HMS

analytical software

HMS analytical software

LLM-MVC

A simple design pattern for structuring standard LLM tasks with generic prompts

Dr. Robert Bauer
Senior Data Scientist
HMS Analytical Software GmbH

data2day 19.09.2024

Introduction

Problem Statement

- Unstructured text from LLM input/output is difficult to handle and maintain, especially for large systems
- In some cases, working with structured data (models) is the better approach and many productive LLM systems use this
- However: there are different ways how such model centric systems can be designed and controlled: Do we use chains? Agents? DAGs? And do we always need the „big guns“?

Topic of the Talk

We introduce a simple design pattern for model centric LLM systems build upon the concepts of the well known Model-View-Controller architecture

Agenda

01 | About HMS

02 | Model-Driven Prompting

03 | Motivation for Design Patterns

04 | MVC pattern for LLM workflows

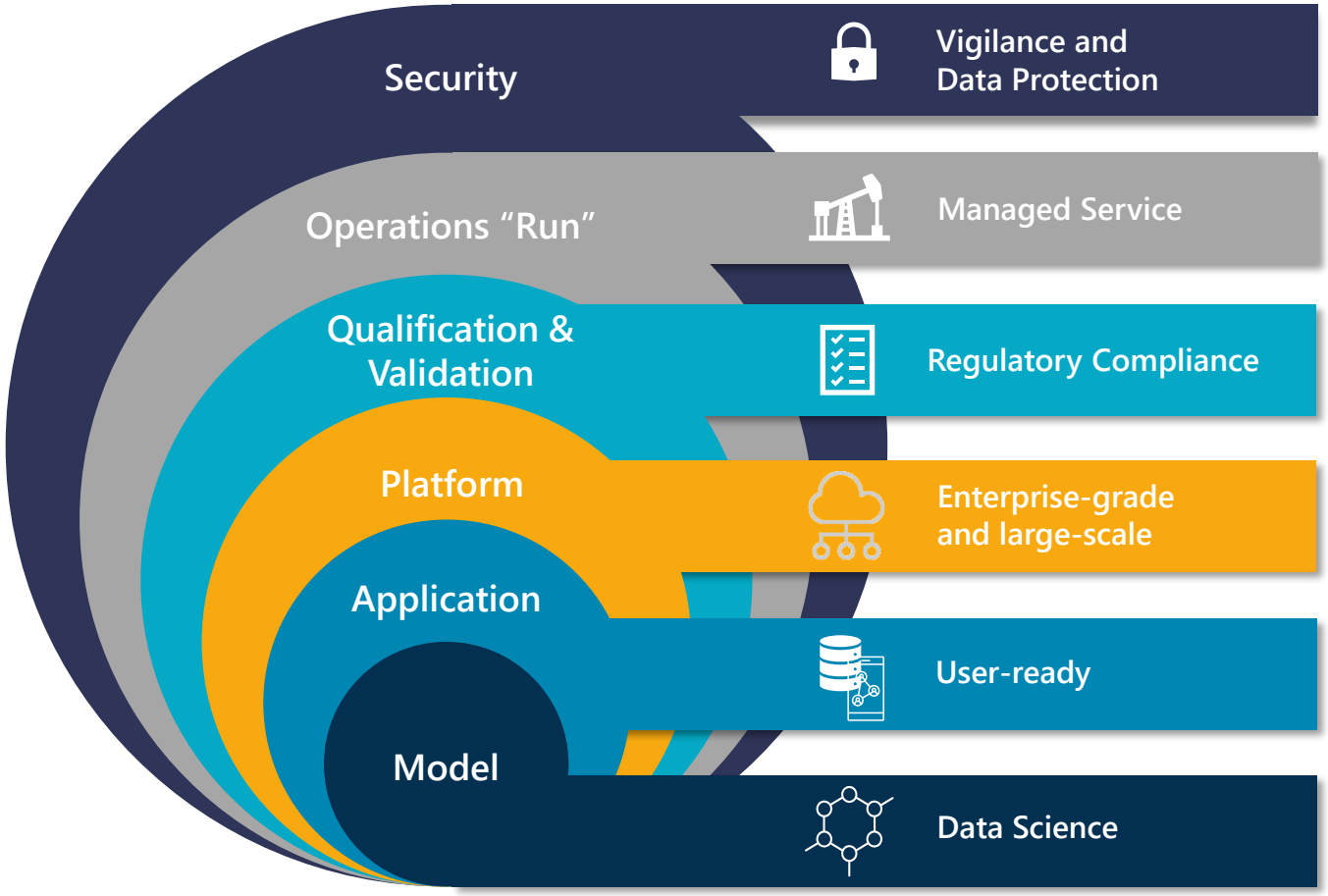
05 | Model Decomposition

06 | Code Examples

07 | Lessons Learned and Summary

01 | About HMS

HMS Analytical Software GmbH



02 | Model-driven Prompting

Model-driven Prompting

Model Driven Engineering

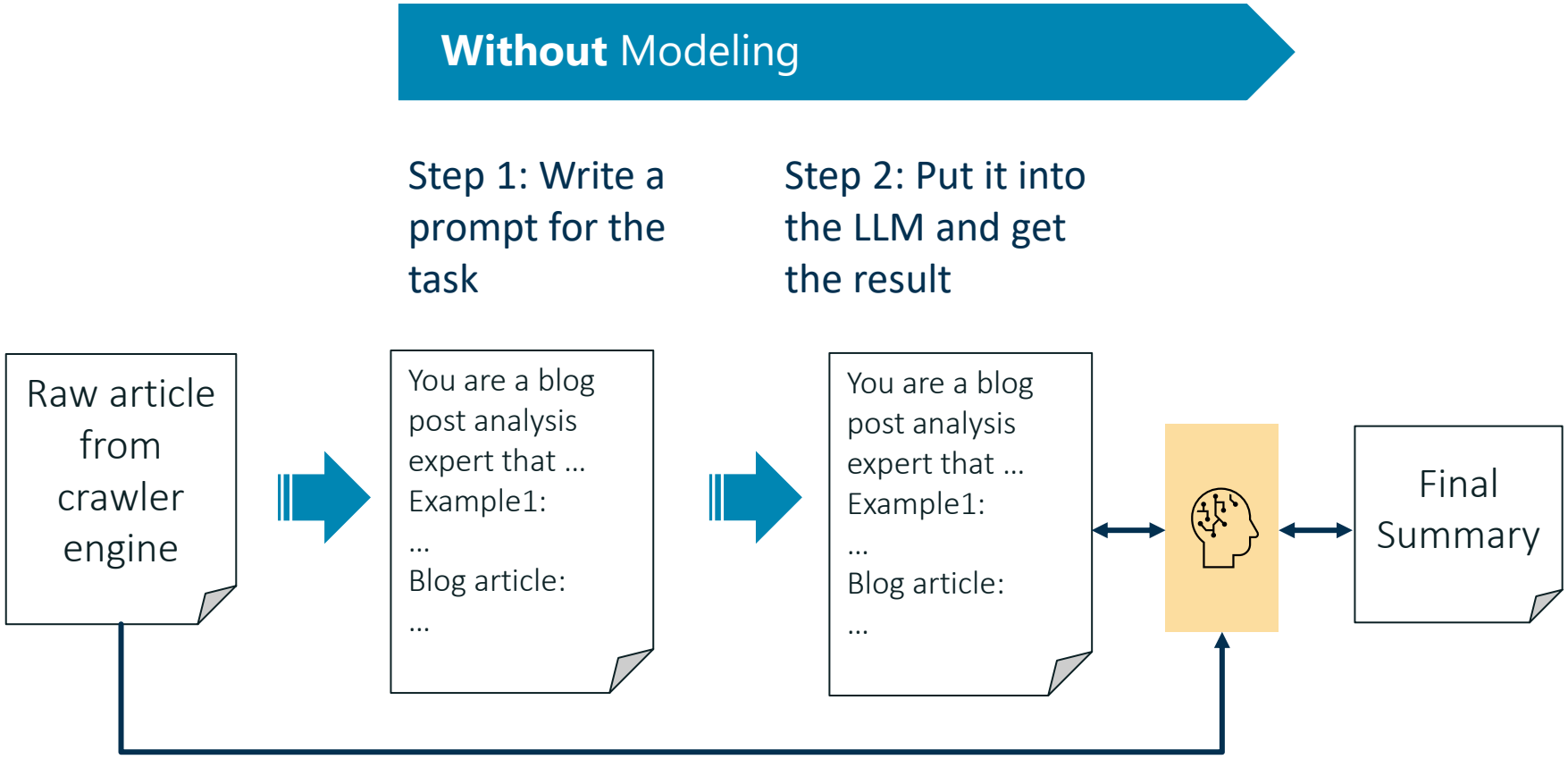
- Create **domain models** as abstract representation of knowledge (e.g. using DSLs) to solve problems
- Can be done with data structure representations such as JSON/YAML/XML/…

Model Driven Prompting

- Use **models for prompting** or meta-models to construct prompts
- Simple but effective approach: JSON schema as template for a to-be-created object
- Could also be referred to as **generic prompting**

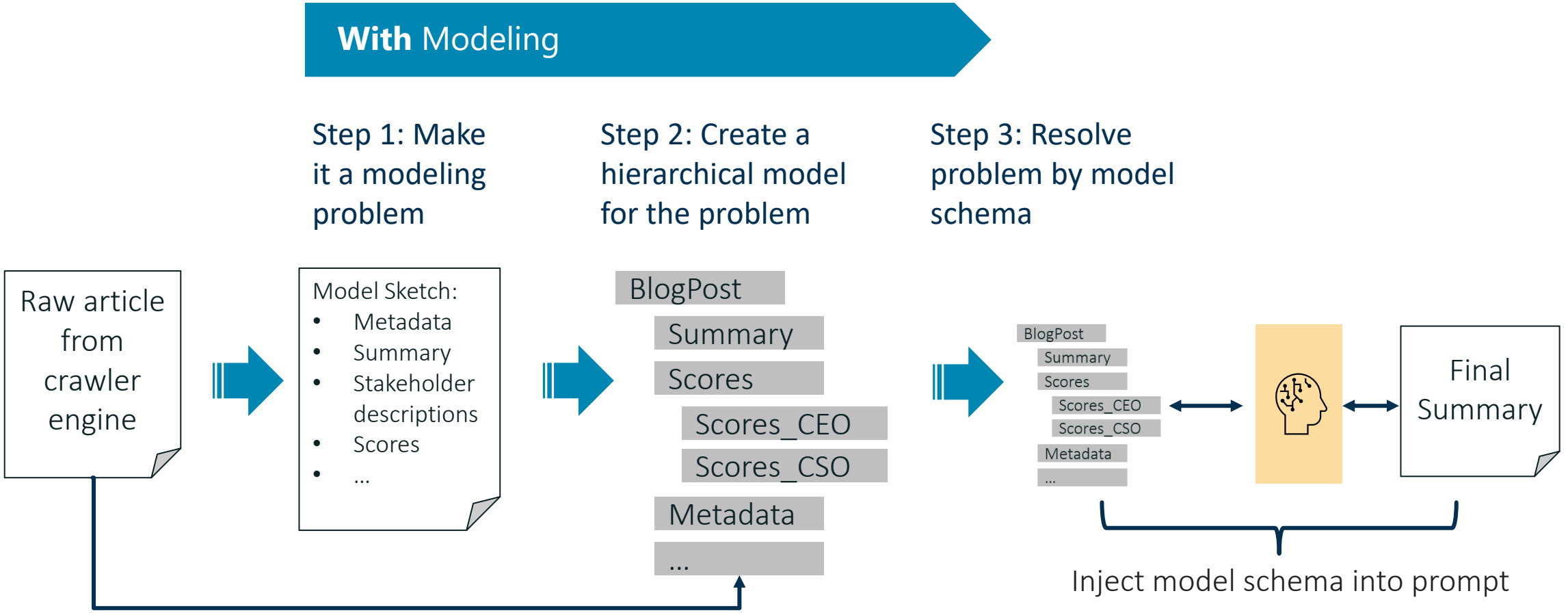
Direct Prompting

Task: Summarize tech blog posts and perform relevance scoring for different stakeholders (CEO, CSO, …)



Model-driven Prompting

Task: Summarize tech blog posts and perform relevance scoring for different stakeholders (CEO, CSO, …)



Model-driven Prompting

```python
from typing import Union


# Pydantic
class Joke(BaseModel):
    """Joke to tell user."""

    setup: str = Field(description="The setup of the joke")
    punchline: str = Field(description="The punchline to the joke")
    rating: Optional[int] = Field(
        default=None, description="How funny the joke is, from 1 to 10"
    )


class ConversationalResponse(BaseModel):
    """Respond in a conversational manner. Be kind and helpful."""

    response: str = Field(description="A conversational response to the user's query")


class Response(BaseModel):
    output: Union[Joke, ConversationalResponse]


structured_llm = llm.with_structured_output(Response)

structured_llm.invoke("Tell me a joke about cats")
```

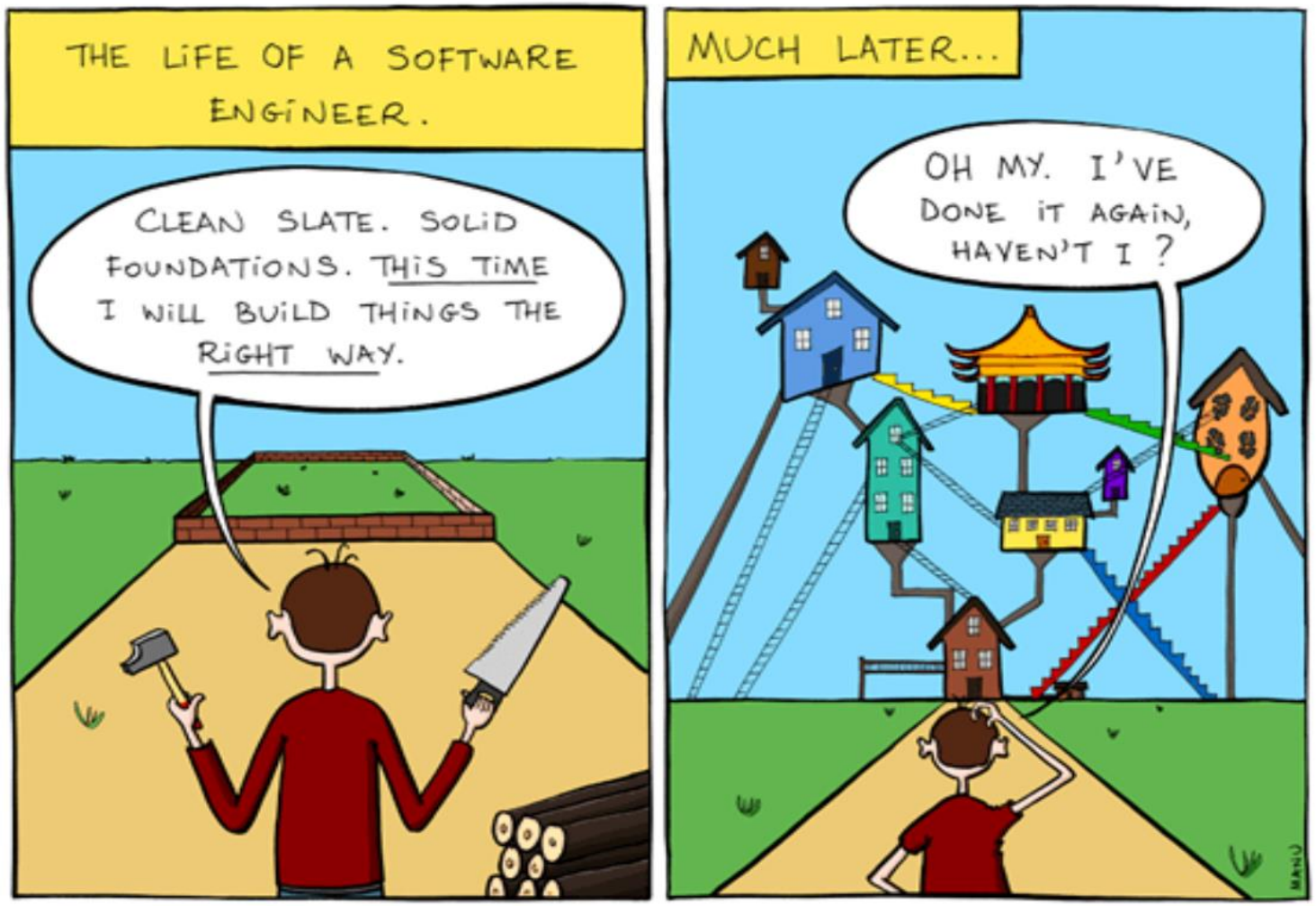
https://python.langchain.com/v0.2/docs/how_to/structured_output/

03 | Motivation for Design Patterns

Design Patterns

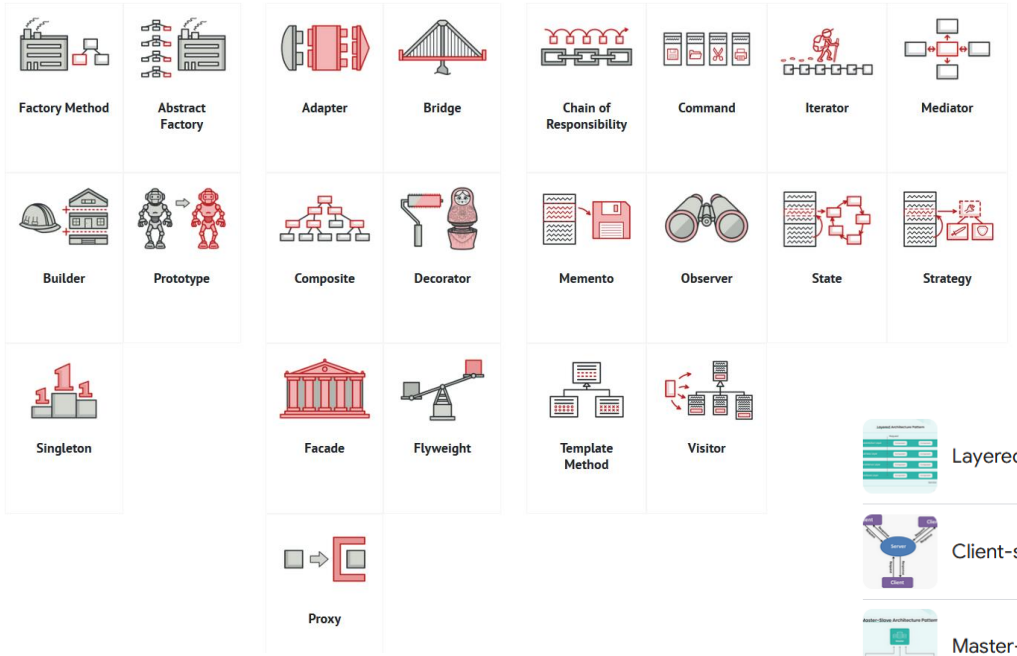
Design Patterns **describe common problems** in one or two words and solve them in a **structured and proofen** way

And they help us avoiding situations like these ☺



Original Source: http://www.bonkersworld.net/building-software/
Accessed via: https://blog.codeinside.eu/2011/11/27/pragmatische-softwareentwicklung/

Examples

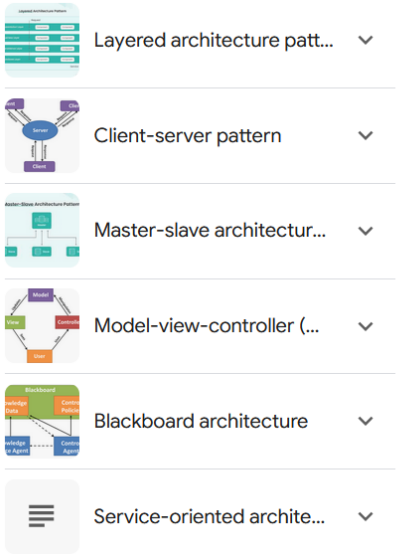


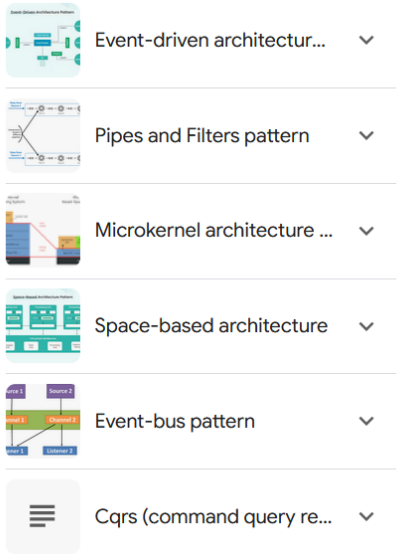
Common software design patters

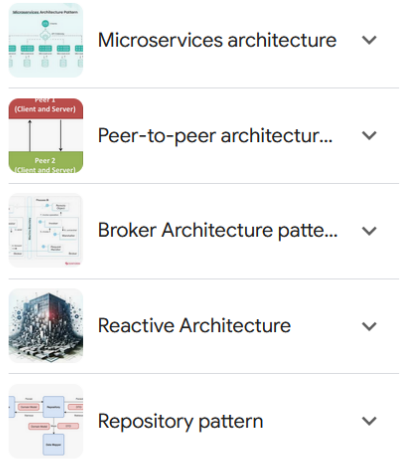
https://refactoring.guru

Common architecture design patters

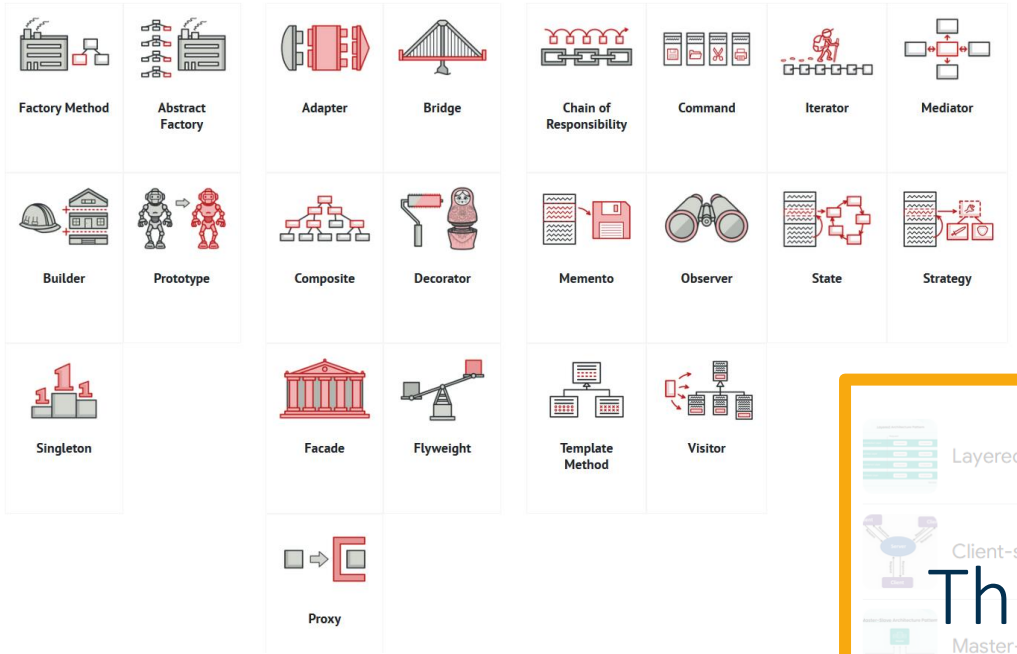
Summary from google search







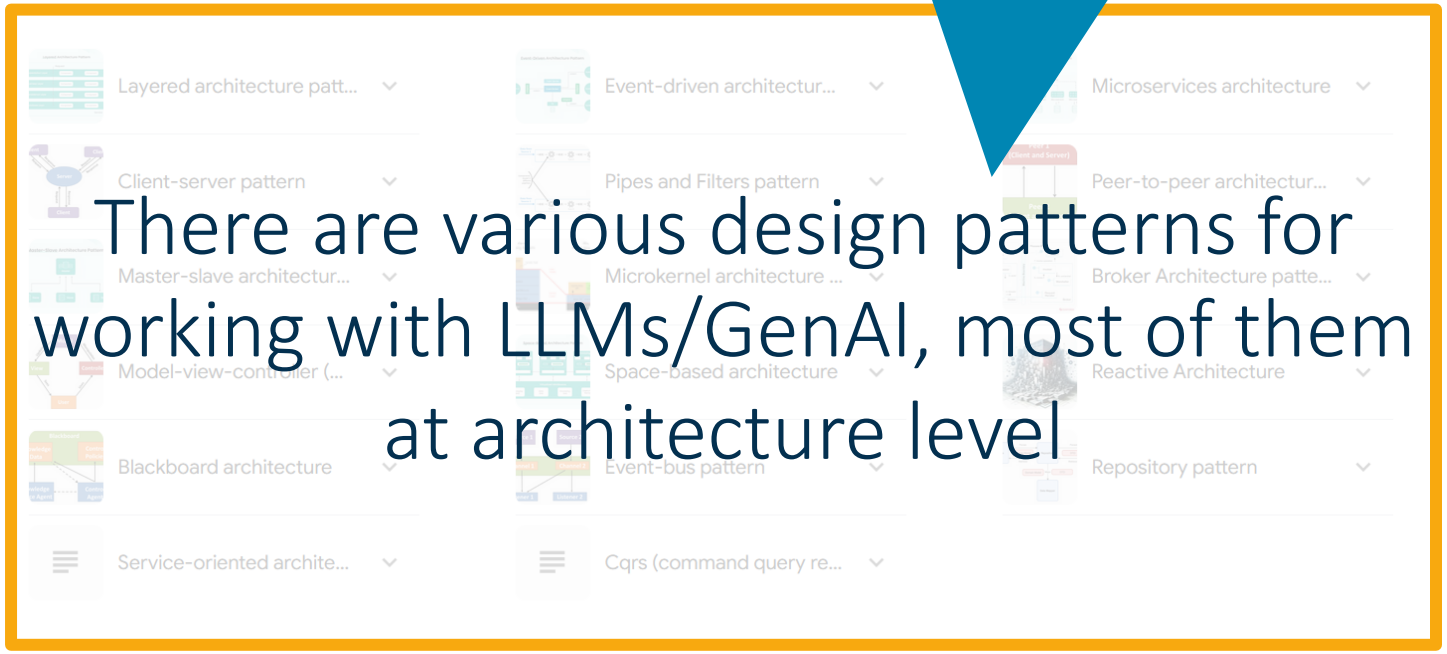
Examples



- Various patterns for RAG / Few-Shotting / Validation and other important LLM topics
- Panel of Experts
- Swarm of Agents
- Red & Blue Team Dual Model Evaluation
- Scratchpad
- …

Common architecture design patters

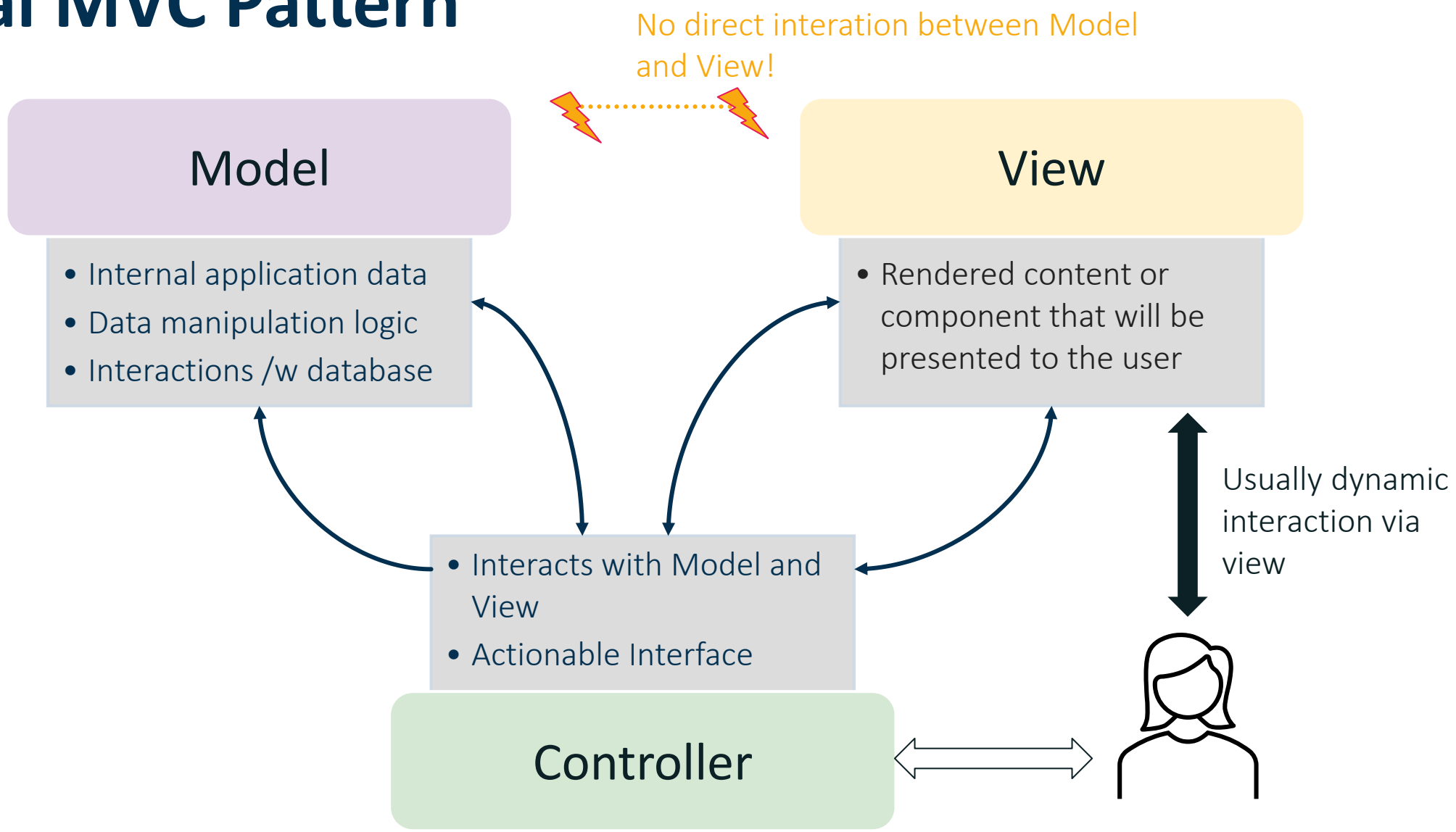
Summary from google search



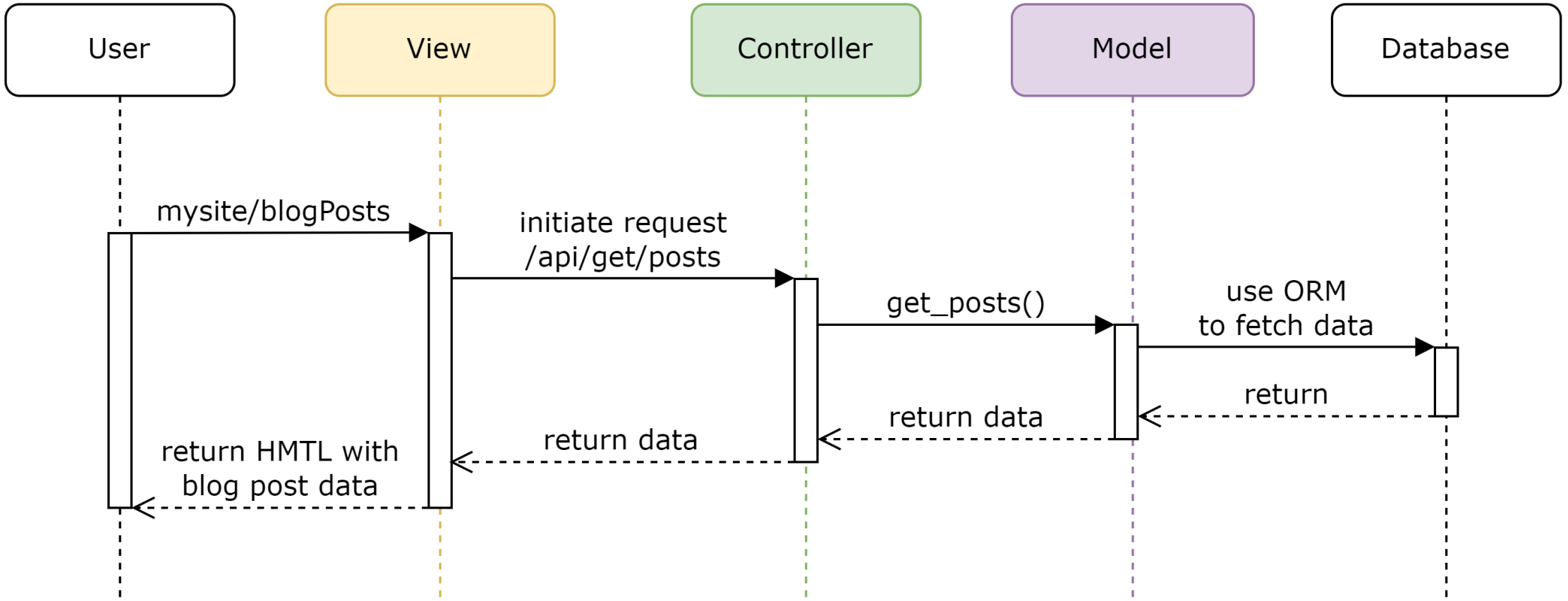
There are various design patterns for working with LLMs/GenAI, most of them at architecture level

04 | MVC pattern for LLM workflows

Original MVC Pattern



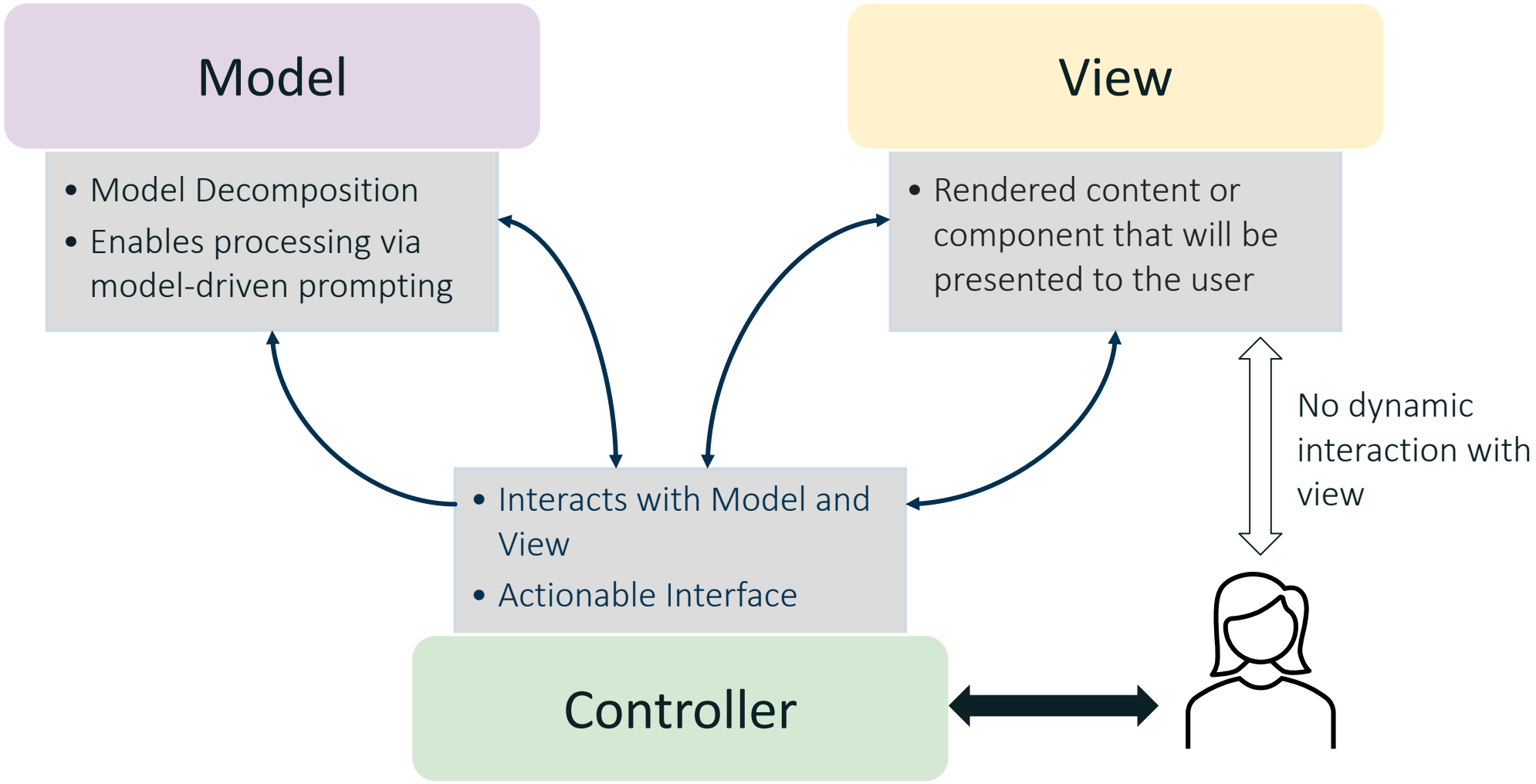
Example Workflow with Original Pattern



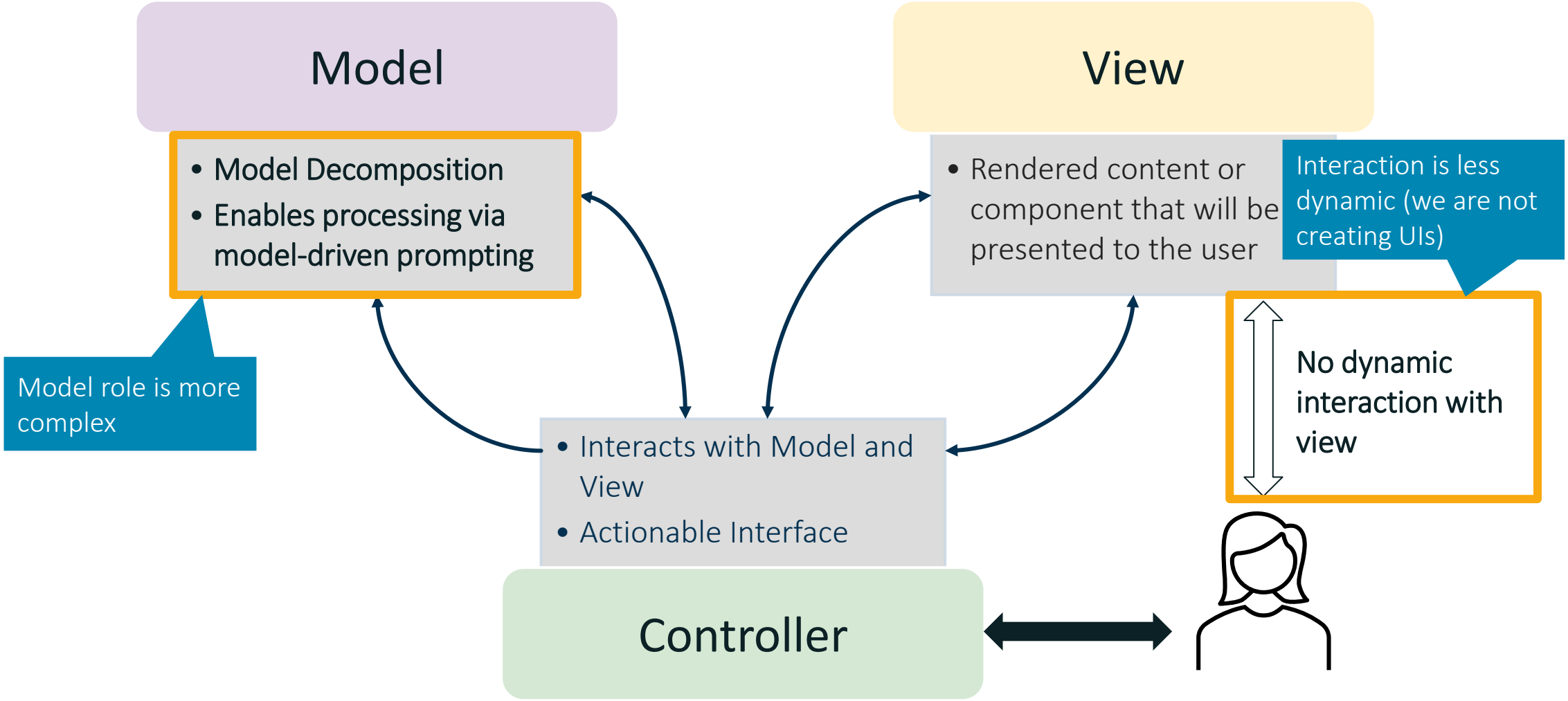
How does MVC help?

Clean structure	• Application is organized into three **distinct components**, each with a **clear scope** • Easy for developers to follow the pattern → clean code
Separation of Concerns	• Responsiblity is distributed between the components • Components can be developed, tested and **maintained individually** from each other
Reusability + Optimization	• Models, Views and transformation functions between them can potentially be re-used • Parts of the implementation can be **validated and optimized individually**

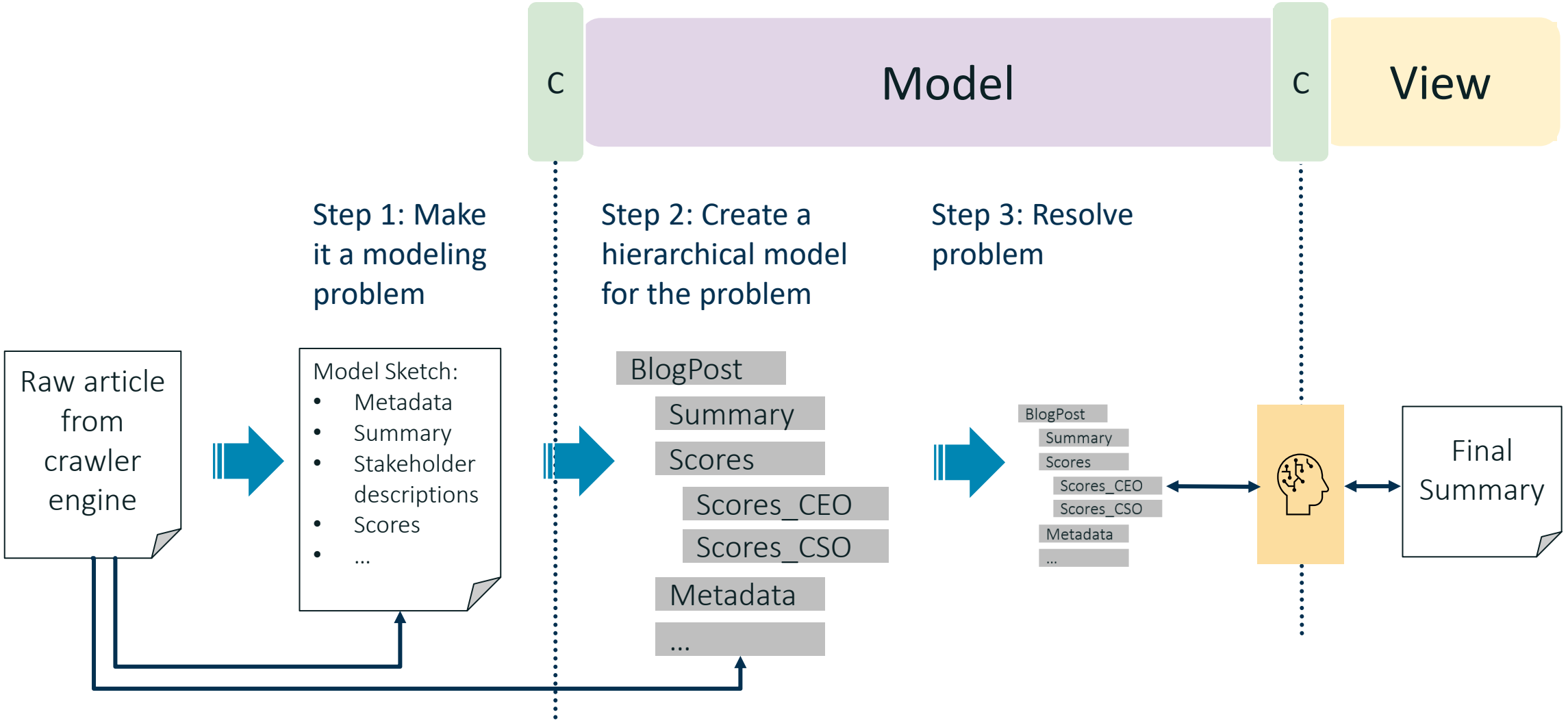
LLM-MVC


Model
• Model Decomposition
• Enables processing via model-driven prompting
View
• Rendered content or component that will be presented to the user
No dynamic interaction with view
• Interacts with Model and View
• Actionable Interface
Controller

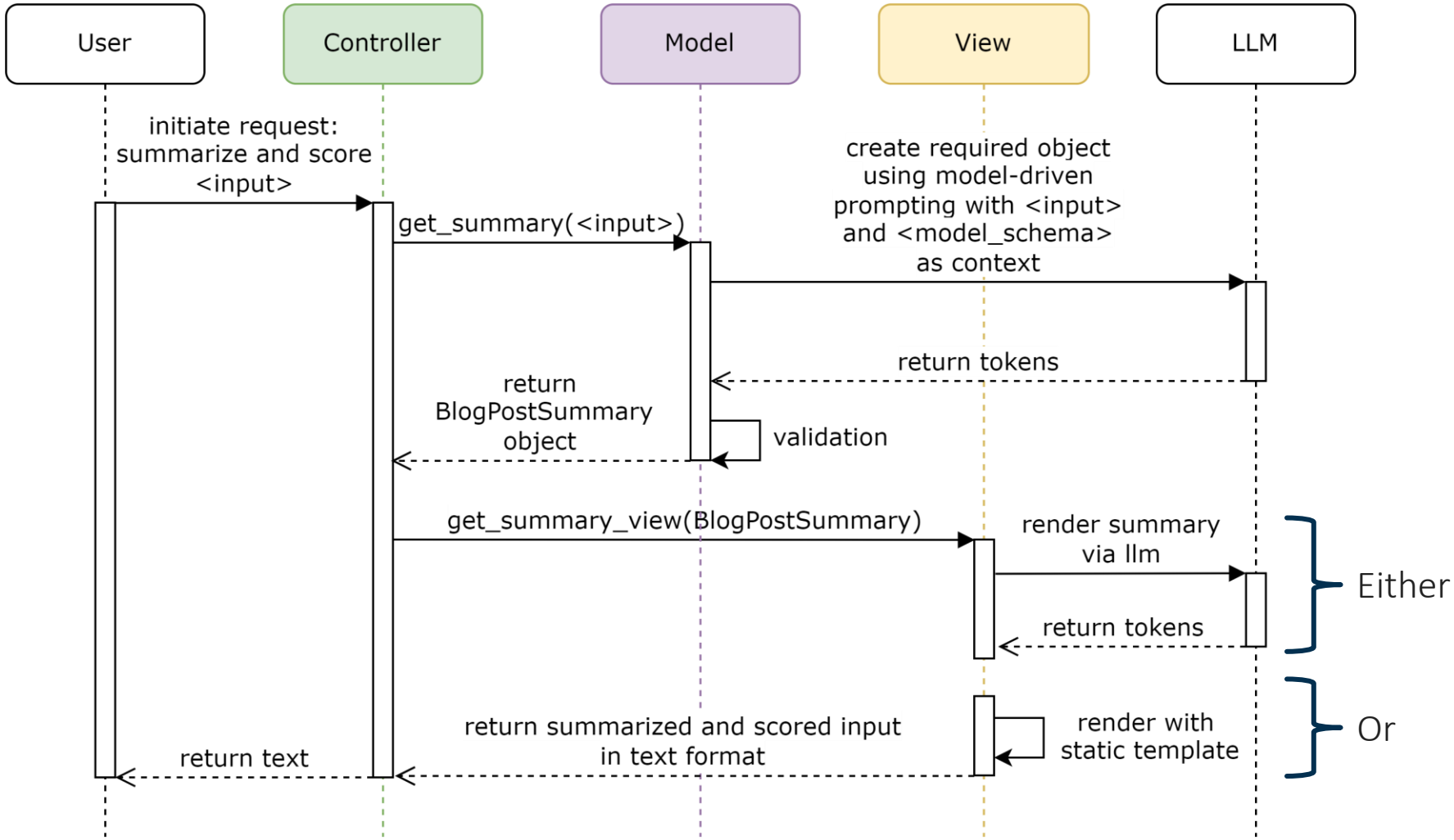
LLM-MVC


Model
• Model Decomposition
• Enables processing via model-driven prompting
Model role is more complex
View
• Rendered content or component that will be presented to the user
Interaction is less dynamic (we are not creating UIs)
No dynamic interaction with view
• Interacts with Model and View
• Actionable Interface
Controller

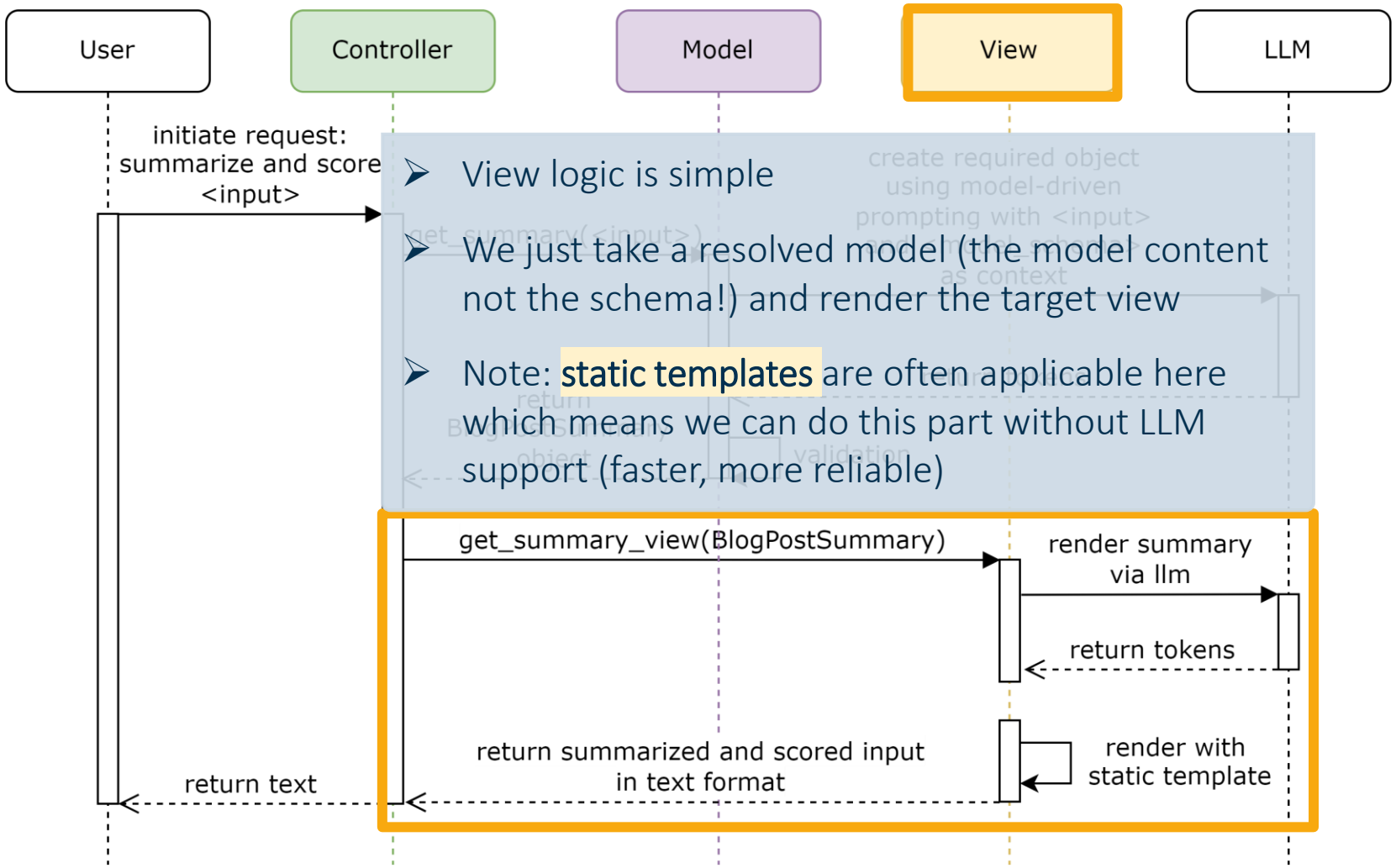
Example Workflow with LLM-MVC



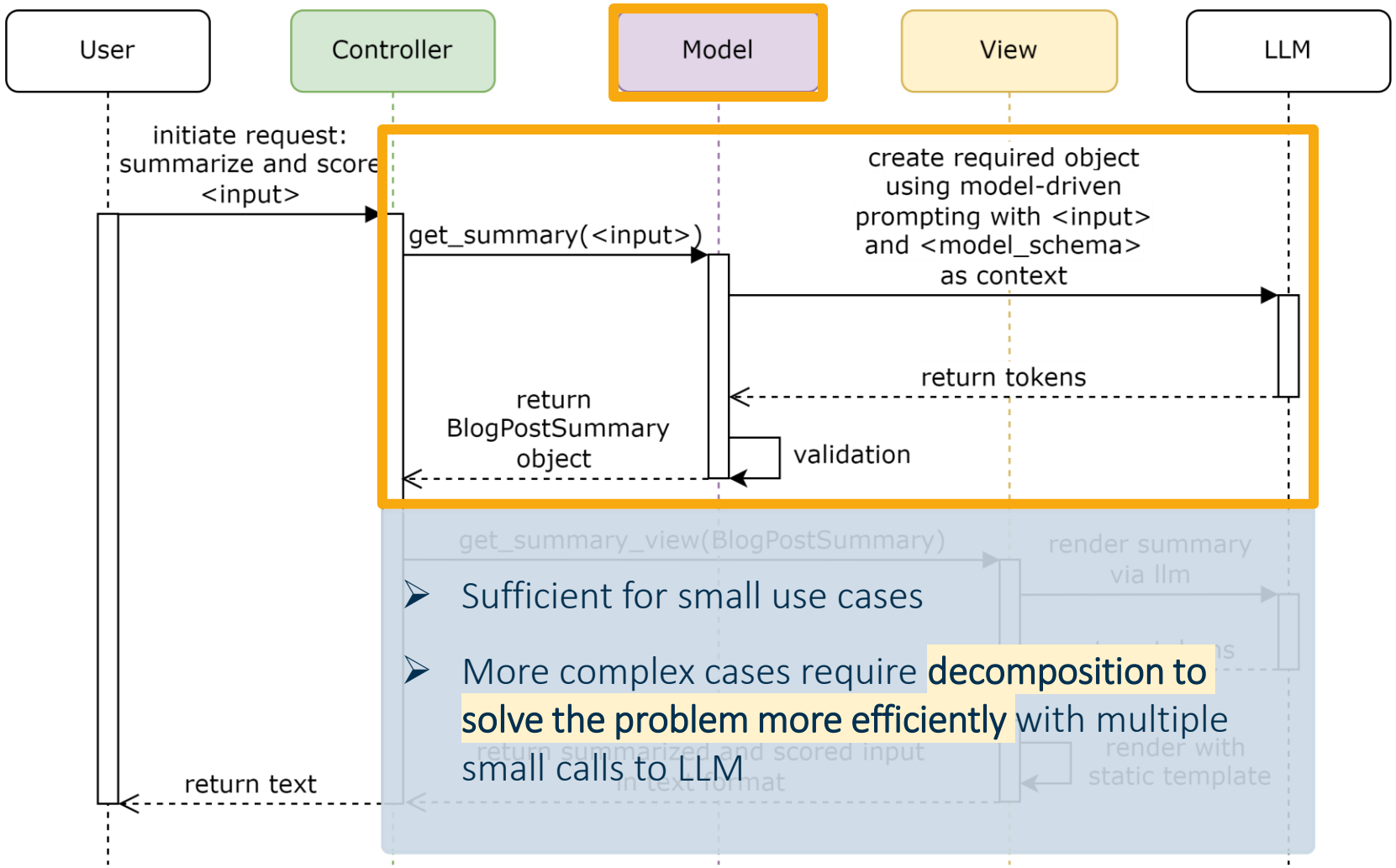
Example Workflow with LLM-MVC



Example Workflow with LLM-MVC



Example Workflow with LLM-MVC



- Sufficient for small use cases
- More complex cases require **decomposition to solve the problem more efficiently** with multiple small calls to LLM

05 | Model Decomposition

Decomposition and Indirect Prompting

Decomposition

- Fundamental principal in software engineering
- „Breaking a complex problem or system into parts that are easier to conceive, understand, program, and maintain” (from Wikipedia)
  https://en.wikipedia.org/wiki/Decomposition_(computer_science)

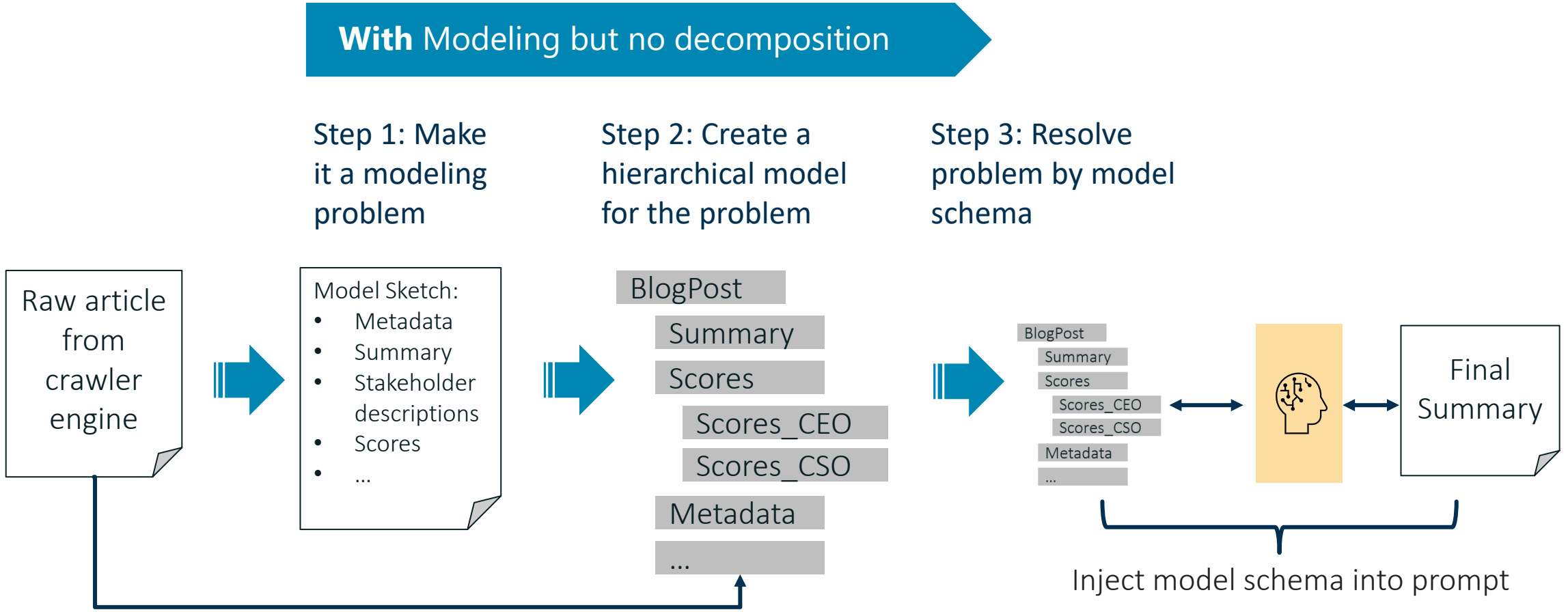
Indirect Prompting

- Combines model driven prompting and decomposition
- **Step 1**: Make your problem a modeling problem
- **Step 2**: Create a domain model and apply decomposition (e.g., via hierarchical modeling)
- **Step 3**: Resolve problem step by step ➔ take some „indirections“ to get to the final results

Recap

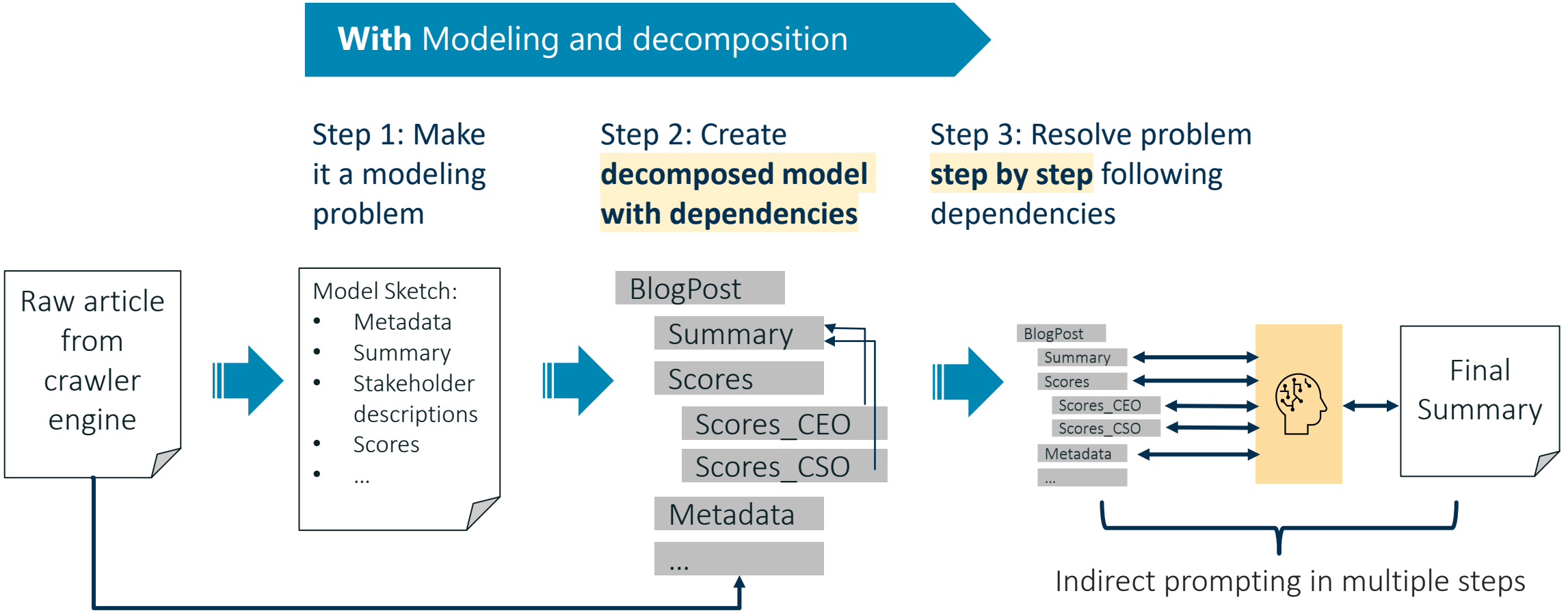
Model-driven Prompting

Task: Summarize tech blog posts and perform relevance scoring for different stakeholders (CEO, CSO, …)



Indirect Prompting

Task: Summarize tech blog posts and perform relevance scoring for different stakeholders (CEO, CSO, …)



Model Decomposition

Core Idea

- We do NOT resolve the complete model in one batch (this is simple)
- Instead we **individually resolve the parts** step by step

Why not in one batch?

- Schemas can get large quickly for complex (real world) use cases
- From our experience, even the latest LLM generations **struggle with large schemata** with lots of annotations
- **Limited control** over individual parts of the model

Dependencies between Models

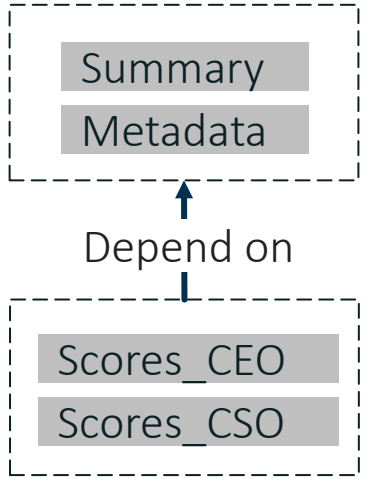
Data Model:

- BlogPost
  - Summary
  - Scores
    - Scores_CEO
    - Scores_CSO
  - Metadata
  - …

Dependency Modeling

- Dependencies tell the resolver what a model needs as input
- This is NOT the hierarchy of the model
- Dependencies are an **explicit modeling decision**
- Dependencies define the context that is used together with the schema of a composite element

Modeled Dependencies:



Wrap-Up

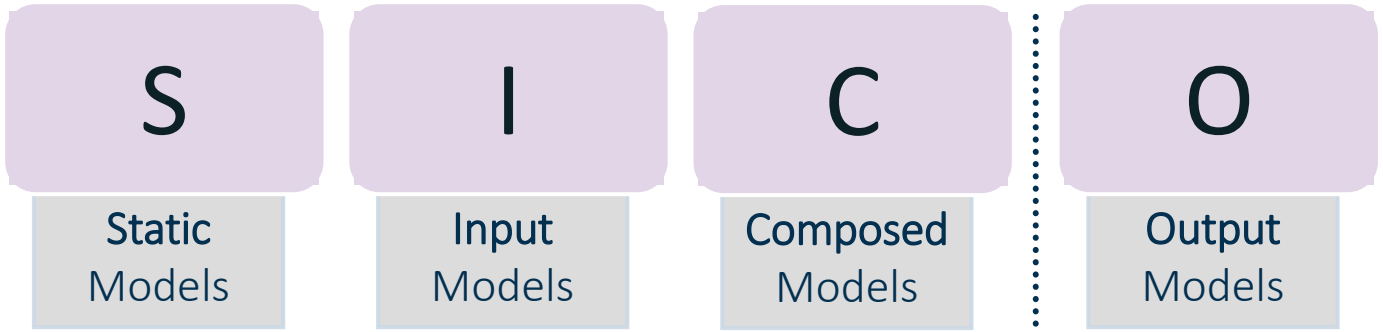
Dependency Management

- There are various different ways how data modeling and dependency management can be done
- Far beyond this talk
- Keep in mind: you can also use the pattern without decomposition to keep things simple

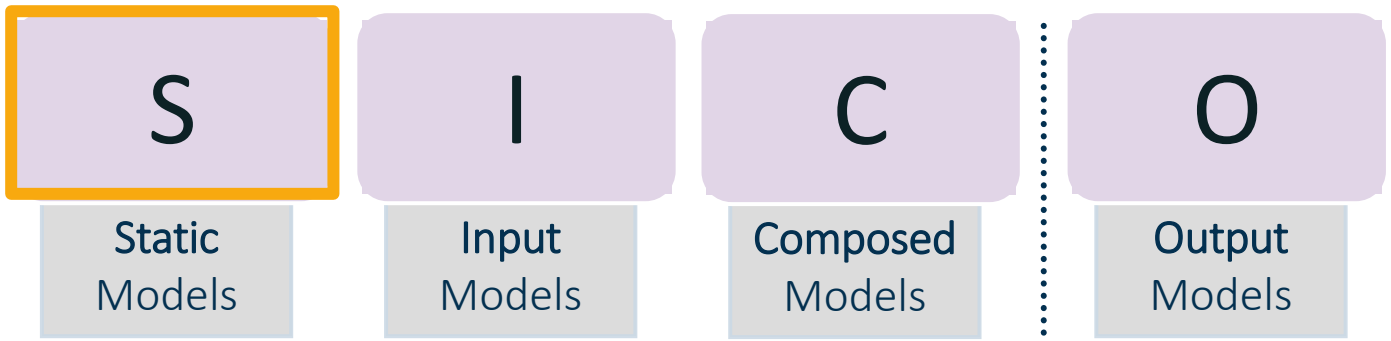
Thinking in „Model Types“

- To get started with decomposition, it can be very helpful to explicitly distinguish models based on their conceptual role
- Next: The basic modeling approach we use for our projects

Model Types

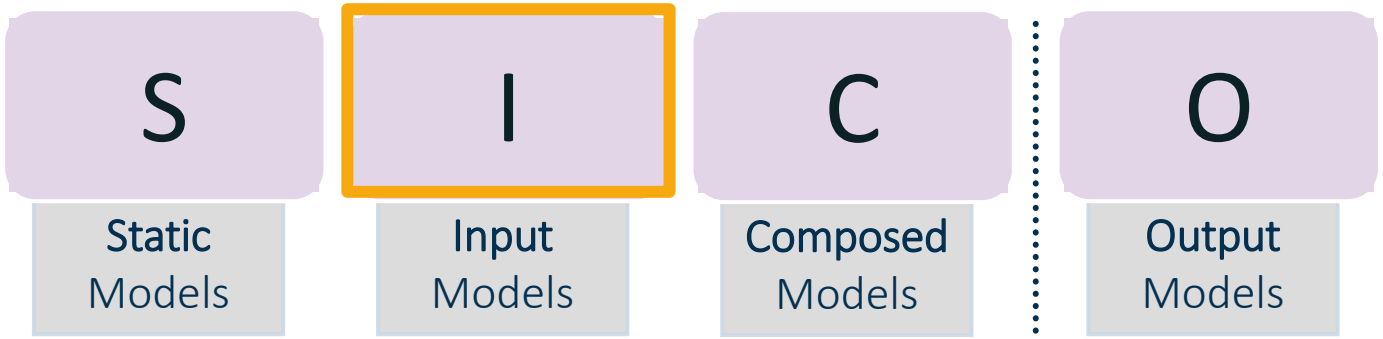

S
I
C
O
Static Models
Input Models
Composed Models
Output Models

Model Types



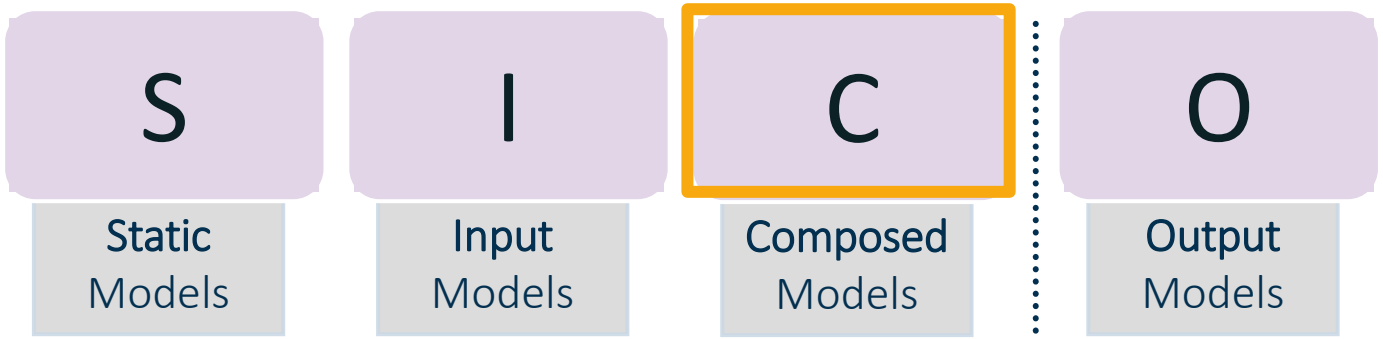
- Describe static a-priori information
- Only used as additional LLM context
- **Never resolved** by the LLM (=LLM never creates an object with schema of this model)
- May be filled from database or RAG system

Model Types



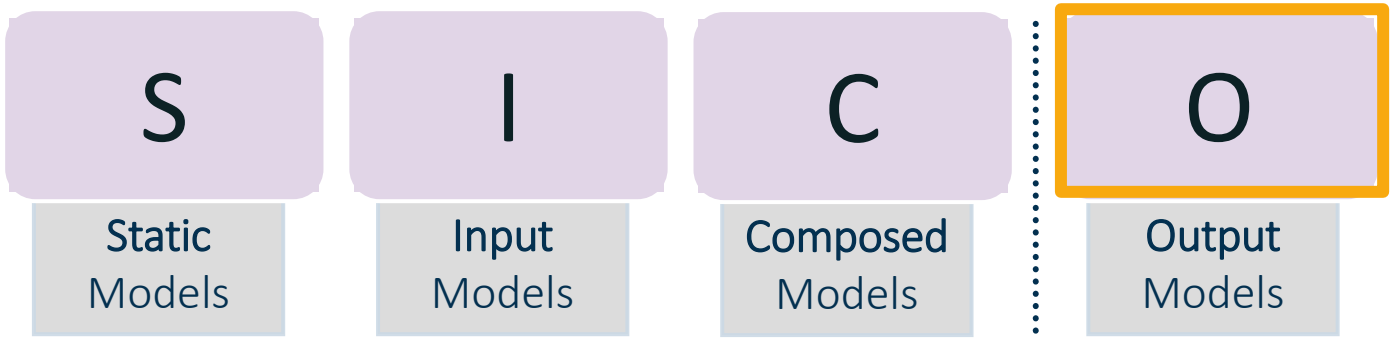
- Models that are **actively resolved** by an LLM with model-driven prompting
- Means the schema of the model with annotations is set as “output format”
- Are always resolved with their complete schema which is different from how composed models are treated

Model Types



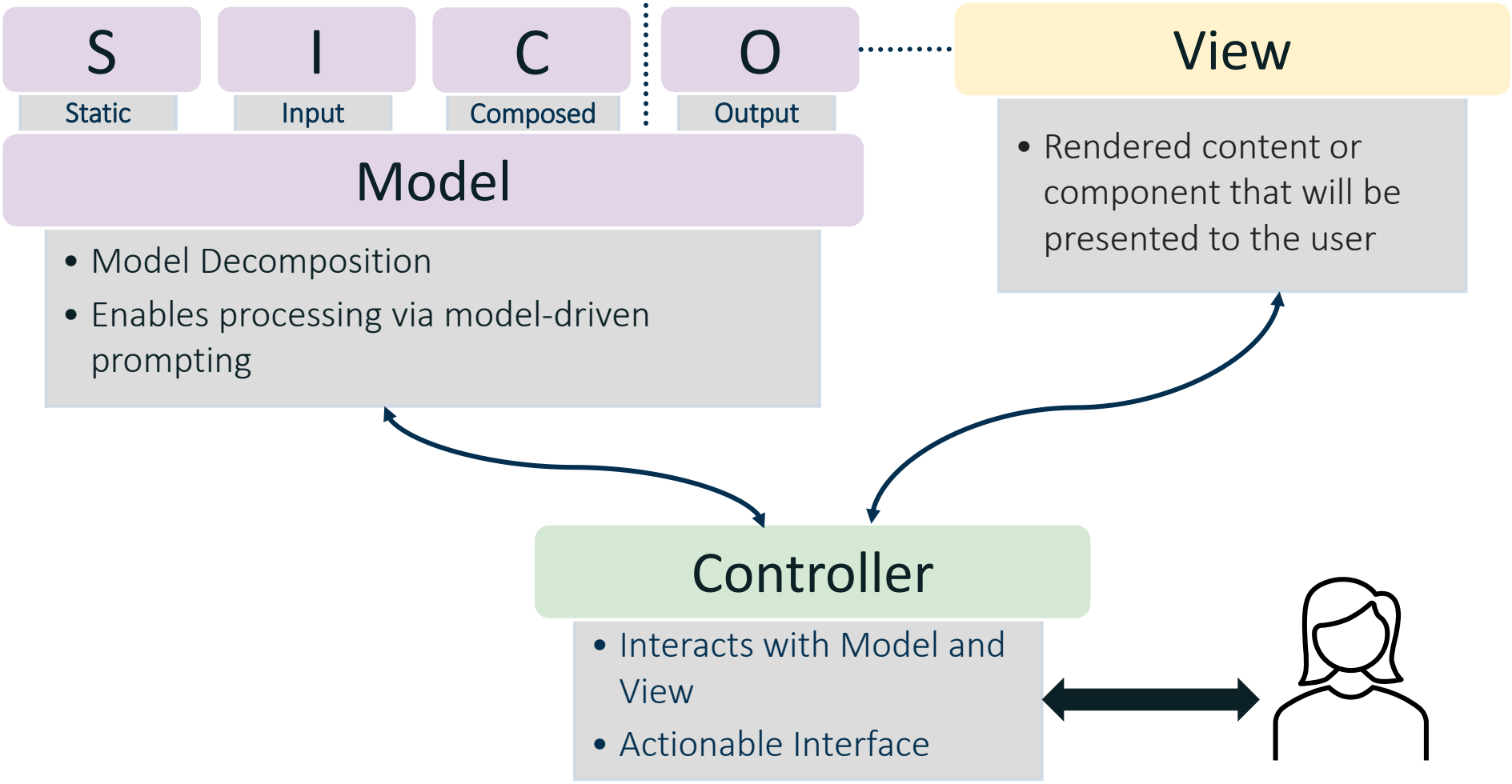
- Models that are not directly resolved due to size/complexity
- Instead, sub-models of the composite are **resolved recursively** after some form of dependency resolution has taken place
- Can consist of static, input and other composed models

Model Types

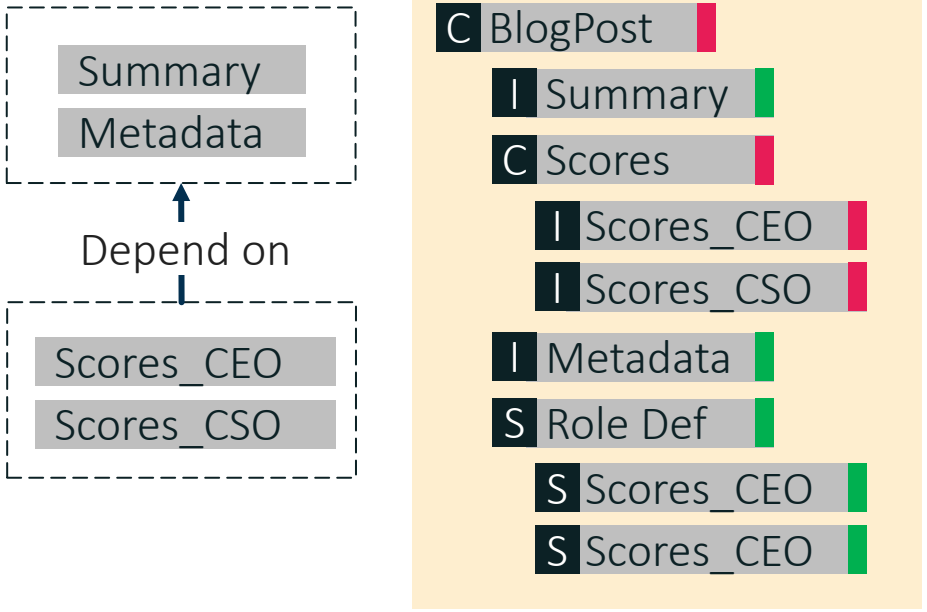


- Models that are derived for view construction
- **Never resolved** by the LLM; if something has to be resolved, its either in the I or the C category!
- Usually a reduced version of one of the other model types

LLM-MVC Overview with Model Types



Step by Step



S Static Models

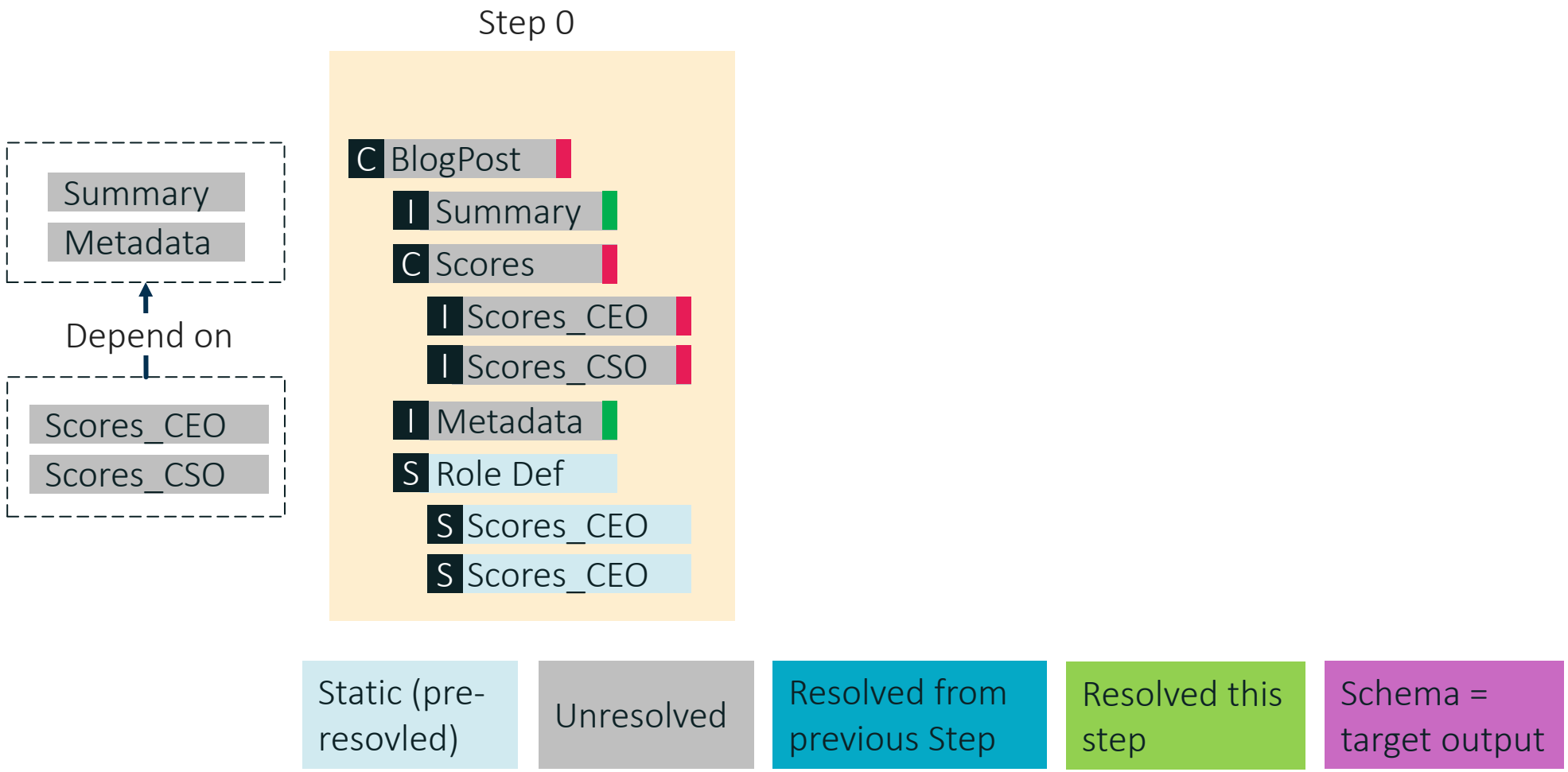
I Input Models

C Composed Models

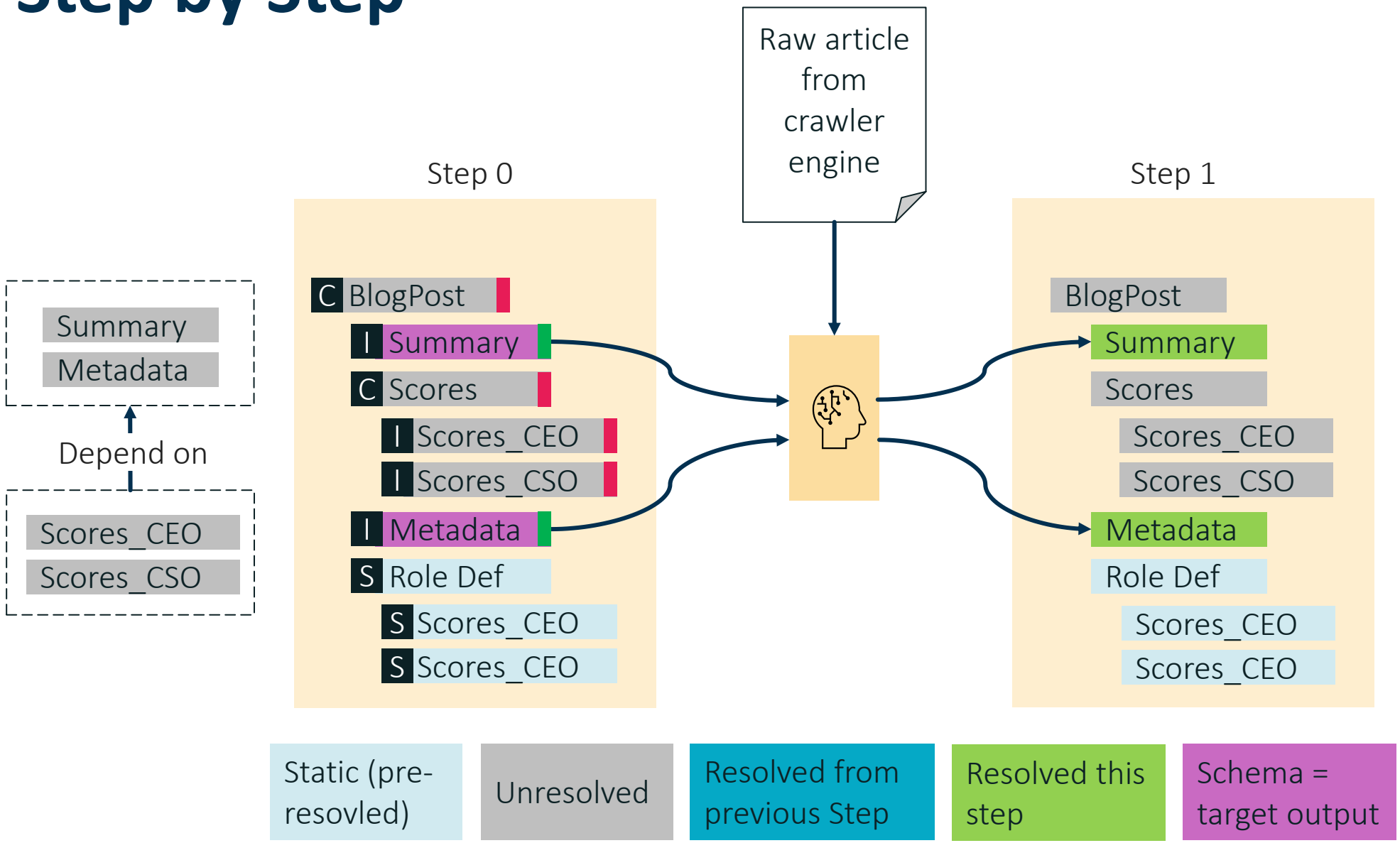
Without Dependencies

With Dependencies

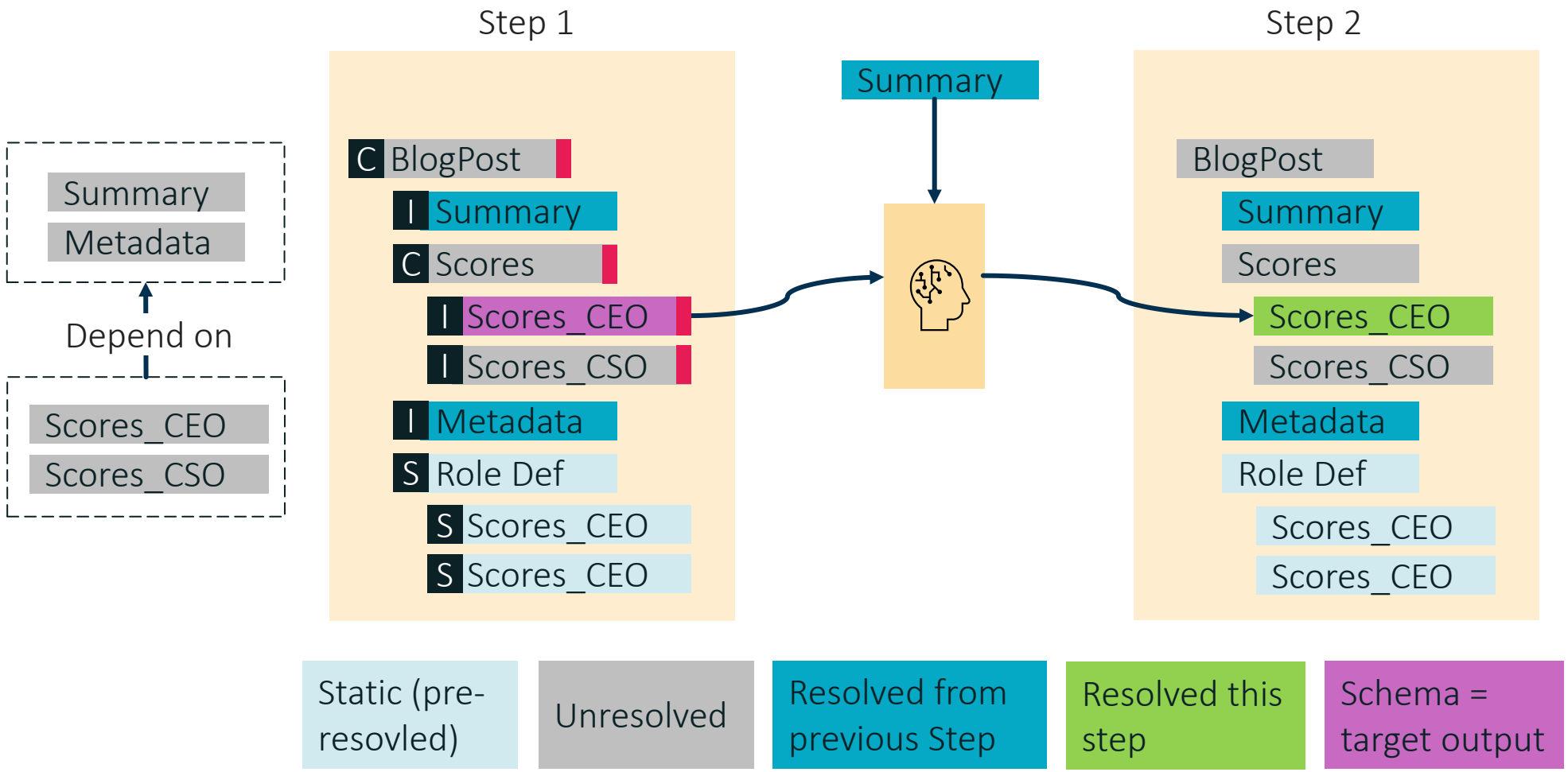
Step by Step


Step 0
Summary
Metadata
Depend on
Scores_CEO
Scores_CSO
C BlogPost
I Summary
C Scores
I Scores_CEO
I Scores_CSO
I Metadata
S Role Def
S Scores_CEO
S Scores_CEO
Static (pre-resovled)
Unresolved
Resolved from previous Step
Resolved this step
Schema = target output

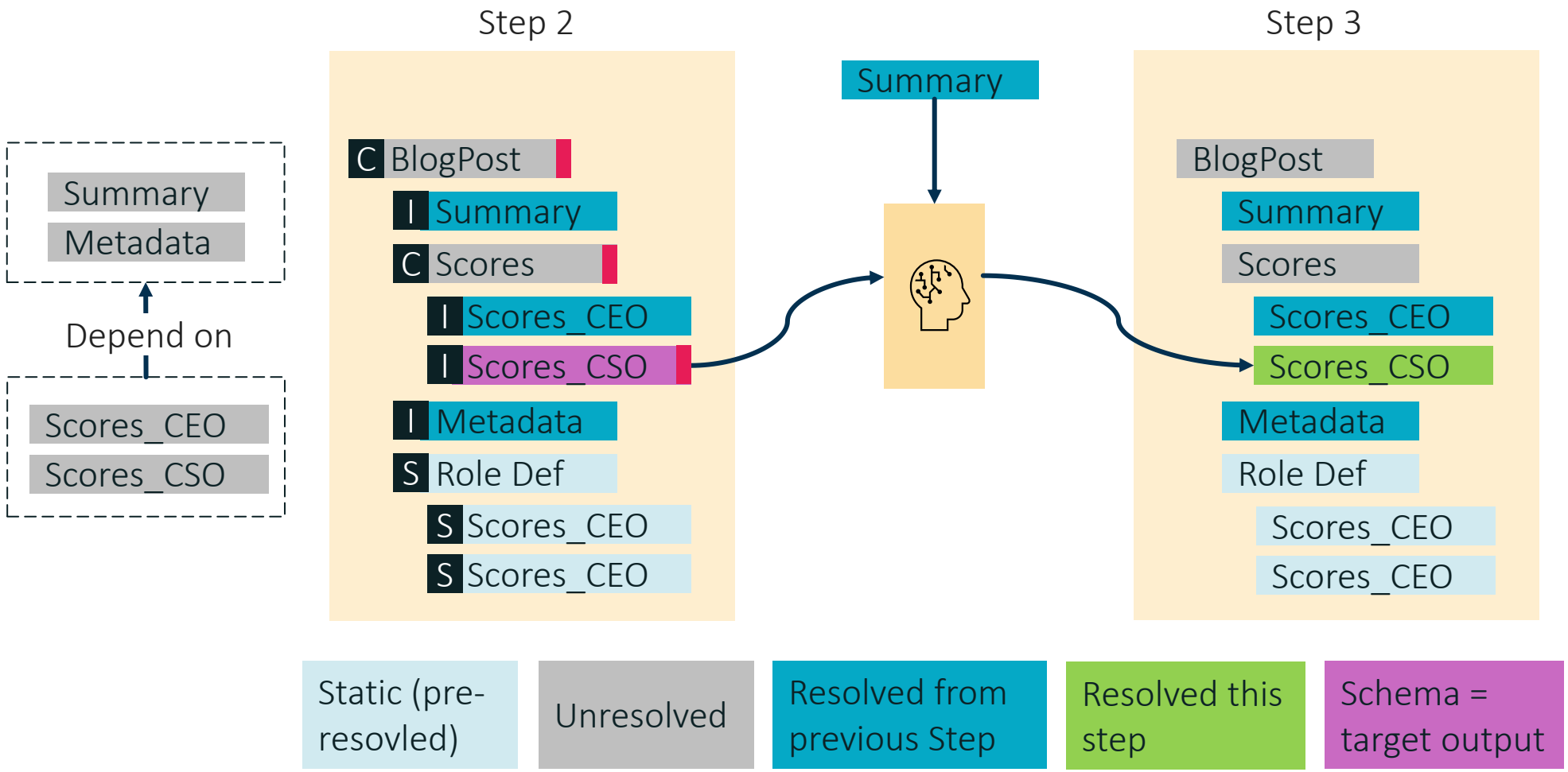
Step by Step


Raw article from crawler engine
Step 0
Step 1
Summary
Metadata
Depend on
Scores_CEO
Scores_CSO
C BlogPost
I Summary
C Scores
I Scores_CEO
I Scores_CSO
I Metadata
S Role Def
S Scores_CEO
S Scores_CEO
BlogPost
Summary
Scores
Scores_CEO
Scores_CSO
Metadata
Role Def
Scores_CEO
Scores_CEO
Static (pre-resovled)
Unresolved
Resolved from previous Step
Resolved this step
Schema = target output

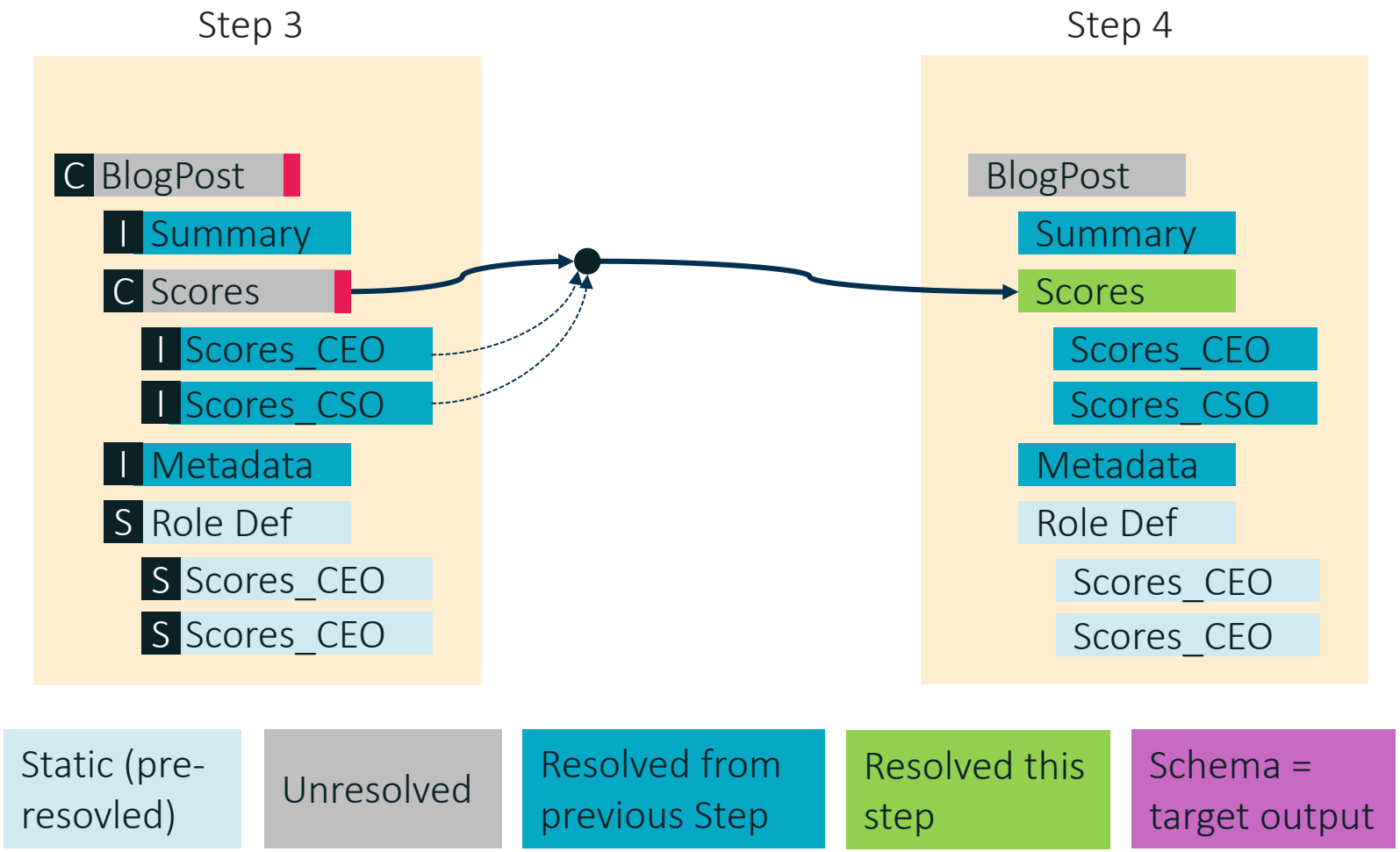
Step by Step


Step 1
Step 2
Summary
Metadata
Depend on
Scores_CEO
Scores_CSO
Summary
C BlogPost
I Summary
C Scores
I Scores_CEO
I Scores_CSO
I Metadata
S Role Def
S Scores_CEO
S Scores_CEO
BlogPost
Summary
Scores
Scores_CEO
Scores_CSO
Metadata
Role Def
Scores_CEO
Scores_CEO
Static (pre-resovled)
Unresolved
Resolved from previous Step
Resolved this step
Schema = target output

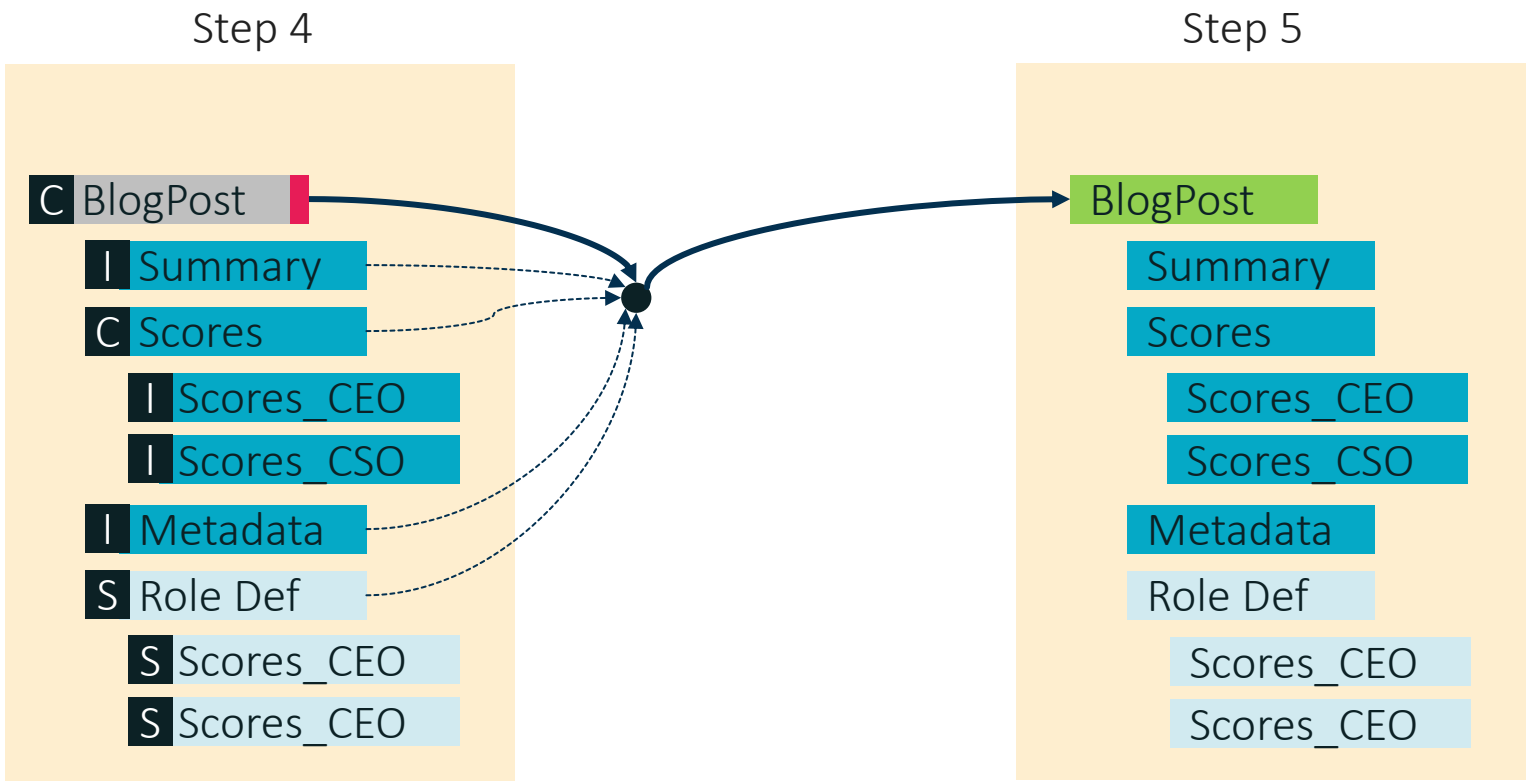
Step by Step


Summary
Metadata
Depend on
Scores_CEO
Scores_CSO
Step 2
C BlogPost
I Summary
C Scores
I Scores_CEO
I Scores_CSO
I Metadata
S Role Def
S Scores_CEO
S Scores_CEO
Summary
Step 3
BlogPost
Summary
Scores
Scores_CEO
Scores_CSO
Metadata
Role Def
Scores_CEO
Scores_CEO
Static (pre-resovled)
Unresolved
Resolved from previous Step
Resolved this step
Schema = target output

Step by Step


Step 3
C BlogPost
I Summary
C Scores
I Scores_CEO
I Scores_CSO
I Metadata
S Role Def
S Scores_CEO
S Scores_CEO
Step 4
BlogPost
Summary
Scores
Scores_CEO
Scores_CSO
Metadata
Role Def
Scores_CEO
Scores_CEO
Static (pre-resovled)
Unresolved
Resolved from previous Step
Resolved this step
Schema = target output

Step by Step



Static (pre-resovled)

Unresolved

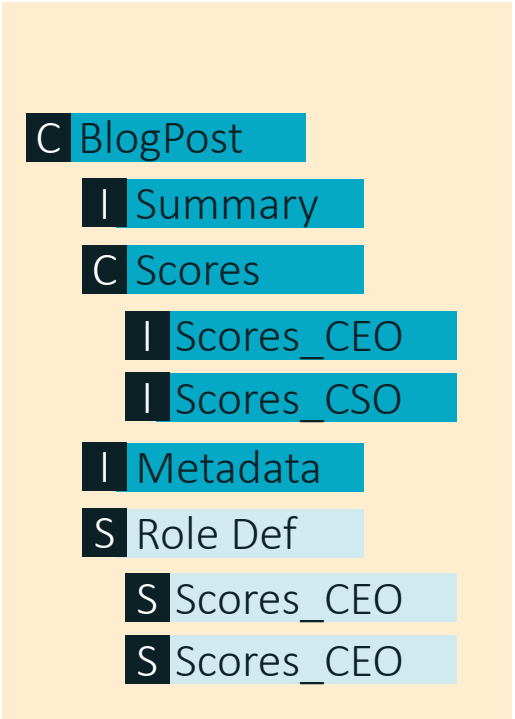
Resolved from previous Step

Resolved this step

Schema = target output

Step by Step

Step 5



Some important observations

- Subsequent calls can use smaller context windows (i.e., not the complete article)
- Smaller context = more control / easier for the LLM to follow instructions/annotations
- Steps can be validated individually!
- Not all models are resolved by LLM → Different types of models
- Models can have dependencies: Scores_XYZ depend on summary, for example
- If parts are changed, only dependent parts need to be updated!
- Process can be fully automated for generic cases (no real need for chains/agents here)

Static (pre-resovled) | Unresolved | Resolved from previous Step | Resolved this step | Schema = target output

06 | Code Examples

07 | Lessons Learned and Summary

Lessons Learned

Patterns and Templates do help

- We frequently use model-centric approaches
- Robust and maintainable
- Easy to explain + Easy to get started with templates

Simplicity is key

- Frameworks such as langchain can be overkill
- LLM-MVC pattern can be build with requests and pydantic alone and is still an powerful abstraction

Partial re-evaluation is a big issue for us

- If small parts of an application are changed, we do not have to run everything from scratch
- Decomposition = re-run only changed + dependent

Partial re-evaluation

Everything New

Costs with Composition		
Scoring1	435	98
Scoring2	605	60
Basics	2002	597
Tags	413	33
Summe	3455	788

+25% +35%

Without Composition		
Summe	2752	585

New Metadata

Costs with Composition		
Scoring1	435	98
Scoring2	605	60
Basics	2002	597
Tags	0	0
Summe	3042	755

+10% +30%

Without Composition		
Summe	2752	585

New Scoring Algo

Costs with Composition		
Scoring1	435	98
Scoring2	605	60
Basics	0	0
Tags	0	0
Summe	1040	158

-63% -73%

Without Composition		
Summe	2752	585

New Tag Algo

Costs with Composition		
Scoring1	0	0
Scoring2	0	0
Basics	0	0
Tags	413	33
Summe	413	33

-85% -95%

Without Composition		
Summe	2752	585

Further Notes and Conclusion

Streaming?	• So far we primarily applied the pattern to batch use cases / use cases without streaming requirements • Streaming + Decomposition do not work well together out-of-the-box (probably still possible)
Use in existing projects	• Depends; basic ideas can be integrated easily • More effort if you truly want to benefit from decomposition
Frameworks	• Approach is framework independent • Can be used with or without major frameworks • We do both in our projects

HMS analytical software

Vielen Dank für die Aufmerksamkeit!

Dr. Robert Bauer
Senior Data Scientist
HMS Analytical Software GmbH

HMS
analytical software