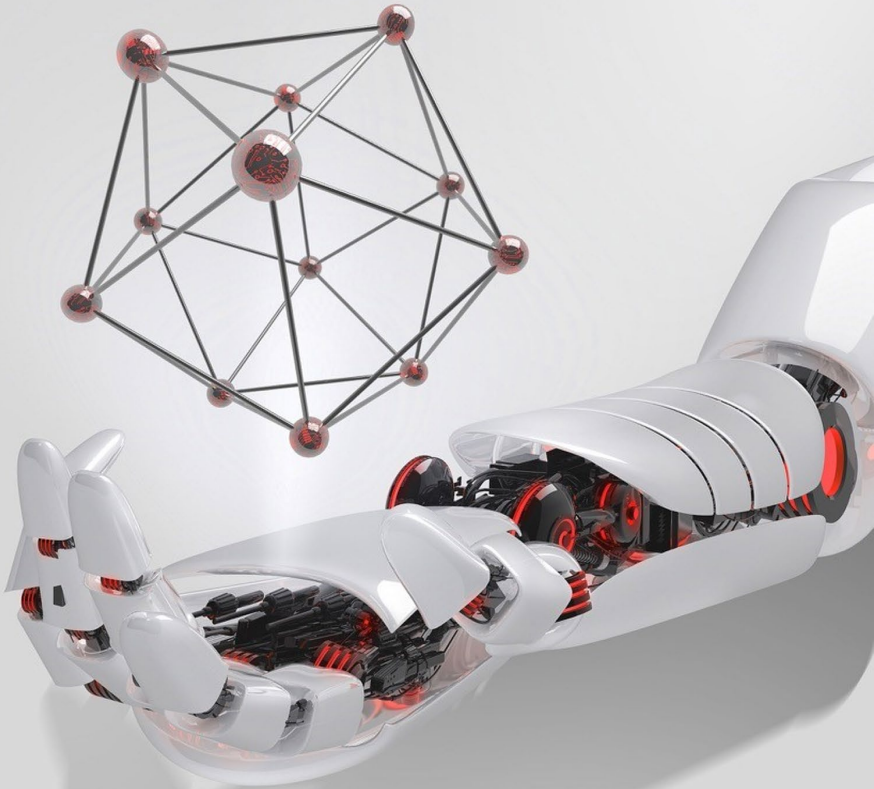
Llumnix:Dynamic Scheduling for Large Language Model Serving

演讲人：赵汉宇

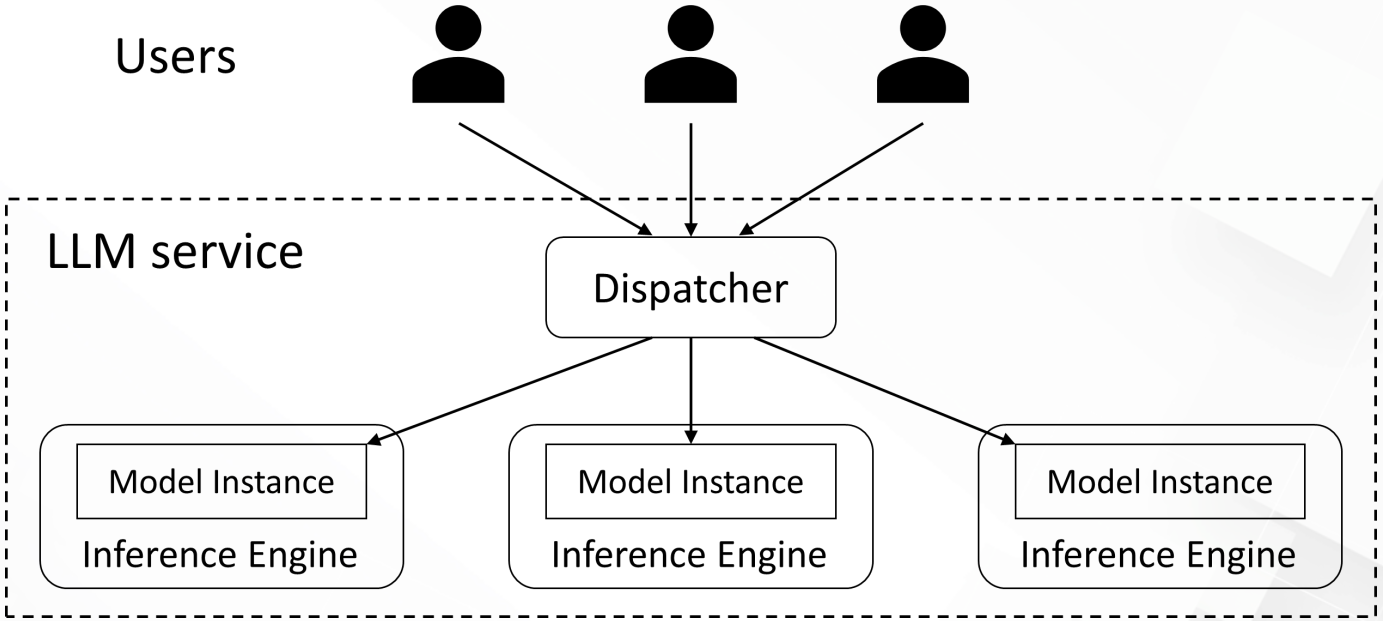
阿里巴巴 阿里云人工智能平台 PAI 技术专家



RAY CONNECT 2024

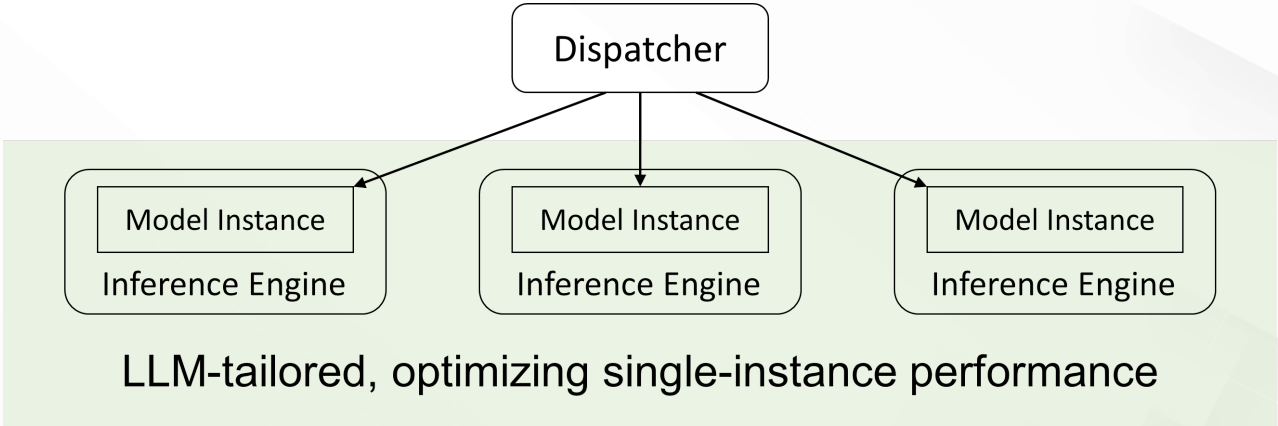
LLM Serving Today: A Cluster Perspective

- A **request dispatcher** + *multiple instances* of an **inference engine**



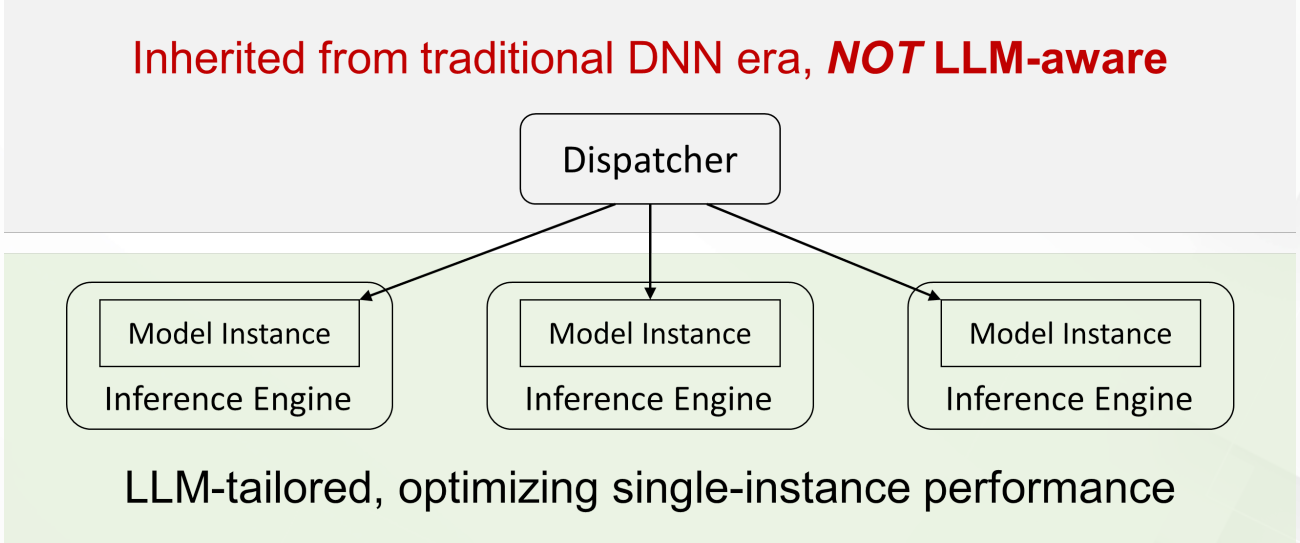
LLM Serving Today: A Cluster Perspective

- A **request dispatcher** + *multiple instances* of an **inference engine**



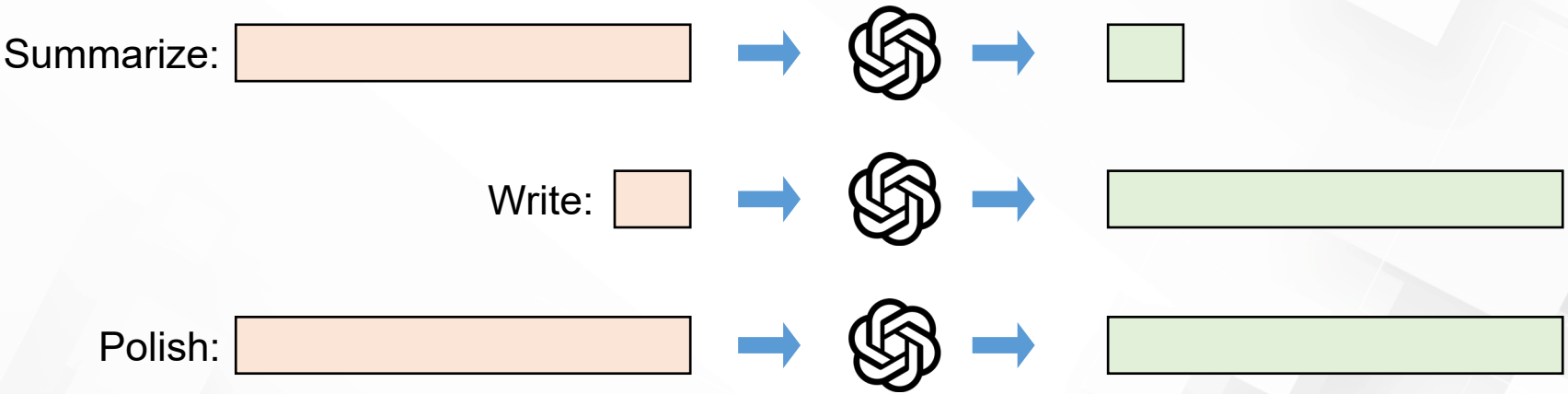
LLM Serving Today: A Cluster Perspective

- A **request dispatcher** + *multiple instances* of an **inference engine**



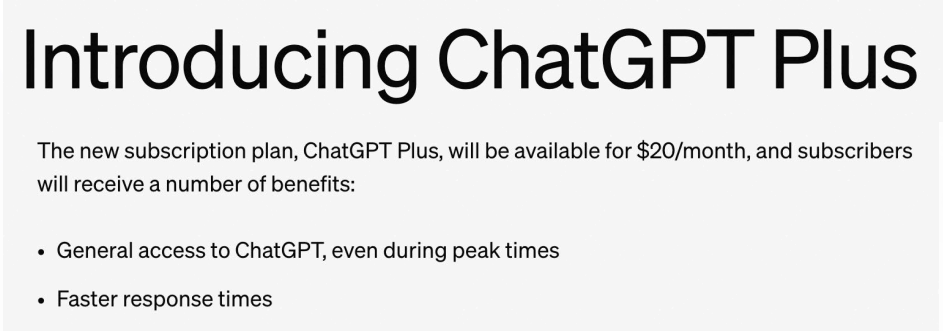
LLM Characteristic (1): *Workload Heterogeneity*

- *Universal* models, *diverse* applications
- Requests are **heterogeneous**
  - *Sequence (input/output) lengths*



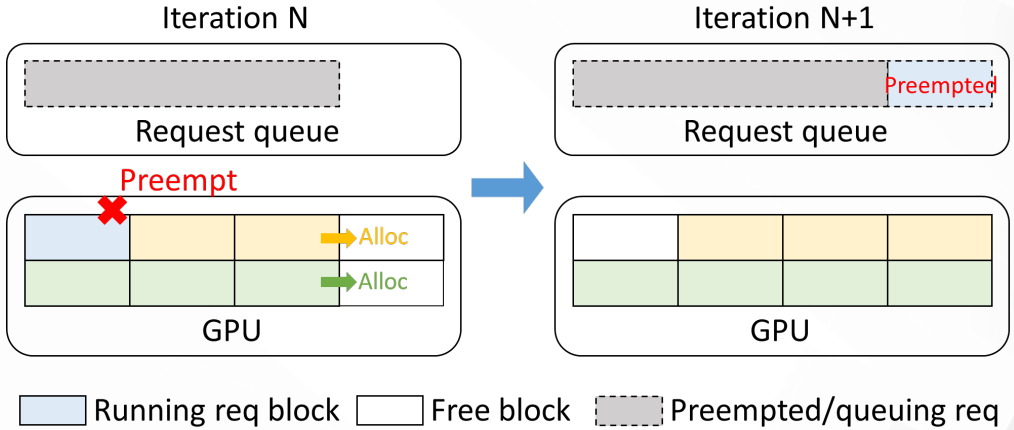
LLM Characteristic (1): *Workload Heterogeneity*

- *Universal models, diverse applications*
- *Requests are **heterogeneous***
  - *Sequence (input/output) lengths*
  - *Latency SLOs: interactive vs. offline, ChatGPT plus vs. normal*



LLM Characteristic (2): *Execution Unpredictability*

- **Autoregressive** execution
  - Output lengths *not known a priori*
  - Dynamic GPU *memory demands of KV caches*
- PagedAttention: paged memory allocation + preemptive scheduling [1]



[1] Kwon et al. Efficient Memory Management for Large Language Model Serving with PagedAttention (SOSP '23)

Challenge (1): Performance Isolation

- Preemptions -> poor tail latencies

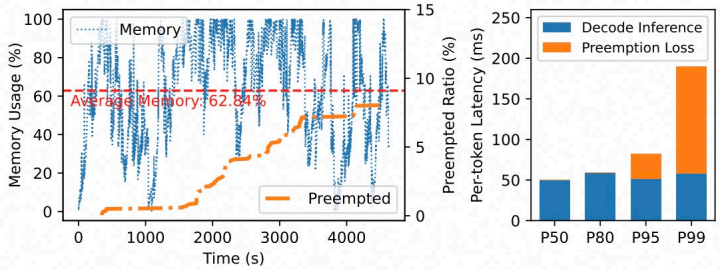


Figure 3: Request preemptions in LLaMA-7B serving.

- Performance interference in a batch

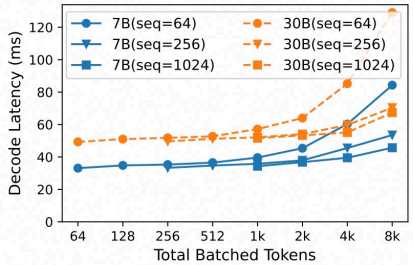


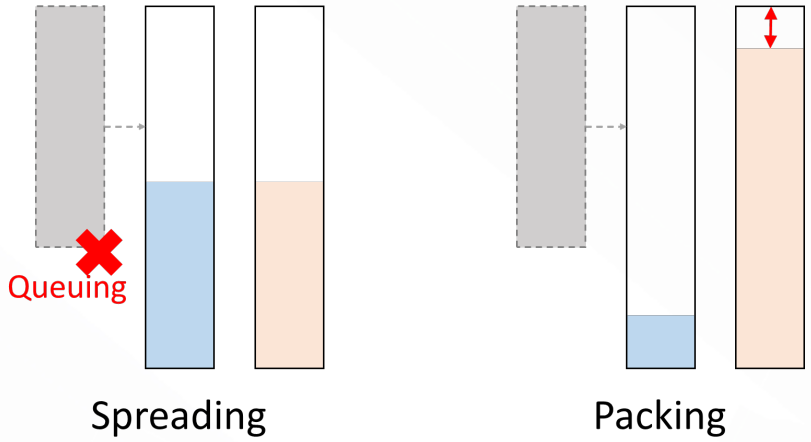
Figure 4: Latencies of one decode step of LLaMA-7B and LLaMA-30B with different sequence lengths and batch sizes.

- Load balancing via *one-shot* dispatching could be suboptimal due to unpredictable execution

Requirement (1): **Continuous** load balancing

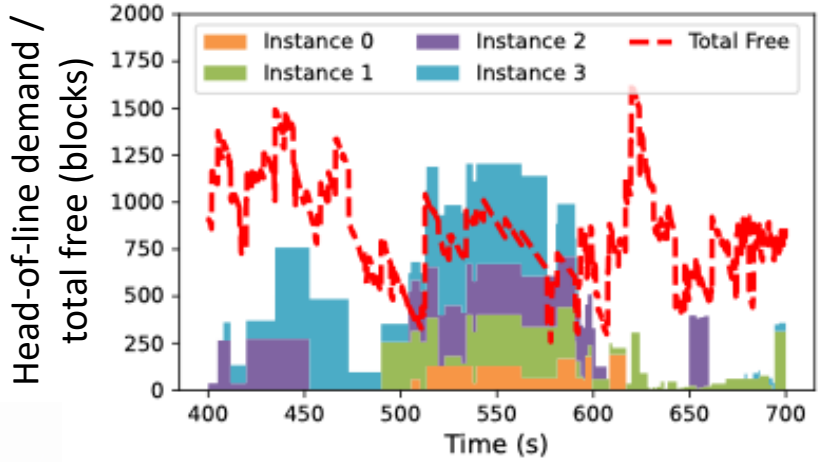
Challenge (2): Memory Fragmentation

- Load balancing -> fragmentation across instances
  - A classic spreading vs. packing tradeoff



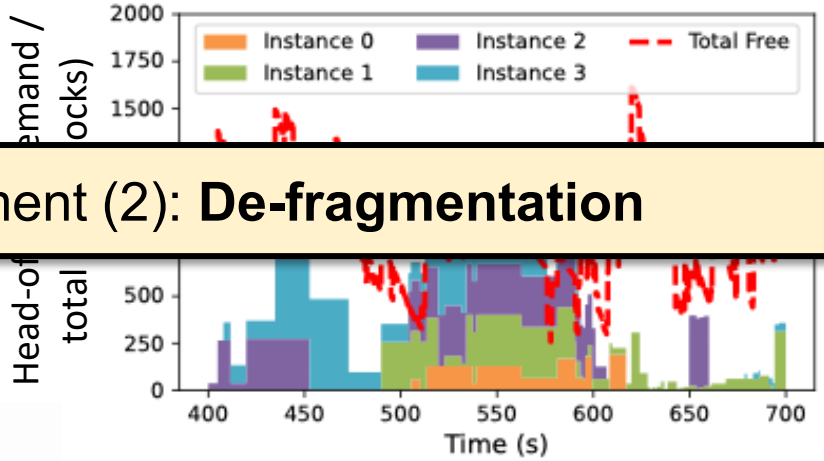
Challenge (2): Memory Fragmentation

- Load balancing -> fragmentation across instances
  - A classic spreading vs. packing tradeoff
- Fragmentation -> **worse queuing delays** (first-token latencies)
  - A large space on one instance needed for the prompt



Challenge (2): Memory Fragmentation

- Load balancing -> fragmentation across instances
  - A classic spreading vs. packing tradeoff
- Fragmentation -> **worse queuing delays** (first-token latencies)
  - A large space on one instance needed for the prompt



Requirement (2): **De-fragmentation**

Challenge (3): Differentiated SLOs

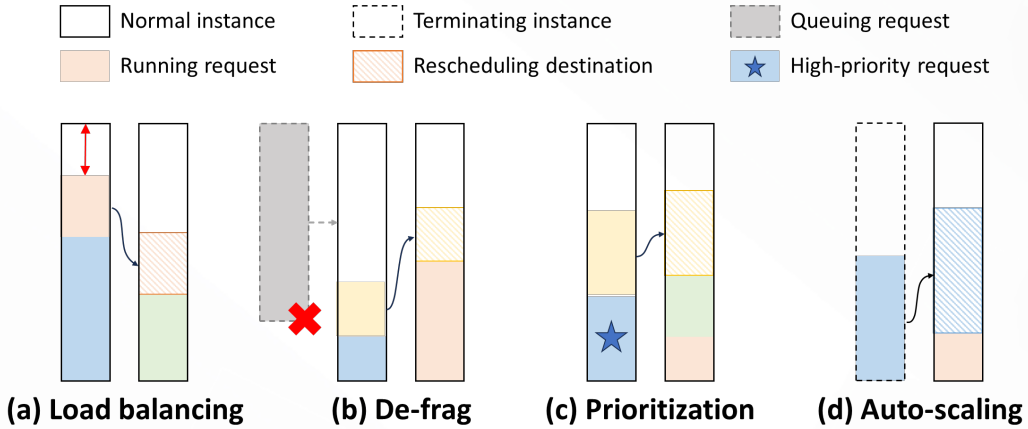
- Existing systems treat all requests **equally**

- Urgent requests could be easily interfered by normal ones
  - Queuing delays
  - Performance interference

Requirement (3): Request **priorities**

Llumnix: Request Scheduling Made Dynamic

- **Continuous rescheduling** across instances
  - Combined with dispatching and auto-scaling
- Powerful in various scheduling scenarios



Design Goals

Our aim: make rescheduling the *norm* in LLM serving

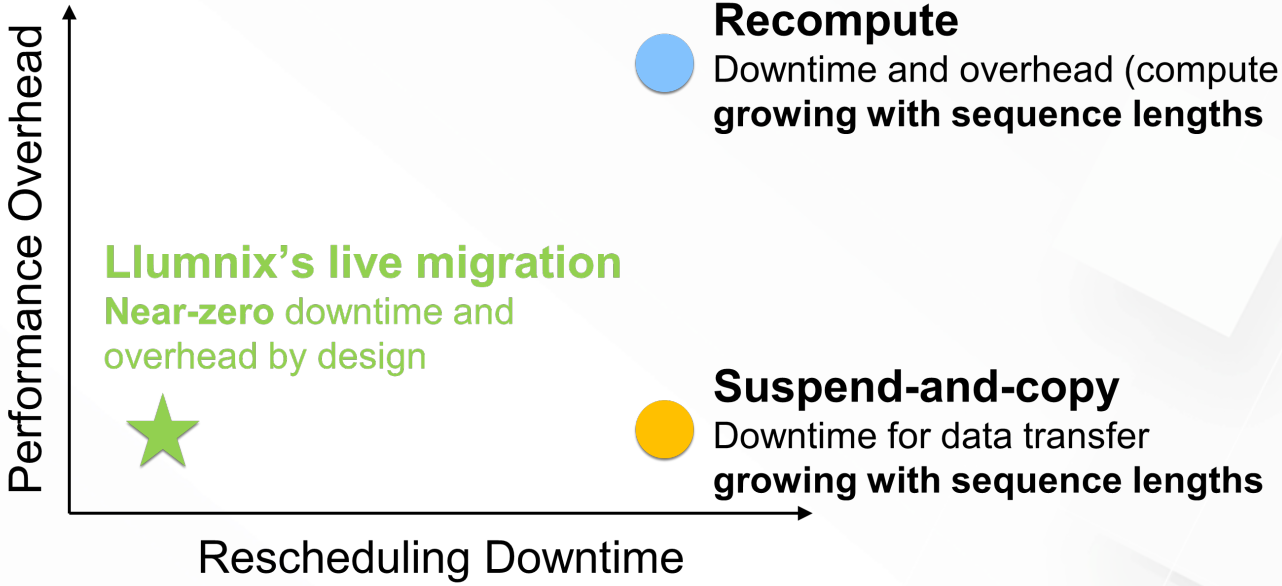
- Efficiency → *Live migration mechanism*
- Scalability → *Distributed scheduling architecture*
- Scheduling Benefits → *Unified, multi-objective scheduling policy*

Design Goals

Our aim: make rescheduling the *norm* in LLM serving

- Efficiency → *Live migration mechanism*
- Scalability → *Distributed scheduling architecture*
- Scheduling Benefits → *Unified, multi-objective scheduling policy*

How to Reschedule KV Caches?

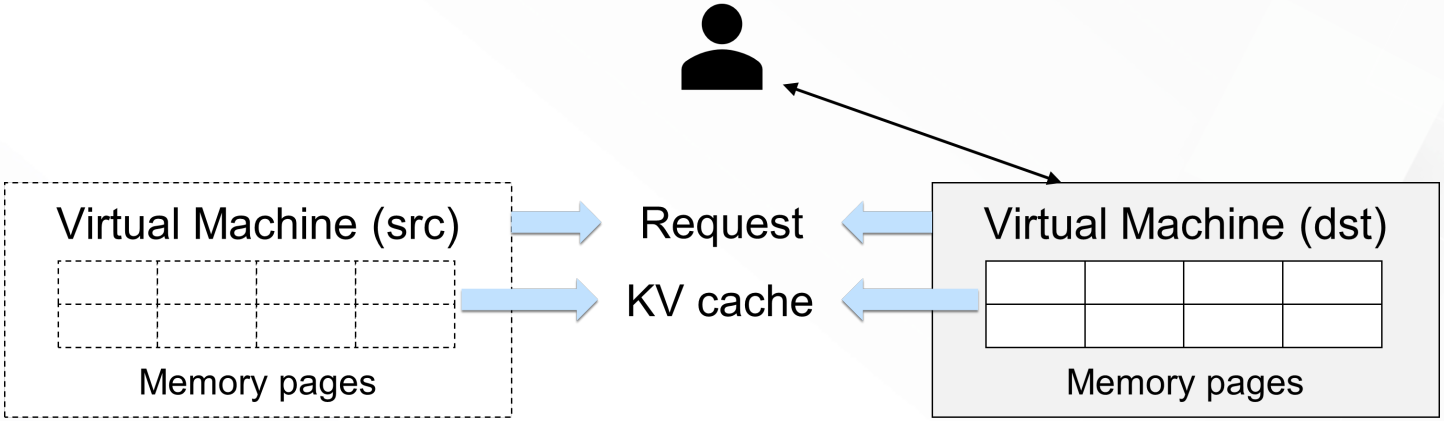


Inspiration: VM Live Migration

Inspiration: VM Live Migration

Inspiration: VM Live Migration

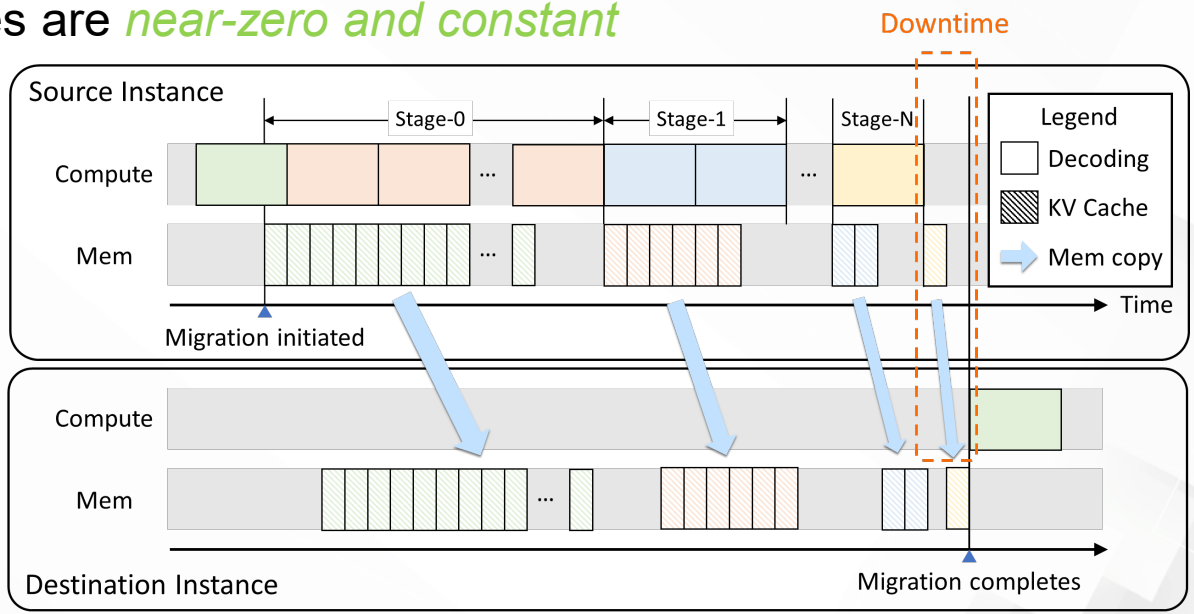
Inspiration: VM Live Migration



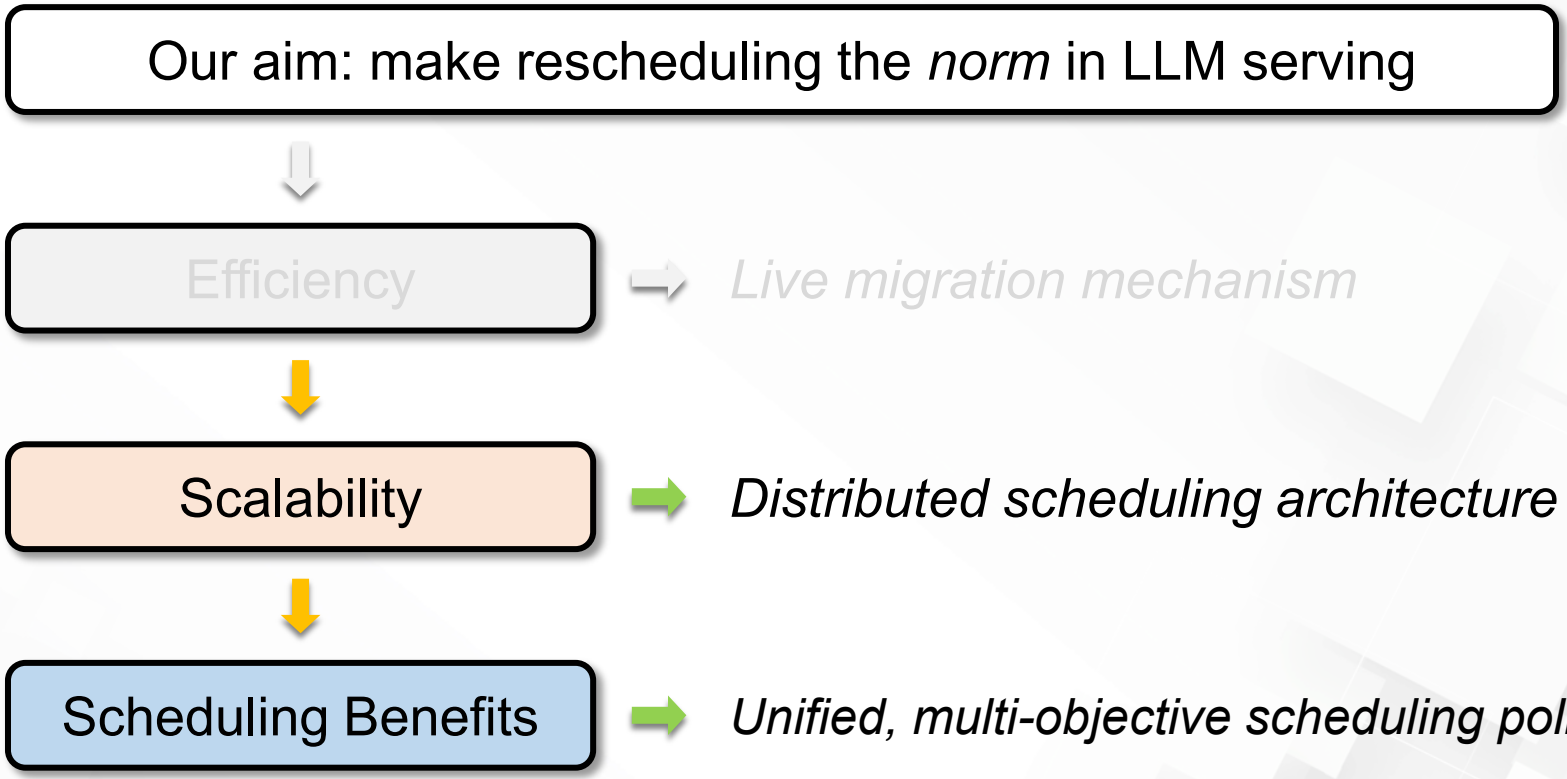
What are dirty pages?

Live Migration of LLM Requests

- KV caches are **append-only**
  - Copy ~~dirty~~ *incremental* blocks iteratively
  - Downtimes are *near-zero and constant*

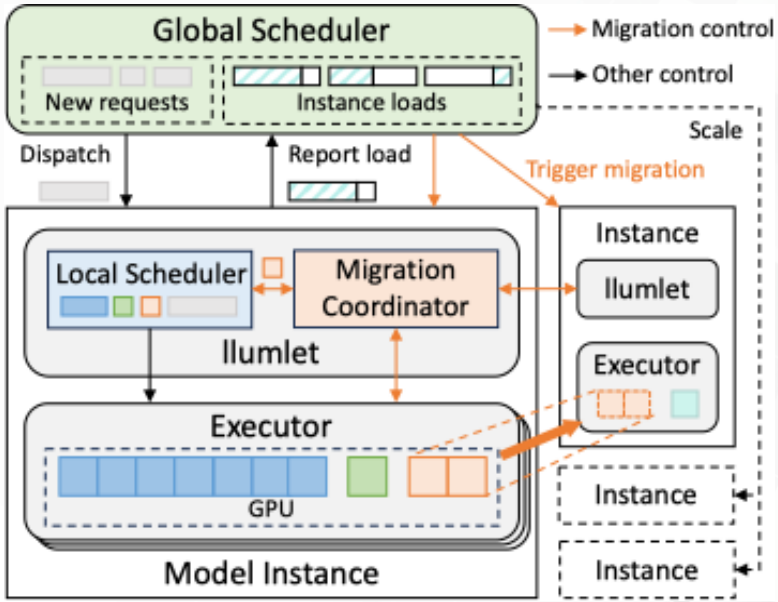


Design Goals



Distributed Scheduling Architecture

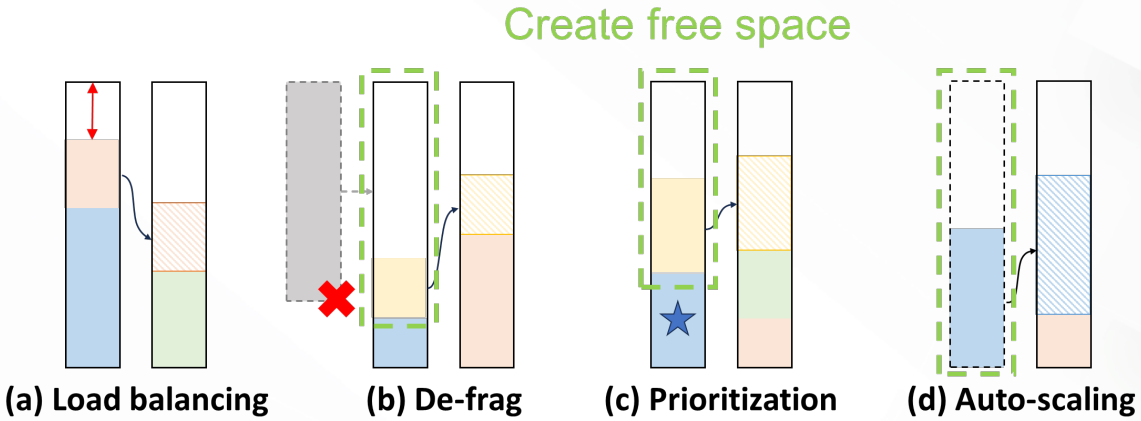
- **Global scheduler** for cross-instance scheduling
- Distributed **llumlet**s for local scheduling
- A narrow interface: **instance load**



Scheduling Policy (sketch)

- **Virtual usage**: unifying multiple objectives

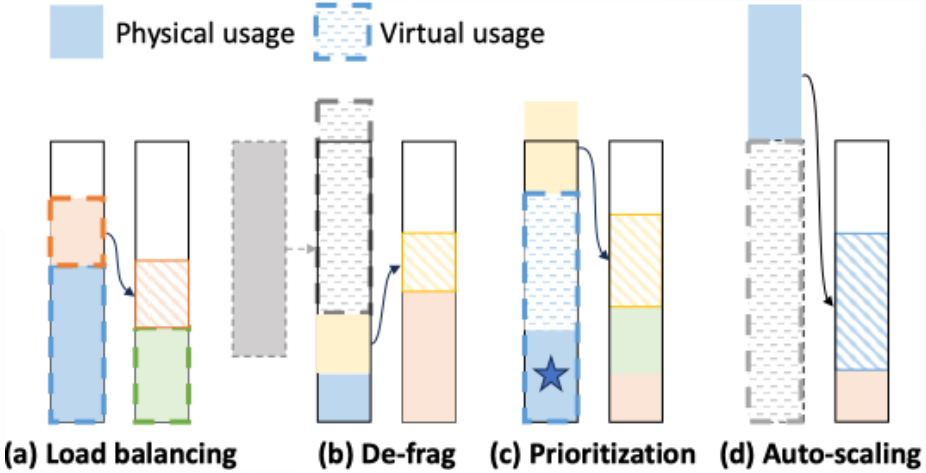
- Policy: load balancing based on virtual usages



Scheduling Policy (sketch)

- **Virtual usage**: unifying multiple objectives

- Policy: load balancing based on virtual usages

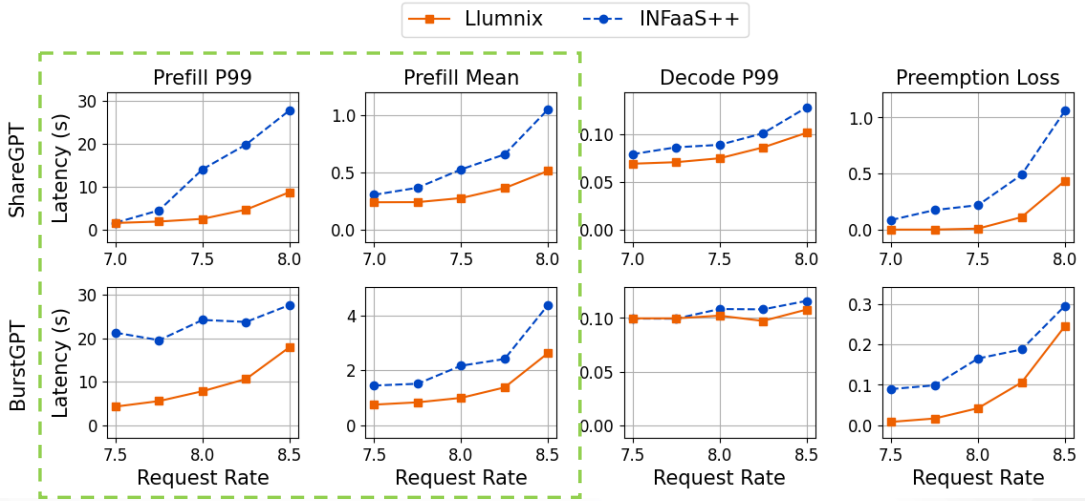


Prototype Implementation and Evaluation

- Implemented as a scheduling layer atop vLLM
- Testbed: 16 A10 GPUs (24GB)
- 4 4-GPU VMs, PCIe 4.0 in each node, 64Gb/s Ethernet across nodes
- Models: LLaMA-7B and LLaMA-30B
- Traces: ShareGPT, BurstGPT, generated power-law distributions

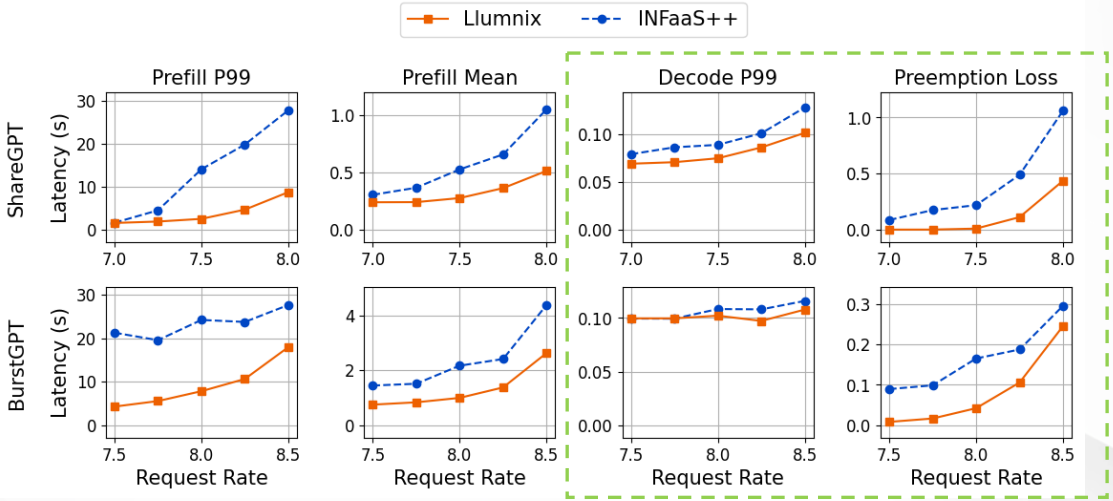
End-to-end Serving Performance (16 LLaMA-7B instances)

- Benefits of migration: compared to dispatch-time load balancing (INFaaS)
  - Up to 2.2x/5.5x for first-token (mean/P99) via de-fragmentation
  - Up to 1.3x for per-token generation P99 via reducing preemptions
- More gains with more diverse sequence lengths (details in our OSDI '24 paper)



End-to-end Serving Performance (16 LLaMA-7B instances)

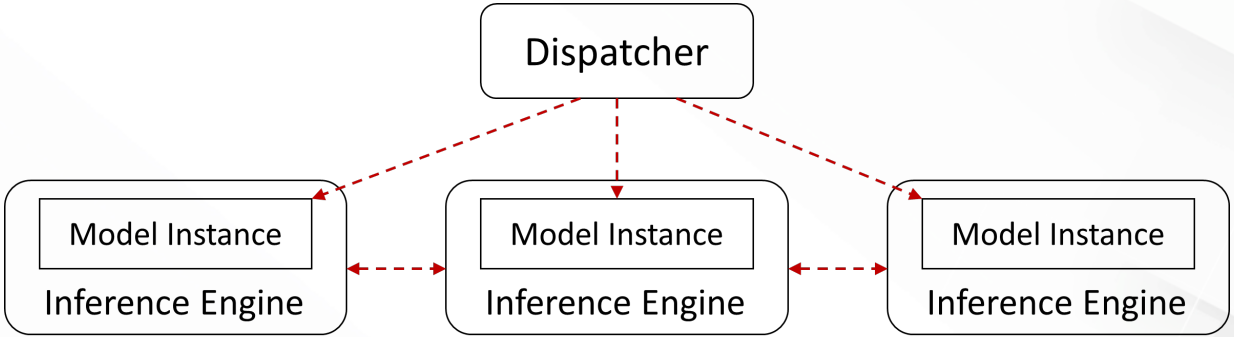
- Benefits of migration: compared to dispatch-time load balancing (INFaaS)
  - Up to 2.2x/5.5x for first-token (mean/P99) via de-fragmentation
  - Up to 1.3x for per-token generation P99 via reducing preemptions
- More gains with more diverse sequence lengths (details in our OSDI '24 paper)



Road to a Product

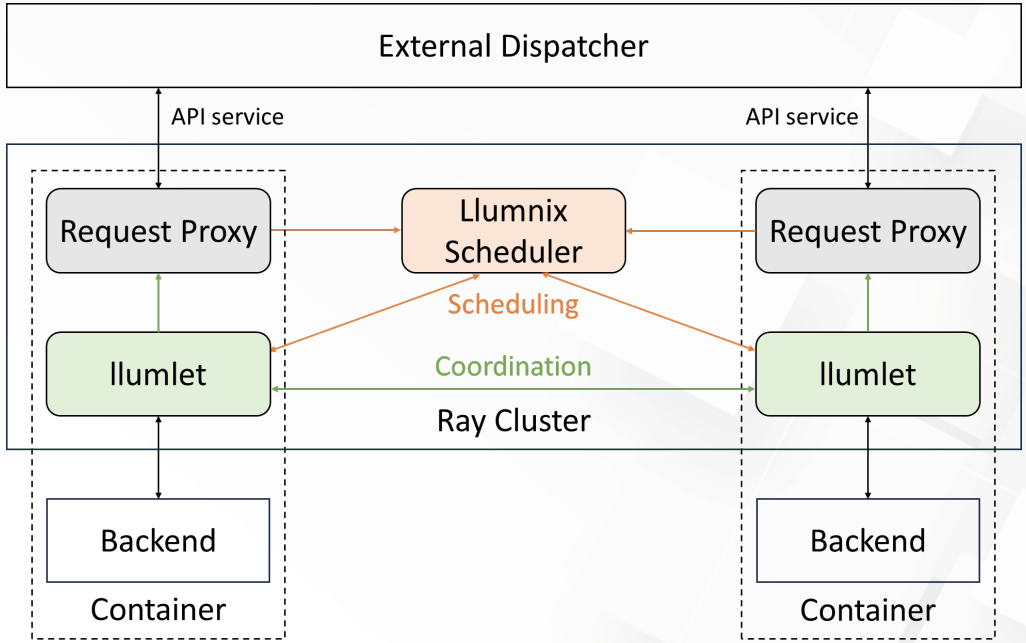
Design Requirements

- Enable fine-grained interaction *between scheduler and instances*, and also *between instances*
  - Hard to be realized efficiently in existing layers



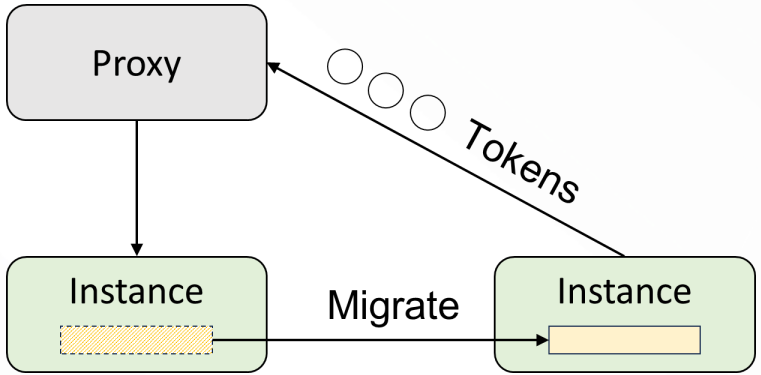
Llumnix as an Indirection Layer

- Compatible with external dispatchers
  - Exposes the same interfaces like inference engines (API endpoints)
- Co-located with inference engines in a Ray cluster
  - Manages components as actors
  - Enables fine-grained communication between Llumnix and inference engines



Request Proxy

- Distributed token forwarding
  - Ensures a steady API endpoint throughout a request's lifecycle (*even migrated*)



Request Proxy

- Distributed token forwarding
  - Ensures a steady API endpoint throughout a request's lifecycle (*even migrated*)

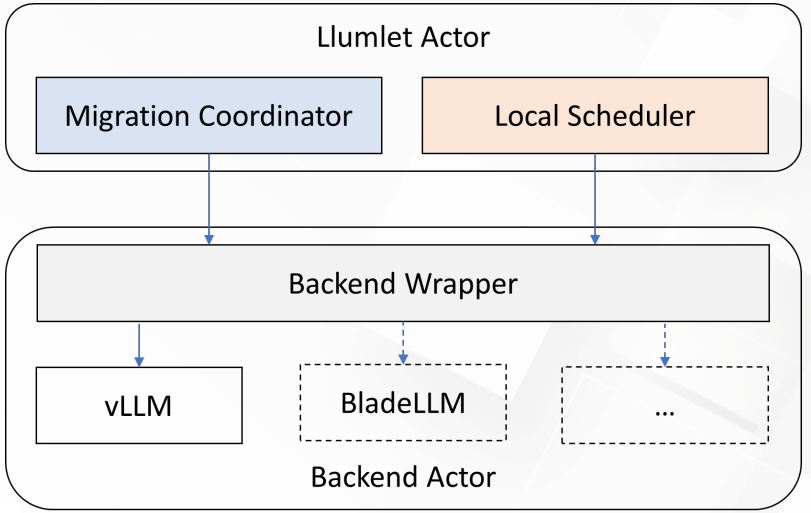
- Similar user interfaces to inference engines
  - Easy to migrate from existing vLLM deployments to Llumnix

```
# vllm
python -m vllm.entrypoints.openai.api_server --model facebook/opt-125m

# llumnix
python -m llumnix.vllm.entrypoints.openai.api_server --model facebook/opt-125m
```

Llumlet and Backends

- Compatible with different inference backends
  - Llumnix's migration is compatible with paged memory allocation by design
  - Backend functionalities abstracted as a common interface
- Supports vLLM and BladeLLM (Alibaba's internal inference engine)

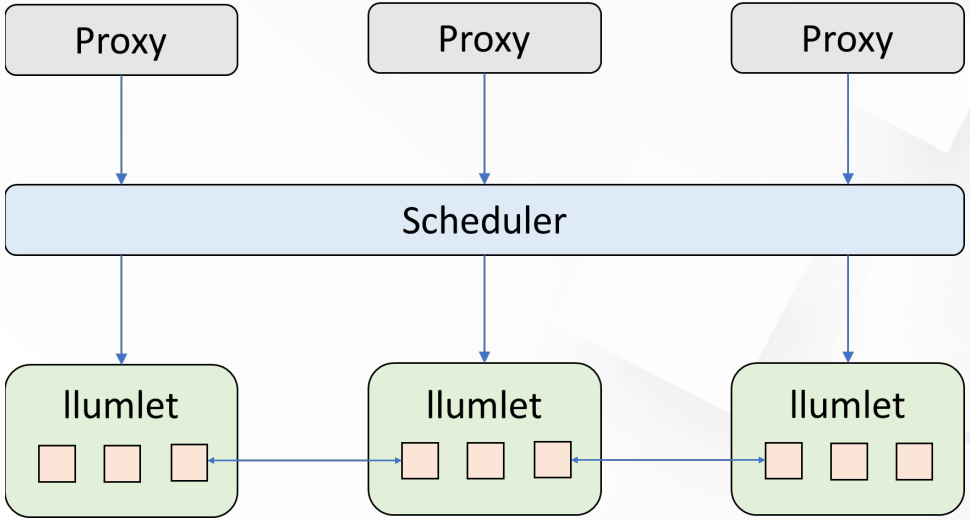


Fault Tolerance

- Llumnix needs careful fault tolerance design
  - In Llumnix, instances are *not* naturally fault-isolated
- Goal: high service availability during failure of any components

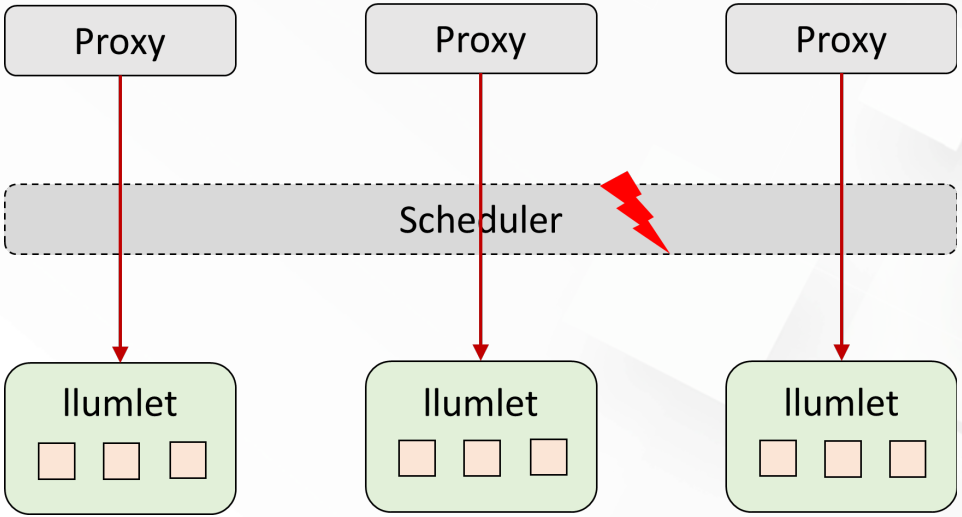
Fault Tolerance

- Scheduler failures
  - Bypass scheduler and disable migration during failover



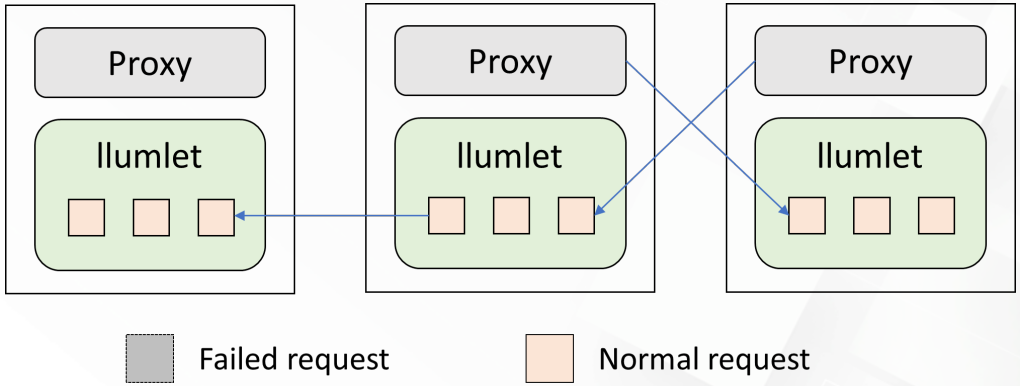
Fault Tolerance

- Scheduler failures
  - Bypass scheduler and disable migration during failover



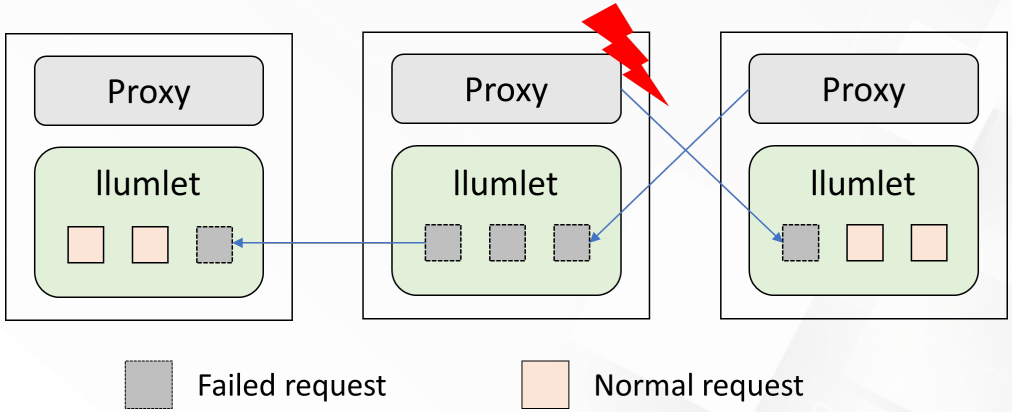
Fault Tolerance

- Scheduler failures
  - Bypass scheduler and disable migration during failover
- Proxy failures
  - Abort all associated requests
- Llumlet failures
  - Abort all running requests
  - Abort all ongoing migrations



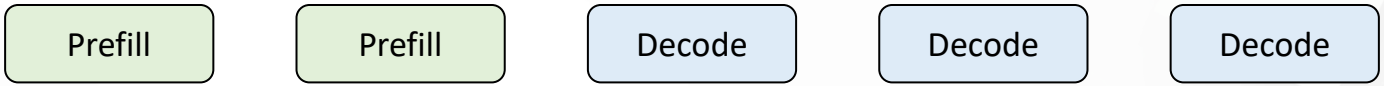
Fault Tolerance

- Scheduler failures
  - Bypass scheduler and disable migration during failover
- Proxy failures
  - Abort all associated requests
- Llumlet failures
  - Abort all running requests
  - Abort all ongoing migrations



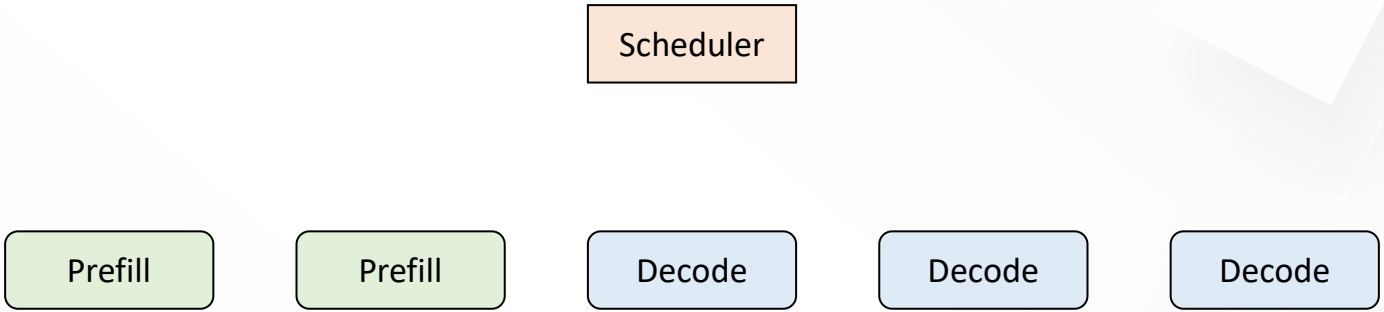
Prefill-Decode Disaggregation

- Isolating P/D to eliminate interference / utilize heterogeneous hardware



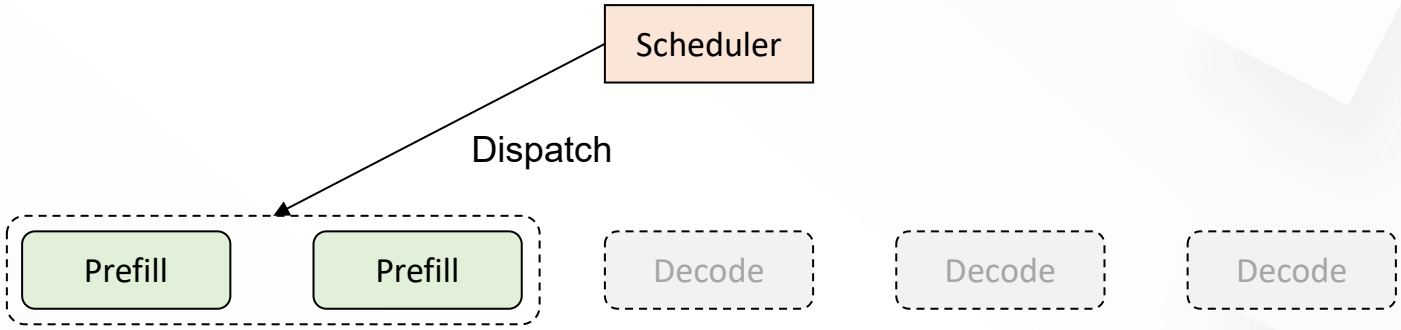
Scheduling-Defined Prefill-Decode Disaggregation

- P-D disaggregation is essentially a *request scheduling policy*
  - A special dispatching rule (P-instances-only)
  - A special migration rule (migrate after one step)



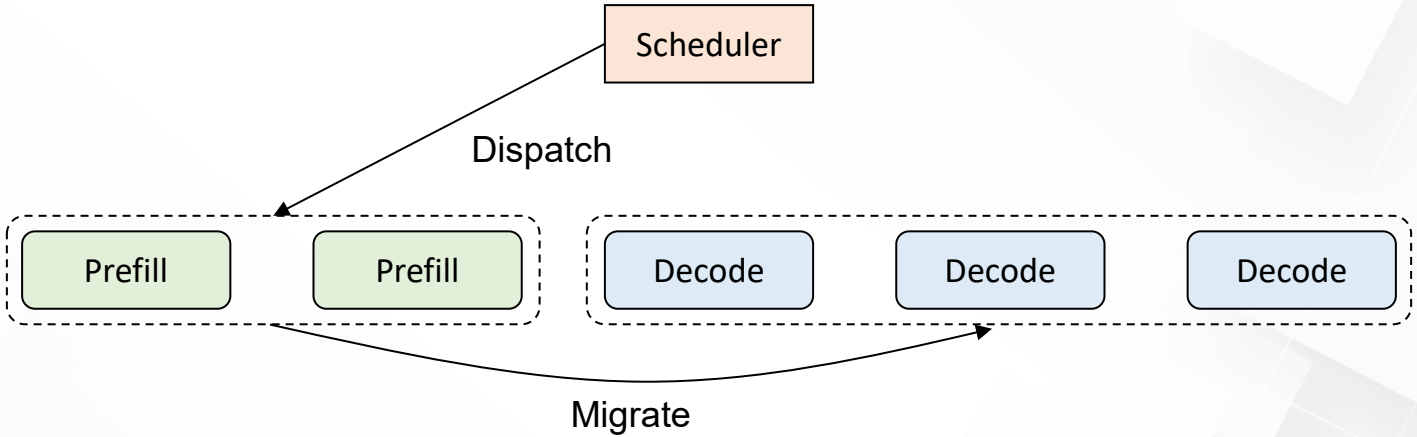
Scheduling-Defined Prefill-Decode Disaggregation

- P-D disaggregation is essentially a *request scheduling policy*
  - A special dispatching rule (P-instances-only)
  - A special migration rule (migrate after one step)



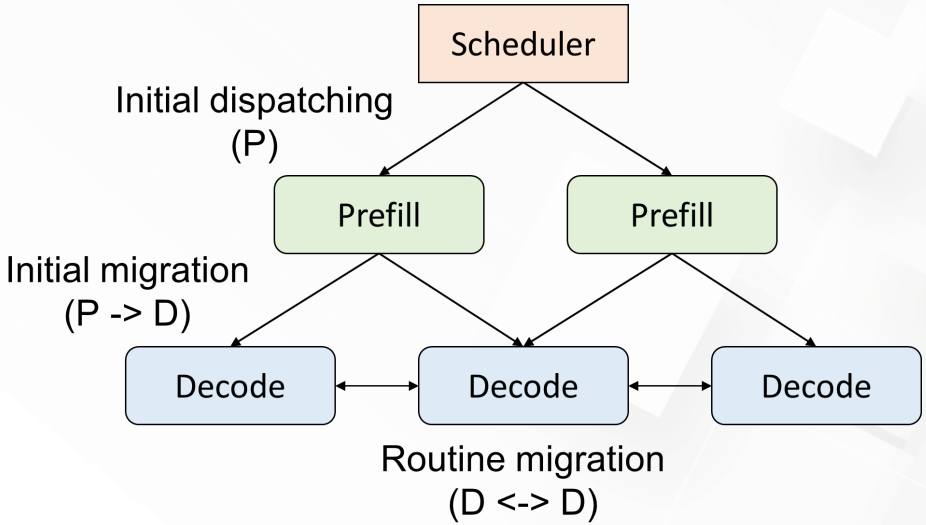
Scheduling-Defined Prefill-Decode Disaggregation

- P-D disaggregation is essentially a *request scheduling policy*
  - A special dispatching rule (P-instances-only)
  - A special migration rule (migrate after one step)



P-D Disaggregation in Llumnix

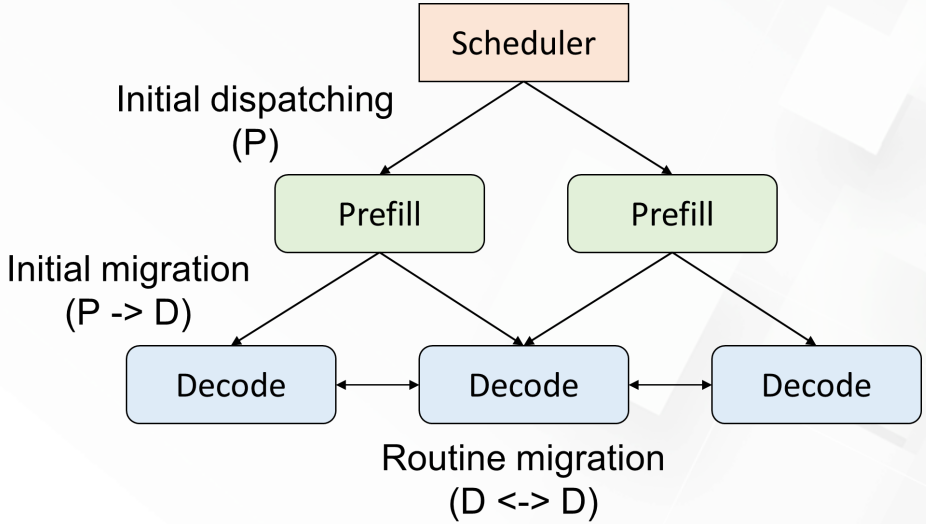
- Reuses most of the system-level mechanisms
  - KV cache transfer, token forwarding, fault tolerance, …
- Non-intrusive to inference engines
- *Seamless integrates with Llumnix's native scheduling capabilities*



Policies are already there!

P-D Disaggregation in Llumnix

- Reuses most of the system-level mechanisms
  - KV cache transfer, token forwarding, fault tolerance, …
- Non-intrusive to inference engines
- *Seamless integrates with Llumnix's native scheduling capabilities*



Policies are already there!

Conclusion

- Dynamic workloads need dynamic scheduling
  - LLMs are no exception

- Llumnix draws lessons from conventional systems wisdom
  - Classic scheduling goals in the new context of LLM serving
  - Implementation of rescheduling with request live migration
  - Continuous, dynamic rescheduling exploiting the migration

https://github.com/AlibabaPAI/llumnix

Thanks