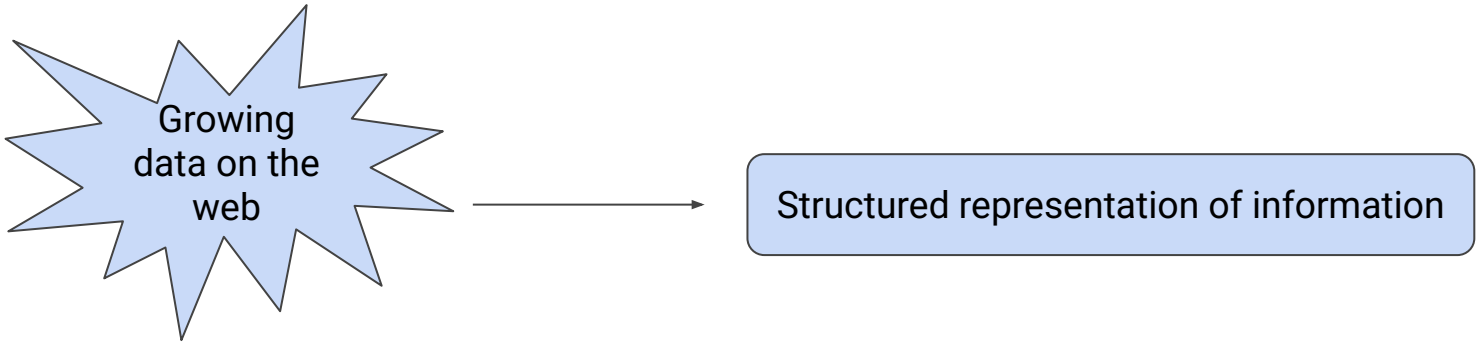
Enabling Automated Setup and Evaluation of SPARQL Engines through Unified Benchmarking Infrastructure

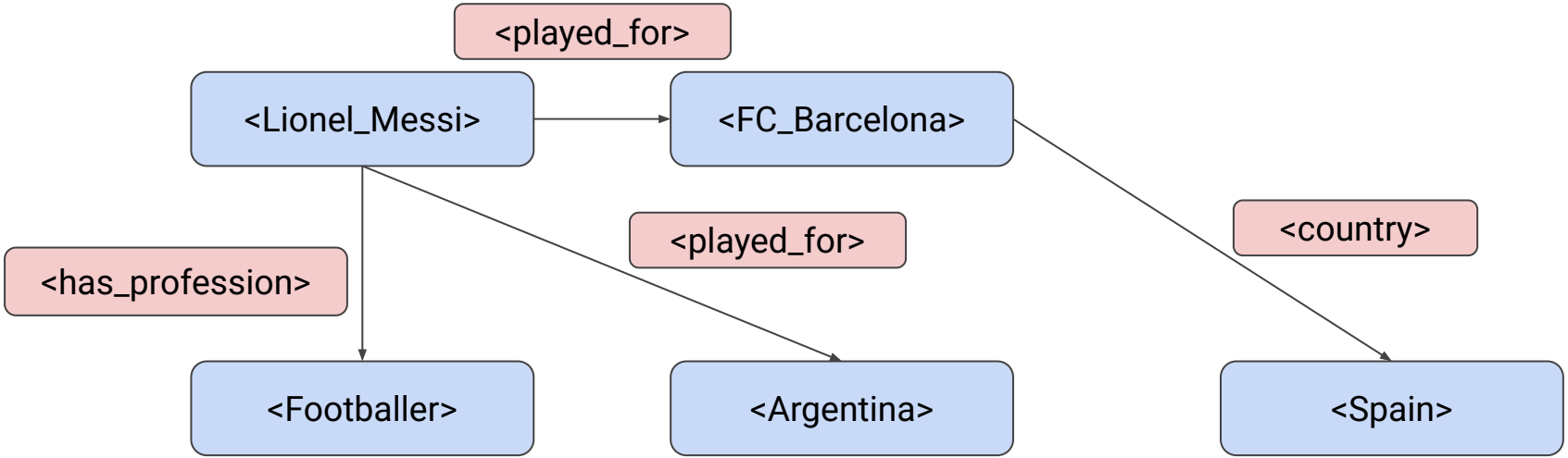
Master’s Thesis by Tanmay Garg

Resource Description Framework (RDF)



<subject>	<predicate>	<object>
<Lionel_Messi>	<has_profession>	<Footballer>
<Lionel_Messi>	<played_for>	<Argentina>
<Lionel_Messi>	<played_for>	<FC_Barcelona>
<FC_Barcelona>	<country>	<Spain>

Graph Representation of Triples - Knowledge Graphs



Querying Knowledge Graphs with SPARQL

Standard query language to extract information from Knowledge Graphs

E.g.: All footballers who have played for both Argentina and FC Barcelona

```
SELECT ?player
WHERE {
  ?player  <has_profession>  <Footballer>  .
  ?player  <played_for>  <Argentina>  .
  ?player  <played_for>  <FC_Barcelona>  .
}
```

SPARQL Engines

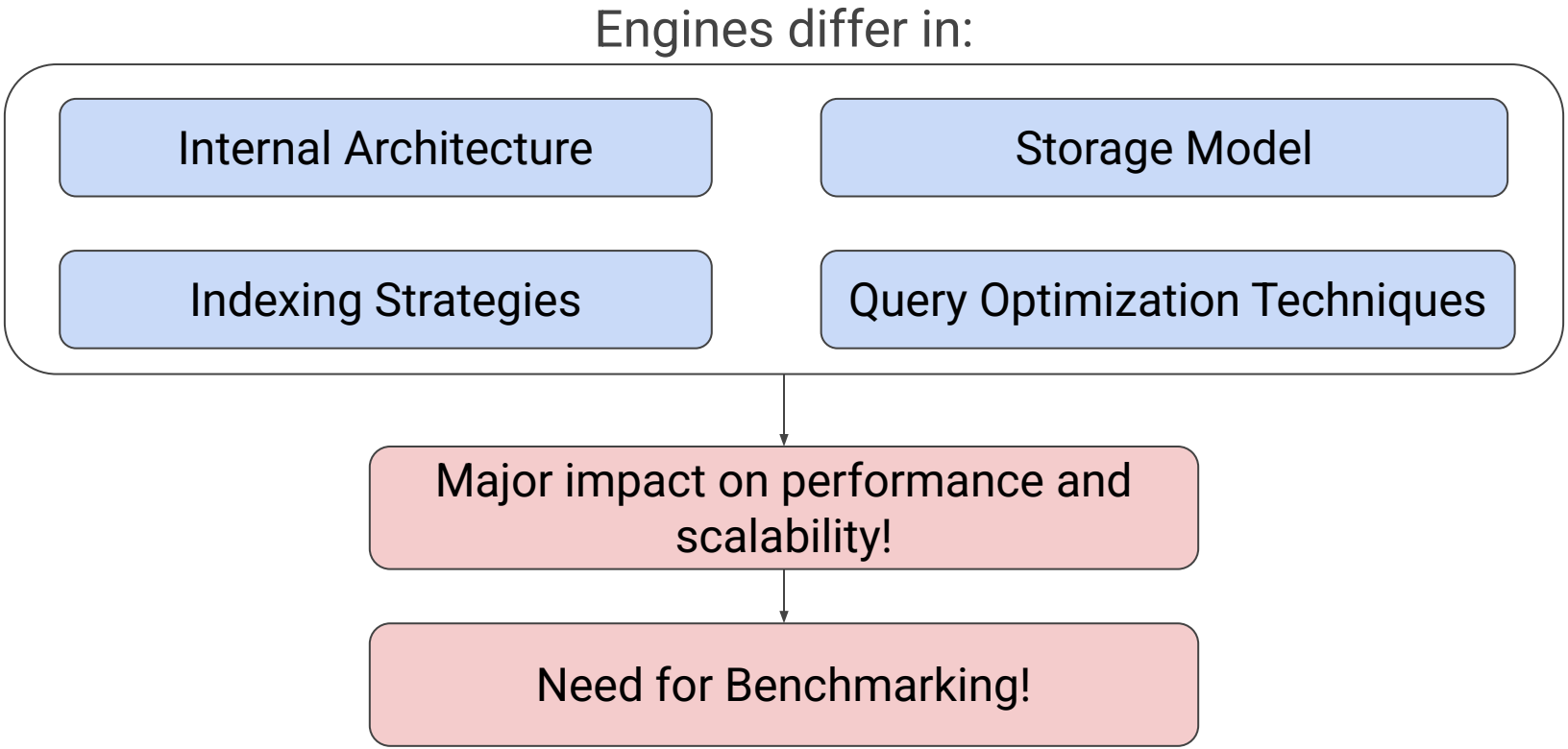
Responsibilities:

- Store and index RDF data
- Process SPARQL queries
- Optimize execution of queries at scale

Examples on the market:

- QLever
- Virtuoso
- MillenniumDB
- Blazegraph
- Apache Jena
- GraphDB
- Oxigraph

Not all SPARQL Engines are the same!



What is a Benchmark?

- Structured and reproducible evaluation of SPARQL engines
- Focus on evaluation of sequential query performance

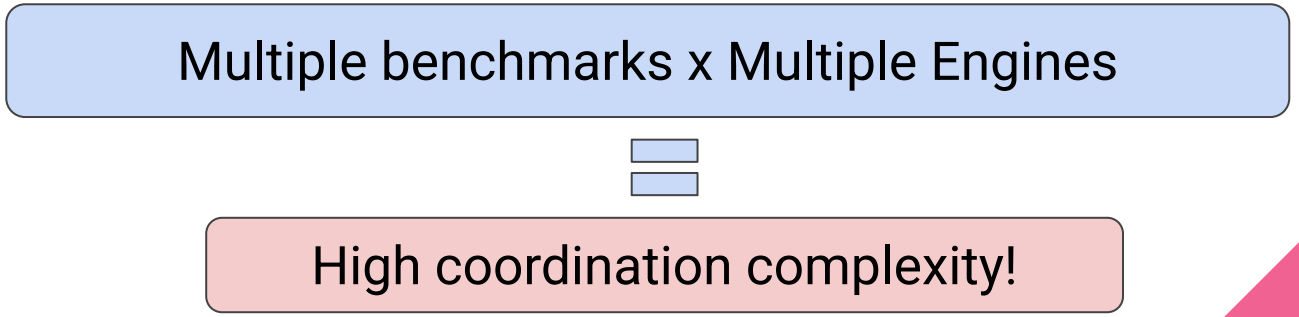
But, benchmarking is challenging!

Challenge 1: Working with multiple engines

- Different installation, and dataset indexing methods
- Complex, and often poorly documented commands
- Configuration quirks, which significantly affect performance
- Difficulty grows when benchmarking multiple engines

Challenge 2: Working with multiple benchmarks

- Some benchmarks provide predefined queries
- Some benchmarks provide software to generate queries from query templates
- Burden of query execution and result collection left to the user



Challenge 3: Interpreting Performance Metrics

- Benchmarks often output raw tables of numbers
- Manual processing needed for meaningful insights
- Limited support for side-by-side engine comparison
- Correctness matters, not just speed

Questions?

What if we could index and serve RDF datasets for multiple engines using simple, uniform commands without worrying about low-level internal details?

Building on a Strong Foundation: **QLever-control**

7 Engines

1 Workflow

```
<qengine> setup-config
<qengine> get-data
<qengine> index
<qengine> start
<qengine> query
<qengine> stop
```

No engine-specific friction!

Modular, easy-to-extend with engine-specific logic isolation

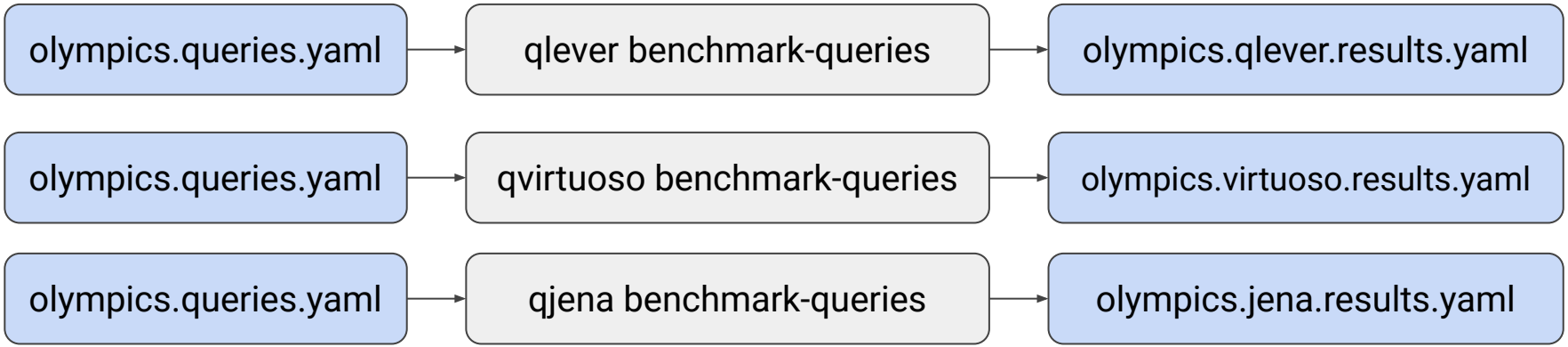
What if we could have a single, uniform way to execute a set of benchmark queries for all the engines?

Uniform query execution across multiple engines with:

`<qengine> benchmark-queries`

The `benchmark-queries` command

- Sequentially query the engines and store the runtimes and results
- Single implementation for QLever
  - Other engines inherit from QLever's implementation and simply override the default SPARQL endpoint URL



Questions?

What if benchmark results were easier to interpret,
with side-by-side comparisons and correctness checks?

The Evaluation Web Application

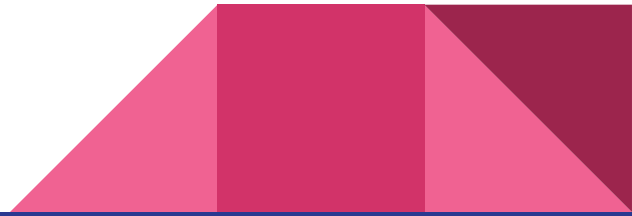
SPARQL Engine Evaluation Setup

7 Engines, 8 benchmarks:

- 3 benchmarks at small scale (~ 50 million triples)
- 3 benchmarks at medium scale (~ 500 million triples)
- 2 benchmarks at large scale (~ 8 billion triples)

Single machine

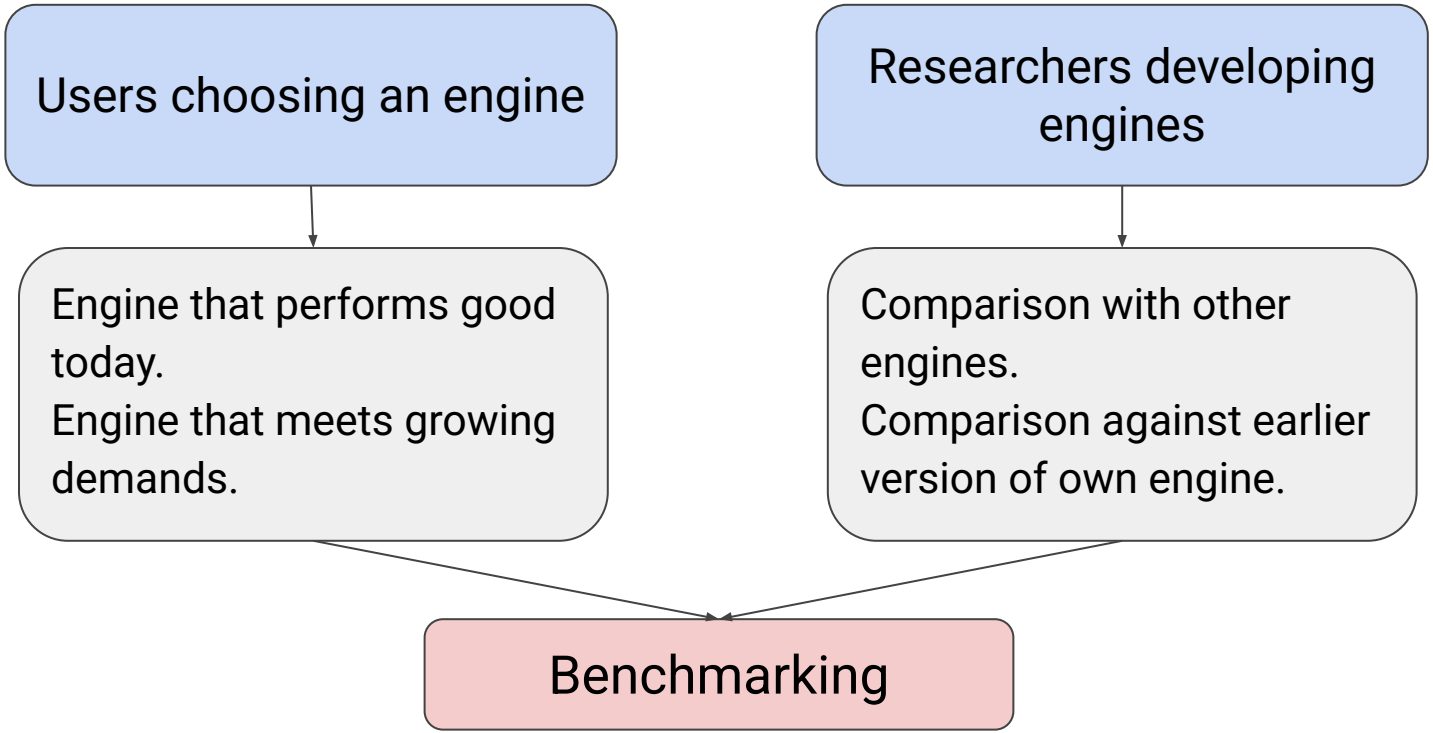
- AMD Ryzen 9 5900X CPU (12 cores, 24 threads, 3.7 GHz)
- 128 GiB of DDR4 memory
- 3.6 TB NVMe SSD storage



Thank You

Additional backup slides

Evaluation Use Cases



What if benchmarking could be easier?

What if we could index and serve RDF datasets for multiple engines using simple, uniform commands without worrying about internal details?

What if we could have a single, uniform way to execute a set of queries for all the engines?

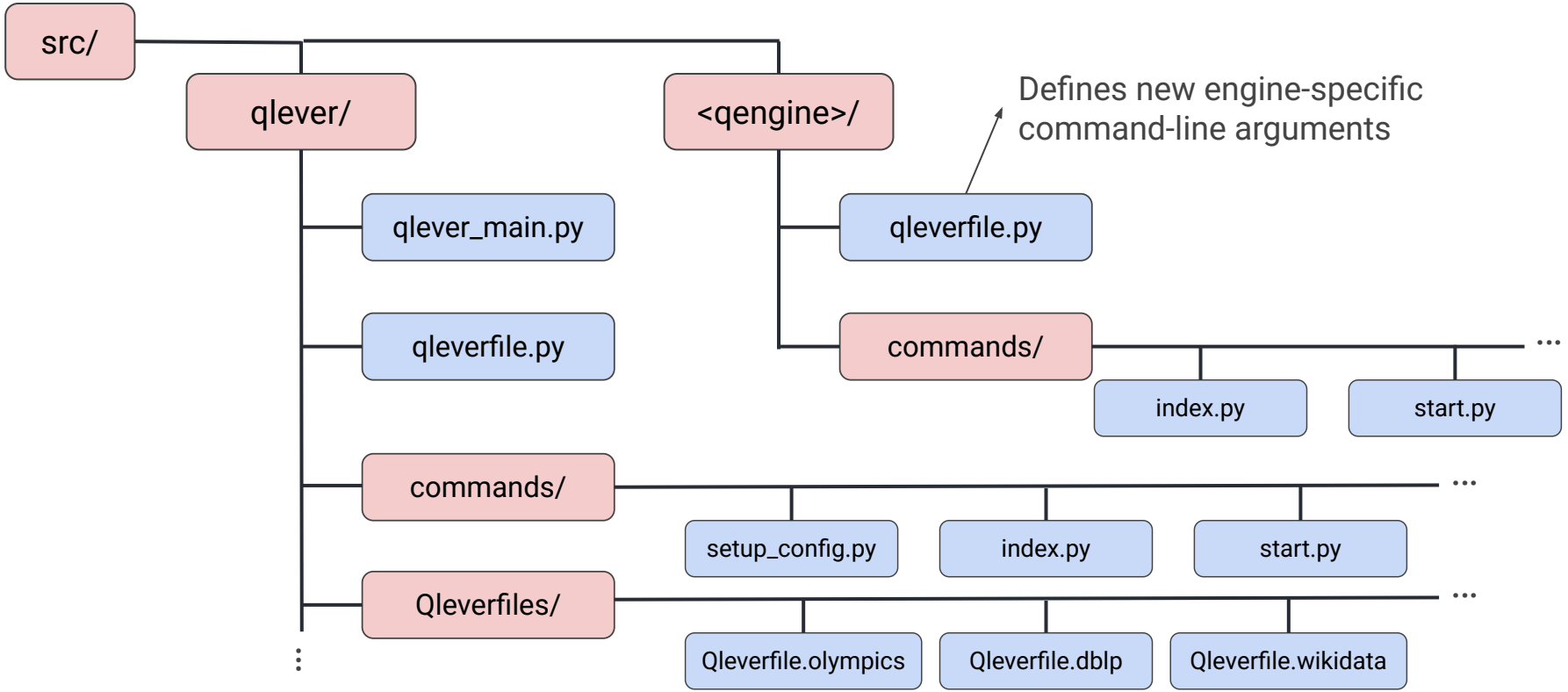
What if benchmark results were easier to interpret,
with side-by-side comparisons and correctness checks?

Building on a strong foundation: **QLever-control**

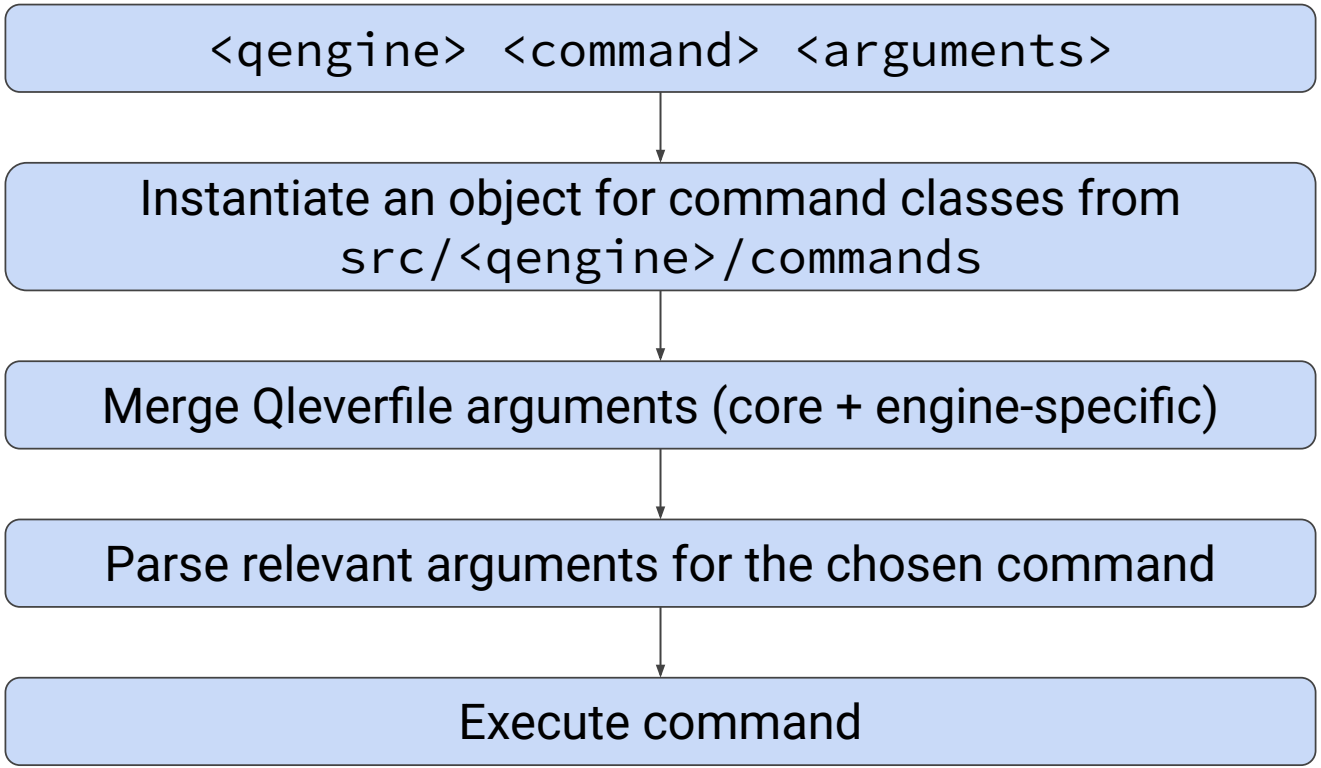
- Simple, uniform commands to set up QLever
- No need to know internal engine commands!
- Underlying execution steps can be optionally shown
- Modular design which is easy to extend
- Supports both native binaries and containers

How QLever-control was modified to support multiple engines

QLever-control directory structure



Program Execution Flow



Command Implementation Strategy

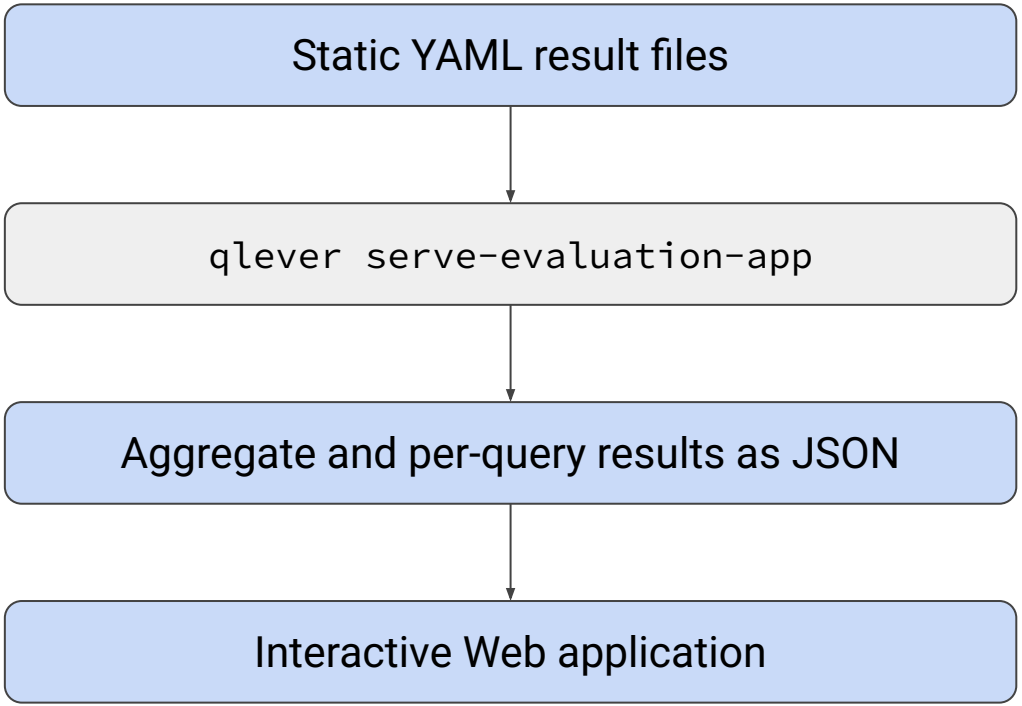
GOAL: Minimize code duplication across engines

Engine-agnostic	Derived	Non-reusable
Identical across engines	Mostly shared logic	Highly engine-specific
`get-data`	`setup-config`	`index`
`log`	`stop`	`start`
	`query`	

Benefits of the Modular Design

- Minimal changes to core QLever-control
- Clear separation of engine-specific code
- Reduced code duplication
- Easy to extend to new engines
- Better code discovery & maintainability

From Static Results to Interactive Evaluation



Benchmarks at 3 different scales for 7 engines

	Benchmark	Dataset	Triples
S	SP^{2}Bench v1.1	SP^{2}Bench data-generator	~ 50 million triples
	Sparqloscope SP^{2}Bench	SP^{2}Bench data-generator	~ 50 million triples
	Watdiv v0.6	Watdiv data-generator	~ 55 million triples
M	SP^{2}Bench v1.1	SP^{2}Bench data-generator	~ 500 million triples
	Sparqloscope DBLP	DBLP (01.09.2025)	~ 525 million triples
	Watdiv v0.6	Watdiv data-generator	~ 550 million triples
L	Sparqloscope Wiki-truthy	Wikidata-truthy (13.06.2025)	~ 8 billion triples
	WDBench	Wikidata-truthy (13.06.2025)	~ 8 billion triples