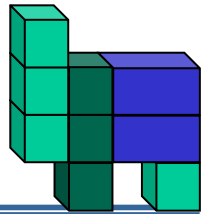
# TetriSched: global rescheduling with adaptive plan-ahead in dynamic heterogeneous clusters

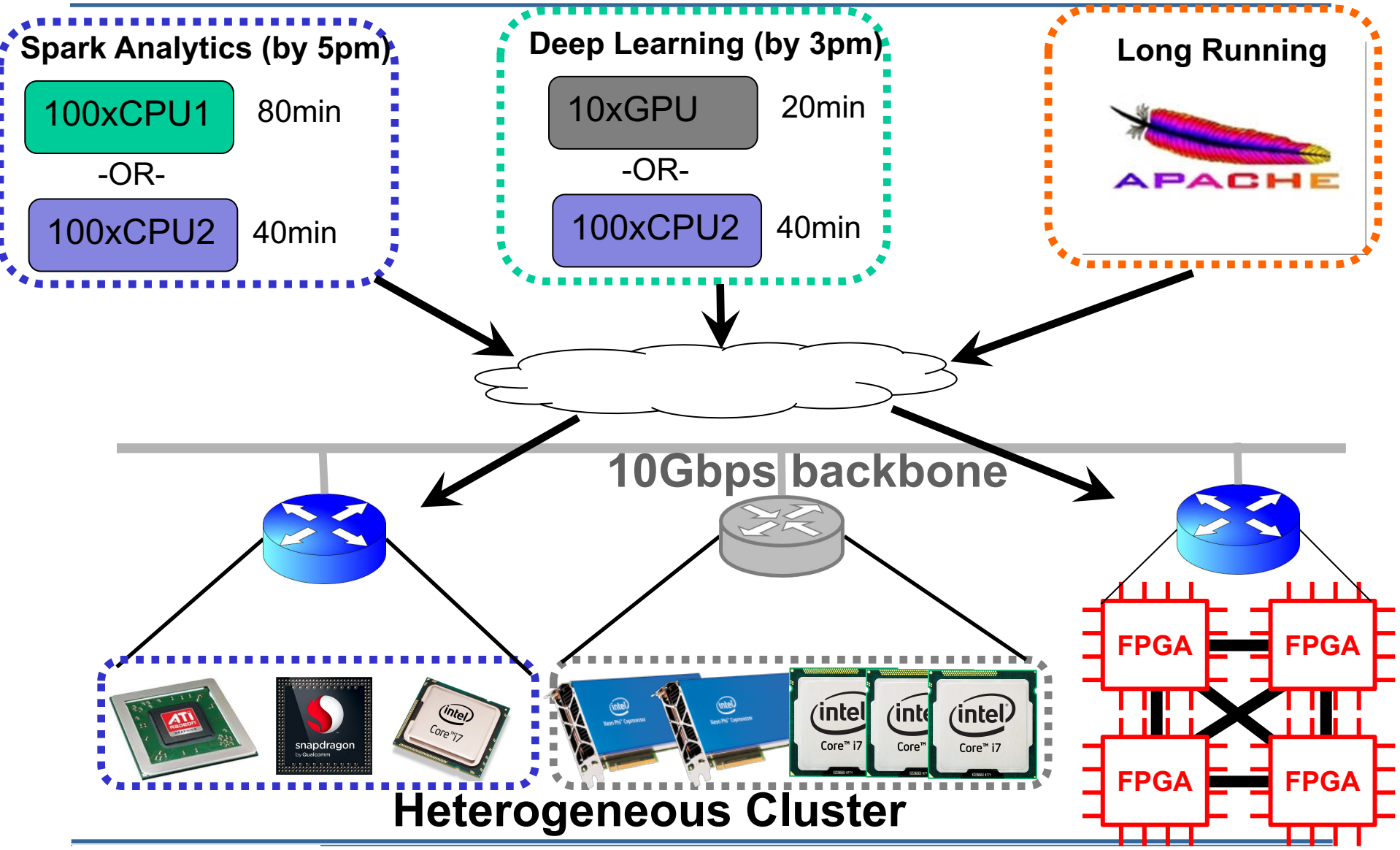
*Alexey Tumanov*

Timothy Zhu, Jun Woo Park,

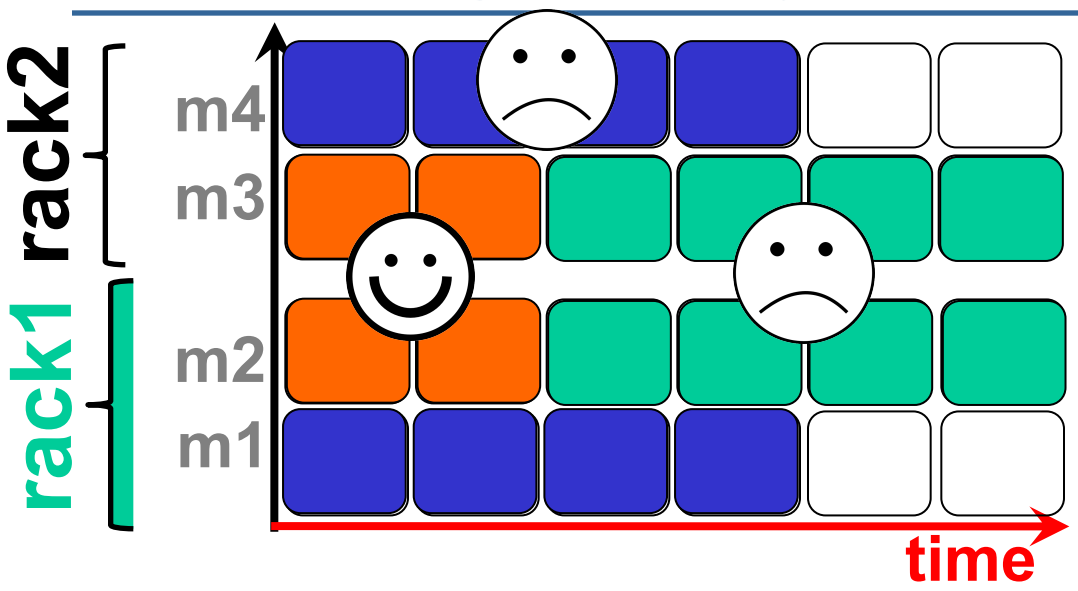
Michael Kozuch, Mor Harchol-Balter,
Gregory R. Ganger

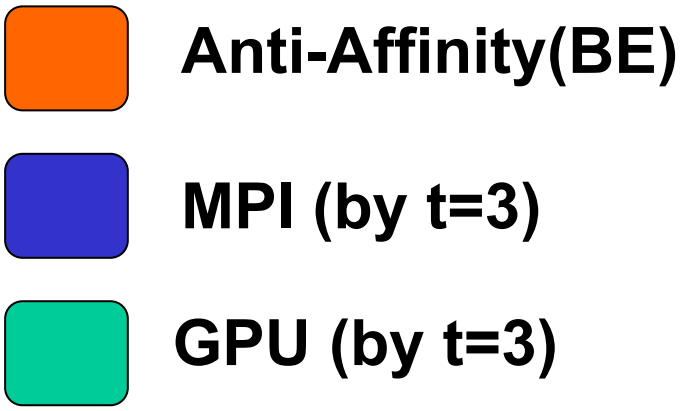


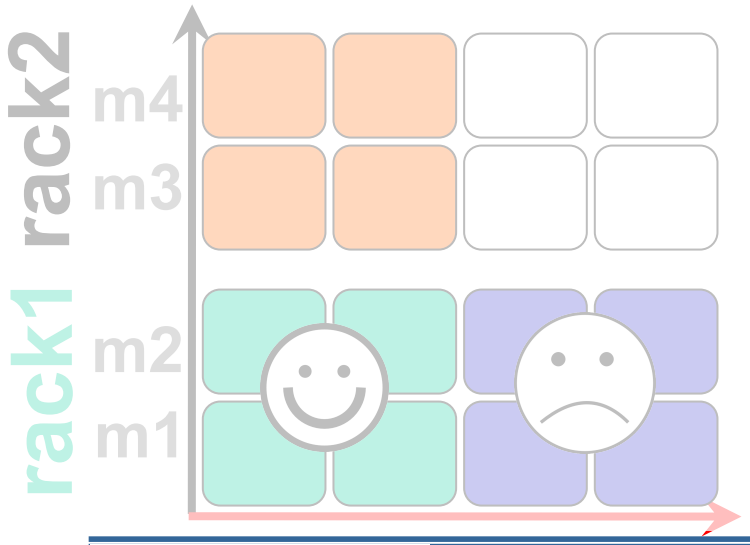
# Motivation

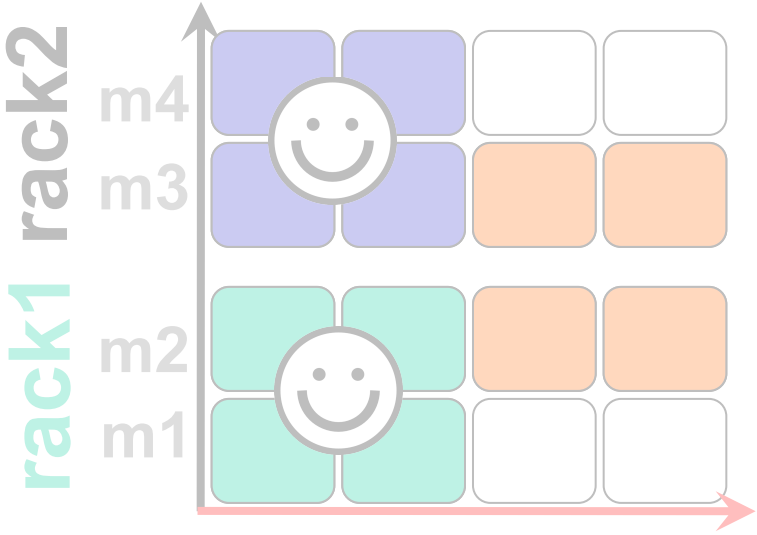

Spark Analytics (by 5pm)
100xCPU1
80min
-OR-
100xCPU2
40min
Deep Learning (by 3pm)
10xGPU
20min
-OR-
100xCPU2
40min
Long Running
APACHE
10Gbps backbone
FPGA
FPGA
FPGA
FPGA
Heterogeneous Cluster

# Heterogeneity Amplifies Options



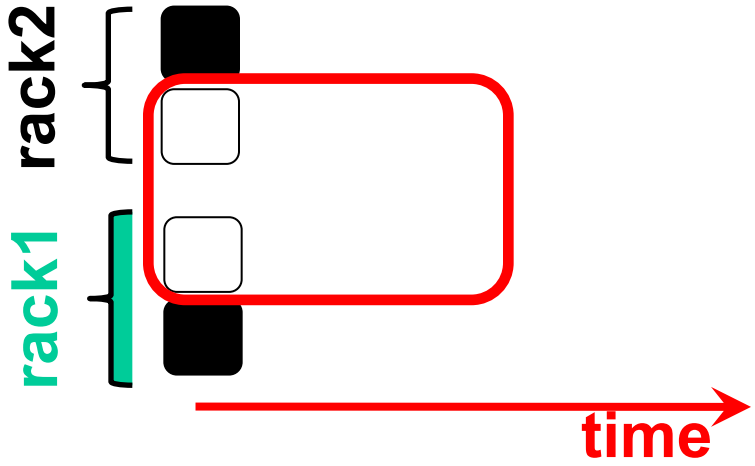



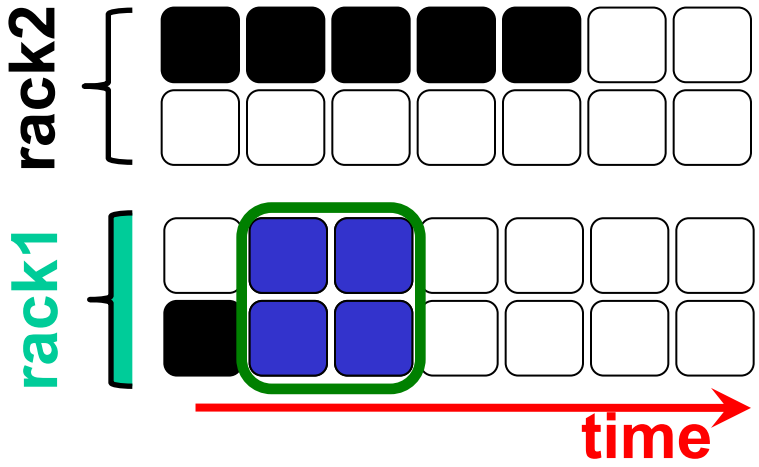
# Exploiting Time Flexibility (Plan-ahead)

- Previously: just look at current state



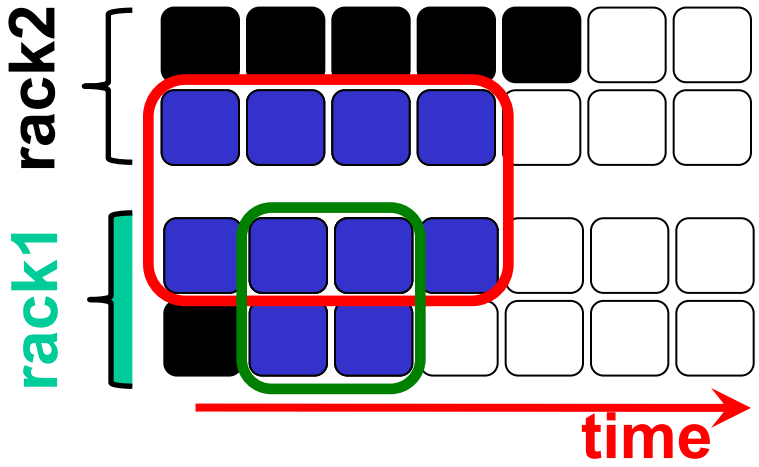
# Exploiting Time Flexibility (Plan-ahead)

- Previously: just look at current state
- Plan-ahead: estimate runtimes and choices



# Exploiting Time Flexibility (Plan-ahead)

- Previously: just look at current state
- Plan-ahead: estimate runtimes and choices
  - should this job wait for better placement?

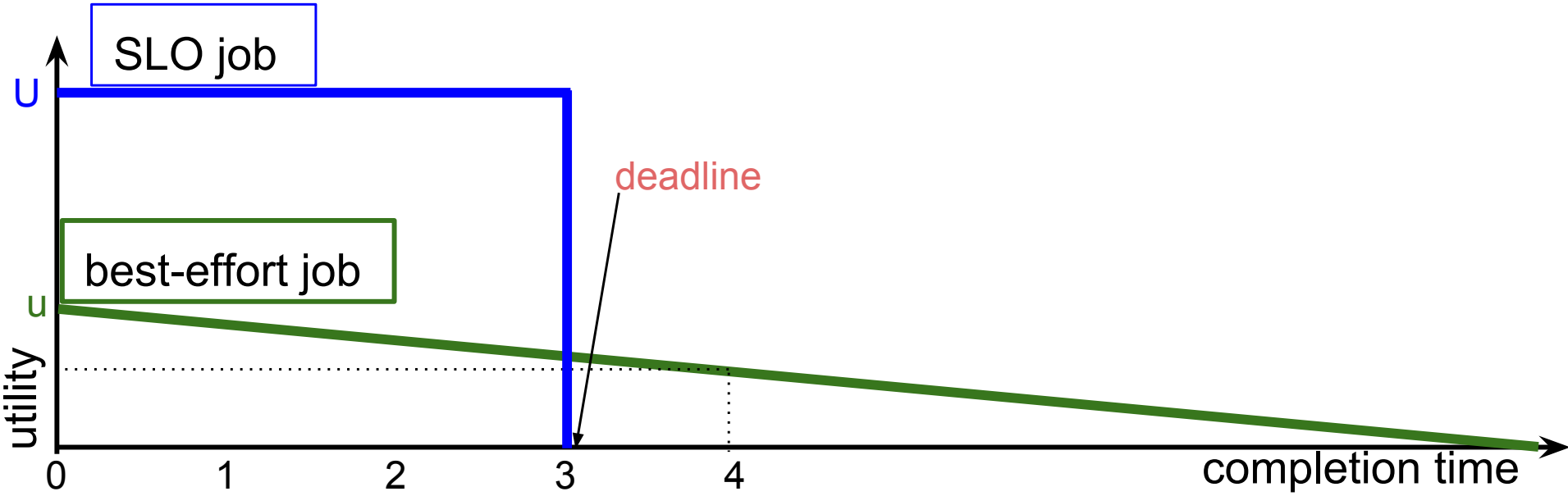


# Key Challenges

- Leverage runtime estimates *robustly*
- Express combinatorially many options succinctly
  - Including quantifying their relative merit
- Exploit this knowledge to improve allocation
  - All in a practical manner …

# Quantify: Internal Utility Functions

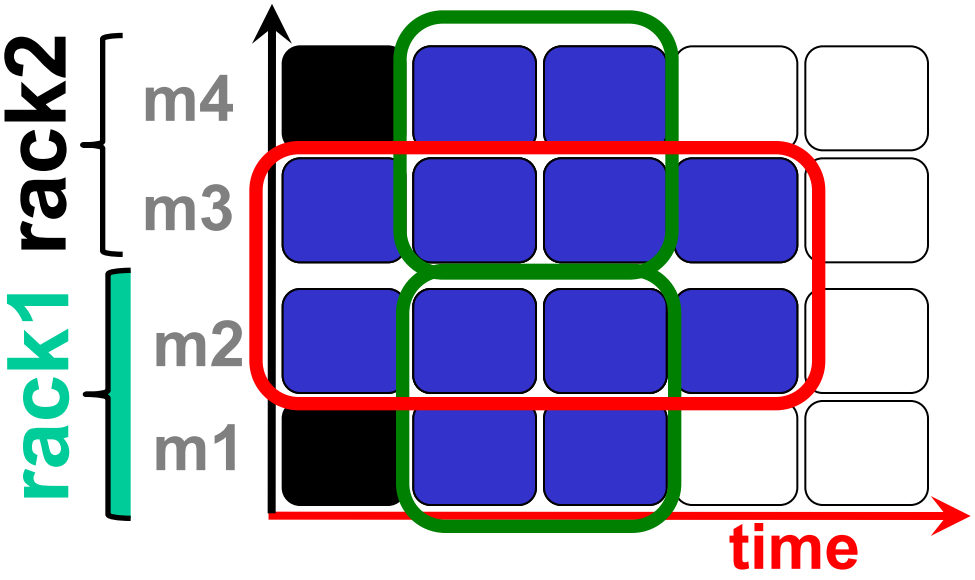
- SLO jobs: zero value after deadline
- BE jobs: lower value for longer completion time
- Higher value for more important jobs (**U** > **u**)



# Express: Space-Time Request Language

- Utility u(p,t): placement p@t → scalar value **u**

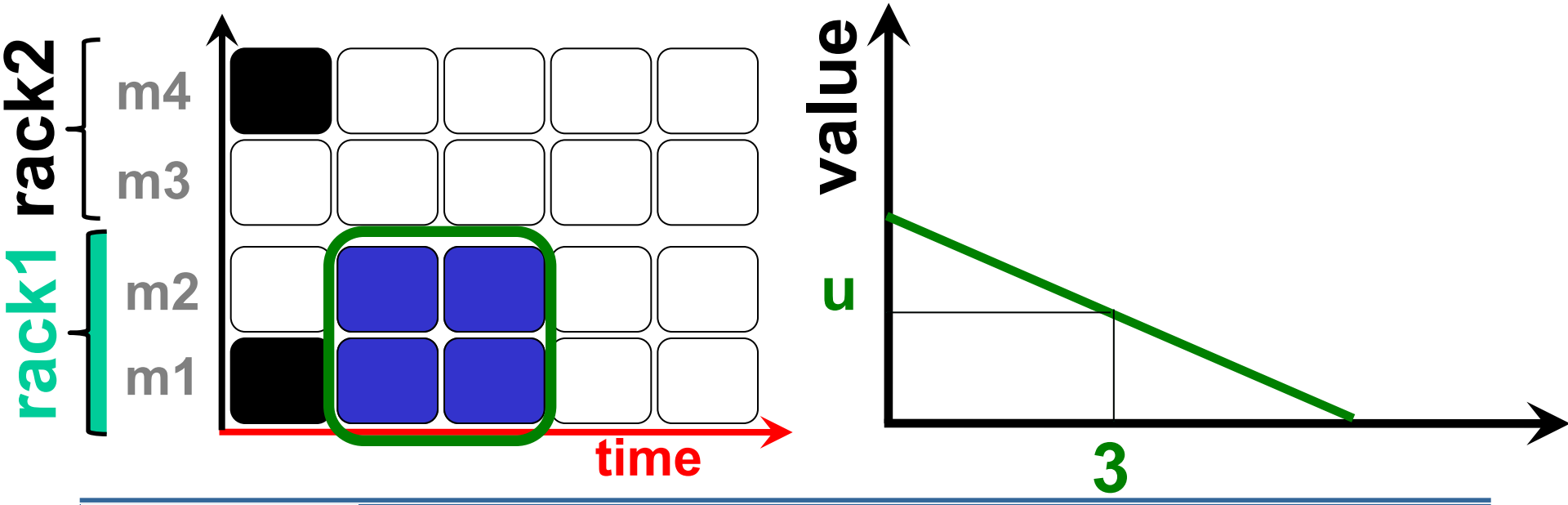
- **"n Choose k" (nCk)**
  n → refers to a group of nodes to choose from
  k → how many nodes to choose



# Express: Space-Time Request Language

- Utility u(p,t): placement p@t → scalar value **u**

**nCk** ($m_i \in \text{rack1}$, **k=2, s=1, d=2, u** )

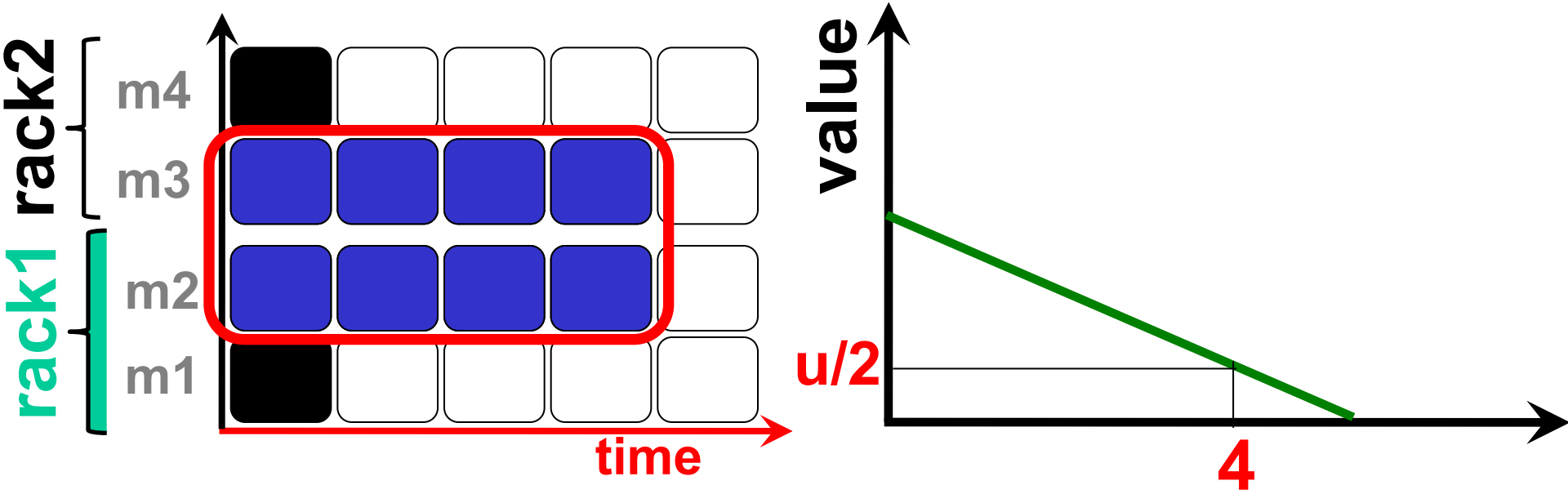


# Express: Space-Time Request Language

- Utility u(p,t): placement p@t → scalar value **u**

**nCk** ($m_i \in \text{rack1}$, **k=2, s=1, d=2, u** )

**nCk** ( $\cup m_i$, **k=2, s=0, d=4, u/2**)

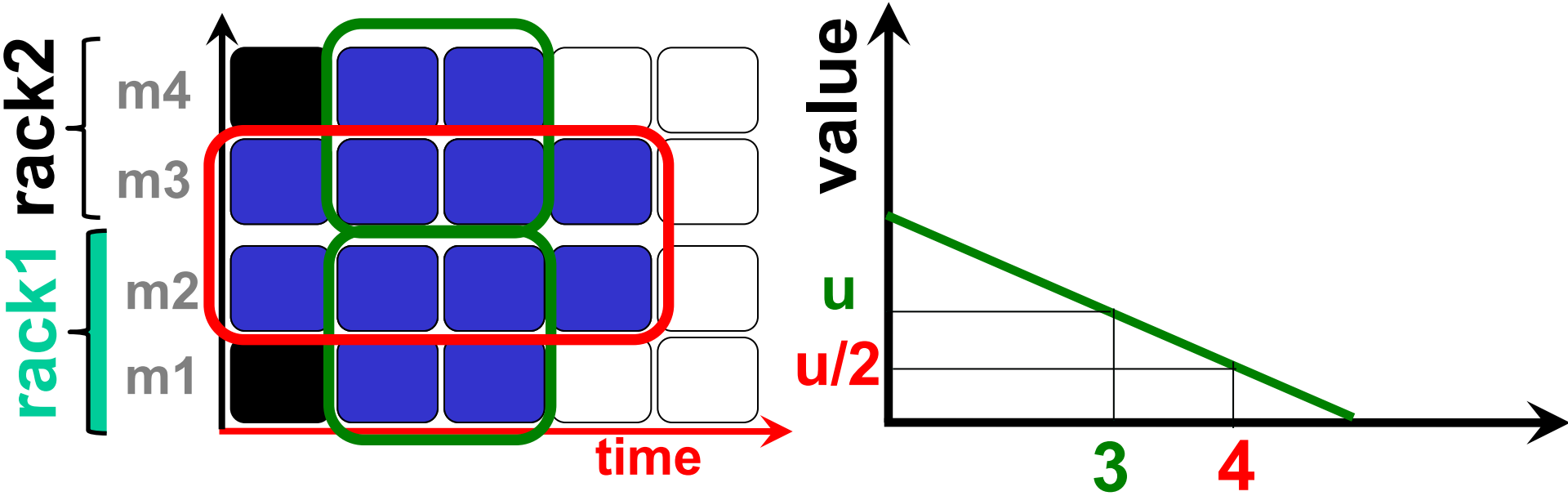


# Express: Space-Time Request Language

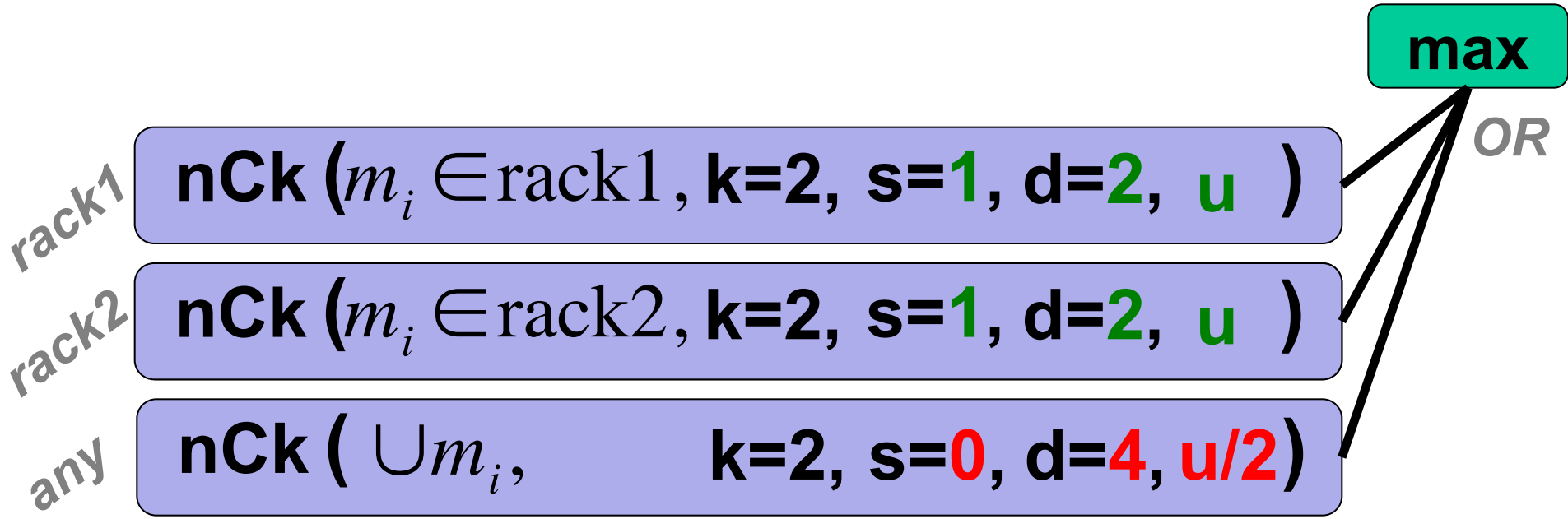
- Utility u(p,t): placement p@t → scalar value **u**

**nCk** ($m_i \in \text{rack1}$, **k=2, s=1, d=2, u** )

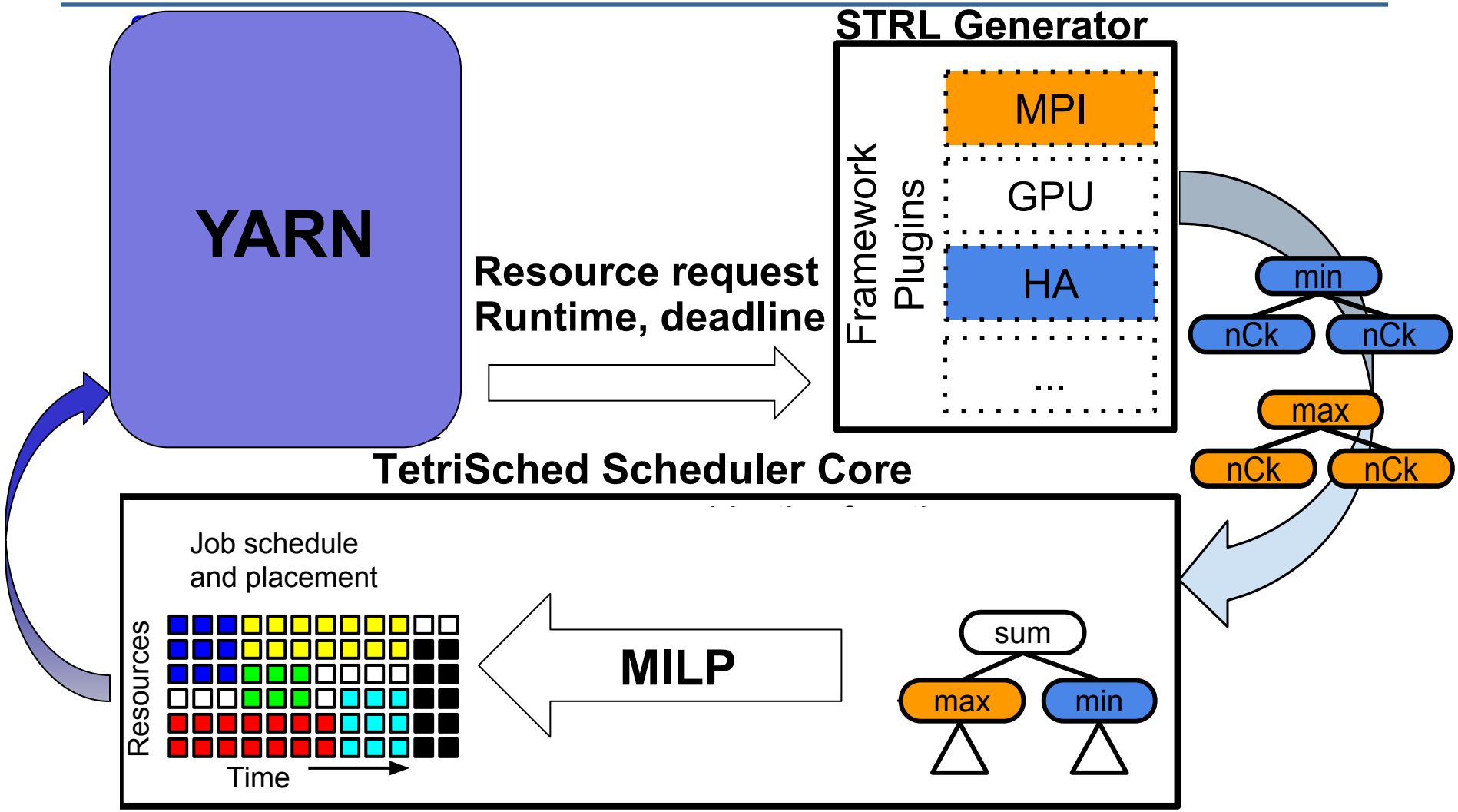
**nCk** ( $\cup m_i$, **k=2, s=0, d=4, u/2**)



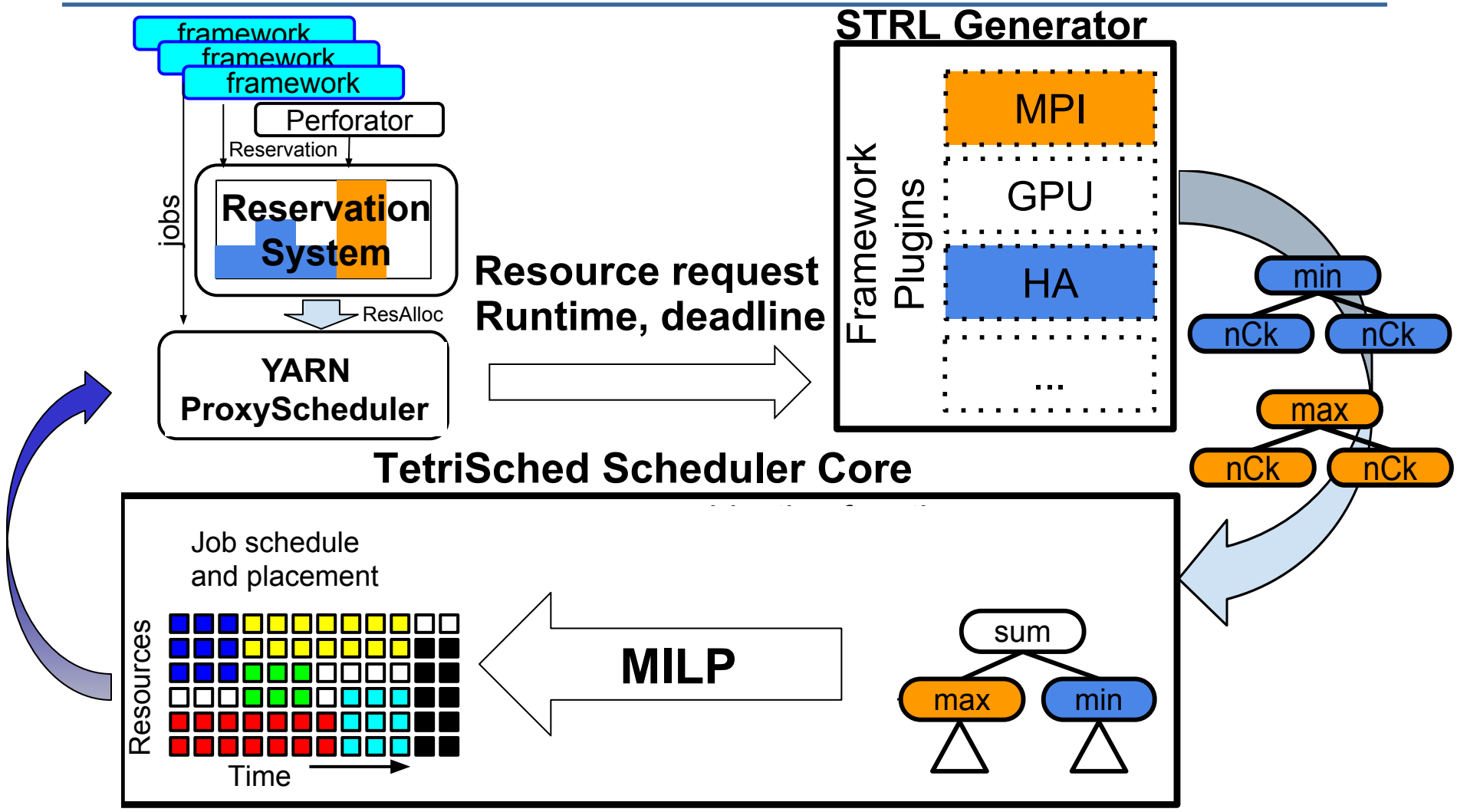
# STRL Expression Composition



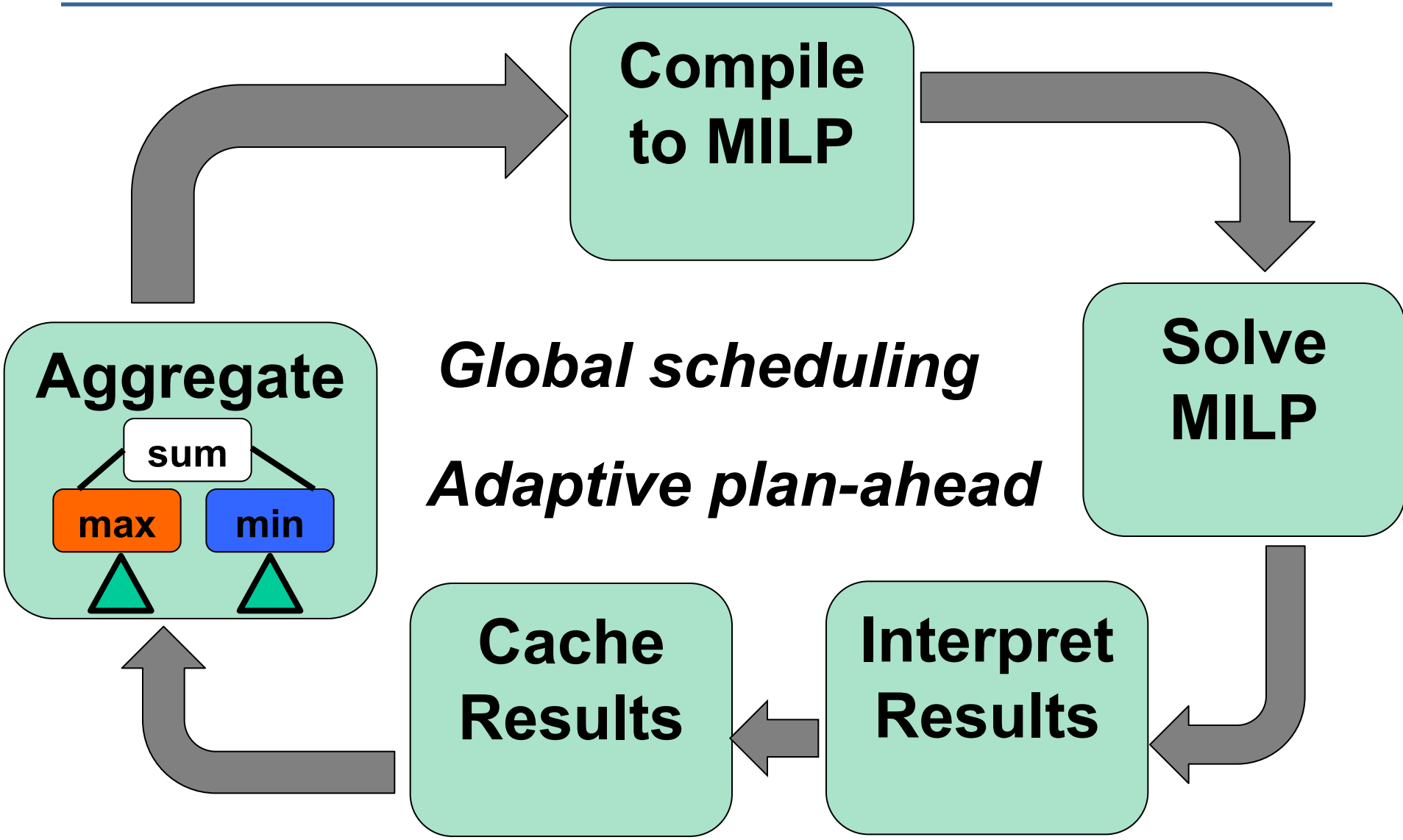
# TetriSched System Architecture



# TetriSched System Architecture



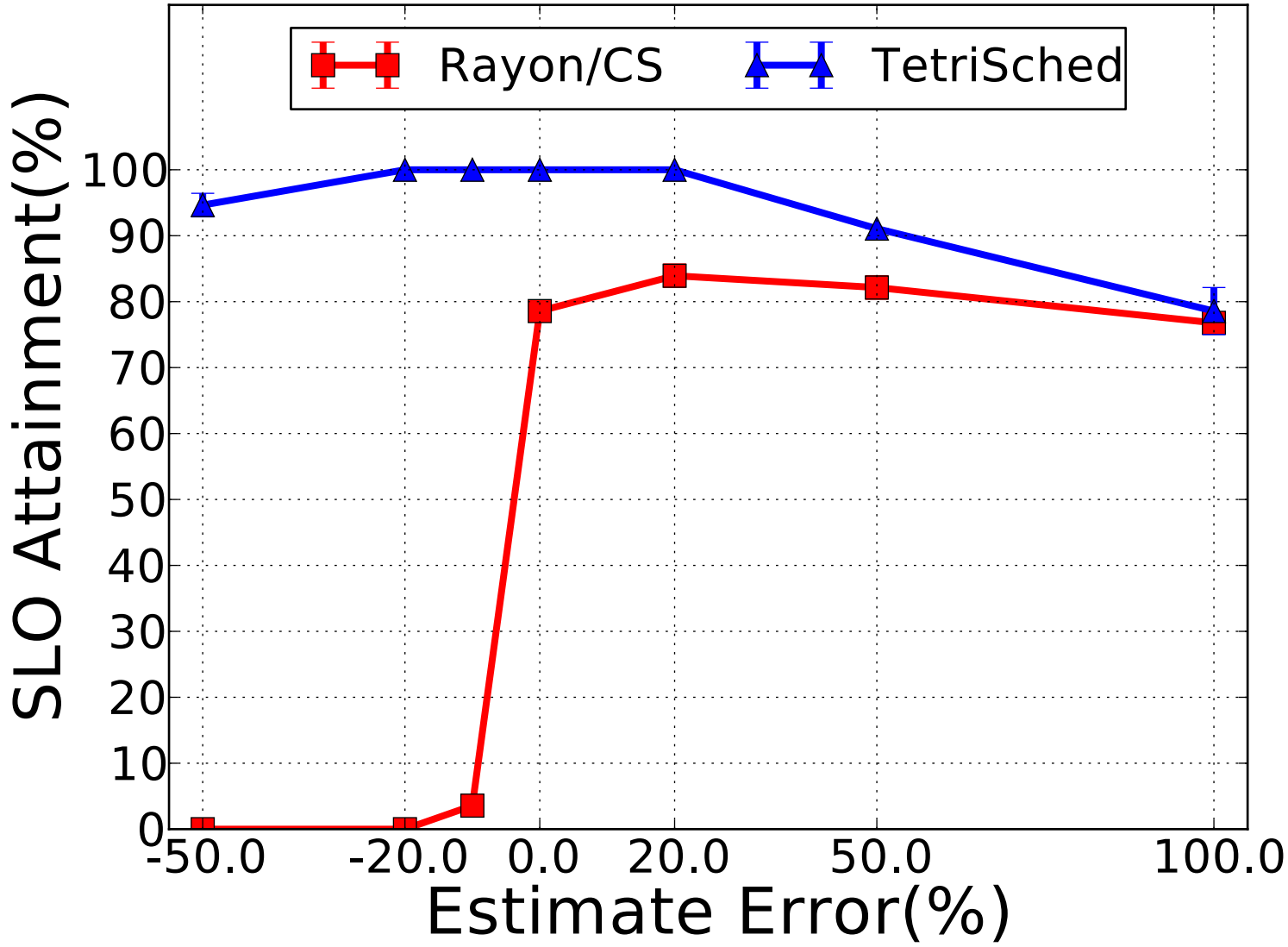
# TetriSched Scheduler Core



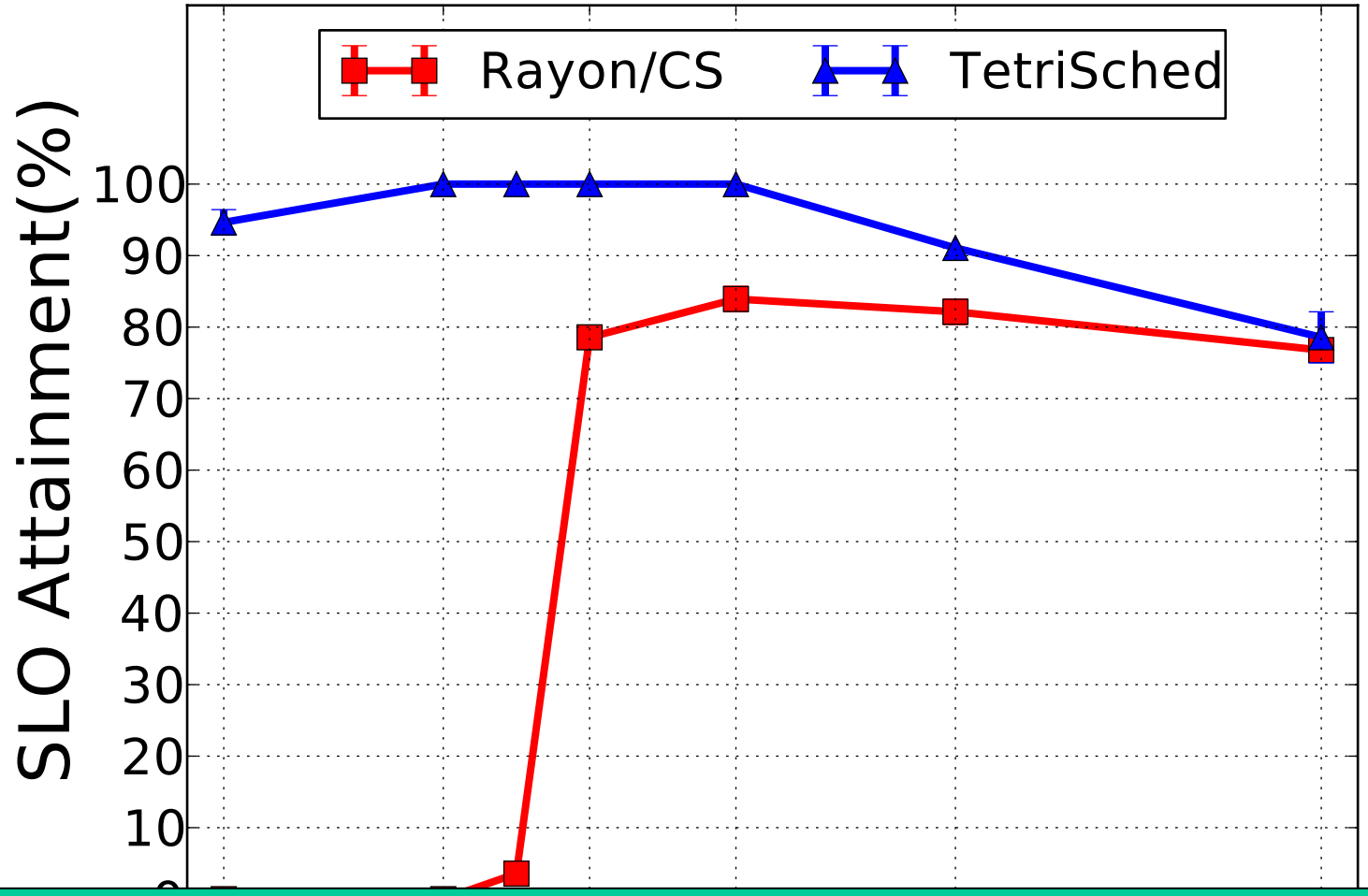
# Experimental Results

- Real cluster:
  - 256 nodes (8 racks, 32 nodes/rack)
- Workloads:
  - Production derived: Facebook SLO + Yahoo BE job mix
- Systems compared:
  - **Rayon / CapacityScheduler**
  - **Rayon / TetriSched**
- Takeaway:
  - *With global rescheduling and plan-ahead, TetriSched outperforms Rayon/CapacityScheduler stack*

# Leverage runtime estimates robustly

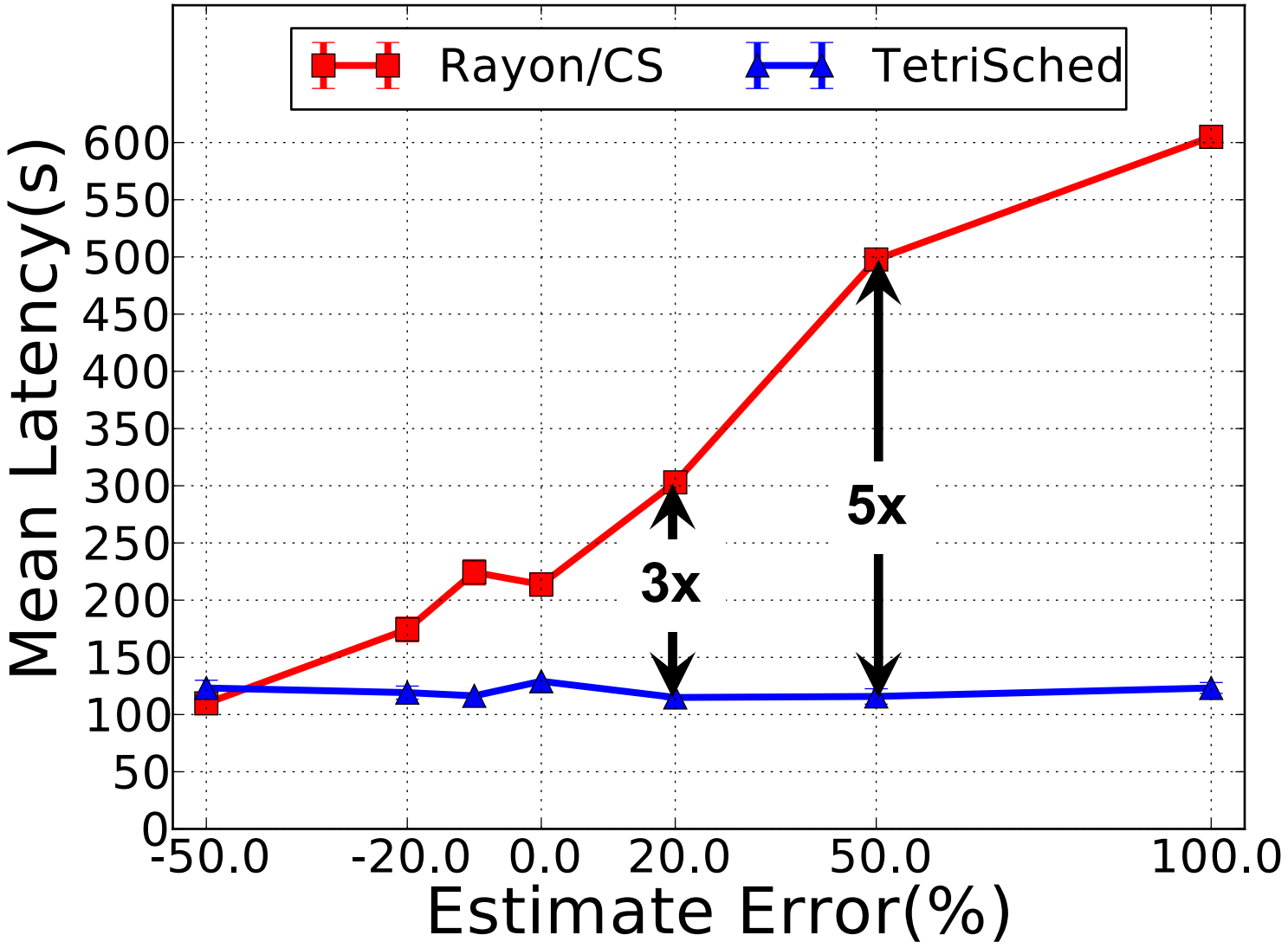


# Leverage runtime estimates robustly

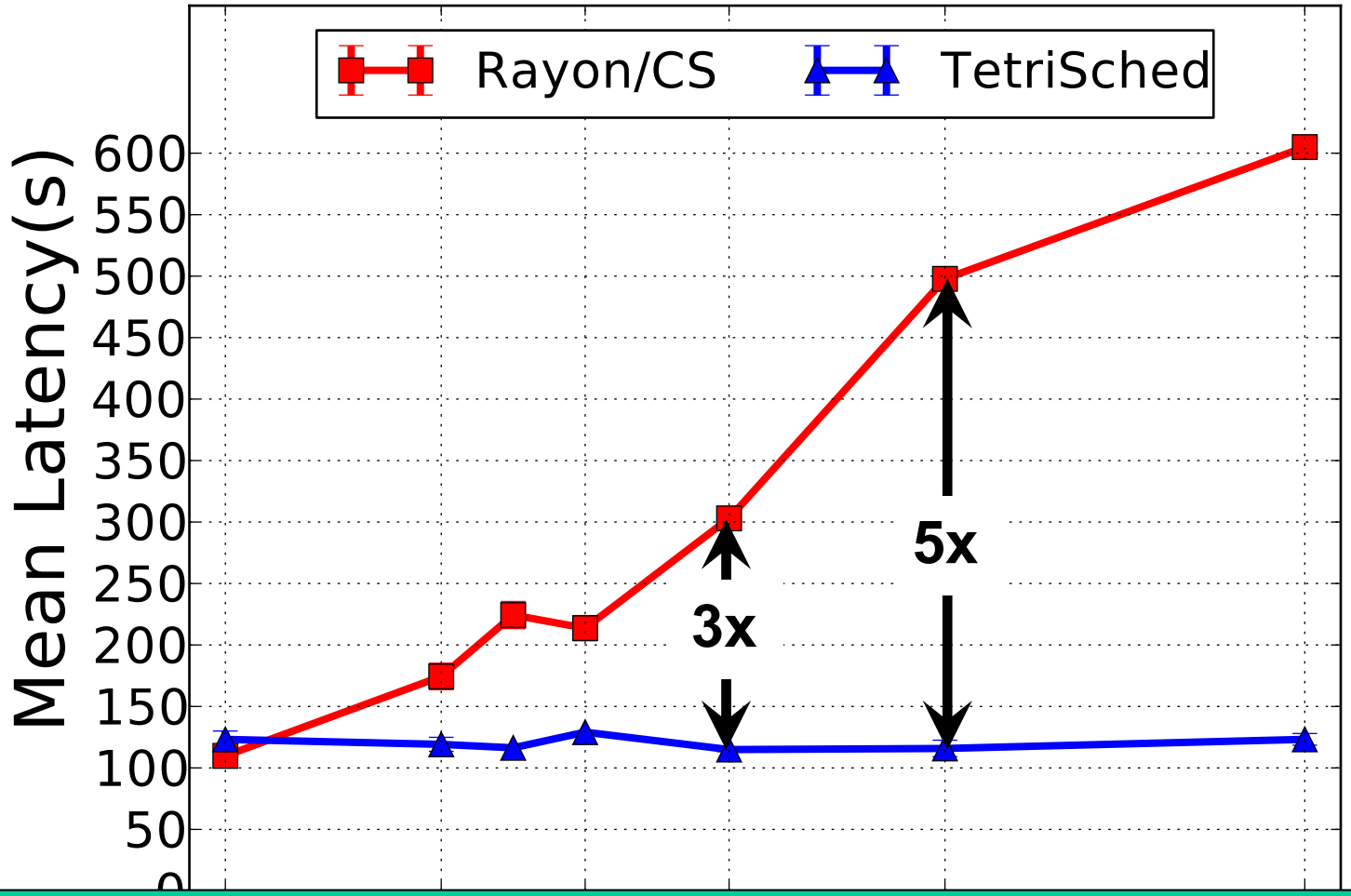



**Achieves high SLO attainment with 2x error**

# Leverage runtime estimates robustly

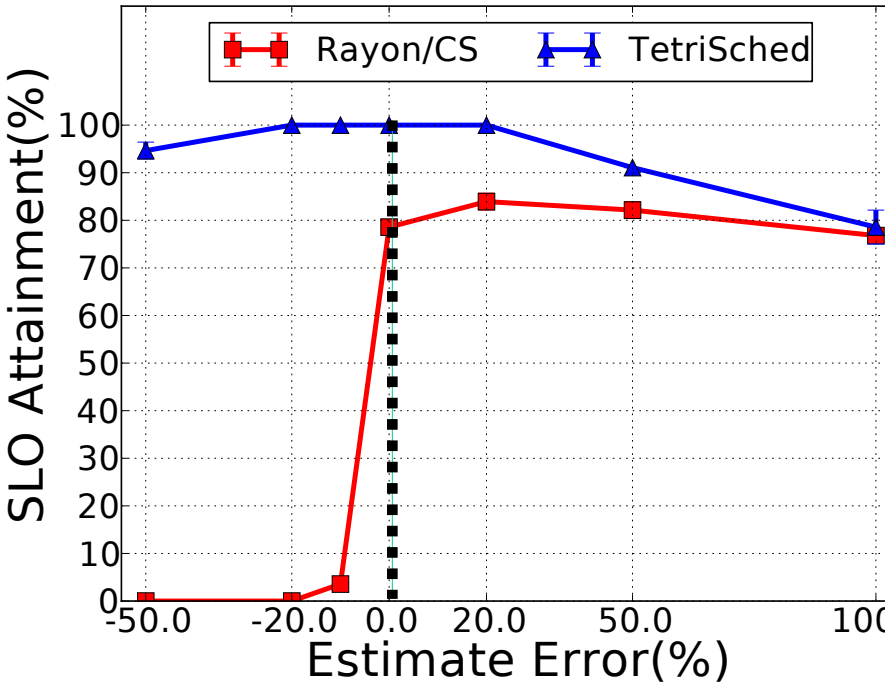


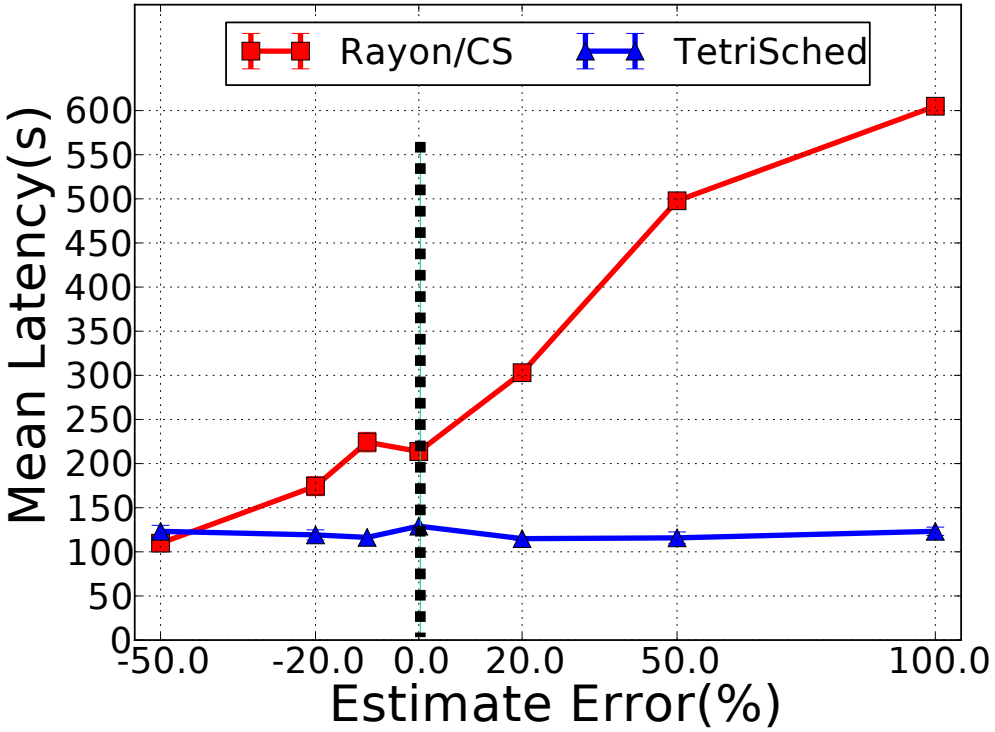
# Leverage runtime estimates robustly




**Achieves lower best-effort latency**

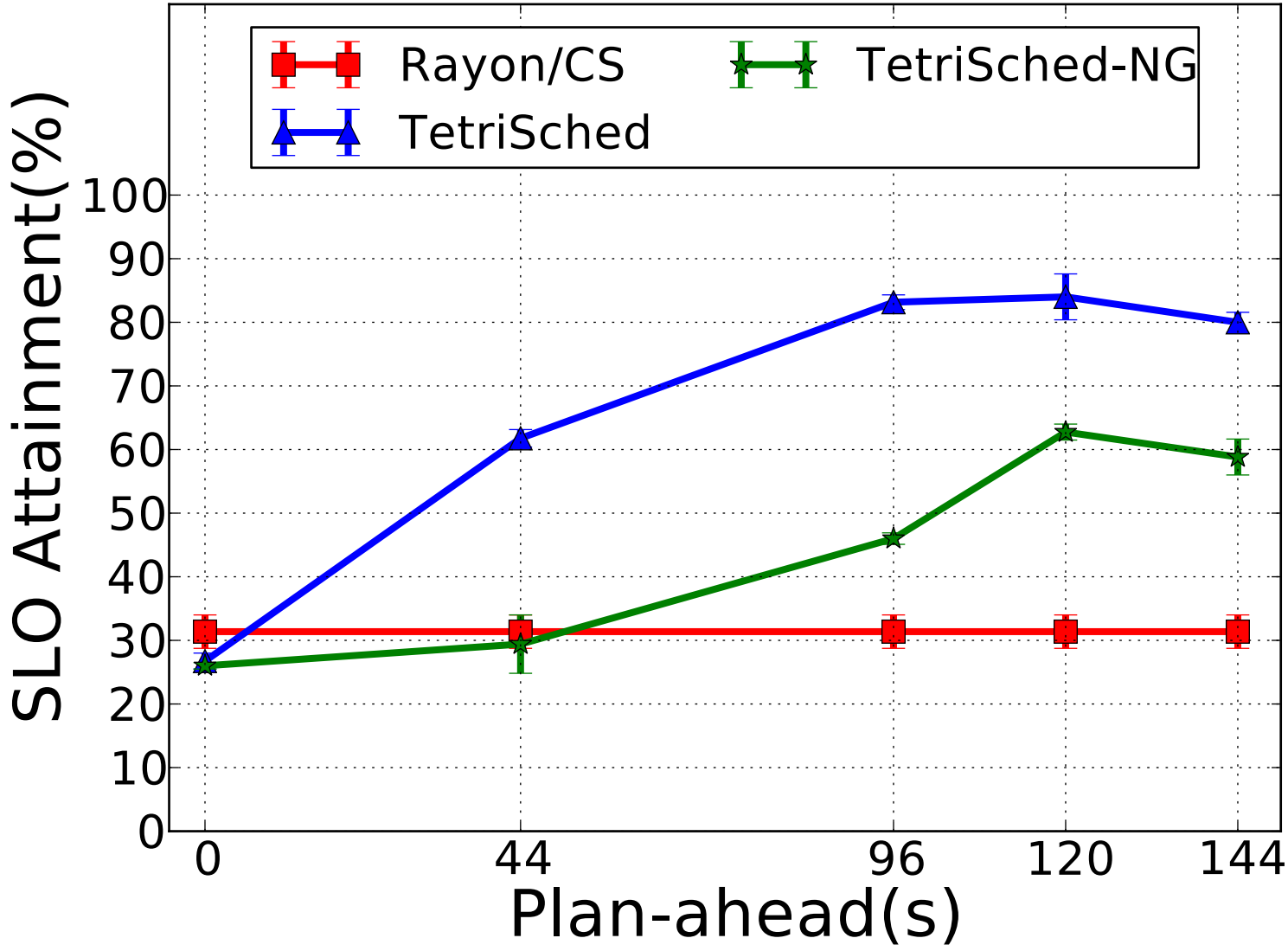
# Leverage runtime estimates robustly



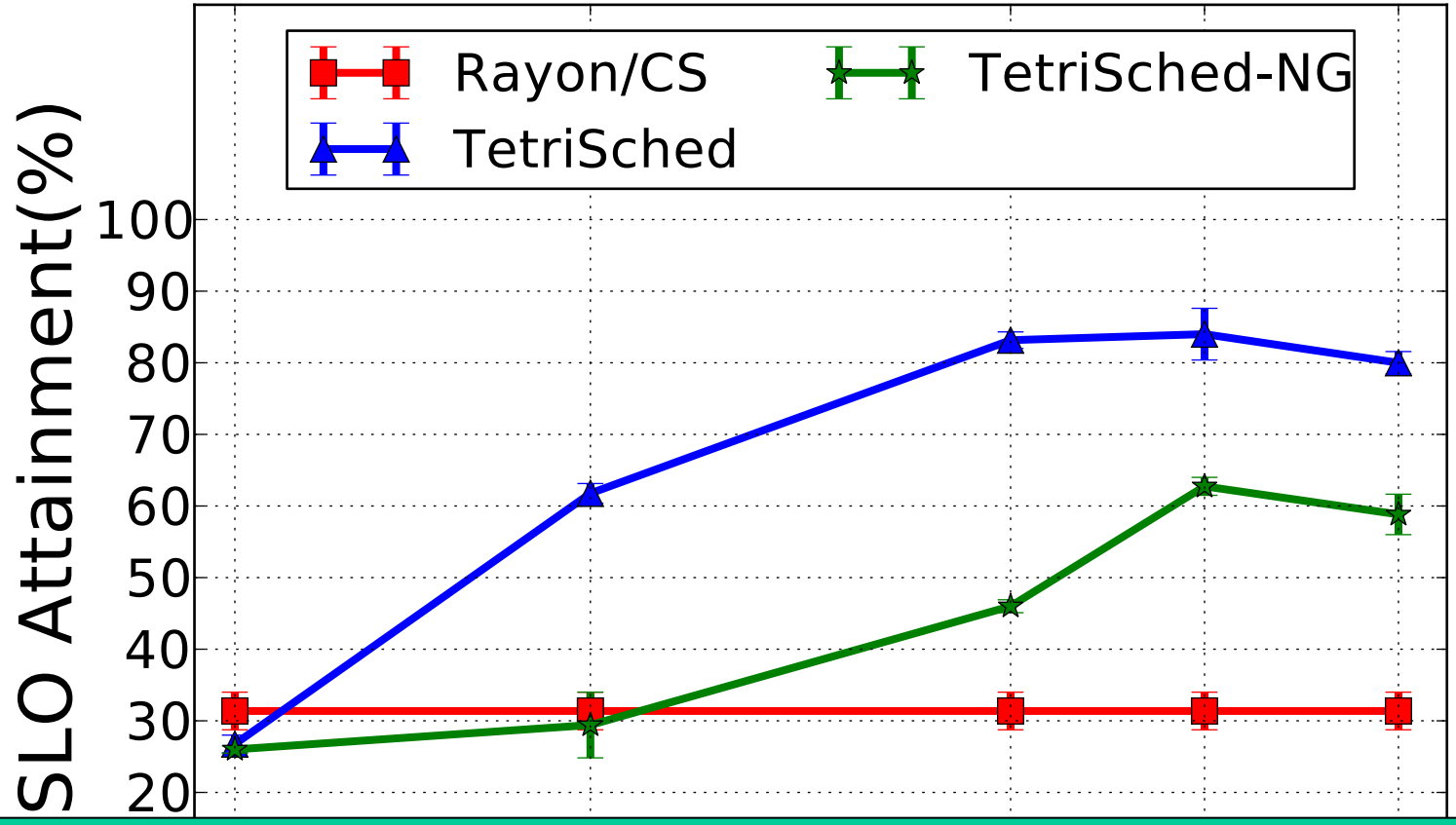




**Exploits runtime estimates better & more robustly**

# Benefit from plan-ahead and global



# Benefit from plan-ahead and global




**Plan-ahead adds 2.5x in SLO attainment**

**Global scheduling further improves perf.**

# Takeaway Results

- Each primary TetriSched feature is needed:
  - Soft constraint support yields 2x over baseline
  - Plan-ahead support yields 2.5x over baseline
  - Global scheduling yields 40% over baseline
- Scales to sizeable clusters
  - 256 node real cluster
  - 1000-node simulated cluster

# Conclusions

- Modern clusters induce complex scheduling tradeoffs
  - ***Scheduling is hard – must schedule harder!***

- *TetriSched*:
  - General support for space-time preferences (STRL)
  - Leveraging runtime estimates robustly (plan-ahead)
  - Global scheduling (STRL+MILP)

- *Key takeaway result*:
  - Significantly higher SLO attainment, lower latency