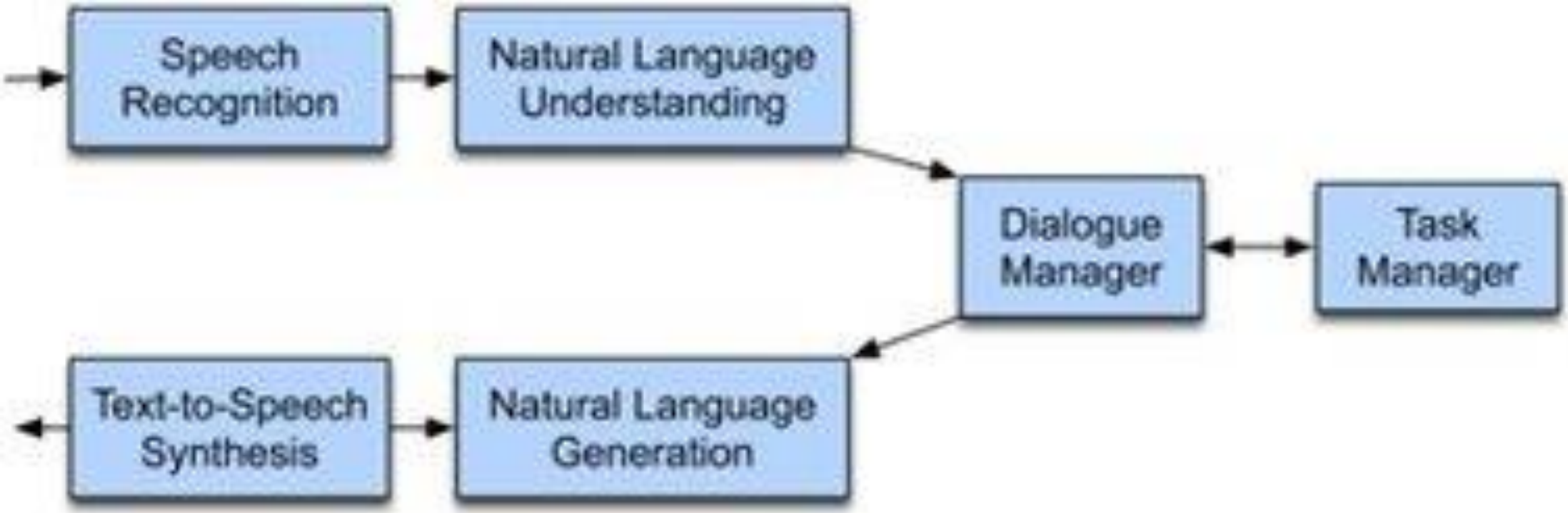
Knowledge Graphs, Search, and Question Answering Systems

EE596

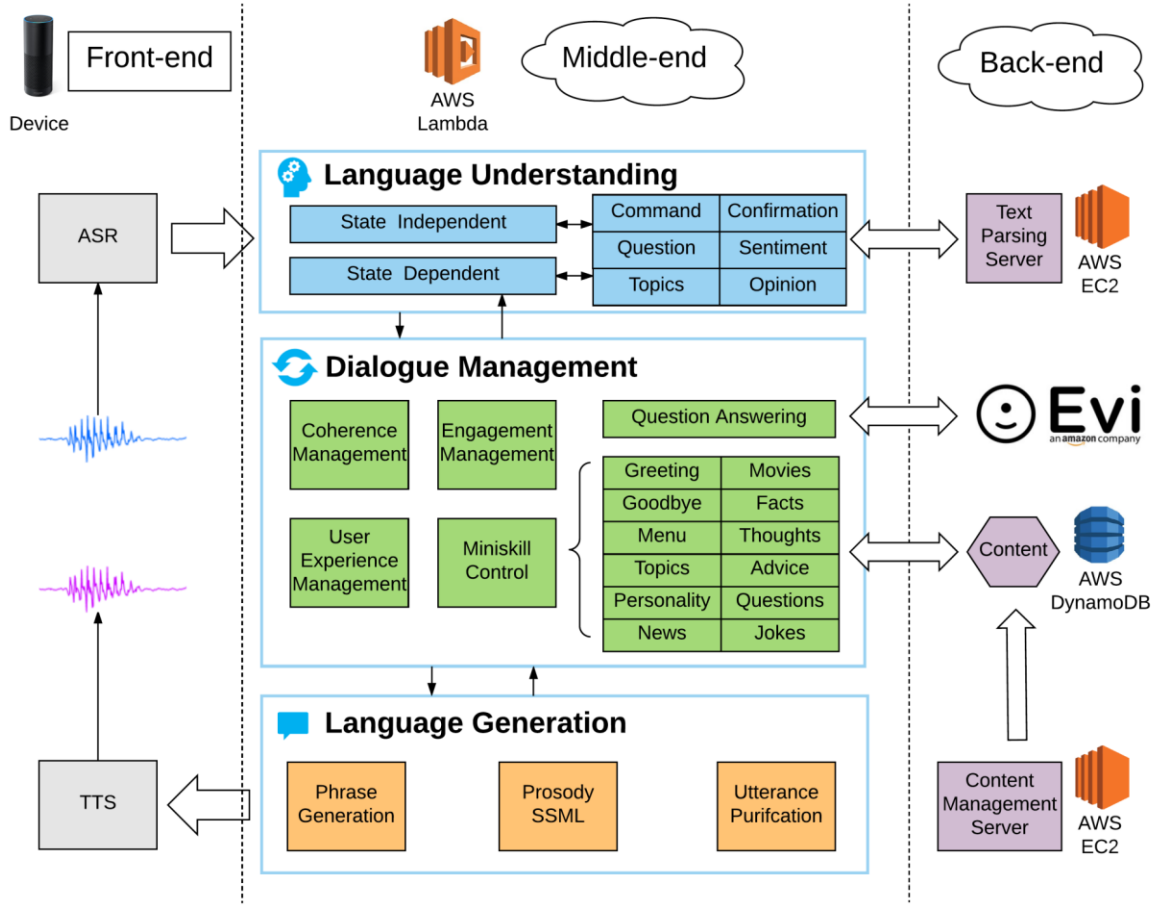
Conversational AI

5/8/2018

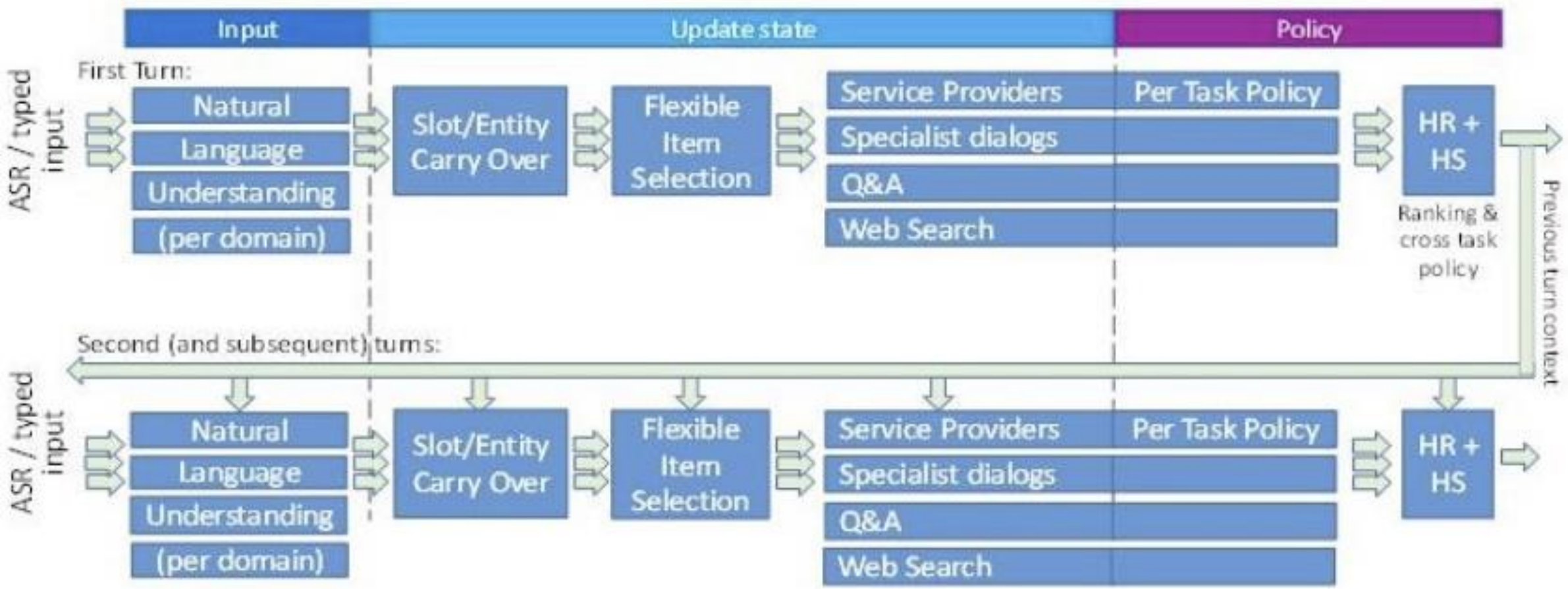
Typical Dialog System Architecture



Recall: SoundingBoard Architecture



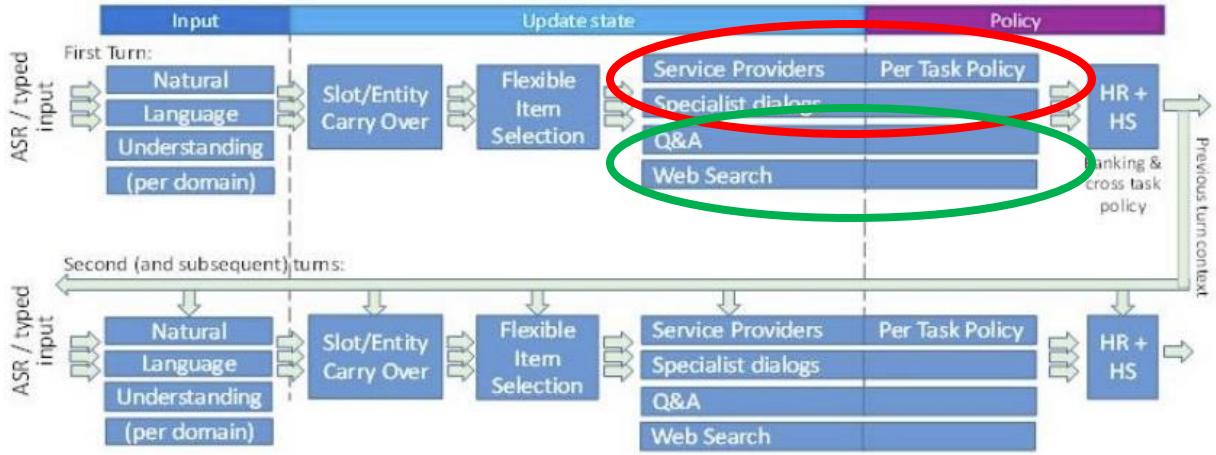
“Commercial” Dialog System Architecture



(Sarikaya *et* al., 2016)

“Commercial” Dialog System Architecture

- Task completion providers:
  - Execute actions on behalf of users
  - Interact with external services
- “Baseline” systems:
  - QA: find answers to specific questions users might ask
    - “Who is the president of the United States?”
  - Web search: fallback experience
    - “most famous French poet 1800s”



Q&A vs. Web Search (vs. Task Completion)

- When can we use a question answering system?
  - Answer is known (unambiguously)
  - Answer is specific entity in the world: *e.g.* "`Space Needle`"
- When do we need to fall back to web search?
  - Answer cannot be unambiguously defined
    - How do we define "`best French poet`"?
  - When there is no task capable of answering the user's request
    - *e.g.* "`what is my BMI if I am 6ft tall and weigh 165lbs?`"
  - Answer requires inference beyond system capabilities
    - *e.g.* "`how many calories would I expand if I went to the top of the Space Needle on foot?`"
    - *e.g.* "`set an alarm for 20 minutes before sunrise`"

Knowledge Representation

- Additional question: how to represent knowledge?
  - Unstructured (raw text)
  - Semi-structured (HTML docs)
  - Structured (relational database, knowledge graph)

Knowledge Representation

- Additional question: how to represent knowledge?
  - Unstructured (raw text)
  - Semi-structured (HTML docs)
  - Structured (relational database, knowledge graph)
- Today: talk about web search, QA systems and KGs

Outline

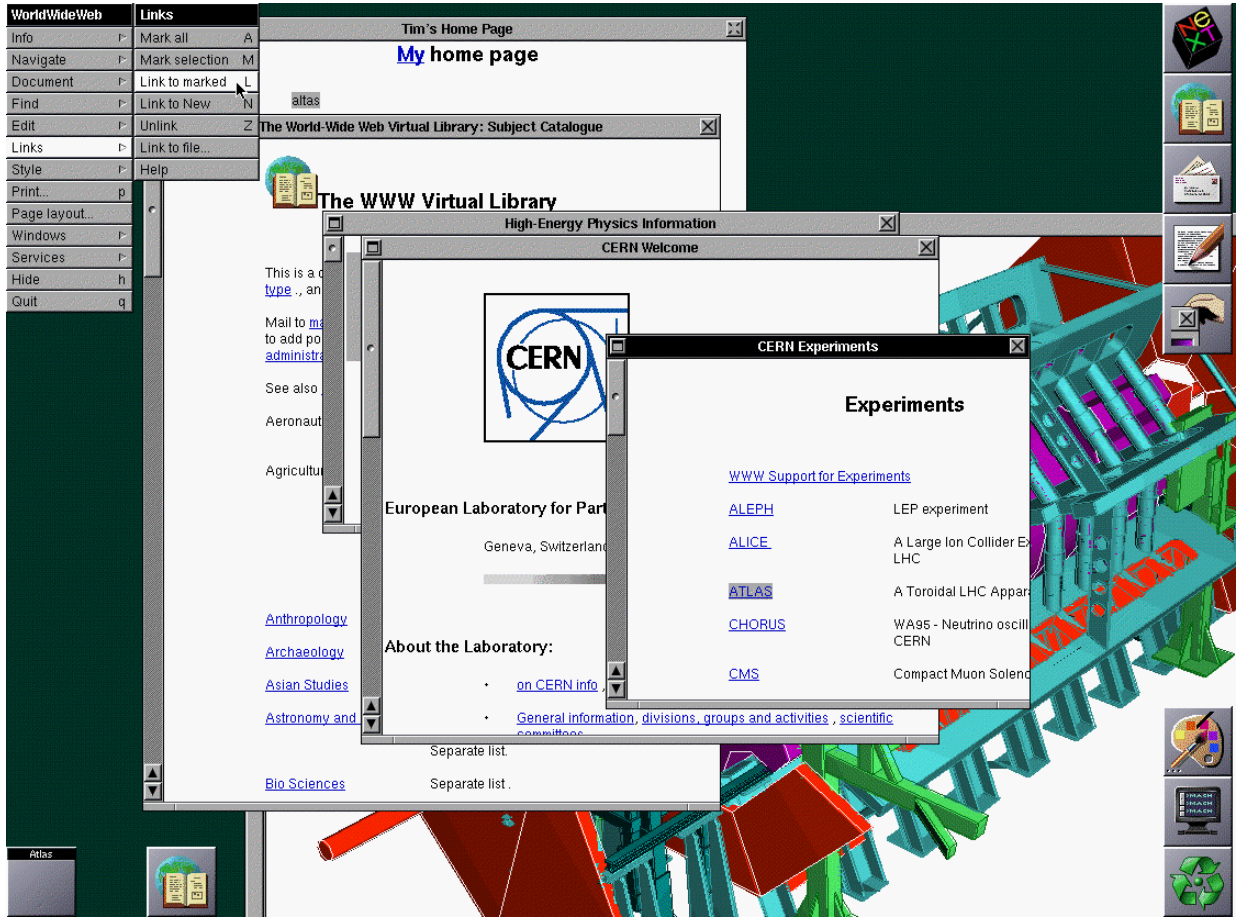
- “Baseline” dialog systems
  - Web search
  - QA systems
- Knowledge graphs (at scale)
  - Representations
  - Building
  - Inference
- Knowledge-driven dialog systems

Outline

- “Baseline” dialog systems
  - Web search
  - QA systems
- Knowledge graphs (at scale)
  - Representations
  - Building
  - Inference
- Knowledge-driven dialog systems

(Web) Search Engines

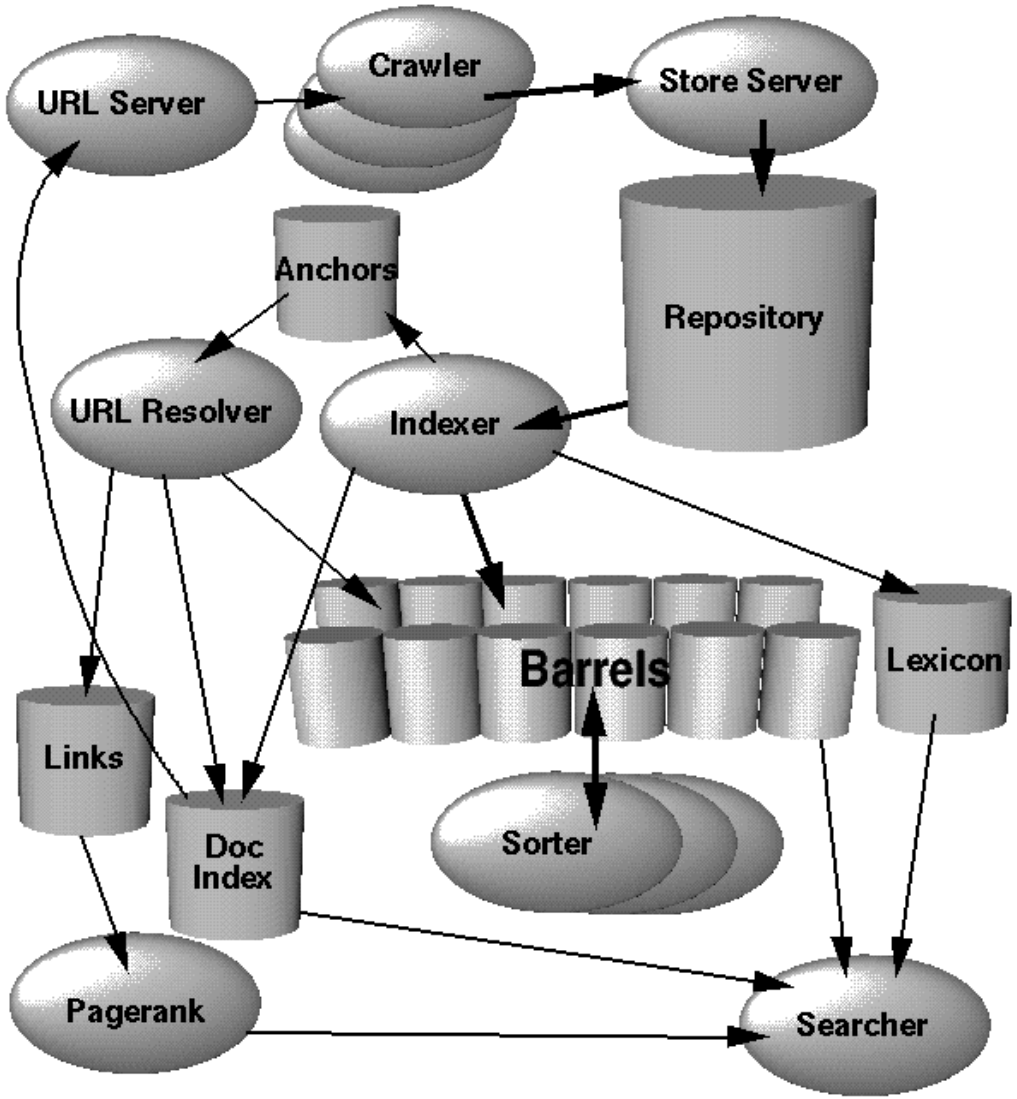
- Semi-structured/unstructured documents
  - Often with markup
  - Links connect pairs of docs
- Web search: given query, find best-matching document(s)



https://www.w3.org/History/1994/WWW/Journals/CACM/screensnap2_24c.gif

Anatomy of a Search Engine

- Brin & Page, 2000
- Describes an initial version of Google
- Core components:
- Index side:
  - Crawler – retrieve documents from web
  - Indexer – extract information from docs
  - Barrels/Lexicon/DocIndex – core search engine data structures
- Querying side:
  - Searcher – retrieve matching documents
  - PageRank – rank matched documents

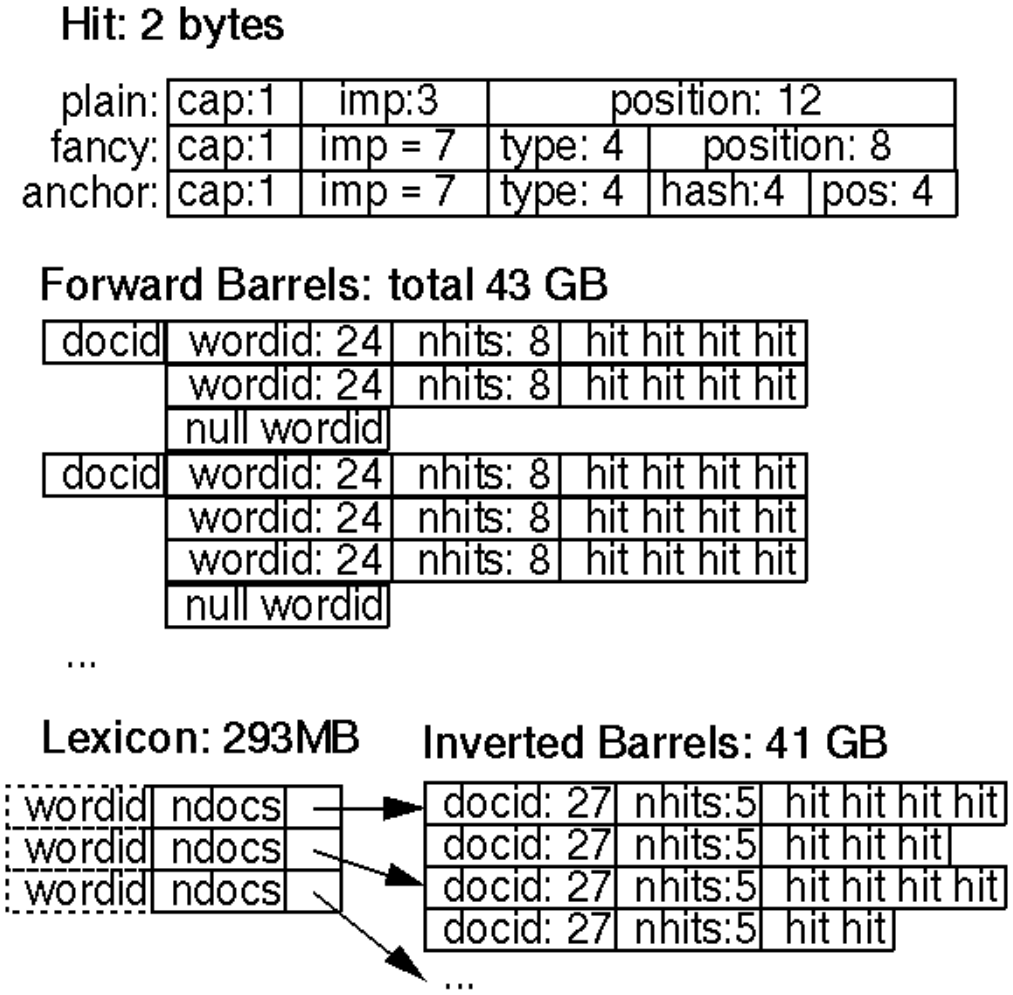


Crawling

- Primarily an engineering problem!
- How to deal with web-scale processing?
  - Lots of caching & parallelism (*e.g.* DNS lookups)
  - Asynchronous IO, data queues
- How to deal with errors?
  - Many errors very rare but can cause significant problems
    - *e.g.* crawl an online game – crawler starts interacting with the game
  - Need good recovery strategies from rare errors, very robust programming

Core Indexing Data Structures

- Lexicon: efficient storage of all words in index:
  - Hashtable (Google paper)
  - Alternatives: B-tree, trie…
- Hit: vector of occurrences of a word in a document
- Forward index: map docids to words
- Inverted index: map words to docids
  - Key to make query process fast
- Barrels: efficient data structure for storing indexes (hits)



Query Execution

Two step process:

1. Candidate generation: efficient search over index data structures
   - Essentially merge sort over inverted index barrels
2. Re-ranking: many features
   - Location of words (title, body, anchor text)
   - Word proximity (how close are words in query to each other in document?)
   - TF-IDF features (paper doesn't explicitly mention this)
   - PageRank: model of user behavior
     - Weigh links to page by count & reliability of each link
     - More links from diverse pages are good
     - Links from highly-ranked pages are also good

“Modern” Search Engines

- Many advances in last 15 years
- Much more sophisticated indexing
  - Support indexing of different document types (contents & metadata)
  - Increased scale (much larger indexes)
- More sophisticated ranking
  - Typically, multiple ranking “layers”
    - L1: generate subset of results potentially relevant
    - L2+: re-rank using increasingly sophisticated techniques, personalized features
- Query processing techniques
  - Query reformulation
  - Query prediction (auto-suggest)

Outline

- “Baseline” dialog systems
  - Web search
  - QA systems
- Knowledge graphs (at scale)
  - Representations
  - Building
  - Inference
- Proactive dialog systems
- Knowledge-driven dialog systems

Question Answering

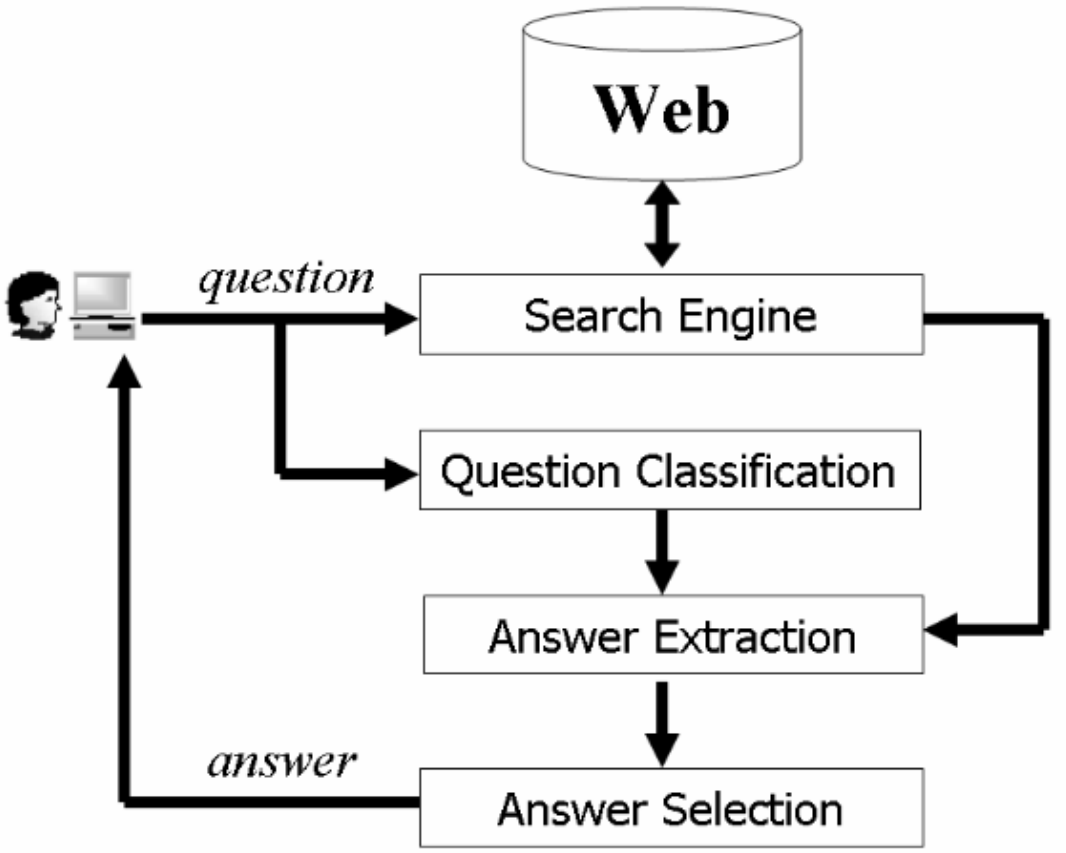
- Important task in the academic (& commercial) IR community
  - TREC (Text REtrieval Conference): track dedicated to Q&A (2000-2007)
- Core idea: identify answer passage directly in indexed documents
  - Return answer, not link to document
- Many different approaches:
  - Data mining (search for short facts using keywords)
  - Information retrieval (search for facts in web-scale corpora)
  - NLP/NLU-based (POS tagging, syntactic/semantic parsing, NER)
  - Inference systems (semantic parsing, discourse, graph methods)

Question Answering

- Important task in the academic (& commercial) IR community
  - TREC (Text REtrieval Conference): track dedicated to Q&A (2000-2007)
- Core idea: identify answer passage directly in indexed documents
  - Return answer, not link to document
- Many different approaches:
  - Data mining (search for short facts using keywords)
  - Information retrieval (search for facts in web-scale corpora)
  - NLP/NLU-based (POS tagging, syntactic/semantic parsing, NER)
  - Inference systems (semantic parsing, discourse, graph methods)

Web-Based QA Systems

- Focus on *wh*-questions
  - "`Who killed Kennedy?`"
- Typical architecture:
  - Search Engine: find documents which may contain answer
  - Question Classification: determine type of desired answer (*e.g.* factoid, description, definition)
  - Answer Extraction: find answer candidates in documents
  - Answer Selection: rank answers based on IR/similarity techniques



Gupta & Gupta, 2012

AskMSR (Banko *et* al., 2001; Brill *et* al., 2001)

- Key idea: exploit redundancy
- Query-side: generate multiple queries: simple rewrite patterns
- Retrieval-side:
  - Retrieve results from search index
  - Compute *n*-gram patterns in results
  - Filter *n*-gram patterns based on frequency & match to rewrite patterns
  - Tile *n*-grams: simple NLG (concatenative)

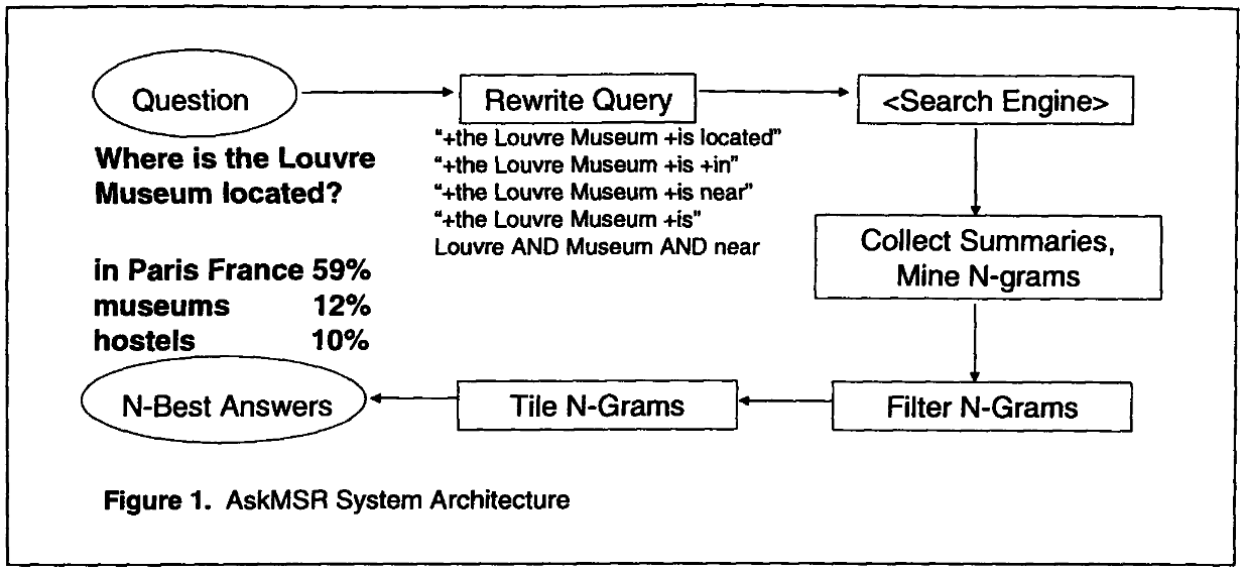


Figure 1. AskMSR System Architecture

Web-based Question Answering: Revisiting AskMSR (Tsai *et* al., 2015)

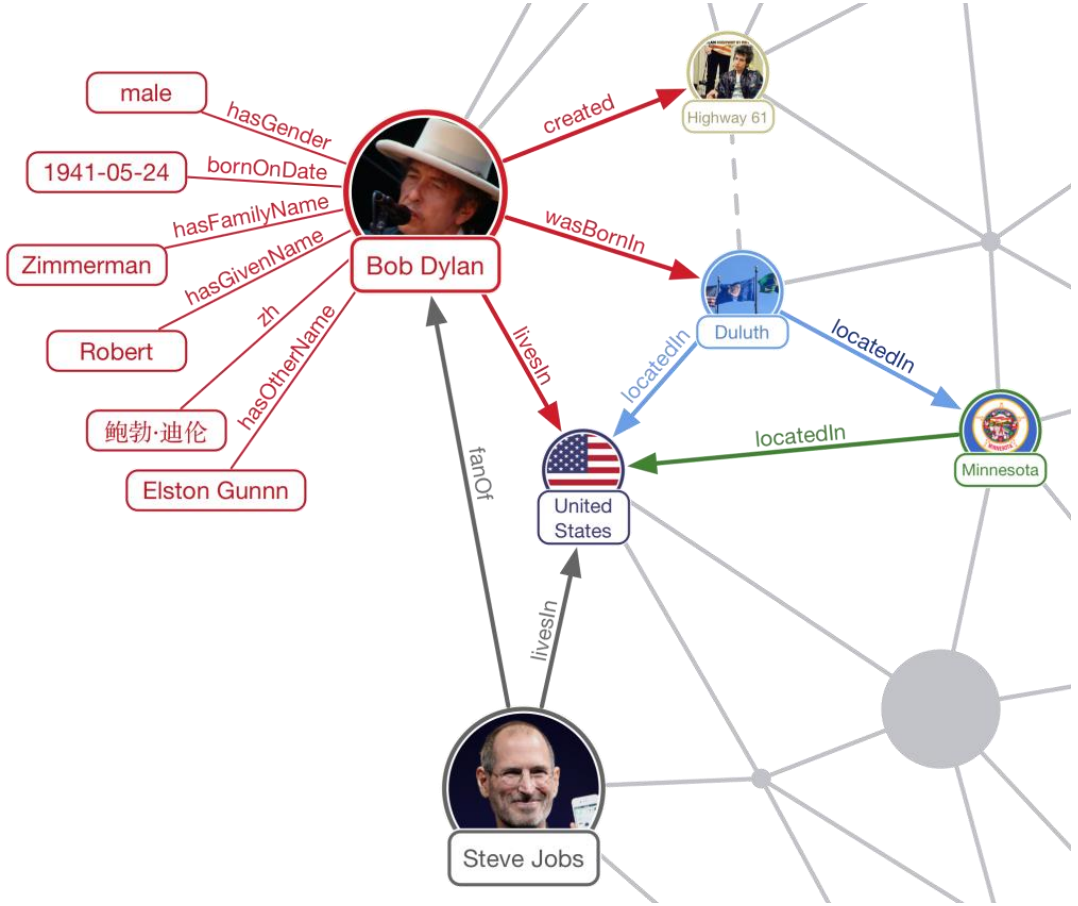
- Key finding: query reformulation less important
  - Queries now often included in web indexes alongside answers
  - Reformulation is now part of web search
- Architecture more similar to that described in Gupta & Gupta, 2012
  - Question classification: 13 categories, rule-based mapping
  - Answer extraction: apply NER if question is entity typed; else use *n*-grams
  - Filtering: remove *n*-grams with certain properties (contain verbs, stop words…)
  - Tiling: similar to original AskMSR (concatenative NLG)
  - Ranking: binary classifier which combines:
    - WordNet-based vector space features
    - Wikipedia-based text similarity features
    - Other lexical & NER features

Outline

- "Baseline" dialog systems
  - Web search
  - QA systems
- Knowledge graphs (at scale)
  - Representations
  - Building
  - Inference
- Knowledge-driven dialog systems

Knowledge Graphs

- Large repositories of structured information, containing:
  - Entities (persons, locations, organizations, etc.)
  - Relationships between entities
- Structure means:
  - Entities have *types*
  - Relationships have *types*
- Types form ontology
  - Types themselves are graph entities
  - Relationships between types (*e.g.* inheritance)



https://www.ambiverse.com/wp-content/uploads/2017/03/KnowledgeGraph-Named-Entities-Bob-Dylan-Relations-1024x846.png

Knowledge Graph Examples

- Academic community:
  - DBPedia: extract information out of Wikipedia (automatically + manual rules)
  - Freebase: freely editable KG (now defunct)
  - YAGO: Wikipedia + WordNet + GeoNames
  - NELL (Never-Ending Language Learning): automatically extracted from web
- Commercial:
  - Google Knowledge Graph (originated from Freebase)
  - Microsoft Satori
  - Facebook Entity Graph
  - Amazon product catalog (incl. reviews, recommendations, etc.)
  - Many other private KGs (e.g. Dominos Pizza product catalog)

Scale of Knowledge Graphs

- Academic/open KGs:

Knowledge Graph	Number of Entities	Number of Relationships	Number of Types
DBPedia	6.6 million	13 billion (facts)	760 classes, 3000 properties
YAGO	10 million	120 million (facts)	350,000 classes
NELL	13.5 million NPs	50 million beliefs	271 semantic categories 370,000 concepts, 350,000 properties

- Commercial KGs:
  - Not open – hard to estimate
  - Schema.org – commercial consortium-backed ontology:
    - ~600 entity types, 900 relationship types

Knowledge Graph Tasks

- How to represent?
- How to build and maintain?
- How to query?

Outline

- "Baseline" dialog systems
  - Web search
  - QA systems
- Knowledge graphs (at scale)
  - Representations
  - Building
  - Inference
- Knowledge-driven dialog systems

KG Representations: RDF

- Resource Description Framework (RDF): most popular representation
  - Series of triples: <*subject*, *predicate*, *object*>
  - Often also include a 4th element: confidence (in relationship)
- RDF tuple example:

  subject:Tom_Cruise predicate:Starring_In object:Top_Gun confidence:0.9993
- Typically, subjects/objects/predicates represented as URIs
  - Each entity in the graph has a URI
  - In some graphs, predicates are not part of graph (so no URIs)
  - Objects can sometimes be literals (string/number/etc.)

How to Encode KGs

- Depends on purpose!
- Academic graphs often stored as XML or JSON-LD formats
  - But these formats tend to be verbose
  - Binary representations can be used to save space (especially in commercial KGs)
- Storage formats (XML/JSON-LD) not well-suited for graph consumption
  - RDF in general is not very developer-friendly
  - Developers like to work with classes, not sequences of triples!
- Consumption: generate code
  - Each type in the ontology could correspond to a class (for example)
  - Conversion could be automated, but difficulties exist
    - Multiple inheritance, multi-valued properties, localization, etc.

Data Representation Example

Without Markup | Microdata | RDFa | JSON-LD

```
Jane Doe
<img src="janedoe.jpg" alt="Photo of Jane Joe"/>
Professor
20341 Whitworth Institute
405 Whitworth
Seattle WA 98052
(425) 123-4567
<a href="mailto:jane-doe@xyz.edu">jane-doe@illinois.edu</a>
Jane's home page:
<a href="http://www.janedoe.com">janedoe.com</a>
Graduate students:
<a href="http://www.xyz.edu/students/alicejones.html">Alice Jones</a>
<a href="http://www.xyz.edu/students/bobsmith.html">Bob Smith</a>
```

Without Markup | Microdata | RDFa | JSON-LD

```
<script type="application/ld+json">
{
  "@context": "http://schema.org",
  "@type": "Person",
  "address": {
    "@type": "PostalAddress",
    "addressLocality": "Seattle",
    "addressRegion": "WA",
    "postalCode": "98052",
    "streetAddress": "20341 Whitworth Institute 405 N. Whitworth"
  },
  "colleague": [
    "http://www.xyz.edu/students/alicejones.html",
    "http://www.xyz.edu/students/bobsmith.html"
  ],
  "email": "mailto:jane-doe@xyz.edu",
  "image": "janedoe.jpg",
  "jobTitle": "Professor",
  "name": "Jane Doe",
  "telephone": "(425) 123-4567",
  "url": "http://www.janedoe.com"
}
</script>
```

http://schema.org/Person

Ontology Definitions

- Custom ontologies (for different types of graphs):
  - foaf – relations in social networks
  - Dublin Core – publications
  - Good Relations – products
- General ontology: schema.org
  - Not a “global ontology” of the world (but moving in that direction)
  - Problems:
    - Not sufficiently expressive (even backers may develop extensions)
    - Too rich for most developers to use successfully
    - **Tension between expressivity and usability**

Outline

- "Baseline" dialog systems
  - Web search
  - QA systems
- Knowledge graphs (at scale)
  - Representations
  - Building
  - Inference
- Knowledge-driven dialog systems

How to Build a General Knowledge Graph

- Very hard problem!
  - Essentially named entity recognition + entity linking at web scale
- Typical approach:
  - Start with Wikipedia (and other similar catalogs of entities)
    - This gives us a set of initial entities to work with
  - Extract all entity mentions from Wikipedia text (and similar)
  - Link entities extracted from text to entities in the KG
  - Identify relation types (if relations are in KG, similar to above)
- Problems:
  - How to tag entities in text?
  - How to identify correct entity to link to?

Named Entity Recognition (NER)

- One of the classic NLP problems
  - And classic dialog problem: slot tagging!
- Many approaches exist:
  - Dictionary based
  - Pattern based (write FSTs/PCFGs)
  - Supervised ML methods (*e.g.* CRFs, LSTMs)
- Differences between (general) NER and Dialog NLU:
  - NER often done offline – can use more sophisticated models/methods
  - NER CRFs often use complex features: POS tags, syntactic parse features
  - To identify entity type: SVM/Decision trees often used (or DNNs)

(Classical) Entity Linking

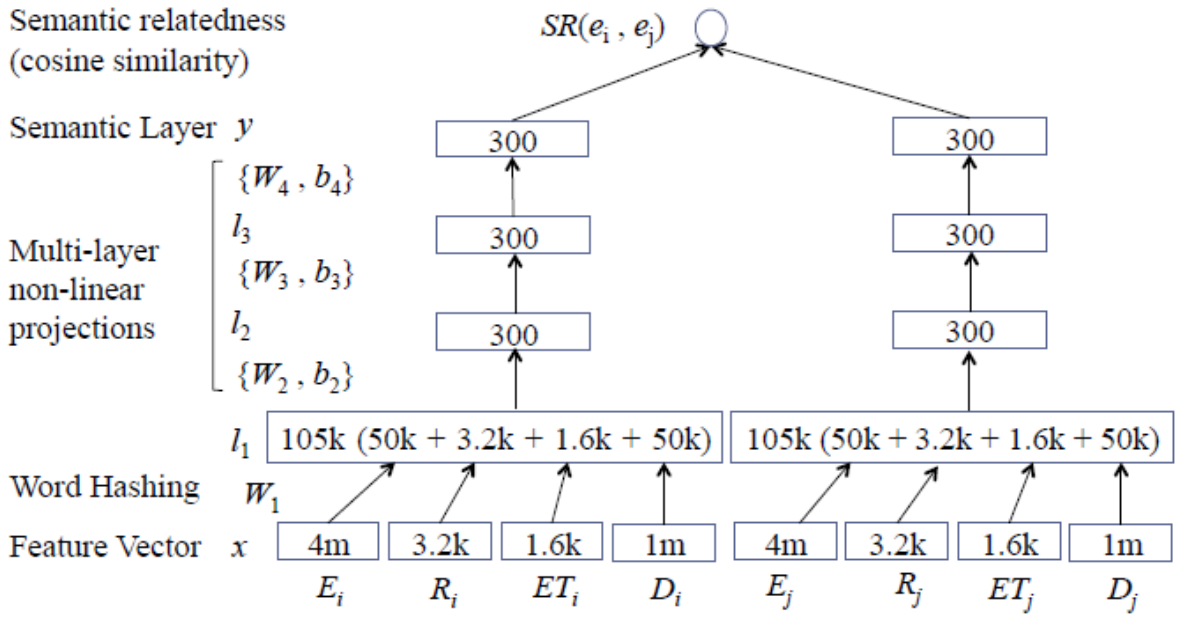
- Typically: two step process
- Step 1: gather candidates
  - Often use IR approaches (entity text/mention to search KG)
- Step 2: rank candidates
  - Context in which entity appears very important
    - *e.g.* if document talks about sports teams, “Bayern” is the soccer team, not the region
  - Features: co-occurrence statistics, textual similarity (*e.g.* word embeddings)
  - Algorithms: various ranking methods (*e.g.* gradient boosted decision trees)

Neural Networks for Entity Linking

- Recent work focuses on using NN architectures
- Note: typically, in academic papers, KG is Wikipedia
  - This is different from "practical" (true) KGs: Google Graph/MS Satori/etc.
  - Can adapt methods to practical KGs
    - *e.g.* use KG entity name instead of Wikipedia page title
- Many architectures possible:
  - Feed forward: He *et* al., 2013; Huang *et* al., 2015; Sun *et* al., 2015
  - Feed forward + attention: Ganea and Hoffman, 2017
  - CNNs: Francis-Landau *et* al., 2016

DSRM for Entity Linking (Huang *et* al., 2015)

- Goal: compute semantic relatedness between entities
  - Not full entity linking system
  - Can be used as reranker
- Features:
  - List of connected entities
  - List of relationships to other entities
  - List of types entity can hold
  - Description of entity
- Word hashing: used for dimensionality reduction
  - Letter *n*-gram technique



Graph Maintenance

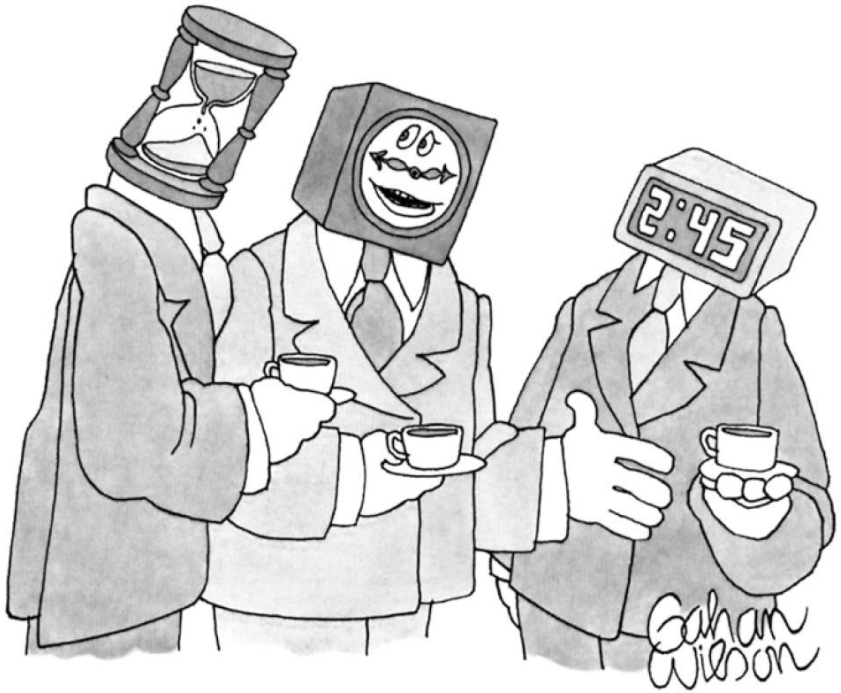
- Many different problems included here:
  - Conflation/deduplication (find possible duplicates & merge if necessary)
  - Refreshing (add new information to existing entities)
    - *e.g.* Who just won the Superbowl
  - Removal (of information in entity that no longer applies)
    - *e.g.* Barack Obama no longer president of the U.S.
  - Ontology (and data) versioning
    - If new relationships added, need to re-crawl to add new information to entities
    - If relationships removed or modified, need to clean up affected entities
    - If types split/inheritance structure changed, need to restructure graph

Merging/Reconciling Knowledge Graphs

- Many applications require integration of multiple KGs:
  - Amazon product catalog: feeds from many merchants
  - Library of Congress: information from many library systems
  - T-Mobile + Sprint HR systems
- Basic approach:
  - Reconcile ontologies (“ontology mapping problem”)
  - Apply entity linking/disambiguation (given reconciled ontologies)

Ontology Mapping Problem

"Hardest" problem in information science!



"Basically, we're all trying to say the same thing."

http://dit.unitn.it/~accord/RelatedWork/Matching/Noy-MappingAlignment-SSSW-05.pdf

(Material on next few slides: Talk by Natasha Noy, UW)

Differences in Ontologies

- Why: the world is complicated!
  - Different applications have different needs
  - Different authors have different interpretations
- What:
  - Language-level mismatches
    - Differences in expressiveness or semantics of ontology language
  - Ontology mismatches
    - Differences in the structure of the ontology itself

Language Mismatches

- Syntactic differences
- Expressiveness – does the language support:
  - Union
  - Intersection
  - Negation
  - Disjoints
- Semantics of primitives
  - Union vs. intersection semantics for multiple domain/range declarations

Ontology Mismatches

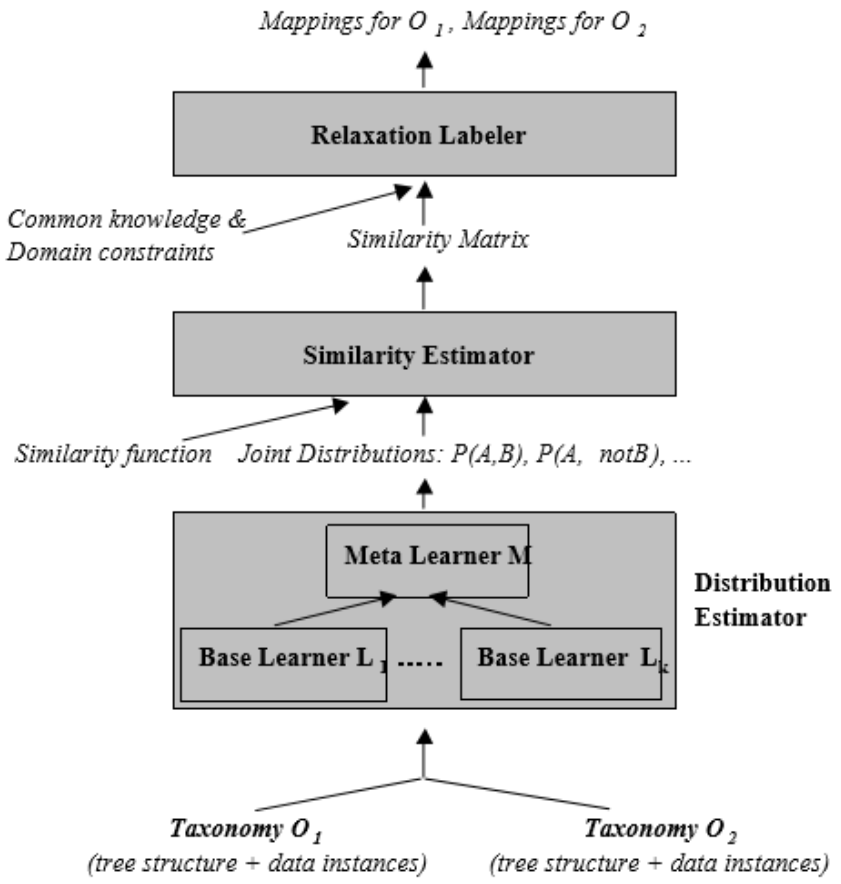
- Same terms describing different concepts
- Different terms describing the same concept
- Different modeling paradigms
  - *e.g.* time intervals – sequence of points? Start/end?
- Different modeling conventions
  - *e.g.* when to subclass? How to represent default values?
- Different level of granularity
  - *e.g.* is “bicycle” represented as subtype, or property of “vehicle”

Building Ontology Mappings

- By hand
  - Still preferred method when best results are required
  - But expensive/human labor intensive
- Rule-based
  - Map into a common ontology (interlingua)
  - Lexical mappings: substring matching, soundex, edit distance, etc.
  - External resources: WordNet similarity (or map into other linguistic resources)
- Graph-based
  - Ontology structure: treat ontologies as graphs and compare subgraphs
- Machine learning

GLUE: ML-Based Ontology Mapping (Doan *et* al.)

- Key idea: compute distributional similarity of ontology nodes
- Distribution estimator:
  - Content learner: use contents of entity and apply Naïve Bayes
  - Name learner: use labels of attributes/entity types and apply Naïve Bayes
  - Meta-learner: weighted sum
- Similarity estimator: apply user-provided similarity function
- Relaxation labeler: iteratively re-estimate labels for nodes in graph given estimates for its neighbors



Outline

- “Baseline” dialog systems
  - Web search
  - QA systems
- Knowledge graphs (at scale)
  - Representations
  - Building
  - Inference
- Knowledge-driven dialog systems

Entity Retrieval

- Different approaches for different tasks:
- Possible tasks:
  - Expert finding
  - General entity ranking in context (*e.g.* temporal context for news)
  - Ad-hoc entity retrieval (AOR)
- Possible approaches:
  - Information retrieval methods
  - SPARQL query execution
- **Note**: recall inference QA from before - this is essentially QA over KGs
  - We will still see a lot of IR-based techniques!

KG Search Tasks

- Expert finding: traditional enterprise scenario
  - Find experts on a particular task, *e.g.* "`Seattle deep learning experts`"
  - Relevant for large social networks (*e.g.* LinkedIn)
  - KG contains: HR records, enterprise documents, presentations, emails…
- Generic entity ranking: generalization of expert finding
  - *e.g.* "`Impressionist art museums in Holland`"
  - Often used to search collections that evolve over time
    - News articles about the same event: contents change over time
    - Leverage temporal sequence of articles to generate better rankings
- Ad-Hoc object retrieval: find specific single entity given complex user query
  - *e.g.* "`Most recent Tom Hanks movie set in Australia in the 1950s`"
  - Not tied to single (or specific) entity types: can search entire KG
  - Users not aware of RDF ontology/schema

Information Retrieval Techniques

- Two methods: document vs. object-centric
  - Both use inverted indexes (typical IR building technique)
- Document-centric: index over document contents
  - Query retrieves ranked list of documents
  - Extract candidate experts from document metadata
- Object-centric: index over candidate experts (*i.e.* people KG entities)
  - Extract names of experts from metadata documents
  - Expert profiles: aggregate all relevant documents to a candidate
  - Build index over aggregate information
  - Query retrieves ranked list of experts (directly)

IR-based AOR: IR Techniques at Scale

- Build inverted indexes over the entire KG
- Difficulty: how to scale index structure
- Horizontal indexing: build two indexes
  - Term index: store list of properties in which a particular term (word) appears
  - Property index: store list of entities which contain particular property
- Vertical indexing: build one index per property
  - Each index stores all terms appearing in that property
  - Smaller footprint (number unique terms is smaller)
  - Need to merge results across multiple indexes & rank (*e.g.* using BM25F)

Beyond IR: Graph-Based AOR

- Two broad methods:
  - Use graph as refinement step
  - Use graph directly as part of search
- Most common: graph-based refinement
- Step 1: generate candidate list using IR techniques (as before)
- Step 2: re-rank top *k* results using graph structure
  - Need in-memory representation of graph that allows fast traversal
  - See (Tonon *et* al., 2012; Zhiltsov *et* al., 2015)

Mixed AoR: Direct Graph Search

- Less computationally efficient, but possible
- Step 1: convert NL query to SPARQL query
  - Essentially semantic parsing + query generation
  - Semantic parser: can leverage NLU-style analysis + syntactic parse
    - Alternatively, do full semantic parse straight away
  - SPARQL: *SPARQL* Protocol and RDF Query Language (recursive acronym)
    - (relatively) efficient for RDF triple store searches
    - essentially SQL over knowledge graphs
- Step 2: execute SPARQL query
  - In-memory graph structure even more important now
  - Speed up graph look-ups using IR indexes

SPARQL Example

Find me all landlocked countries with population greater than 15 million.

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX type: <http://dbpedia.org/class/yago/>
PREFIX prop: <http://dbpedia.org/property/>
SELECT ?country_name ?population
WHERE {
    ?country a type:LandlockedCountries ;
             rdfs:label ?country_name ;
             prop:populationEstimate ?population .
    FILTER (?population > 15000000) .
}
```

Outline

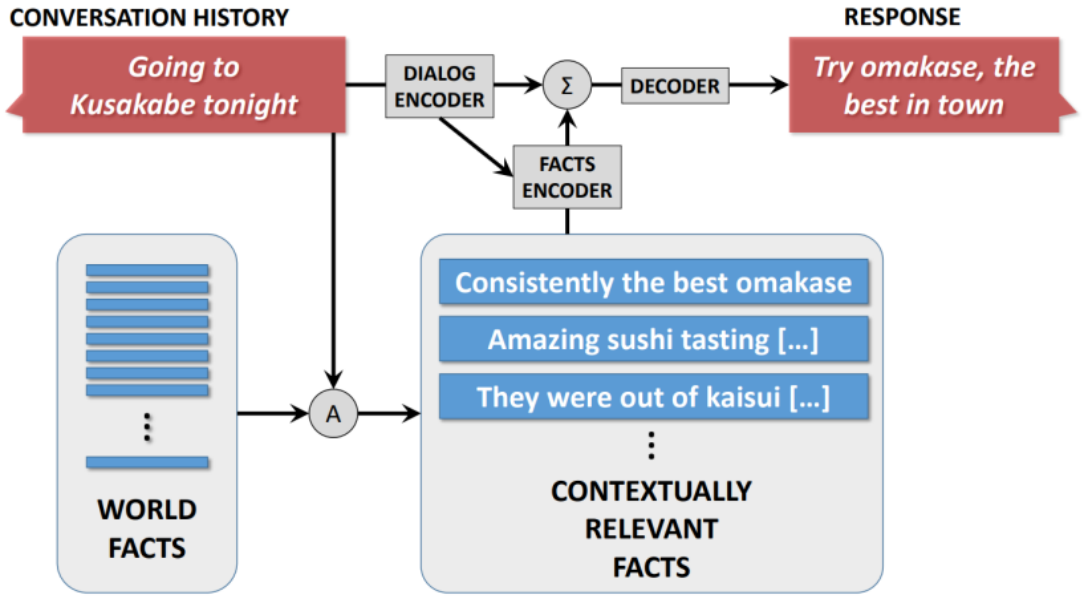
- "Baseline" dialog systems
  - Web search
  - QA systems
- Knowledge graphs (at scale)
  - Representations
  - Building
  - Inference
- Knowledge-driven dialog systems

Task Completion vs. Web/QA Systems

- Task completion:
  - Understanding of straight-forward user intent/goals
  - Ability to carry conversation and execute actions
- Question answering:
  - Ability to perform complex searches over knowledge graphs
  - Only action is `find_entity`
- Web search:
  - Often used for complex queries which cannot be acted upon by single task
- How to combine these with a KG?
  - Most important aspect: how to integrate KG into end-to-end systems?
    - KG lookups tend to be non-differentiable
    - End-to-end systems require propagating gradient across entire network

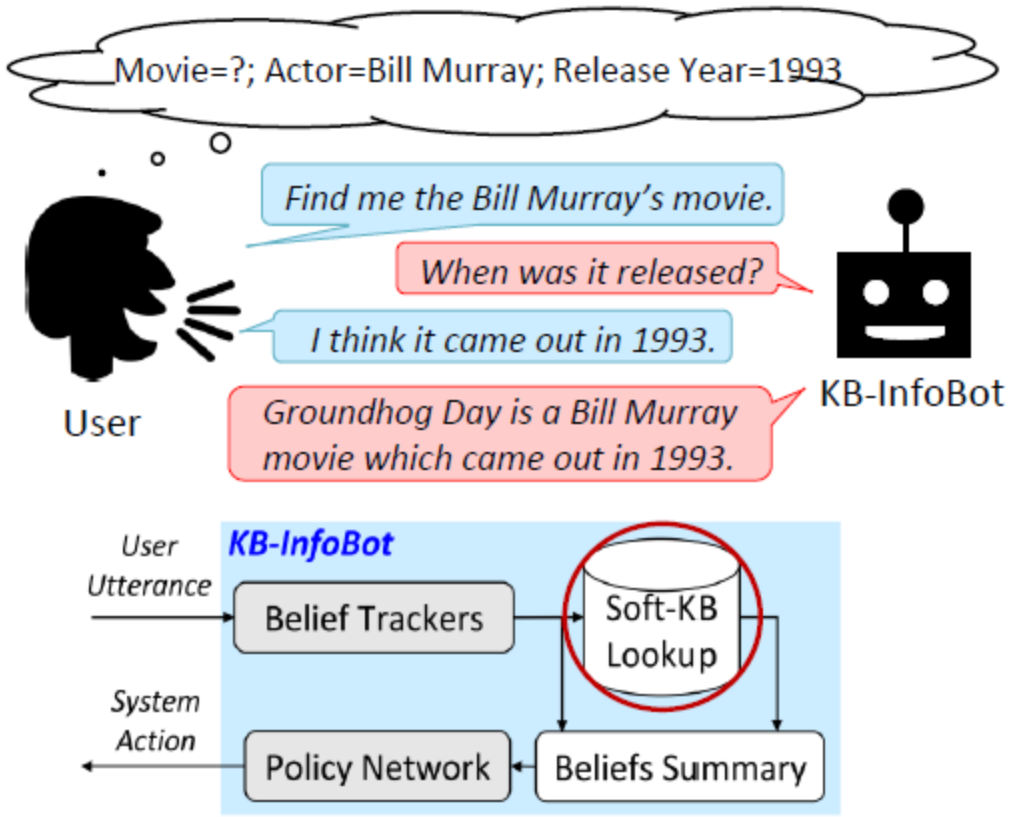
Knowledge Grounded Neural Conversations

- Ghazvininejad *et* al., 2017
- Memory network-like architecture
- Store list of world facts (unstructured database)
- Given user utterance:
  - Select relevant world facts (using IR)
  - Encode facts (using RNN + encoded user utterance)
  - Integrate facts into decoder



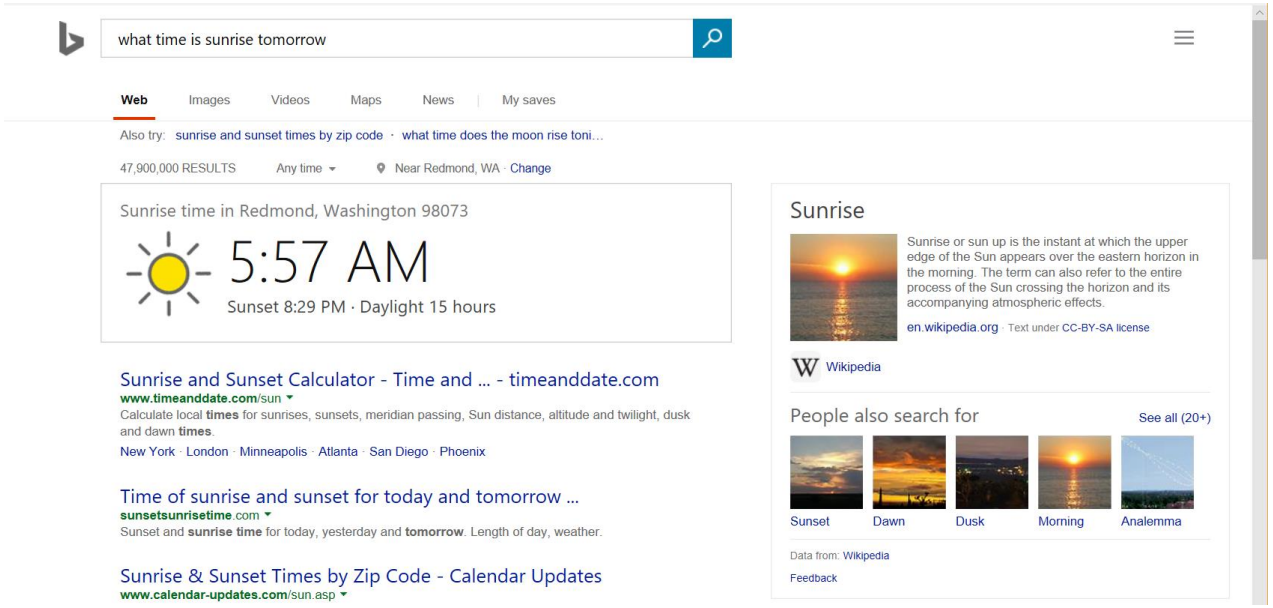
End to End RL-Based InfoBot

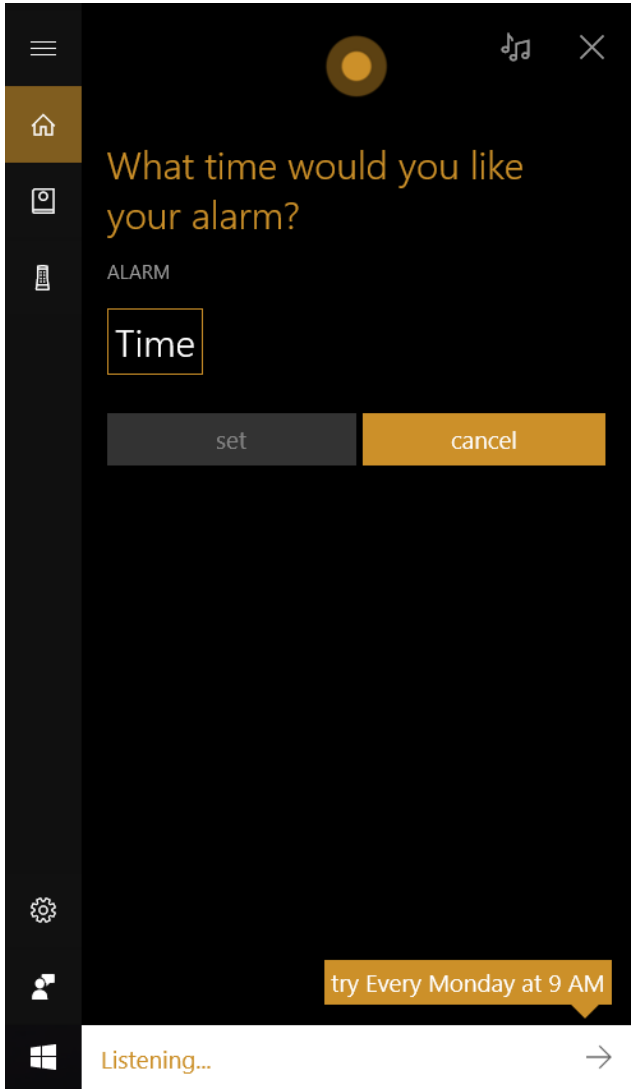
- Dhingra *et* al., 2017
- Key idea: differentiable DB – can train E2E including DB lookups
- Belief tracker: separate GRU/slot
- Policy: GRU -> dense -> softmax
  - Sampled at training to encourage exploration
  - If action=*inform*: provide ordered list of *R* entities from DB (similar to search engine results)
- NLG: templated



What About Complex Task Completion?

“set alarm for 20 minutes before sunrise”



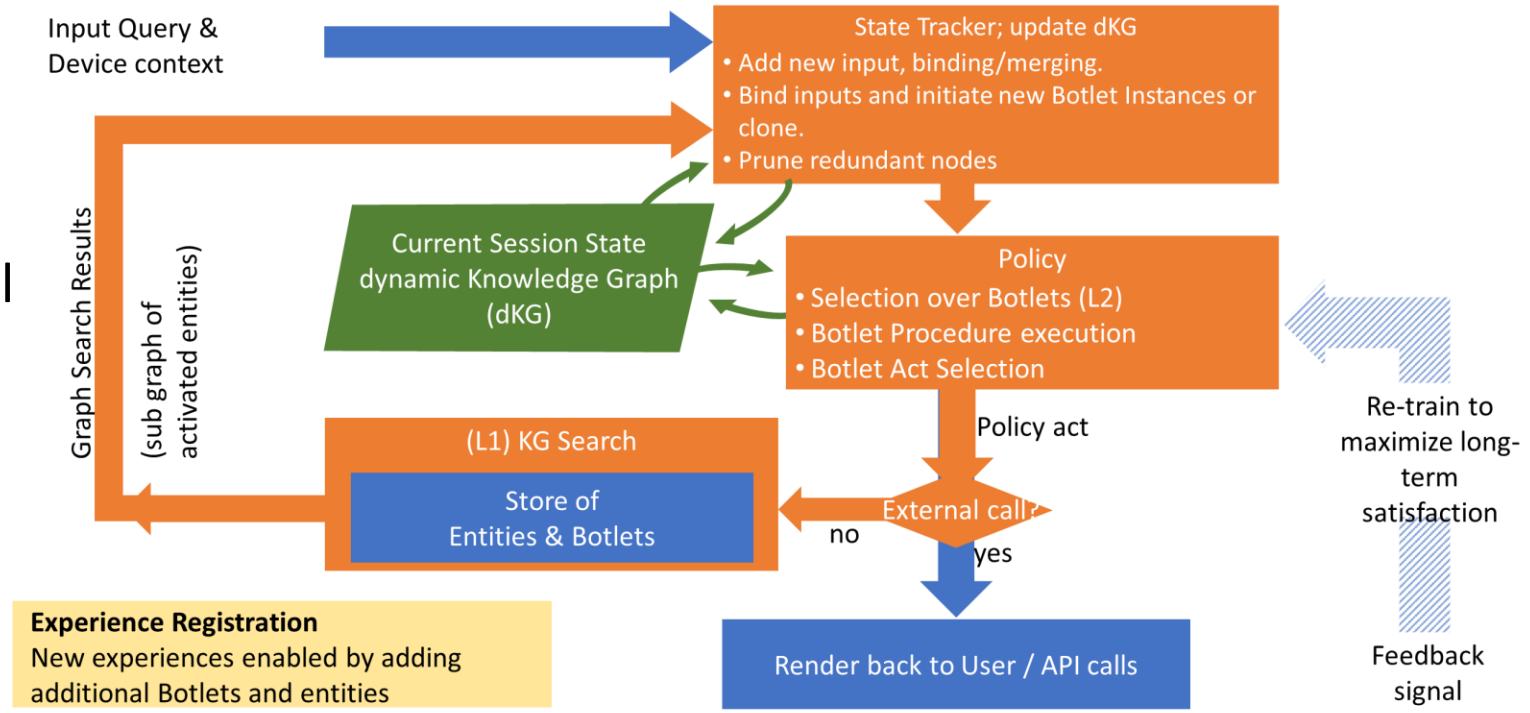


- Current approach:
  - Search for sunrise time
  - Compute (by hand) 20 min earlier
  - Ask assistant to set alarm for that time
- How to do better?

Inference DM Architecture (Crook *et* al., 2018)

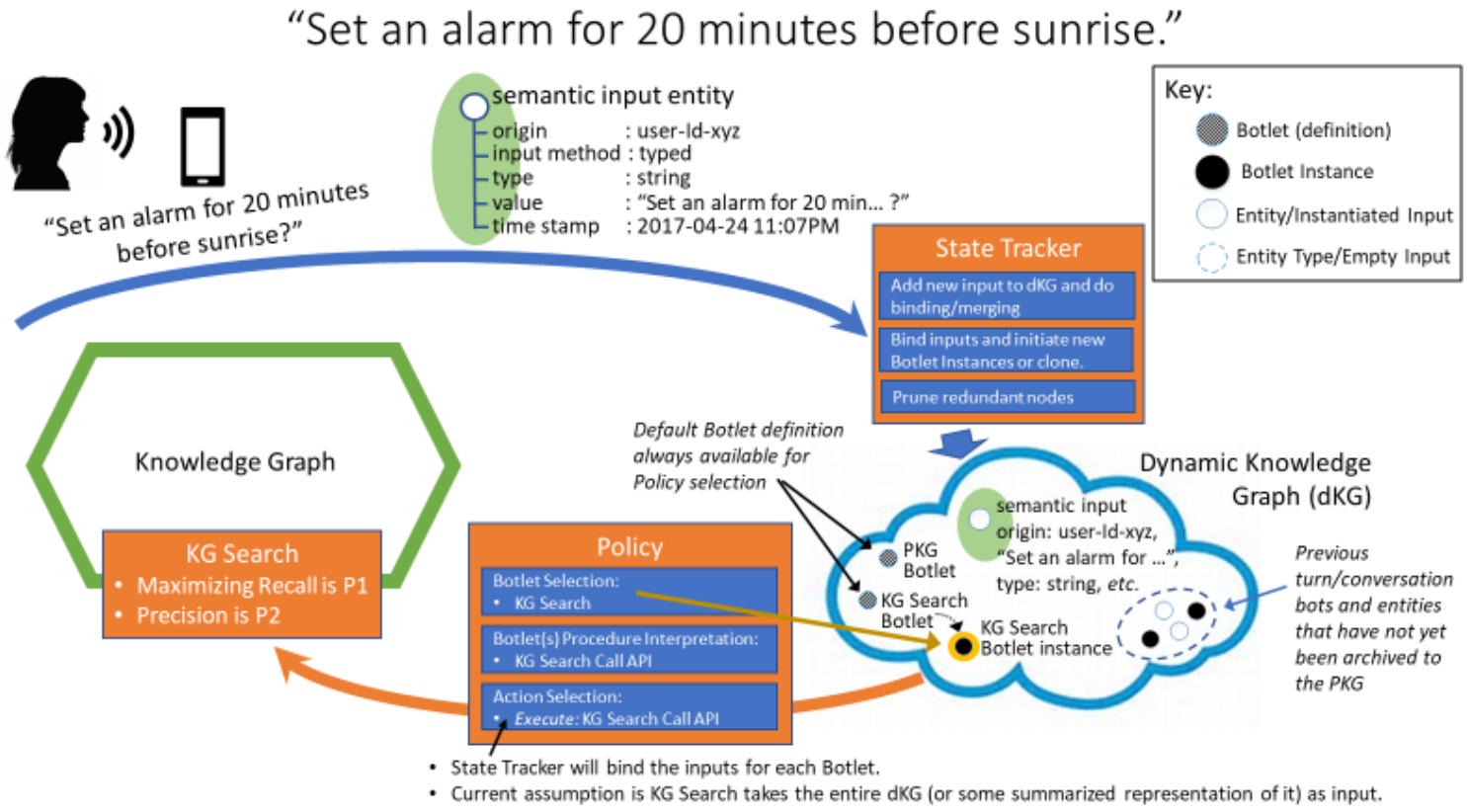
- Define skills/actions as entities in the KG
- Augment typical DM pipeline with:
  - KG search action (which can be implemented in the usual way)
  - Ability to loop back (*i.e.* traverse pipeline multiple times)
  - Decompose NLU problem into multiple steps

  → generate dynamic execution plan



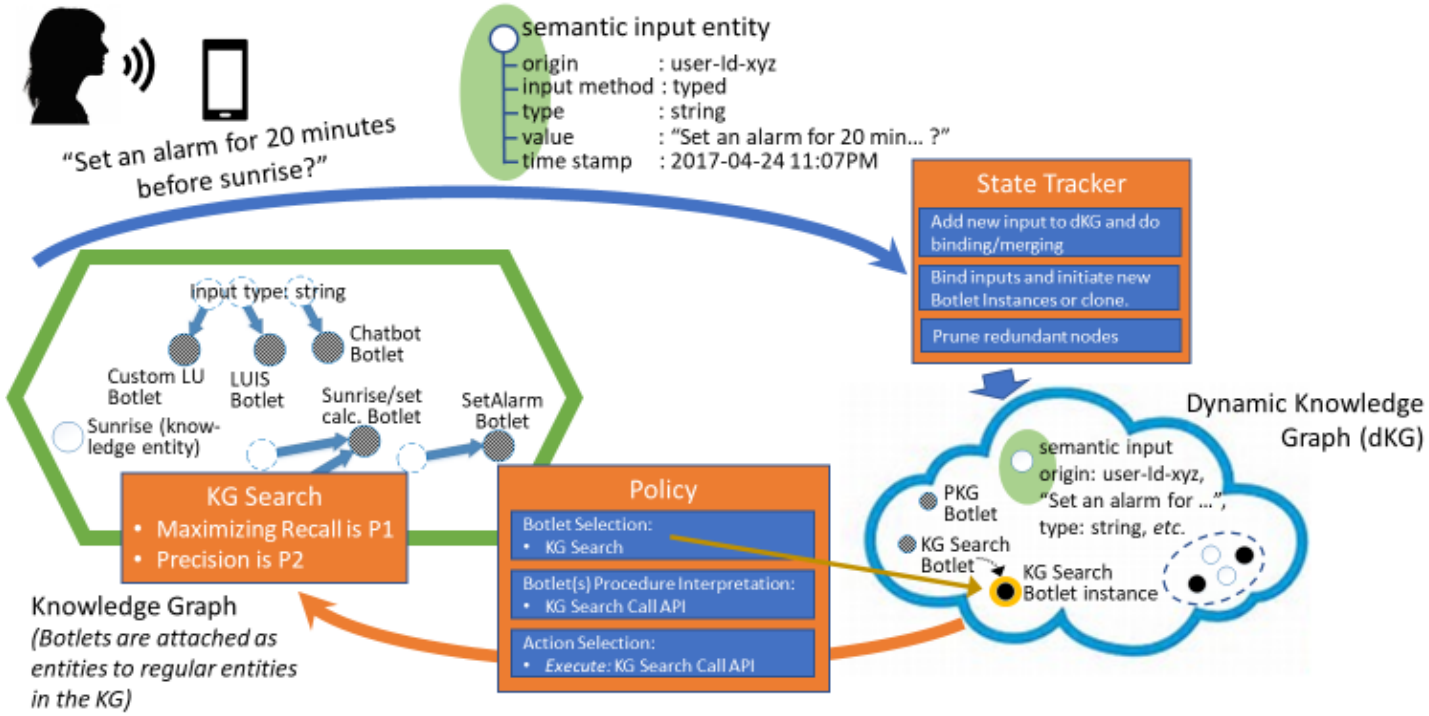
Running Example

- Step 1: process input
  - Mostly empty dKG state
  - Only available action to system: search KG
  - Instantiate search & execute



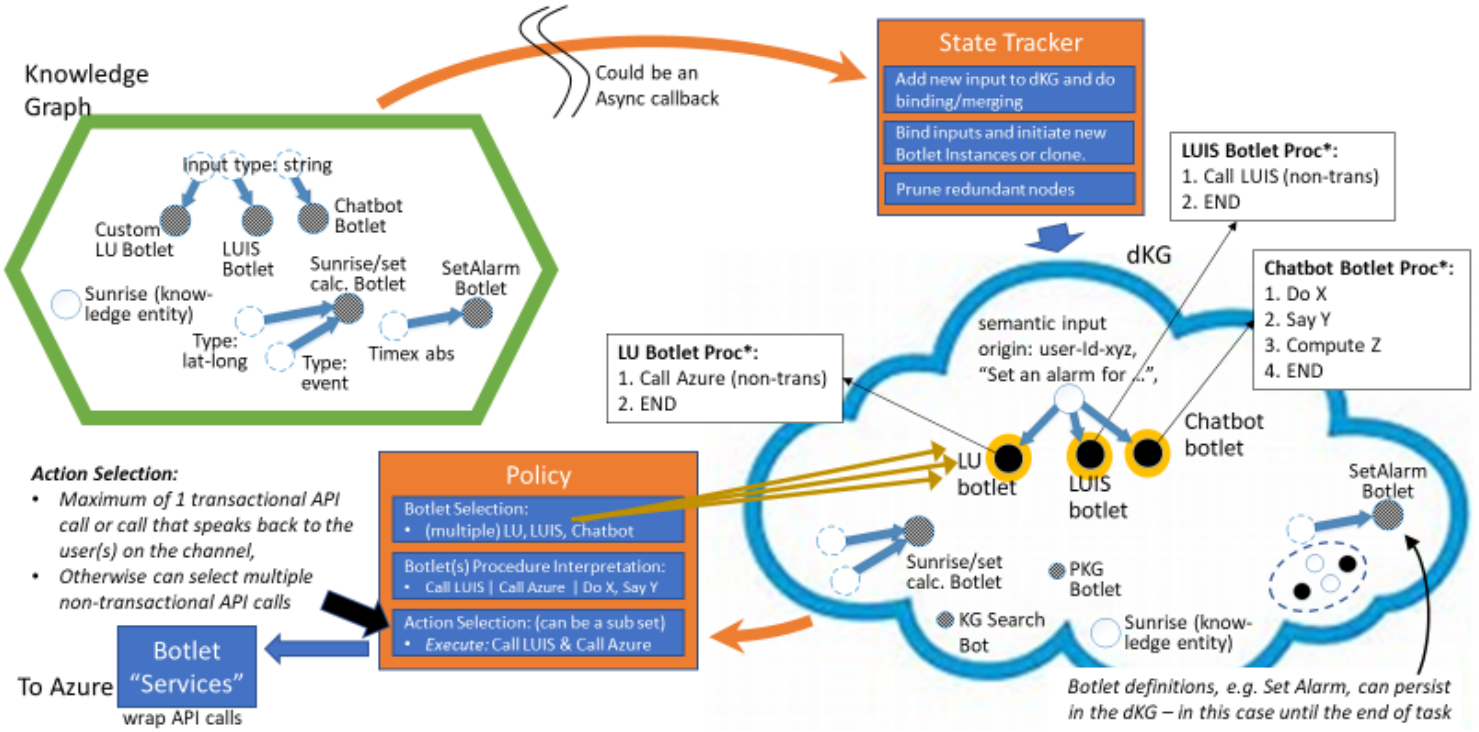
Running Example

- Step 2: KG search
  - Return both entities & “botlets” (skills)
  - Maximize recall (similar to L1 search)



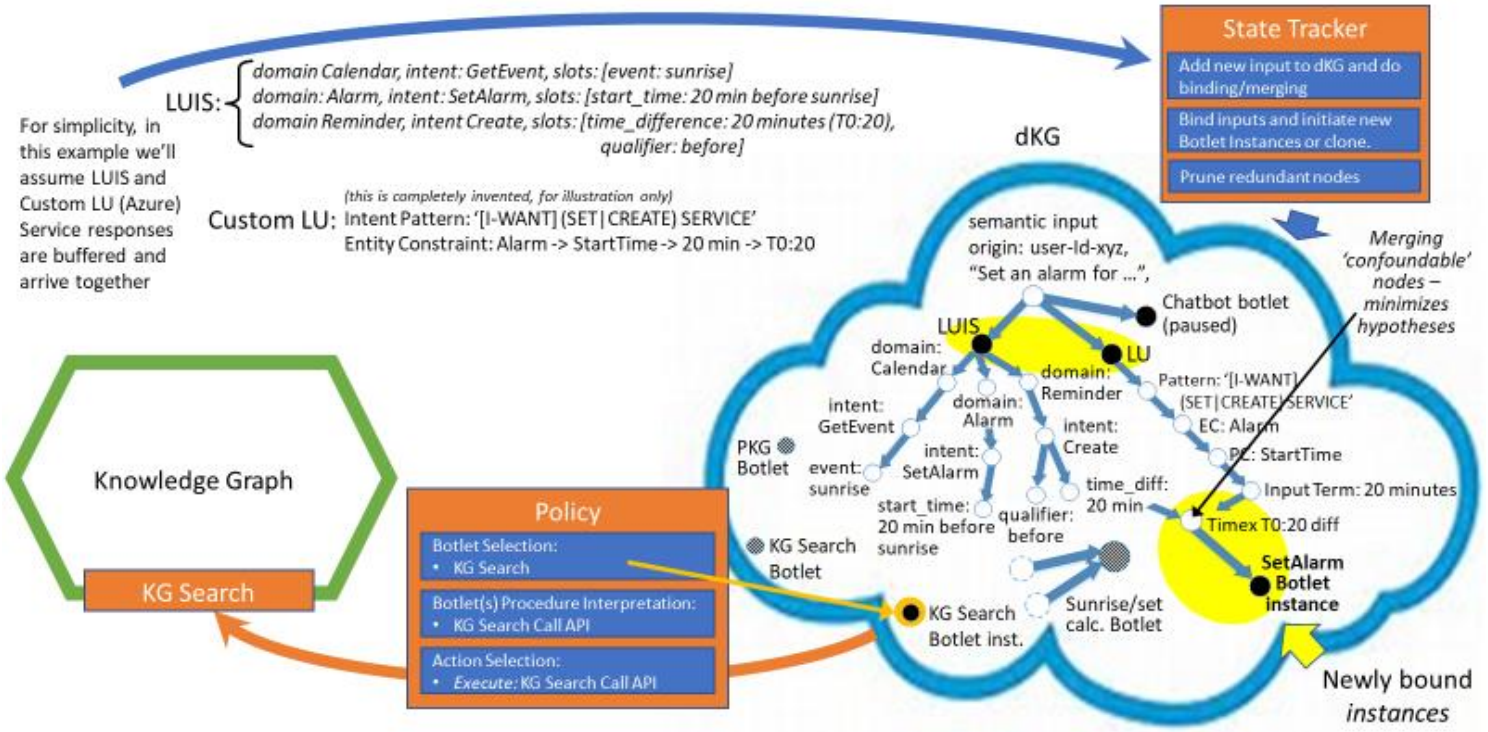
Running Example

- Step 3: Process KG results
  - State Tracker:
    - Integrate results into dKG
    - Select some botlets to instantiate (like L2 search)
      - LU, chat botlets
  - Policy: select botlets to run (like L3 search)
    - LU botlets
- Note: this is the 2nd run of the full loop



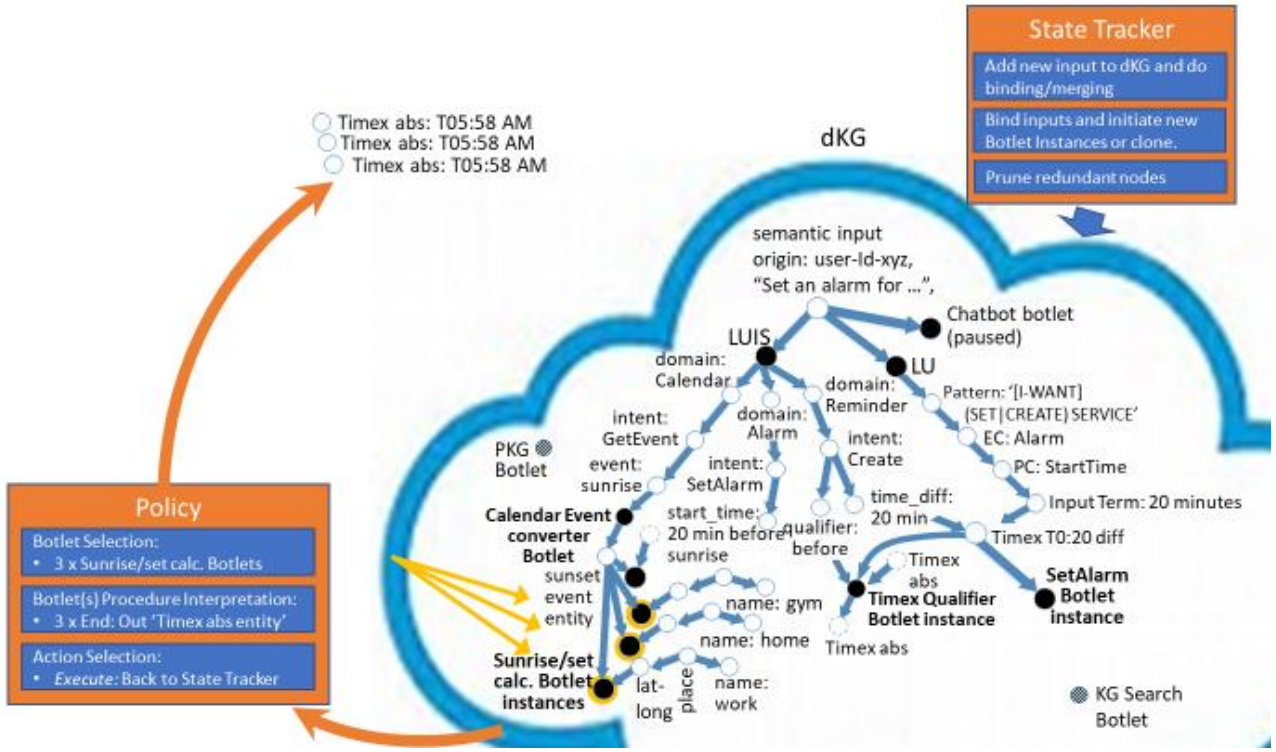
Running Example

- Step 4: Execute selected botlets
  - Get results from two LU botlets
  - Each of them parsed part of the query
  - Integrate parse in dKG (as KG entities)
  - Policy re-evaluates what botlets in dKG can run
    - Nothing interesting can run: search KG once more
- Note: 3rd run through loop…



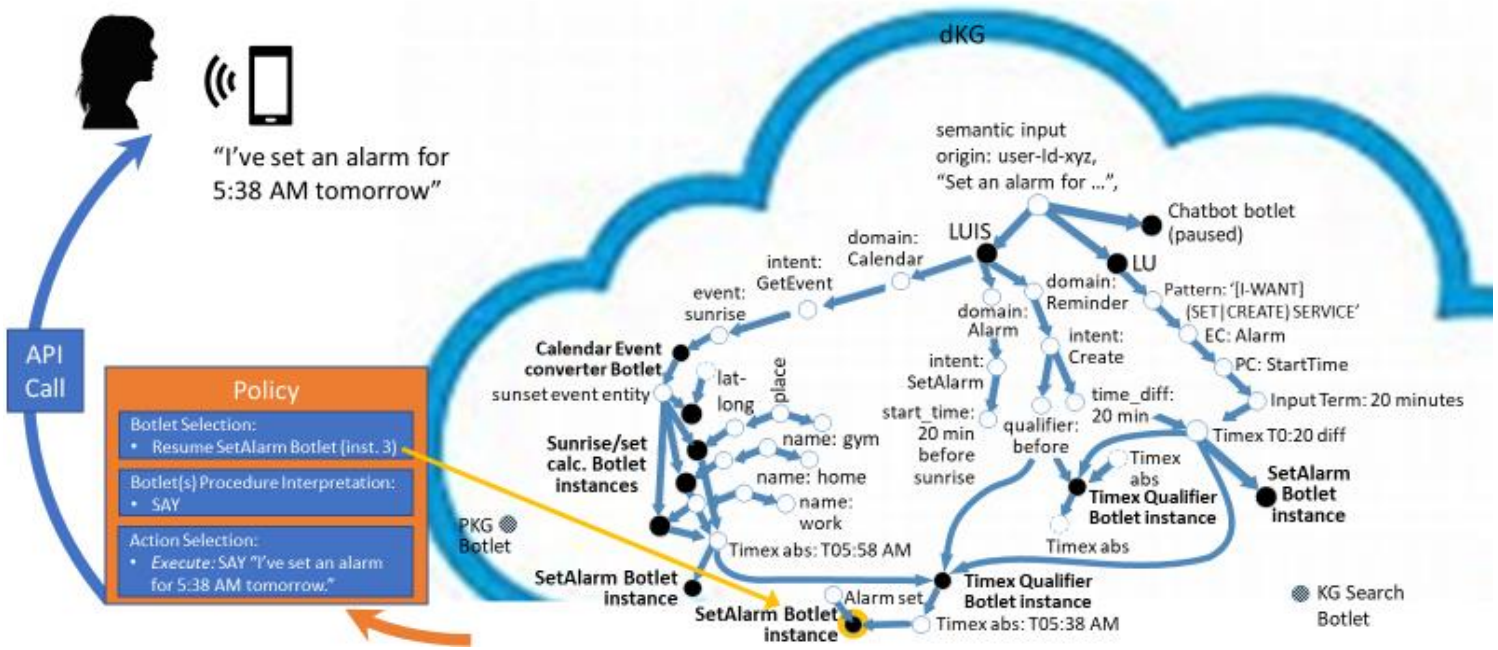
Running Example

- Step N: dKG gets large…
  - We ran calculator botlet
    - Compute T-20…
      - 3 possible interpretations of T
      - So calculator botlet ran 3 times
    - Get back 3 possible Time entities
  - State tracker: add to dKG
    - And make connections to existing entities/botlets (intelligently)
  - Policy: decide what to run next…
- Note: this is the k^{th} run through loop



Running Example

- Step N+M: the end!
  - Policy selects (correct) SetAlarm botlet to run
  - Botlet runs
    - Alarm is set
    - Message is displayed to the user
- Loop stops once system runs NLG+TTS



Training Inference DMs

- Much more complex decision process
- RL: can model state tracker, policy, and even KG search as subgoals
  - Train end-to-end using user simulator
- This prototype system: trained “supervised”
  - Various possible execution plans generated ahead of time from data
  - Only train part of policy (rest hard-coded rules)
- Training at scale: still unsolved problem

Thank you!

Indexing Process

- For each document in the crawled data:
  - Parse document using custom lexer (strip HTML, etc.)
  - Map each word to wordID (using lexicon hashtable)
  - Create hit lists for each word in the document
  - Generate forward barrels & add to forward index
- Generate inverted index:
  - Sort forward barrels by word IDs
  - Separate inverted barrels for title & anchor hits vs. full text
- Lots of parallelization, efficiency tricks

Query Processing (Core Search) Algorithm

1. Parse the query (including stemming, stopwording…)
2. Convert words to wordIDs
3. Seek to start of doclist in the short barrel for each word
4. Scan through doclists until doc found to match all words
5. Compute rank for document (see later how)
6. If in short barrel + end of any doclist, start searching in full barrels
7. Repeat from step 4 (until at least both barrels for one word are done)
8. Sort documents by rank & return top K results.

Note: this is essentially merge part of mergesort + resort by rank

Document Ranking + PageRank

- Different information feeds into ranking system:
  - Type of hit for the word (title, anchor text, body, etc.)
  - Word proximity – how close are words in multi-word query to each other?
    - Different proximity bins for each type of hit and each pair of words
  - TF-IDF scores (Google paper doesn't explicitly mention this)
- Combine all scores & generate IR similarity (cosine similarity)
- PageRank: essentially model of user behavior
  - Weigh links to page by count & reliability of each link
  - More links from diverse pages are good
  - Links from highly-ranked pages are also good