SJTU SAIL (Software Architecture and Infrastructure Lab)

SHANGHAI JIAO TONG UNIVERSITY

Insight of Medea and Neptune

Scheduling of Long Running Applications &
Scheduling Suspendible Tasks for Unified Stream/Batch Applications

Lin Hsu Yalun 2020/09/21

SHANGHAI JIAO TONG UNIVERSITY

Medea

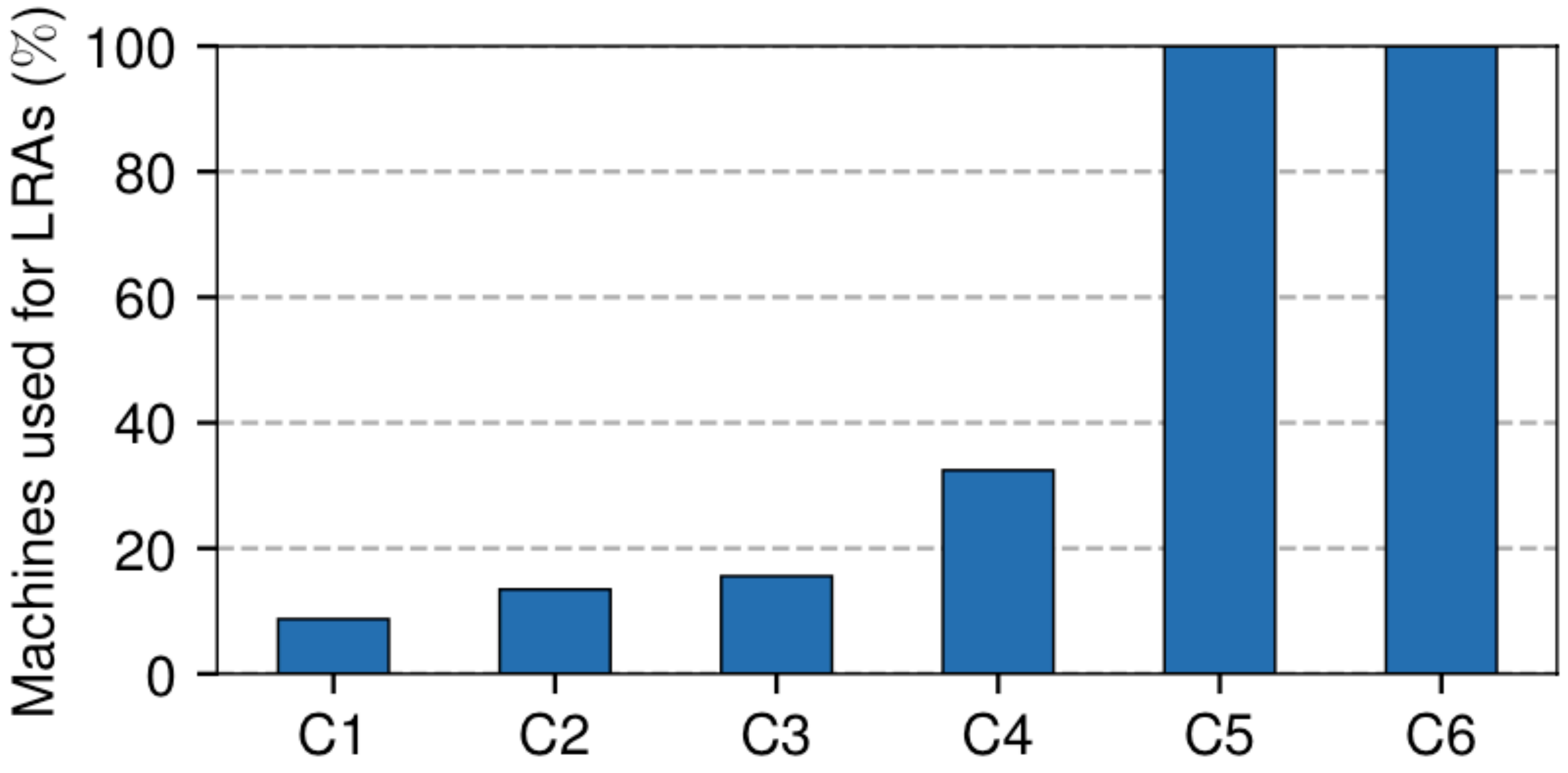
Scheduling of Long Running Applications in Shared Production Clusters

Long-Running Applications (LRAs)

- **Interactive data-intensive** applications
  - Spark, Hive LLAP
- **Streaming** systems
  - Flink, Storm, SEEP
- **Latency-sensitive** applications
  - HBase, Memcached
- **ML** frameworks
  - TensorFlow, Spark ML-lib

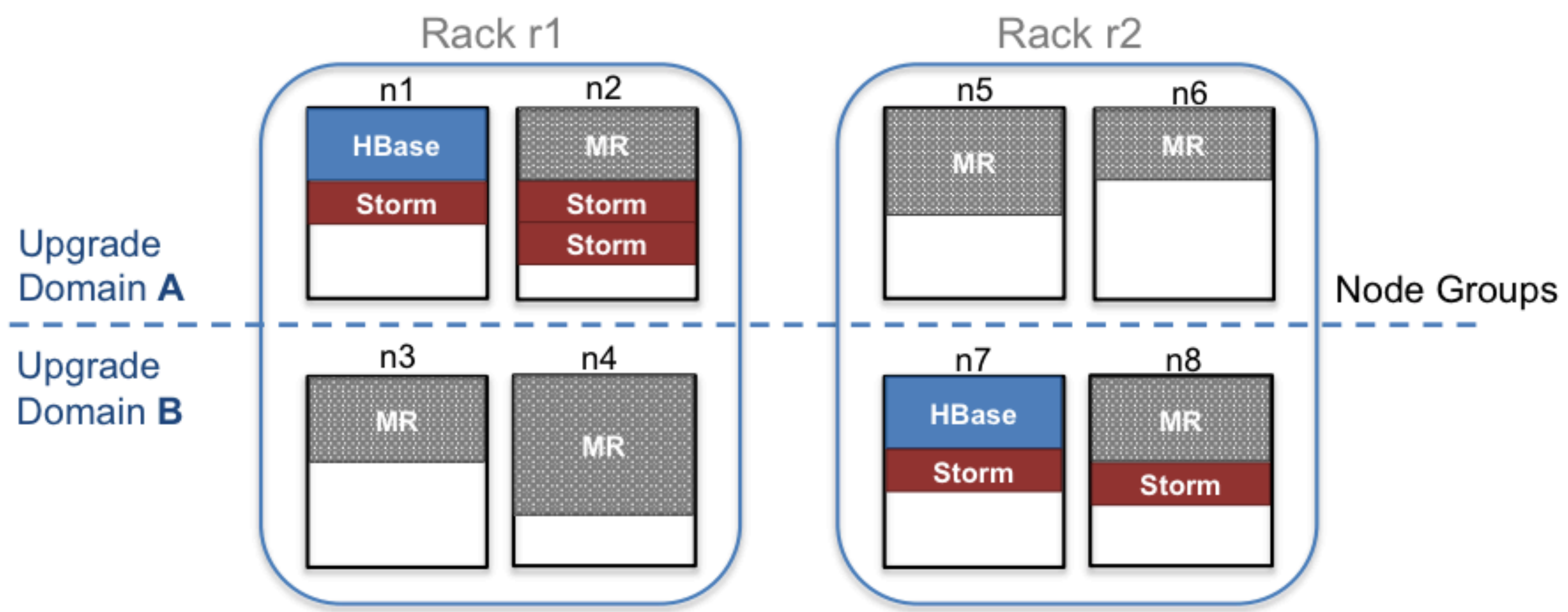
LRAs = applications
with long-running containers
(running from hours to months)

LRAs in Microsoft's analytics clusters



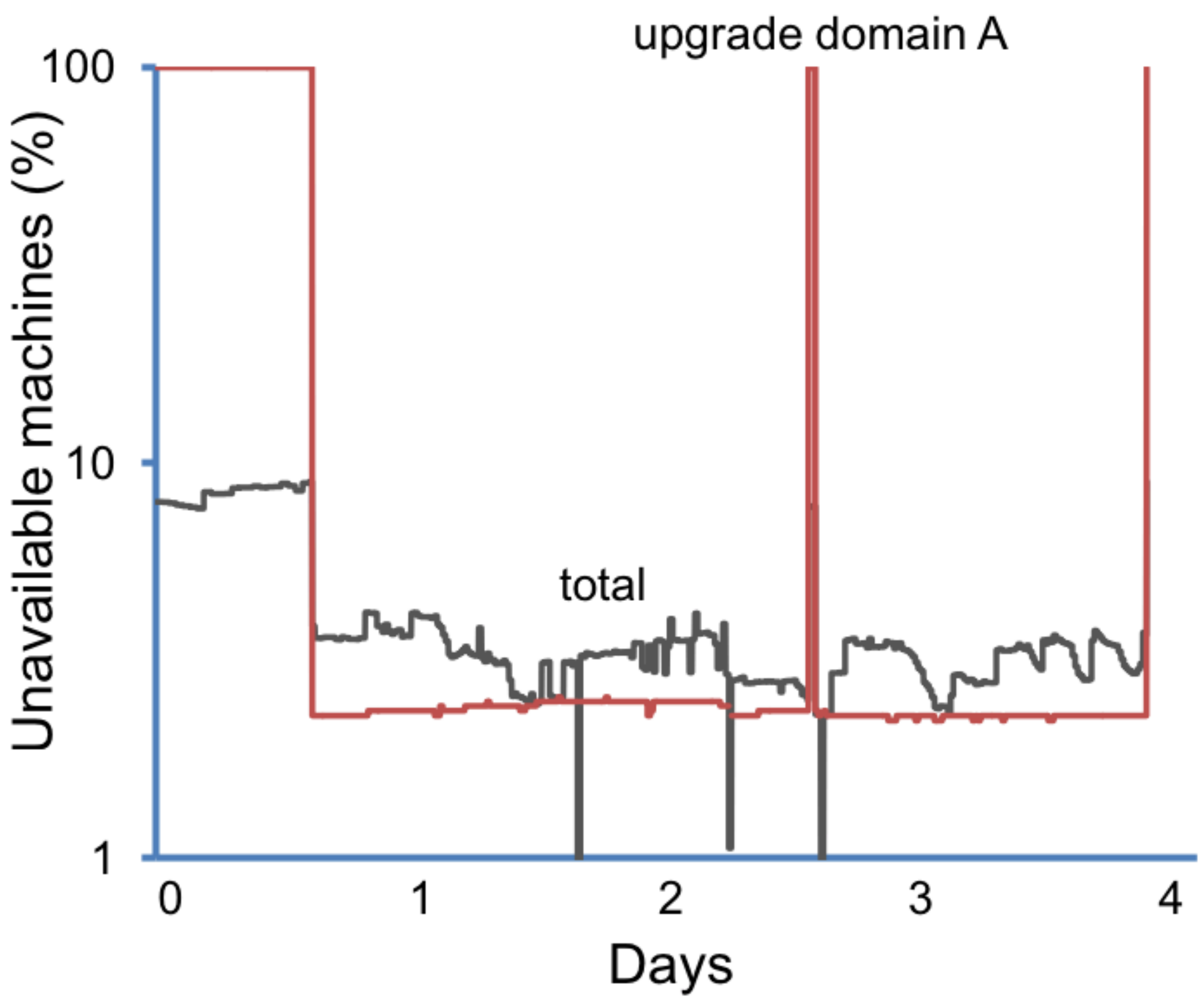
LRA placement is important

LRA scheduling problem



- **Performance**: “Place Storm containers in the same rack as HBase”
- **Cluster objectives**: “Minimize resource fragmentation”
- **Resilience**: “Place HBase containers across upgrade domains"

Machine unavailability in a Microsoft cluster



With random placement, an LRA might lose all containers at once

Challenges

- How to **relate containers** to node groups?
- How to **express** different types of constraints related to LRA containers?
- How to achieve **high quality placement** without affecting task-based jobs?

Medea

- How to **relate containers** to node groups?
- Support container tags and logical node groups
- How to **express** different types of constraints related to LRA containers?
- Introduce expressive cardinality constraints
- How to achieve **high quality placement** without affecting task-based jobs?
- Follow a two-scheduler design

Medea Design

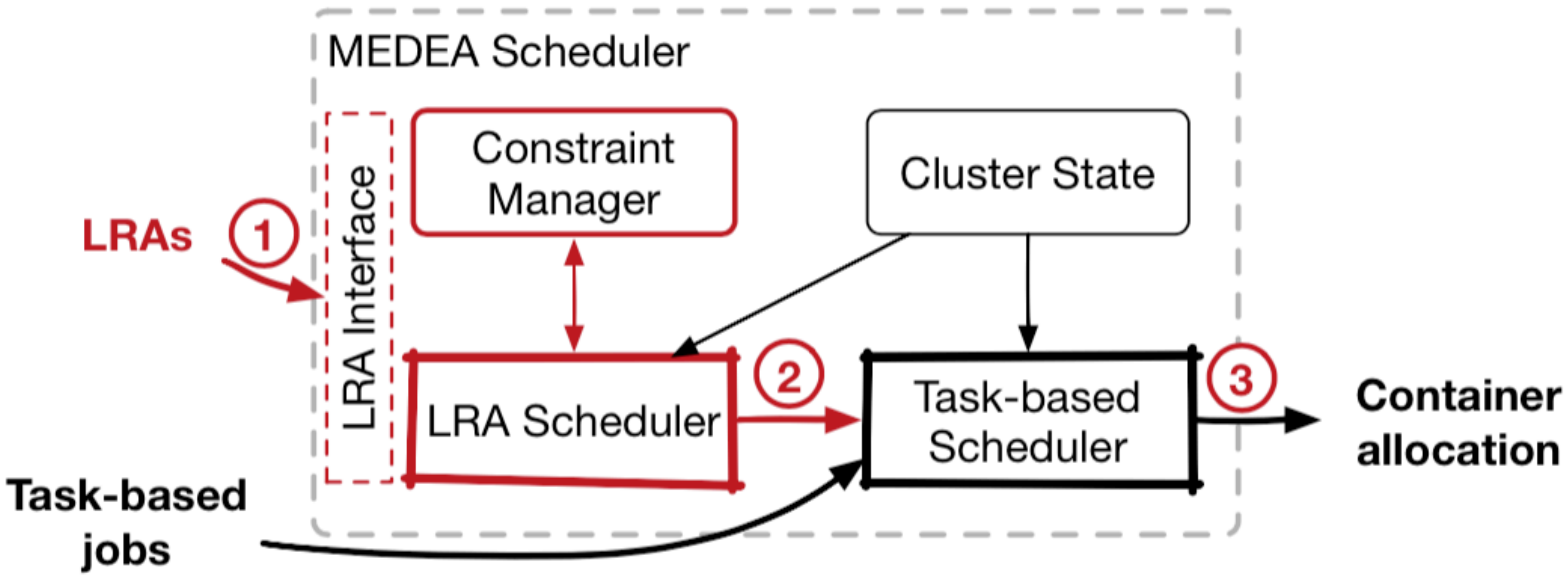
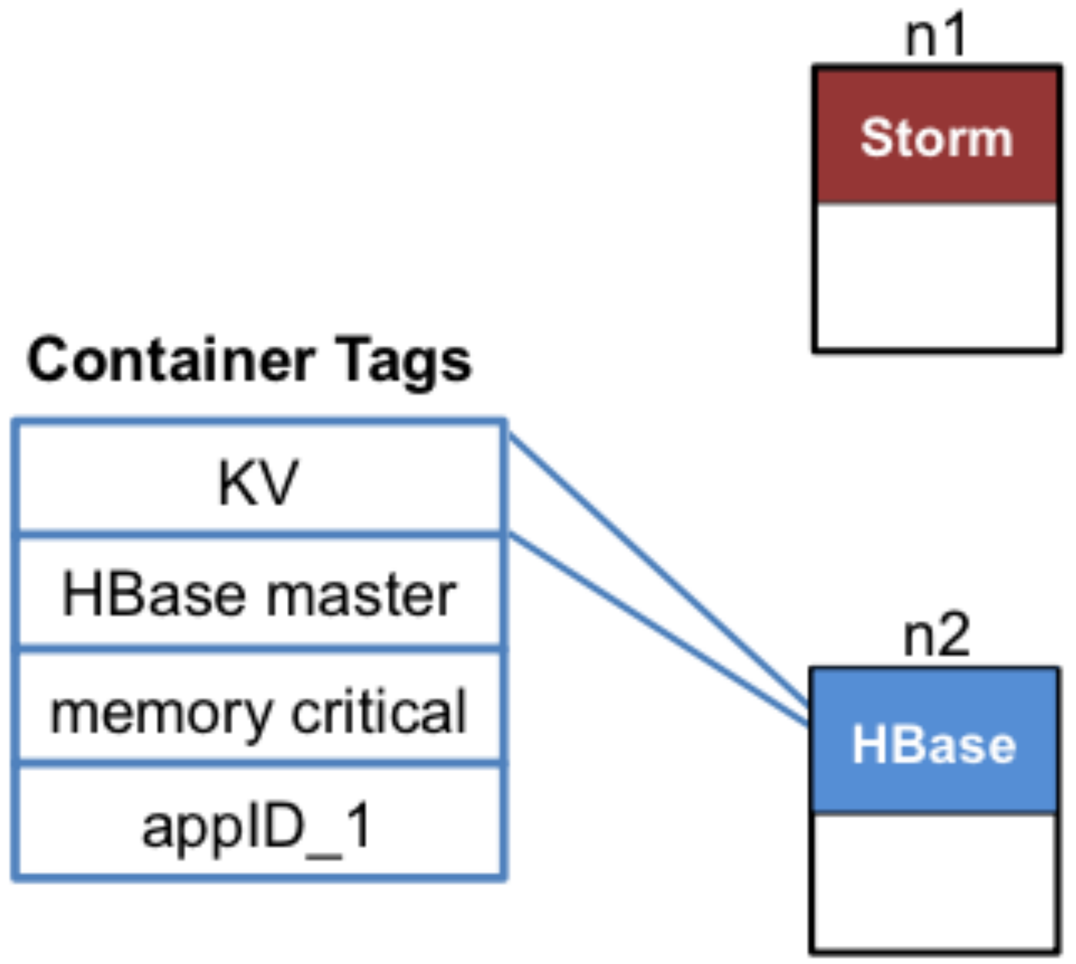


Figure 4: MEDEA scheduler design

Container tagging

- **Idea**: use container tags to refer to group of containers

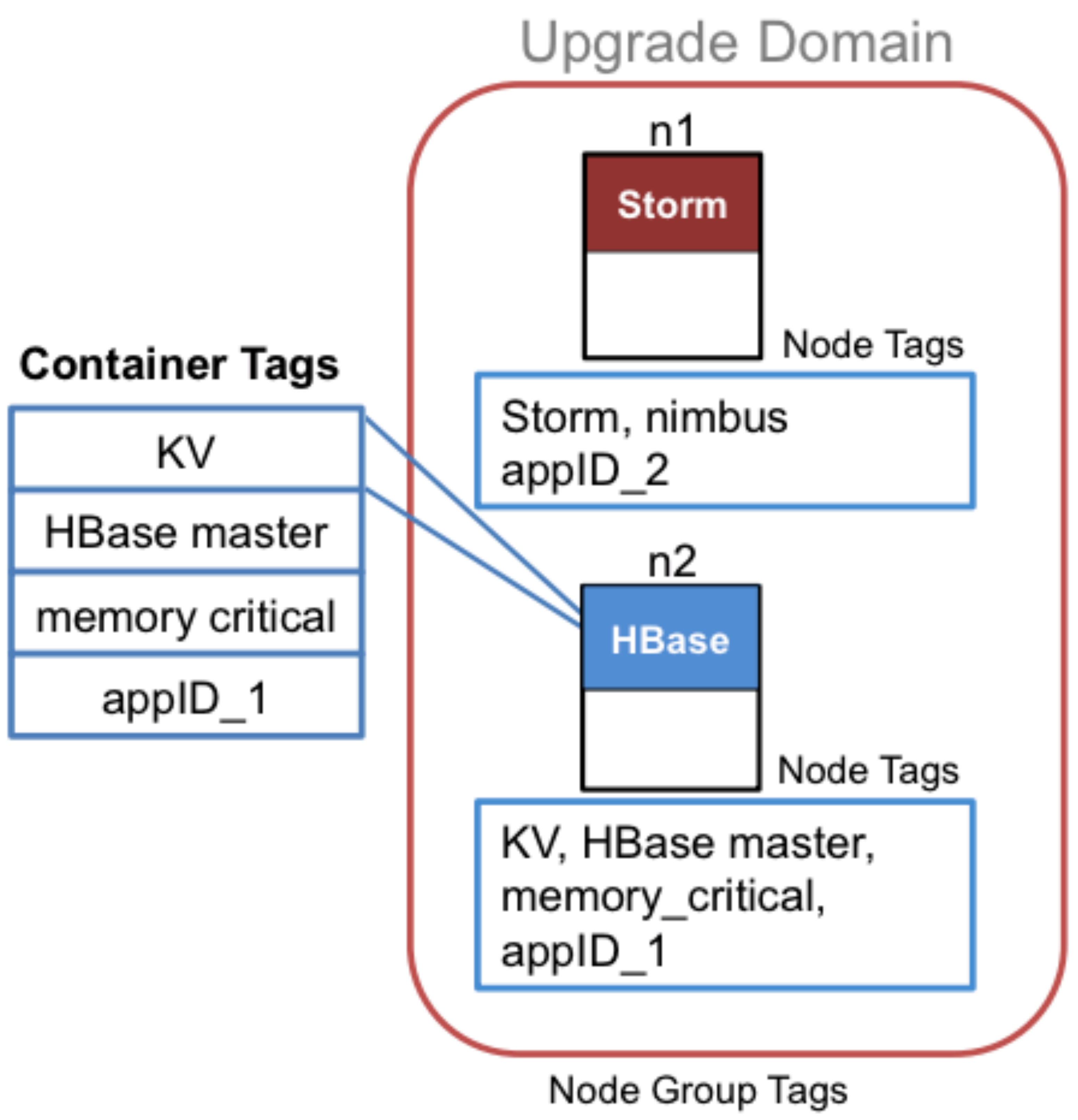
  - Describe
    - application type
    - application role
    - resource specification
    - global application ID



- Can refer to any current or future LRA container

Hierarchical grouping of nodes

- **Idea**: logical node groups to refer to dynamic node sets
  - E.g. node, rack. Upgrade domain
  - Associate nodes with all the container tags that live there
  - Hide infrastructure “spread across upgrade domains”



Defining constraints

- Generic constraints to capture a variety of cases

Source Tag	Node Group	Target Tag	Min cardinality / Max cardinality

  - Min cardinality ≤ occurrences (Target Tag) ≤ Max cardinality

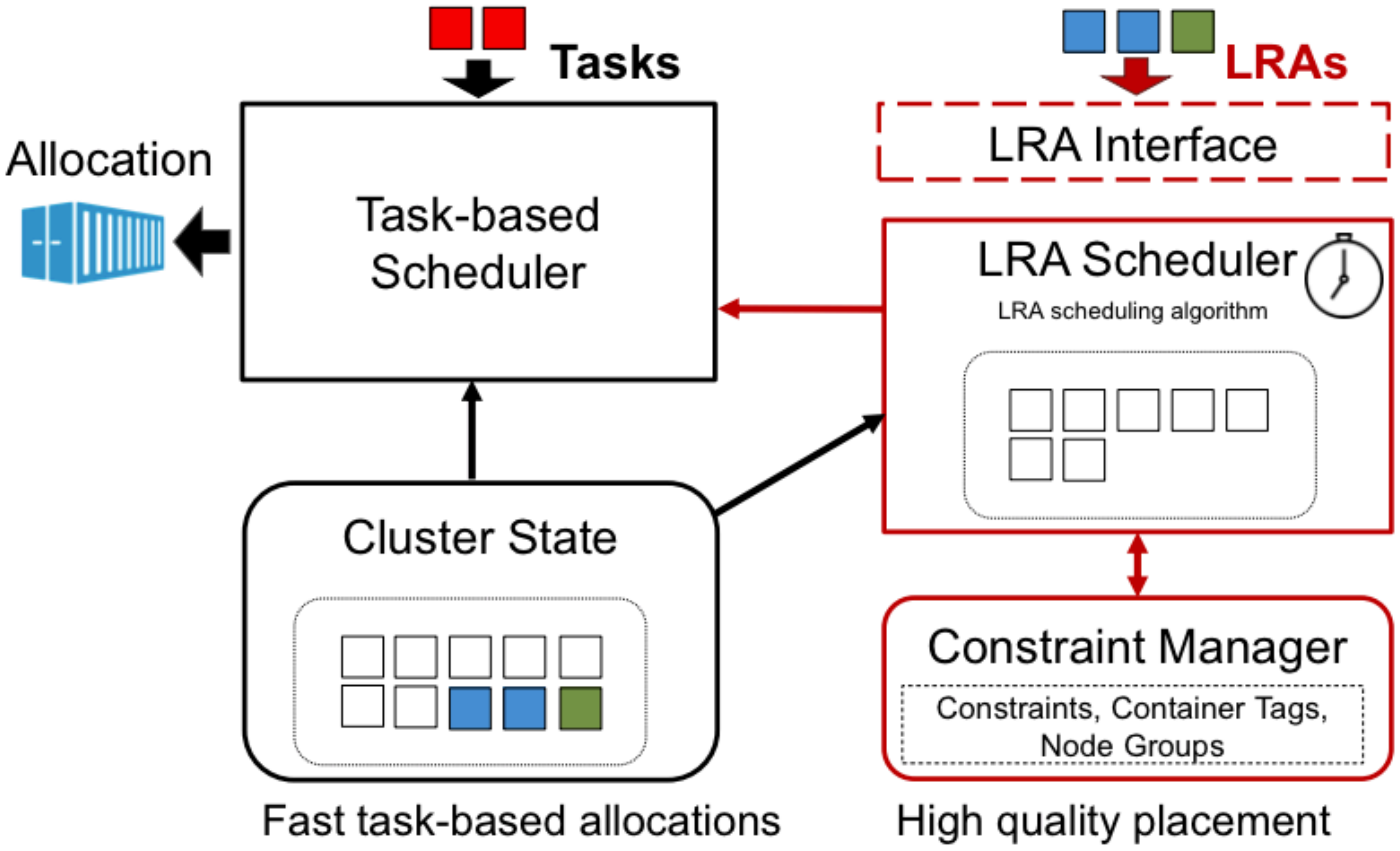
- Affinity “Place Storm containers in the same rack as HBase”

storm	rack	hbase	1 / ∞

- Cardinality “Place up to 5 Storm containers in the same node”

storm	node	storm	0 / 5

Two-scheduler design



ILP-based scheduling algorithm

$$\text{maximize } \frac{w_1}{k}\sum_{i=1}^{k} S_i + \frac{w_2}{m}\sum_{l=1}^{m} v_c^l + \frac{w_3}{N}\sum_{n=1}^{N} z_n \quad (1)$$

subject to:

$$\forall i, j : \sum_{n=1}^{N} X_{ijn} \le 1 \quad (2)$$

$$\forall n : \sum_{i=1}^{k}\sum_{j=1}^{T_i} r_{ij} \cdot X_{ijn} \le R_n^f \quad (3)$$

$$\forall i : \sum_{n=1}^{N}\sum_{j=1}^{T_i} X_{ijn} - T_i S_i = 0 \quad (4)$$

$$\forall n : \sum_{i=1}^{k}\sum_{j=1}^{T_i} r_{ij} \cdot X_{ijn} - B_n(1 - z_n) \le R_n^f - r_{min} \quad (5)$$

For each constraint $C_l = \{\mathtt{s\_tag}, \{\mathtt{c\_tag}, c^l_{min}, c^l_{max}\}, \mathtt{G}\}$, $\forall$ container $t_{i_s j_s} \in \mathtt{s\_tag}$, $\forall$ node set $\mathcal{S} \in \mathtt{G}$:

$$\sum_{n \in \mathcal{S}} \left(\sum_{\substack{i,j:\mathtt{tag} \in t_{ij} \\ t_{ij} \ne t_{i_s j_s}}} X_{ijn} + D_n(1 - X_{i_s j_s n}) \right) - c^l_{min} + c^{l,v}_{min} \ge 0 \quad (6)$$

$$\sum_{n \in \mathcal{S}} \left(\sum_{\substack{i,j:\mathtt{tag} \in t_{ij} \\ t_{ij} \ne t_{i_s j_s}}} X_{ijn} - D_n(1 - X_{i_s j_s n}) \right) - c^l_{max} - c^{l,v}_{max} \ge 0 \quad (7)$$

$$v_c^l = \frac{c^{l,v}_{min}}{c^l_{min}} + \frac{c^{l,v}_{max}}{c^l_{max}} \quad (8)$$

Symbol	Description
k	Number of LRAs to be placed
N	Number of cluster nodes
T_i	Number of containers of LRA i
R_n^f, R_n^u	Free, used resources of node n[6]
m	Total number of constraints
w_i	Weights of components in objective function
B_n, D_n	Sufficiently large integers, used in inequalities
S_i	1 if all containers of LRA i are placed; 0 otherwise
X_{ijn}	1 if container j of LRA i placed at node n; 0 otherwise
r_{ij}	Resource demand of container j of LRA i
r_{min}	Minimum resource demand
$c^{l,v}_{min}, c^{l,v}_{max}$	Violation of cardinalities c_{min}, c_{max} for constraint C_l
v_c^l	Violation for constraint C_l
z_n	1 if free resources $\ge r_{min}$ after placement; 0 otherwise

Evaluation

A 400 node pre-production cluster grouped into 10 racks, supplemented by simulation.

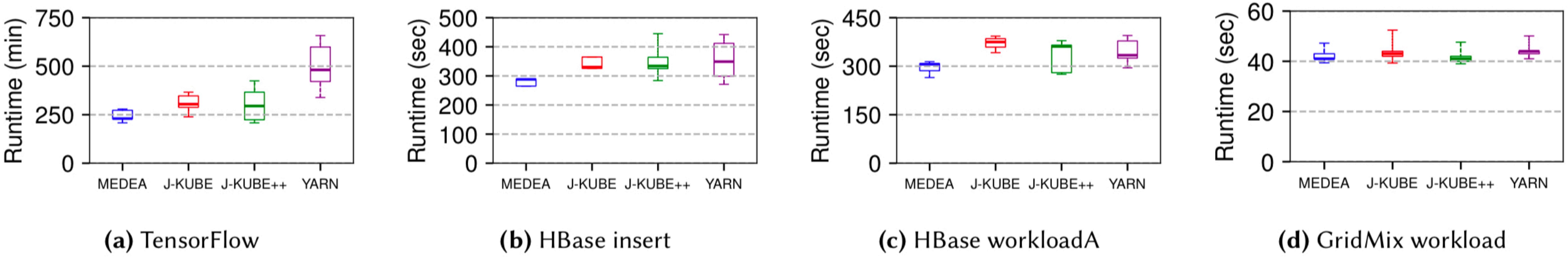


Figure 7: Application performance (lower is better)

Evaluation

A 400 node pre-production cluster grouped into 10 racks, supplemented by simulation.

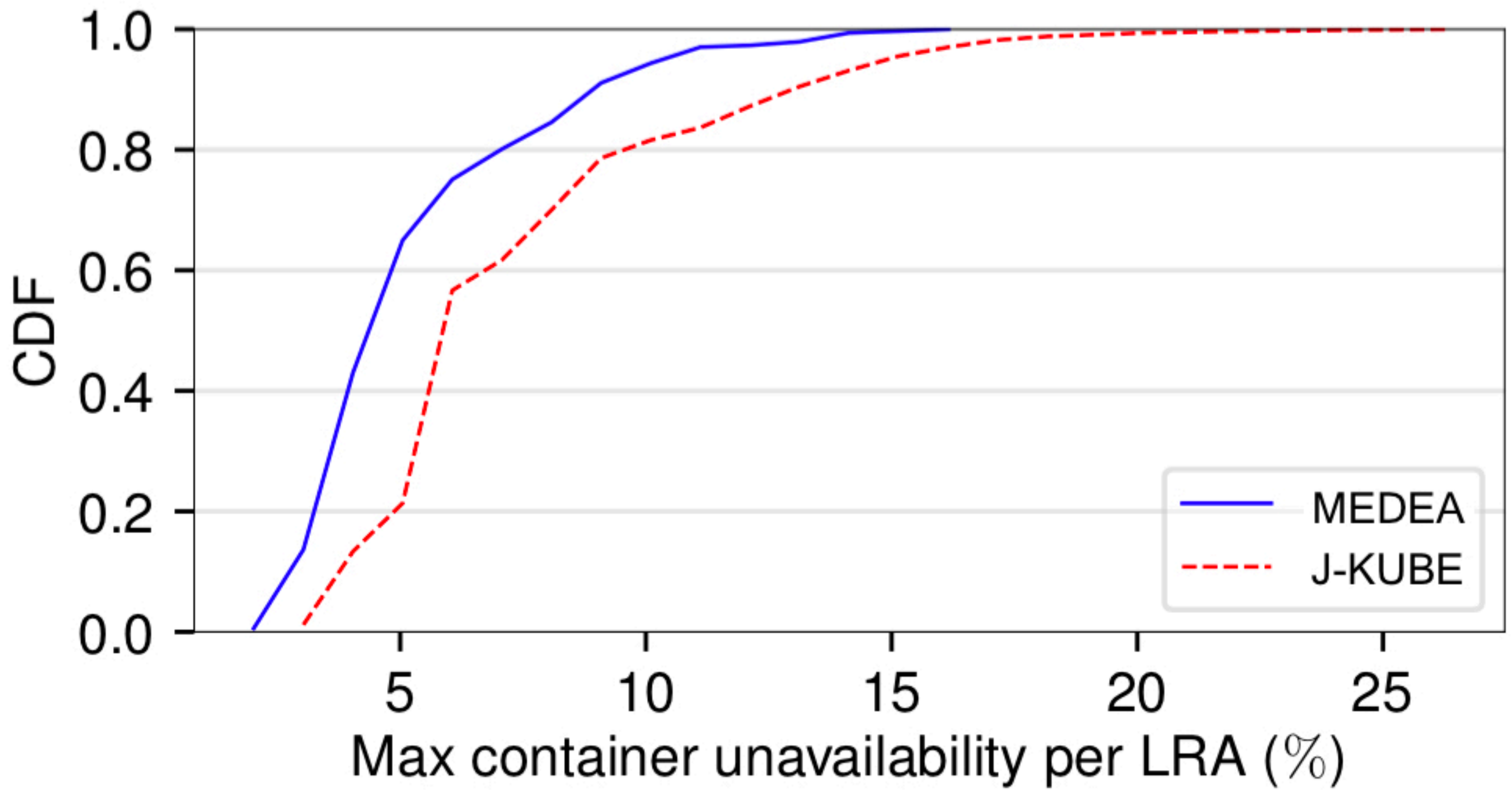
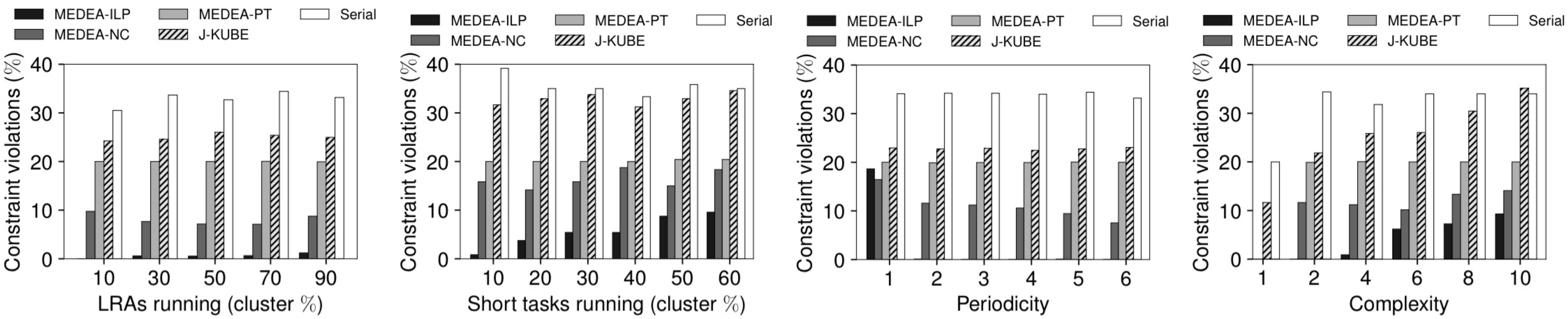


Figure 8: Application resilience over 15 days

Evaluation

A 400 node pre-production cluster grouped into 10 racks, supplemented by simulation.



(a) Varying cluster LRA utilization (b) Varying task-based utilization (c) Varying scheduling intervals (d) Varying constraint complexity

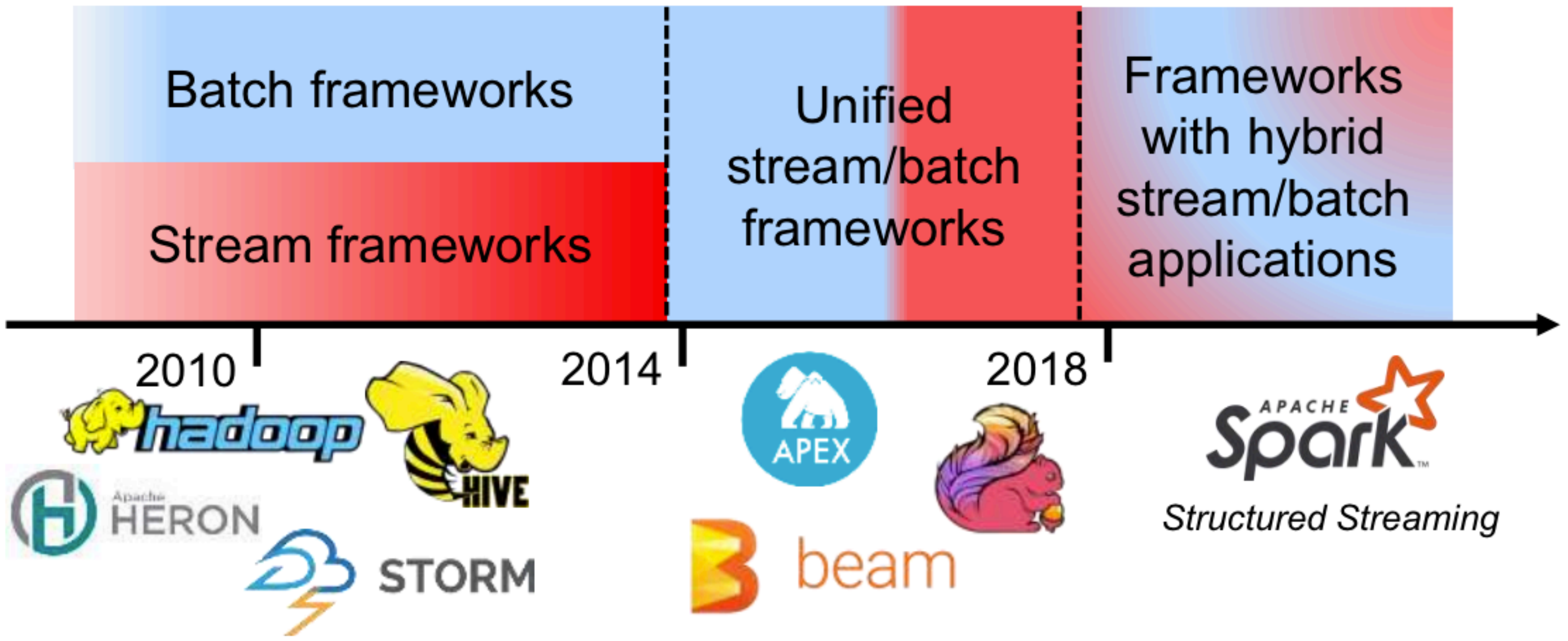
Figure 9: Constraint violations

Neptune

Scheduling Suspendable Tasks
for Unified Stream/Batch Applications

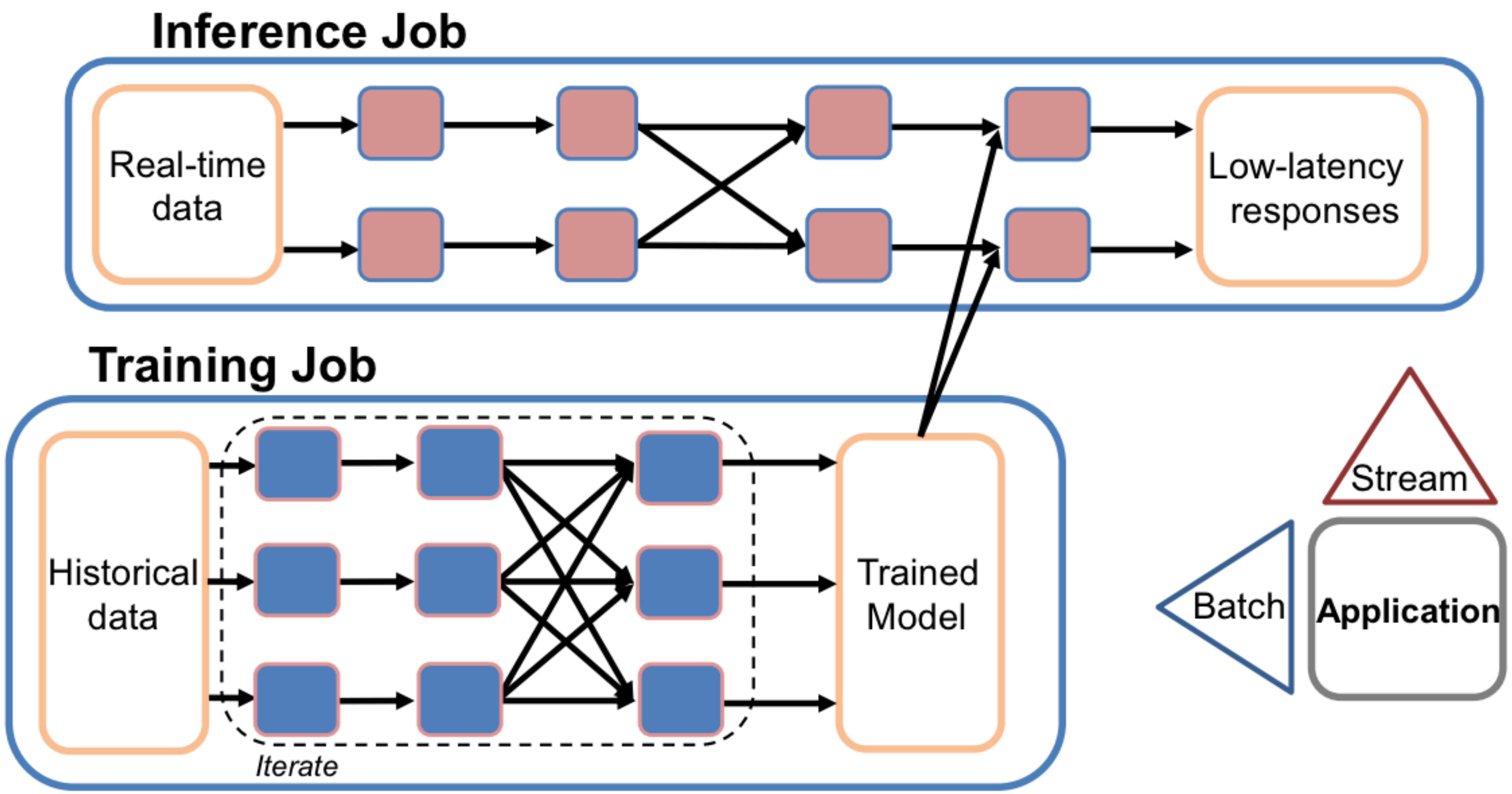
SHANGHAI JIAO TONG UNIVERSITY

Evolution of analytics frameworks



UNIFICATION

Unified application example

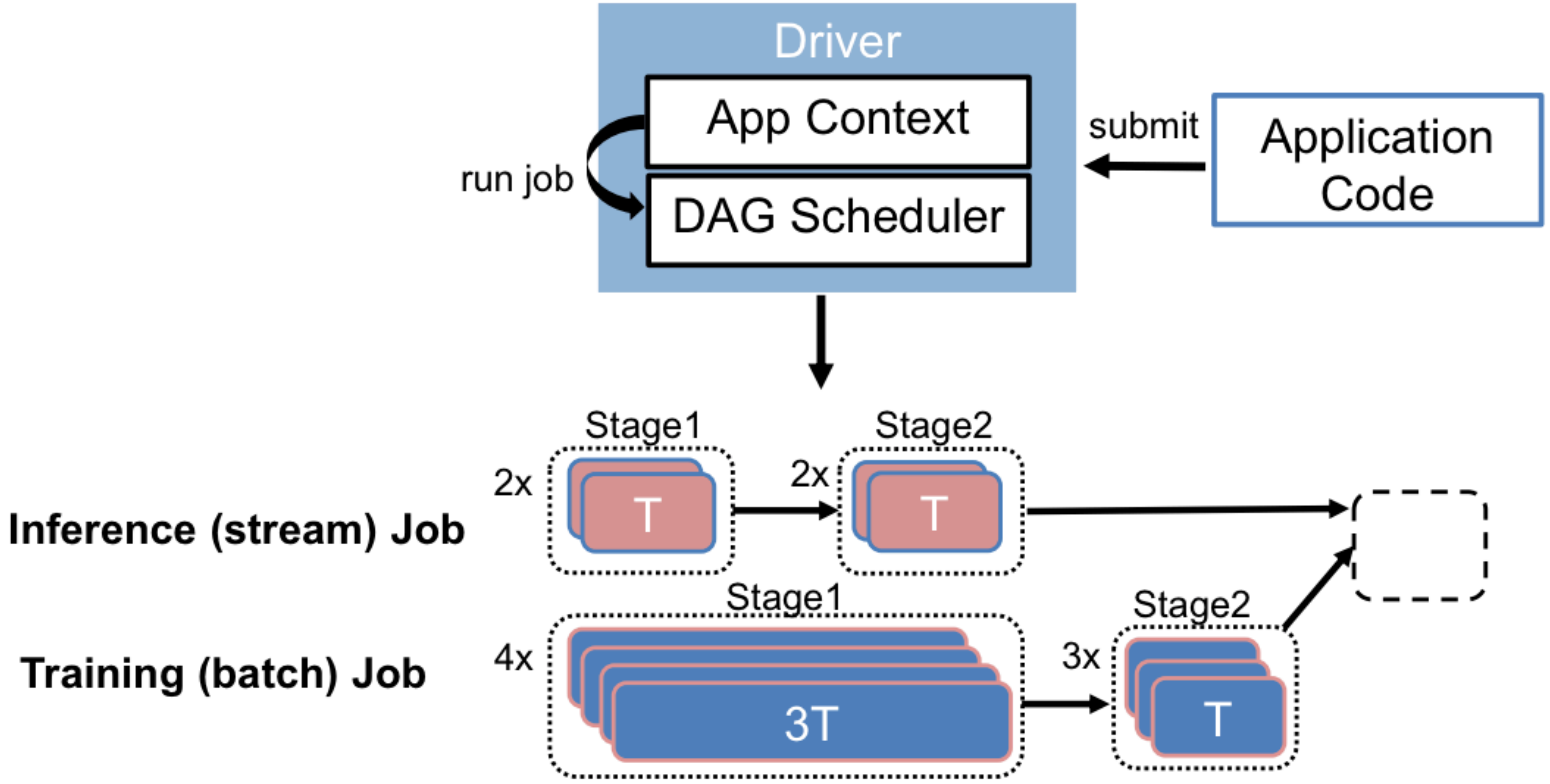


Stream/Batch application requirements

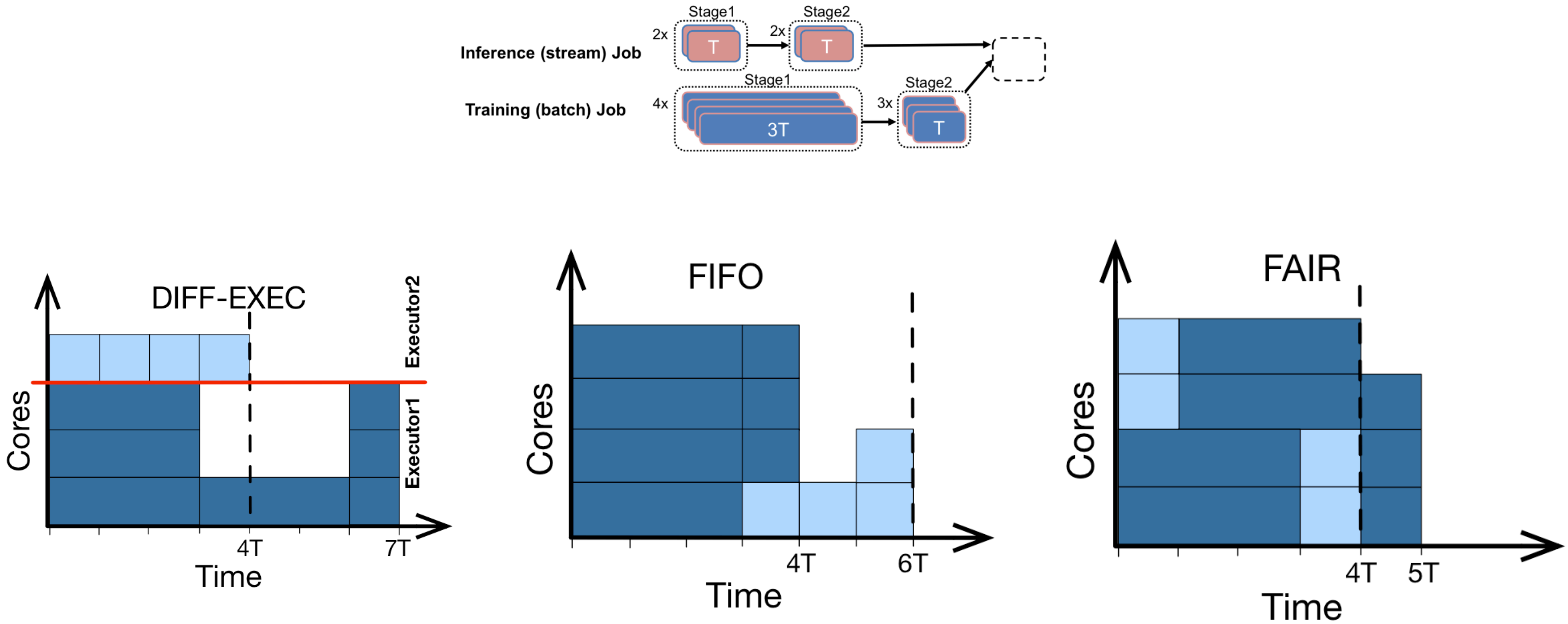
- **Requirements**
  - Latency: Execute inference job with minimum delay
  - Throughput: Batch jobs should not be compromised
  - Efficiency: Achieve high cluster resource utilization

Challenge: schedule stream/batch jobs to satisfy their diverse requirements

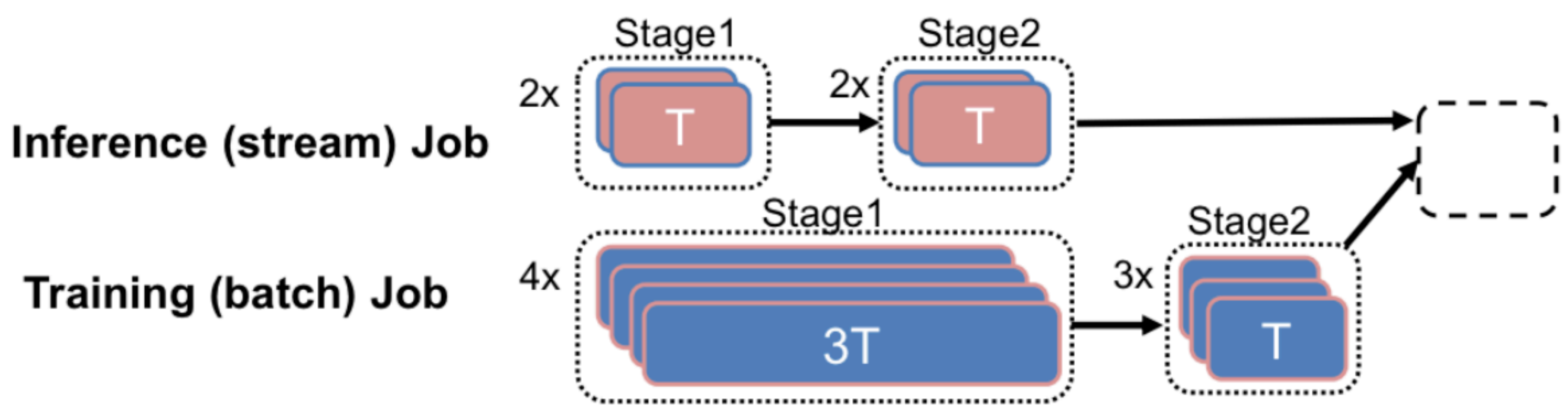
Stream/Batch application scheduling

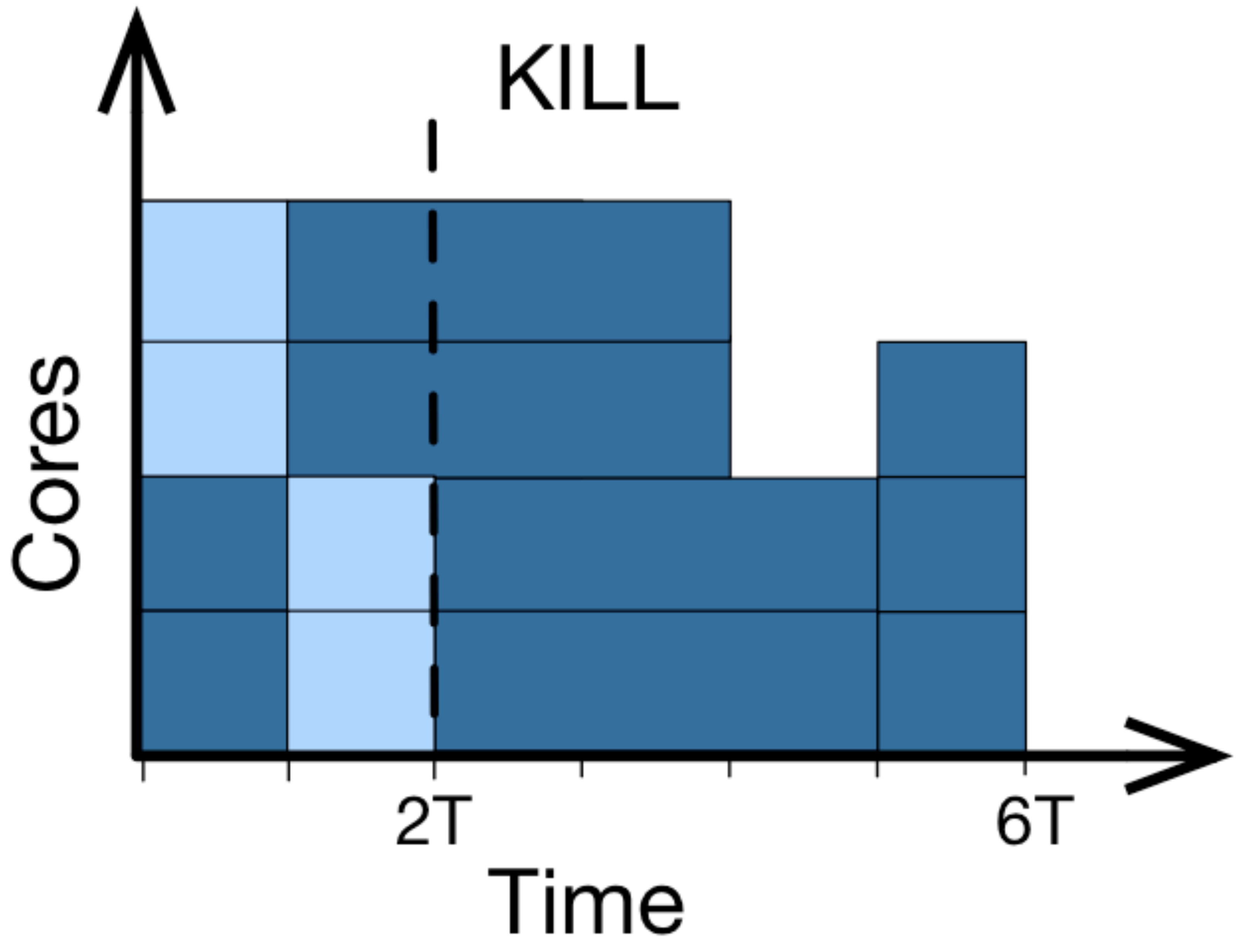


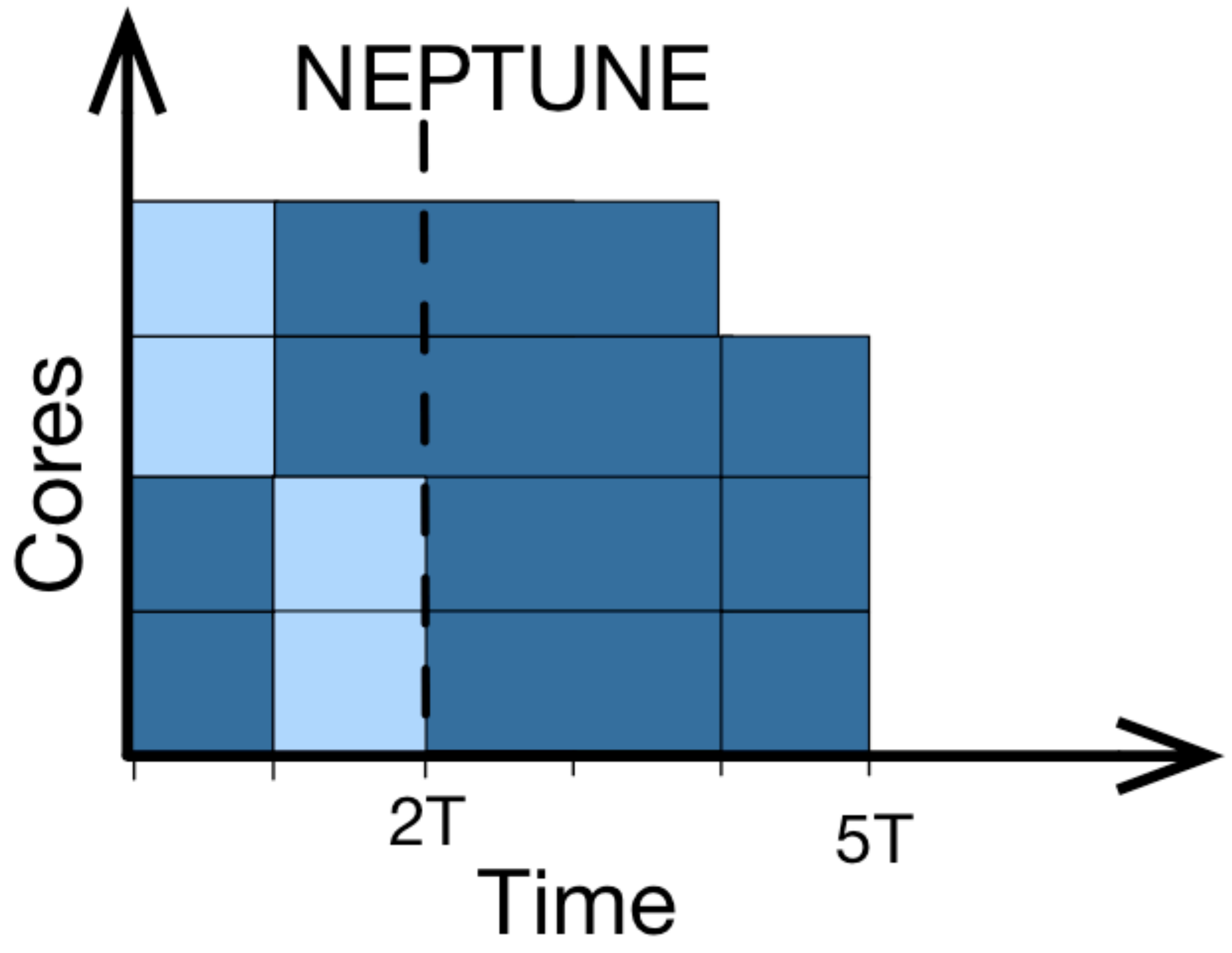
Stream/Batch application scheduling



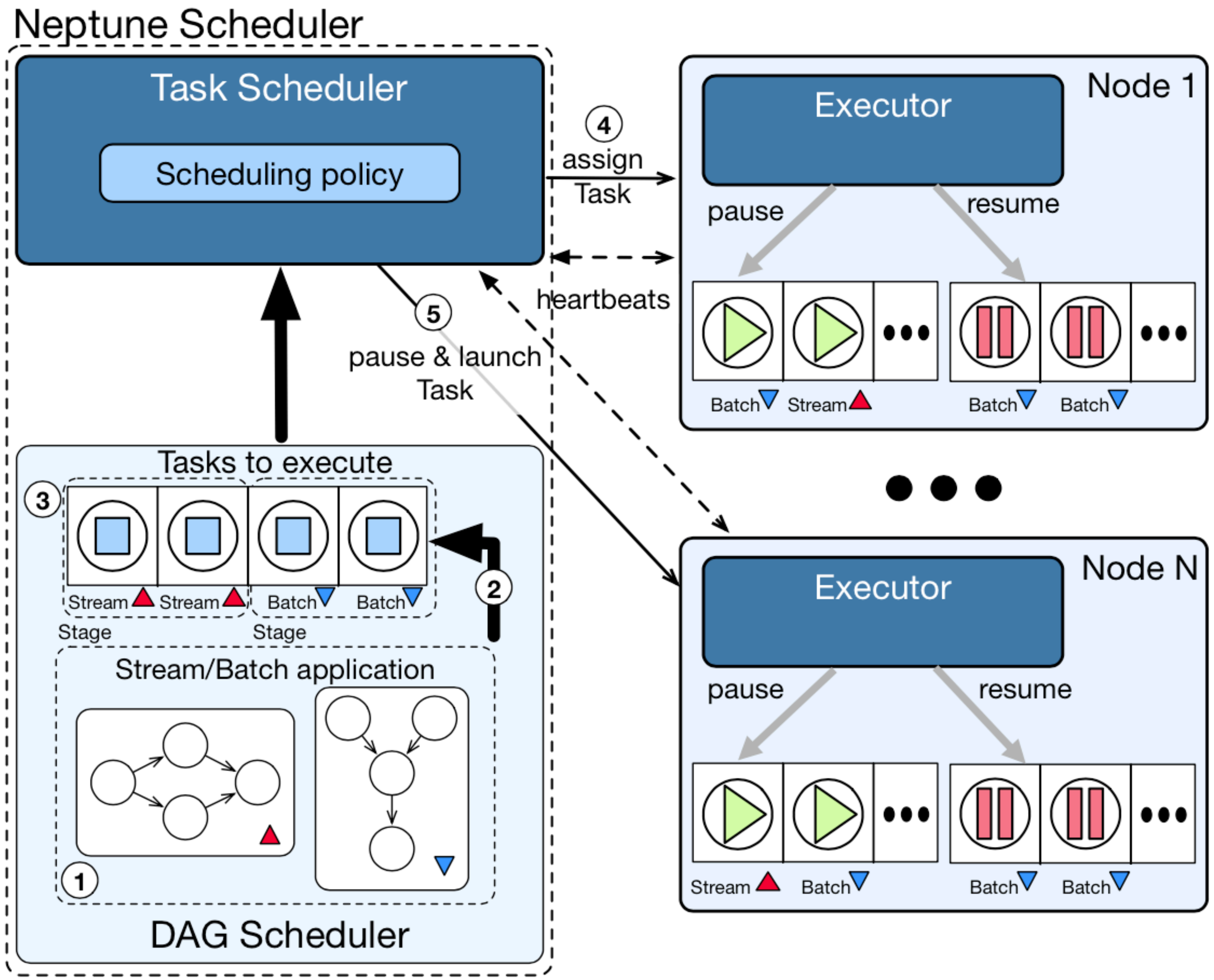
Stream/Batch application scheduling







Neptune Design



Suspendable Tasks

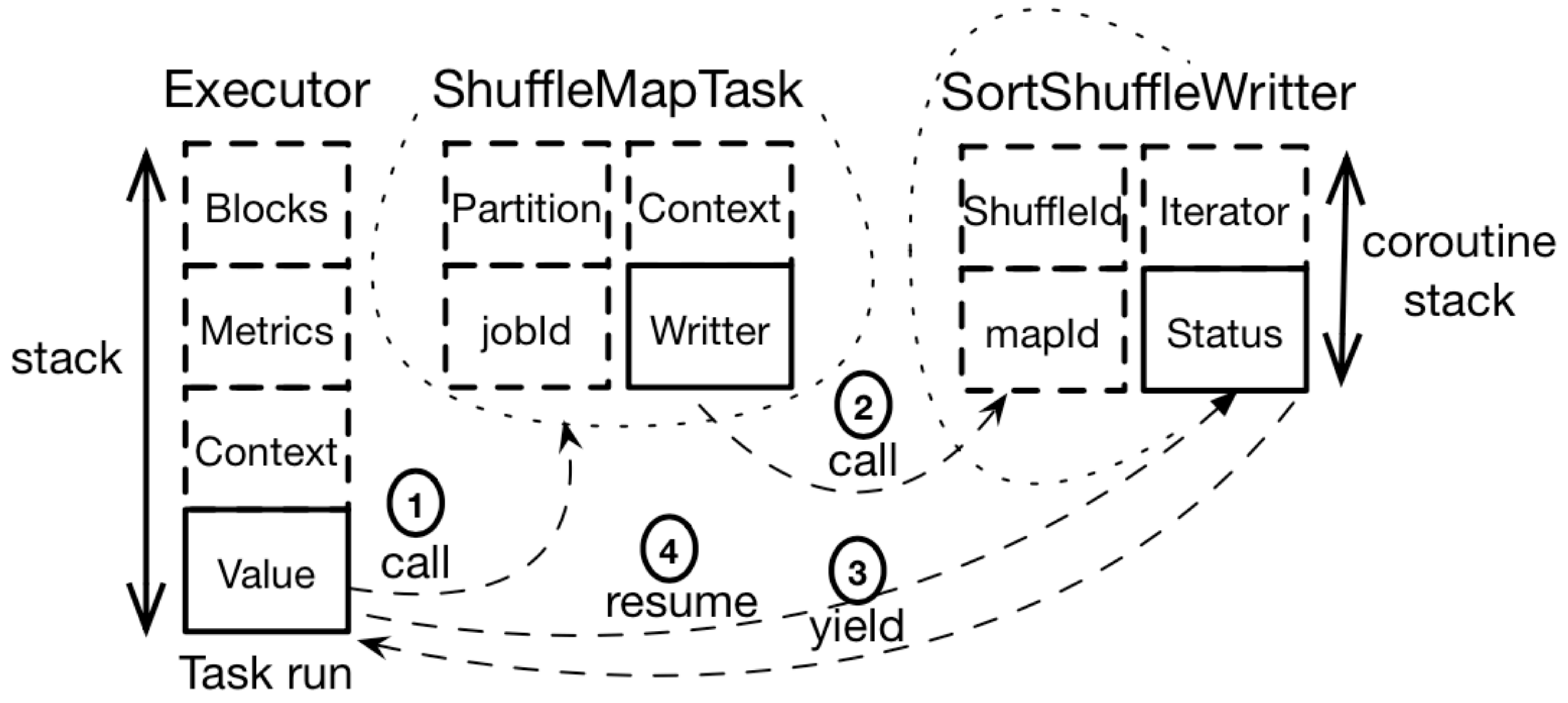
Neptune uses stackful coroutines to implement suspendable tasks, which have a yield point after the processing of each record.

Listing 2: Collect coroutine task

```
val collect : (TaskContext, Iterator[T]) -> (Int, Array[T]) =
  coroutine {(context: TaskContext, itr: Iterator[T]) =>
    val result = new mutable.ArrayBuffer[T]
    while (itr.hasNext) /* iterate records */
    | result.append(itr.next) /* append record to dataset */
    | if (context.isPaused()) /* check task context */
    ⌊ ⌊ yieldval(0) /* yield value to caller */
  result.toArray /* return result dataset */
```

Suspendable Tasks

Neptune uses stackful coroutines to implement suspendable tasks, which have a yield point after the processing of each record.



Scheduling policies

- **Idea**: policies trigger task suspension and resumption
  - Guarantee that stream tasks <u>bypass</u> batch tasks
  - Satisfy <u>higher-level objectives</u> i.e. balance cluster load
  - Avoid <u>starvation</u> by suspending up to a number of times

- **Load-balancing** (LB): takes into account executors' memory conditions and equalize the number of tasks per node
- **Locality- and memory aware** (LMA): respect task locality preferences in addition to load-balancing

LMA Scheduling policies

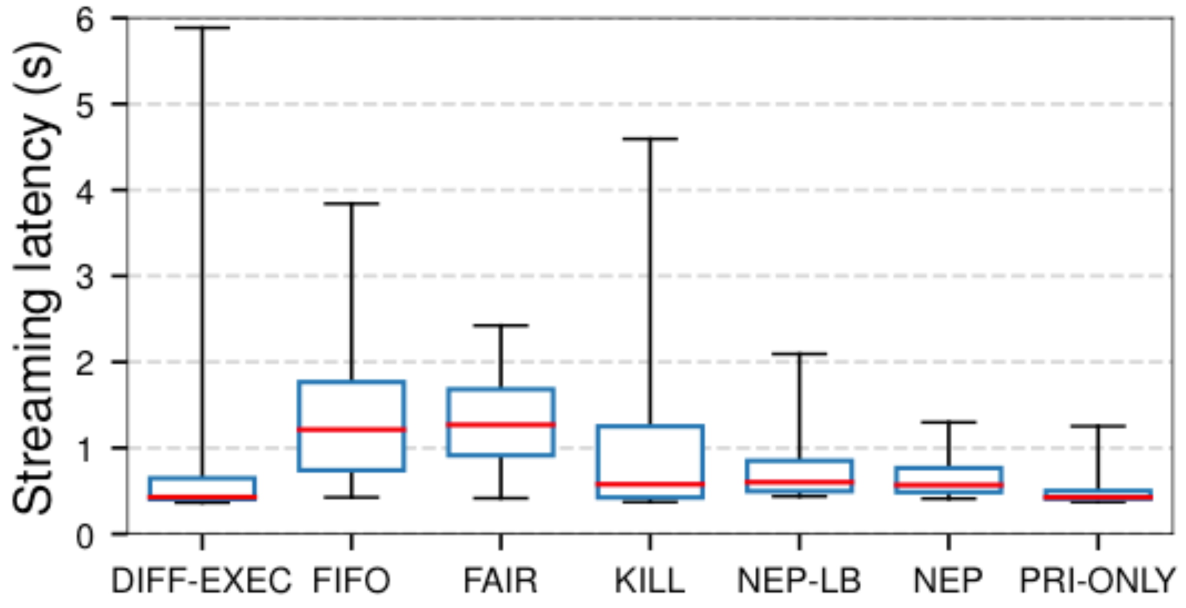
Algorithm 1: LMA scheduling policy

Input: List Executors, // In descending preference order
List ExecutorsMetricsWindow // Executor metrics

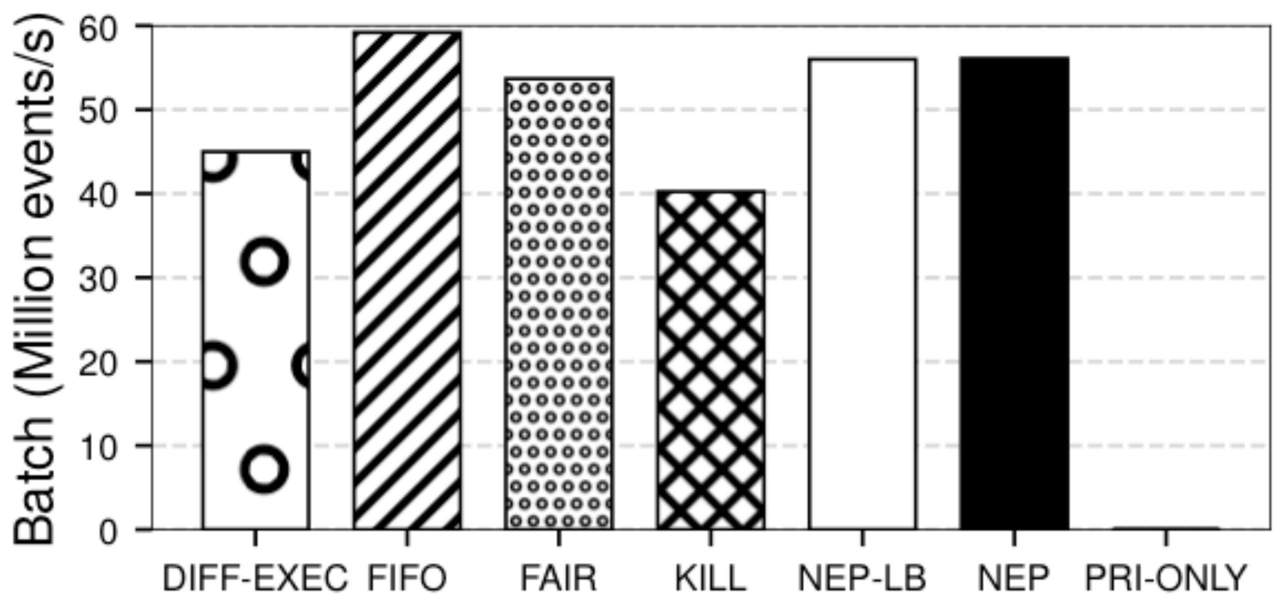
Upon event *HeartBeat hb from Executor e* **do**
  ExecutorsMetricsWindow[e].push(hb.metrics)
  Executors.updateOrdering
return

Upon event *Submit Task t* **do**
  // Cache local executor
  Executor $t_{exec} \leftarrow$ hostToExecutor.get(t.cacheLocation)
  if t_{exec} *is None* **or** t_{exec}*.freeMemory* $<$ *threshold* **then**
    // Executor with the least pressure
    $t_{exec} \leftarrow$ Executors.head
  if t_{exec} *has availableCores* **then**
    // Launch task t on free cores
    t_{exec}.Launch(t)
  else
    // Suspend task t_p and launch t
    Task $t_p \leftarrow t_{exec}$.tasks.filter(LowPriority)
    t_{exec}.PauseAndLaunch(t, t_p)
return

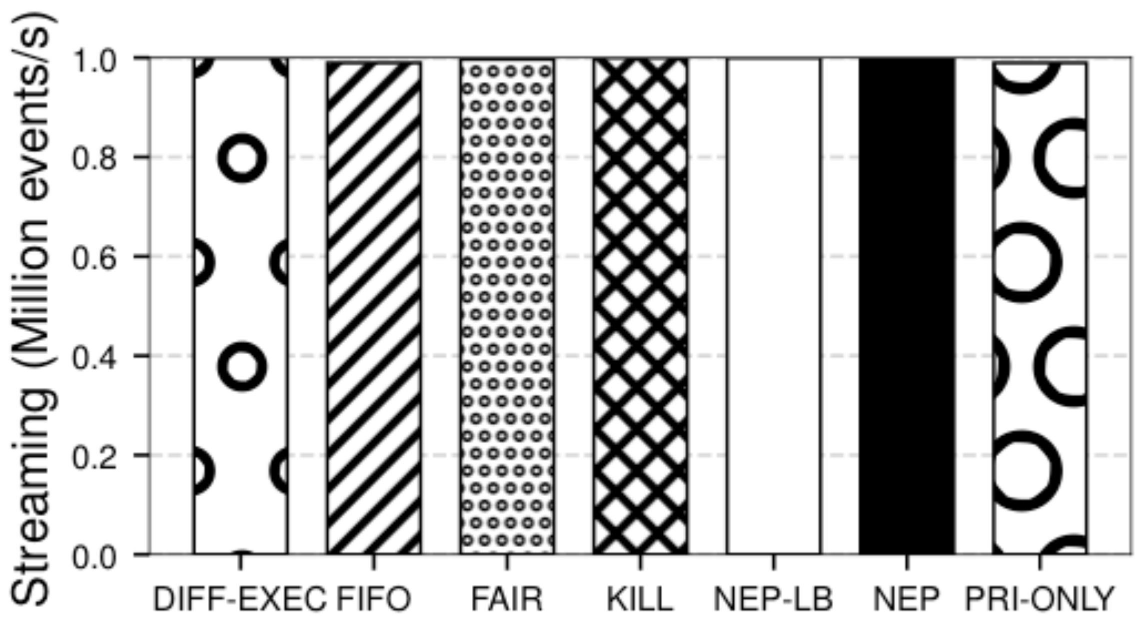
Evaluation



(a) Streaming query latency

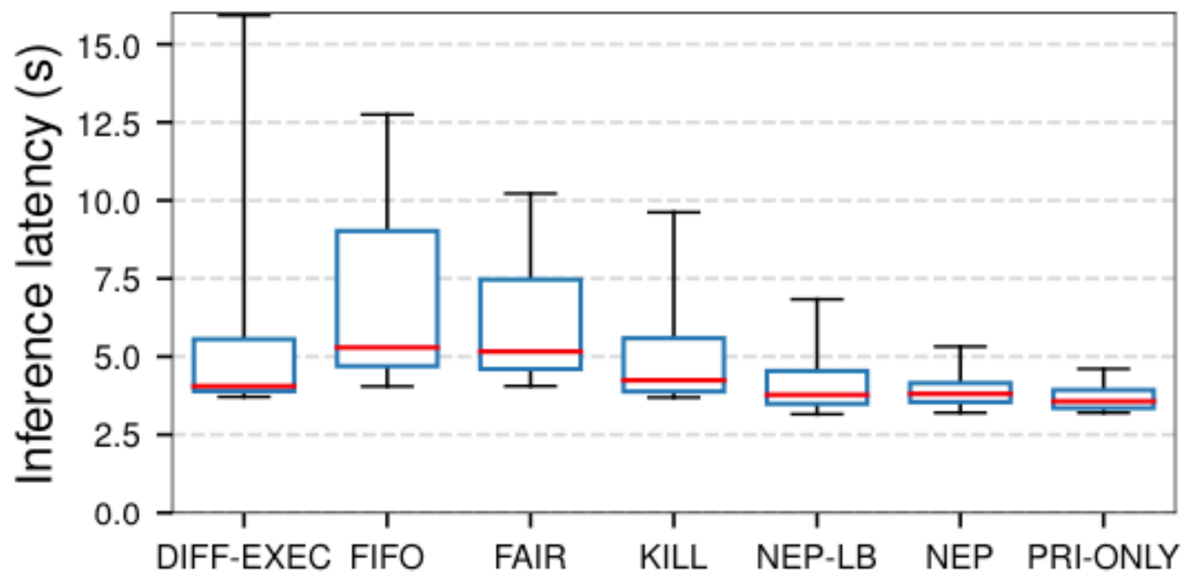


(b) Batch query throughput

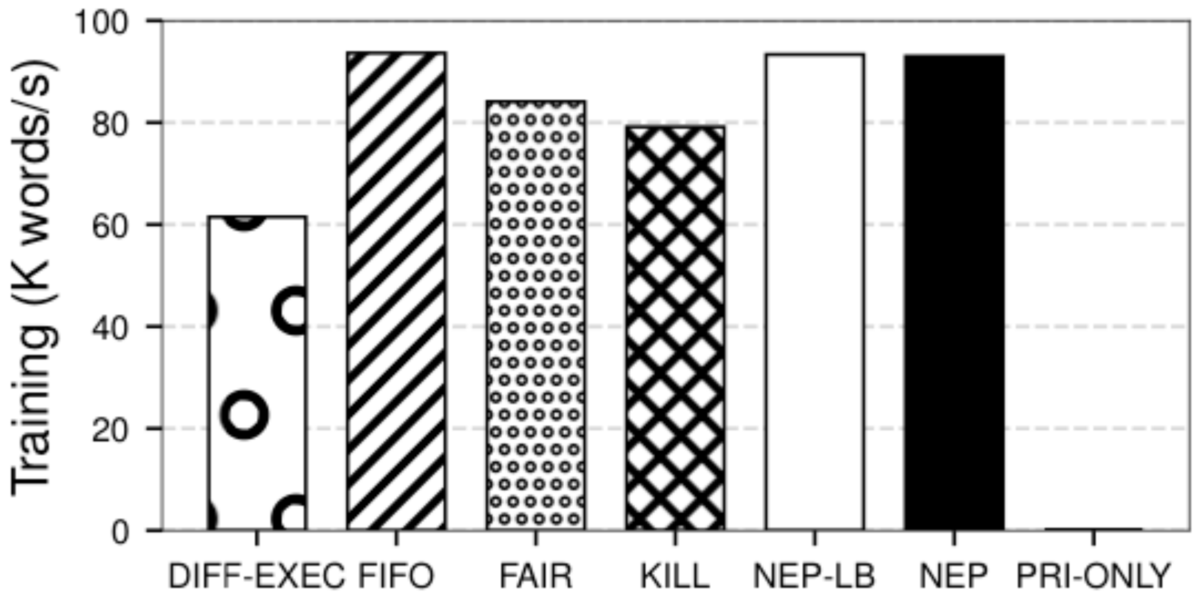


(c) Streaming query throughput

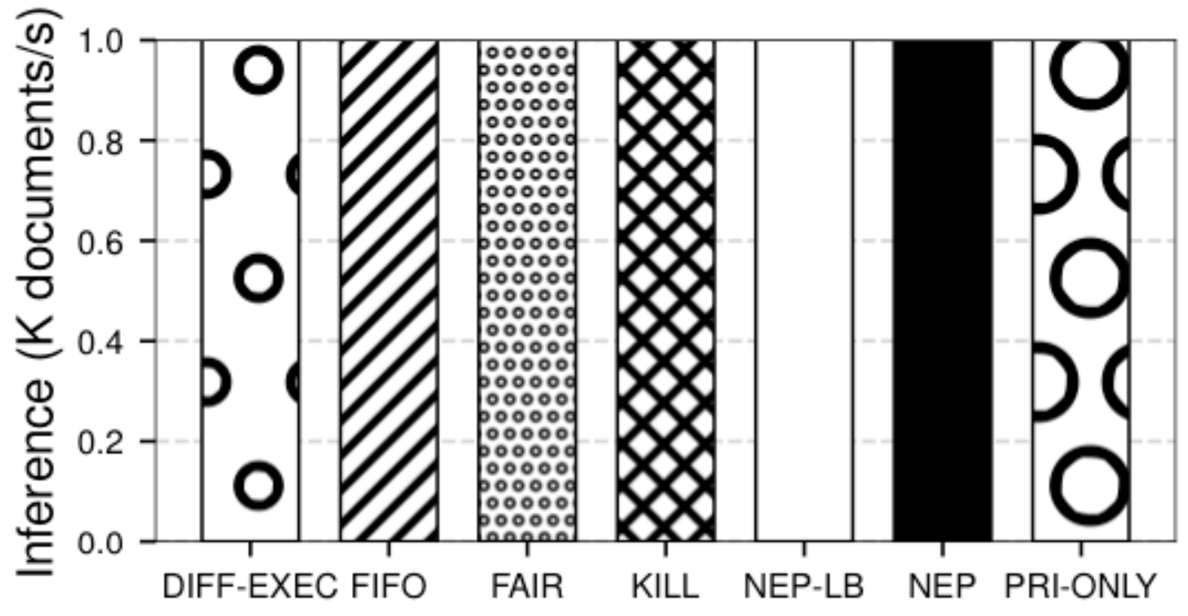
Figure 9: Yahoo Streaming benchmark (streaming + batch)



(a) Inference latency

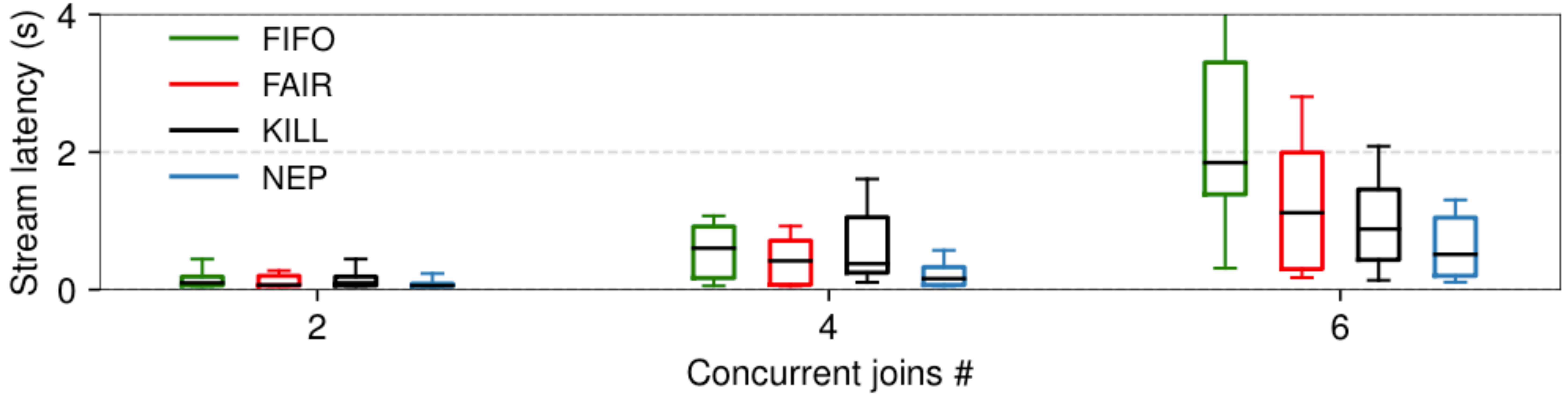


(b) Training query throughput



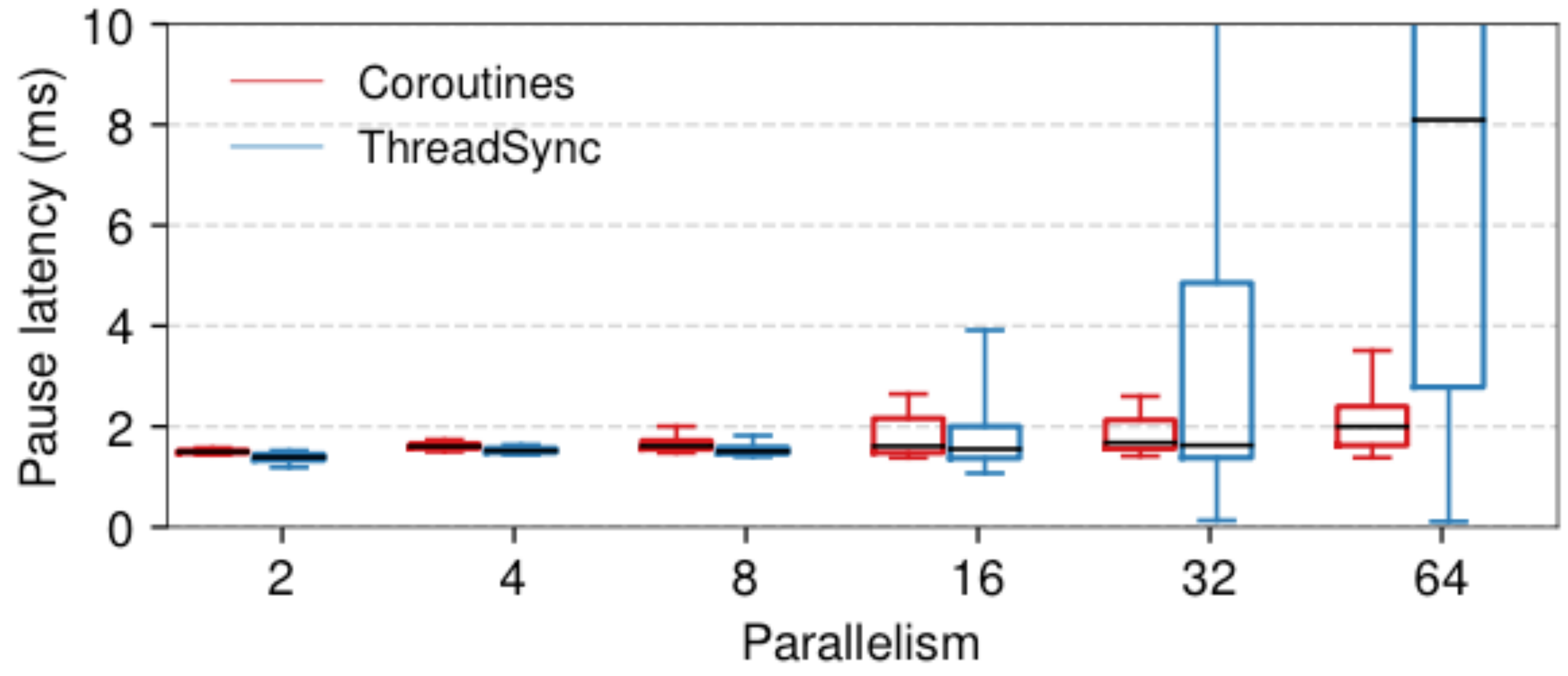
(c) Inference query throughput

Evaluation



Stream latency (s)
4
2
0
FIFO
FAIR
KILL
NEP
2
4
6
Concurrent joins #

Evaluation



Coroutines
ThreadSync
Pause latency (ms)
10
8
6
4
2
0
2
4
8
16
32
64
Parallelism

Summary

- **Medea**:
  - Support container tags and logical node groups
  - Expressive cardinality constraints
  - Two scheduler design

- **Neptune**
  - Suspendable Tasks
  - LMA Scheduling policies

Thanks