vLLM workflow

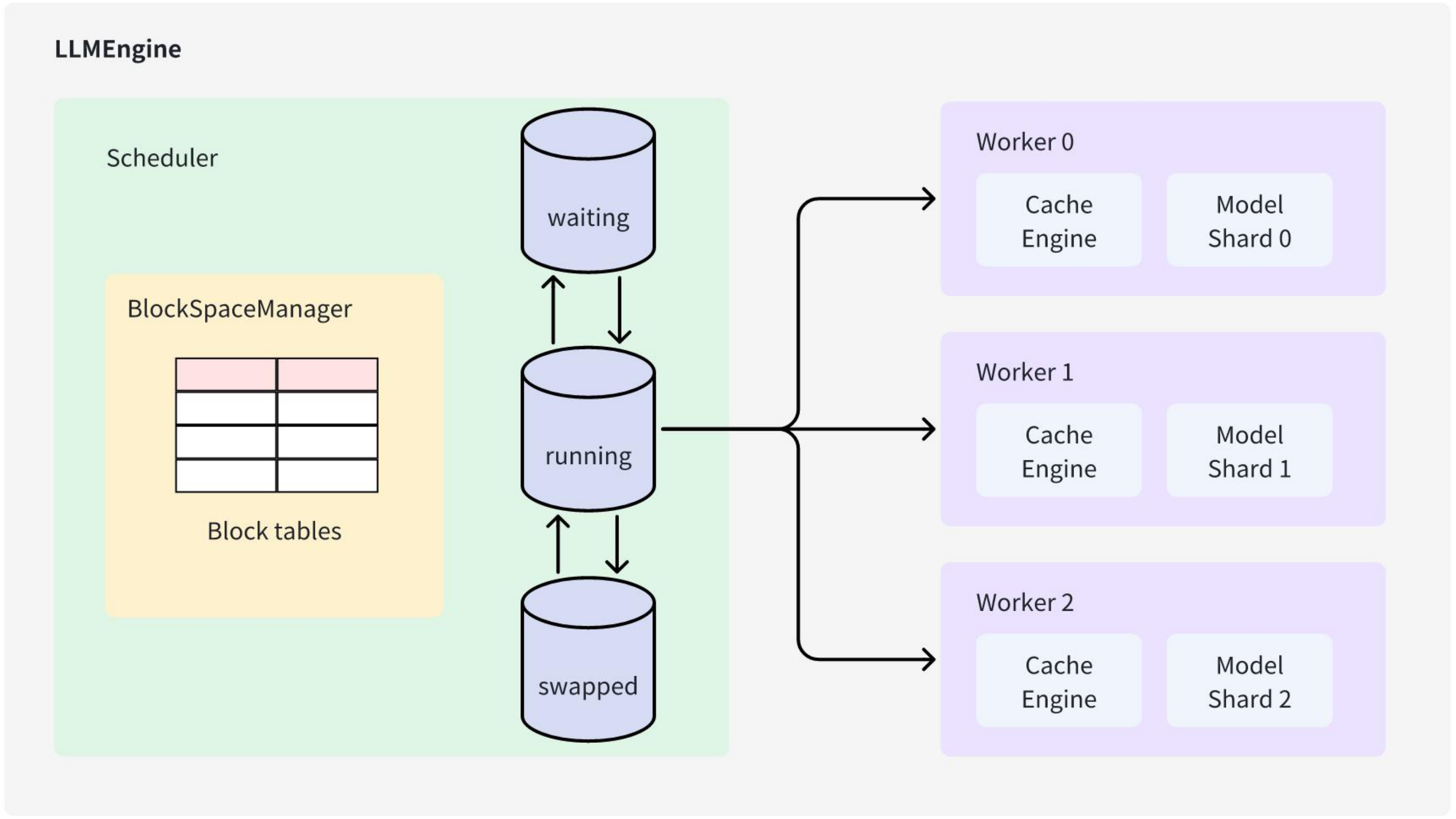
Chenye Wang

Oct 29, 2024

[1] https://github.com/vllm-project/vllm/tree/main/vllm

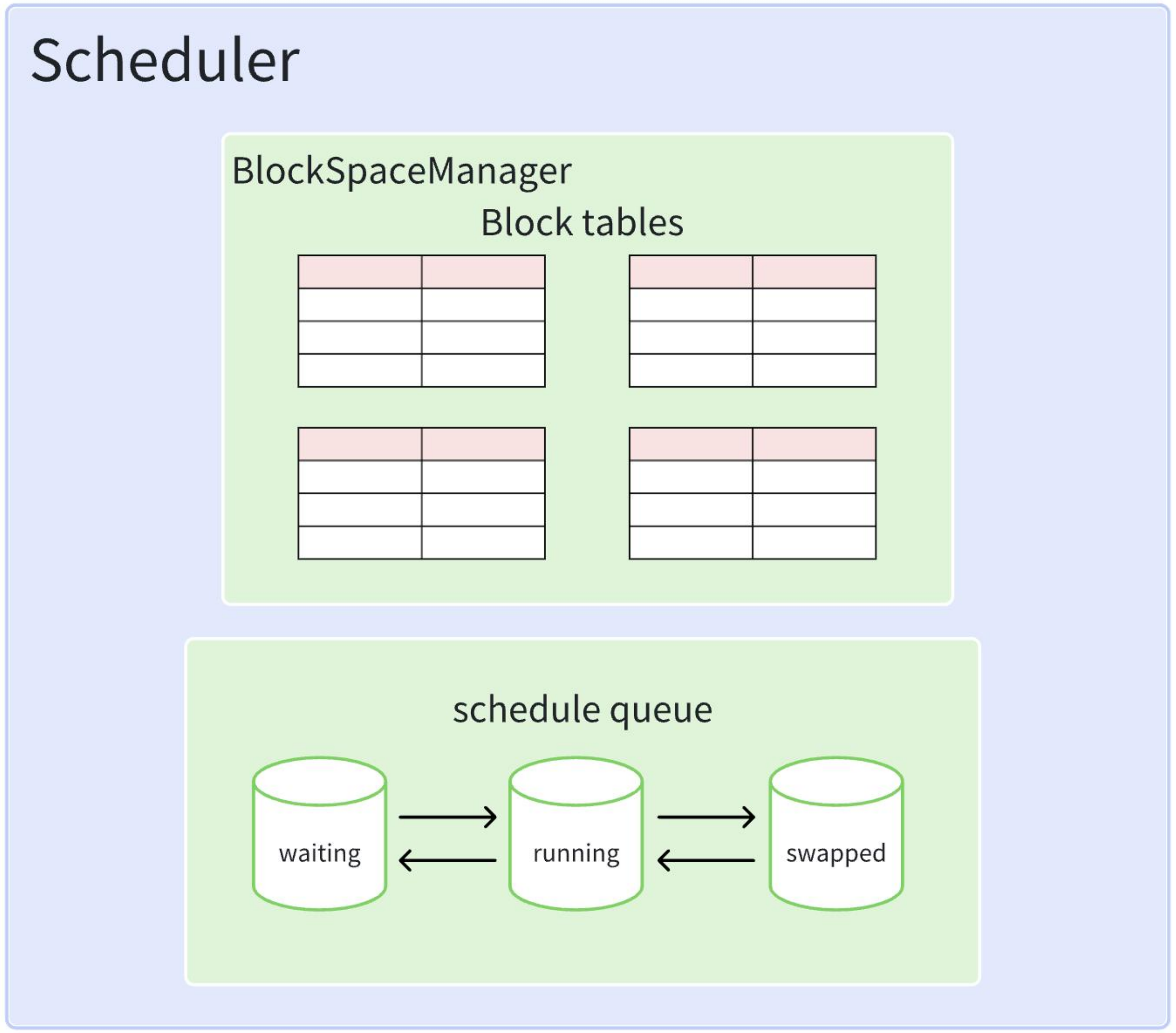
Frame

- LLMEngine
  - offline inference
  - online serving
- Scheduler
  - request schedule
- Worker
  - model inference



Scheduler

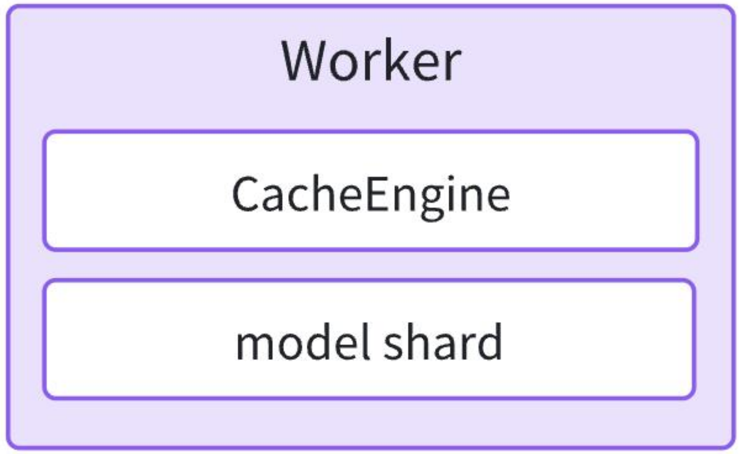
- Iteration-level strategy
  - reschedule requests after generating a token
    - unfixed batch size ✓
  - different from Orca
    - prefill phase
    - decode phase
- BlockSpaceManager
  - maintain block table
- Queue
  - FCFS

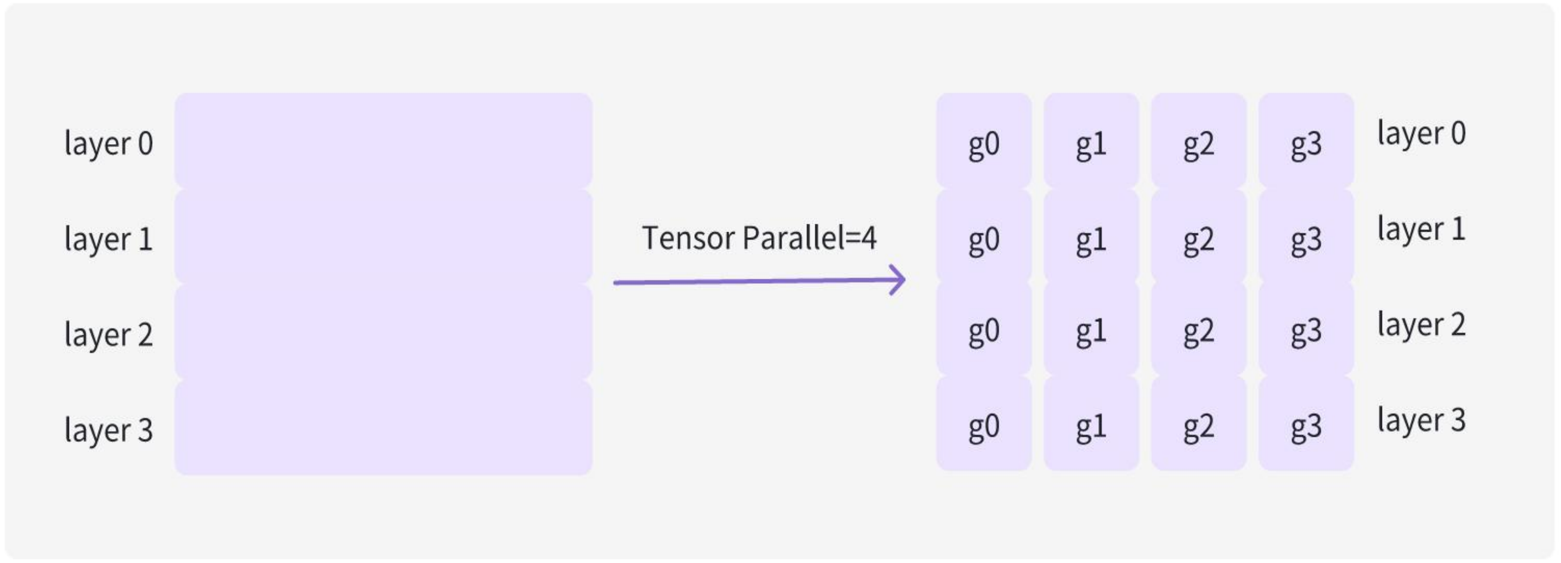


[1] Orca: A Distributed Serving System for Transformer-Based Generative Models. OSDI, 2022.

Worker

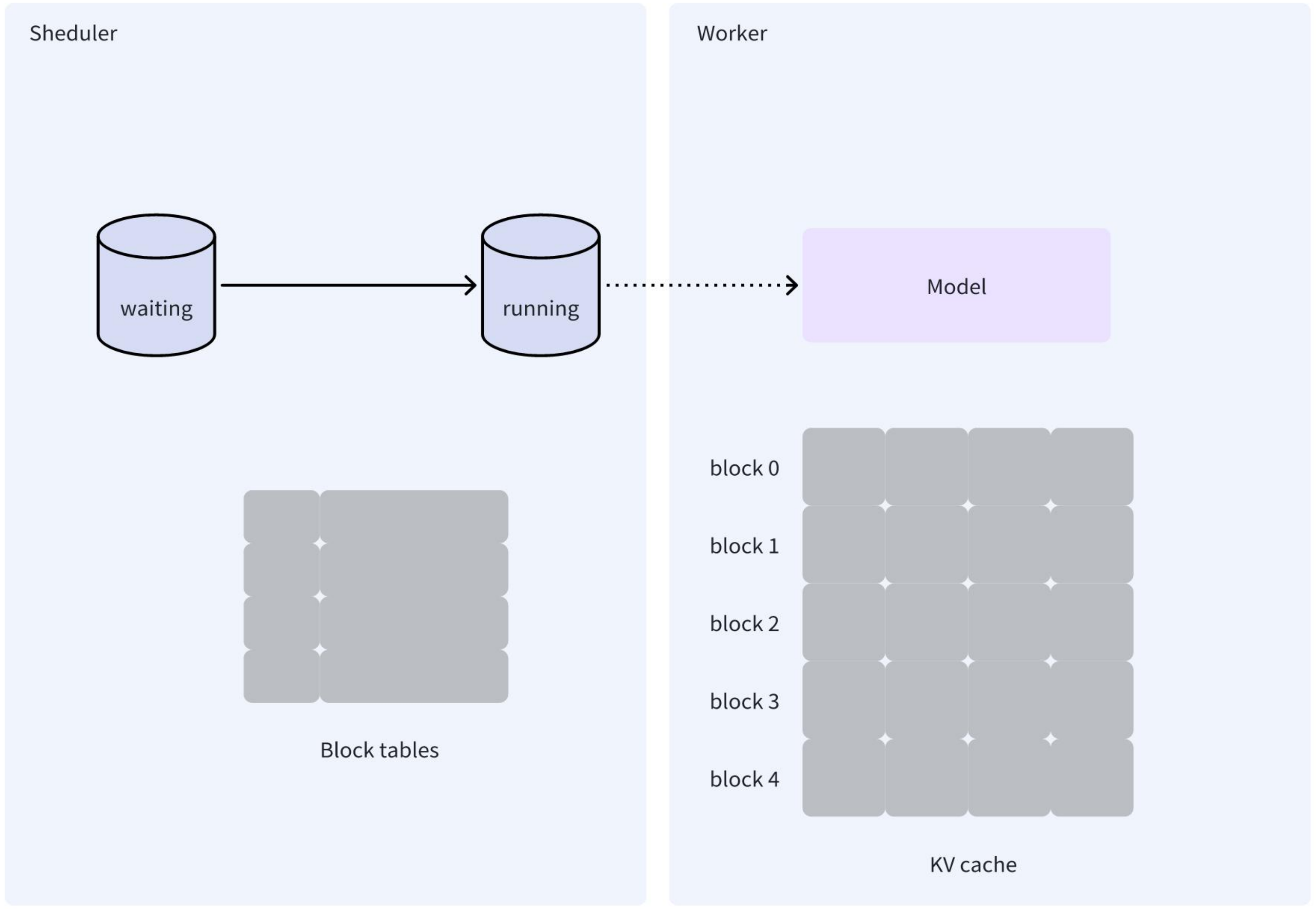
- Correspond to a GPU
- CacheEngine
  - manage KV cache
- Tensor Parallel
  - distributed





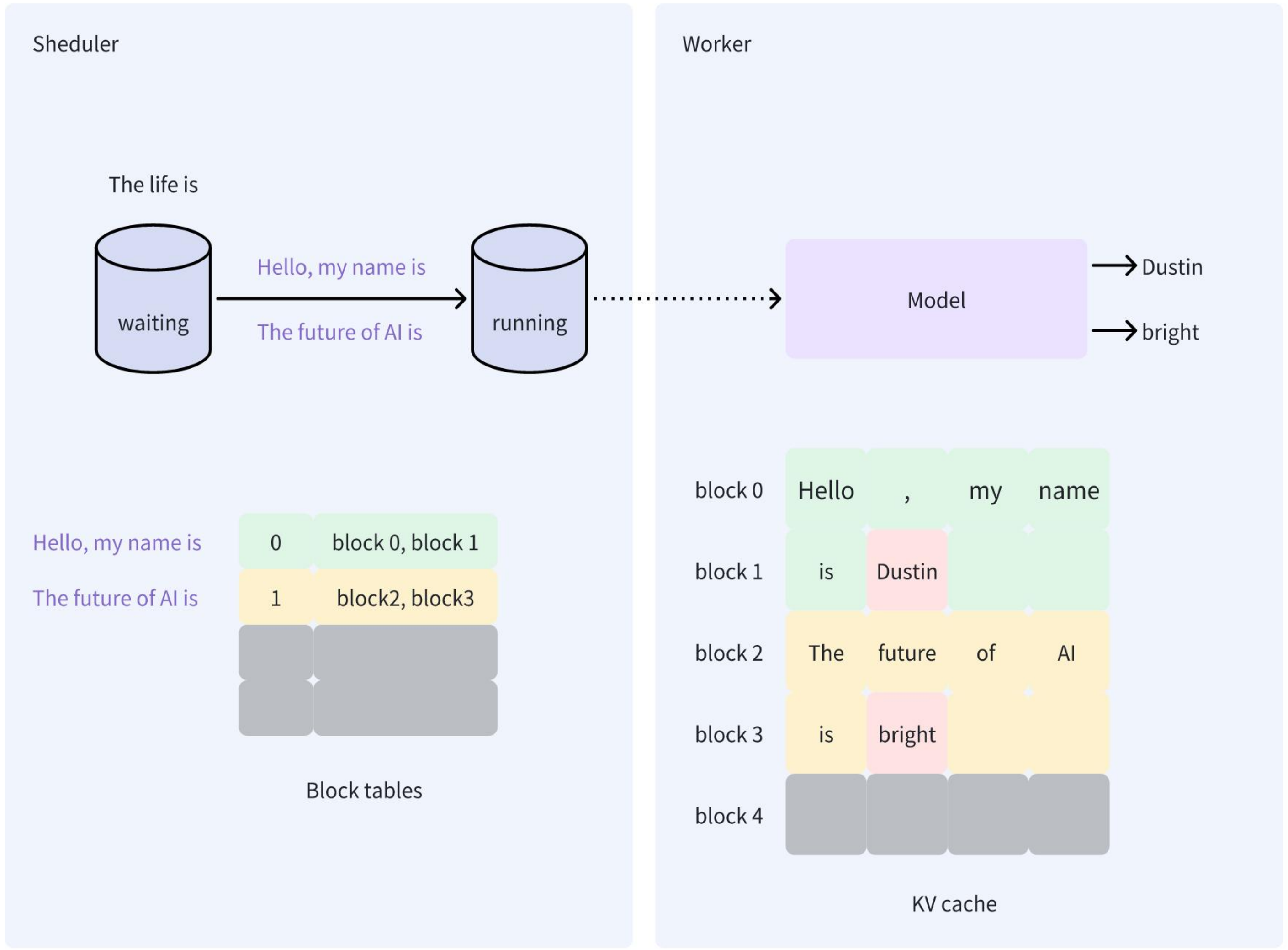
Workflow -- initialization

- Block tables
- Model
- KV cache



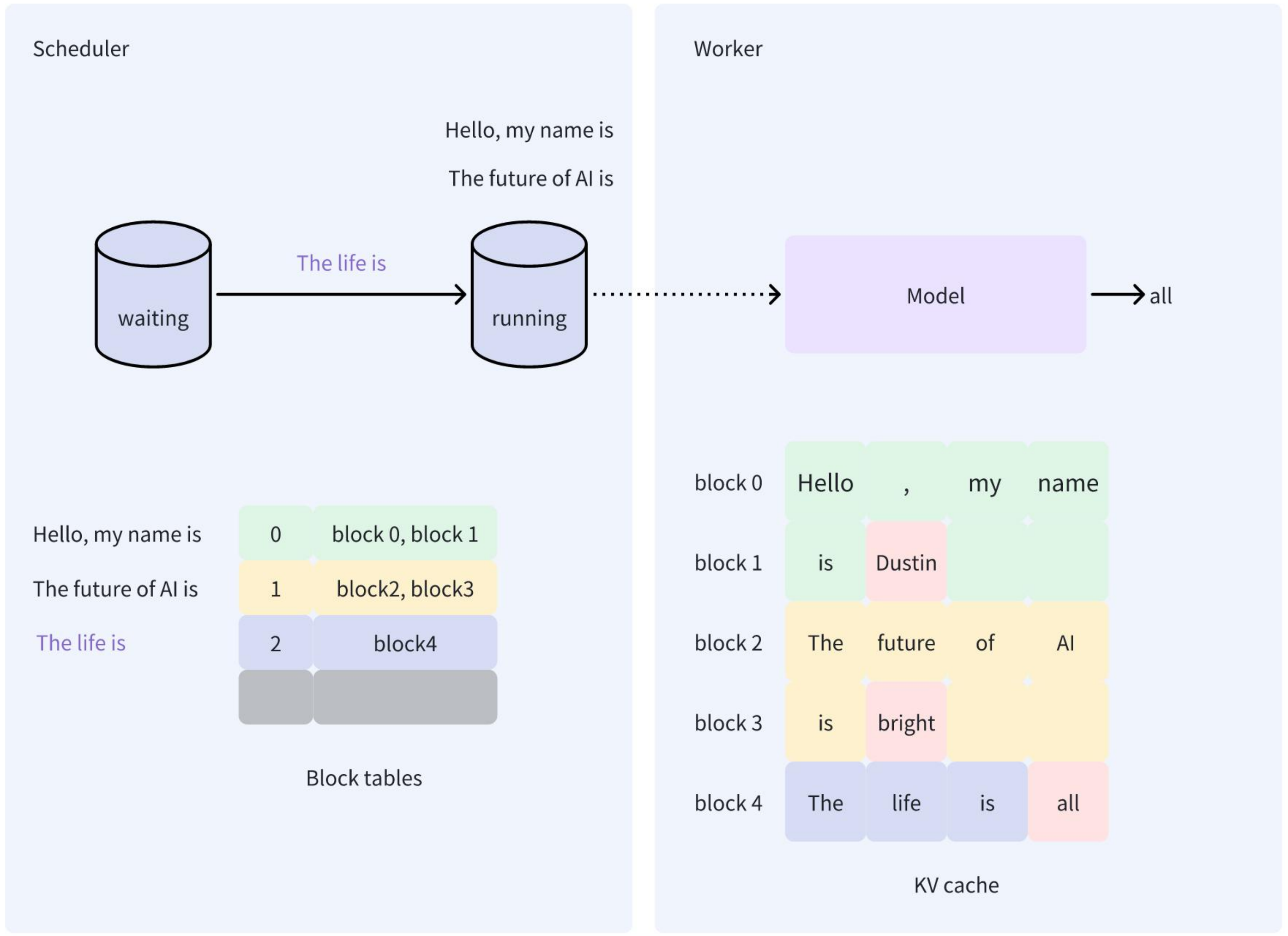
Workflow -- schedule and inference

- FCFS



Workflow -- schedule and inference

- either Prefill
- or Decode



How does vLLM generate output?

- Let’s start from a common example, top down

```python
prompts = [
    "Hello, my name is",
    "The president of the United States is",
    "The capital of France is",
    "The future of AI is",
]

sampling_params = SamplingParams(temperature=0.8, top_p=0.95)

llm = LLM(model="facebook/opt-125m")

outputs = llm.generate(prompts, sampling_params)

for output in outputs:
    prompt = output.prompt
    generated_text = output.outputs[0].text
    print(f"Prompt: {prompt!r}, Generated text: {generated_text!r}")
```

Enter LLM

- Initialize LLMEngine

```
# vllm/entrypoints/llm.py
class LLM:
    def __init__(
        self,
        model: str,
        ...
    ) -> None:
        engine_args = EngineArgs(
            model=model,
            ...
        )
        self.llm_engine = LLMEngine.from_engine_args(
            engine_args, ...)
        ...
```

Generate

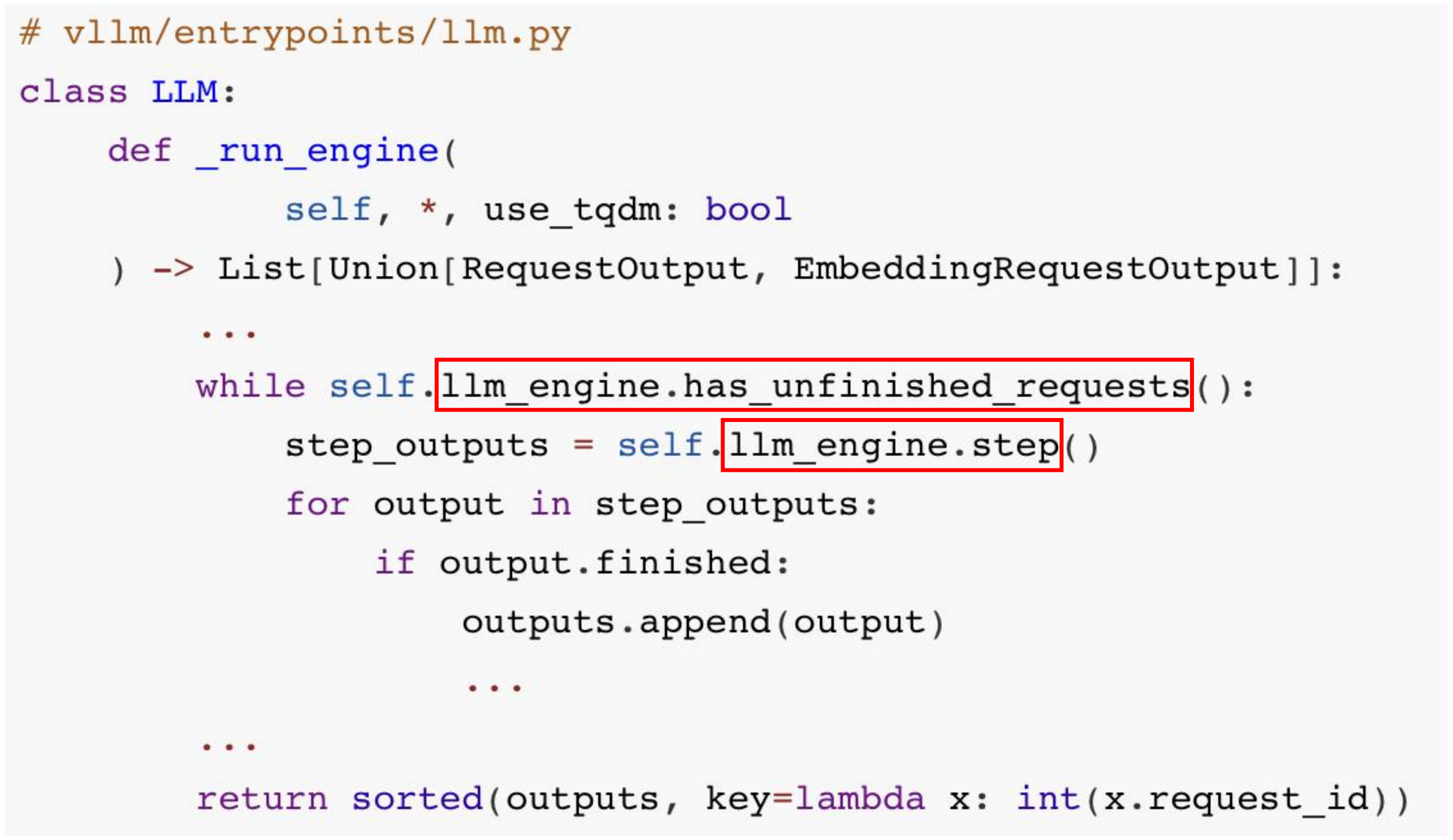
- Add all requests, then run engine

```
# vllm/entrypoints/llm.py
class LLM:
    def generate(
        self,
        prompts: ...,
        sampling_params: ...,
        prompt_token_ids: ...,
        use_tqdm: bool = True,
        ...
    ) -> List[RequestOutput]:
        ...
        self._validate_and_add_requests(
            prompts=...,
            params=sampling_params,
            ...)

        outputs = self._run_engine(use_tqdm=use_tqdm)
        return LLMEngine.validate_outputs(outputs, RequestOutput)
```

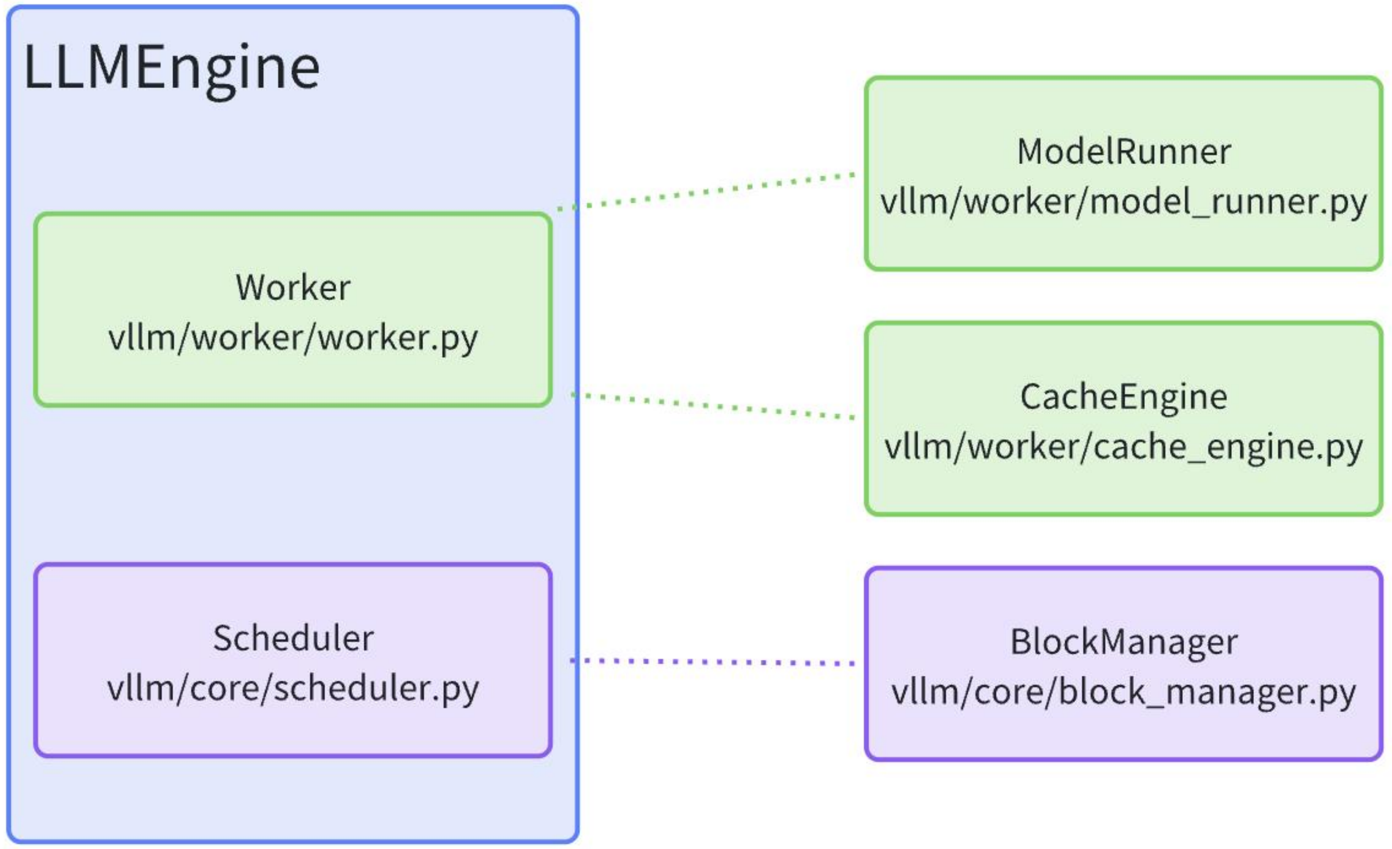
Run engine

- If has unfinished request, step!



```python
# vllm/entrypoints/llm.py
class LLM:
    def _run_engine(
            self, *, use_tqdm: bool
    ) -> List[Union[RequestOutput, EmbeddingRequestOutput]]:
        ...
        while self.llm_engine.has_unfinished_requests():
            step_outputs = self.llm_engine.step()
            for output in step_outputs:
                if output.finished:
                    outputs.append(output)
                    ...
        ...
        return sorted(outputs, key=lambda x: int(x.request_id))
```

Enter LLMEngine and ...


LLMEngine
Worker
vllm/worker/worker.py
Scheduler
vllm/core/scheduler.py
ModelRunner
vllm/worker/model_runner.py
CacheEngine
vllm/worker/cache_engine.py
BlockManager
vllm/core/block_manager.py