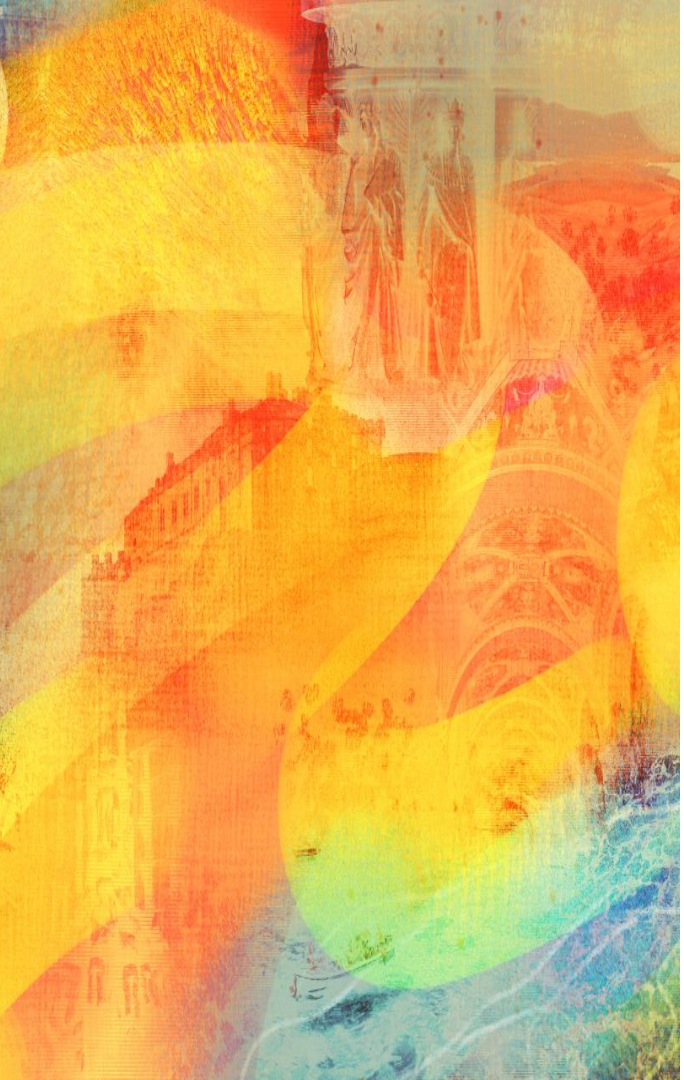
GRAPHS, GRAPHS EVERYWHERE...

Many thanks to Tatiana Grechishcheva for help in preparing the Knowledge Graph slides



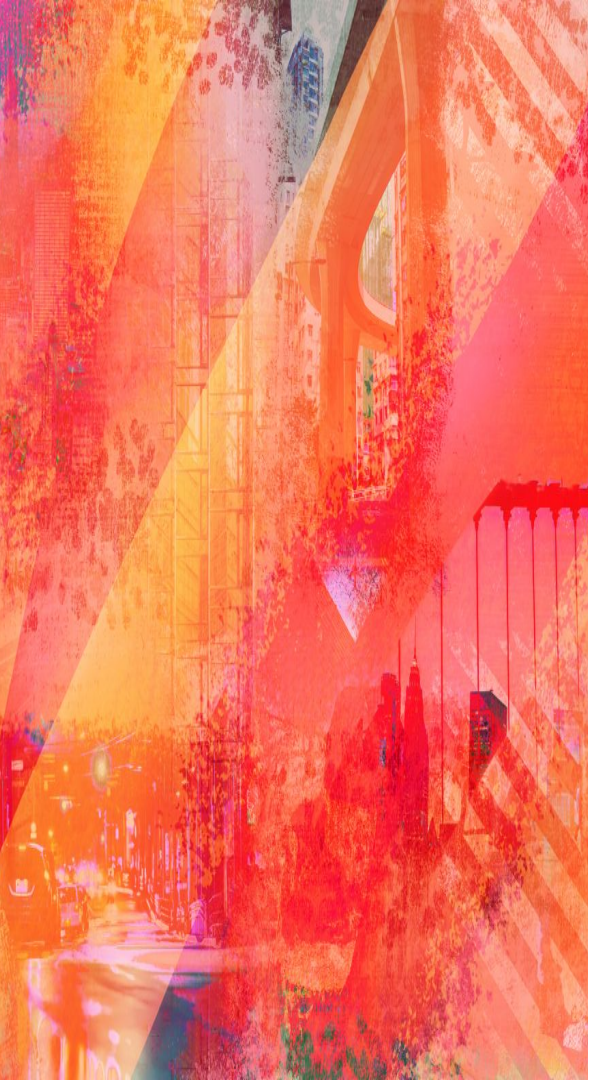
Graphs, graphs everywhere...

Before diving into the options you have for building multi-agent systems, let’s have a quick refresher on the different kinds of graphs involved

- Graphs for representing knowledge
  - Entity-centric
  - Document-centric
  - Hybrid
- Graphs for representing computation flow
  - State machines
  - Event-driven computation

Graphs representing knowledge

- The first important kind of graph is that representing raw knowledge we’re trying to retrieve
- Here it’s important to understand the difference between the traditional entity-centric “knowledge graph” linking entities via their relationships, and the new varieties: document-centric and mixed



A classic knowledge graph:

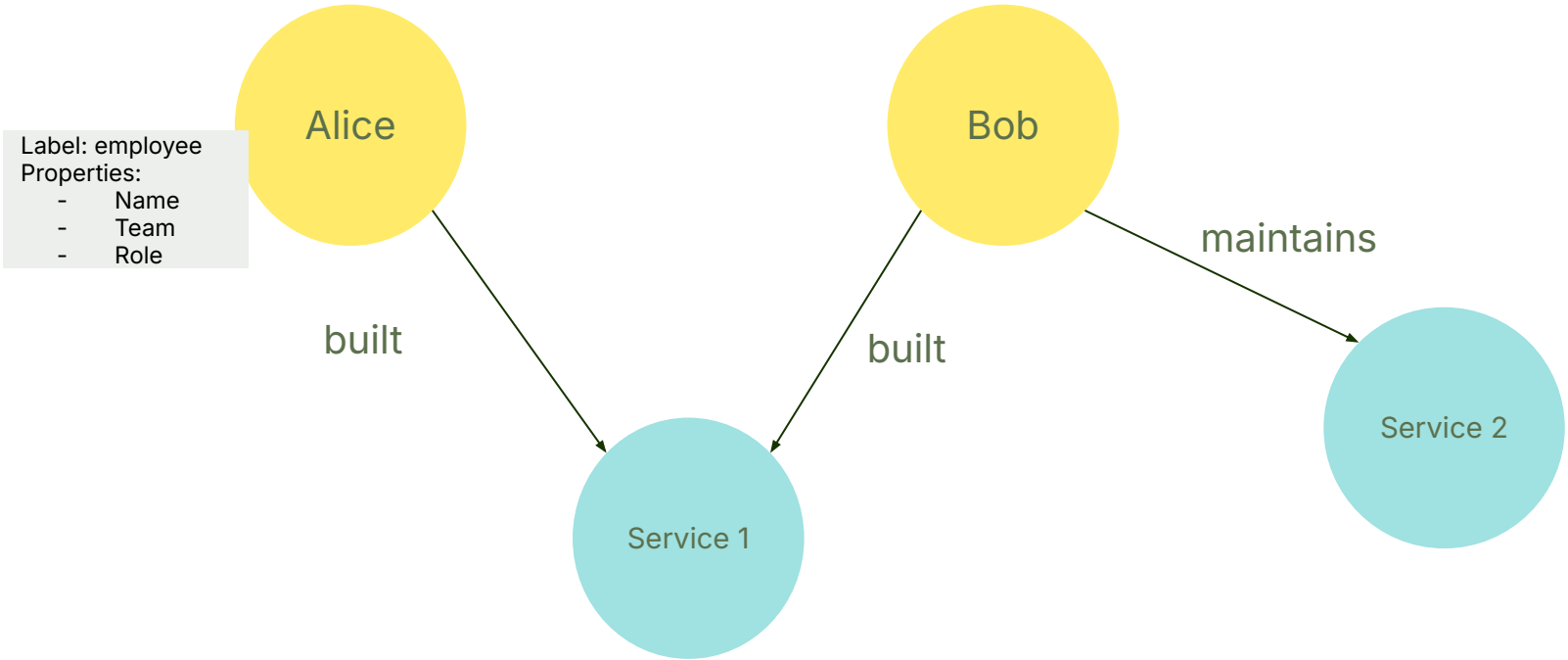
Graph vertices (nodes) are **entities,** and the edges connecting them are their **relationships**.

Nodes have **labels** (entity types) to group them together.

Relationships have a **type** and **direction**.

Both nodes and relationships can have **properties**

Example of an entity-centric Knowledge Graph



Cypher - language for querying KG

Multi-hop queries!

WHO WAS BUILDING SERVICES WITH ALICE?

```
MATCH (alice: Person {name: “Alice”}) - [:BUILT] →(s)← [:BUILT] - (coBuilder)
RETURN coBuilder.name, s.name
```

Why the resurgence of knowledge graphs?

- LLMs make it easy (if computationally expensive) to auto-generate entity-centric knowledge graphs, by extracting entities and their relationships
- Graphs are the natural next step beyond embedding similarity for RAG retrieval

Entity-centric vs document-centric graphs

Section 1:

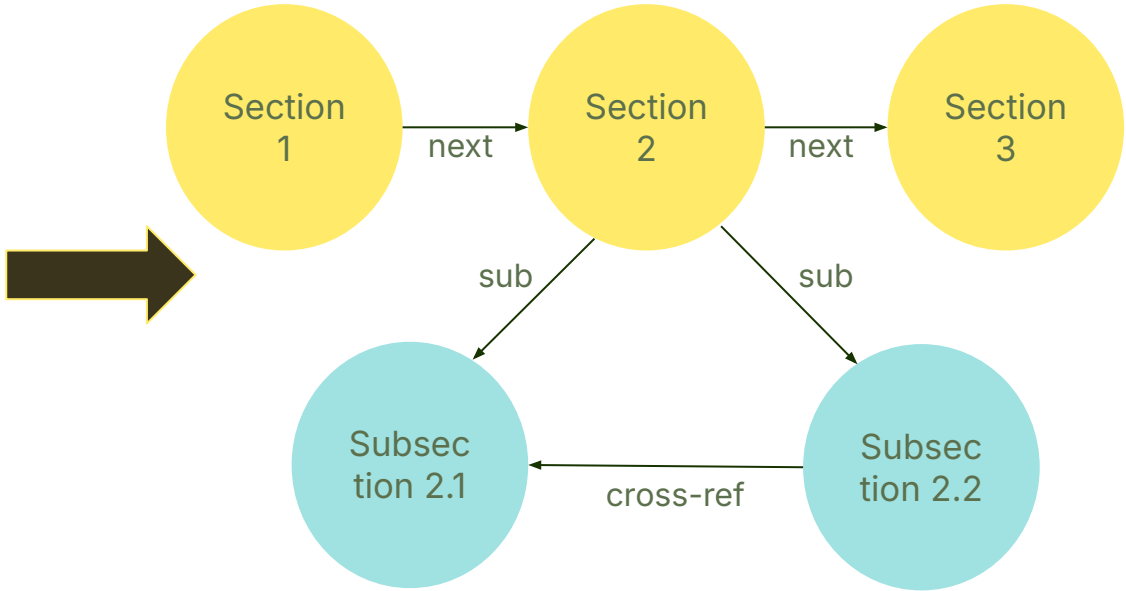
Section 2:

Subsection 2.1

Subsection 2.2

Section 3:

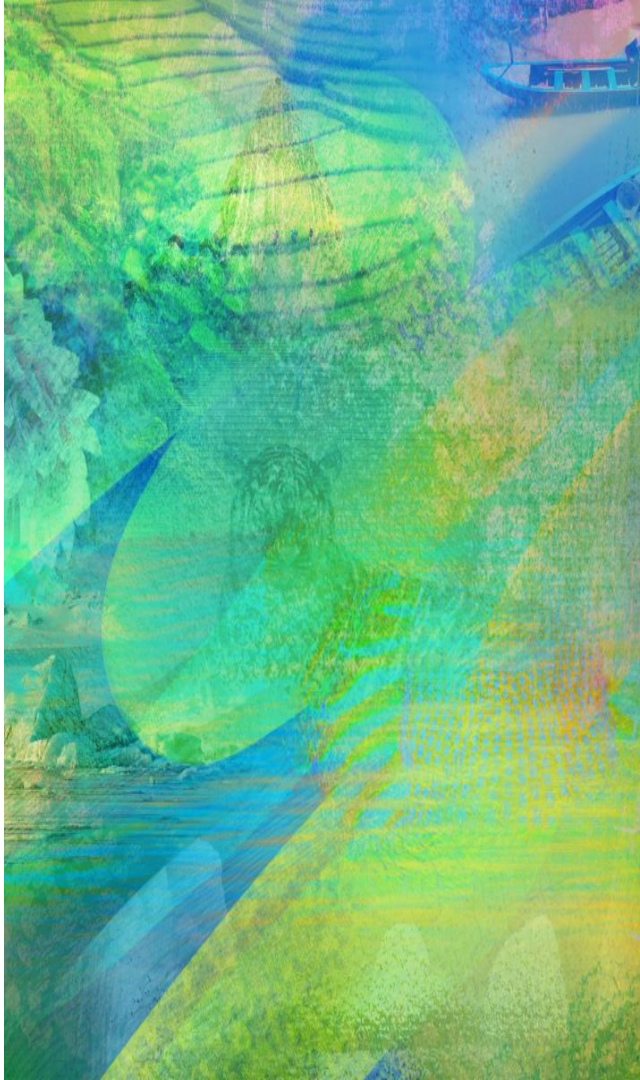
....

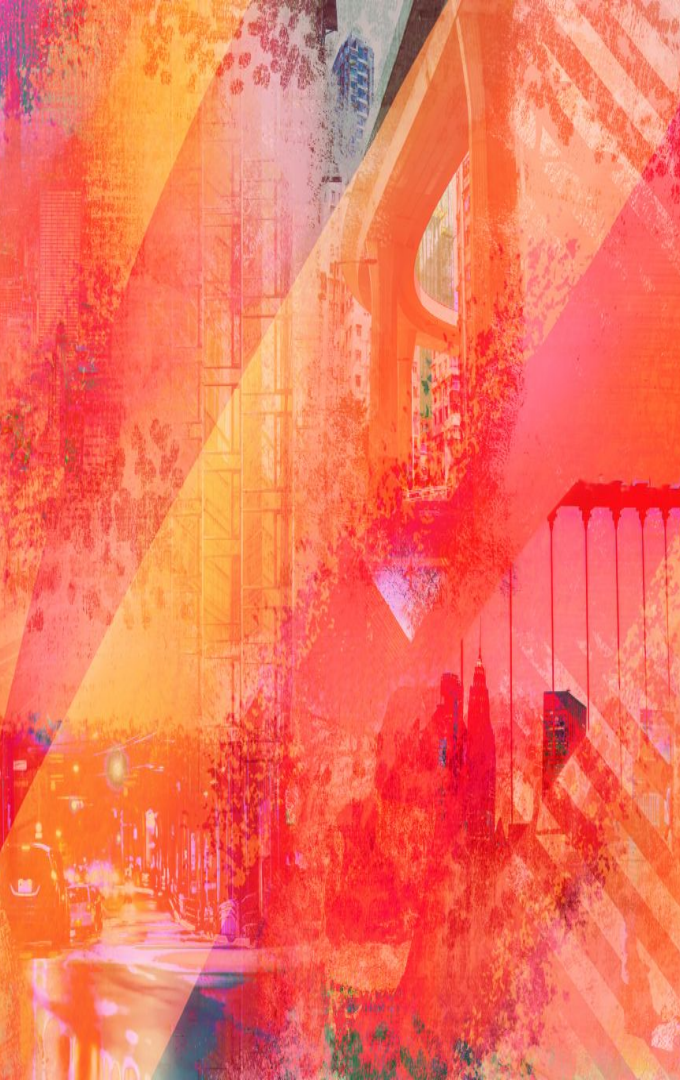


Mixed graphs

- There’s no reason why you should restrict your graph to only one type of nodes and edges
- It’s completely reasonable to have some of the vertices be document chunks, and others be concepts or keywords contained in those chunks
- In fact, that’s how **Datastax’s Astra DB** works!

https://medium.com/building-the-open-data-stack/a-guide-to-graph-rag-a-new-way-to-push-the-boundaries-of-genai-apps-f616d47758a0





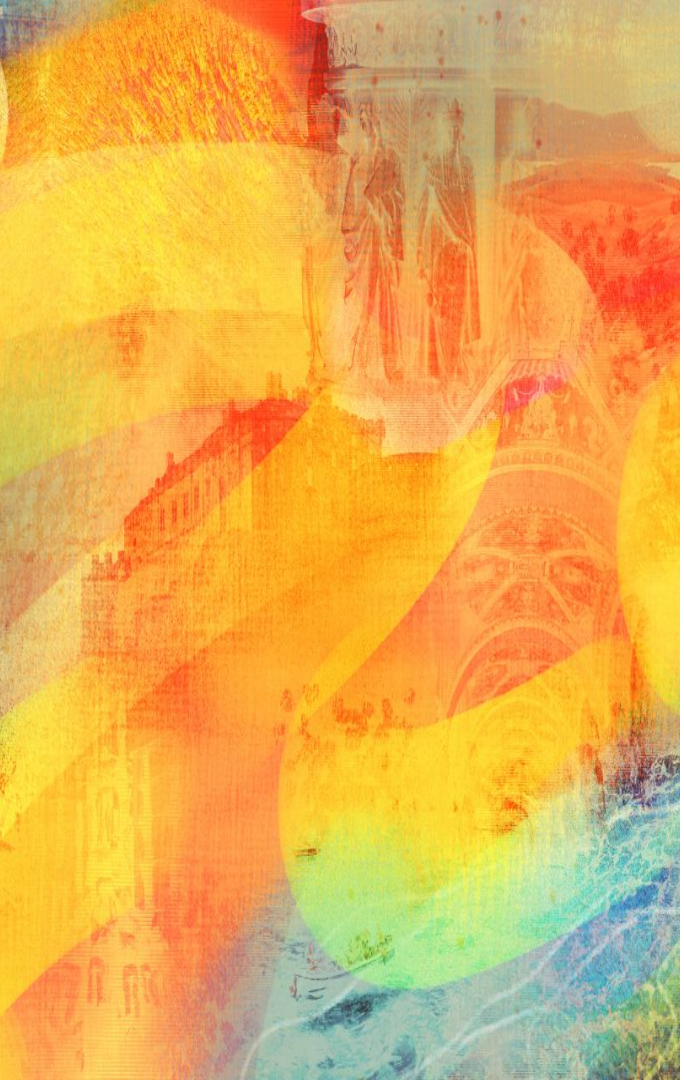
Graphs to define computation logic

Another, quite distinct usage of directed graphs is defining computation logic, in particular in multi-agent systems. This has two important, distinct varieties:

- State machine graph
- Event propagation graph

State machine graph

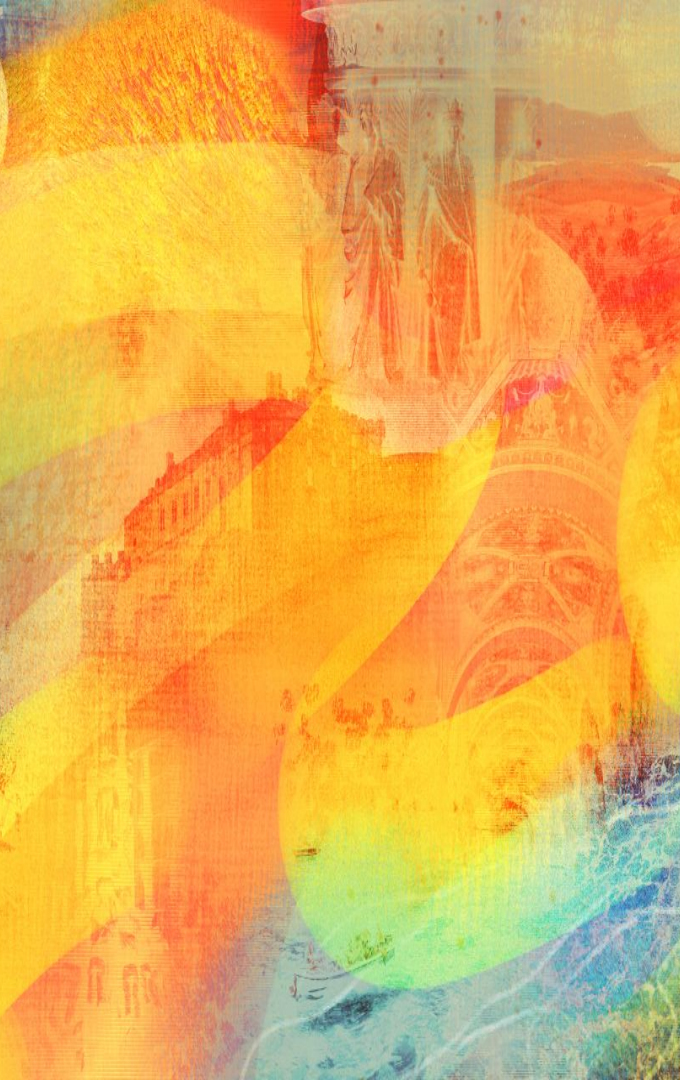
- Most important examples: Langgraph, OpenAI Swarm
- Exactly one node at a time is active
- There is a single system “state” that the active node can access
- The active node inspects and modifies the state and decides which of its children nodes will become active next



Event-driven computation graph

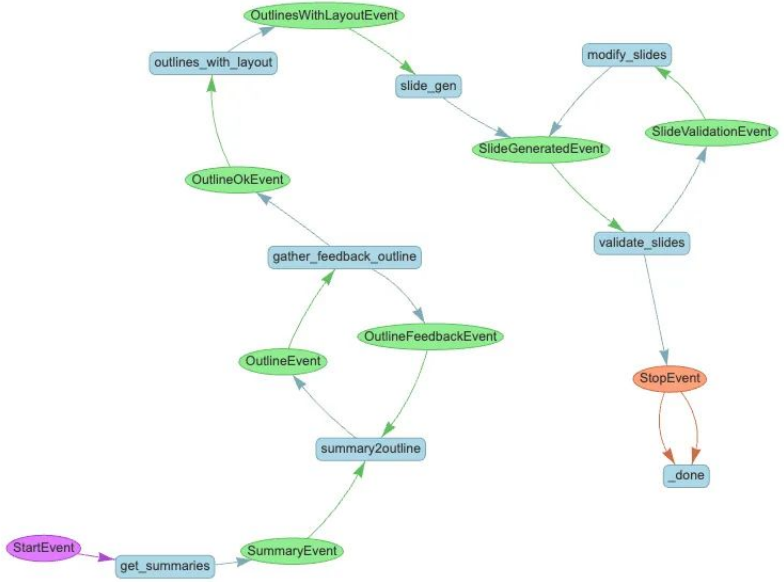
- Most important example in GenAI space: LlamaIndex Workflows
  - Many examples in other contexts, eg Faust
- Each node in the graph emits one or multiple event types
- Each node in the graph listens to one or multiple event types
- Two nodes are considered connected if the second node listens to an event the first one can emit

https://motleycrew.readthedocs.io/en/latest/examples/event_driven.html



Example:

A slide generation workflow



Source:https://towardsdatascience.com/how-i-streamline-my-research-and-presentation-with-llamaindex-workflows-3d75a9a10564

Actually...

- A state machine can be looked at as a special case of an event-driven computation graph, in which
  - Each edge is a distinct event type
  - Every node
    - Has to respond after receiving a single event
    - Emits exactly one event in response
    - Holds no state beyond that contained in the event
- As state machines are such a common and neat abstraction, it makes sense to single them out
- However, in the following discussion when I refer to “event-driven computation” I implicitly include state machines