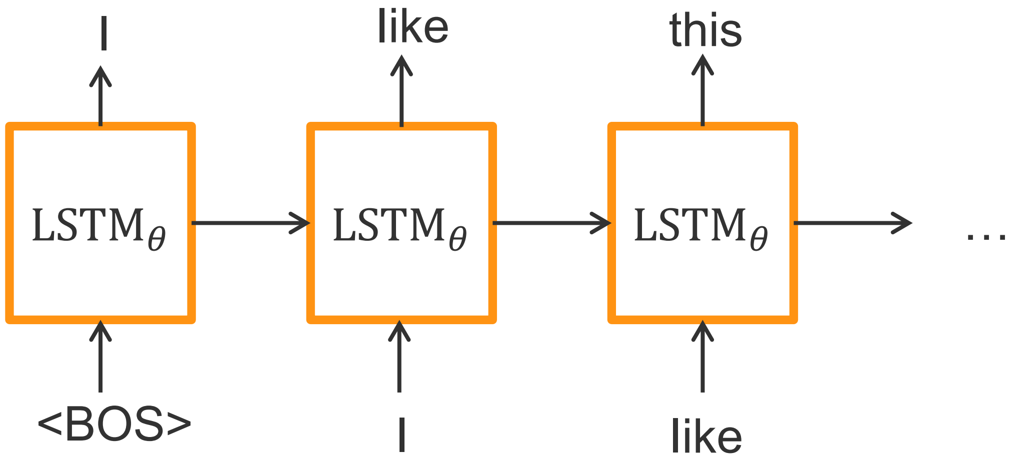
Example:

(I, like, this, …)

$$\cdots p_\theta\,(like \mid I)\; p_\theta(this \mid I, like) \cdots$$

Architecture (1.1)

LSTM RNN



Architecture (1): Language Model

$\min_\theta \mathcal{L}(\theta, D) + \Omega(\theta)$

- Calculates the probability of a sentence:
  - Sentence:

$$\boldsymbol{y} = (y_1, y_2, \ldots, y_T)$$

$$p_\theta(\boldsymbol{y}) = \prod_{t=1}^{T} p_\theta(y_t \mid \boldsymbol{y}_{1:t-1})$$

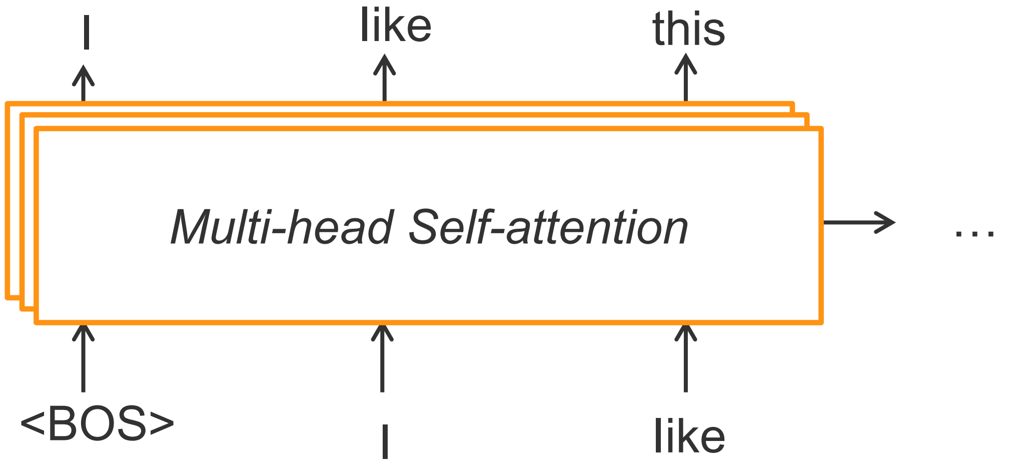
Example:

(I, like, this, …)

$$\cdots p_\theta\,(like \mid I)\, p_\theta(this \mid I, like) \cdots$$

Architecture (1.2)

Transformer



Petuum

Carnegie Mellon University
School of Computer Science

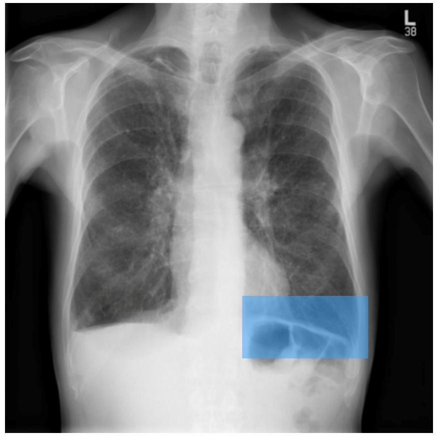
Architecture (2): Conditional Language Model

$\min_\theta \mathcal{L}(\theta, D) + \Omega(\theta)$

- Conditions on additional task-dependent context $\boldsymbol{x}$
  - Machine translation: source sentence

I like this movie. ⟶ Ich mag diesen film.

  - Medical image report generation: medical image



⟶ … There is chronic pleural-parenchymal scarring within the lung bases. No lobar consolidation is seen. …

Architecture (2): Conditional Language Model

$\min_\theta \mathcal{L}(\theta, D) + \Omega(\theta)$

- Conditions on additional task-dependent context $\boldsymbol{x}$

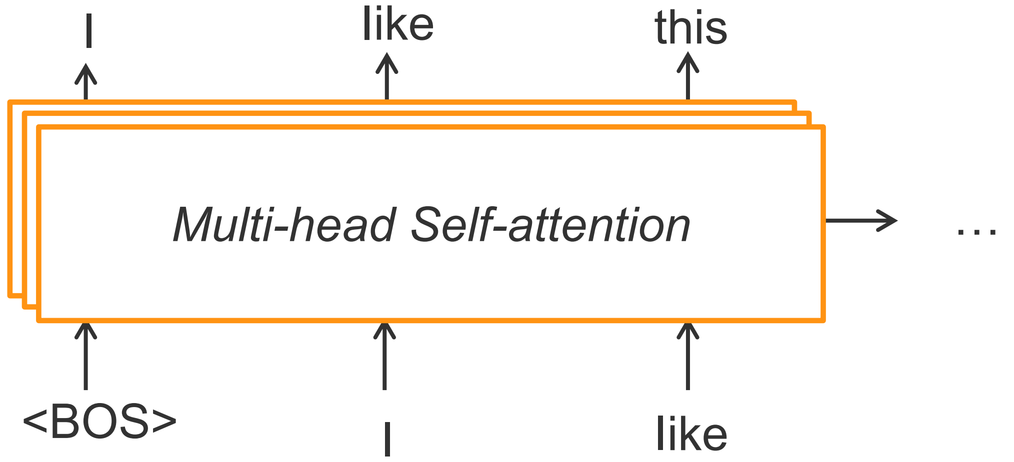
$$p_\theta(\boldsymbol{y} \mid \boldsymbol{x}) = \prod_{t=1}^{T} p_\theta(y_t \mid \boldsymbol{y}_{1:t-1}, \boldsymbol{x})$$

Architecture (2): Conditional Language Model

$\min_\theta \mathcal{L}(\theta, D) + \Omega(\theta)$

- Conditions on additional task-dependent context $\boldsymbol{x}$

$$p_\theta(\boldsymbol{y} \mid \boldsymbol{x}) = \prod_{t=1}^{T} p_\theta(y_t \mid \boldsymbol{y}_{1:t-1}, \boldsymbol{x})$$



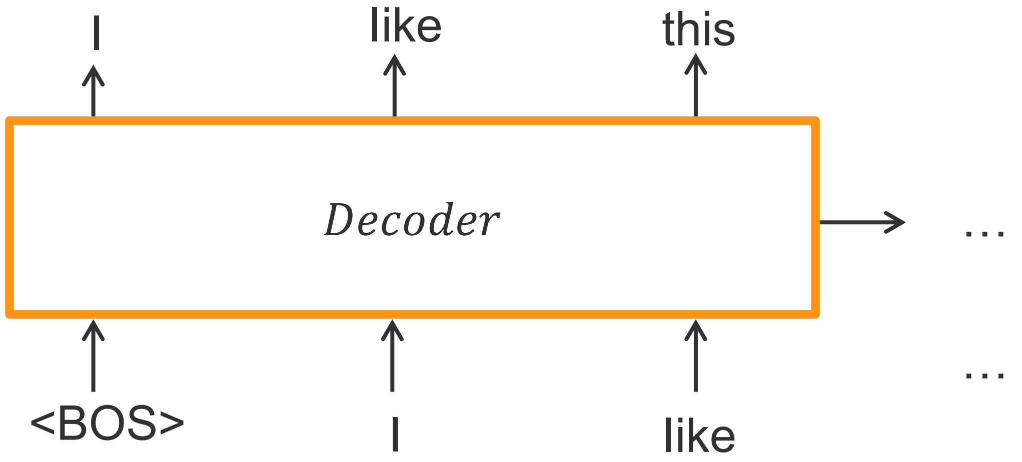
Architecture (2): Conditional Language Model

$\min_\theta \mathcal{L}(\theta, D) + \Omega(\theta)$

- Conditions on additional task-dependent context $\boldsymbol{x}$

$$p_\theta(\boldsymbol{y} \mid \boldsymbol{x}) = \prod_{t=1}^{T} p_\theta(y_t \mid \boldsymbol{y}_{1:t-1}, \boldsymbol{x})$$

- Language model as a decoder



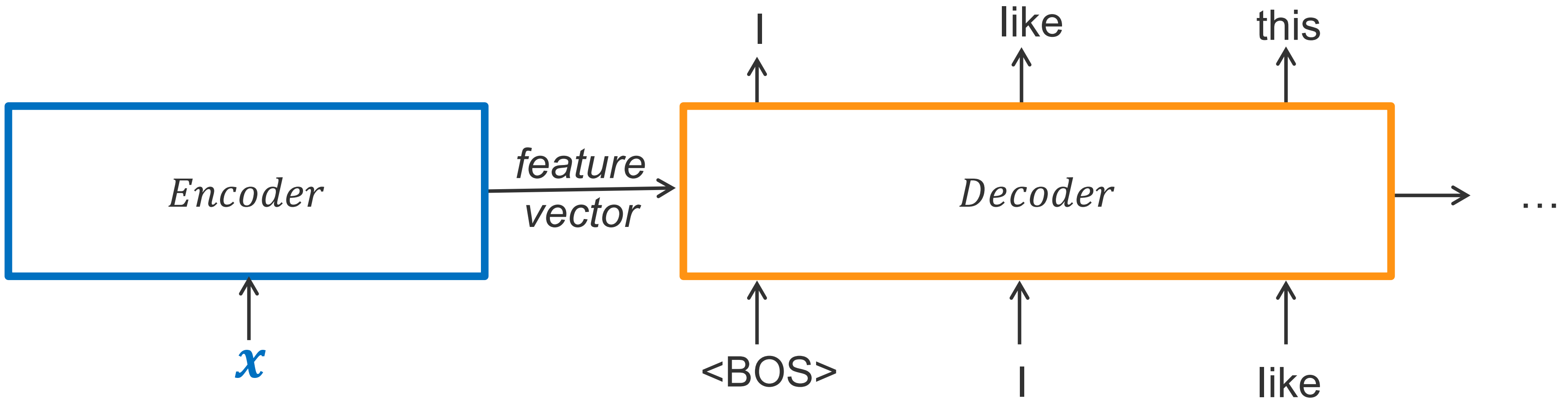
Architecture (2): Conditional Language Model

$\min_\theta \mathcal{L}(\theta, D) + \Omega(\theta)$

- Conditions on additional task-dependent context $\boldsymbol{x}$

$$p_\theta(\boldsymbol{y} \mid \boldsymbol{x}) = \prod_{t=1}^{T} p_\theta(y_t \mid \boldsymbol{y}_{1:t-1}, \boldsymbol{x})$$

- Language model as a decoder
- Encodes context with an encoder

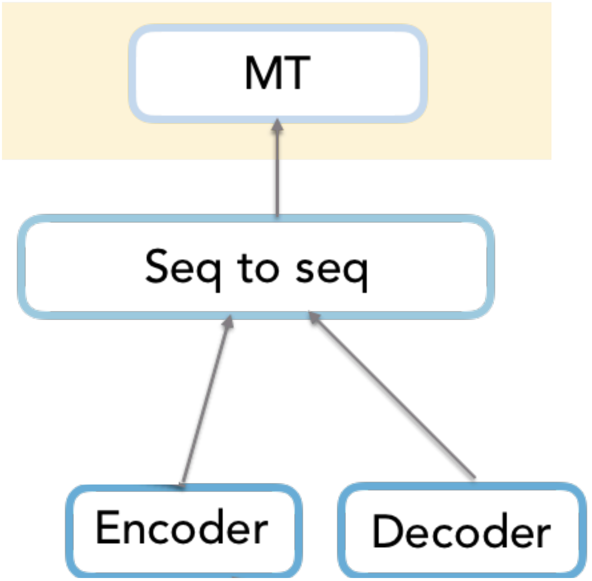


Petuum
Carnegie Mellon University
School of Computer Science

Architecture Graph

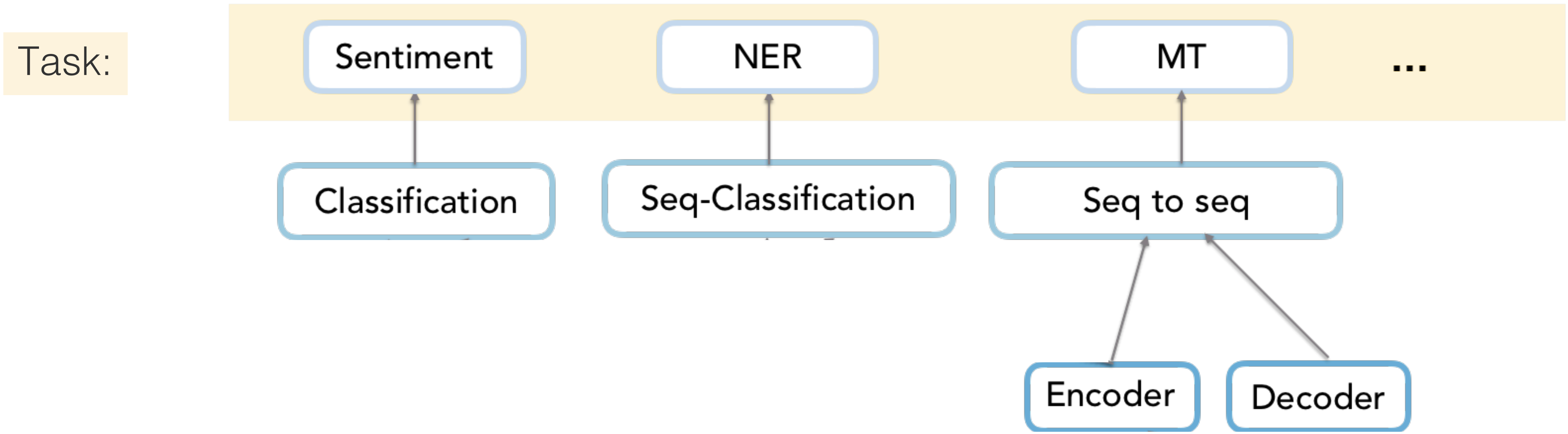
$\min_\theta \mathcal{L}(\theta, D) + \Omega(\theta)$

Task:



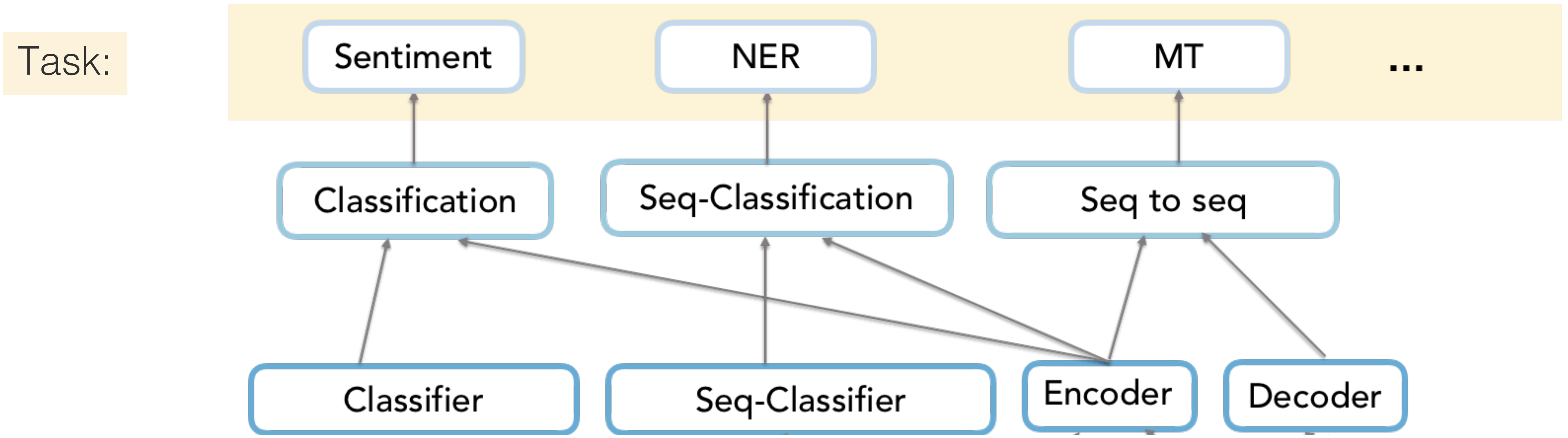
Architecture Graph

$\min_\theta \mathcal{L}(\theta, D) + \Omega(\theta)$



Architecture Graph

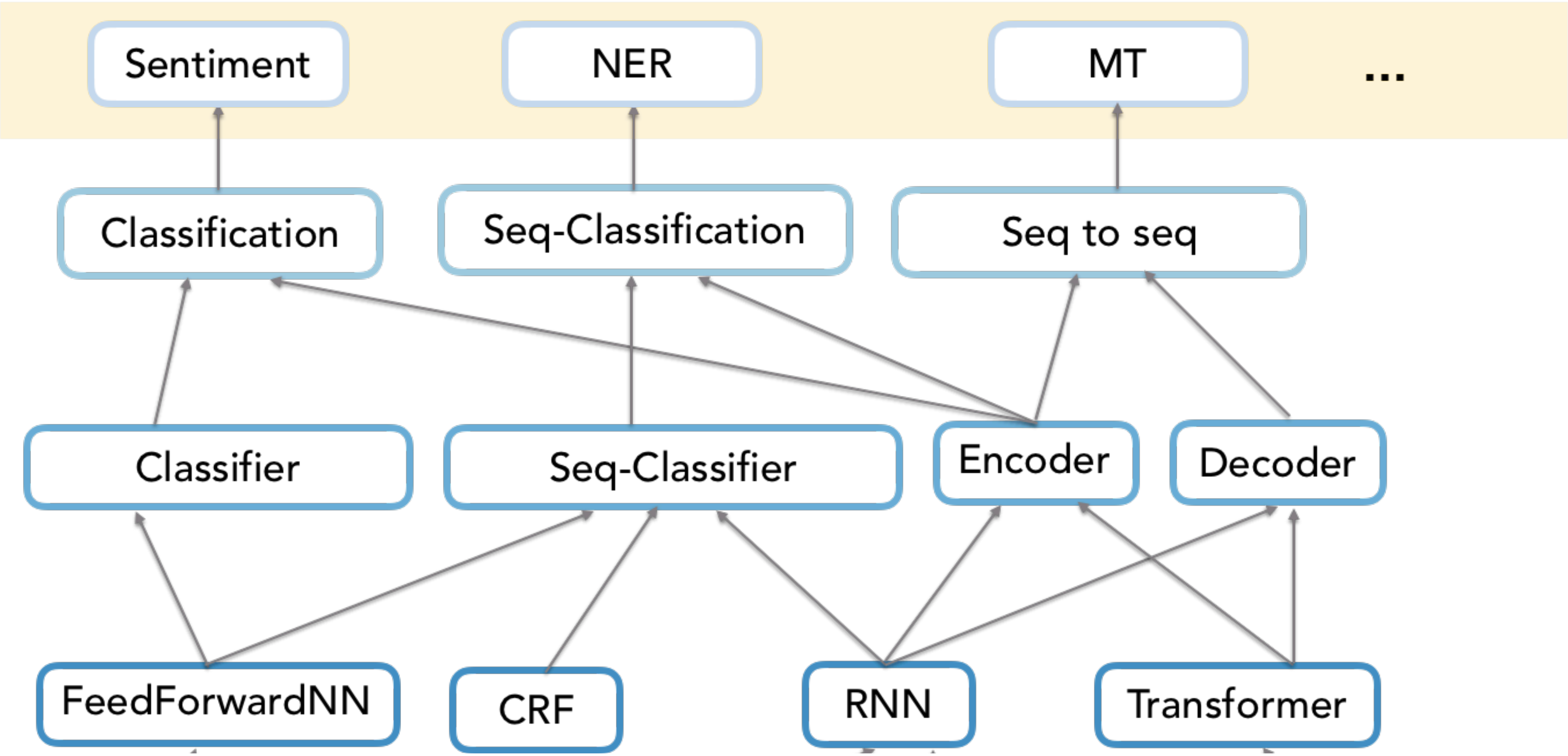
$\min_\theta \mathcal{L}(\theta, D) + \Omega(\theta)$



Architecture Graph

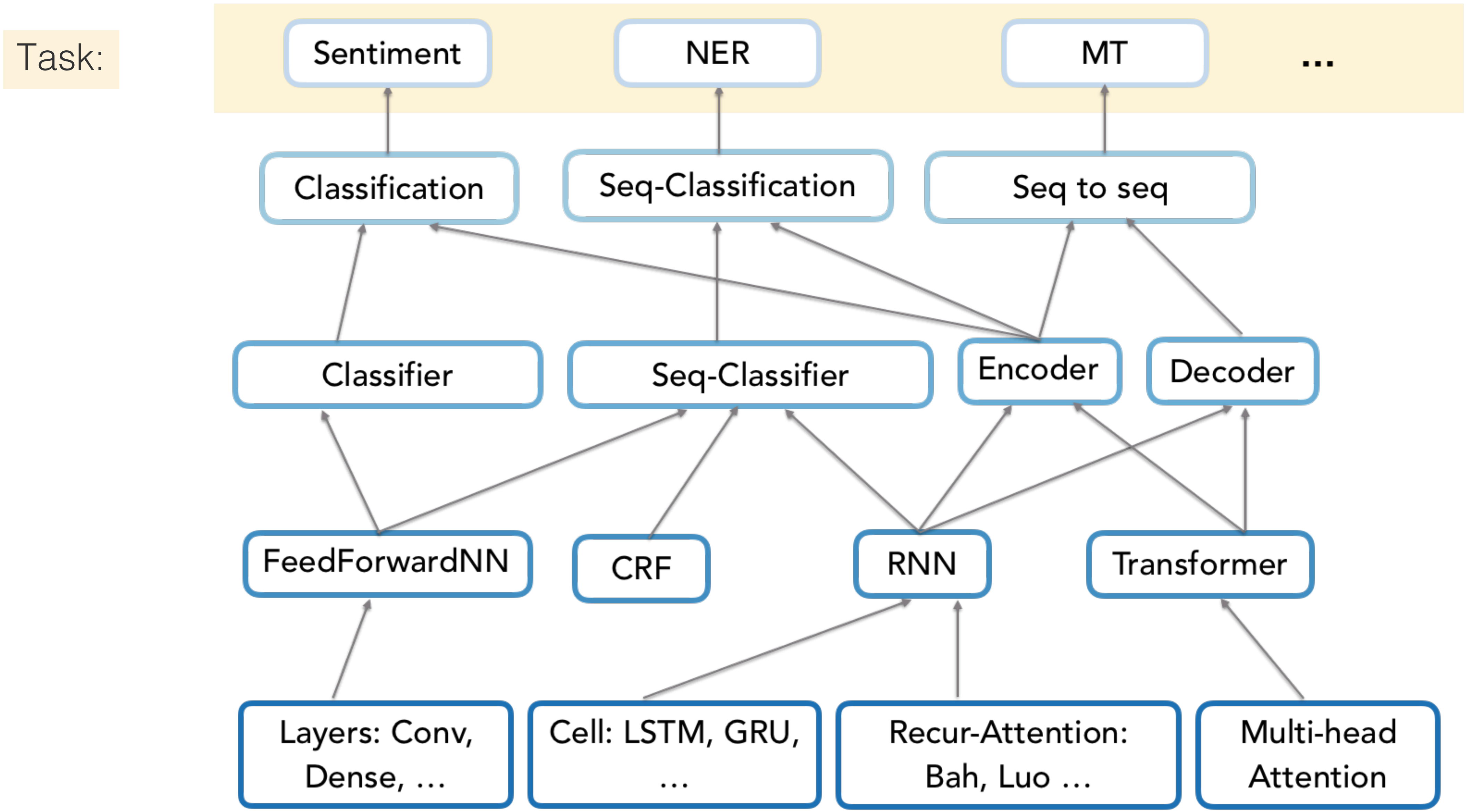
$\min_\theta \mathcal{L}(\theta, D) + \Omega(\theta)$

Task:



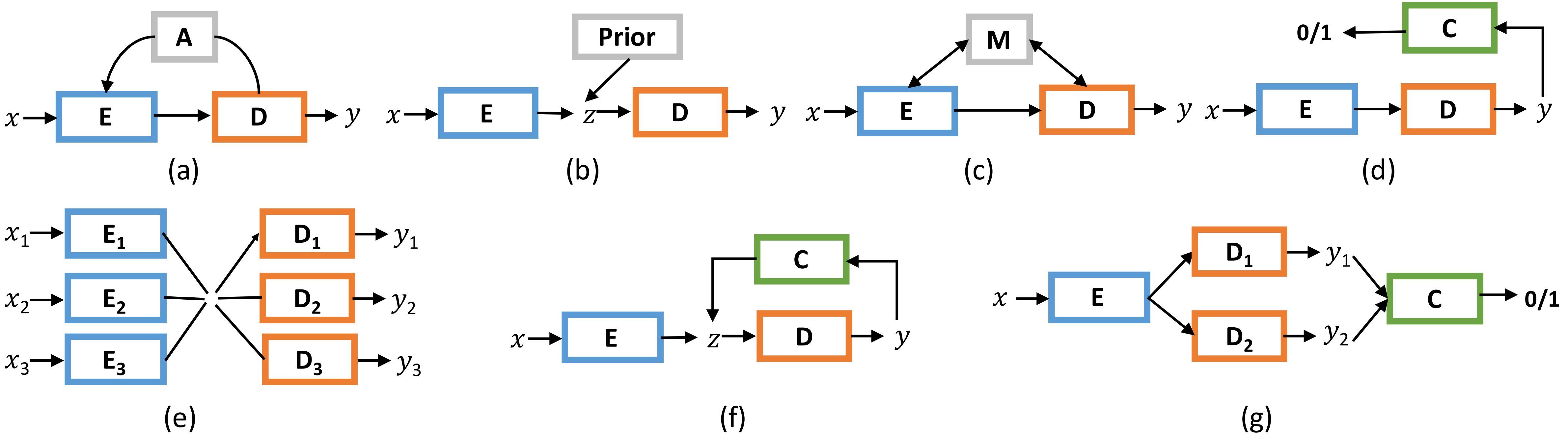
Architecture Graph

$\min_\theta \mathcal{L}(\theta, D) + \Omega(\theta)$



Complex Composite Architectures

$\min_{\theta}\ \mathcal{L}(\theta, D) + \Omega(\theta)$



E refers to encoder, D to decoder, C to Classifier, A to attention, Prior to prior distribution, and M to memory

ML Components

$\min_\theta \mathcal{L}(\theta, D) + \Omega(\theta)$

Constraint

Loss

Learning

Inference

Architecture

decoder

LSTM RNN

Attention RNN

Transformer

...

encoder

classifier

...

ML Components

$\min_\theta \mathcal{L}(\theta, D) + \Omega(\theta)$

Constraint

Loss

Learning

Inference

Architecture

decoder

LSTM RNN

Attention RNN

Transformer

...

encoder

classifier

...

Learning, Inference & Loss: Many Variations

$\min_\theta \mathcal{L}(\theta, D) + \Omega(\theta)$

active learning
weak/distant supervision
reward-augmented MLE
data re-weighting
data augmentation
maximum likelihood estimation

imitation learning
intrinsic reward
inverse RL
actor-critic
RL as inference
softmax policy gradient
policy gradient

adversarial domain adaptation
GANs
knowledge distillation
prediction minimization
energy-based GANs

Supervision forms: Data examples | Reward | Auxiliary model

Learning, Inference & Loss (1): Maximum Likelihood Estimation

$\min_\theta \mathcal{L}(\theta, D) + \Omega(\theta)$

Learning

Given data examples $\mathcal{D} = \{(x^*, y^*)\}$

$$\max_\theta \mathbb{E}_{(x^*,y^*)\sim\mathcal{D}}[\log p_\theta(y^* \mid x^*)]$$

$$\prod_{t=1}^{T} p_\theta(y_t^* \mid y_{1:t-1}^*, x^*)$$

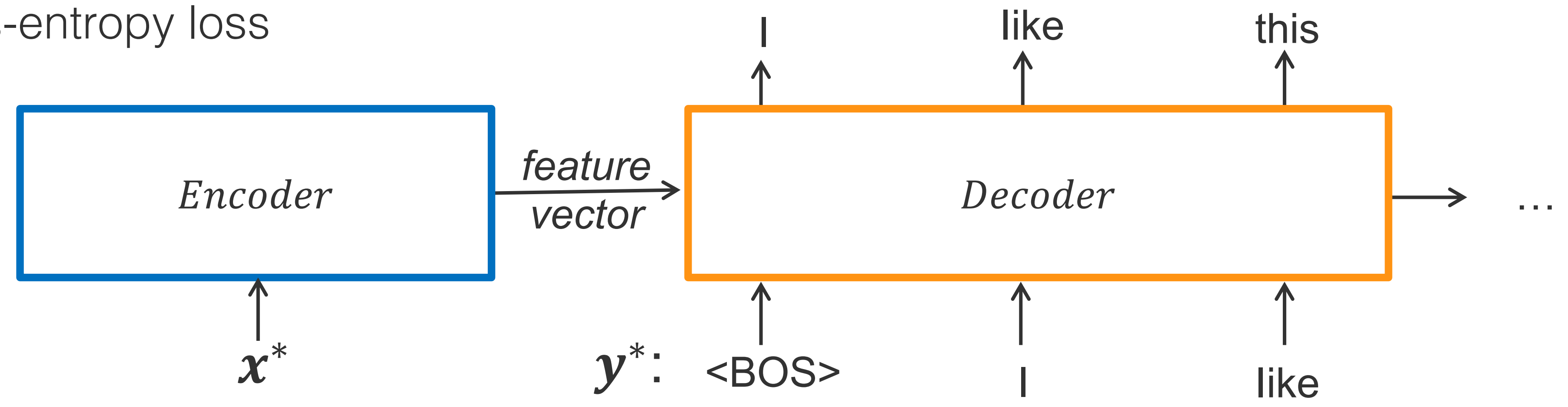
Inference

Teacher-forcing decoding:

For every step t, feeds in the previous ground-truth tokens $y_{1:t-1}^*$ to decode next step

Loss

Cross-entropy loss



Learning, Inference & Loss (2): Policy Gradient

$\min_\theta \mathcal{L}(\theta, D) + \Omega(\theta)$

Learning

Optimizes expected task reward $R(\hat{\boldsymbol{y}}, \boldsymbol{y}^*)$

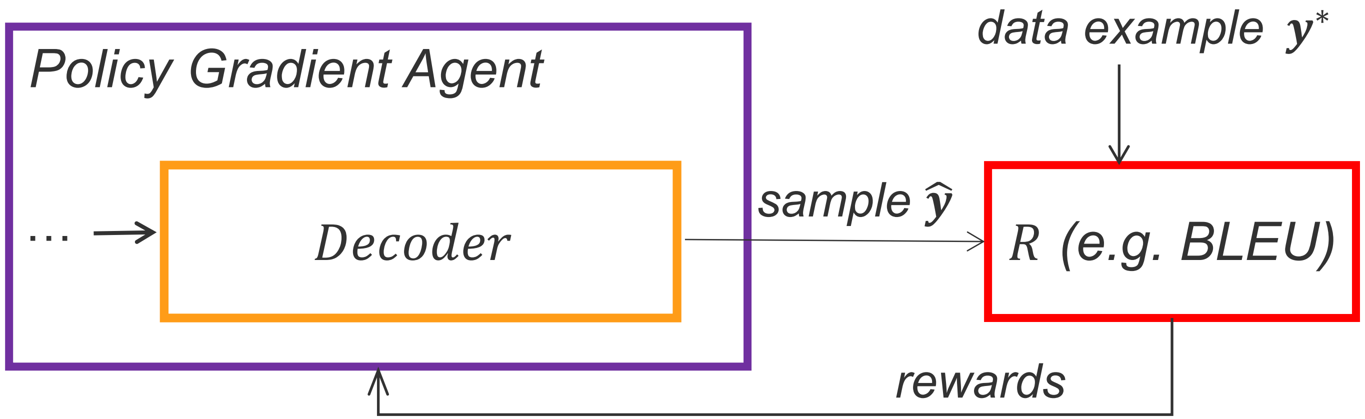
$$\max_\theta \mathbb{E}_{\hat{\boldsymbol{y}} \sim p_\theta(\boldsymbol{y} \mid \boldsymbol{x})} [\, R(\hat{\boldsymbol{y}}, \boldsymbol{y}^*) \,]$$

Loss

- Policy gradient loss
- Policy gradient loss w/ baseline
- …

Inference

- Greedy decoding
- Sampling decoding
- Beam search decoding
- Top-k / Top-p decoding
- …



Learning, Inference & Loss (3): Adversarial Learning

$\min_\theta \mathcal{L}(\theta, D) + \Omega(\theta)$

Learning

- A discriminator is trained to distinguish b/w *real* data examples and *fake* generated samples
- The model is trained to fool the discriminator

Loss

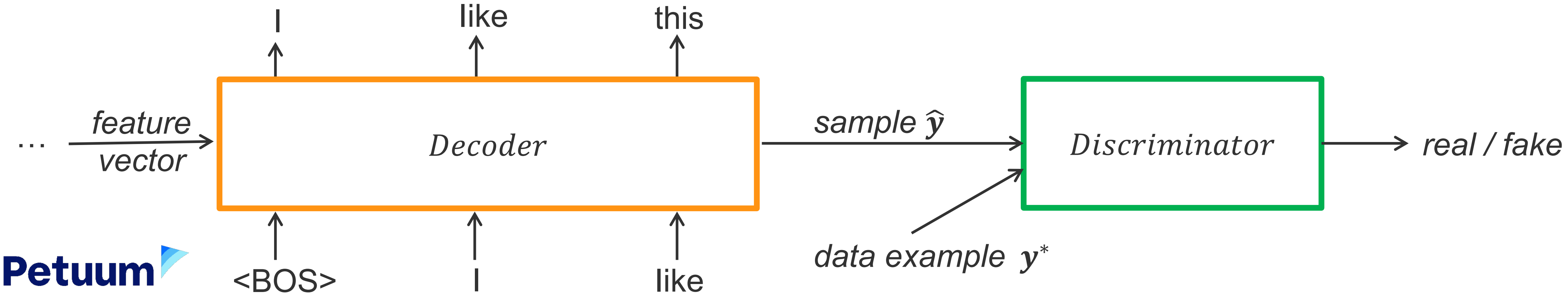
- Binary adversarial loss
- Feature-matching adversarial loss

Inference

Gumbel-softmax decoding:

Uses a differentiable approximation of sample $\hat{\mathbf{y}}$ for gradient backpropagation

$$\frac{\partial \mathcal{L}(\hat{\mathbf{y}})}{\partial \theta} = \frac{\partial \mathcal{L}(\hat{\mathbf{y}})}{\partial \hat{\mathbf{y}}} \frac{\partial \hat{\mathbf{y}}}{\partial \theta}$$



Petuum

ML Components

$\min_\theta \mathcal{L}(\theta, D) + \Omega(\theta)$

Constraint	Loss	Learning	Inference	Architecture
	Cross-entropy	MLE	Teacher-forcing	decoder
	Binary Adv loss	Adversarial	Gumbel-softmax	LSTM RNN
	Matching Adv loss	Reinforcement	Sample	Attention RNN
	PG loss	…	Greedy	Transformer
	PG loss + baseline		Beam-search	…
	…		Top-k sample	encoder
			…	classifier
				…

ML Components

$\min_\theta \mathcal{L}(\theta, D) + \Omega(\theta)$

Constraint	Loss	Learning	Inference	Architecture
	Cross-entropy	MLE	Teacher-forcing	decoder
	Binary Adv loss	Adversarial	Gumbel-softmax	LSTM RNN
	Matching Adv loss	Reinforcement	Sample	Attention RNN
	PG loss	…	Greedy	Transformer
	PG loss + baseline		Beam-search	…
	…		Top-k sample	encoder
			…	classifier
				…

Constraint (1): Conventional Constraints

$\min_\theta \mathcal{L}(\theta, D) + \Omega(\theta)$

Many choices for get different statistical properties:

- Normality, Sparsity, KL, sum, …

Constraint (2): Structured Knowledge

$\min_\theta \mathcal{L}(\theta, D) + \Omega(\theta)$

Structured knowledge as constraints

Sentiment classification:

- ``Food was good, but the service was very disappointing.''

Logic rule:

- Sentence $\boldsymbol{x}$ with structure A-but-B => sentiment of B dominates

Constraint function: $f(\boldsymbol{x} = \text{sentence}, \boldsymbol{y} = \text{sentiment}) = \text{truth value}$

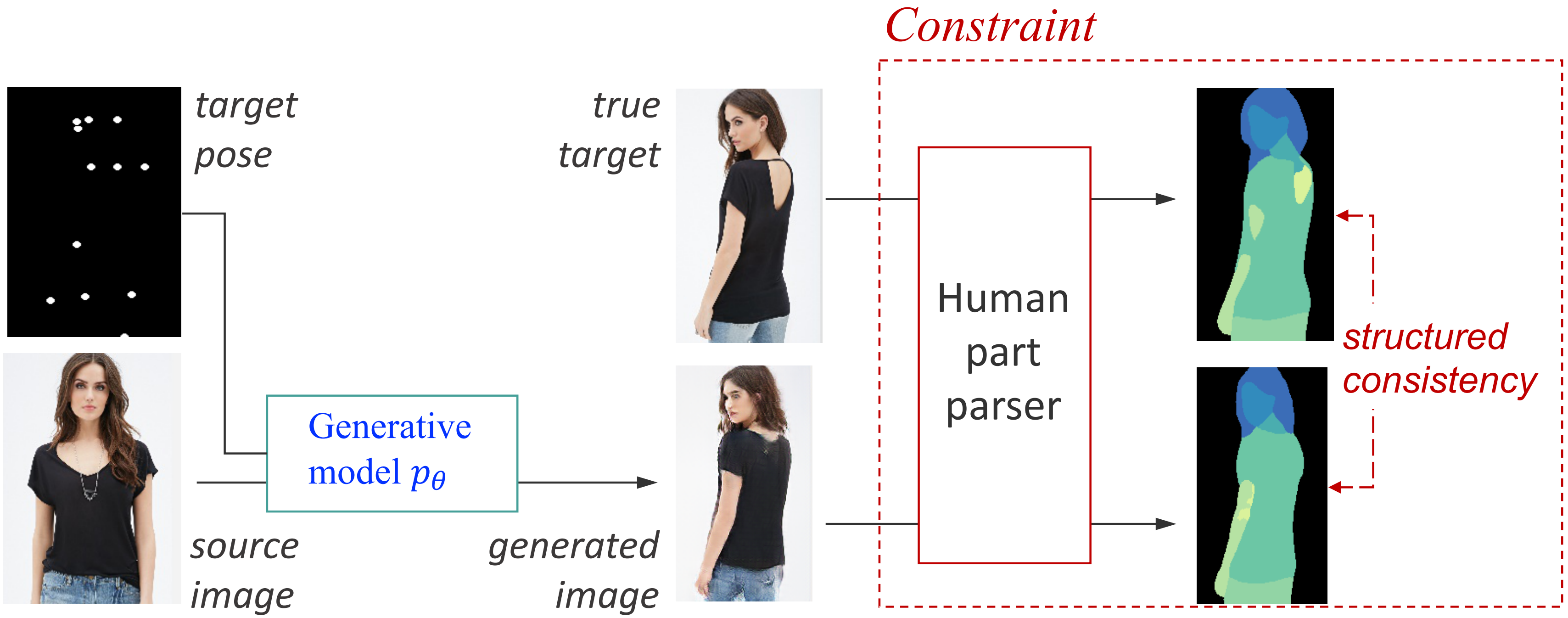
Constraint (2): Structured Knowledge

$\min_\theta \mathcal{L}(\theta, D) + \Omega(\theta)$

Structured knowledge as constraints

Human Image Generation

Constraint function: $f(\boldsymbol{y} = \text{generated}, \boldsymbol{o} = \text{ground truth}) = \text{match score}$



Constraint (2): Structured Knowledge

$\min_\theta \mathcal{L}(\theta, D) + \Omega(\theta)$

- Constraint function: $f(\boldsymbol{x}, \boldsymbol{y}) \in \mathbb{R}$
  - Higher f value, better $(\boldsymbol{x}, \boldsymbol{y})$ in the light of the knowledge
- Model: $p_\theta(\boldsymbol{y}|\boldsymbol{x})$
- Posterior Regularization [Hu et al., 2018, 2016; Zhu et al., 2014; Ganchev et al. 2010]

$$\min_{\theta,q} \mathcal{L}(\boldsymbol{\theta}, q) = \mathcal{L}(\boldsymbol{\theta}) + \mathrm{KL}\big(q(\boldsymbol{y}|\boldsymbol{x}) \,||\, p_\theta(\boldsymbol{y}|\boldsymbol{x})\big) + \xi$$

$$s.t. \quad \mathbb{E}_{q(\boldsymbol{y}|\boldsymbol{x})}[\, f(\boldsymbol{x}, \boldsymbol{y}) \,] \geq 1 - \xi$$

Regular loss
(e.g., cross-entropy loss)

Constraint

- Related: constraint-driven learning [Chang et al.,2007], generalized expectation [Mann & MaCallum, 2007], learning from measurements [Liang et al., 2009]

Constraint (2): Structured Knowledge

$\min_\theta \mathcal{L}(\theta, D) + \Omega(\theta)$

$$\mathcal{L}(\boldsymbol{\theta}, q) = \mathcal{L}(\boldsymbol{\theta}) + \mathrm{KL}\big(q(\boldsymbol{y}|\boldsymbol{x}) \,||\, p_\theta(\boldsymbol{y}|\boldsymbol{x})\big) + \xi$$

$$s.t.\quad \mathbb{E}_{q(\boldsymbol{y}|\boldsymbol{x})}[\, f(\boldsymbol{x}, \boldsymbol{y}) \,] \geq 1 - \xi$$

- $\min_q \mathcal{L}(\theta, q)$ $\Longrightarrow$ $q^*(\boldsymbol{y}|\boldsymbol{x}) \propto p_\theta(\boldsymbol{y}|\boldsymbol{x}) \exp\left\{ f(\boldsymbol{x}, \boldsymbol{y}) \right\}$

Combines the model and the knowledge
— teacher model

- $\min_\theta \mathcal{L}(\theta, q^*)$ $\Longrightarrow$ $\max_\theta \mathbb{E}_{q^*(\boldsymbol{y}|\boldsymbol{x})}\left[\log p_\theta(\boldsymbol{y}|\boldsymbol{x}) \right]$

The model imitates the teacher model predictions
— student model

ML Components

$\min_\theta \mathcal{L}(\theta, D) + \Omega(\theta)$

Constraint	Loss	Learning	Inference	Architecture
				decoder
L1 / L2	Cross-entropy	MLE	Teacher-forcing	LSTM RNN
Logical	Binary Adv loss	Policy Gradient	Gumbel-softmax	Attention RNN
Structured	Matching Adv loss	Adversarial	Sample	Transformer
…	PG loss		Greedy	…
	PG loss + baseline		Beam-search	encoder
	…		Top-k sample	classifier
			…	…

ML Components

$$\min_{\theta} \mathcal{L}(\theta, D) + \Omega(\theta)$$

Constraint	Loss	Learning	Inference	Architecture
L1 / L2	Cross-entropy	MLE	Teacher-forcing	decoder
Logical	Binary Adv loss	Policy Gradient	Gumbel-softmax	LSTM RNN
Structured	Matching Adv loss	Adversarial	Sample	Attention RNN
…	PG loss	Adv + RL	Greedy	Transformer
	PG loss + baseline	Reward-aug.	Beam-search	…
	…	…	Top-k sample	encoder
			…	classifier
				…

ML Components

$\min_\theta \mathcal{L}(\theta, D) + \Omega(\theta)$

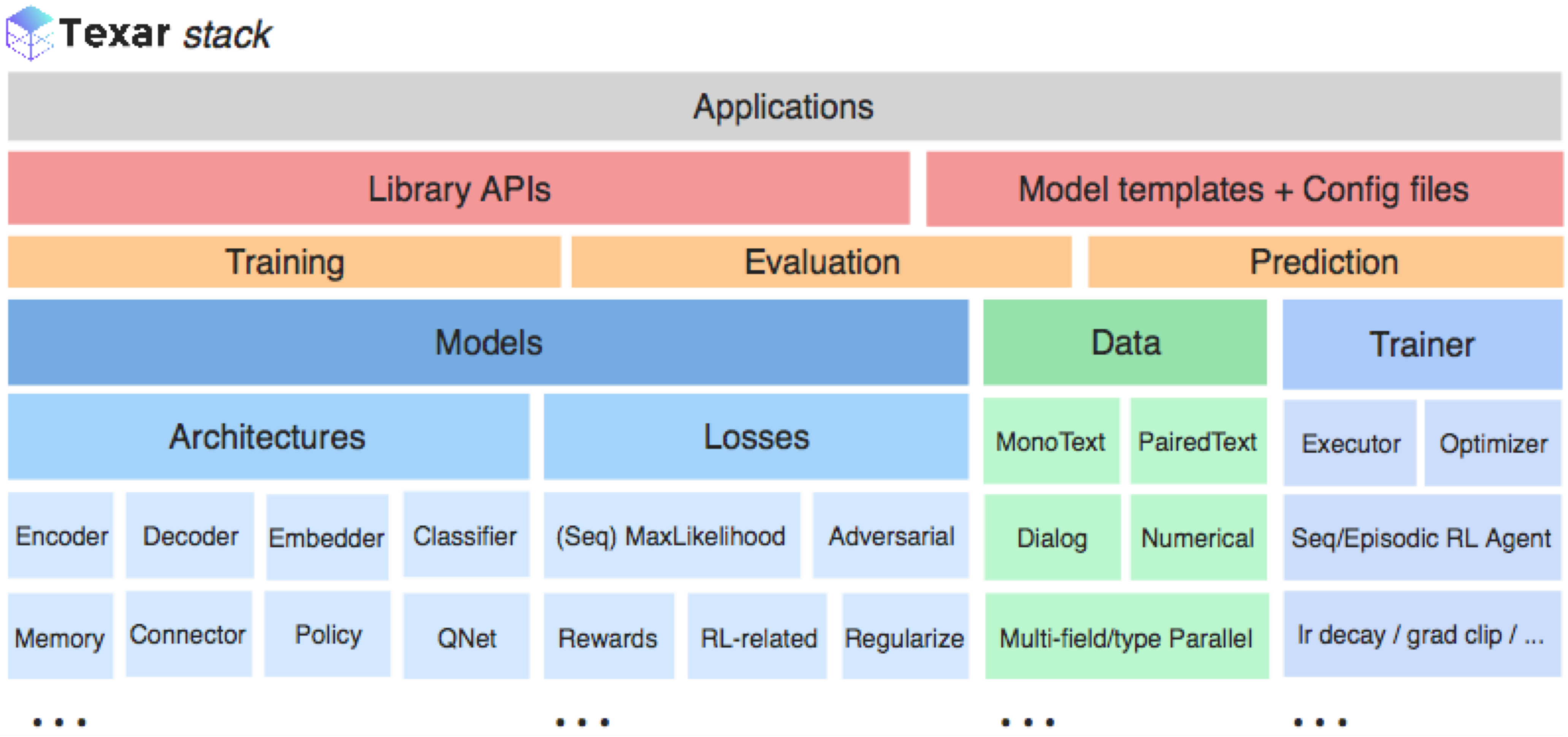
Constraint	Loss	Learning	Inference	Architecture
L1 / L2	Cross-entropy	MLE	Teacher-forcing	decoder
Logical	Binary Adv loss	Policy Gradient	Gumbel-softmax	LSTM RNN
Structured	Matching Adv loss	Adversarial	Sample	Attention RNN
…	PG loss	Adv + RL	Greedy	Transformer
	PG loss + baseline	Reward-aug.	Beam-search	…
	…	…	Top-k sample	encoder
			…	classifier
				…

Operationalize Composable ML with Texar

called as subroutines | *uniform interfaces*

Constraint	Loss	Learning	Inference	Architecture
L1 / L2	Cross-entropy	MLE	Teacher-forcing	decoder
Logical	Binary Adv loss	Policy Gradient	Gumbel-softmax	LSTM RNN
Structured	Matching Adv loss	Adversarial	Sample	Attention RNN
…	PG loss	Adv + RL	Greedy	Transformer
	PG loss + baseline	Reward-aug.	Beam-search	…
	…	…	Top-k sample	encoder
			…	classifier
				…

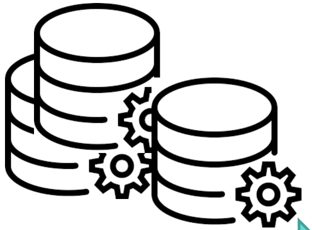
Operationalize Composable ML with Texar



Implementation with Texar

Running Example: Machine Translation

raw data



cleaning
tokenizing
vocabulary
truncation
…

source.dat

I like this movie.

Lovely and poignant

Insanely hilarious!

…

target.dat

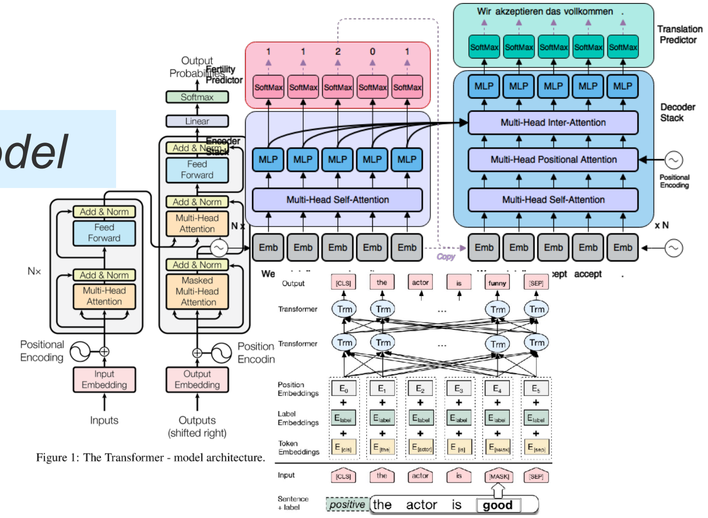
Ich mag diesen film.

Schön und ergreifend

Wahnsinnig witzig!

…

clean data



model

training

Maximum likelihood training

Reinforcement learning

Adversarial

Finetuning

evaluation
post-processing

…

Running Example: Machine Translation

Implementation with Texar

Running Example: Machine Translation

Data

```python
# Read data
dataset = PairedTextData(data_hparams)
batch = DataIterator(dataset).get_next()
```

YAML config files

```yaml
data_hparams:
  batch_size: 64
  num_epochs: 10
  shuffle: True
  source_dataset:
    files: ‘source.txt’
    vocab_file: ‘vocab.txt’
    max_seq_length: 100
    bos_token: ‘<BOS>’
    eos_token: ’<EOS>’
  target_dataset:
    …
…
```

Implementation with Texar

Running Example: Machine Translation

YAML config files

Data

```
# Read data
dataset = PairedTextData(data_hparams)
batch = DataIterator(dataset).get_next()
```

Architecture & Inference

Running Example: Machine Translation

Implementation with Texar

Data

Architecture & Inference

```
# Read data
dataset = PairedTextData(data_hparams)
batch = DataIterator(dataset).get_next()
# Encode
embedder = WordEmbedder(dataset.vocab.size, hparams=embedder_hparams)
```

YAML config files

```
1. embedder_hparams:
2.   embedding_dim: 256
3.   dropout_rate: 0.9
4.   regularization: 'L1L2'
5. …
```

Implementation with Texar

Running Example: Machine Translation

YAML config files

Data

```python
# Read data
dataset = PairedTextData(data_hparams)
batch = DataIterator(dataset).get_next()
```

Architecture & Inference

```python
# Encode
embedder = WordEmbedder(dataset.vocab.size, hparams=embedder_hparams)
encoder = TransformerEncoder(hparams=encoder_hparams)
```

```yaml
encoder_hparams:
  num_blocks: 16
  num_heads: 8
  hidden_dim: 256
  output_dim: 128
  dropout_rate: 0.8
…
```

Implementation with Texar

Running Example: Machine Translation

Data

```
# Read data
dataset = PairedTextData(data_hparams)
batch = DataIterator(dataset).get_next()
```

Architecture & Inference

```
# Encode
embedder = WordEmbedder(dataset.vocab.size, hparams=embedder_hparams)
encoder = TransformerEncoder(hparams=encoder_hparams)
enc_outputs = encoder(embedder(batch['source_text_ids']),
                      batch['source_length'])
```

Implementation with Texar

Running Example: Machine Translation

Data

Architecture & Inference

```
# Read data
dataset = PairedTextData(data_hparams)
batch = DataIterator(dataset).get_next()
# Encode
embedder = WordEmbedder(dataset.vocab.size, hparams=embedder_hparams)
encoder = TransformerEncoder(hparams=encoder_hparams)
enc_outputs = encoder(embedder(batch['source_text_ids']),
                      batch['source_length'])
# Build decoder
decoder = AttentionRNNDecoder(memory=enc_outputs,
                              hparams=decoder_hparams)
# Maximum Likelihood Estimation
## Teacher-forcing decoding
outputs, length, _ = decoder(decoding_strategy='teacher-forcing',
                             inputs=embedder(batch['target_text_ids']),
                             seq_length=batch['target_length']-1)
```

Implementation with Texar

Running Example: Machine Translation

Data

Architecture & Inference

Learning Loss

```
# Read data
dataset = PairedTextData(data_hparams)
batch = DataIterator(dataset).get_next()
# Encode
embedder = WordEmbedder(dataset.vocab.size, hparams=embedder_hparams)
encoder = TransformerEncoder(hparams=encoder_hparams)
enc_outputs = encoder(embedder(batch['source_text_ids']),
                      batch['source_length'])
# Build decoder
decoder = AttentionRNNDecoder(memory=enc_outputs,
                              hparams=decoder_hparams)
# Maximum Likelihood Estimation
## Teacher-forcing decoding
outputs, length, _ = decoder(decoding_strategy=’teacher-forcing’,
                             inputs=embedder(batch['target_text_ids']),
                             seq_length=batch['target_length']-1)
## Cross-entropy loss
loss = sequence_sparse_softmax_cross_entropy(
    labels=batch['target_text_ids'][:,1:], logits=outputs.logits, seq_length=length)

```

Implementation with Texar

Running Example: Machine Translation

Data

Architecture & Inference

Learning Loss

```
# Read data
dataset = PairedTextData(data_hparams)
batch = DataIterator(dataset).get_next()
# Encode
embedder = WordEmbedder(dataset.vocab.size, hparams=embedder_hparams)
encoder = TransformerEncoder(hparams=encoder_hparams)
enc_outputs = encoder(embedder(batch['source_text_ids']),
                      batch['source_length'])
# Build decoder
decoder = AttentionRNNDecoder(memory=enc_outputs,
                              hparams=decoder_hparams)
# Maximum Likelihood Estimation
## Teacher-forcing decoding
outputs, length, _ = decoder(decoding_strategy=’teacher-forcing’,
                             inputs=embedder(batch['target_text_ids']),
                             seq_length=batch['target_length']-1)
## Cross-entropy loss
loss = sequence_sparse_softmax_cross_entropy(
    labels=batch['target_text_ids'][:,1:], logits=outputs.logits, seq_length=length)
```

Maximum likelihood Estimation

Implementation with Texar

Switching between Learning Algorithms

Data

Architecture & Inference

Learning Loss

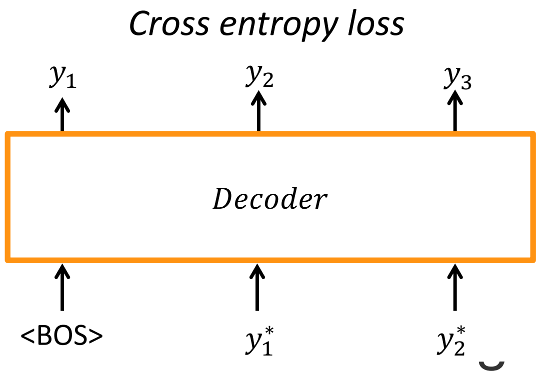
```
# Read data
dataset = PairedTextData(data_hparams)
batch = DataIterator(dataset).get_next()
# Encode
embedder = WordEmbedder(dataset.vocab.size, hparams=embedder_hparams)
encoder = TransformerEncoder(hparams=encoder_hparams)
enc_outputs = encoder(embedder(batch['source_text_ids']),
                      batch['source_length'])
# Build decoder
decoder = AttentionRNNDecoder(memory=enc_outputs,
                              hparams=decoder_hparams)
# Maximum Likelihood Estimation
## Teacher-forcing decoding
outputs, length, _ = decoder(decoding_strategy='teacher-forcing',
                             inputs=embedder(batch['target_text_ids']),
                             seq_length=batch['target_length']-1)
## Cross-entropy loss
loss = sequence_sparse_softmax_cross_entropy(
    labels=batch['target_text_ids'][:,1:], logits=outputs.logits, seq_length=length)
```

Keep unchanged

Maximum likelihood Estimation

Switching from MLE to Reinforcement Learning

- Maximum likelihood

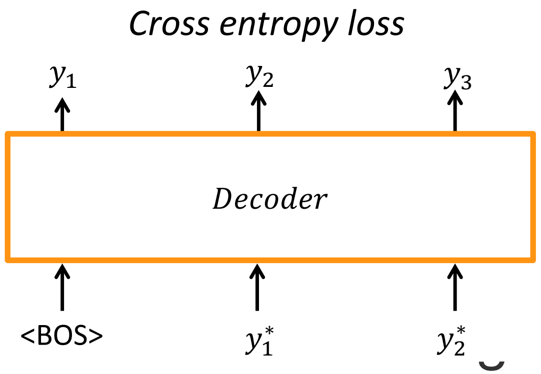


- R

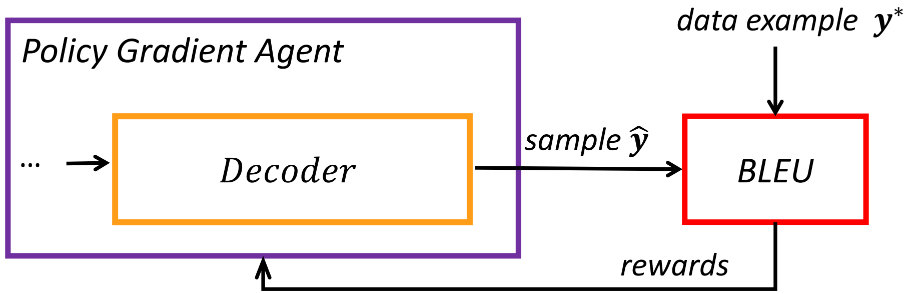
```
# Teacher-forcing decoding
outputs, length, _ = decoder(decoding_strategy='teacher-forcing',
                             inputs=embedder(batch['target_text_ids']),
                             seq_length=batch['target_length']-1)
# Cross-entropy loss
loss = sequence_sparse_softmax_cross_entropy(
    labels=batch['target_text_ids'][:,1:], logits=outputs.logits, seq_length=length)
```

Switching from MLE to Reinforcement Learning

- Maximum likelihood



- R

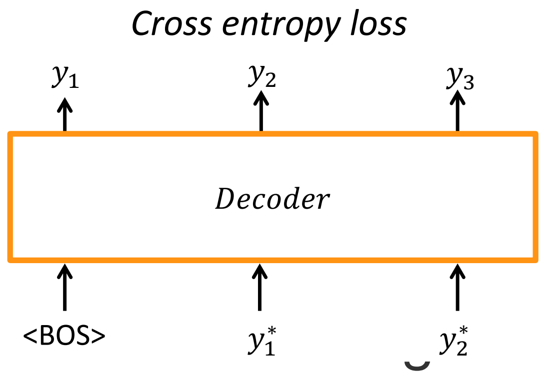



```
# Teacher-forcing decoding
outputs, length, _ = decoder(decoding_strategy=‘teacher-forcing’,
                             inputs=embedder(batch['target_text_ids']),
                             seq_length=batch['target_length']-1)
# Cross-entropy loss
loss = sequence_sparse_softmax_cross_entropy(
   labels=batch['target_text_ids'][:,1:], logits=outputs.logits, seq_length=length)
```

```
# Random sample decoding
outputs, length, _ = decoder(decoding_strategy=‘random_sample’,
                             start_tokens=[BOS]*batch_size, end_token=EOS,
                             embedding=embedder)
# Policy gradient agent for learning
agent = SeqPGAgent(
   samples=outputs.sample_id, logits=outputs.logits, seq_length=length)

for _ in range(STEPS):
   samples = agent.get_samples()
   rewards = BLEU(batch[‘target_text_ids’], samples)  # Reward
   agent.observe(rewards)
```

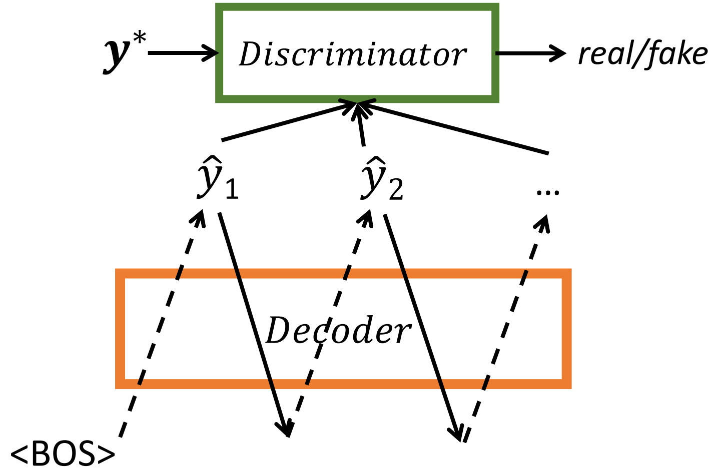
Switching from MLE to Adversarial Learning

- Maximum likelihood




```
# Teacher-forcing decoding
outputs, length, _ = decoder(decoding_strategy='teacher-forcing',
                             inputs=embedder(batch['target_text_ids']),
                             seq_length=batch['target_length']-1)
# Cross-entropy loss
loss = sequence_sparse_softmax_cross_entropy(
    labels=batch['target_text_ids'][:,1:], logits=outputs.logits, seq_length=length)
```

- A



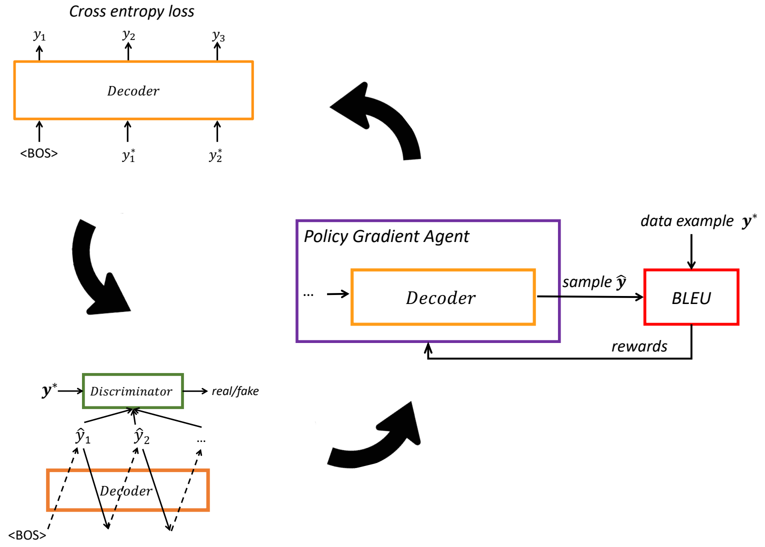

```
# Gumbel-softmax decoding
outputs, _, _ = decoder(decoding_strategy= ='gumbel-softmax',
                        start_tokens=[BOS]*batch_size, end_token=EOS,
                        embedding=embedder)

discriminator = Conv1DClassifier(hparams=conv_hparams)
# Binary adversarial loss
G_loss, D_loss = binary_adversarial_losses(
    embedder(batch['target_text_ids'][:, 1:]),
    embedder(soft_ids=softmax(outputs.logits)),
    discriminator)
```

Summary of MT in Texar

- Highly modularized programming
  - Data, architecture, loss, inference, learning, …
  - Intuitive conceptual-level APIs

```
# Read data
dataset = PairedTextData(data_hparams)
batch = DataIterator(dataset).get_next()
# Encode
embedder = WordEmbedder(dataset.vocab.size, hparams=embedder_hparams)
encoder = TransformerEncoder(hparams=encoder_hparams)
enc_outputs = encoder(embedder(batch['source_text_ids']),
                      batch['source_length'])
# Build decoder
decoder = AttentionRNNDecoder(memory=enc_outputs,
                              hparams=decoder_hparams)
# Maximum Likelihood Estimation
## Teacher-forcing decoding
outputs, length, _ = decoder(decoding_strategy='teacher-forcing',
                             inputs=embedder(batch['target_text_ids']),
                             seq_length=batch['target_length']-1)
## Cross-entropy loss
loss = sequence_sparse_softmax_cross_entropy(
  labels=batch['target_text_ids'][:,1:], logits=outputs.logits, seq_length=length)
```

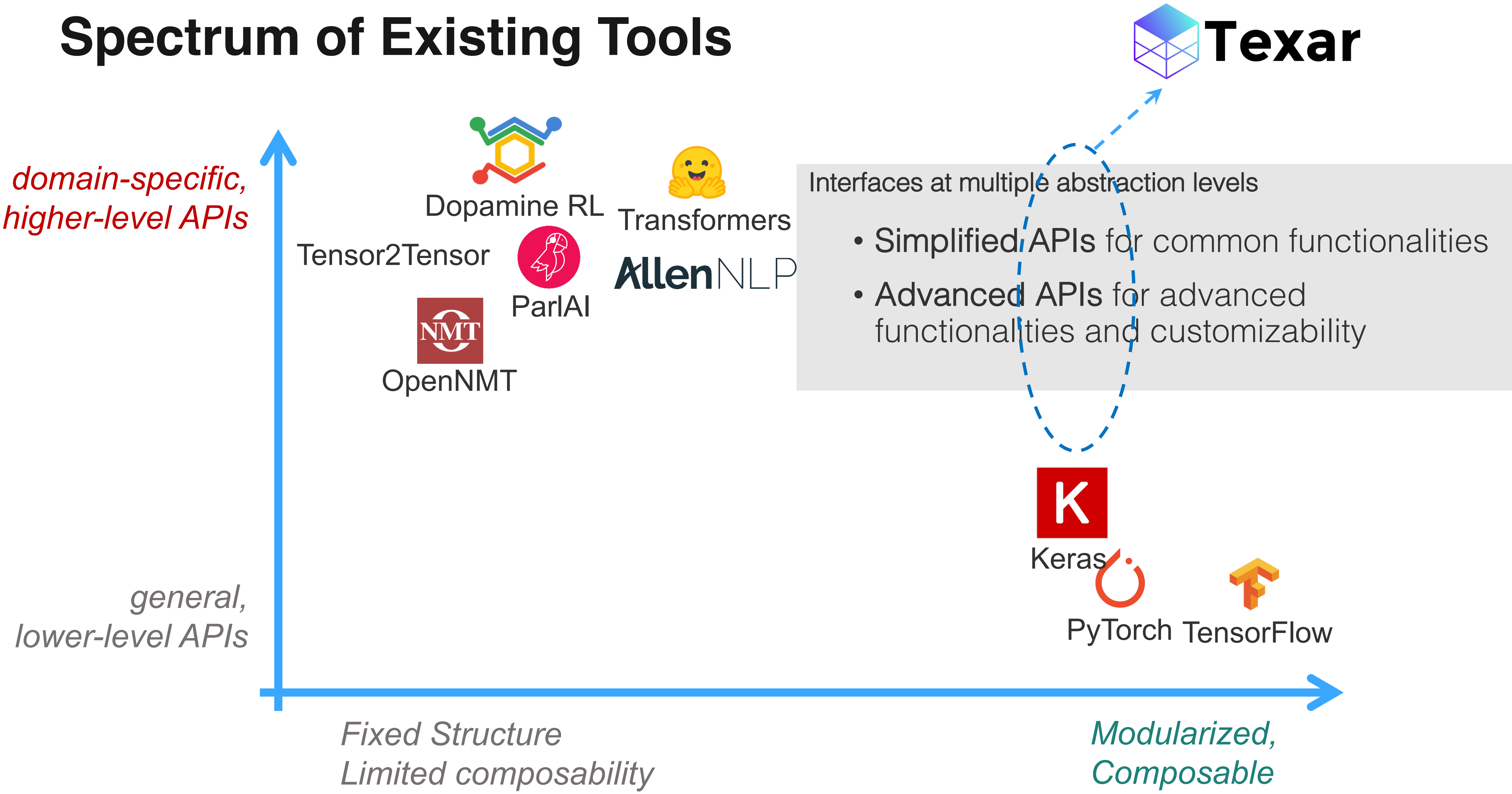
- Easy switch between learning algorithms
  - Plug in & out modules
  - No changes to irrelevant parts



Support of TensorFlow and PyTorch

- Texar is built upon TF and PyTorch
  - Texar-TF & Texar-PyTorch: mostly the same interfaces!
  - Higher-level intuitive APIs without loss of flexibility
  - Lots of ML components ready to use
- Combine the best design of TF and PyTorch
  - TF:
    - Easy and efficient data processing APIs
    - Excellent factorization of ML modules
    - Turnkey model training processor
  - PyTorch:
    - Intuitive programming interfaces
    - Transparent variable scope and sharing to users

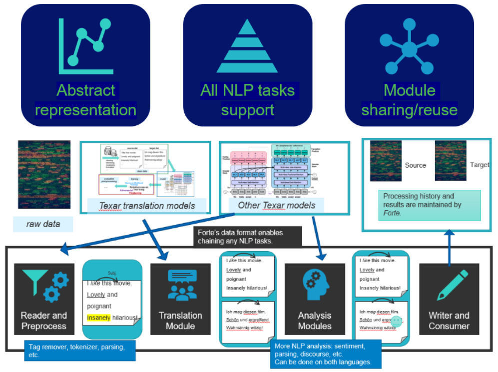
Spectrum of Existing Tools



Applications of Texar

Many products built on Texar

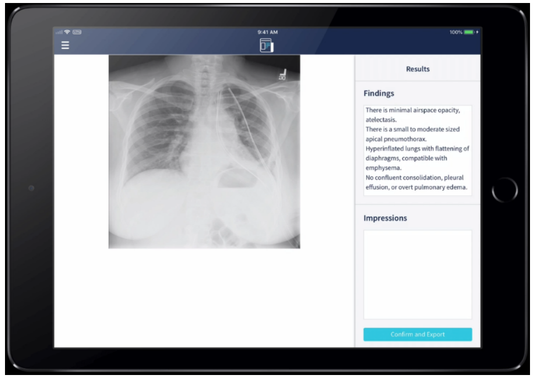
FORTE



- FORTE – templates for larger complex NLP applications

- Chest X-Ray report writer
- Medical Registry report writer
- ICD coding system

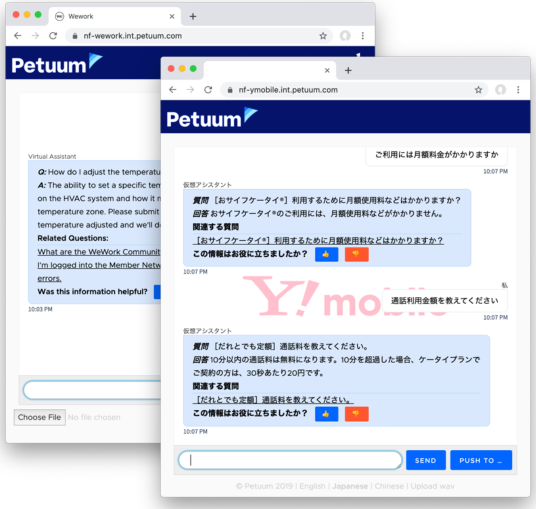
Chest X-Ray Report Writer



- Financial knowledge base builder
- Financial summary/report writer

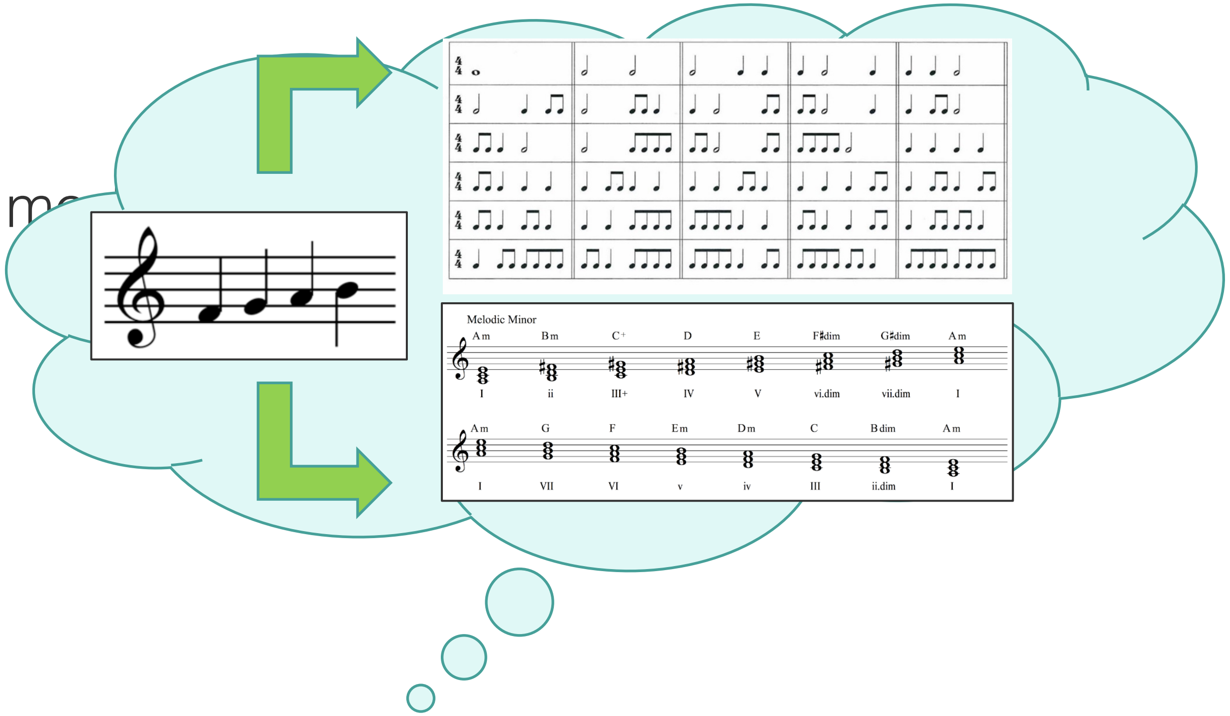
- Multi-Lingual Cognitive Chat Bots
  - For Call Center Support
  - For Retail In-Store Assistance

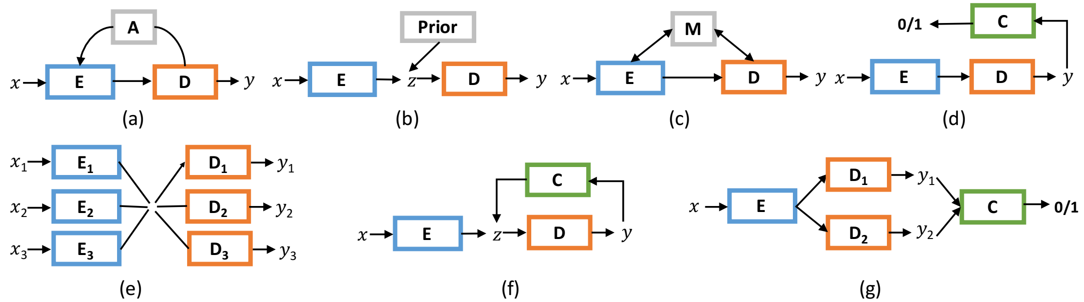
Multi-Lingual Cognitive Chat Bots



Composable ML – Take-Home Message

- Composable ML
  - Basic “musical notes” for complex ML m
  - (or just think of it as Lego for ML)
- Texar

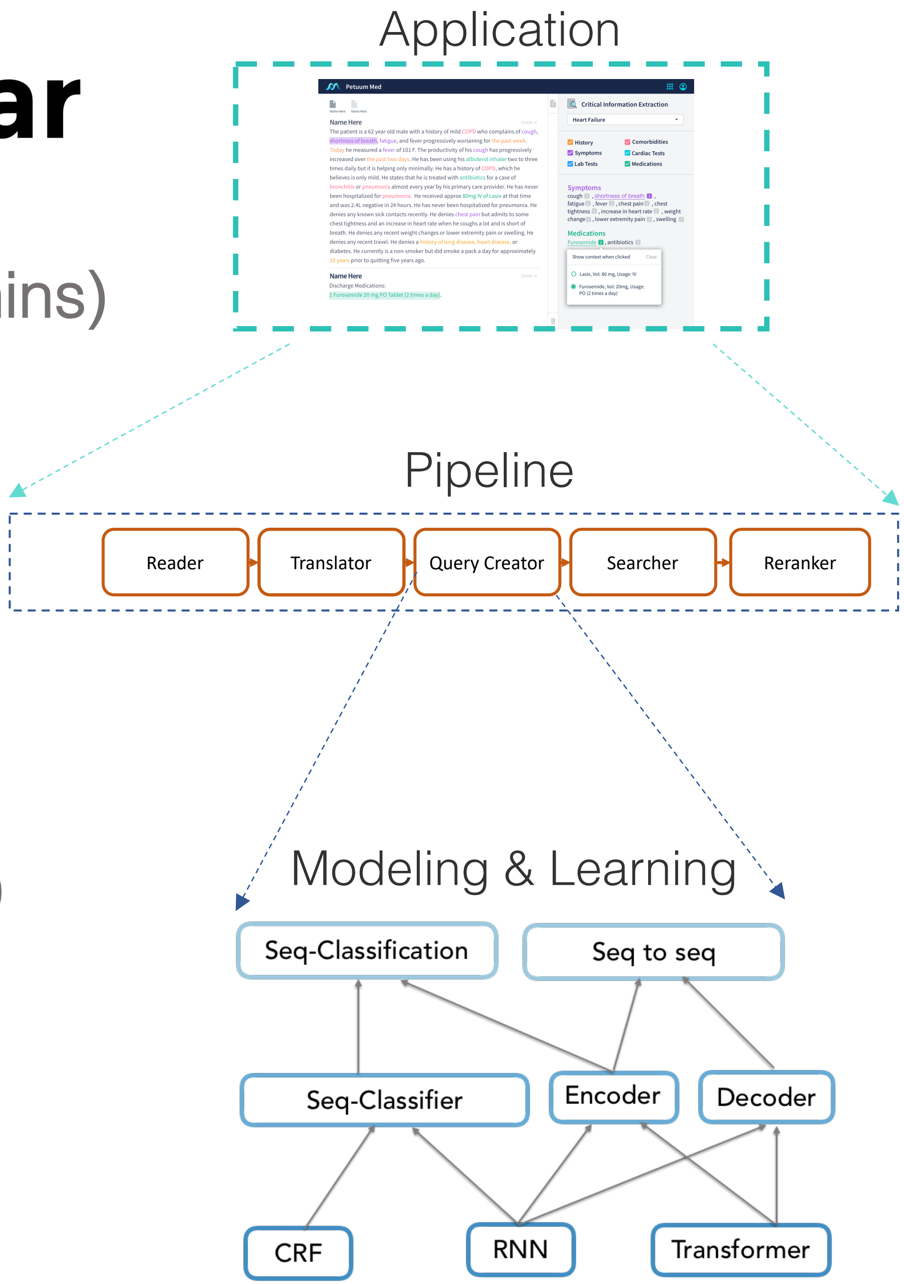




Agenda

Forte Texar

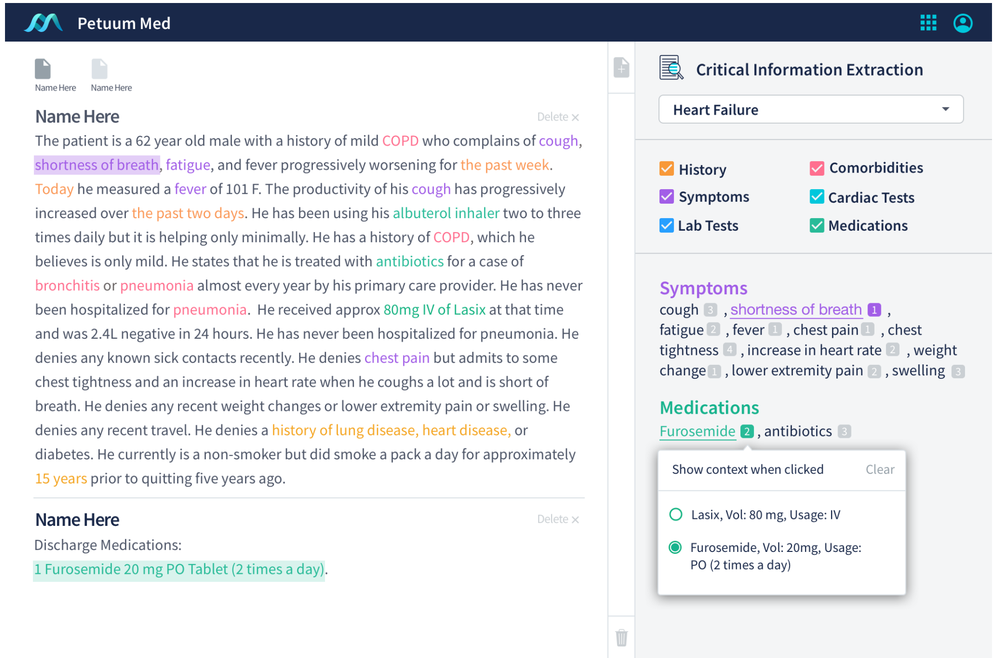
- Natural Language Processing Overview (10mins)
- Modularizing NLP Pipeline (35mins)
  - Complexity of NLP pipeline
  - A standardized view of NLP pipeline
  - A standardized implementation of NLP pipeline
- Short break & QA (5mins)
- Modularizing NLP Model & Learning (30mins)
  - Composable ML
- QA (10mins)



Petuum
Carnegie Mellon University
School of Computer Science

What’s Next

- Data Manipulation
    - Data augmentation
    - Data visualization
    - Multi-modal data manipulation
- Task Inter-operation
    - Joint learning
    - Joint inference
- Automate ML workflow
    - Automate model composing, learning



PETUUM

Thank you

Carnegie Mellon University
School of Computer Science