DiffKV: Differentiated Memory Management for Large Language Models with Parallel KV Compaction

Yanqi Zhang, Yuwei Hu, Runyuan Zhao, John C. S. Lui, and Haibo Chen

Presenters: Chengru Yang, Jiawei Yi

Agenda

1 Background

2 Insights and Challenges

3 System Design

4 Evaluation and Conclusion

Agenda

1 Background

2 Insights and Challenges

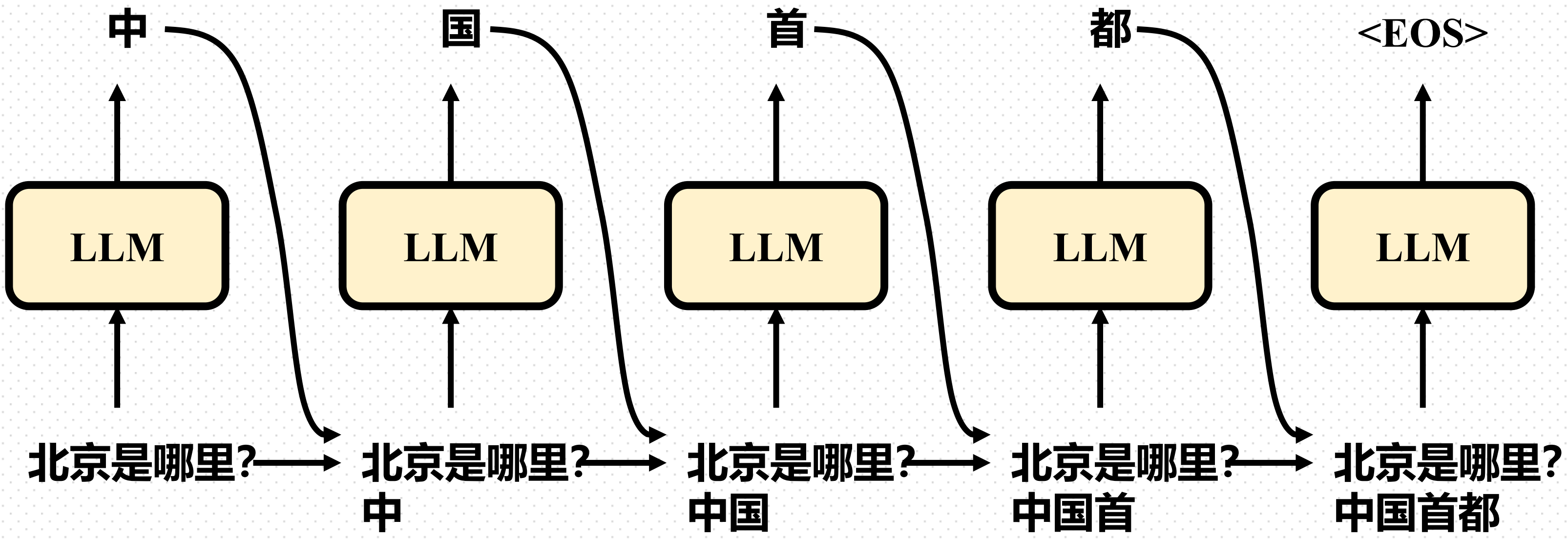
3 System Design

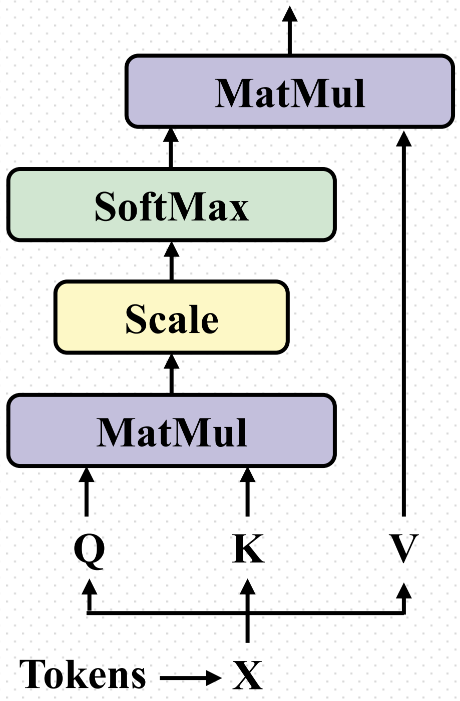
4 Evaluation and Conclusion

Background

❑Autoregressive LLM inference

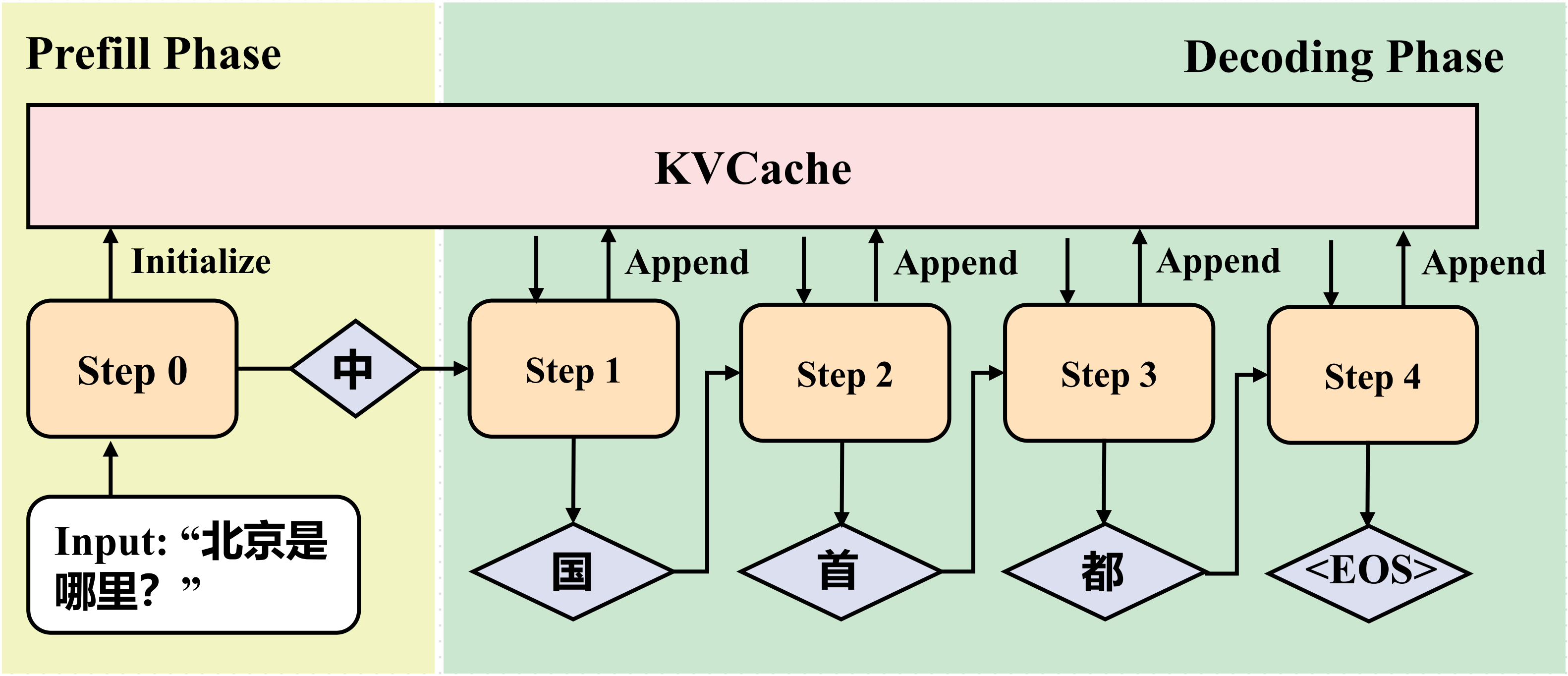
- ❖Generate new tokens step by step
- ❖Each step’s generation relies on all the input and generated tokens
- ❖Attention is computed from Query, Key and Value, all derived from tokens

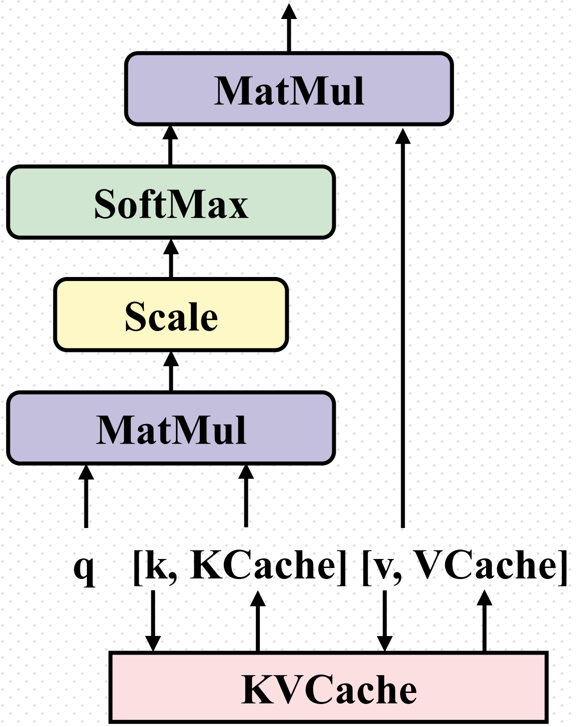




Background

- ❑KVCache: trade memory for inference efficiency
  - ❖Cache KV vectors to eliminate redundant computations
  - ❖A core component for LLM inference

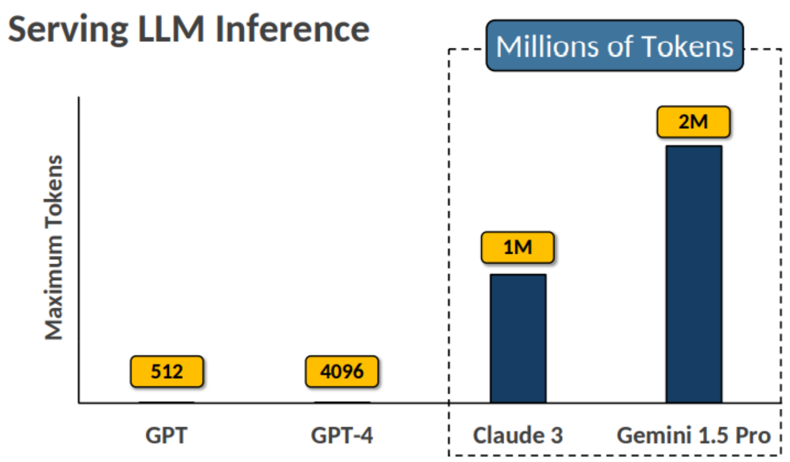




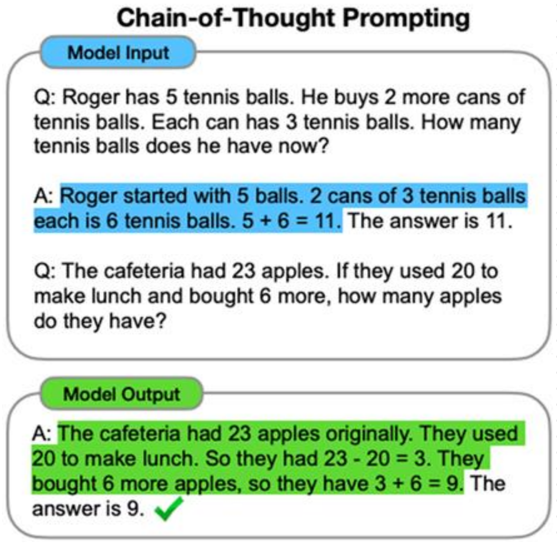
Background

❑Problem: high GPU memory footprint of KVCache

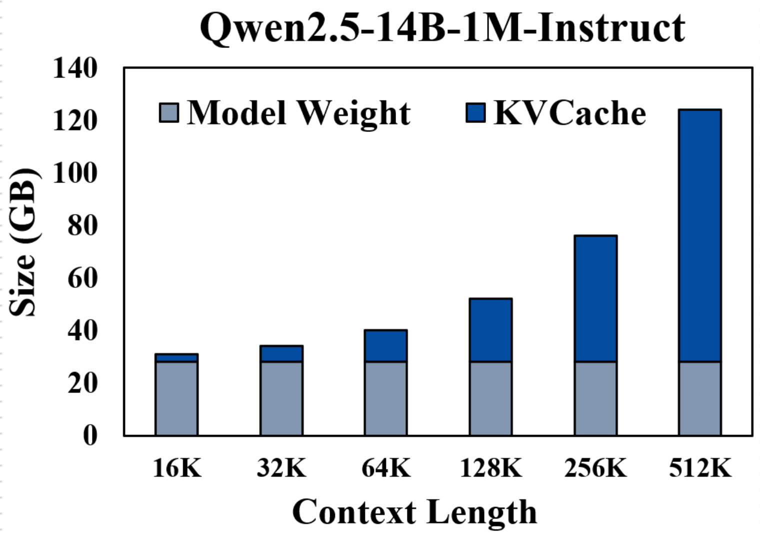
- ❖May exceed GPU memory capacity
- ❖High attention computation latency in bandwidth-bound decoding phase



Longer context window



Longer model outputs

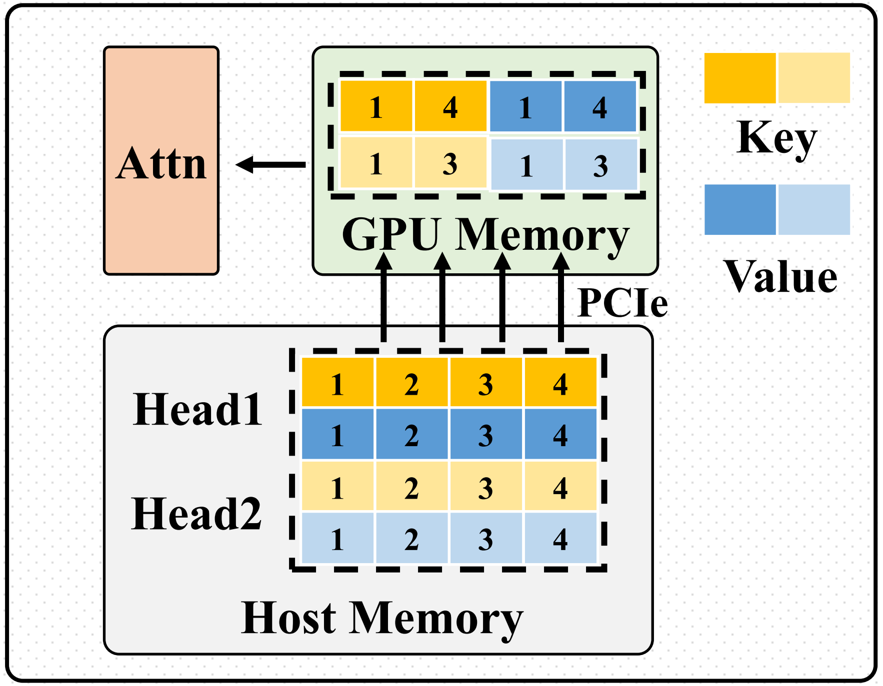


KVCache scales linearly with #tokens

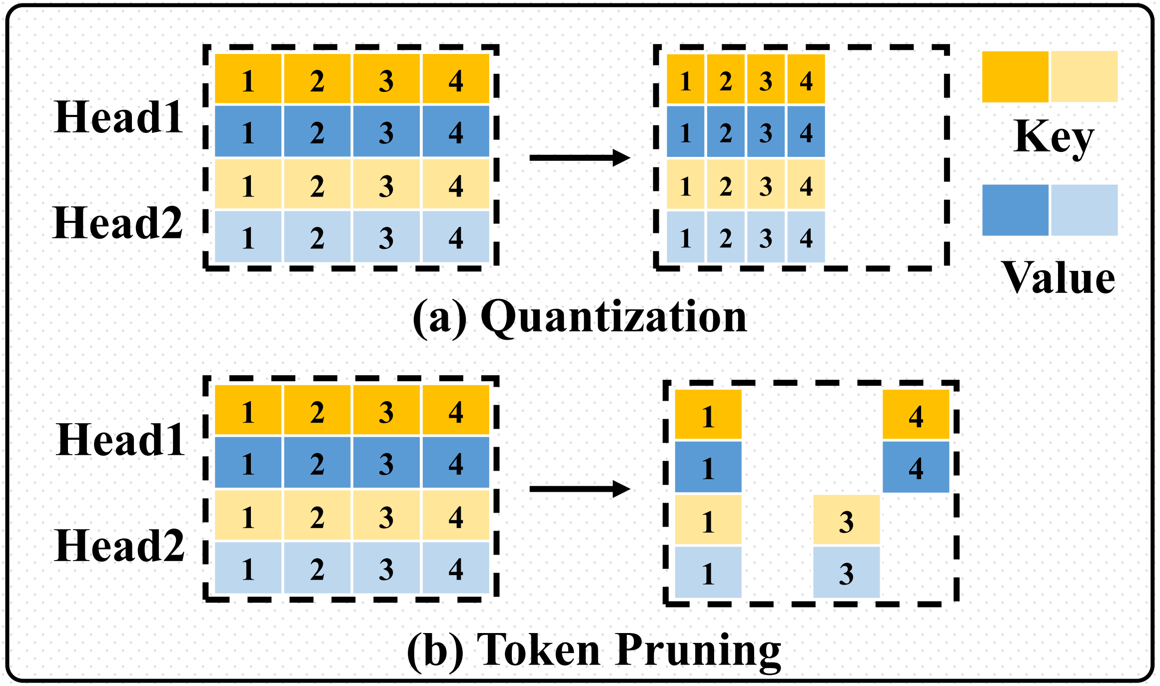
Background

❑KVCache memory footprint reduction approaches

- ❖KVCache offloading with sparse attention
- ❖KVCache compression



Offloading + Sparse attention



Compression

Background

- **This paper focuses on KVCache compression**
  - **How to compress KVCache?**
    - **Combine quantization and token pruning to form a hierarchical compression strategy**
  - **How to manage compressed KVCache?**
    - **Adapt to paged KVCache, a must for industrial practices**

Agenda

1 Background

2 Insights and Challenges

3 System Design

4 Evaluation and Conclusion

Insights and Challenges

- ❑Q1: How to compress KVCache?
- ❑Insights for KVCache compression
  - ❖Differentiated impacts of Keys and Values
  - ❖Differentiated token importance
  - ❖Differentiated attention head sparsity patterns

Insights and Challenges

❑Differentiated impacts of Keys and Values

$$\mathrm{Attn}(\mathrm{Q},\mathrm{K},\mathrm{V})_i = \sum_{j=1}^{i} \underbrace{\mathrm{softmax}\left(\frac{\mathrm{QK}^\top}{\sqrt{d}}\right)_{ij} |\mathbf{v}_j|}_{\text{Coefficient}} \underbrace{\frac{\mathbf{v}_j}{|\mathbf{v}_j|}}_{\text{Unit vector}}$$

Attention output is determined by:

- **Attention scores (impacted by Keys)**
- **Norm of Values (impacted by Values)**

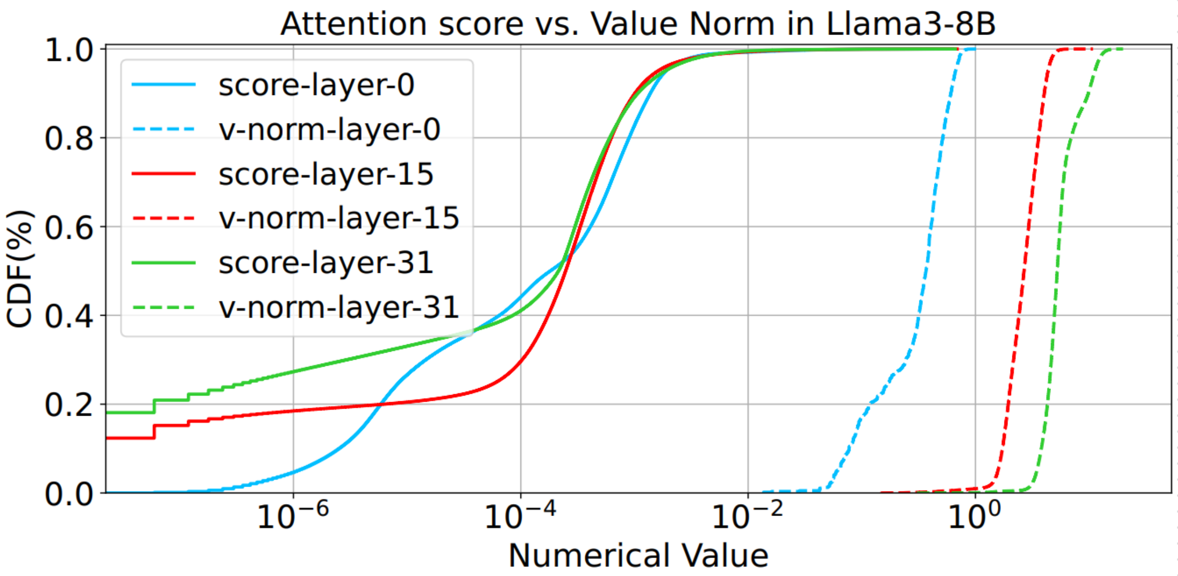
Insights and Challenges

❑Differentiated impacts of Keys and Values

$$\mathrm{Attn}(Q,K,V)_i = \sum_{j=1}^{i} \underbrace{\mathrm{softmax}\left(\frac{QK^\top}{\sqrt{d}}\right)_{ij} |\mathbf{v}_j|}_{\text{Coefficient}} \underbrace{\frac{\mathbf{v}_j}{|\mathbf{v}_j|}}_{\text{Unit vector}}$$

Attention output is determined by:

- **Attention scores (impacted by Keys)**
- **Norm of Values (impacted by Values)**

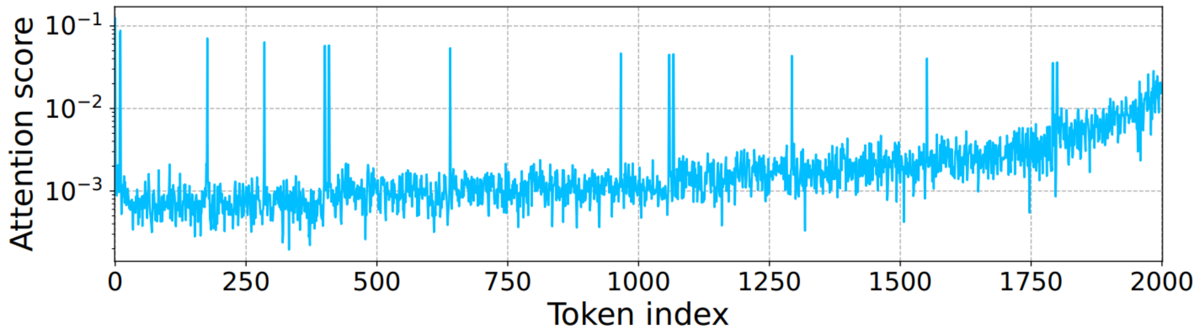


- **Attention scores spans from 10^{-8} to 10^{0}**
- **V-norm spans only from 10^{-2} to 10^{1}**

Higher quantization precision for Keys!

Insights and Challenges

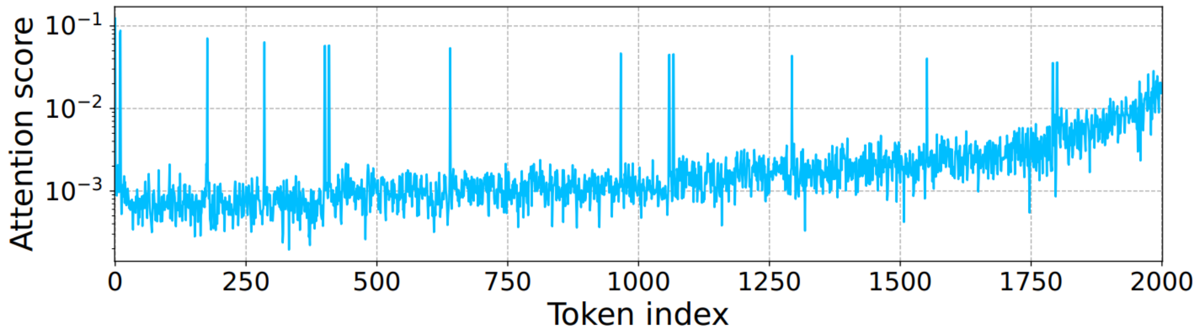
❑Differentiated token importance



- ➢ **High precision for the critical ones**
- ➢ **Low precision for less-critical ones**
- ➢ **Pruning for the least-critical ones**

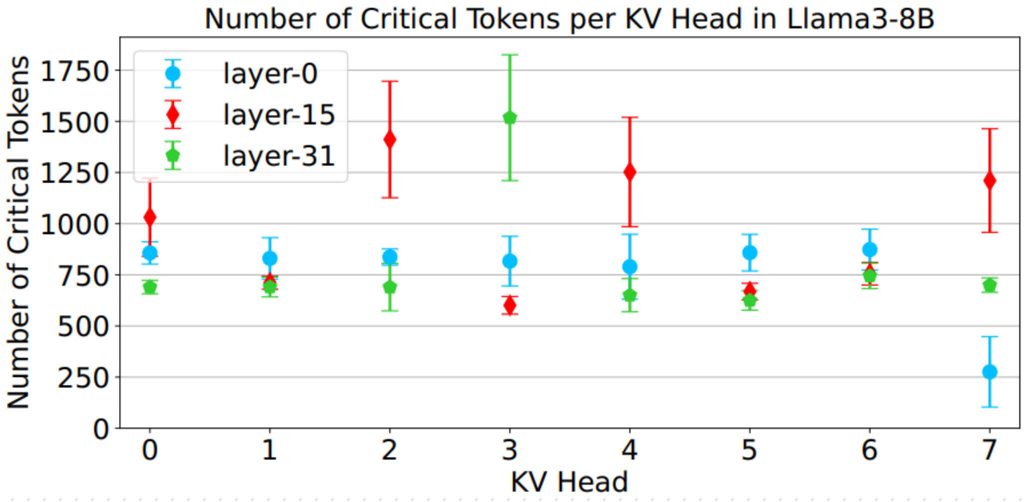
Insights and Challenges

❑Differentiated token importance



- ➢ **High precision for the critical ones**
- ➢ **Low precision for less-critical ones**
- ➢ **Pruning for the least-critical ones**

❑Differentiated, dynamic attention head sparsity patterns



Sparsity patterns vary across requests, heads

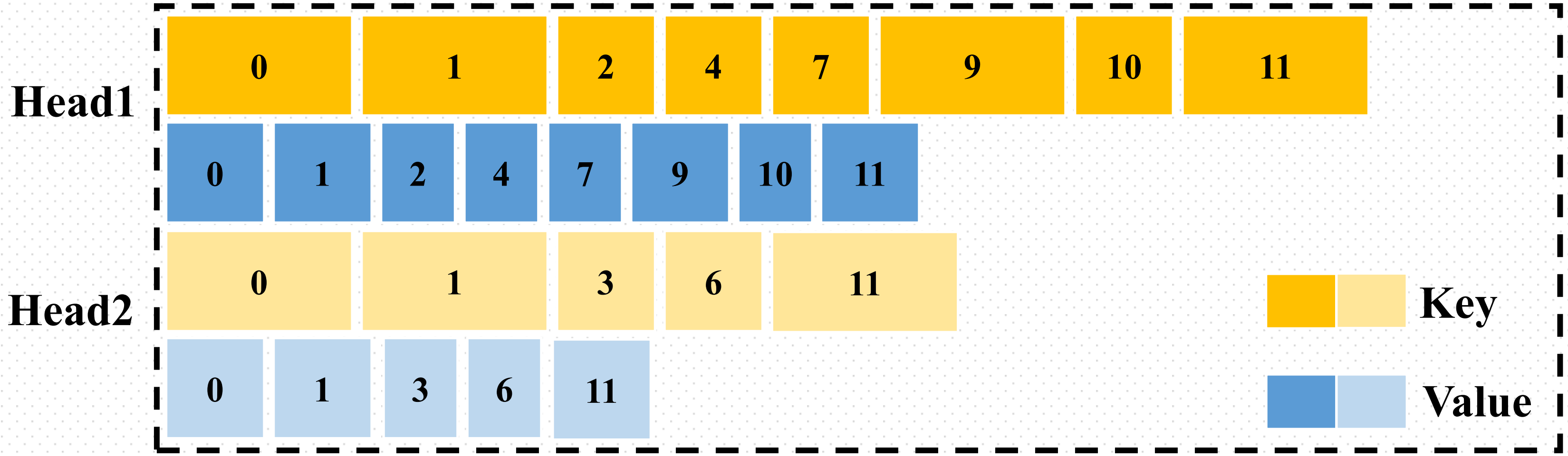
- ➢ **A dynamic, head-wise compression strategy is required**

Insights and Challenges

- Q1: How to compress KVCache?
- Insights for KVCache compression
  - Differentiated impacts of Keys and Values
    - Different quantization precision for keys and values
  - Differentiated token importance
    - Hierarchical compression strategy for tokens of different importance
  - Differentiated, dynamic attention head sparsity patterns
    - Dynamic, head-wise compression

Insights and Challenges

- Q2: How to manage compressed KVCache?
- Challenge for adaption to paged KVCache
  - Differentiated memory layout of Keys, Values, tokens and attention heads



  - How to design a scalable, GPU-based page management mechanism that minimizes memory fragmentation?

Agenda

1 Background

2 Insights and Challenges

3 System Design

4 Evaluation and Conclusion

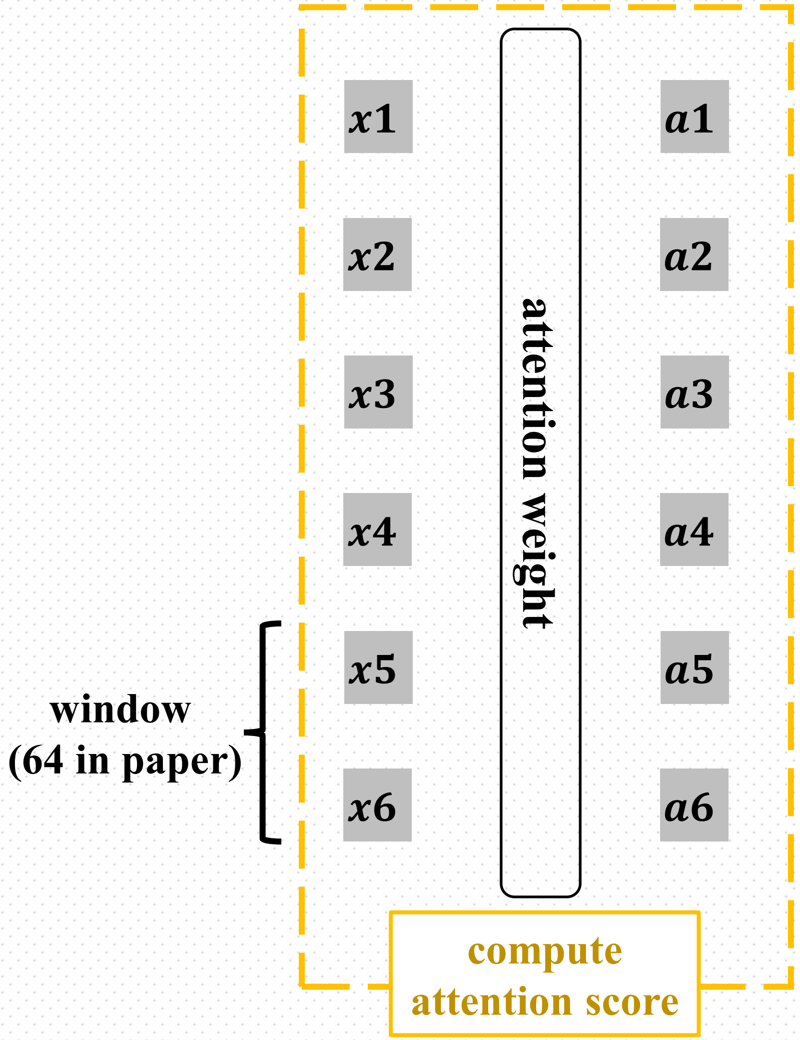
System Design

❑KV Compaction Policy (prefill)

- ❖Compute attention score
  - ➢Recent window to keep all token’s KV in window
- ❖Compute token significance
  - ➢Token significance is calculated by averaging the following tokens’ attention socre to it
- ❖Choose compaction strategies
  - ➢Alow and Ahigh are analyzed offline for each model
  - ➢For ith token, Alow and Ahigh are devided by i

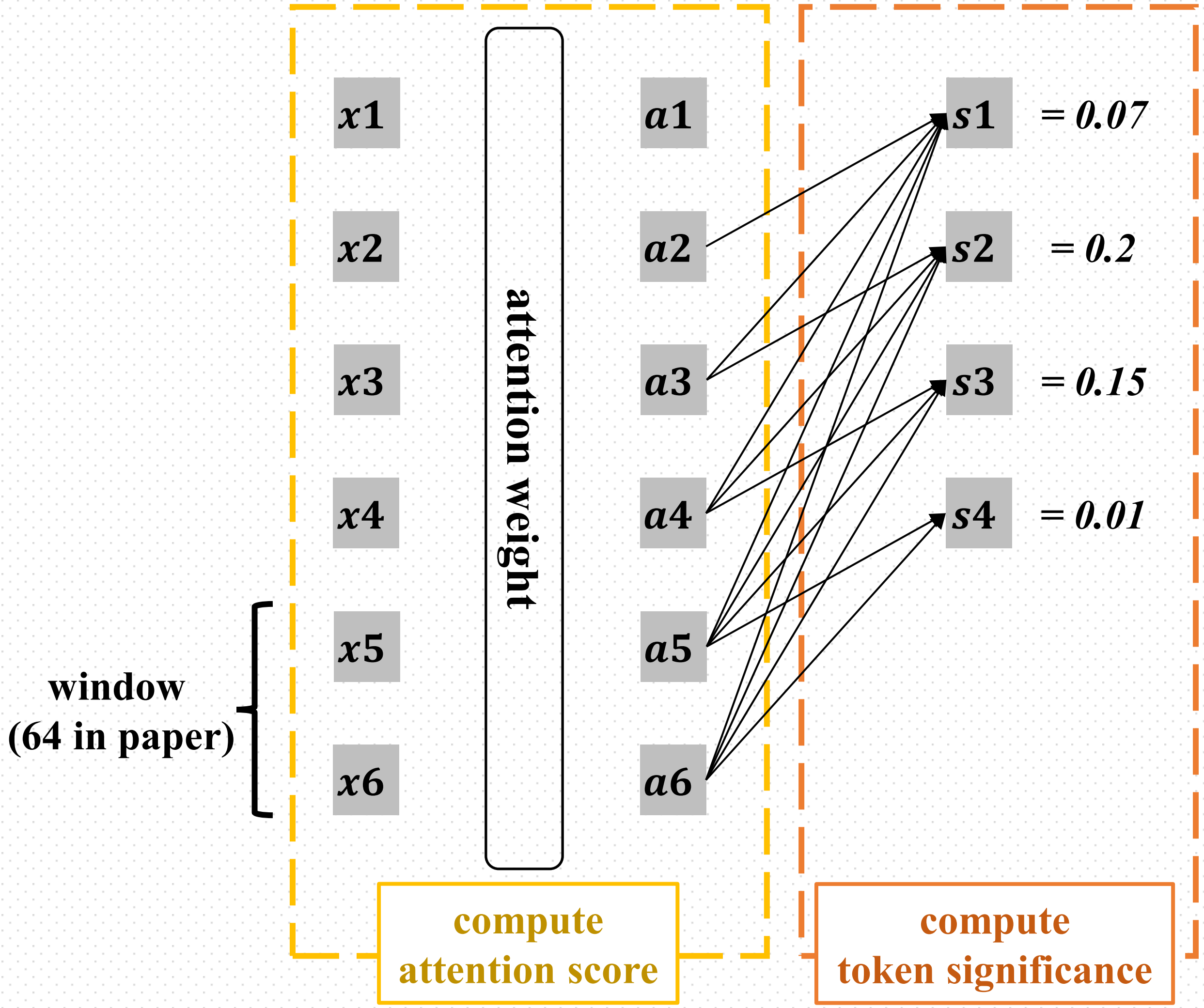
System Design

❑KV Compaction Policy (prefill)



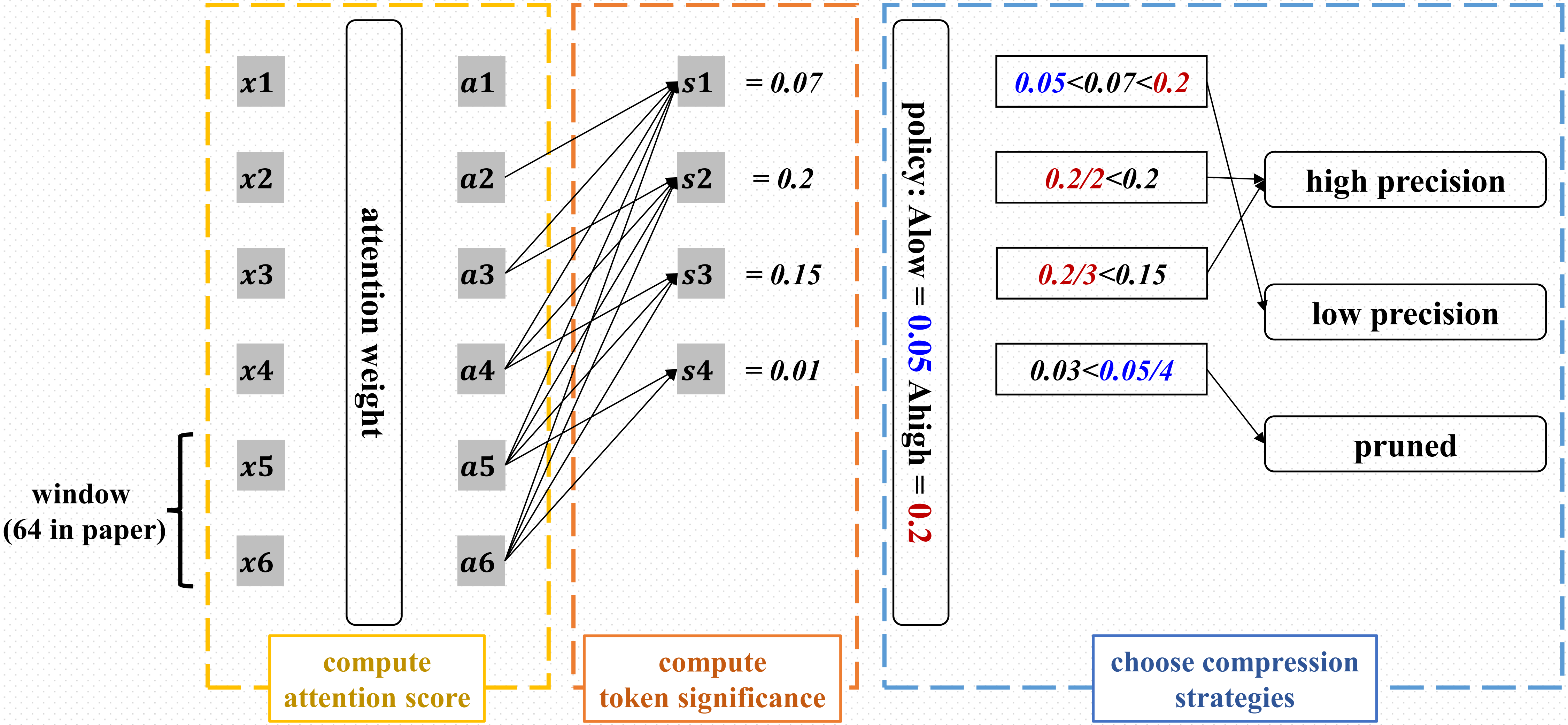
System Design

❑KV Compaction Policy (prefill)



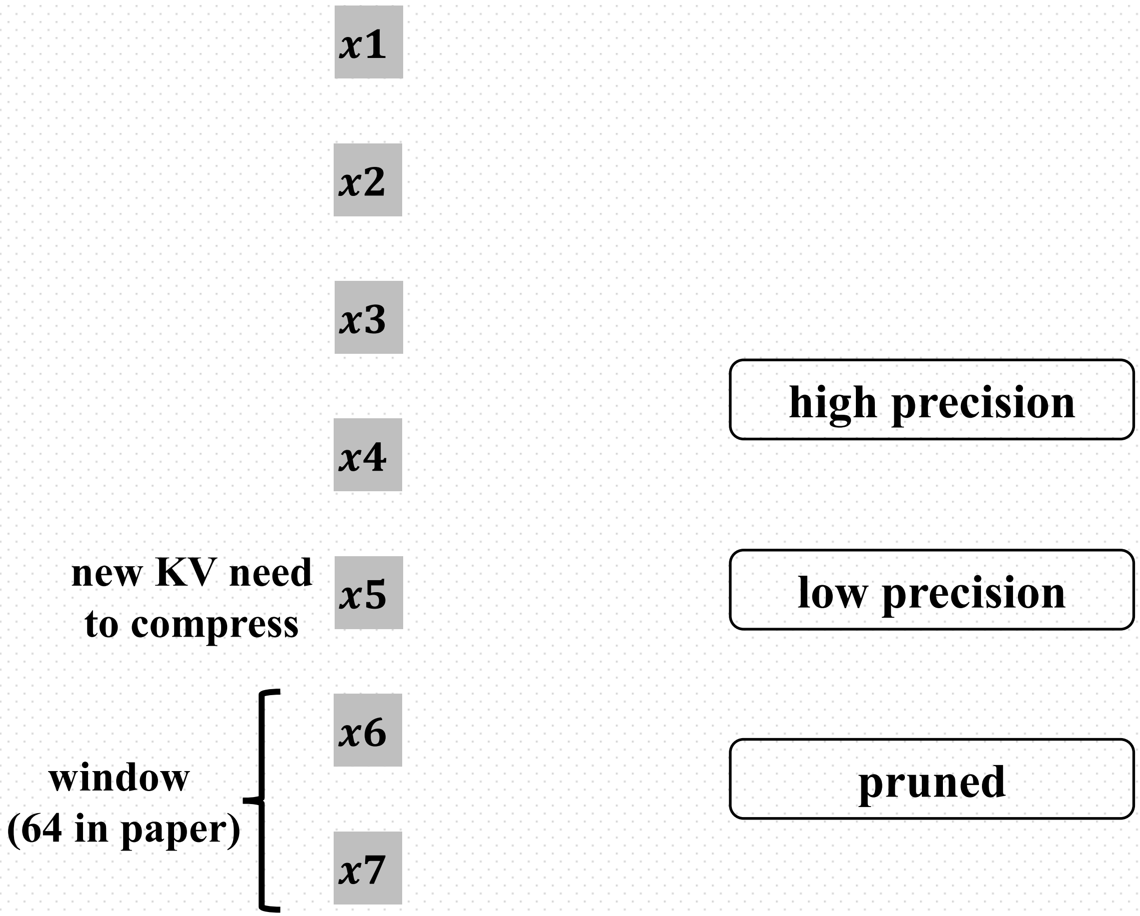
System Design

❑KV Compaction Policy (prefill)



System Design

❑KV Compaction Policy (decode)

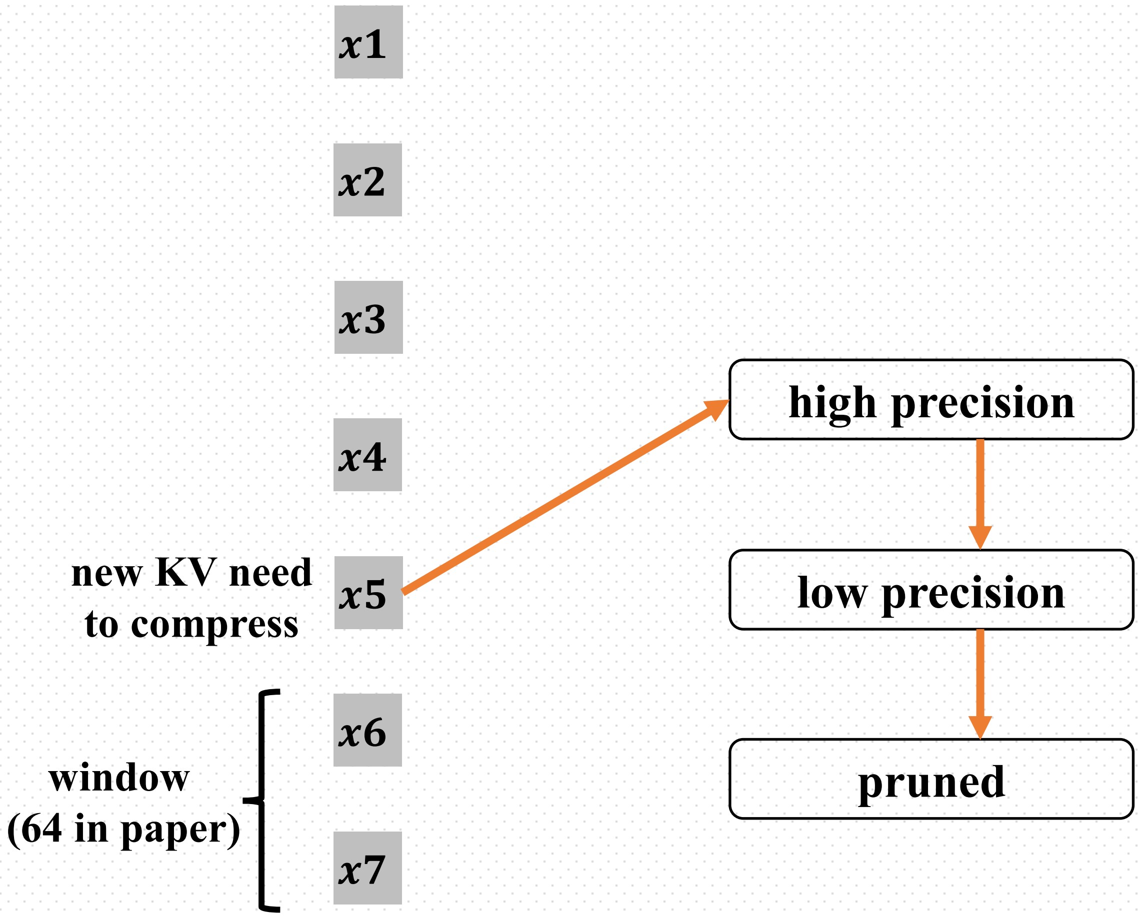


Algorithm 1: KV compression policy (generation)

```
Input: Parameters α_h, α_l; High & low precision P_h & P_l
Input: Candidate token t_c; Sequence length N
Input: High & low precision KV cache KV_h & KV_l
Function: Significance Score; Quantization Quant;
if Score(t_c) ≥ α_h/N then
    KV_h.add(Quant(t_c, P_h))
    t_v = argmin_{t∈KV_h}(Score(t))
    if α_l/N ≤ Score(t_v) < α_h/N then
        KV_h.remove(t_v),  KV_l.add(Quant(t_v, P_l))
    else if Score(t_v) < α_l/N then
        KV_h.remove(t_v)
    end if
else if Score(t_c) ≥ α_l/N then
    KV_l.add(Quant(t_c, P_l))
    t_v = argmin_{t∈KV_l}(Score(t))
    if Score (t_v) < α_l/N then
        KV_l.remove(t_v)
    end if
end if
```

System Design

❑KV Compaction Policy (decode)

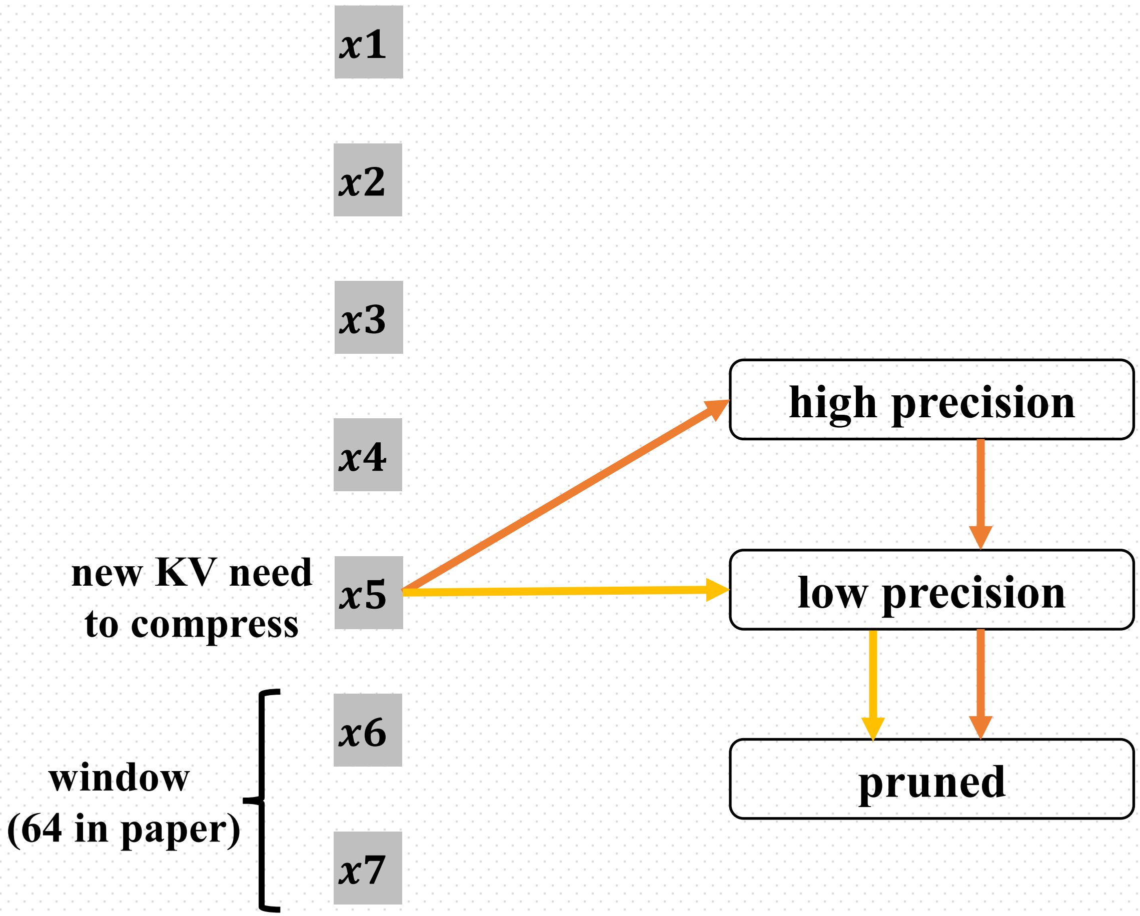


Algorithm 1: KV compression policy (generation)

Input: Parameters α_h, α_l; High & low precision P_h & P_l
Input: Candidate token t_c; Sequence length N
Input: High & low precision KV cache KV_h & KV_l
Function: Significance *Score*; Quantization *Quant*;
if Score$(t_c) \geq \frac{\alpha_h}{N}$ **then**
    KV_h.add(Quant(t_c, P_h))
    $t_v = argmin_{t \in KV_h}$(Score$(t)$)
    if $\frac{\alpha_l}{N} \leq$ Score$(t_v) < \frac{\alpha_h}{N}$ **then**
        KV_h.remove(t_v), KV_l.add(Quant(t_v, P_l))
    else if Score$(t_v) < \frac{\alpha_l}{N}$ **then**
        KV_h.remove(t_v)
    end if
else if Score$(t_c) \geq \frac{\alpha_l}{N}$ **then**
    KV_l.add(Quant(t_c, P_l))
    $t_v = argmin_{t \in KV_l}$(Score$(t)$)
    if Score $(t_v) < \frac{\alpha_l}{N}$ **then**
        KV_l.remove(t_v)
    end if
end if

System Design

❑KV Compaction Policy (decode)

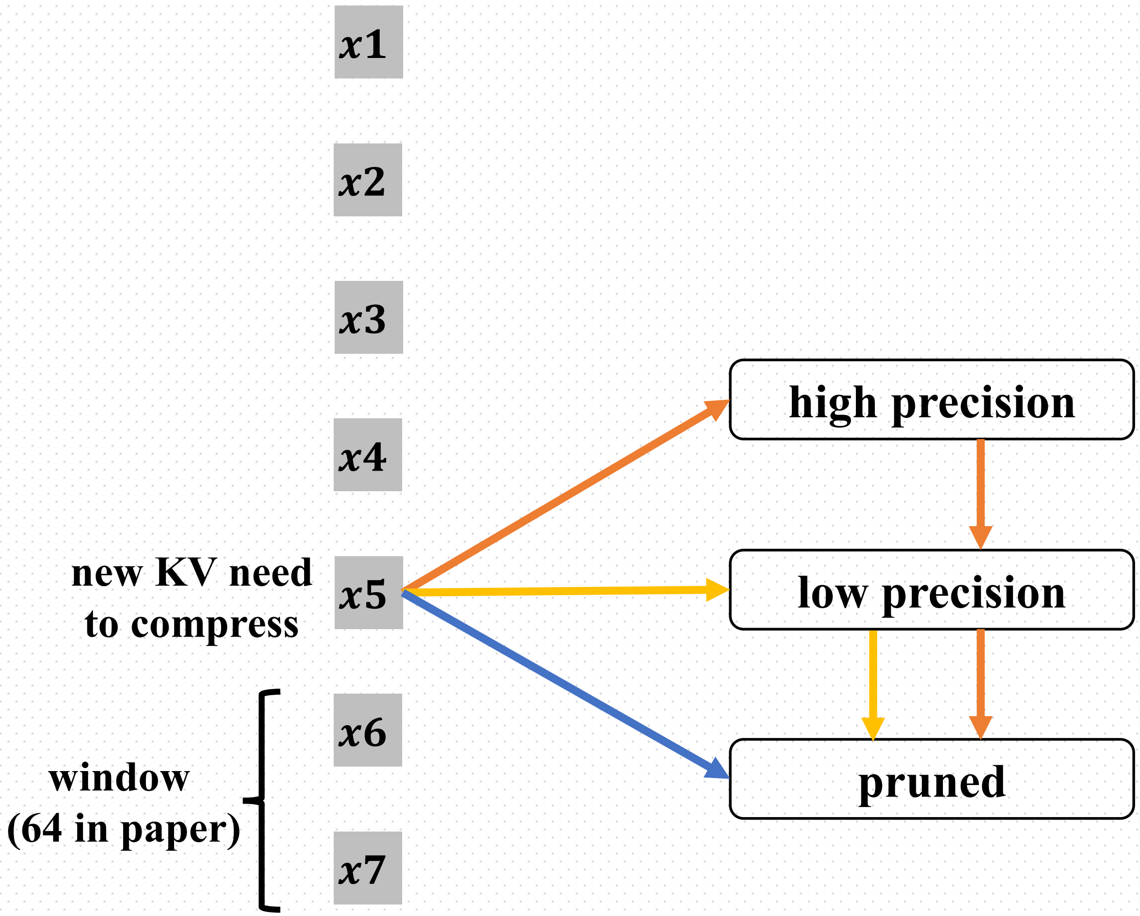


Algorithm 1: KV compression policy (generation)

Input: Parameters α_h, α_l; High & low precision P_h & P_l
Input: Candidate token t_c; Sequence length N
Input: High & low precision KV cache KV_h & KV_l
Function: Significance *Score*; Quantization *Quant*;
if Score$(t_c) \geq \frac{\alpha_h}{N}$ **then**
  KV_h.add(Quant(t_c, P_h))
  $t_v = argmin_{t \in KV_h}$(Score$(t)$)
  if $\frac{\alpha_l}{N} \leq$ Score$(t_v) < \frac{\alpha_h}{N}$ **then**
    KV_h.remove(t_v), KV_l.add(Quant(t_v, P_l))
  else if Score$(t_v) < \frac{\alpha_l}{N}$ **then**
    KV_h.remove(t_v)
  end if
else if Score$(t_c) \geq \frac{\alpha_l}{N}$ **then**
  KV_l.add(Quant(t_c, P_l))
  $t_v = argmin_{t \in KV_l}$(Score$(t)$)
  if Score $(t_v) < \frac{\alpha_l}{N}$ **then**
    KV_l.remove(t_v)
  end if
end if

System Design

❑KV Compaction Policy (decode)



Algorithm 1: KV compression policy (generation)

Input: Parameters α_h, α_l; High & low precision P_h & P_l
Input: Candidate token t_c; Sequence length N
Input: High & low precision KV cache KV_h & KV_l
Function: Significance *Score*; Quantization *Quant*;
if Score$(t_c) \geq \frac{\alpha_h}{N}$ **then**
  KV_h.add(Quant(t_c, P_h))
  $t_v = argmin_{t \in KV_h}$(Score$(t)$)
  if $\frac{\alpha_l}{N} \leq$ Score$(t_v) < \frac{\alpha_h}{N}$ **then**
    KV_h.remove(t_v), KV_l.add(Quant(t_v, P_l))
  else if Score$(t_v) < \frac{\alpha_l}{N}$ **then**
    KV_h.remove(t_v)
  end if
else if Score$(t_c) \geq \frac{\alpha_l}{N}$ **then**
  KV_l.add(Quant(t_c, P_l))
  $t_v = argmin_{t \in KV_l}$(Score$(t)$)
  if Score $(t_v) < \frac{\alpha_l}{N}$ **then**
    KV_l.remove(t_v)
  end if
end if

System Design

❑Data structure for memory management

Unified Pages

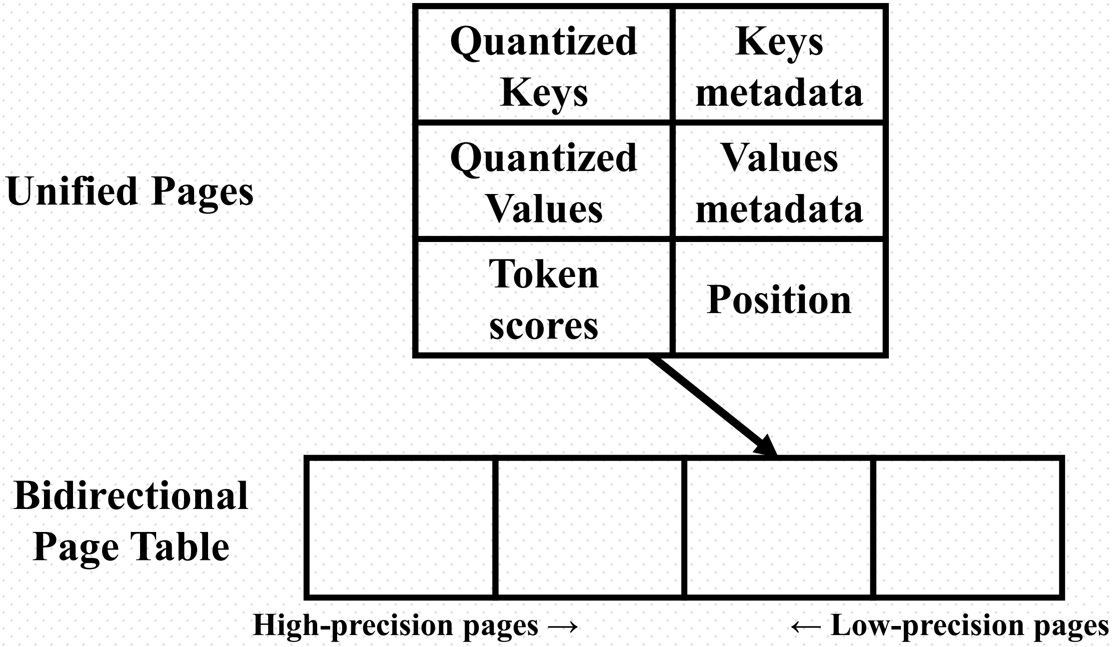
Quantized Keys	Keys metadata
Quantized Values	Values metadata
Token scores	Position

GPU memory is partitioned in to:

- ➢ Six data sigments for Keys and Values
- ➢ Tokens per page vary with quantization settings

System Design

❑Data structure for memory management



GPU memory is partitioned in to:

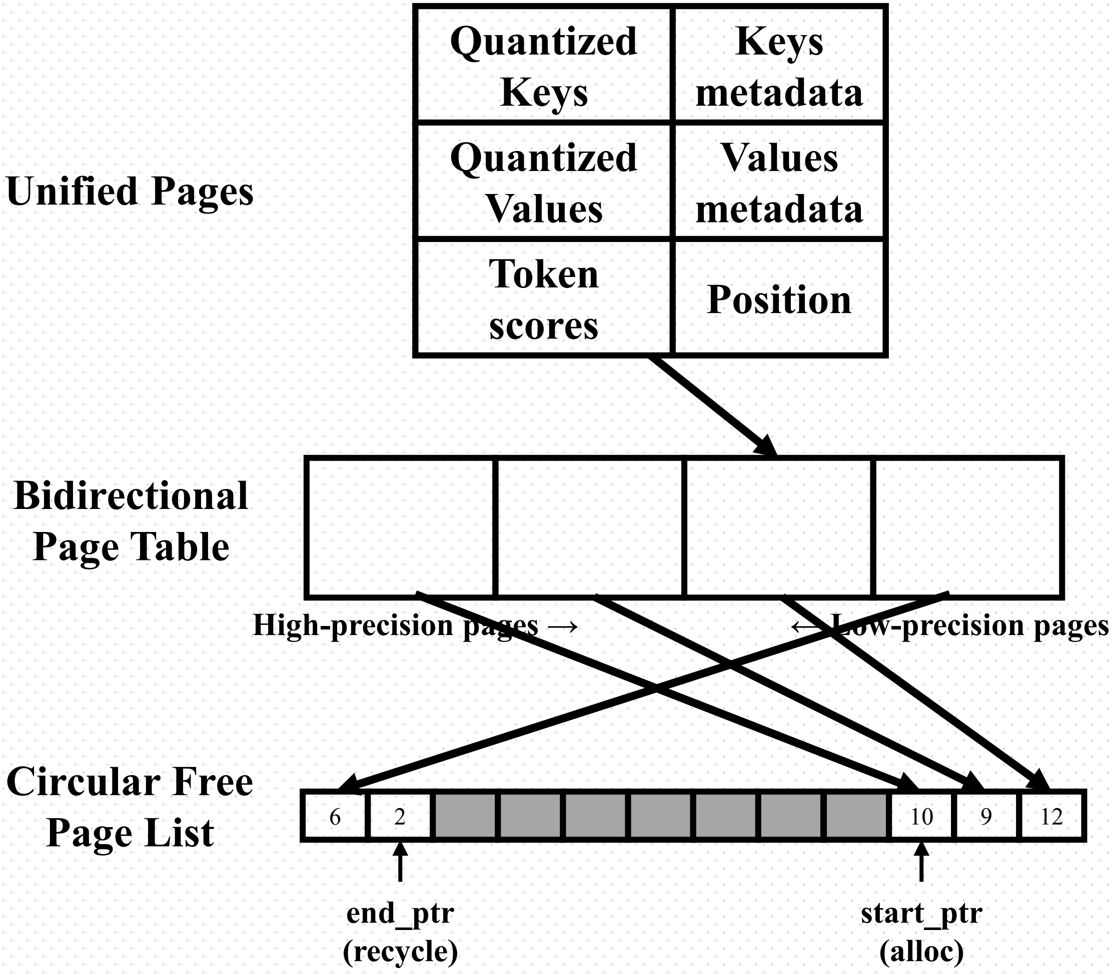
- Six data sigments for Keys and Values
- Tokens per page vary with quantization settings

Page Table for per-head and per-request:

- Avoid duplicated metadata for different precisions
- Entry size uses high-precision pages to prevent overflow

System Design

❑Data structure for memory management



GPU memory is partitioned in to:

- Six data sigments for Keys and Values
- Tokens per page vary with quantization settings

Page Table for per-head and per-request:

- Avoid duplicated metadata for different precisions
- Entry size uses high-precision pages to prevent overflow

Circular page list for parallel KV compaction:

- Two pointers for page allocation and recycling
- Use parallel prefix-sum to alloc and recycle

System Design (workflow)

Request:

$x1$

$x2$

$x3$

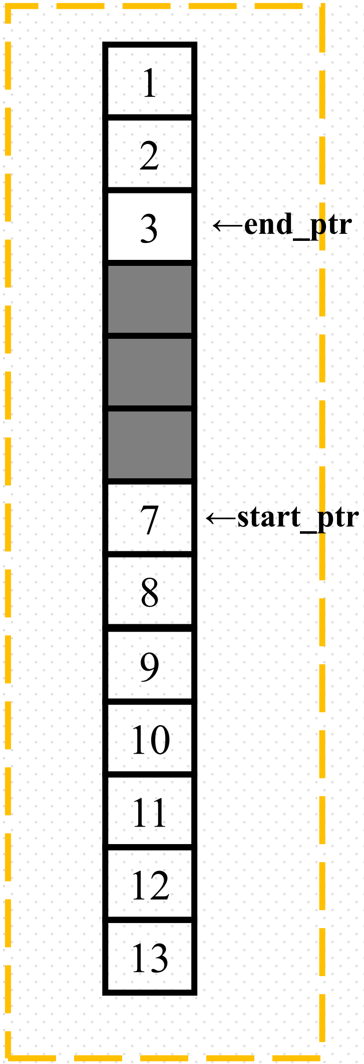
$x4$

$x5$

$x6$

$x7$

$x8$



Initial State

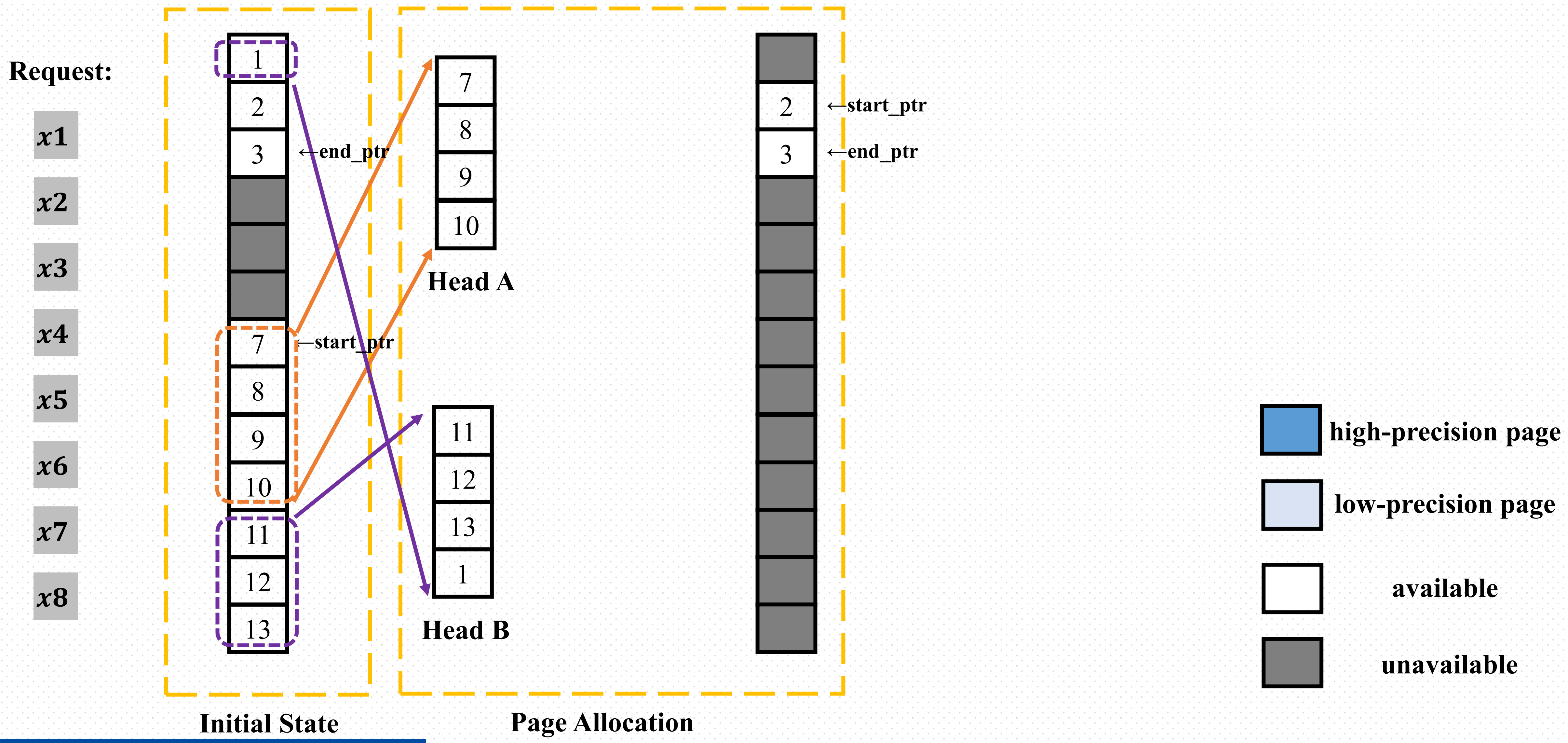
high-precision page

low-precision page

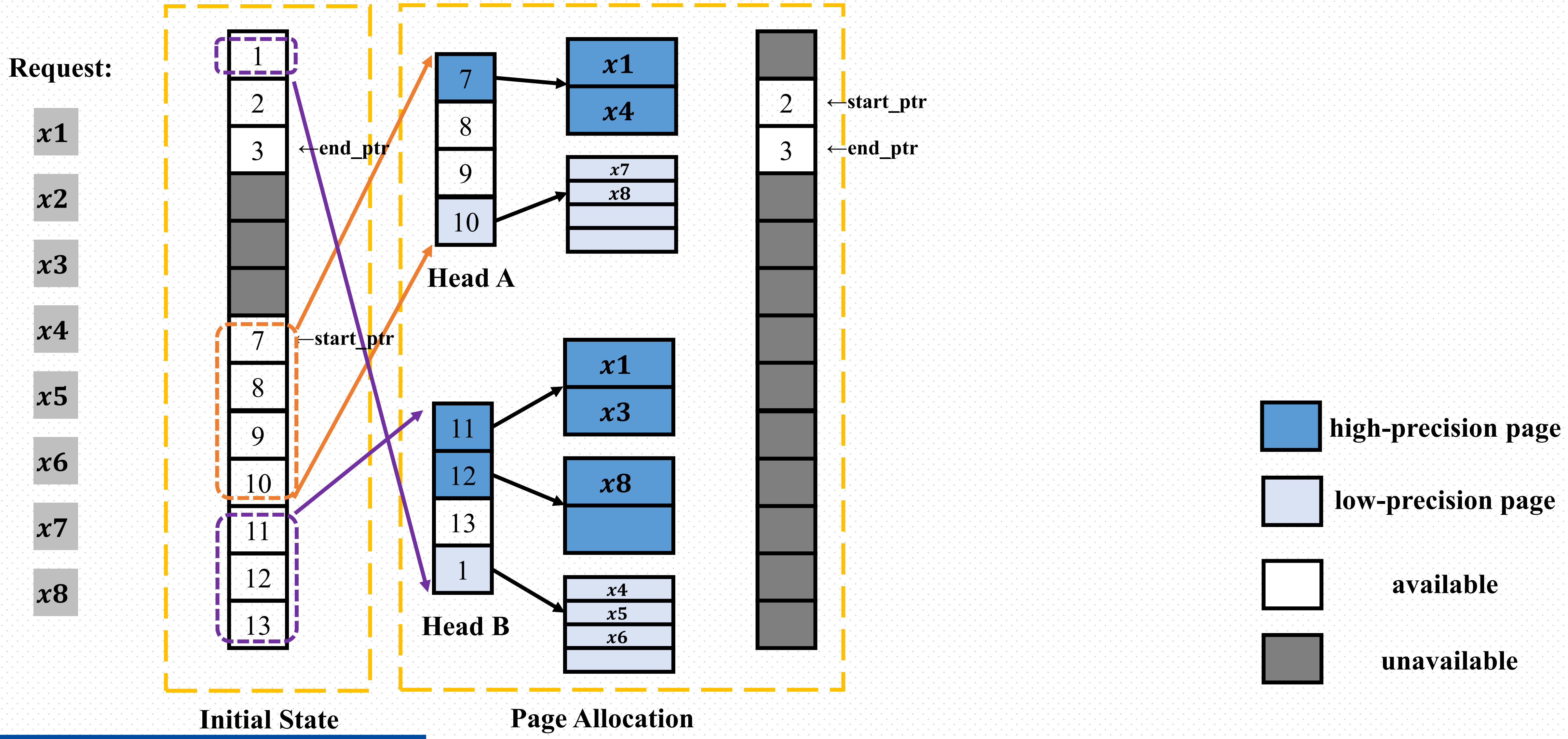
available

unavailable

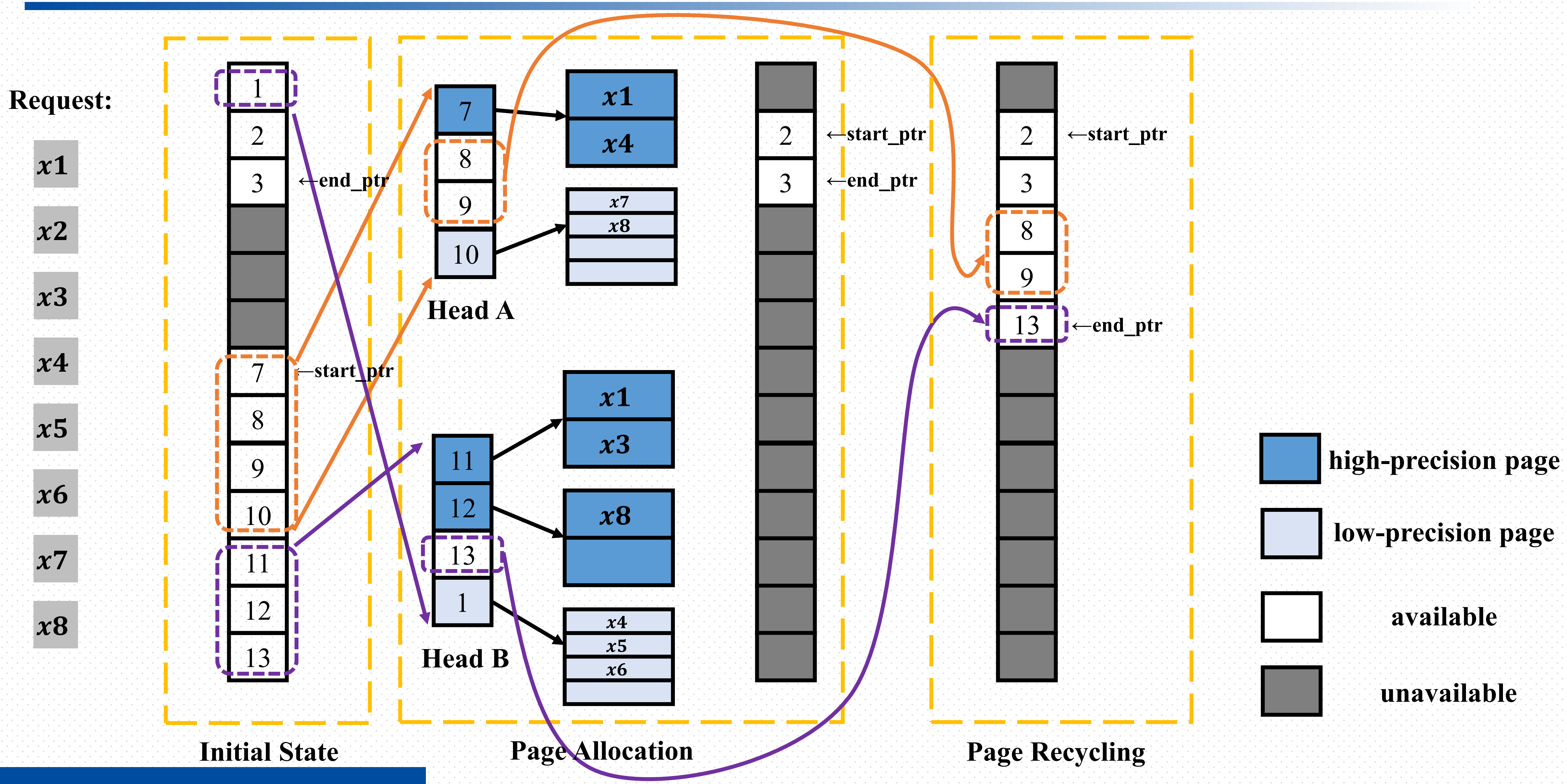
System Design (workflow)



System Design (workflow)



System Design (workflow)



Agenda

1 Background

2 Insights and Challenges

3 System Design

4 Evaluation and Conclusion

Evaluation

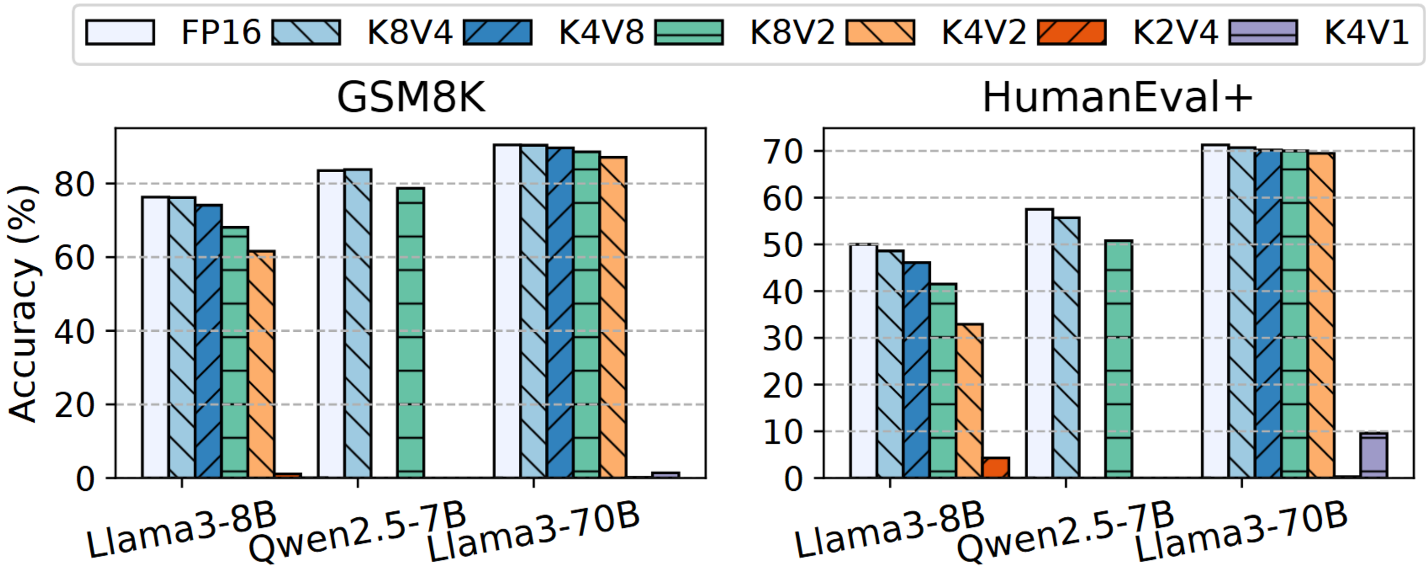
❑Evaluation Setup

- ❖Models: Llama3-8B/70B、Qwen2.5-7B/32B 、QwQ-32B、R1-Distill-Qwen-14B、R1-Distill-Llama-8B
- ❖Device: NVIDIA L40GPU (48GB)
- ❖Evaluation metrics: accuracy / score + throughput / latency
- ❖Weights are stored in FP16 precision

Evaluation

❑Differentiated KV Compression Policy

❖Differentiated KV Quantization



- K8V4 ≈ FP16
- K8V4 > K4V8 (Qwen2.5-7B)
- K4V2 can keep some acc
- Lower bound for V is 2 bit

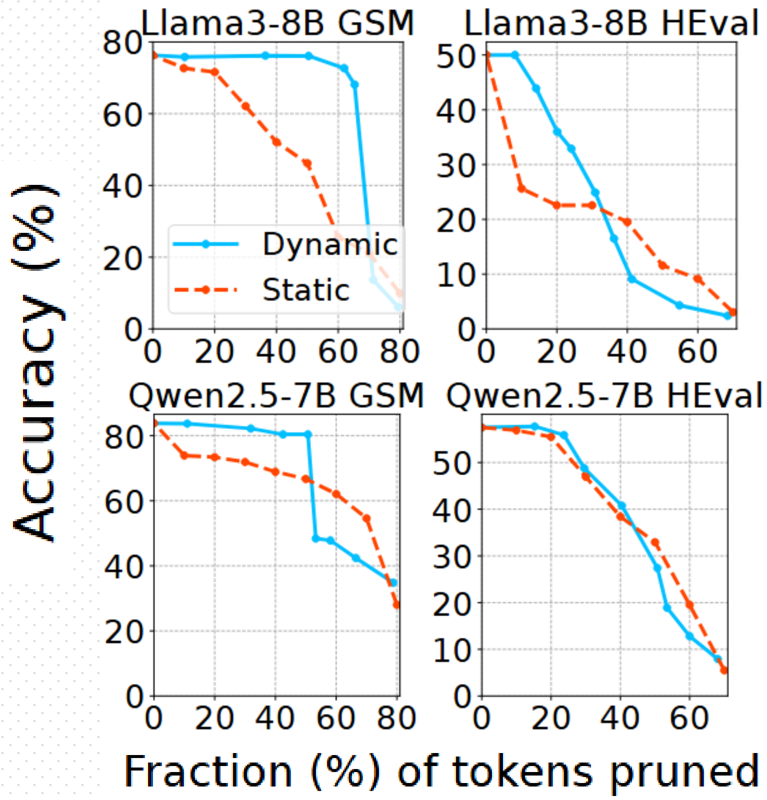
Choose K8V4 for high precision

K4V2 for low precision

Evaluation

❑Differentiated KV Compression Policy

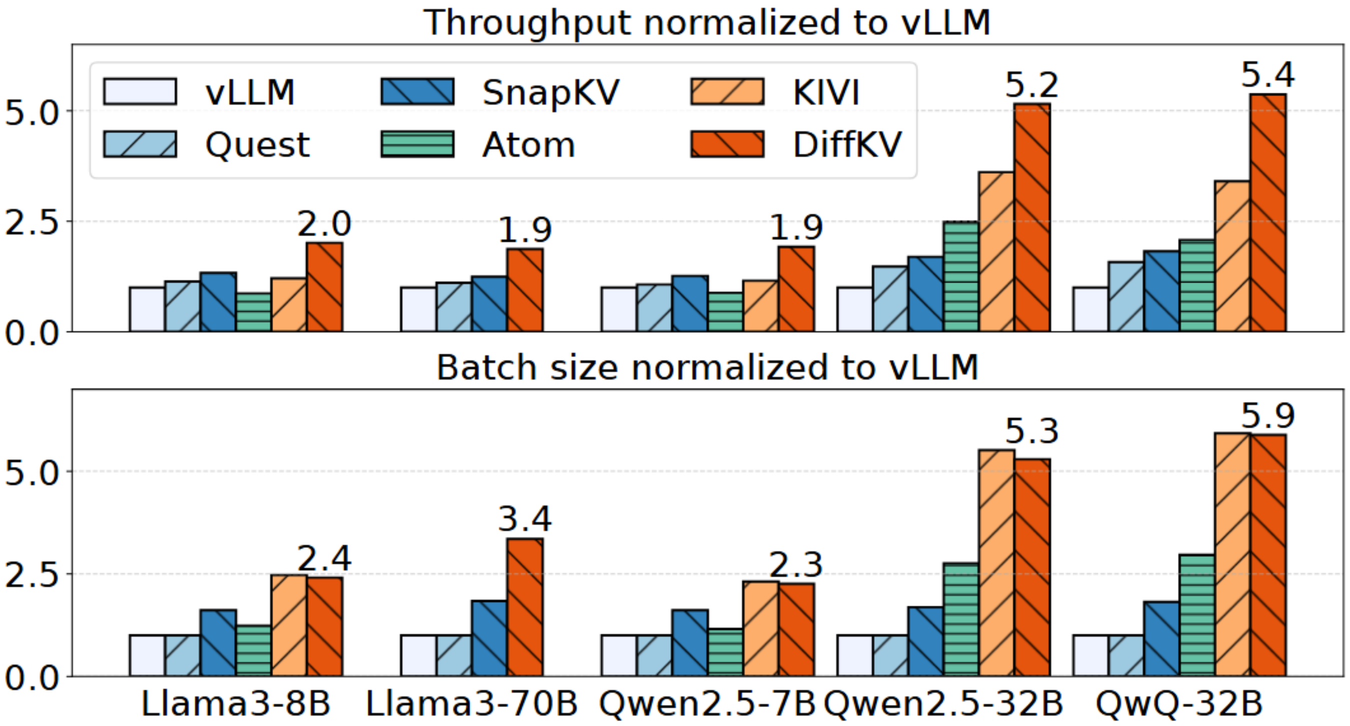
❖Dynamic Sparsity for heads and requests



- Dynamic: DiffKV sets different pruning rate for per-request and per-head under total pruning rate of a static value
- Static: DiffKV set same pruning rate for all requests and heads
- Dynamic always perform better than Static when pruning rate less than 50%

Evaluation

❑System Performance



DiffKV achieves highest throughput than others

- ➢ **Quest only compute significant attention**
- ➢ **SnapKV pruns insignificant token**
- ➢ **Atom use 4-bit quantization**
- ➢ **KIVI use 2-bit quantization**

Conclusion

- **Problem:**
  - **KV cache dominates GPU memory; existing quantization/pruning is coarse-grained and inefficient, limiting batch size and throughput.**
- **Key Findings:**
  - **Keys matter more than values for quality**
  - **Different requests, heads and tokens matter**
- **Solution:**
  - **Differentiated KV compression (K/V mixed precision + per-head and per-request dynamic compaction strategy)**

Thank you!

Presenters: Chengru Yang, Jiawei Yi