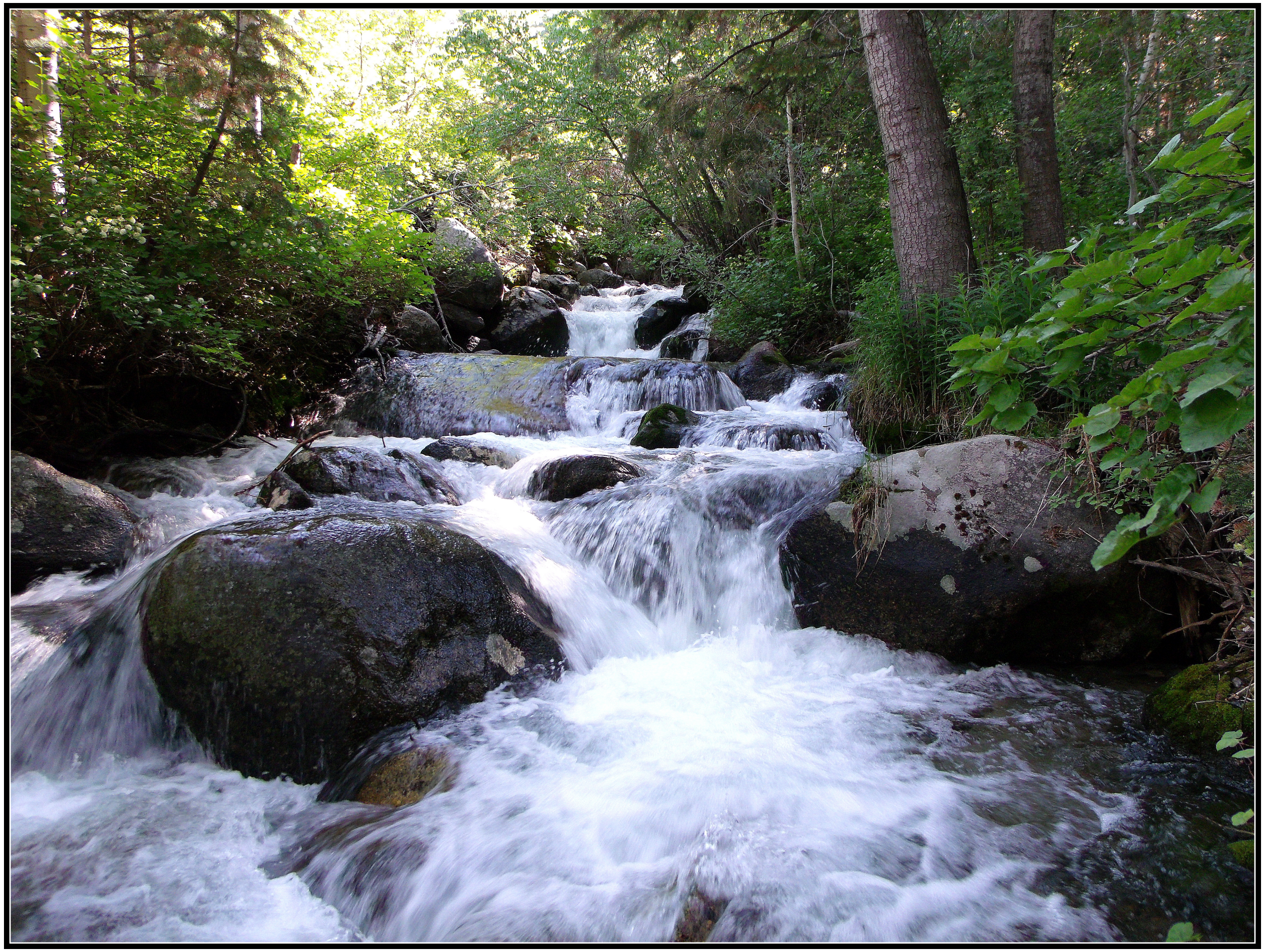
When Two Choices Are not Enough: Balancing at Scale in Distributed Stream Processing

Anis Nasir

Accepted at ICDE 2016, available at arXiv

Stream Processing



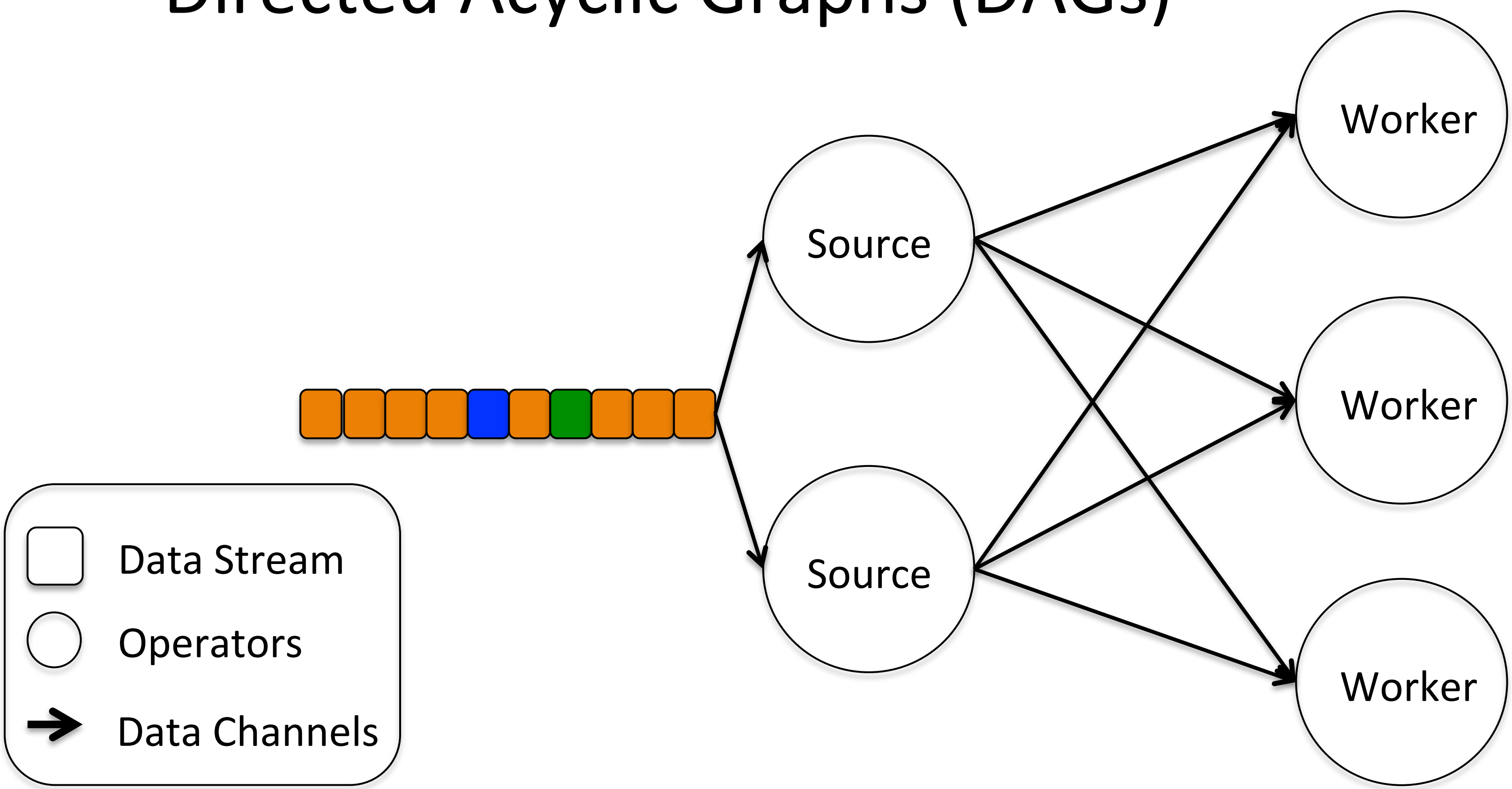
Stream Processing Engines

- Streaming Application
  - Online Machine Learning
  - Real Time Query Processing
  - Continuous Computation

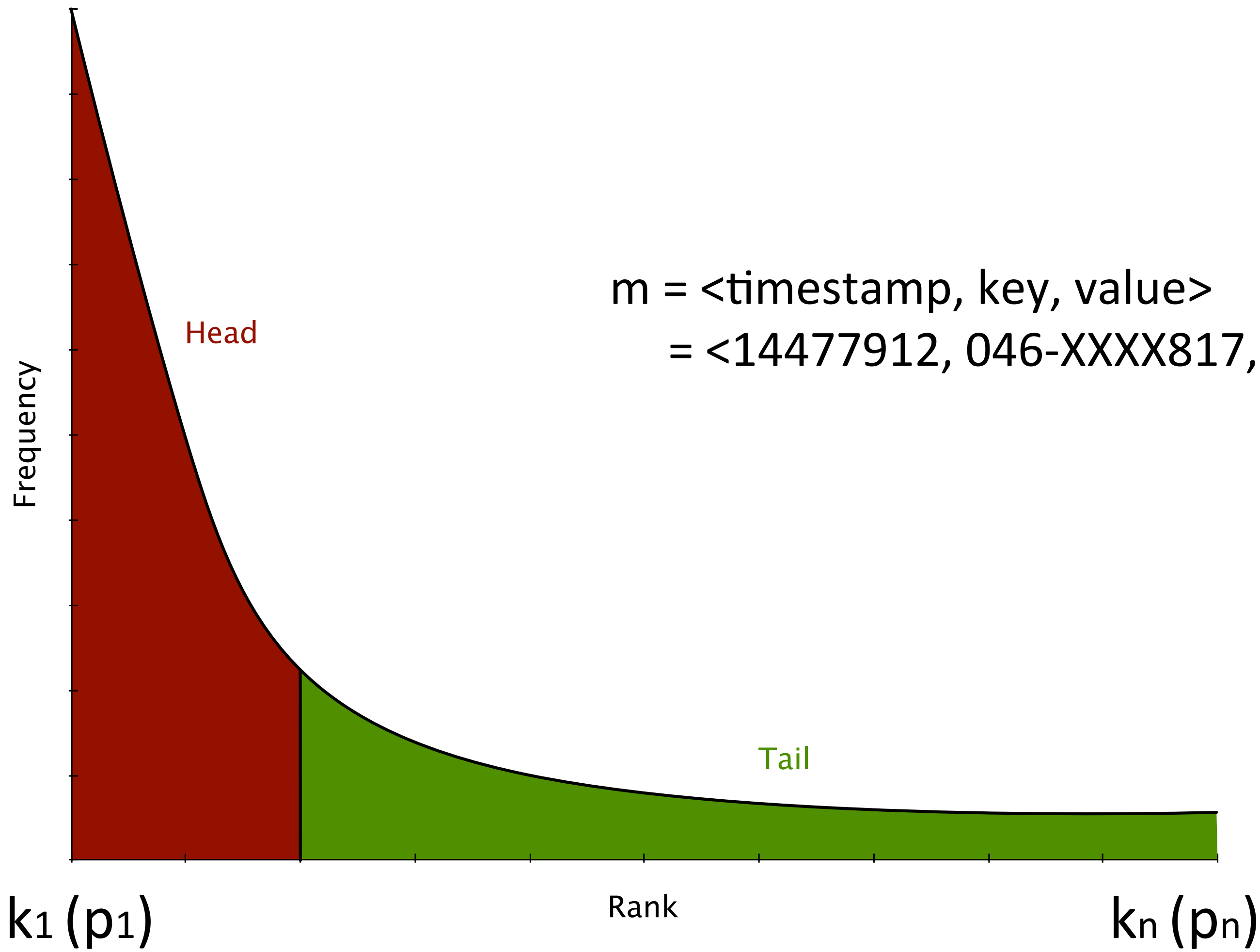
- Streaming Frameworks
  - Storm, S4, Flink Streaming, Spark Streaming

Stream Processing Model

- Streaming Applications are represented by Directed Acyclic Graphs (DAGs)



Data Streams

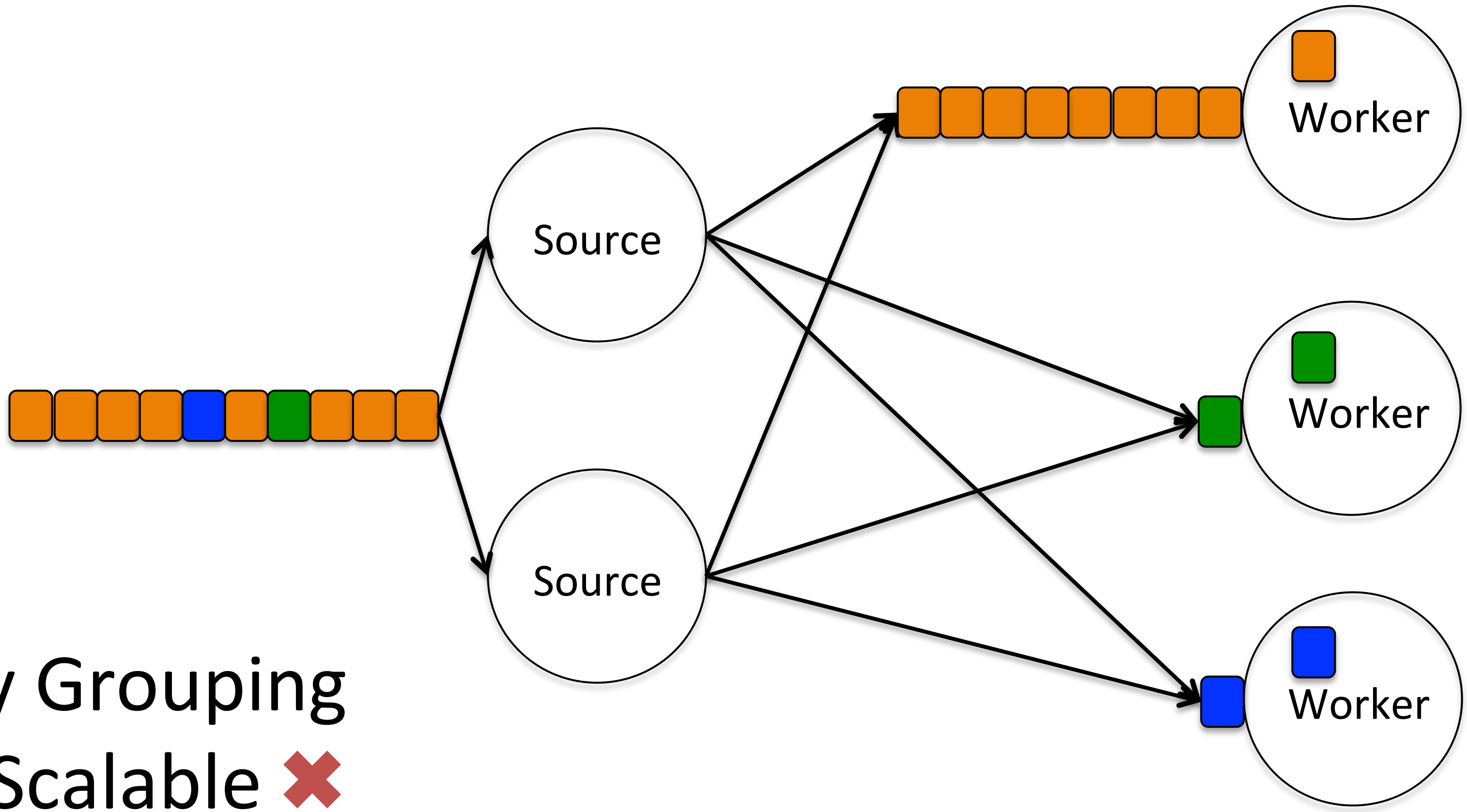


m = <timestamp, key, value>
= <14477912, 046-XXXX817, 217>

Stream Grouping

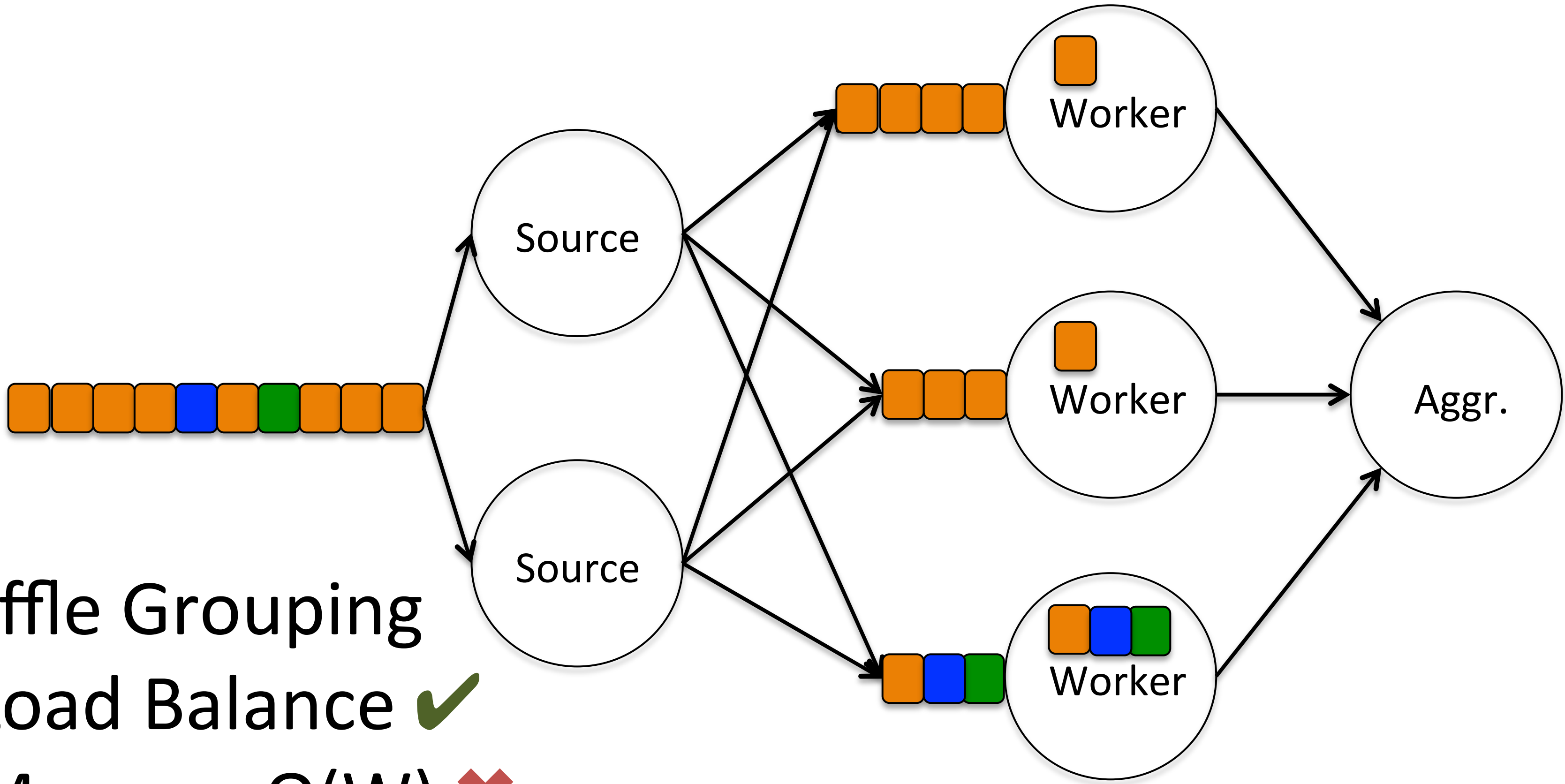
- Key or Fields Grouping (Hash Based)
  - Single worker per key
  - Stateful operators
- Shuffle Grouping (Round Robin)
  - All workers per key
  - Stateless Operators
- Partial Key Grouping
  - Two workers per key
  - MapReduce-like Operators

Key Grouping



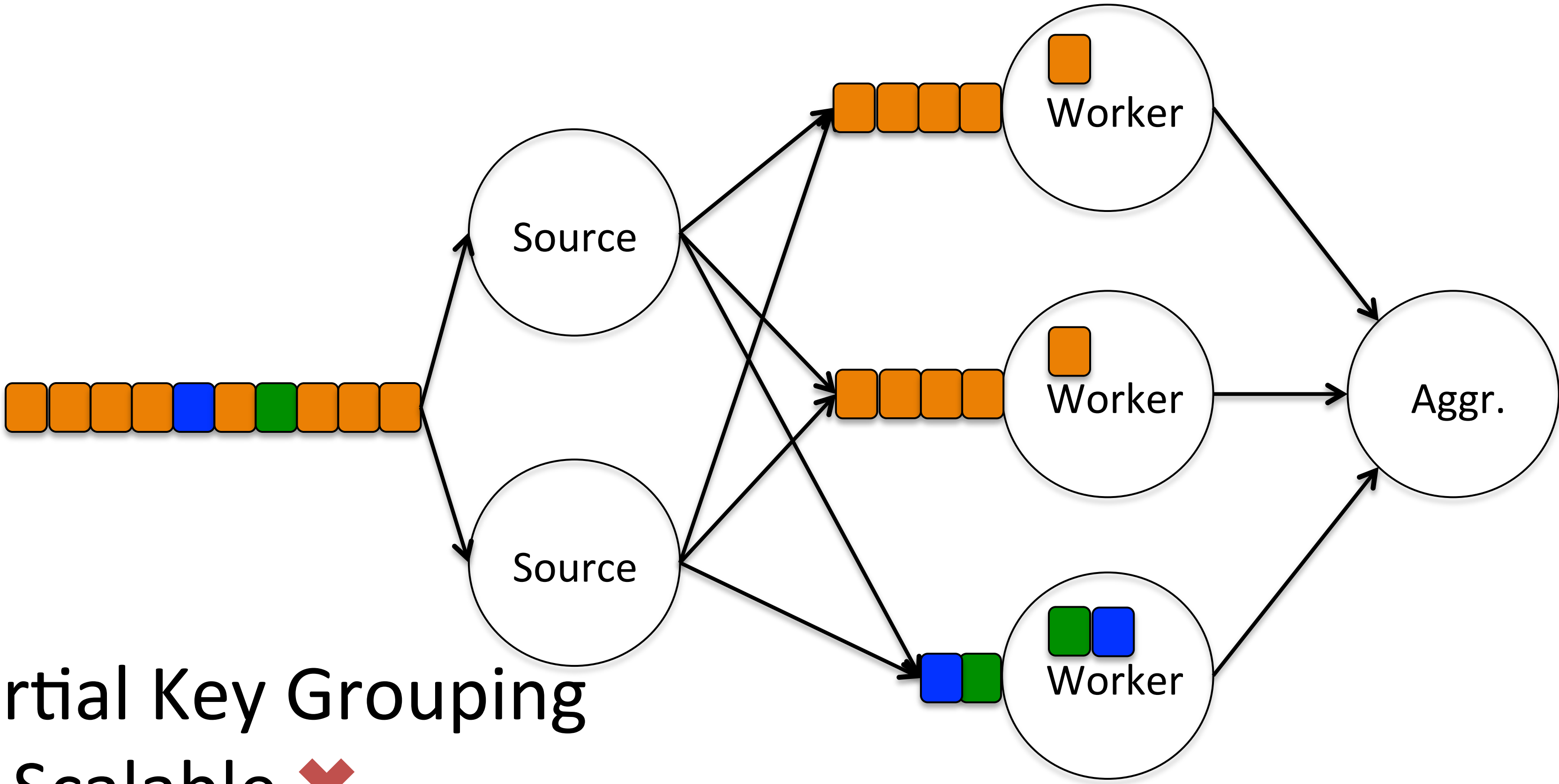
- Key Grouping
  - Scalable ✖
  - Low Memory ✔
  - Load Imbalance ✖

Shuffle Grouping



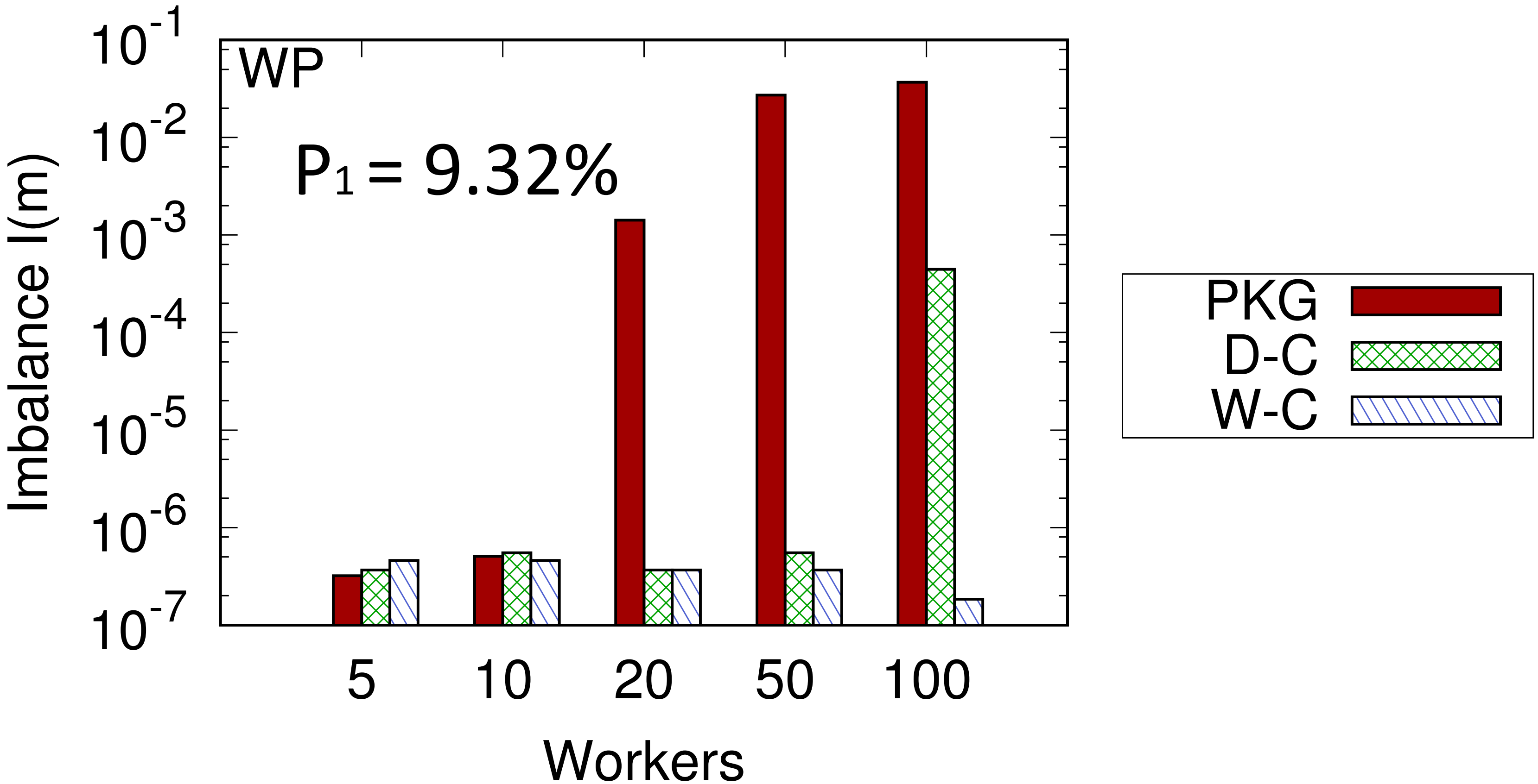
- Shuffle Grouping
  - Load Balance ✔
  - Memory O(W) ✖
  - Aggregation O(W) ✖

Partial Key Grouping



- Partial Key Grouping
  - Scalable ✖
  - Low Memory ✔
  - Load Imbalance ✖

Partial Key Grouping



Problem Formulation

- Input is a unbounded sequence of **messages** from a **key** distribution

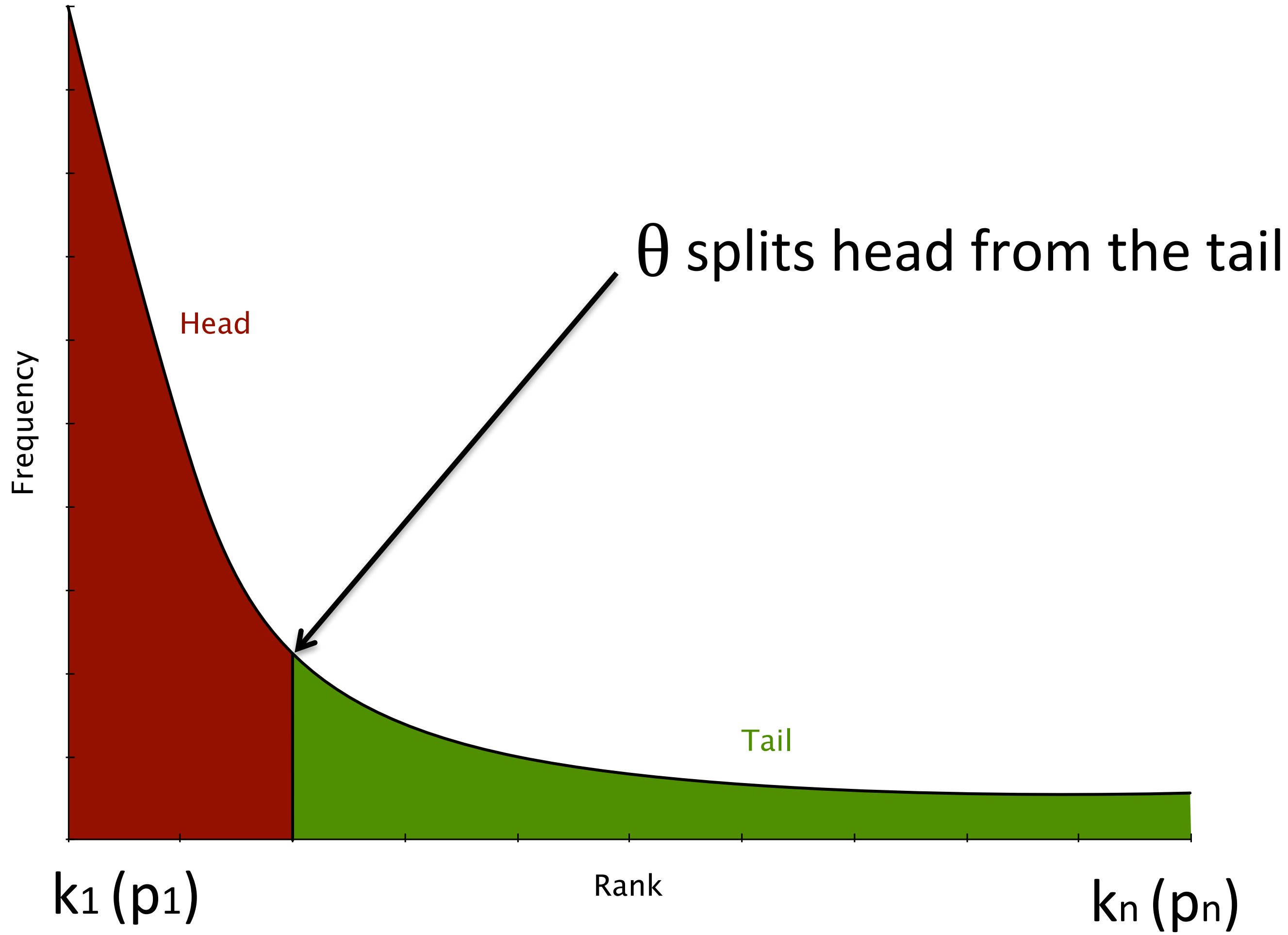
- Each message is assigned to a **worker** for processing (i.e., filter, aggregate, join)

- Load balance properties
  - Memory Load Balance
  - Network Load Balance
  - **Processing Load Balance**

- Metric: Load Imbalance

$$I(t) = \max_i(L_i(t)) - \operatorname{avg}_i(L_i(t)), \text{ for } i \in \mathcal{W}$$

High Level Idea



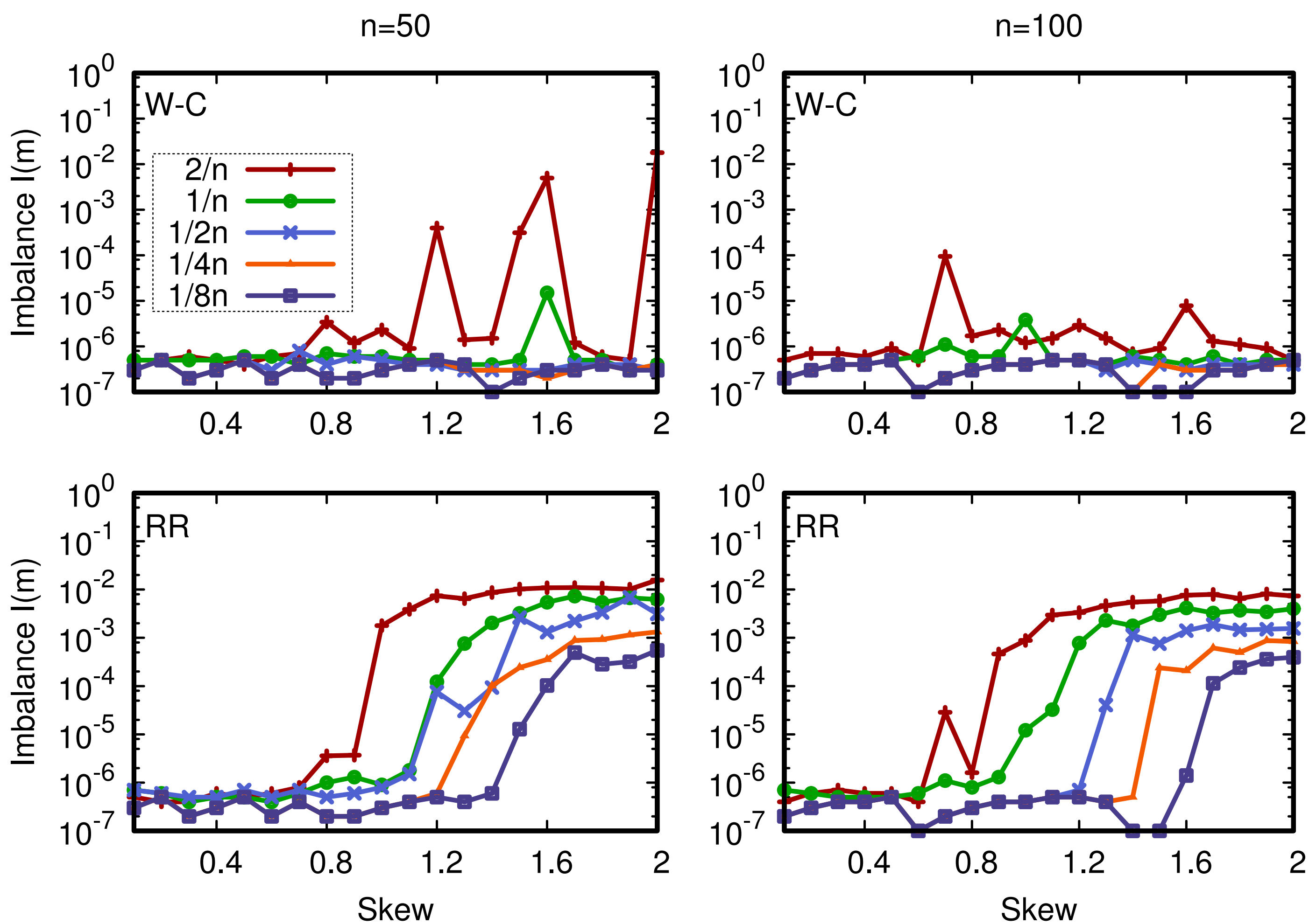
How to find optimal threshold?

- Any key that exceeds the capacity of two workers require more than two workers $\mathbf{p_i \geq 2/(n)}$

- We need to consider the collision of the keys while deciding the number of workers

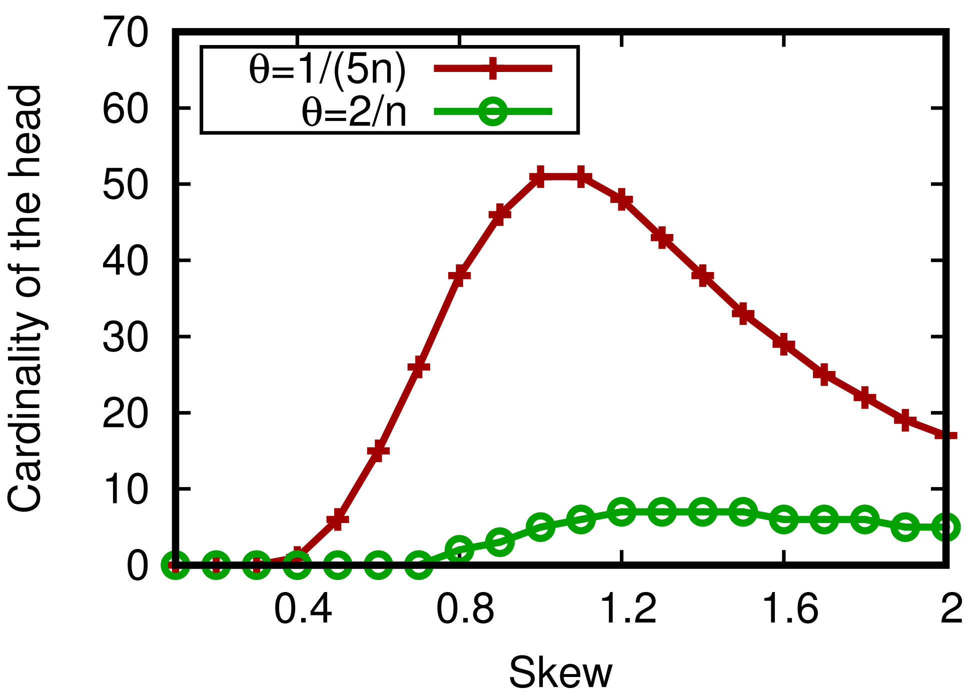
- PKG guarantees nearly perfect load balance for $\mathbf{p_1 \leq 1/(5n)}$

How to find optimal threshold?



How many keys are in the Head?

- Plots for the number of keys in Head for two different thresholds



How many workers for the Head?

- **D-Choices:** adapts to the frequencies of the keys in the Head

- **W-Choices:** allows all the workers for the keys in the head

- **Round-Robin:** employs shuffle grouping for the keys in the Head

How many workers for the Head?

- **How to assign a key to set of d workers?**

- Greedy-d: uses d different hash functions
  - generate set of d candidate workers
  - assign the key to least loaded of those workers

- In case of W-Choices, all the workers are the candidate for a key

How to find the optimal d?

- We can write our problem as an optimization problem

$$\begin{aligned} \underset{d}{\text{minimize}} \quad & f(d; \mathcal{D}, \theta) = d \times |\mathcal{H}_{\mathcal{D},\theta}| \\ \text{subject to} \quad & \underset{d}{\mathbb{E}}\left[I(m)\right] \leq \epsilon \,. \end{aligned}$$

How to find optimal d?

- We can rewrite the constraint

$$\sum_{i \le h} p_i + \left(\frac{b}{n}\right)^d \sum_{h < i \le |H|} p_i + \left(\frac{b}{n}\right)^2 \sum_{i > |H|} p_i \le \left(\frac{b}{n}\right) + \varepsilon$$

- For instance for the first key with p_1

$$p_1 + \left(\frac{b}{n}\right)^d \sum_{1 < i \le |H|} p_i + \left(\frac{b}{n}\right)^2 \sum_{i > |H|} p_i \le \left(\frac{b}{n}\right) + \varepsilon$$

How to find optimal d?

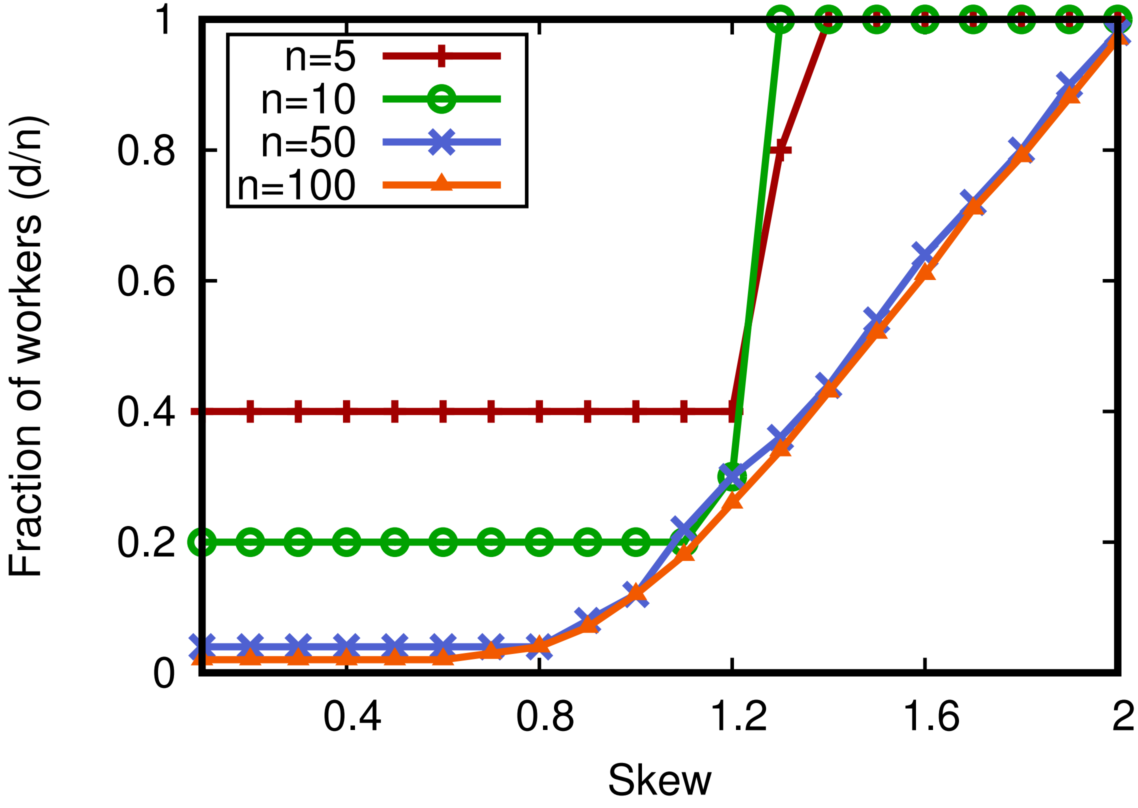
- We can rewrite the constraint

$$\sum_{i \le h} p_i + \left(\frac{b}{n}\right)^d \sum_{h < i \le |H|} p_i + \left(\frac{b}{n}\right)^2 \sum_{i > |H|} p_i \le \left(\frac{b}{n}\right) + \varepsilon$$

where

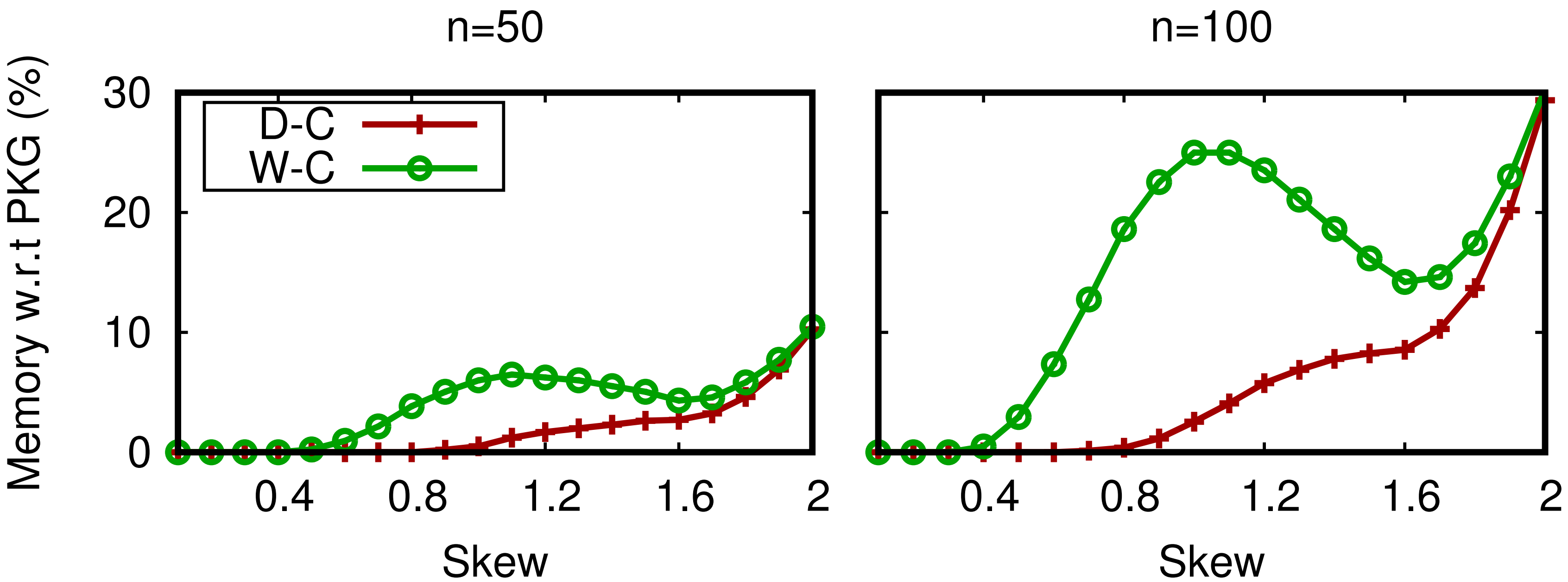
$$b = n - n\left(\frac{n-1}{n}\right)^{h \times d}$$

What are the values of d?



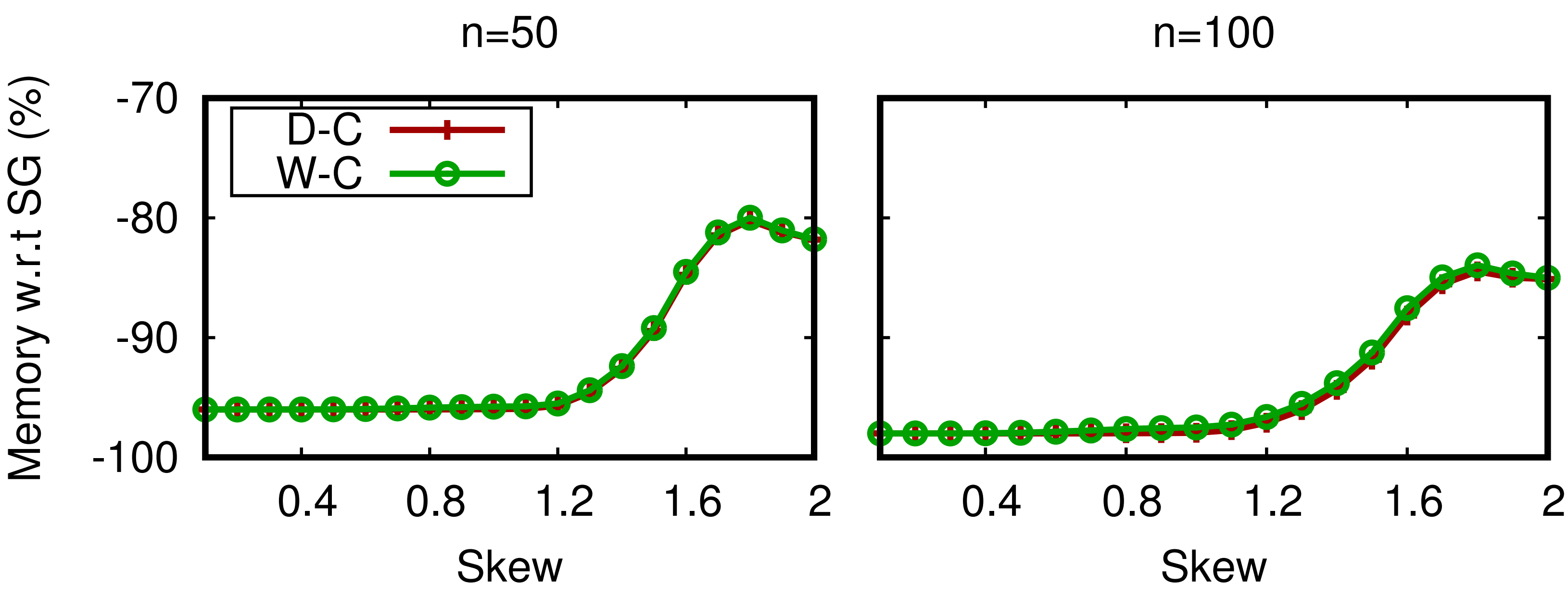
Memory Overhead

- Compared to PKG



Memory Overhead

- Compared to SG



Experimental Evaluation

- Datasets

Dataset	Symbol	Messages	Keys	$p_1(\%)$
Wikipedia	WP	22M	2.9M	9.32
Twitter	TW	1.2G	31M	2.67
Cashtags	CT	690k	2.9k	3.29
Zipf	ZF	10^7	10^4,10^5,10^6	$\propto \frac{1}{\sum x^{-z}}$

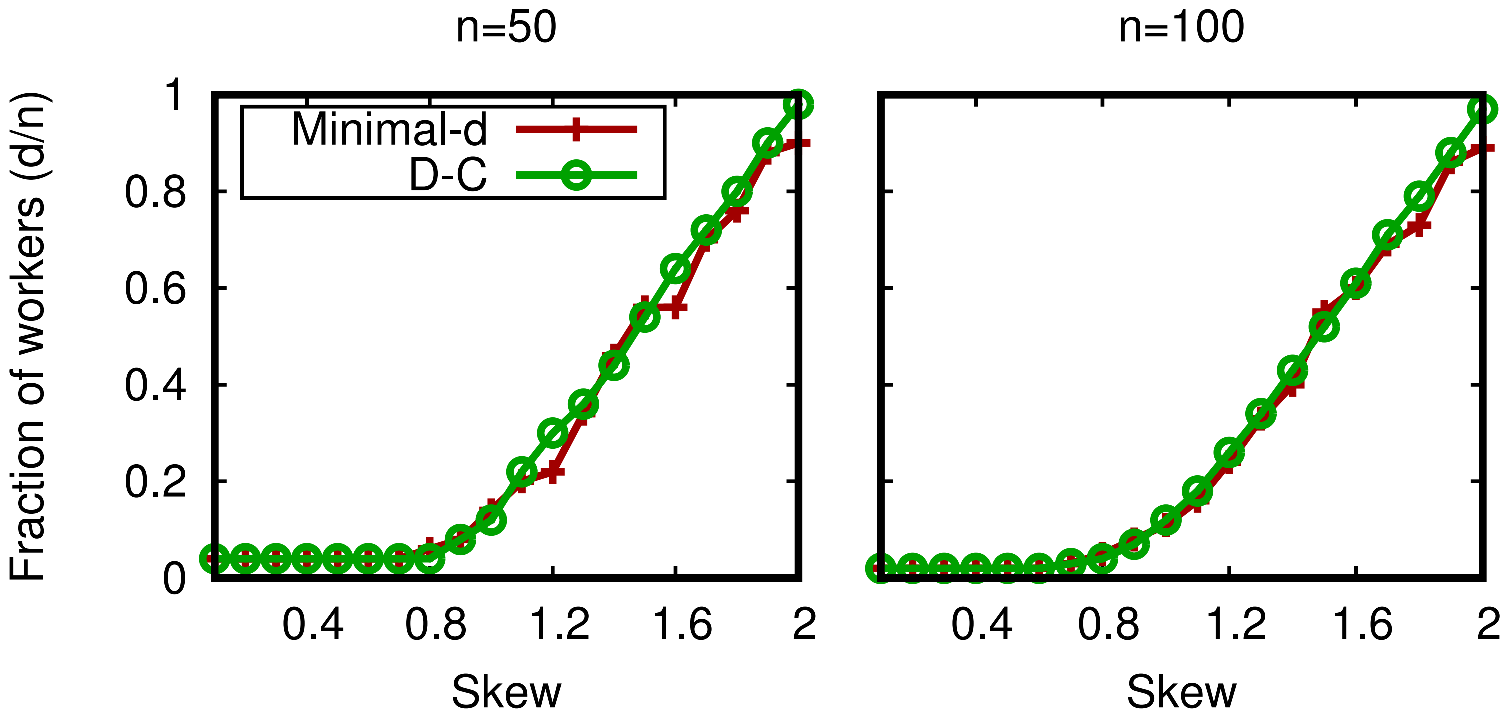
Experimental Evaluation

- Algorithms

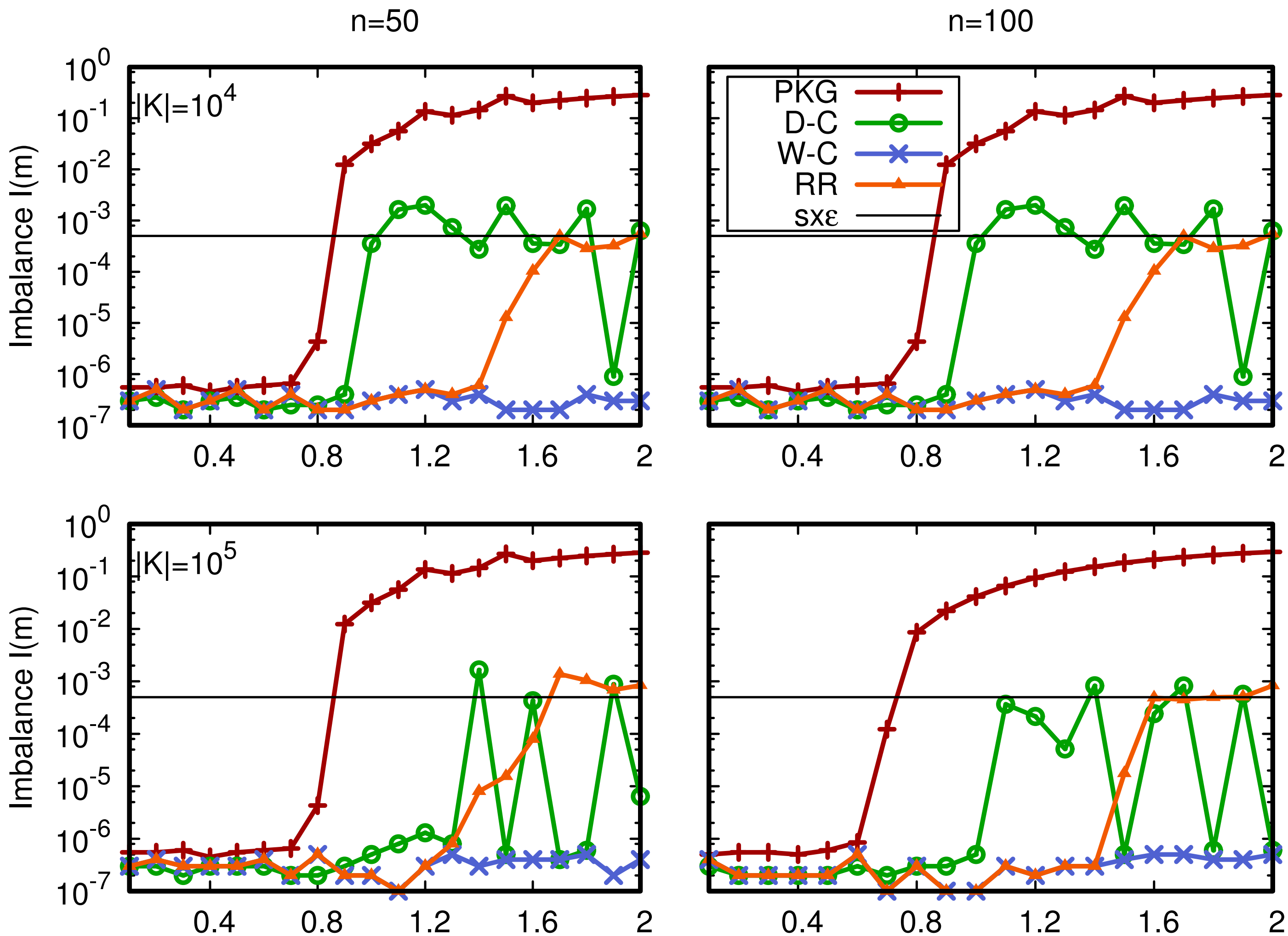
Symbol	Algorithm	Head vs. Tail
D-C	D-Choices	Specialized on head
W-C	W-Choices	
RR	Round-Robin	
PKG	Partial Key Grouping	Treats all keys equally
SG	Shuffle Grouping	

How good are estimated d?

- Comparison of estimated d versus the minimal experimental value of d

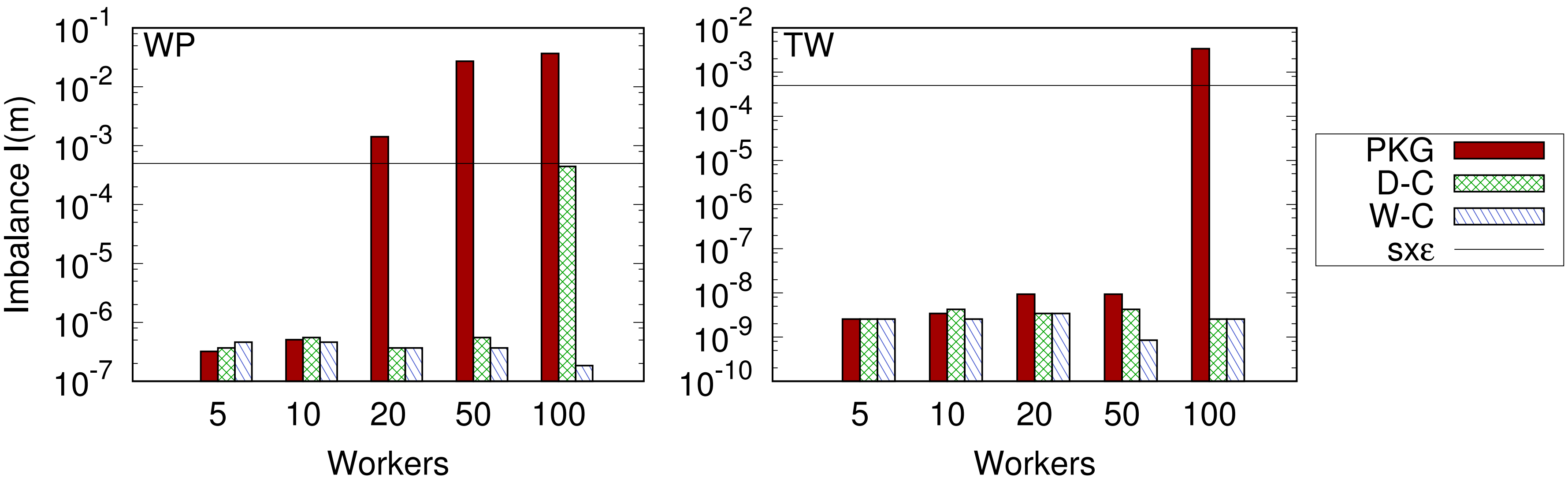


Load Imbalance for Zipf



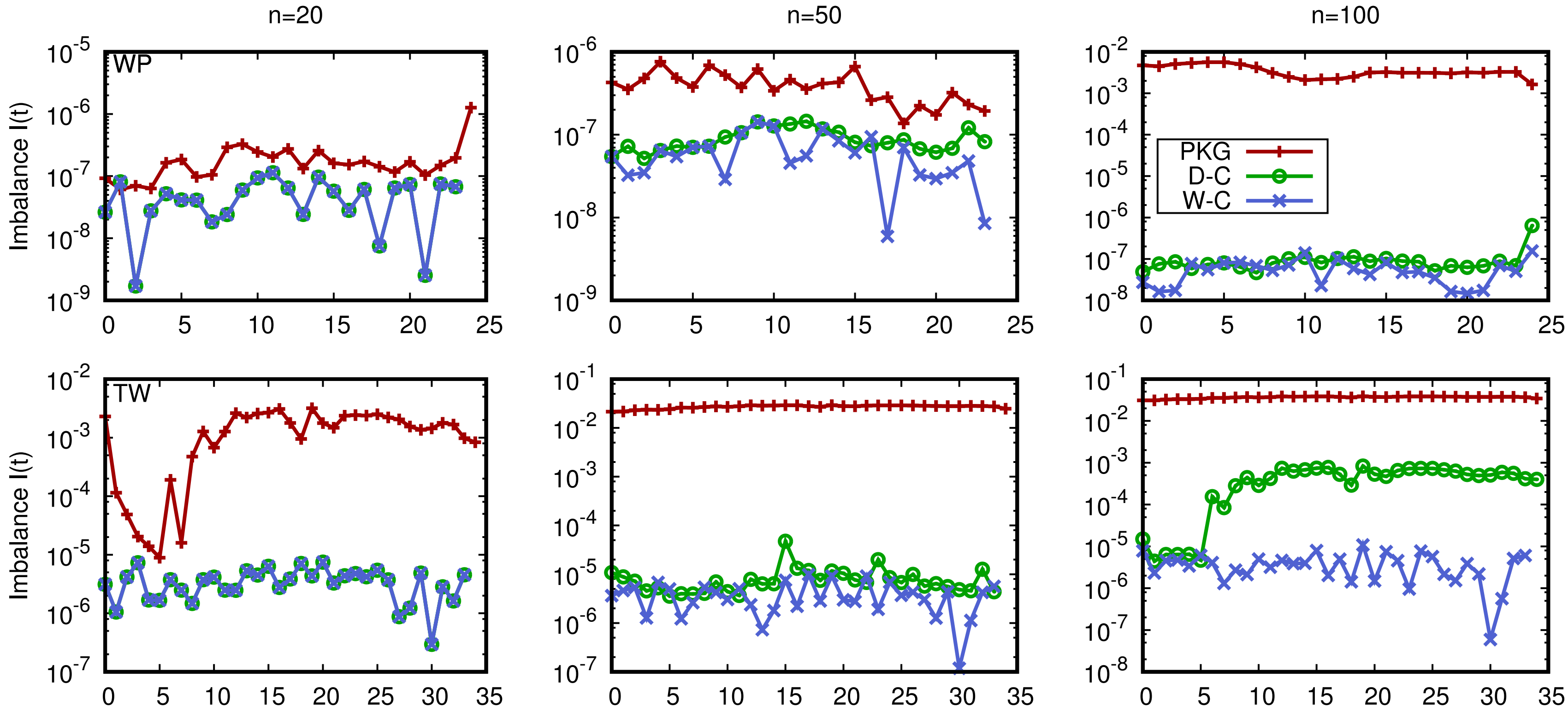
Load balance for real workloads

- Comparison of D-C, WC with PKG in terms of load balance



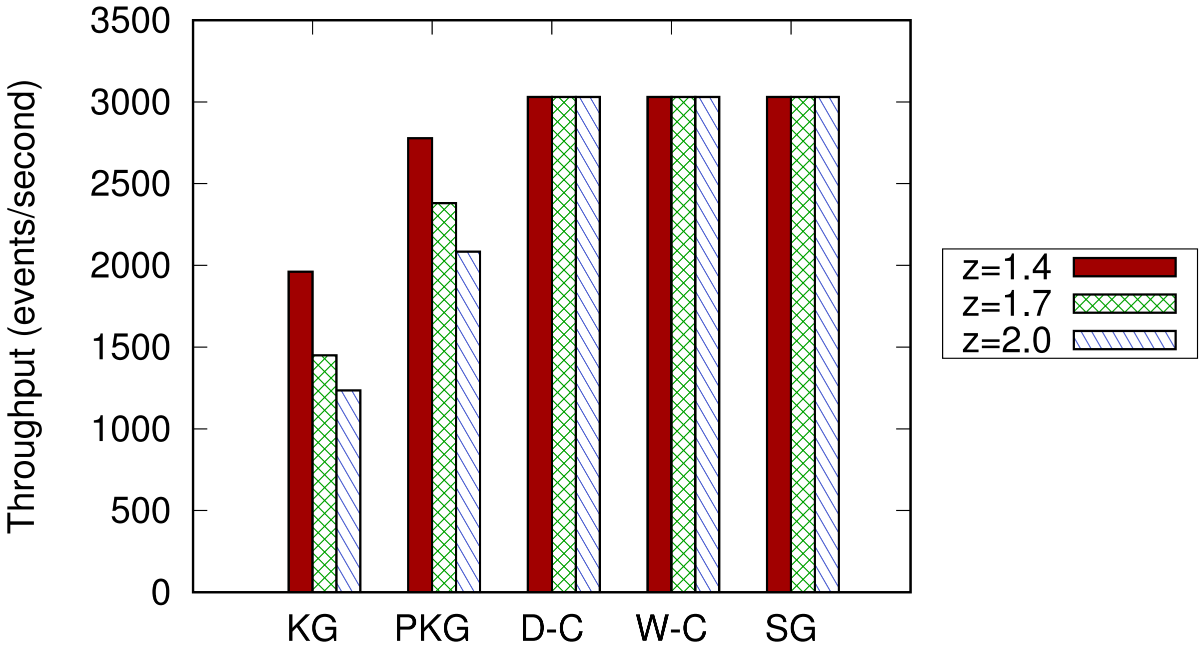
Load Imbalance over time

- Load imbalance over time for the real-world datasets



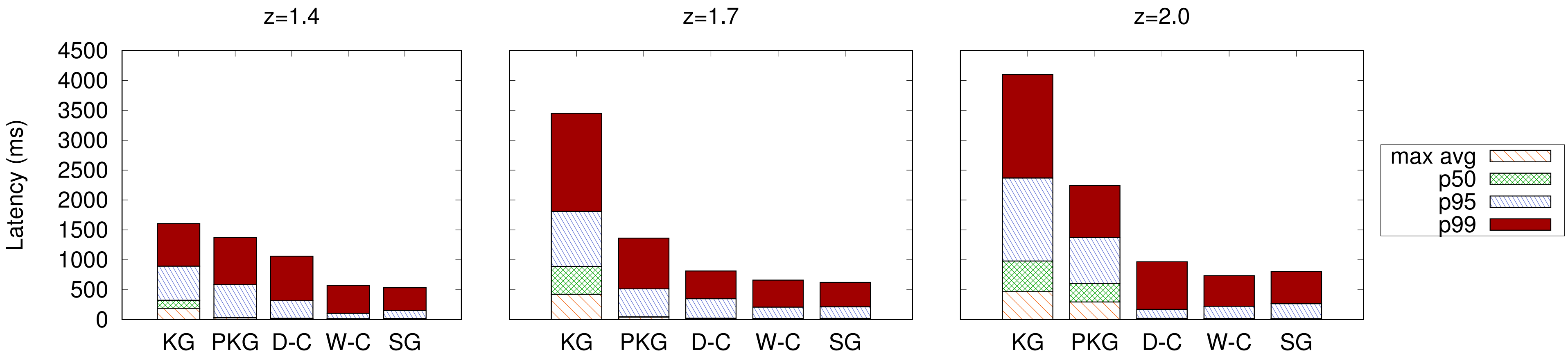
Throughput on real DSPE

- Throughput on a cluster deployment on Apache Storm for KG, PKG, SG, D-C, and W-C on the ZF dataset



Latency on a real DSPE

- Latency (on a cluster deployment on Apache Storm for KG, PKG, SG, D-C, and W-C



Conclusion

- We propose two algorithms to achieve load balance at scale for DSPEs
- Use heavy hitters to separate the head of the distribution and process on larger set of workers
- Improvement translate into 150% gain in throughput and 60% gain in latency over PKG

When Two Choices Are not Enough: Balancing at Scale in Distributed Stream Processing

Anis Nasir

Accepted at ICDE 2016, available at arXiv