MITRE

MTR220588
MITRE TECHNICAL REPORT

Knowledge Base Population

An Orchestrated Path from Content to Insight

Authors:
Ransom K. Winder
Joseph P. Jubinski
Samuel L. Bayer
Nathan L. Giles
Merwyn G. Taylor
Jason D. Duncan

2024

This technical data deliverable was developed using contract funds under Basic Contract No. W56KGU-18-D-0004.

Annapolis Junction, MD

Table of Contents

Knowledge Base Population: An Orchestrated Path from Content to Insight

Abstract

Unstructured content, especially text, contains important facts about entities of interest, but given the infeasible prospect of manually reviewing all available content for these facts, a viable solution is to employ automatic extraction to a knowledge base. This process involves recognizing, collecting, and adapting the important content into a machine-readable format, which, in turn, can be queried and explored to answer questions about entities, their properties and relationships, and what inferences can be drawn from them. Applying such techniques introduces certain challenges of organization, consistency, flexibility, and extensibility to new technologies or different domains.

The Knowledge Base Population (KBP) work addresses this through a top-down system design able to integrate established or cutting-edge tools and technologies to support each of its stages of workflow, while aiming at developing a repeatable approach and reusable architectural design that will serve many different domains. This initial work suggests a core workflow supporting the most fundamental activities and lays out potential extensions once the baseline has been engineered.

Acknowledgements

The authors would like to acknowledge the considerable insight and guidance provided by our peer reviewers, Stacey Bailey, Michael Smith, and Chuck Howell.

Introduction

Making informed decisions and discovering important facts are perennially important activities. Moreover, many organizations have access to more content than ever before. The downside to this excess of documents, images, and other files is that there are far too many of them than can be read manually. This limitation poses the problem of how do we examine these sources to learn what facts they contain and also persist them in a manner that makes them readily accessible?

A viable and powerful solution is to engage in automated reading of the content, promising the ability to scale to a rate far beyond a purely manual process and also creating output that can be readily represented in a machine-interpretable format. Analytics underpinned by artificial intelligence (AI) and often specifically machine learning (ML) can be applied to different stages of analysis to convert unstructured content into the valuable knowledge it contains. The structured and aggregate knowledge that results is also conducive to machine and manual analysis that would reveal connections and realizations that might never emerge otherwise when document sources are looked at independently or ignored because of their enormous size. Yet using such an approach introduces new problems.

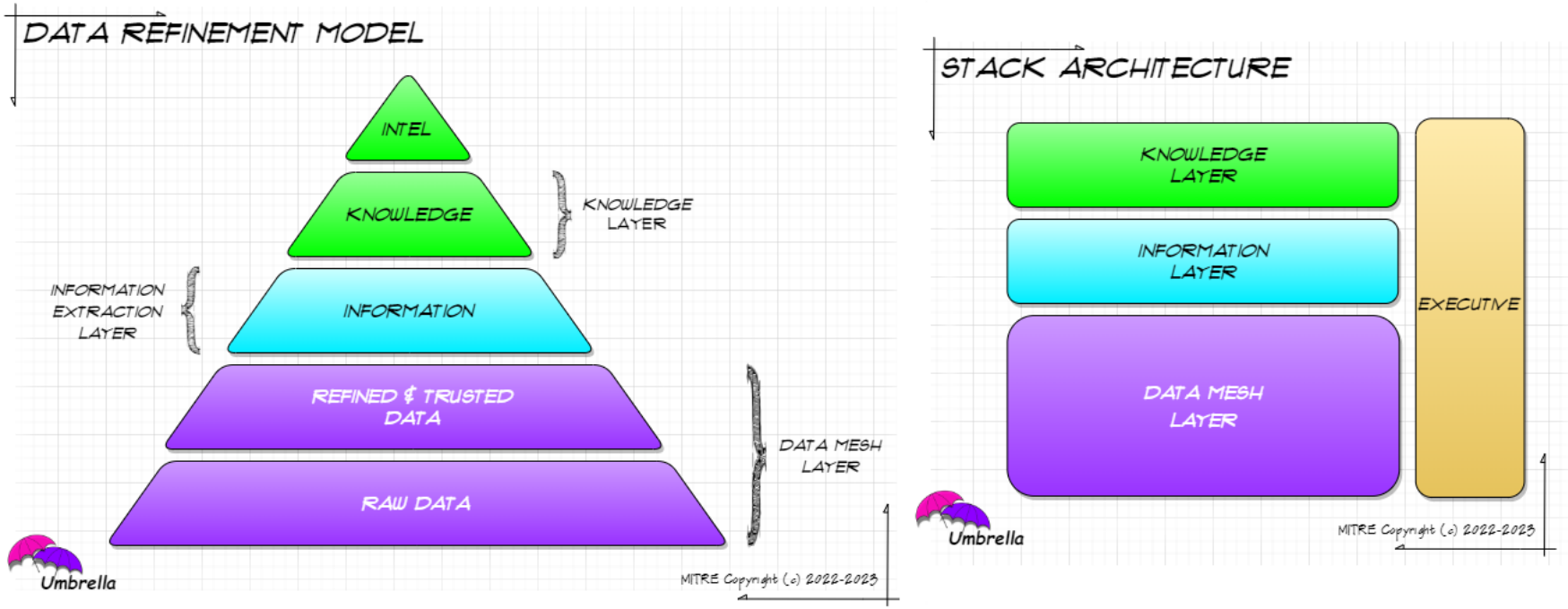

Figure 1: Data Refinement Model and Stack Architecture

There are several challenges that should be solved for this type of knowledge extraction system.

First, such a system tends to be complex, and therefore an organized solution is required. Second, while analytics are powerful, they are also brittle and subject to certain known shortcomings, which levies assurance requirements on the system and how it will perform. Third, because these analytics can be overtaken by new state-of-the-art techniques, a technology refresh will be required regularly. Thus, an intuitive and flexible solution is needed that can handle different use case questions, support ease-of-integration plug-and-play analytics, and exhibit a low barrier to use by domain experts. To the previous point, some consistency of representation is required to allow for stability of solutions during each refresh. Fourth, the domain where such a system can be employed may vary or change over time, which suggests a generalizable approach is required for a successful solution. Finally, the content and knowledge in the content may be denser and richer than either analytics or a knowledge representation can capture, so there is value in the solution also efficiently encoding meaning of the documents in a way that can aid in downstream discovery. Further, the system should be extensible to support new supplemental workflows. This document describes a knowledge base population approach that will address all the problems listed above with the solutions suggested.

System Architecture Overview

The creation of intelligence from data through automated or semi-automated means assumes a considerable underlying amount of raw data that undergoes refinement and extraction toward information, knowledge, and ultimately intelligence, as depicted in Figure 1, which illustrates this process as both a model of data refinement layers (left) and a stack architecture diagram (right). For either perspective, the Data Mesh Layer is responsible for storing, indexing, and ultimately refining the data. The Information Layer identifies the information held in this mix of structured, semi-structured, and unstructured content. This information is used to inform the creation of facts that are represented in a knowledge base in the Knowledge Layer, which is in turn accessed to inform intelligence. The Executive is the sequence of code that orchestrates the workflow from the query against the data mesh all the way to the decision processes of what candidates are offered for delivery to downstream knowledge repositories.

The proposed high-level system architecture for the KBP effort is depicted in Figure 2, which represents an orchestrated ensemble of multiple, independent, potentially individually developed systems working together toward a larger goal, or a “system of systems.” This system draws on a sizeable data lake of largely unstructured content, its files principally intended for human consumption (e.g., text, video, images). While there may be useful structured content as well, such as previously refined data from trusted sources, the unstructured content[1] poses a greater challenge given that it is beyond the scope of manual assessment and requires automated information extraction to capture the knowledge it contains relevant to a specific use case. This content-to-knowledge activity suggests five marked stages that progress across three layers.

[1] The unstructured content may include embedded structured or semi-structured content, such as tables appearing between paragraphs.

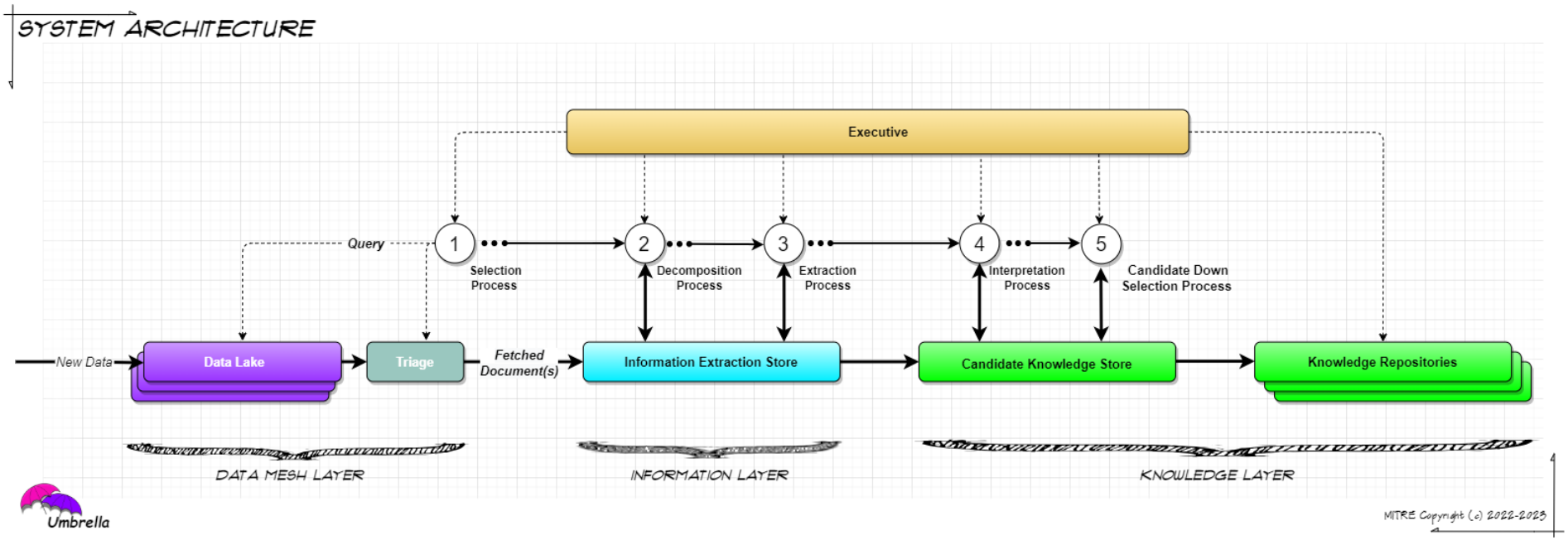

Figure 2: System Architecture

The first stage is the **Selection Process** (1), which draws on the unstructured content held in the Data Mesh Layer. A subset of the data lakes' content is triaged and chosen to be ingested into the Information Layer, possibly after being identified as relevant either through an examination of its sources, metadata, or indices.

The second stage, the **Decomposition Process** (2), will reduce complex documents ingested in the Information Layer into constituent parts that can undergo extraction (e.g., pulling the plain text content from a PDF, zoning) if needed.

The third stage, the **Extraction Process** (3), is devoted to identifying and structuring the important facts found in the content to populate the Information Layer. These facts can be mentions of entities of interest (co-referenced as appropriate), relationships among the entities, the events that represent the wider contexts, and other elements like times that bound when statements hold or topics relevant to a specific domain or use case. Tools in this extraction process are likely to rely on AI/ML solutions expressly trained to discover different kinds of knowledge in unstructured or semi-structured content.

The fourth stage ensures the extracted content from the Information Layer is properly added to the Candidate Knowledge Store in the Knowledge Layer, which holds the accumulated candidate knowledge. In this **Interpretation Process** (4), extracted results are adapted to fit the meaning and organization of a consistent model that defines the candidate knowledge store, the entities that appear in the results are linked to entities already in the store if such determinations can be made, and the metadata that describes these results (e.g., provenance, uncertainty) is preserved. The primary operating assumption of this stage is that the content would arrive only from the Information Layer, even though it is feasible that knowledge from downstream repositories might also be passed into the Candidate Knowledge Store.

The last stage is the **Candidate Down Selection Process** (5). This can occur asynchronously, taking elements from the candidate knowledge and advancing these to storage in downstream knowledge repositories once the elements become available after the Interpretation Process. These repositories may contain different subsets of overall knowledge or different representational structures. These downstream knowledge repositories are grounded in specific missions, so this general workflow cannot consistently articulate how they ingest the knowledge sent to them.

Data Mesh Layer

The KBP workflow will first operate on the Data Mesh Layer as called out in Figure 3. At a high level, the data lakes, of which there may be more than one, are central to this activity. New data will periodically be added and indexed in them, and the downstream processes will query them to produce relevant document subsets to serve as input to downstream knowledge generation processes. *Data triage* is this activity of reducing the size of data down to a relevant set to process through the entire system.

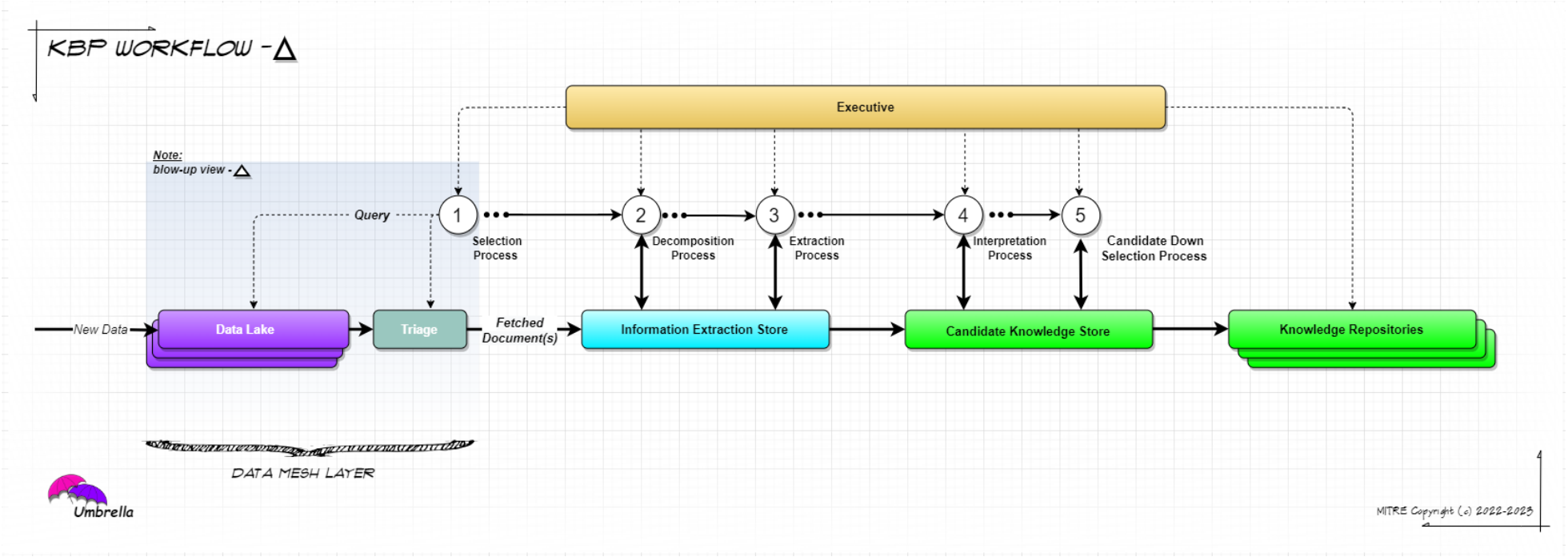

Figure 3: KBP Workflow ▲, the Data Mesh Layer

Figure 4 depicts a more detailed view of the processes that can leverage the data lake(s) to select documents to send downstream. It illustrates how the data lakes would be periodically refreshed with new data that is ingested (A), normalized (B), and indexed (C) for storage in a given data lake. The Ingest Process will bring the data in from any number of other sources. The Normalize Process will ensure that *if* the data needs to be in the same format, all data moving forward will be given that standard format; this is not always necessary. The Index Process will automatically examine the metadata, header data, and/or the content of the data to find searchable tags.[2] The Selection Process (1) can query the data lakes through a Fetch Process (G) to gather subsets of documents to submit to instances of the downstream workflow that will convert their content into knowledge. The basis for this activity can arise from analyst queries, queries submitted on behalf of a mission, or automated housekeeping that refreshes the knowledge base.

As a supplement to the data lakes, where data is retrievable by indices and metadata, a vector store provides a different query capability. Ingesting the same content indexed for the data lakes in the Ingest Process (D), the vector store (e.g., Weaviate[3]) will convert the data's content into vector embeddings in the Vectorization Process (E) (e.g., for text this could be through a process like text2vec[4]). In the Query Process (F), unstructured queries of differing size and specificity (e.g., text, images) can then be converted to vectors and used to find data that occupies a close position in that same high-dimensional vector space, where proximity of points in that space is indicative of semantic similarity between the content. This provides an alternative way to discover relevant data entries based on the similarity of the content without exclusively relying on knowing the correct indices, metadata, or keywords.

In some cases, a cold start of the system may be necessary. A cold start [1, 2, 3] is when there is not yet any content populating the knowledge base. A more detailed discussion of the cold start process appears later in this document. The initial seed for the cold start can emerge through the process described above and be selected by a variety of means. A baseline method for choosing what informs the cold start can be finding relevant documents in the data lakes based on indices. Other alternatives include selecting all documents received between certain dates or selecting all documents timestamped with certain dates.

Drawing on the vector store provides another means to cold start the knowledge base. In this technique, a summary or textual definition of what is wanted in the knowledge base can be the prompt that elicits similar documents from the searchable vector store. These queries (F) can be performed as a question-and-answer activity against the vector store or using text descriptions drawn from taxonomic classes as a search query that finds semantically similar data instances to populate an initial knowledge base centered around specific entities of interest. If the vector store contains incomplete text for a document (e.g., an abstract), the full document can be retrieved from the data lakes.

---

[2] An alternate information extraction process may also occur up-front (C') if the workflow requires all documents undergo information extraction. This aligns to the processes described in the Information Layer section.
[3] https://weaviate.io/, however many other companies provide a similar capability such as Pinecone (https://www.pinecone.io/) and qdrant (https://qdrant.tech/).
[4] Such as is used in Weaviate, although there are a host of "*_2vec" alternatives.

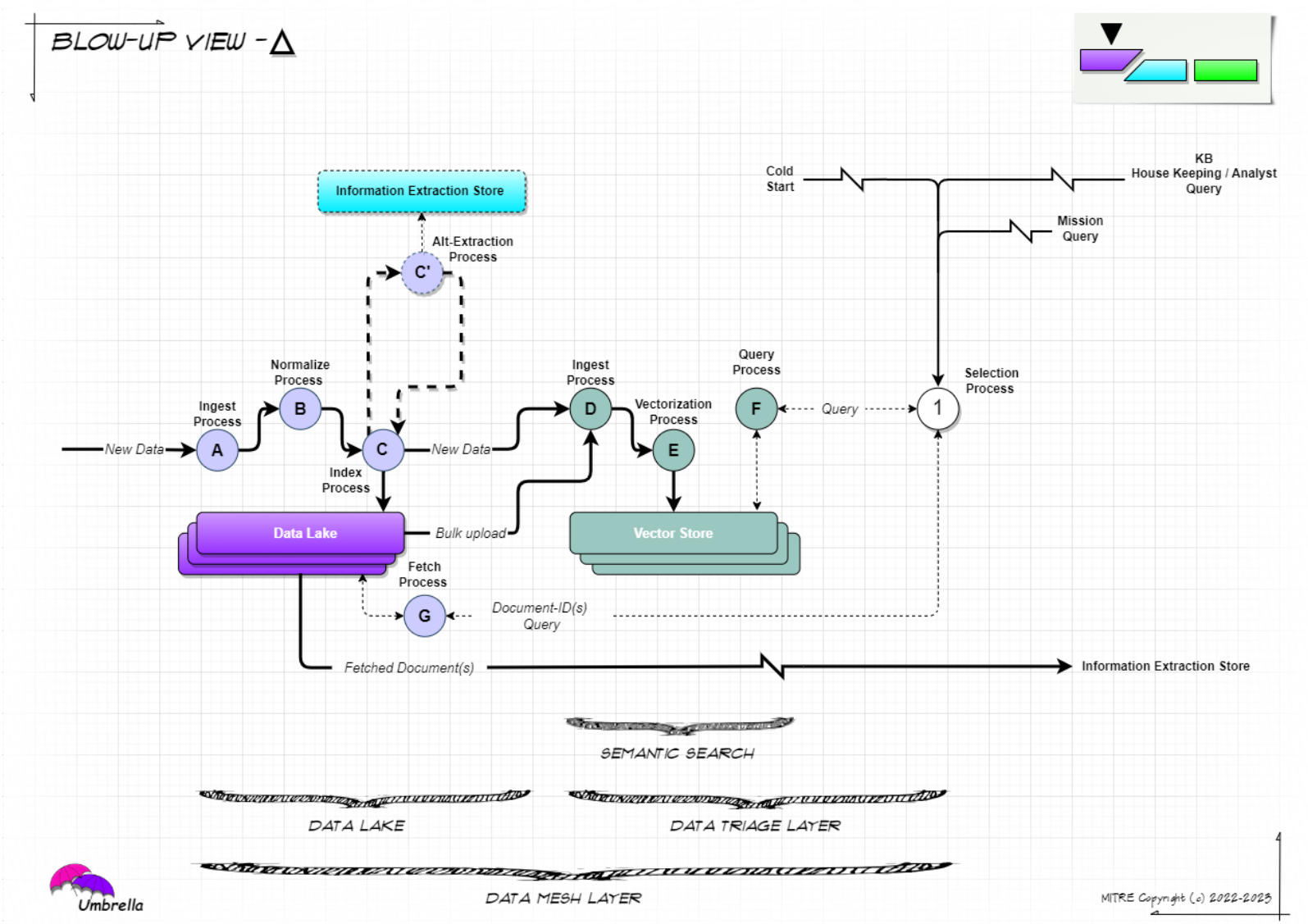

Figure 4: KBP Workflow ▲, Blow-up

Information Layer

Documents and data fetched by the Selection Process (1) next undergo processing in the Information Layer as highlighted in Figure 5. The center of activity is the Information Extraction Store, which acts as a repository for both the subset of documents from the Selection Process and interim results of important content in that data rendered in a structured format. Within this layer, the main activities can be coarsely characterized as a Decomposition Process (2) and an Extraction Process (3).

Decomposition Process

The Decomposition Process always follows the Selection Process. It ensures that content that is in a complex state is transformed to a form that conforms to the input expectations of the downstream Extraction Process. For example, the incoming data might be in a Microsoft PowerPoint file with unstructured text, charts, images, or embedded video or audio, which might be suitable for extraction using tools like Apache Tika [4]. Or it could be images of documents that require text be recognized optically, which could be accomplished with software like Tesseract [5].

The Decomposition Process ensures the plain text is made available for downstream analysis to extract the relevant entities, features, and relationships. In most current natural language processing toolchains that are fed by upstream complex documents (e.g., PDF, PowerPoint), this Decomposition Process is relatively simple: it produces a single text document stream, without respecting the structured relationships between the different text streams within the source document (e.g., footers, image captions). Such a process is inadequate, but addressing its shortcomings is likely an architecture task of its own, requiring crucial changes both to 1) the interaction between the Decomposition Process and its downstream steps and 2) the core operation of the individual components within the downstream Extraction Process. This effort is properly outside the scope of the current project, and accordingly, this architecture assumes this simple Decomposition Process for the time being.

Document zoning is also part of the Decomposition Process. This activity classifies sequences of text within a document into zones fitting certain categories and can be used to establish different portions of the document that distinguish content or types of content from metadata. This may influence which downstream analysis is applied against zones of text based on their category.

In general, the workflow described below assumes that plain text is the output of this process that continues forward. This is to simplify the knowledge base population activity to accommodate the most frequent source of knowledge (that is, what is rendered in human-readable textual language), but it does not preclude additional activities that work natively against other unstructured content such as images, figures, or video. Some additional restrictions for early implementations of this pipeline are described and justified below.

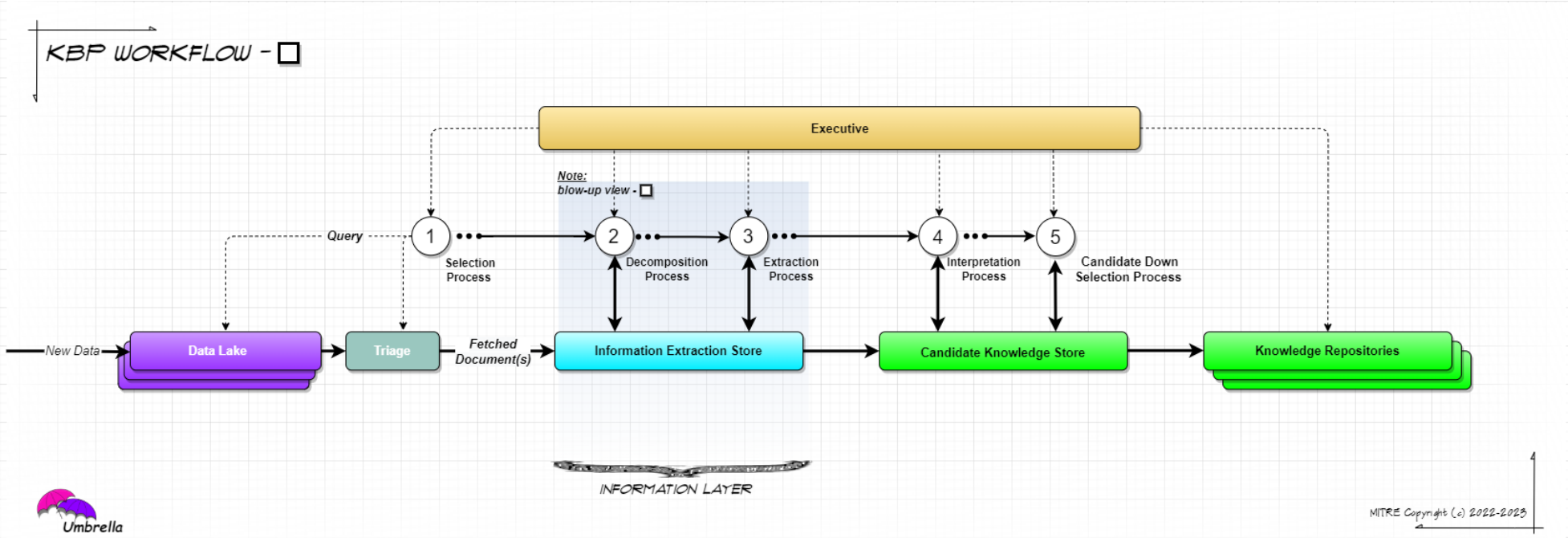

Figure 5: KBP Workflow ■, the Information Layer

Extraction Process: Text

The Extraction Process will frequently follow the Decomposition Process, although the specific activities it performs will vary with the input and the use cases driving the extraction. Its purpose is to find and isolate facts that are useful in the content, structuring them so they can be machine readable and suitable for becoming a part of the knowledge base. Examples of workflow through the Extraction Process are expanded in a set of blow-up figures depicted later. This section also provides a theory representing what is extracted from the text. This representation is distinct from the knowledge representation that occurs downstream, and it is instead intended to provide a flexible and analytic-agnostic means to preserve what is produced and consumed by extraction analytics.

In this section, we specifically consider extraction against textual content. This invites some restrictions on the scope, principally that the Decomposition Process has ensured that plain text or some semi-structured formats (e.g., free text fields in CSV files) are provided to the Extraction Process.[5] It also suggests a few architectural expectations, which include the following:

- The Extraction Process is not monolithic but instead consists of many interdependent analytic components. These cover activities such as entity mention extraction, time extraction, coreference resolution (within document), relation extraction, and event extraction. While these could all be folded into a single solution, the expectation is that arbitrary components from different sources should be allowed to interoperate to ensure that the best-of-breed choices can be made for any given step. This is especially important as the pipeline dependency means that errors in one stage will propagate forward and lead to larger errors in later stages.
- The executive component of the system has direct control over these components as opposed to treating them as a black box. This is because such components exhibit substantial cost differences, suggesting considerable value in allowing for load-balancing across the components.
- Because of the eclectic nature of analytic components, to ensure they have a consistent interaction, the architecture assumes the use of adapters, or wrappers for information extraction components that mediate what they provide the executive and submit to the Information Extraction Store. Adapters should be as simple as possible.
- The Extraction Process is not expected to access the downstream knowledge repository. This is an assumption to aim for a greater simplicity of design and computation, but it is not a requirement. That said, any design that uses the knowledge base as an input to its decisions should have a strong justification for incurring the added complexity.
- Components are applied to input in sequence, not in parallel. As with the above, this is a simplifying assumption. Any parallelization should happen at the document level.
- Documents may be reprocessed by later workflow instances, especially as capabilities evolve or improve.

[5] Early implementations also assumed a focus on English as the only represented language, although what is described in this section is extensible to foreign language and the downstream processes are language agnostic.

- Not everything that a given component produces in the Extraction Process will advance to the knowledge base. Some information may not be relevant to the knowledge base and only assists components in this process that have a dependency on it. Some information may not be possible to resolve against the knowledge base's model. In principle, we aim to minimize the amount of standardization to what is domain-independent, language-independent, and necessary to retain. Nevertheless, a considerable measure of standardization is expected on these results to ensure that a common scheme and semantics reaches the knowledge base. This means that adapters for components will output:
    - An opaque serialization of the component's raw output, which is to say what the component produces if operating independent of the architecture.
    - A standardized representation that the adapter has created or augmented from the raw analytic output.

With these expectations in mind, the Extraction Process leverages a theory put forth in this document of representing what is extracted from text, including coverage of entities and events, relations and features, and metadata (e.g., provenance), in the same generic structure to allow disparate analytics to contribute to the Information Extraction Store. This theory can be expressed in five distinct elements: 1) a set of instances, 2) a set of equivalence classes, 3) a set of groups, 4) a set of event relations, and 5) a log. Figure 6 illustrates an example of the first of these elements, the instances.

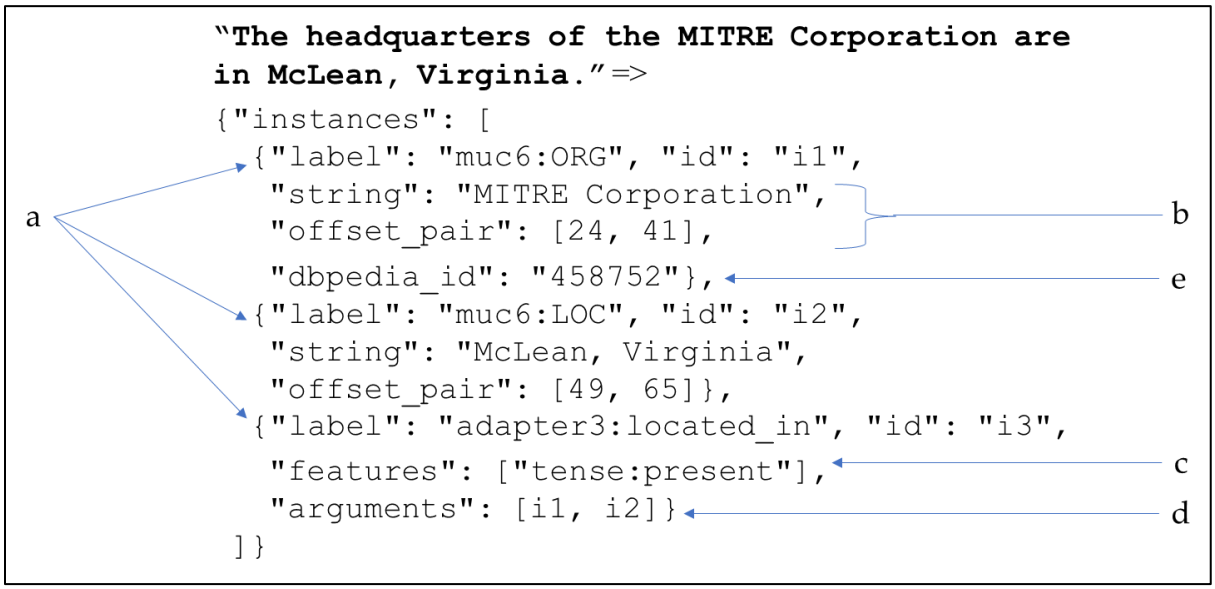

Figure 6: Example Instances Conforming to Extraction Process Theory

Instances are distinct things that are asserted to exist and have the potential for an identified temporal component. Importantly, this theory means that instances will include entities, events, *and* relations, all given a shared structure for consistency in the Extraction Process, which must be able to accommodate results from a variety of extraction software.

An instance (as depicted in Figure 6) will have a unique ID (obligatory) and optionally contain fields for

a) a label,
b) a name,
c) a set of features,
d) a sequence of (optionally named) arguments, and
e) a normalization.

The label (a) most closely maps to the idea of a semantic, taxonomic type, essentially describing what this instance "is," but it is not obligated to align to an ontology used downstream in the knowledge base. The label should however be a namespaced token, composed of a namespace (e.g., the adapter that produced it, the annotation guideline it fits) and an arbitrary string. Notably, because of the label flexibility, some labels may not map to semantic types and will not become part of the downstream knowledge. The name (b) consists of the string and offsets (that is, Unicode character indices where the string starts and ends) drawn from the text source. Morphological features (c) are of the sort that might matter where a given instance is an interim result to be processed further; these can distinguish proper names, nominal phrases, and pronouns but also could include number, gender, person, tense, polarity, etc. The arguments (d) are of chief relevance in relations or events captured in instances and connect to other instances' IDs, which creates a consistent and robust way to reference any other instance. The normalizations (e) are namespaced fields that contain any JSON-serializable object and can hold other interpretations of an instance (e.g., TIMEX2 encodings, DBPedia IDs); there is no restricted list of what normalizations could be used.

The **equivalence classes** are a set of unique IDs for instances that are asserted to refer to the same instance (i.e., be equivalent to one another). This capability is in service of within-document coreference and aligning text mentions found by two different adapters in the same document.

The **groups** (shown in Figure 7) contain a set of members (also indicated by unique IDs that refer to instances), which can accommodate explicit plurals where the members of the group are known but may be referred to collectively in other fields.

"Joe Biden and Donald Trump debated." =>

```
{"instances": [
  {"label": "muc6:PER", "id": "i1",
   "string": "Joe Biden", ...},
  {"label": "muc6:PER", "id": "i2",
   "string": "Donald Trump", ...},
  {"label": "adapter2:debate",
   "features": ["tense:past"],
   "string": "debated",
   "arguments": ["i3"]}
 ],
 "groups": [
  {"id": "i3", "members": ["i1", "i2"]}
 ]}
```

Figure 7: Group example

The set of **event relations** encompasses a small number of relations intended to bridge events and times, but which themselves do not need to be reified. This includes such timestamping connections that covers concepts such as *before*, *after*, *holds within*, *holds throughout*, *overlaps*, *starts at*, or *ends at*. Event relation consist of a relation name (a namespaced token that might semantically map to one of the example concepts listed) and a sequence of arguments (containing the unique IDs of other instances). What they lack is an ID of their own, as these do not require reference in the theory and do not need to be reified as other relationships are.

The **log** covers both a list of the relevant events that have gone into the creation of the theory and a store that captures the component uncertainty that arises in the creation. The list covers the provenance of IE components executed and the creation and adjustments of each instance, covering each of the elements that appear in the theory. The uncertainty expressed in the log arises from the uncertainty expressed by the analytics that produced a given instance or other element and nothing else. That distinction is made because the landscape of what amounts to "uncertainty" can be quite broad (uncertainty arising from errors in extraction, linguistic doubt and vagueness, source reliability, etc.), but distinguishing and accounting for these is carried out in the knowledge layer.

A selection of typical processes that will produce the elements of the theory are depicted in Figures 8 through 11. These examples represent common workflows that the system will undergo, and they are not mutually exclusive. A given document may experience more than one such workflow. They also indicate what must happen for a given workflow, even if they do not exhaustively show everything that can happen.

In a typical **relation extraction** workflow (Figure 8), after a document is decomposed (H) and its text is isolated, entity mentions (i.e., the extents of text that represent specific entities that would be assigned a label) are extracted (I) as a set of instances. These are in turn coreferenced (J) so that textual extents that represent the same entity are indicated as such in the equivalence classes. Following this, relations for the entities are extracted (K), capturing labeled connections with arguments aligned to text extent IDs that represent the entities participating in the relationship.

This relationship, in turn, becomes an instance. At this point, the results of each process are retained in the Information Extraction Store. This specific workflow does not preclude other methods of extracting relations, but serves as a commonly used example.

In a typical **domain entity** workflow (Figure 9), the document decomposition process (H) precedes an extraction process that identifies textual extents directly related to a specific domain of interest (L). These are more likely to be topics or concepts rather than entities that are likely to be coreferenced, but they will become instances included in the Information Extraction Store.

In a typical **event** workflow (Figure 10), the processing captures complex instances that represent events, which have participating entities and may be timestamped. After decomposition (H), a named entity (M) and coreference (N) activity will find and establish equivalences between textual mentions. Alongside this, textual extents representing time are extracted (P) from the documents and represented as instances, and resolved to a global timeline (Q) to reduce the ambiguity and allow for understanding temporal ordering. Both processes will be used as input to the event recognition process (O), which indicates what an event is by its label, what entities participate by its arguments, and how it is properly timestamped by event relations. Each of these instances is preserved in the Information Extraction Store.

In a **bypass** workflow (Figure 11), all normal analytic processes that produce results are bypassed. This happens when the ingest already includes the kind of structured information that analytics would extract from raw content to put in the Information Extraction Store. This does not eliminate the need to translate the content so that it conforms to the consistent theory of how results should be represented in the Information Extraction Store; this simply does not involve further analytic work.

Each of these workflows operates in service of the possibility that their results may be sent downstream to the Candidate Knowledge Store.

Before being sent to the Candidate Knowledge Store, each workflow will have some measure of interaction with the Information Extraction Store. This store ensures that many different workflows can be composed without requiring a direct dependency on the previous analytic. It holds documents and their interim results, where each of these instances consist of a unique ID, a Unicode character sequence representing the document's content, and an ordered sequence of information extraction processing records, which is to say a record of the application of adapters (i.e., information extraction components wrapped to produce content conforming to the theory described). In turn, each information extraction processing record contains the name of the adapter, a timestamp for the adapter's completed processing, the current state of the information extraction process theory derived from the document, an optional information extraction component cache (essentially, the raw output of the component wrapped by the adapter, which is not intended to be interpretable by our system), and a timestamped list of all adapters previously executed on the document.

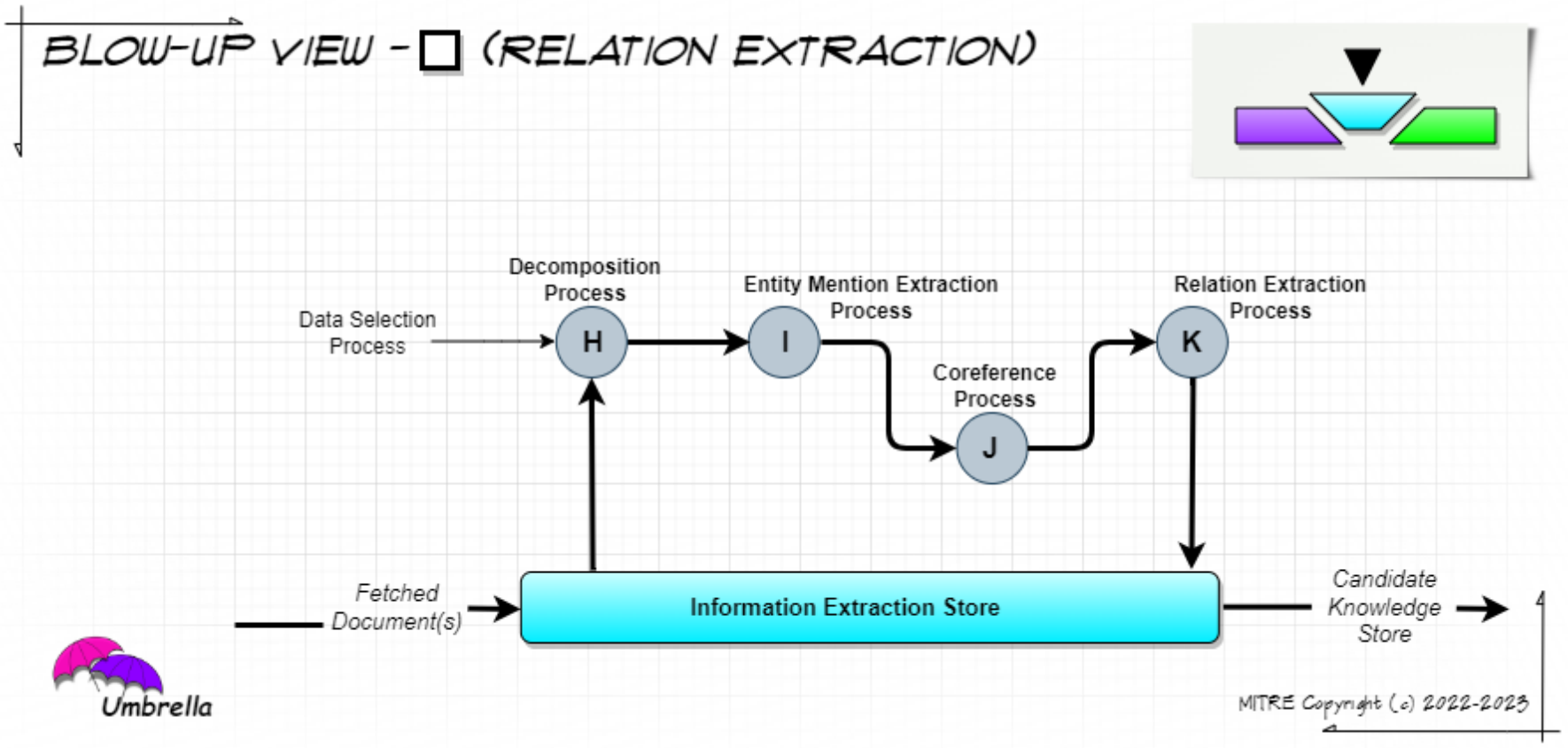

Figure 8: Relation Extraction Workflow

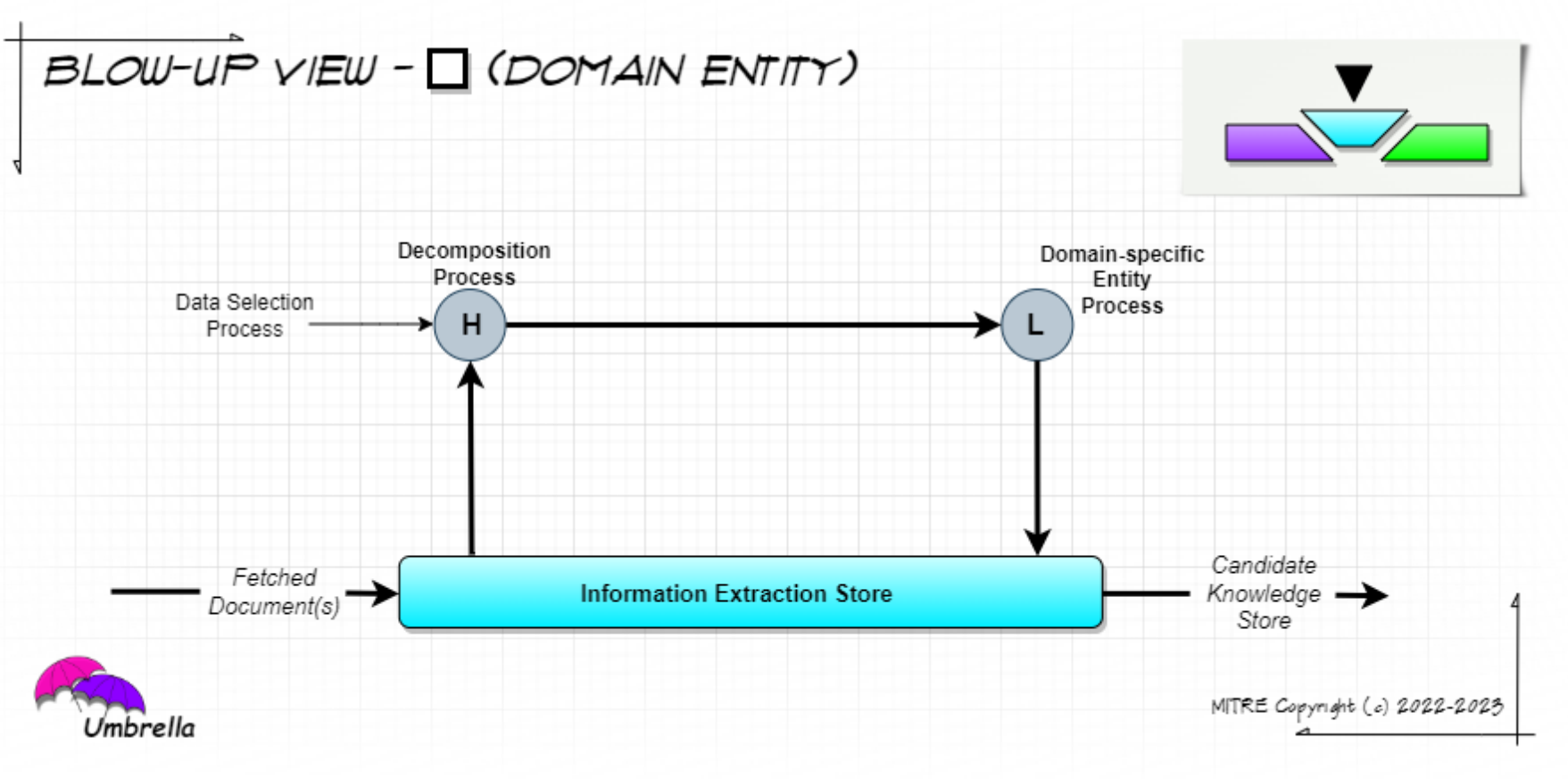

Figure 9: Domain Entity Workflow

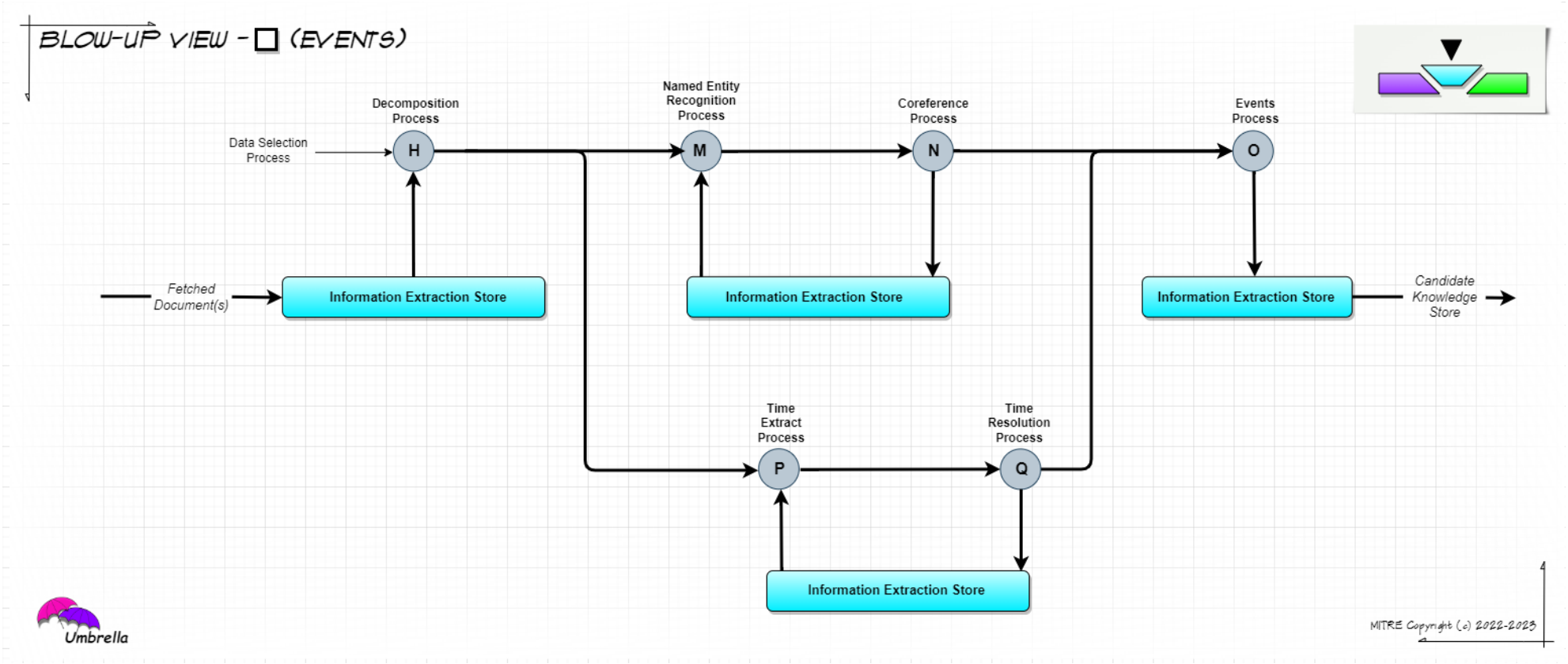

Figure 10: Event Workflow

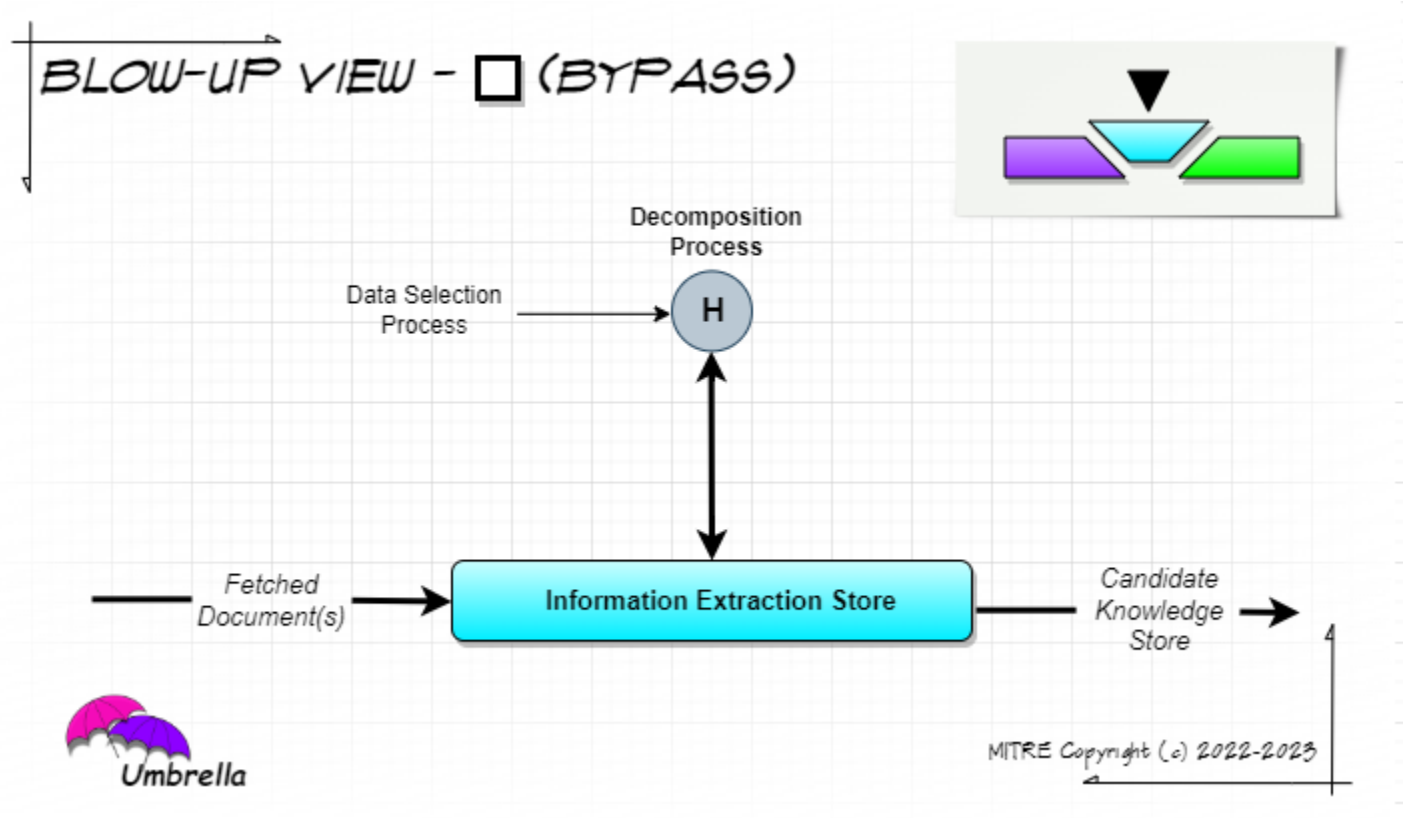

Figure 11: Bypass Workflow

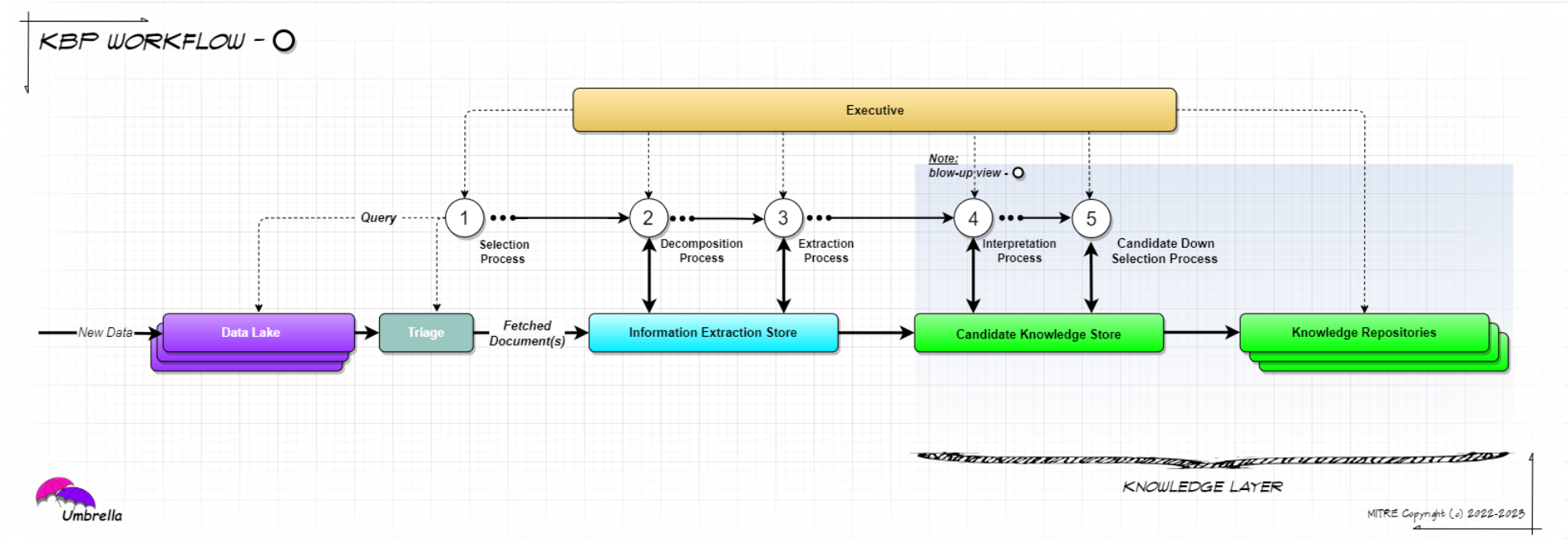

Figure 12: KBP Workflow ●, the Knowledge Layer

Extraction Process: Other Media

What has been described in the context of principally text-based documents can be expanded to include other media (e.g., images, audio, video). There is also the potential to convert some of these to text or extract plain text from them which can go through the information extraction pipeline for text. However, the emphasis of this pipeline in its current form is text, so these are excluded for simplicity, although they will likely appear and receive a similar treatment in future iterations of this document.

Knowledge Layer

After the workflow has passed through decomposition (2) and extraction (3) from unstructured content into structured information, it continues to the Knowledge Layer, as shown in Figure 12. This knowledge typically has a graph structure, meaning it consists of assertions that describe nodes and edges (representing entities and their relationships and properties). This allows for vast interconnections to develop between entities based on the sources while preserving the important descriptive semantics describing them and how they relate. Moreover, such knowledge graphs in this context rely on an ontology of consistent types and relationships, which are defined, and potentially rules for inference over them.

Interpretation Process

The first step in the Knowledge Layer is to convert this structured information into assertions that can be stored consistently in a knowledge base. This occurs in the Interpretation Process (4), which entails a) mapping, or ensuring the structure of what emerges from the interpretation conforms to a consistent representation and semantics, and b) entity resolution, or ensuring the entities retained after interpretation are linked to existing entities already present in the knowledge as appropriate. Incoming candidate knowledge will be held in the Candidate Knowledge Store but will undergo several stages of processing to reach a retained representation.

Recall that while the theory described to support the Extraction Process has a consistent language for representing extracted content, there is no assumption of semantic consistency between what any two analytics will produce. For example, analytic A might label people as “A:Human” while analytic B might label them as “B:Person” instead. However, once these entities are passed to the Knowledge Layer, such differences in labels need to be resolved to a common meaning that also aligns with the ontology of what the Knowledge Store uses to describe the universe. Therefore, namespaced labels in the Information Extraction Store instances must be mapped to the appropriate ontological classes defined in the Knowledge Layer. Any complex inferencing required for changing entity classes would be reserved to occur downstream. This would cover cases such as when an analytic has over- or underspecified a class or made an error that other evidence exposes.

Further, the flexible structure defined for extraction information from multiple analytics in the Extraction Process may not conform to how information is organized in the Knowledge Layer. For example, while the theory in the Information Layer reifies relationships as instances, these may be represented as object properties (i.e., a link connecting two entities) in the Knowledge Layer. Likewise, features with literals (e.g., strings, integers) may become data properties (i.e.,

links connecting an entity subject to a literal). The instances' normalizations (that is, the interpretations or encodings that enrich an instance with external reference information) will also need to be normalized to the ontology if they are to be preserved.

Note the Extraction Process is where the principal analysis work of the text occurs; the Interpretation Process (R), as seen in Figure 13, that maps the extraction results to the Knowledge Layer's ontology is simpler. Interpretation is based on a set of rules that are intended to minimize the loss of translating from one semantics to another, making the process thoroughly explainable. Importantly, it is not the case that everything in the Information Layer will advance to the Knowledge Layer. Rather, only what is relevant to a given use case downstream will go through, leaving some of what analytics may produce unconverted. This is reasonable for analytic results that are primarily intended as input to other analytics versus intended to produce entities and relationships to capture long-term.

As part of interpretation, entity resolution is undertaken to merge the entities. This assumes the common case where the Candidate Knowledge Store retains entities found in content processed earlier or already defined before processing. Some of these entities may represent the same individuals found elsewhere in the mapping process. Entity resolution recognizes when equivalent entities exist, merging the entities (and their properties) when they do. This entity resolution can be based on a variety of criteria including matching names and properties. It relies on reaching a sufficient threshold of match between the features describing extracted instances and existing entries. If a match is determined, features and evidence of the newly extracted knowledge are merged into the existing entry in the candidate knowledge. Otherwise, a new entry is created in the candidate knowledge for what was just extracted. During the entity resolution, it may be prudent to revisit certain assertions to refresh whether the decision to merge or keep distinct is still correct as more knowledge is created. It is not a trivial process or one that can be performed with a guarantee of no error.

Following the interpretation, a validation (S) step helps ensure that the structure produced by the interpretation matches the expected schema and does not create any violations where the entities or properties are misused.[6] Following this filtering (T) will follow criteria for removing certain assertions that should not propagate forward to the Candidate Knowledge Store (e.g., clearly duplicate statements).

Ontology

The Interpretation Process also assumes that the Knowledge Layer has a predesigned knowledge model, or ontology. This can be derived from or extended from an existing upper-level ontology, such as Basic Formal Ontology (BFO) [6] or the Suggested Upper Merged Ontology (SUMO) [7], possibly using mid-level extensions (such as CCO for BFO and MILO for SUMO), which are useful in helping to express the utility of the upper layer whose purpose can sometimes be mystifying when viewed in isolation. Further extensions can be made toward specific domains of interest (e.g., nuclear chemistry), specified as self-consistent domain-related hierarchies (i.e.,

[6] For example, in RDF, SHACL (https://en.wikipedia.org/wiki/SHACL#Validation) is a language that could enable such validation.

"microtheories") that are children of the upper and mid layers. With that in mind, the approach to the ontology supporting the Knowledge Layer seeks to achieve the following goals:

1. *Stable and consistent upper-level ontology.*

An ontology represents knowledge as a hierarchical set of possible primitive entities with their properties and relationships to other entities. At the upper level of the hierarchy, these entities and properties should be universal regardless of the application of the entire ontology. Key to achieving something this universal is that it be stable, which is to say it will remain fixed once established, and consistent, which is to say it does not exhibit any conflict between terms or their definitions. Using a well-vetted ontology that has proven stable as a reference is generally a best practice where possible.

2. *Capacity for microtheory extensions for any domain's use cases.*

The upper-level ontology is insufficient to cover even a small portion of how a full ontology will be applied. A microtheory represents a supplementary set of entities and properties that is self-consistent and covers a specific topic area, or domain. Moreover, it will be able to model any content or structure of knowledge that is needed to fit a use case, which is a set of questions applied against the knowledge model that it should be able to address. Finally, the microtheory should extend from the upper-level or mid-level ontology, namely the highest elements in a microtheory's hierarchy should be children of elements in the upper-level or mid-level ontology.

3. *Brisk comprehensibility of the ontology.*

A complete ontology that covers a domain's use cases presents a challenge in that it may exhibit a level of detail that creates a barrier to entry for using a knowledge model properly. Thus, aiming for a brisk comprehensibility of the ontology means that users should be able to add to and query it without devoting themselves to becoming as expert on the entire ontology as its designers. Documentation of the notable classes and properties in an ontology can indicate which are the worthiest of attention without artificially sacrificing the greater structure.

4. *Flexibility in defining or employing microtheories.*

While the upper-level ontology should be fixed, domain details and supported use cases may change with sufficient frequency that the microtheories need to be flexibly swapped, configured, and tailored, which is to say able to accommodate the domain or use cases changes. This flexibility can either be in how the microtheories are defined (i.e., the specific hierarchy of allowed entities, relations, and properties) or employed (i.e., swapping one microtheory for another with updates).

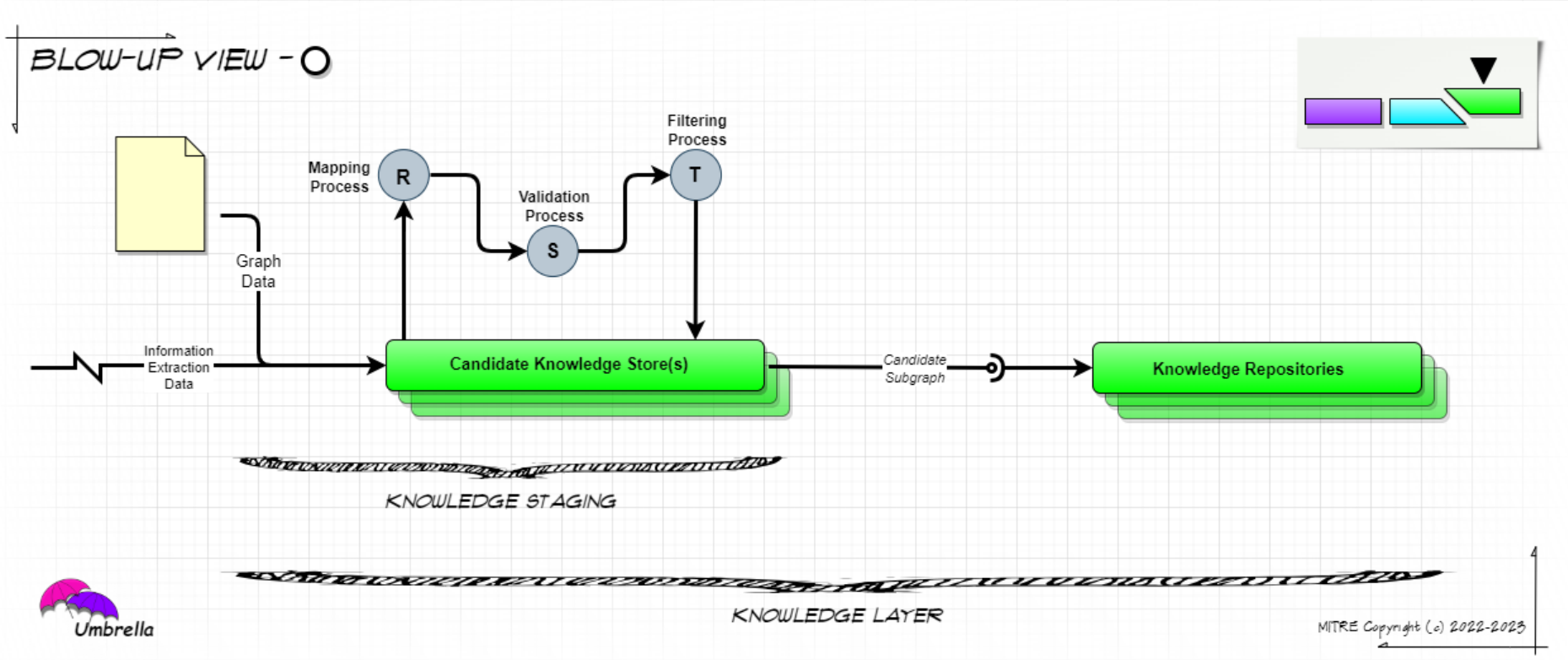

Figure 13: KBP Workflow ●, Blow-up

Candidate Down Selection Process

The Candidate Down Selection Process (as marked in Figure 12) is where statements contained in the Candidate Knowledge Store are reduced to a subset that will be sent to another knowledge store within the KBP system (described in the overview below) or to one or more Knowledge Repositories external to the KBP system. This can be initiated by an ad hoc request (e.g., a query for entities meeting certain criteria), or it can follow a regular schedule where certain entities by type or features are submitted to the store(s) to which they are relevant. This process also relies on an assessment of the quality of the statements to be submitted. Confidence is the general term we use here to indicate a measure of how much trust should be placed in an assertion being accurate, but the nature of assessing confidence is complex. It is a significant challenge to arrive at a consistent metric for confidence that may account for vagueness of the original content, overtly stated uncertainty in the original content, trustworthiness of the source, and analytic accuracy. Specifics of confidence are beyond the scope of this paper.

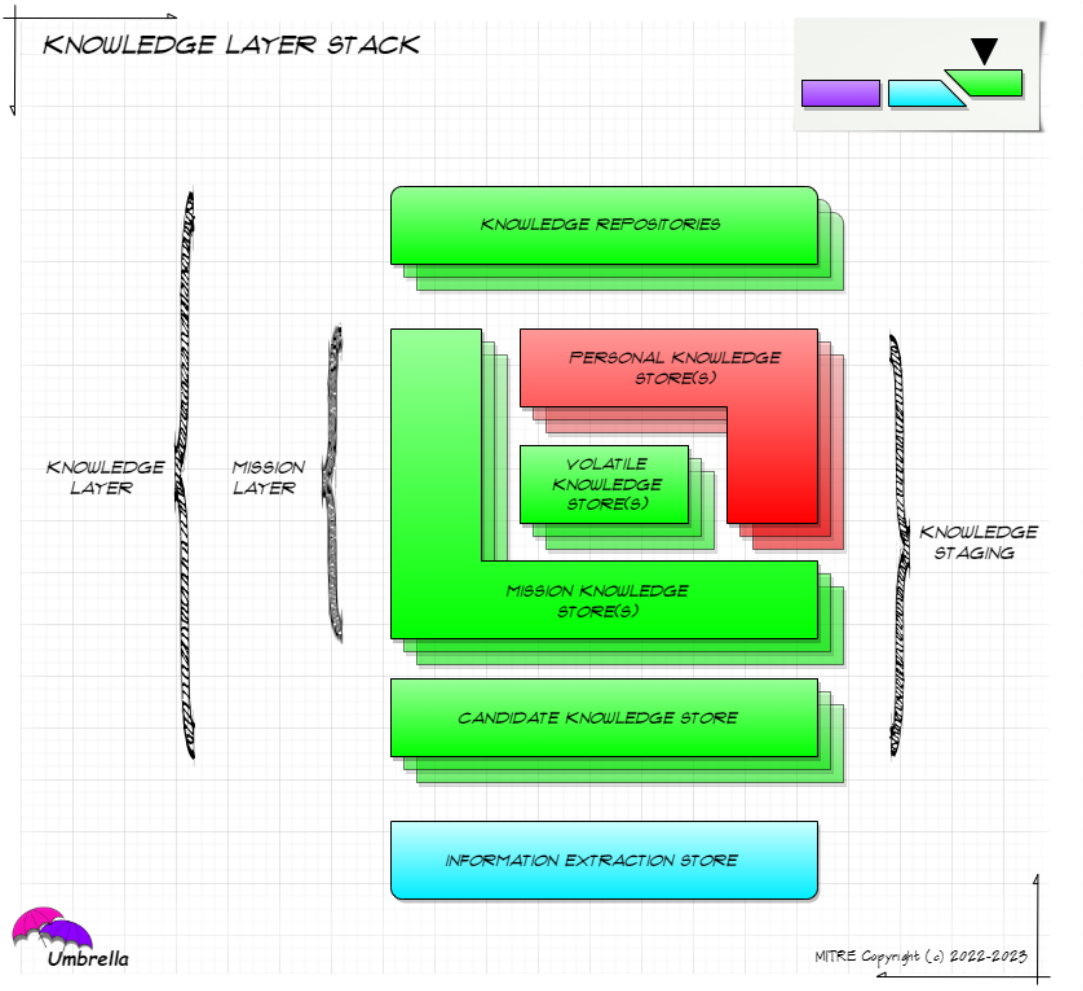

Figure 14: Knowledge Layer Stack

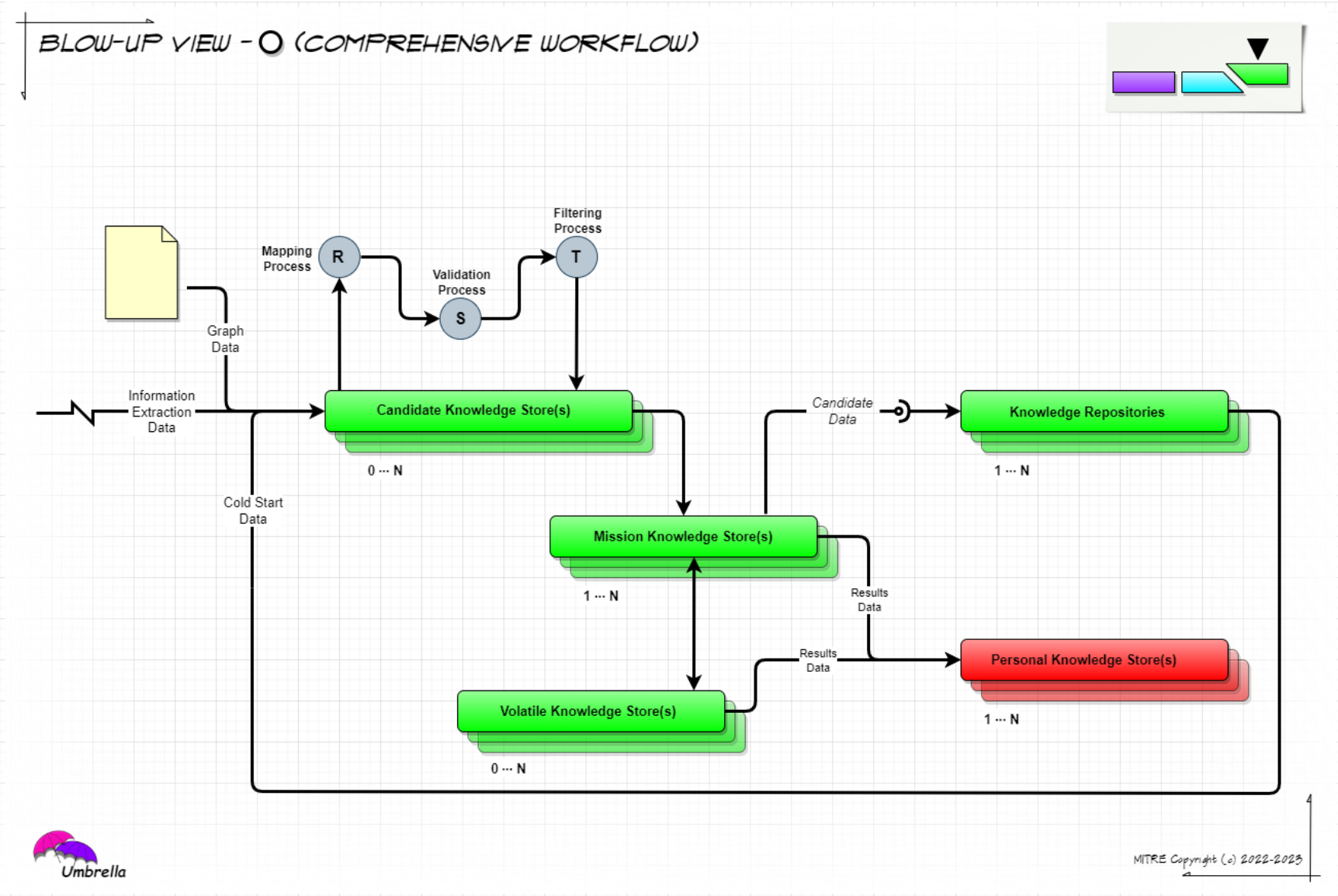

Figure 15: General Approach to a Comprehensive Workflow of Knowledge Stores

Overview of Knowledge Stores

Various types of knowledge stores can serve different needs downstream of the processes described above. These reflect the different ways such knowledge stores are used, and the requirements levied against them. This section delineates several notable knowledge store types, describing their distinctive features and how they fit into the pipeline. A stack diagram in Figure 14 shows the knowledge layer that builds atop the results provided by the execution of information extraction. These stores break down into the Candidate Knowledge Store, the Mission Knowledge Store(s), the Volatile Knowledge Store(s), the Personal Knowledge Store(s), and the external Knowledge Repositories. A workflow encompassing each of these different knowledge stores in context is provided in Figure 15.

Candidate Knowledge Store

The Candidate Knowledge Store can be considered the hub of the activities within the KBP system; extracted information will flow into it, and selected elements of it may be submitted to external Knowledge Repositories. The relationship between the Candidate Knowledge Store and downstream recipients suggests there are useful ways to expand from the Candidate Knowledge Store into multiple knowledge stores with different expectations and use cases. While the default scenario anticipates a single Candidate Knowledge Store, it is also possible that a given application of this platform will require multiple such stores.

Implicit throughout the document is that the Candidate Knowledge Store is specifically a repository for knowledge graphs produced by the information extraction store (iteratively or in a batch) and that its contents may be assessed for accuracy and relevance before a transfer downstream. It arises principally from what is extracted from unstructured content, but can also take structured sources, provided they also undergo the interpretation to align them to the Candidate Knowledge Store's model. As noted above, a key concern is that many sources of error and uncertainty will inevitably be a part of its make-up, which adds one motivation to the list of reasons for why rollback (i.e., elimination of statements) may be required if the Candidate Knowledge Store is maintained for an extended period of time; other reasons might include the retention time is exceeded or information is not deemed current.

An important consideration of the Candidate Knowledge Store is that because it is likely quite sizable, it may not be feasible to implement it with a graph database (e.g., Neo4J). Instead, it may be best structured in a more scalable solution that nevertheless preserves the content that makes it conform to a knowledge graph and allows for search of relevant content that can be submitted to downstream graph databases.

Mission Knowledge Stores

While a Candidate Knowledge Store is comprehensive (i.e., contains any and all data that is fed into the KBP system at the earliest stage), downstream of this are one or more Mission Knowledge Stores aimed toward addressing specific topics relevant to a mission need. Unlike the Candidate Knowledge Store, the Mission Knowledge Stores should be graph databases with a rich ability to query and pose questions to produce subgraphs to answer questions tied to the core competencies of the system.

Methods for determining what content in a Candidate Knowledge Store should feed a given Mission Knowledge Store are not uniform. This may be based on neighborhoods around a given entity, subgraphs with content exhibiting certain keywords, or subgraphs that arise from specific sources deemed relevant.

Volatile Knowledge Stores

While the Mission Knowledge Stores intend to reflect the knowledge extracted from input content relevant to a given mission, they can also serve as a basis for individuals or small teams to explore subgraphs that interest them for a given mission or use case question. In addition to a greater focus, this sort of exploration may also require a provision for less certain—or speculative—facts to allow for different scenarios that the evidence does not widely support, but which a specific user or team wishes to entertain. This would give the ability to create subgraphs that reflect states that follow from trusted, extracted, and possible facts, which are the outcomes of questions that include conditions of "what if X holds?" One does not want the overall Mission Knowledge Stores to be flooded with facts that exist only to create proposed possibility, so a model is required to allow for such statements to have a limited scope to a scenario visible to only the individual or team asking the question. Moreover, an individual would not necessarily want every subgraph to be affected by facts associated with a specific "what if" question, so being able to maintain these as separate contexts is necessary. This ability to fork off contexts that encompass the relevant assertions from a Mission Knowledge Store and less certain assertions intended to aid in answering the question is captured by the Volatile Knowledge Stores. These are characterized as "volatile" because they are impermanent and may only live while a question is being asked unless they are archived on completion. Elevating the content unique to the Volatile Knowledge Store back to a given Mission Knowledge Store, for instance, should be done with manual oversight. As conceived, the Volatile Knowledge Stores will consist of assertions that are only visible to an individual or small team. Importantly, the Volatile Knowledge Stores are modifiable by users,[7] even though they also leverage or draw on assertions captured in the Mission Knowledge Store.

Personal Knowledge Stores

Like Volatile Knowledge Stores, the Personal Knowledge Stores are also user-driven, but specifically aimed to take content from Mission Knowledge Stores and Volatile Knowledge Stores to organize the information that is directly intended to inform an analyst product. A key value of a Volatile Knowledge Store is to ensure that the Mission Knowledge Store content is more accessible and useful. Some of the graph expectations here may be softened for ease of use by analysts.

The approach taken here is to allow more user modification and input to increase engagement and the ability to tailor a graph to a relevant subgraph and allow it to lean on additional facts to arrive at an answer. These represent a step beyond even the Volatile Knowledge Stores to further bridge the knowledge stores and any downstream products that can be generated using them. For example, Mission Knowledge Stores and Volatile Knowledge Stores both assume some facility on the part of the user to query, modify, and interpret the graph to arrive at something personally

[7] Underscoring their "volatility" as compared to Mission Knowledge Stores.

relevant. Being able to reduce the complexity of this interaction and increase the power of a user to alter or inject content into a personally-scoped subgraph should be considered when trying to close the divide between the graphs and the ultimate analytic artifacts they inform. There is a potential for interim representations somewhere between the graph and a report based on analysis that may exhibit some qualities of both graphs and also human-readable material. This is still an area of active study and consideration.

Knowledge Repositories

Strict assumptions about downstream Knowledge Repositories are avoided in the architecture presented here, other than accepting they are external or beyond the scope of the control of the KBP system and are likely to have greater permanence (hence the distinction between stores and repositories). However, there are notable subclasses of such repositories that are expected to exist which can have distinctive relationships to the system beyond being target recipients. A key example is a Trusted Knowledge Store, which is characterized by human curation, containing highly credible assertions, and usually a scope that is comprehensive with respect to the needs of a given organization. As such, a Trusted Knowledge Store might receive results selected from a Candidate Knowledge Store, but the path to addition to its graph may not be trivial as it would rely on human assessment of the selected candidate assertions to determine which, if any, are included as credible and relevant, an activity outside the control of the KBP system. Moreover, a Trusted Knowledge Store is a potential source of assertions that would be relevant to the Candidate Knowledge Store to allow for additional context and facilitate inference within the Candidate Knowledge Store. This is a much less prohibitive activity as the system can ingest the Trusted Knowledge Store's assertions that are relevant through its pipeline, bypassing information extraction and undergoing the interpretation, to add the input to the Candidate Knowledge Store's graph.

Knowledge Store System Case Studies

In this section, we describe two different applications of KBP and, at a high level, how they take the general approach and apply it to a specific implementation for knowledge stores.

Case Study 1: General Knowledge at Scale

The first case study aims to describe a comprehensive scope, where the upstream data-to-knowledge pipeline is intended to serve a category of data that is not limited to a single domain, but instead intends to capture all streams of data available into a single interconnected graph that represents the totality of knowledge. This single graph is held in a single Candidate Knowledge Store, and because of its size and scope, it is likely not a traditional graph store capable of robust inference. Instead, this store is more likely mapping its knowledge to a scalable cloud database solution, even if its entries reflect the structure of a graph store. Downstream of this are individual mission stores that are graph stores, although they are scoped to represent only relevant subgraphs of the Candidate Knowledge Store. These in turn can inform any number of Volatile Knowledge Stores, Personal Knowledge Stores, or downstream Knowledge Repositories. This variation is depicted in Figure 16; the principal difference between this and the comprehensive workflow is there being a single Candidate Knowledge Store.

Case Study 2: Science & Technology Intelligence Support

An alternate use case for knowledge base population is in support of Science and Technology Intelligence (S&TI), which is required to scale with the tremendous influx of available and relevant, structured and unstructured data characterizing technology and the real-world landscape of research, funding, development, engineering, and commerce. While narrower than the general knowledge use case, it also anticipates a considerable effort in automatically populating knowledge graphs with this content. The Candidate Knowledge Store is not needed in this case, as the queries around specific science and technology disciplines and topics are used to directly populate the Mission Knowledge Stores with the documents and entities related to them. Use cases for this activity typically are not driven by the creation of a master landscape of technological concepts and real-world entities related to them (e.g., documents, authors, funding organizations). Instead, it is a pattern that is typically repeated for specific technology areas, each representing a different mission. Because of this, mission knowledge bases focused on specific technologies are populated independently to allow for analysts to examine the history and dynamics around a given technology independently. In the specific existing implementations so far engineered to support this use case, there has not been a need for Volatile Knowledge Stores, although several downstream applications with built-in visualizations serve in the capacity of ad hoc Personal Knowledge Stores. Moreover, this has been engineered to be shareable with downstream Knowledge Repositories. A high-level depiction of this implementation is provided in Figure 17.

Command and Control of Knowledge

The Selection Process allows for the transfer of knowledge upstream or downstream through the Knowledge Layer. There are two main patterns reflected here and depicted in Figure 18.

The first is the results advancement pattern, which takes a query that collects a relevant portion of a given mission subgraph to be transitioned downstream to one or more Knowledge Repositories. Notably, this content is likely to be gated, that is to say curated or quality checked before it is ingested.

The second is the reference collection pattern, which is almost the reverse. This queries a downstream Knowledge Repository for trusted subgraphs which are then sent through the Interpretation Process Pipeline's activities, although bypassing the Candidate Knowledge Store (by way of the *dashed* lines into R and out of T), to be aligned with and then placed in a specific Mission Knowledge Store. This is expected to be resolved against existing entities in the store, but it can also serve as a "cold start" of such a store if the store begins empty. The typical cold start for KBP is described in the next section.

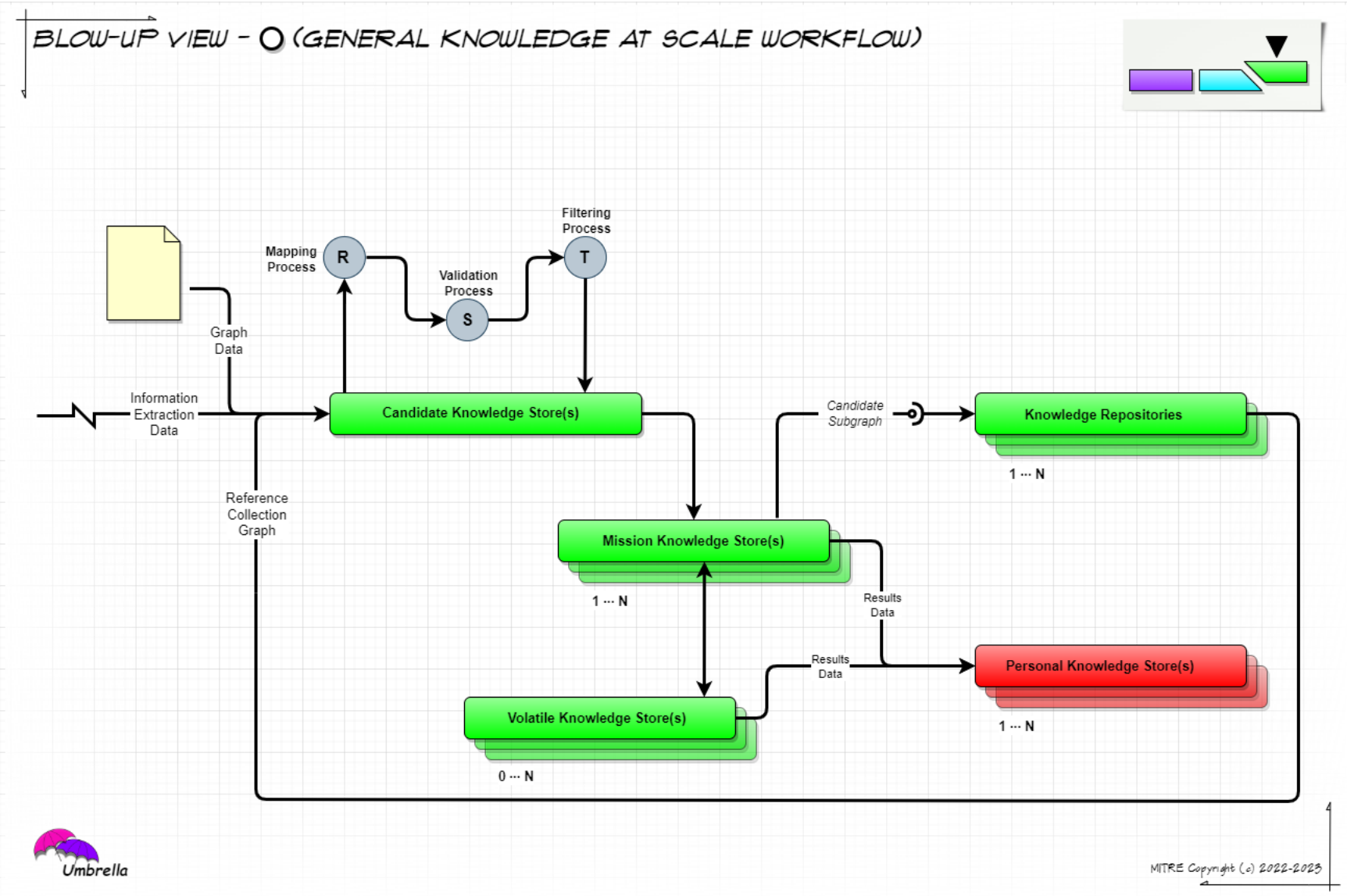

Figure 16: General Knowledge at Scale Implementation

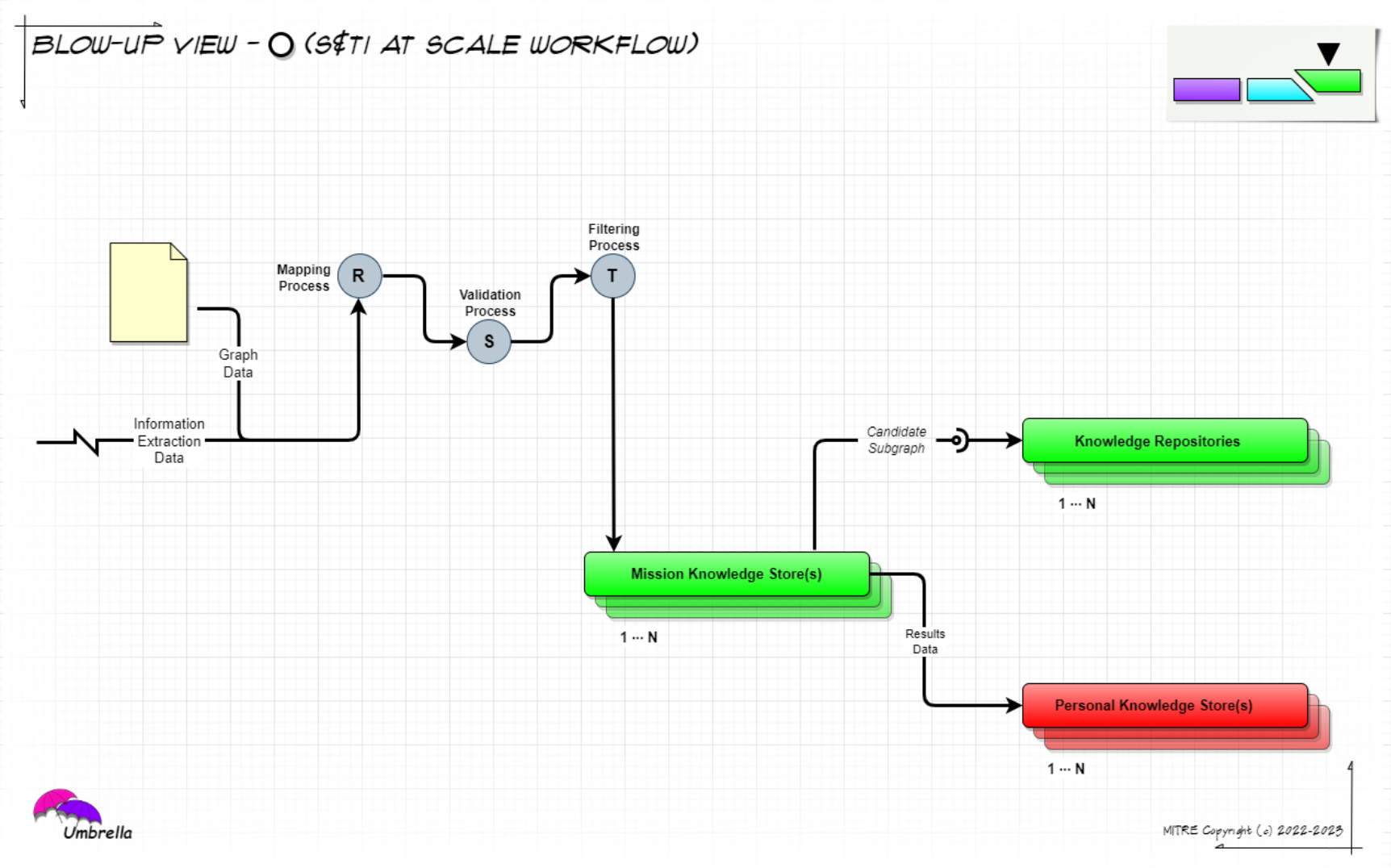

Figure 17: Science & Technology at Scale Knowledge Implementation

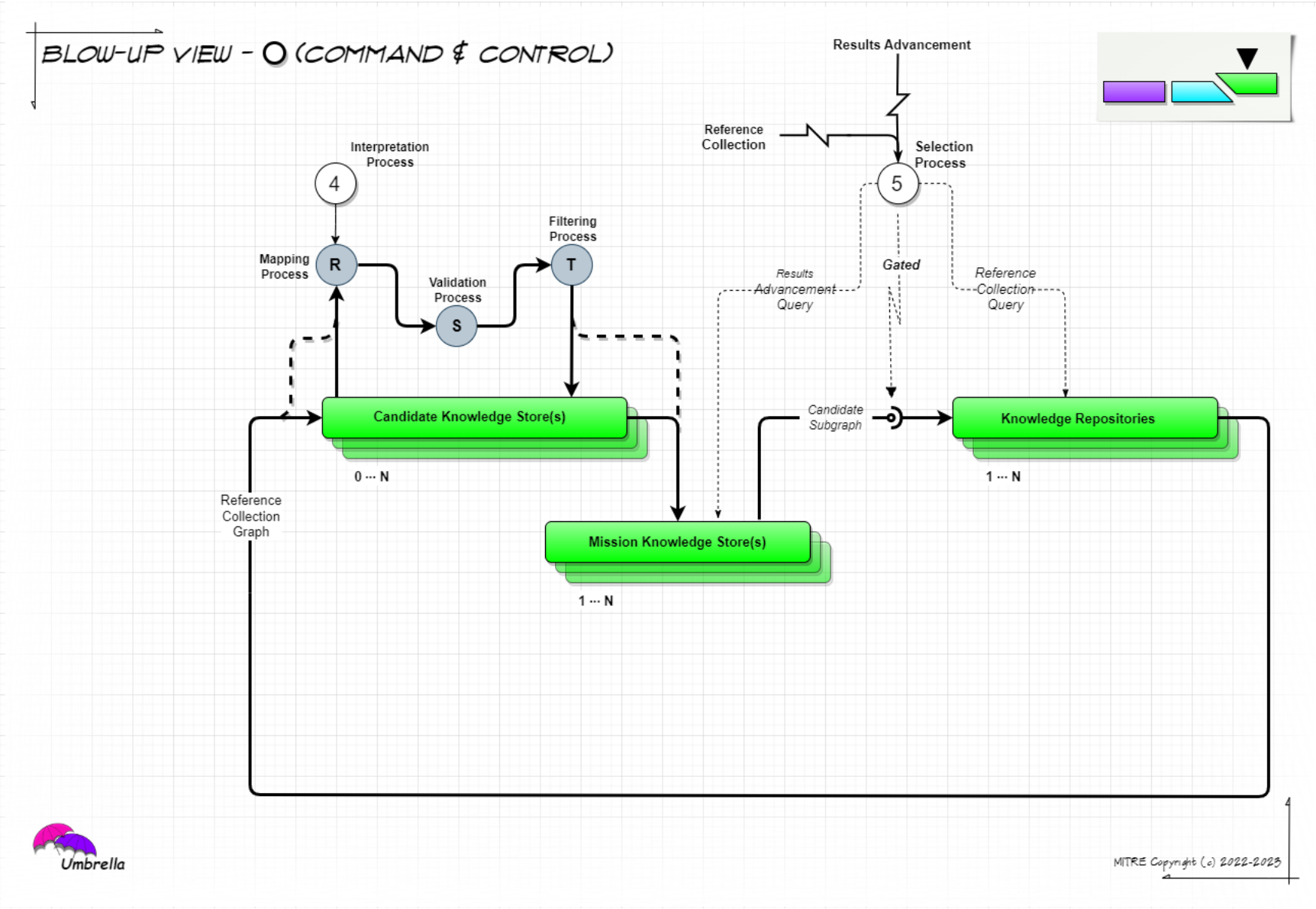

Figure 18: Command and Control of Knowledge

Cold Start

Often there is an implicit assumption that the KBP process will always be used to augment an existing knowledge base and is already underway in its operation. Nevertheless, it is important to consider what is required when approaching a cold start of the system toward a knowledge base that is not populated or where there is not yet the connectivity that such a knowledge graph requires to be useful [1, 2, 3]. This section presents the expected phases for a cold start, augmented by a specific example that fits a use case we previously described.

Issues with the cold start span the different layers of the architecture. One plausible scenario begins before even what constitutes the appropriate data is identified. For example, consider the situation described in the case study described previously on support to science and technology intelligence, where one may have an interest in a specific topic but little more to go on beyond that and perhaps some keywords and definitions that would be deemed relevant. If one has no more than this, how does one arrive as a seed graph that should be the initial knowledge base? The following diagrams walk through the KBP system with a method to answer this question. A high-level view of the cold start from initialization to a populated knowledge base is presented in Figure 19, and subsequent figures break this high-level diagram into its constituent phases.

Phase One (Figure 20) consists of the Data Mesh Layer activity executes such that content ingested either document-by-document or in bulk into the data lake (consisting of unstructured data and captured metadata) is also vectorized against a language model to be held in a vector store where the vectors represent the embeddings of that document, allowing them to be searchable by a mathematical distance from similarly vectorized queries. The vector store conforms to a schema that indicates what portions of a document and its metadata are used in the embeddings.

Through that phase, the process executes against all available data. When a request is made for a cold start based on a topic (or a set of topics) the Phase Two (Figure 21) activity initiates a run, which involves a selection process that requires an initial set of seed terms. To expand the capability past keywords to allow close alignment of the semantics of what a user actually seeks, the terms are mapped against the taxonomy (and importantly the taxonomy's explicit human-readable definitions), which are retrieved and compiled into a complete query that covers detailed descriptions of what is relevant.

Using the text gathered in Phase Two, Phase Three (Figure 22) submits the semantic query to the vector store as a vector embedding. Content in the vector store that is within a certain Euclidean distance to the vector is retrieved as query results. These results capture both the pertinent metadata captured in the semantic store and a relevant chunk of the content that was deemed similar to the query. It is in Phase Four (Figure 23) that the complete documents referred to in the semantic store that the query identified are fetched to undergo extraction and population of the knowledge base. Phase Five (Figure 24) depicts the movement of the documents retrieved through the Information Layer and into the Knowledge Layer as described in the earlier sections. Ultimately, a given document becomes a graph of the relevant entities (country, person,

document, and organization examples illustrated in the figure) which are further connected to topics derived from the query.

Once these graphs are provided, the cold start also must consider how to best populate the knowledge graph, a consideration that bears on entity resolution. The entities with the least ambiguous identifiers could, for instance, be the initial entities as they are most apt to be resolved based on their identifiers. Entities with a greater chance of overlapping identifiers (e.g., persons who might share the same name) should be downstream as they could use affiliations with other more uniquely identifiable entities to further resolve them (e.g., national citizenship, employing organization, educational institution).

When the cold start is accomplished, the artifacts it produced can give the insights needed to discover what is missing from the graph to allow for iteration on the process to pull in more documents to undergo extraction and mapping to the knowledge store. This additional iteration can also be spurred by something interesting that was discovered in the graph (as opposed to the absence of something the expected or required for the graph) in which case content describing the discovered information can dictate the query that brings in more documents to expand the knowledge in a direction dictated by those who curate the knowledge store.

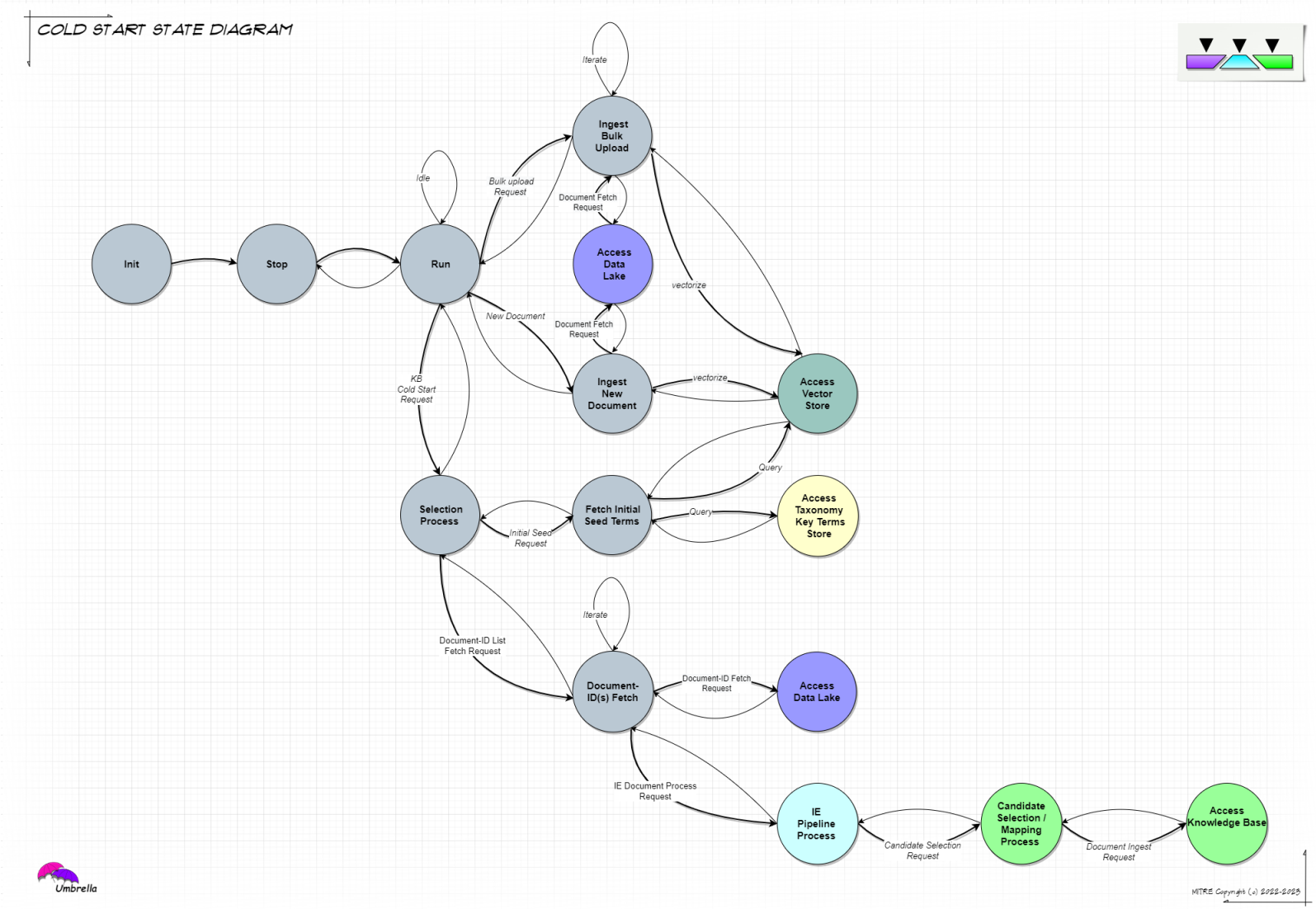

Figure 19: Cold Start State Diagram

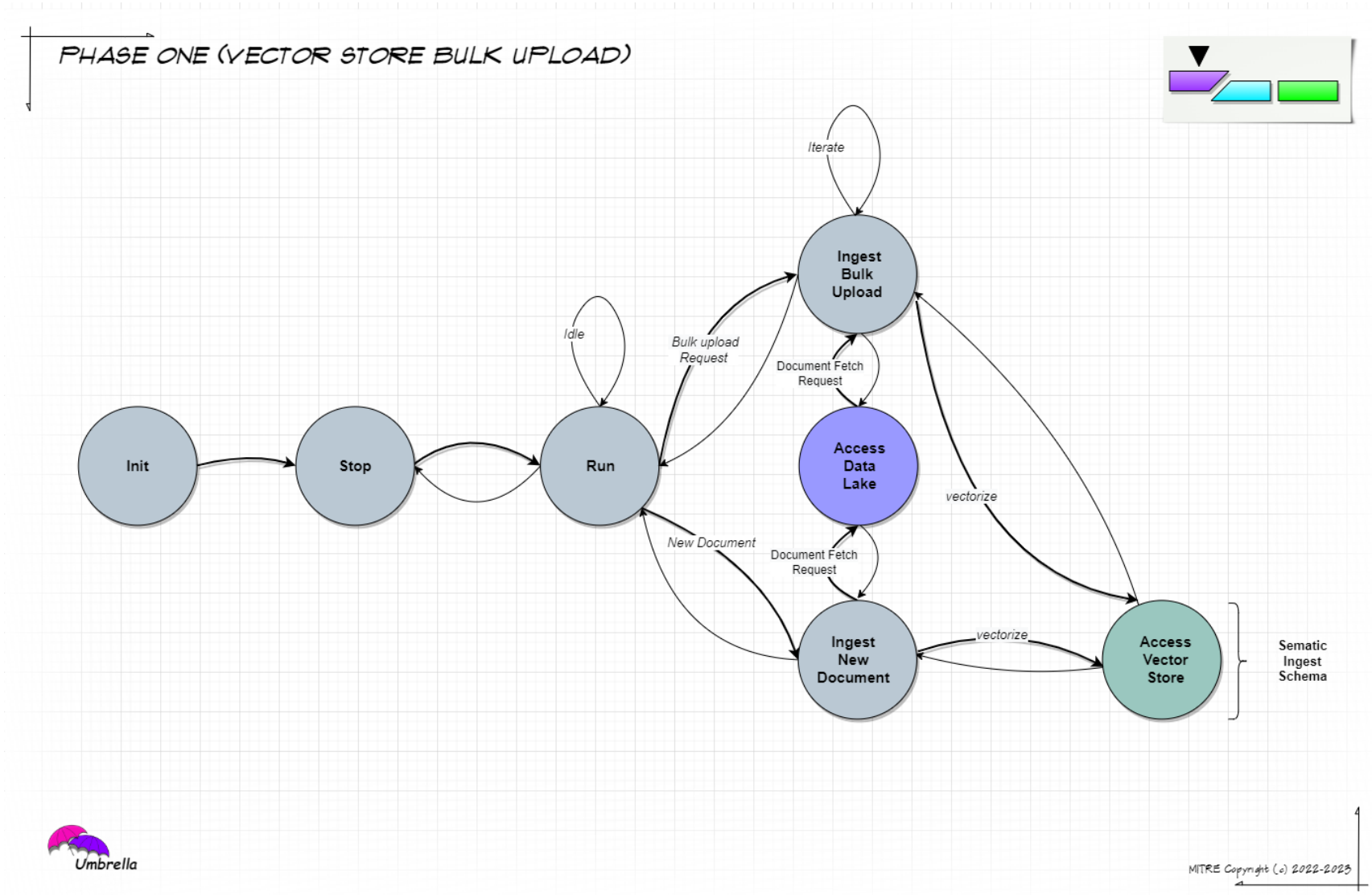

Figure 20: Cold Start, Phase One

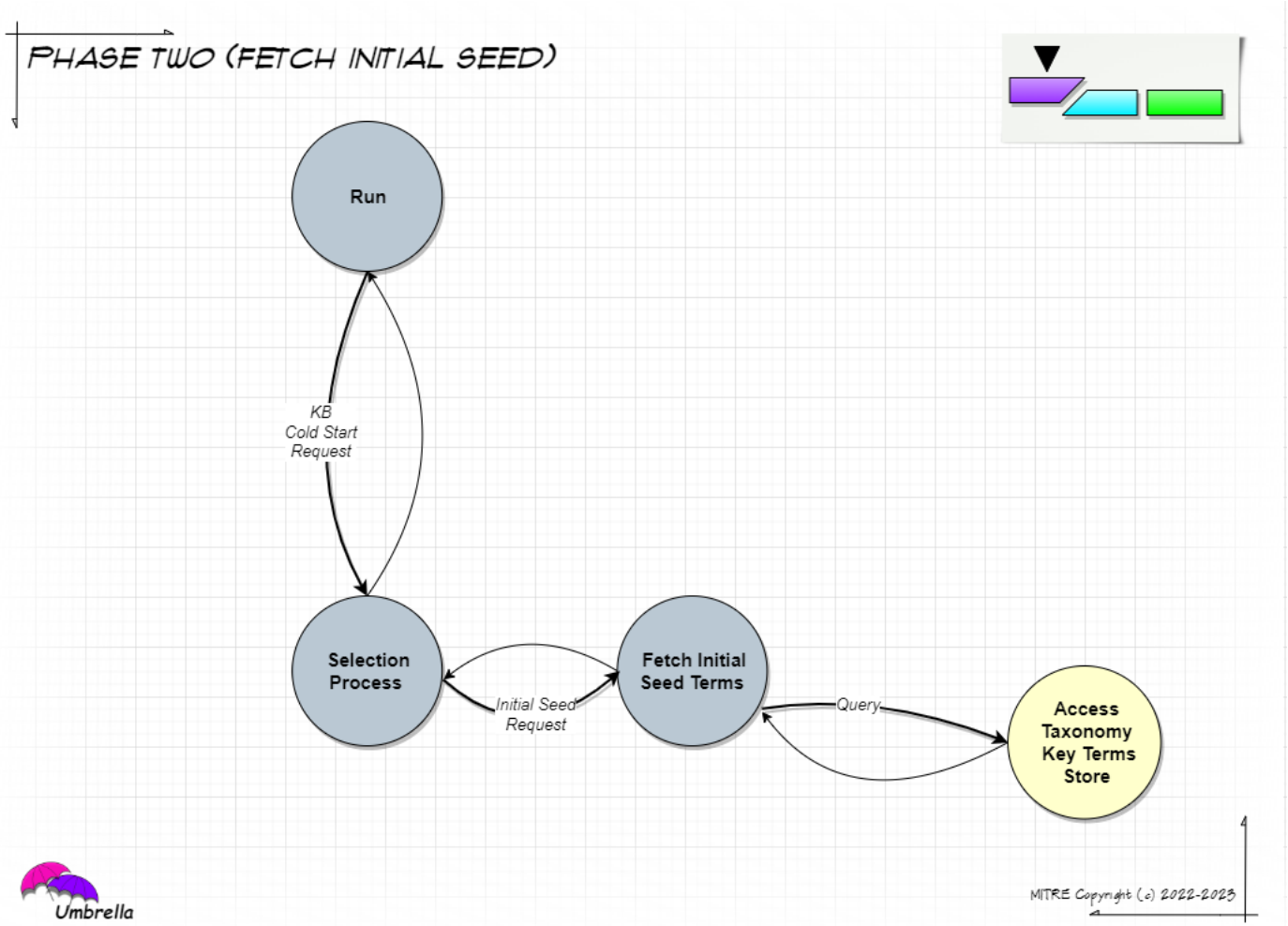

Figure 21: Cold Start, Phase Two

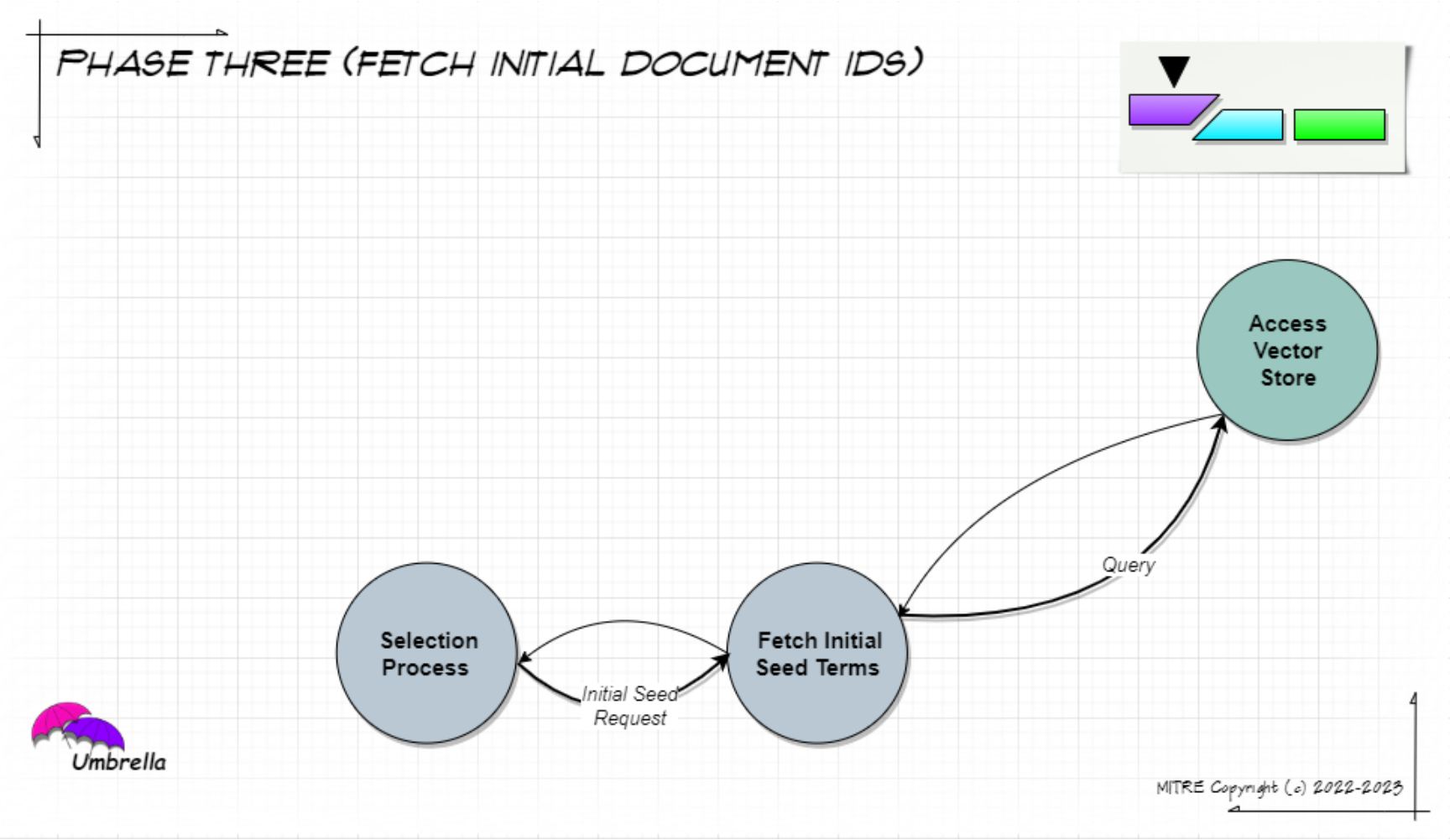

Figure 22: Cold Start, Phase Three

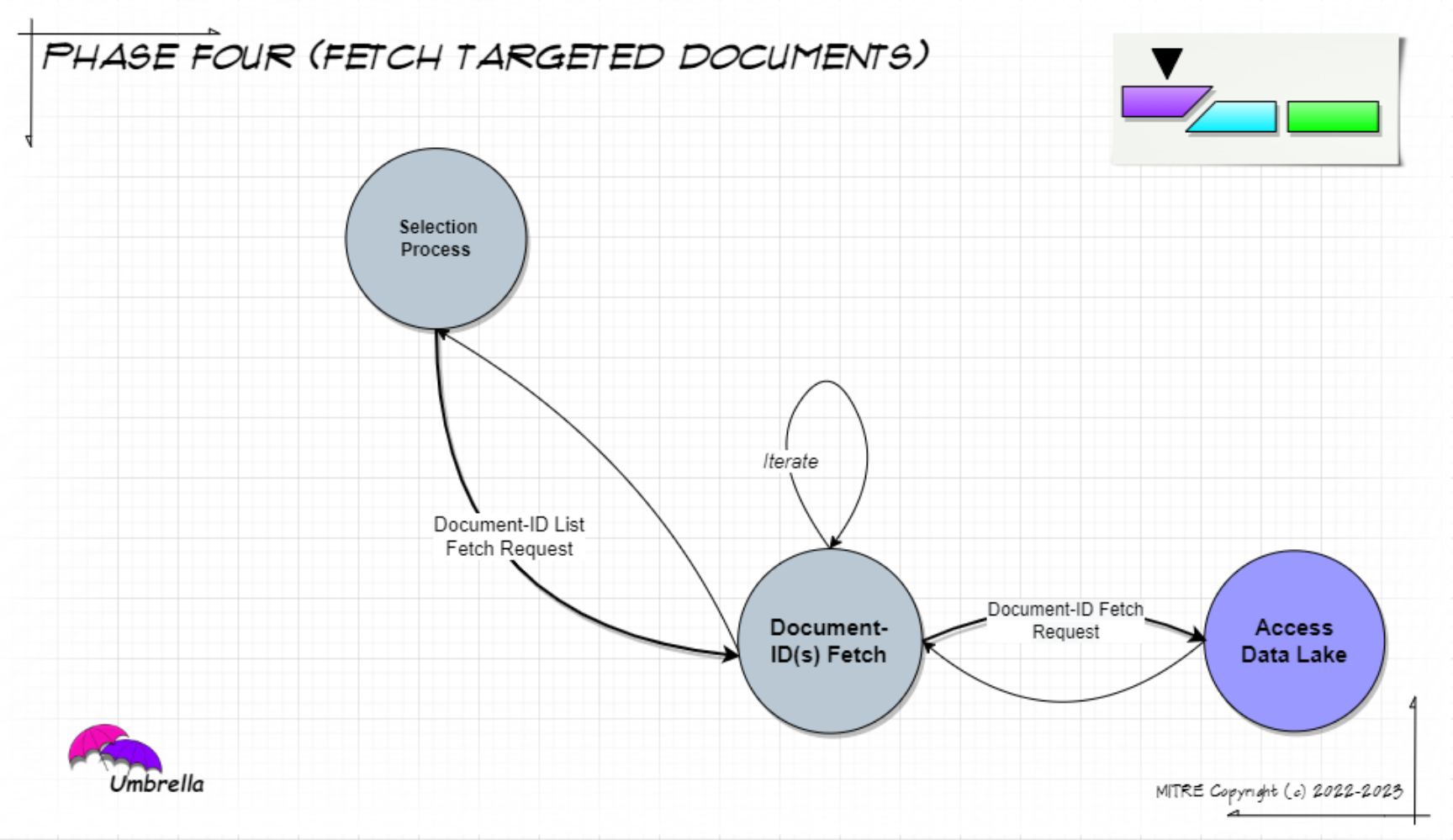

Figure 23: Cold Start, Phase Four

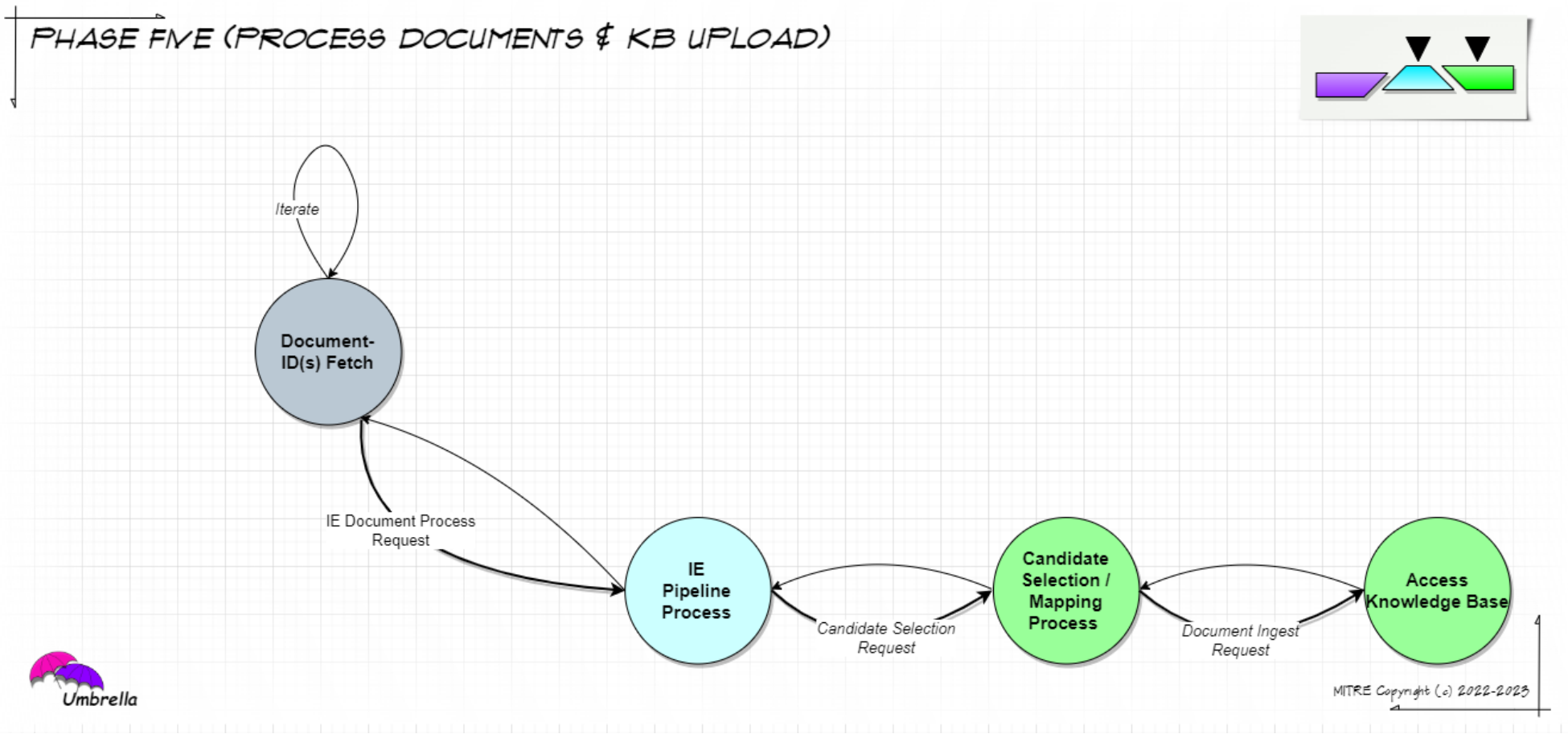

Figure 24: Cold Start, Phase Five

Conclusion

The amount of unstructured content far outstrips a manual ability to read and capture all the knowledge it contains. This is true across many domains, meaning that the automated solution that KBP offers has wide applicability. Many issues that inform KBP remain active areas of research (e.g., uncertainty, entity resolution) that can be further enabled by having this architecture as an apparatus on which to study the problems and eventually arrive at ever more efficient and accurate solutions on how to arrive at timely, impactful, and informed decisions based on copious data.

Glossary

Adapter: wrappers for information extraction components that mediate what they provide the Executive and submit to the Information Extraction Store.

Candidate Down Selection Process: activity in the workflow that determines which proposed assertions in the candidate knowledge store should be transferred to one or more downstream knowledge repositories used in practice.

Candidate Knowledge Store: a repository for knowledge content produced by the information extraction store, the contents of which may be assessed for accuracy and relevance before transfer to downstream knowledge repositories that require them.

Component: a generalized term for any distinct analytic application operating in the Extraction Process with the purpose of extracting one or more results from data or that data augmented by a previous component's execution.

Cold Start: in the context of a Knowledge Base Population System, this occurs when document processing is started with an empty Knowledge Store; that is, there are no assertions about entities, relations, or events in the knowledge graph from which the system can expand and interpret incoming document information.

Data Mesh Layer: phase of workflow that encompasses access, query, and triage of multiple data lakes to produce input documents for information extraction.

Data Lake: a scalable repository that allows you to store, index, and query documents (structured and unstructured).

Data Property: a named link in the Knowledge Layer that bridges an entity subject to a literal object.

Data Triage: the phase that ensures the correct and necessary data is retrieved from the data lake(s) and made available to a given instance of a workflow through the system.

Decomposition Process: activity in the workflow that breaks complex documents (e.g., PDF, Office) into metadata and plaintext content that will undergo extraction.

Entity: a thing extracted from content and ultimately represented as a typed individual in the Knowledge Layer, which acts as a node argument of object and data properties.

Equivalence Class: a set of unique identifiers for instances that are asserted to refer to the same instance.

Executive: code that orchestrates the workflow from the query against the data lake to the decision processes of what candidates are offered for delivery to downstream knowledge repositories.

Extraction Process: activity in the workflow that performs extraction of entities, relationships, and other content (e.g., topics); this will also include the coreference of mentions of entities within a document.

Fetched Documents: a subset of documents in the data lake that a query has selected to pass to information extraction.

Group: a set of members (instances) that can accommodate explicit plurals in the Information Layer.

Information Extraction Component Cache: the "raw" output of a component wrapped by an adapter.

Information Layer: phase of workflow that encompasses extraction and resolution.

Extraction Processing Record: a record of the application of adapters to the document.

Information Extraction Store: a repository for both the subset of documents fetched for information extraction and the interim results of the extraction process on that subset.

Instance: a distinct thing in the Information Layer, asserted to exist and inclusive of entities, events, and relationships.

Interpretation Process: activity in the workflow that includes 1) mapping entities to the semantics and structure of the candidate knowledge store, 2) linking entities discovered by way of information extraction to the entities extant in the candidate knowledge store, and 3) preserving the provenance and uncertainty metadata from the source.

Knowledge Layer: phase of workflow that encompasses activities on the candidate knowledge store, including the selection of assertions to send to other knowledge repositories.

Knowledge Repositories: any number of repositories outside the direct control of the workflow but which are the ultimate recipient of assertions discovered by the overarching process executing on the data.

Knowledge Staging: the phase that ensures the correct knowledge assertions are selected for sending to other knowledge repositories.

Log: list of the relevant events that go into the creation of an instance in the Information Extraction Store.

Mapping: the process of converting an Information Extraction Store instance into a representation compatible structurally and semantically with the Knowledge Layer's ontology.

Microtheory: a set of entities and properties that supplements a more comprehensive upper-level ontology, is self-consistent, and elaborates on a specific topic area.

Mission Knowledge Store: a mission-focused graph database intended to support specific use case questions.

Namespaced Token: a string that contains a namespace (a contextual string) and an arbitrary string (representing a description).

New Data: any data, structured or unstructured, being provided to the data lake.

Object Property: a named link in the Knowledge Layer that bridges one entity to another.

Ontology: a model for knowledge consisting of a taxonomy of entity and property labels as well as rules for proper structure and inference.

Personal Knowledge Store: user-driven knowledge store, specifically aimed to take content from Mission Knowledge Stores and Volatile Knowledge Stores to organize the information that is directly intended to inform an analyst product; graph expectations here may be softened for ease of use.

Query: a request for data placed against the data lake to fetch documents that will undergo information extraction.

Raw Output: the results a component will produce before any mediation from an adapter.

Selection Process: activity in the workflow that performs queries on the data lake to fetch a subset of documents to undergo information extraction.

Upper-level Ontology: an ontology where the entities and properties are largely universal and applicable to any domain or use case.

Trusted Knowledge Store: a knowledge store by human curation, containing highly credible assertions, and usually a scope that is comprehensive with respect to the needs of a given organization.

Volatile Knowledge Store: a knowledge store intended to explore both known facts pulled from the Mission Knowledge Store and speculative “what-if” assertions; intended for use by small teams or individuals and disappear after use unless archived.

Appendix A. AI Assurance

The architecture this document describes largely assumes it will be driven by analytic components informed by AI or ML. These can provide capabilities that exceed what is possible manually, but they also incur certain risks that require an approach that bears in mind a variety of assurance concerns that go beyond just performance in the best case or average case scenarios, especially when it will be deployed, monitored, and sustained.

Core considerations of AI assurance aim to ensure systems enabled by AI be 1) **reliable**, as in performing their behaviors consistently, 2) **robust**, as in being able to adapt and recover after being fielded, 3) **secure**, as in being resistant to being subverted, tampered with, or stolen, 4) **interpretable**, as in having actions and outcomes that can be explained and held to account, 5) **safe**, as in not causing cause unintended harm, and 6) **equitable**, as in fair and not exhibiting bias.

Toward a government context, the Department of Defense in 2021 released a memo[8] that described distinct AI ethical principles that overlap the above assurance considerations. These principles (expressed verbatim from the memo) include being:

- **Responsible**: DoD personnel will exercise appropriate levels of judgment and care, while remaining responsible for the development, deployment, and use of AI capabilities.
- **Equitable**: The Department will take deliberate steps to minimize unintended bias in AI capabilities.
- **Traceable**: The Department's AI capabilities will be developed and deployed such that relevant personnel possess an appropriate understanding of technology, development processes, and operational methods applicable to AI capabilities, including transparent and auditable methodologies, data sources, and design procedure and documentation.
- **Reliable**: The Department's AI capabilities will have explicit, well-defined uses, and the safety, security, and effectiveness of such capabilities will be subject to testing and assurance within those defined uses across Al capabilities' entire life-cycle.
- **Governable**: The Department will design and engineer AI capabilities to fulfill their intended functions while possessing the ability to detect and avoid unintended consequences, and the ability to disengage or deactivate deployed systems that demonstrate unintended behavior.

It is our intent to apply AI technology in a way that strives to meet the principles expressed here.

[8] https://media.defense.gov/2021/May/27/2002730593/-1/-1/0/IMPLEMENTING-RESPONSIBLE-ARTIFICIAL-INTELLIGENCE-IN-THE-DEPARTMENT-OF-DEFENSE.PDF

Works Cited

[1] B. Lika, K. Kolmvatsos and S. Hadjiefthymiades, "Facing the cold start problem in recommender systems," *Expert Systems with Applications,* vol. 41, no. 4, pp. 2065-2073, 2014.

[2] J. Bobadilla, F. Ortega, A. Hernando and J. Bernal, "A collaborative filtering approach to mitigate the new user cold start problem," *Knowledge-Based Systems,* vol. 26, pp. 225-238, 2012.

[3] J. B. Schafer, D. Frankowski, J. Herlocker and S. Sen, "Collaborative Filtering Recommender System," *The Adaptive Web,* pp. 291-324, 2007.

[4] ASF, "Apache Tika," The Apache Software Foundation, [Online]. Available: https://tika.apache.org/. [Accessed 2022].

[5] "Tesseract Documentation," 2022. [Online]. Available: https://tesseract-ocr.github.io/.

[6] R. Arp, B. Smith and A. Spear, Building Ontologies With Basic Formal Ontology, MIT Press, 2015.

[7] I. Niles and A. Pease, "Towards a Standard Upper Ontology," in *Proceedings of the 2nd International Conference on Formal Ontology in Information Systems,* Ogunquit, Maine, 2001.

[8] E. Tabassi, K. Burns, M. Hadjimichael, A. Molina-Markham and J. Sexton, "NISTIR 8269," October 2019. [Online]. Available: https://csrc.nist.gov/publications/detail/nistir/8269/draft.

[9] R. Winder, J. Jubinski, C. Giannella and M. Jones, "ODYSSEY: A SYSTEMS APPROACH TO MACHINE LEARNING SECURITY," April 2021. [Online]. Available: https://www.mitre.org/publications/technical-papers/odyssey-a-systems-approach-to-machine-learning-security.

[10] MITRE, "MITRE Atlas," [Online]. Available: https://atlas.mitre.org/ . [Accessed 2022].