Scheduling Heterogeneous Resources in Cloud Datacenters

Greg Ganger

Alexey Tumanov, Timothy Zhu,

Mor Harchol-Balter, Mike Kozuch

http://www.istc-cc.cmu.edu/

Intel Science & Technology Center for Cloud Computing

Context: wide variety of workload types

- There are many cluster resource consumers
  - Big Data frameworks, elastic services, VMs, …
  - Number going up, not down: GraphLab, Spark, …

Dryad

Hypertable

Cassandra

MPI

hadoop

Pregel

APACHE HTTP SERVER

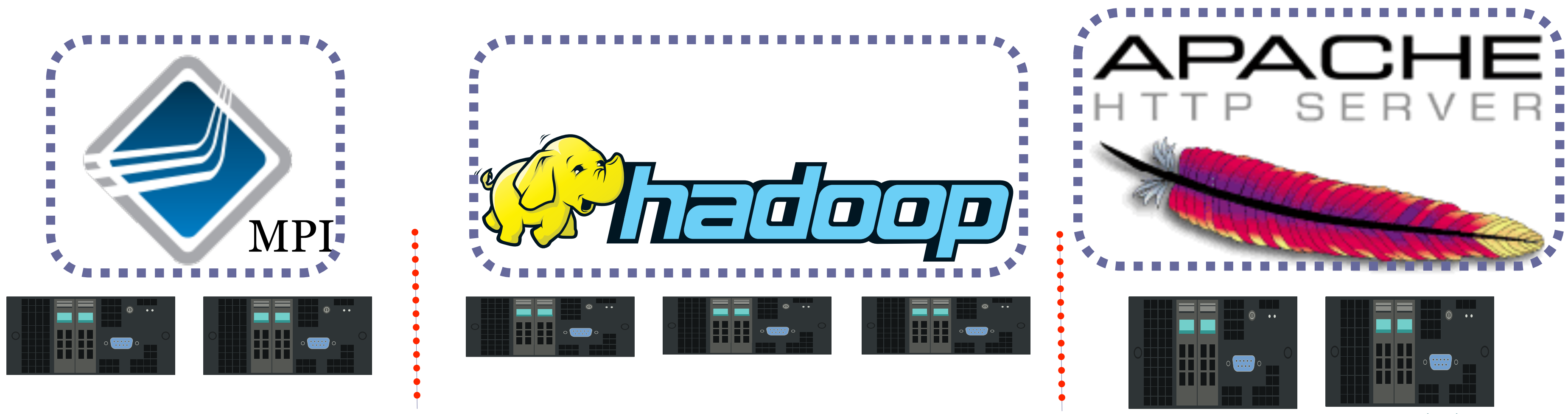
GraphLab

PETUUM

Spark

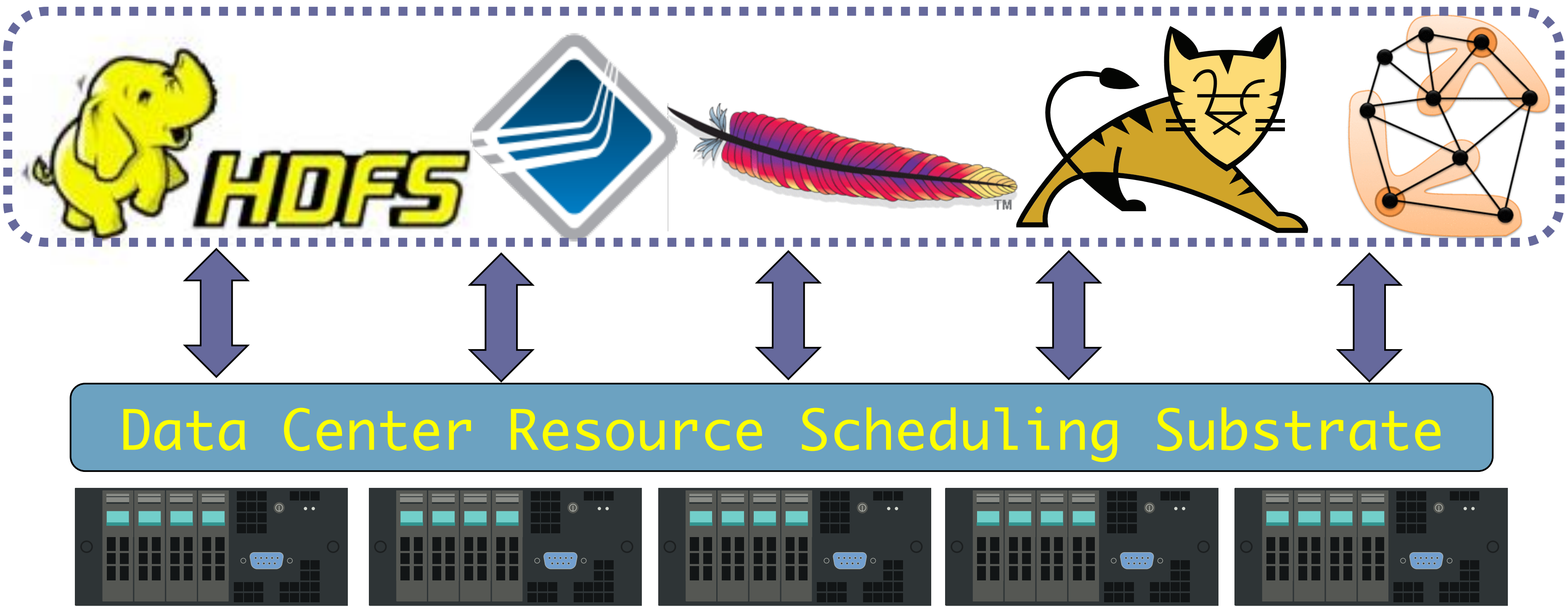
Traditional: separate clusters

- There are many cluster resource consumers
  - Big Data frameworks, elastic services, VMs, …
  - Number going up, not down: GraphLab, Spark, …
- Historically, each would get its own cluster
  - and use its own cluster scheduler
  - and hardware could be specialized = efficiency ↑



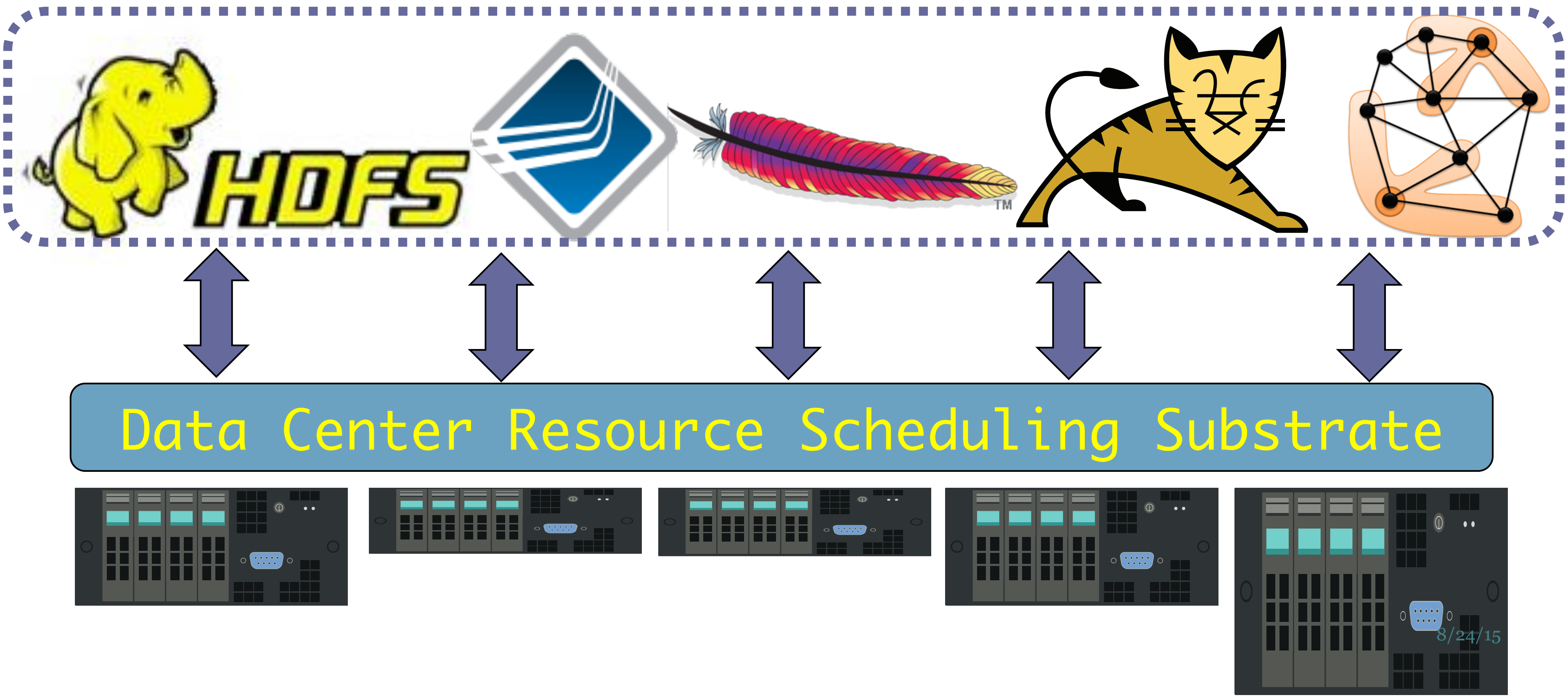
Preferred: dynamic sharing of resources

- Heterogeneous mix of activity types
- Each grabbing/releasing resources dynamically
  - Why? all the standard cloud efficiency story-lines

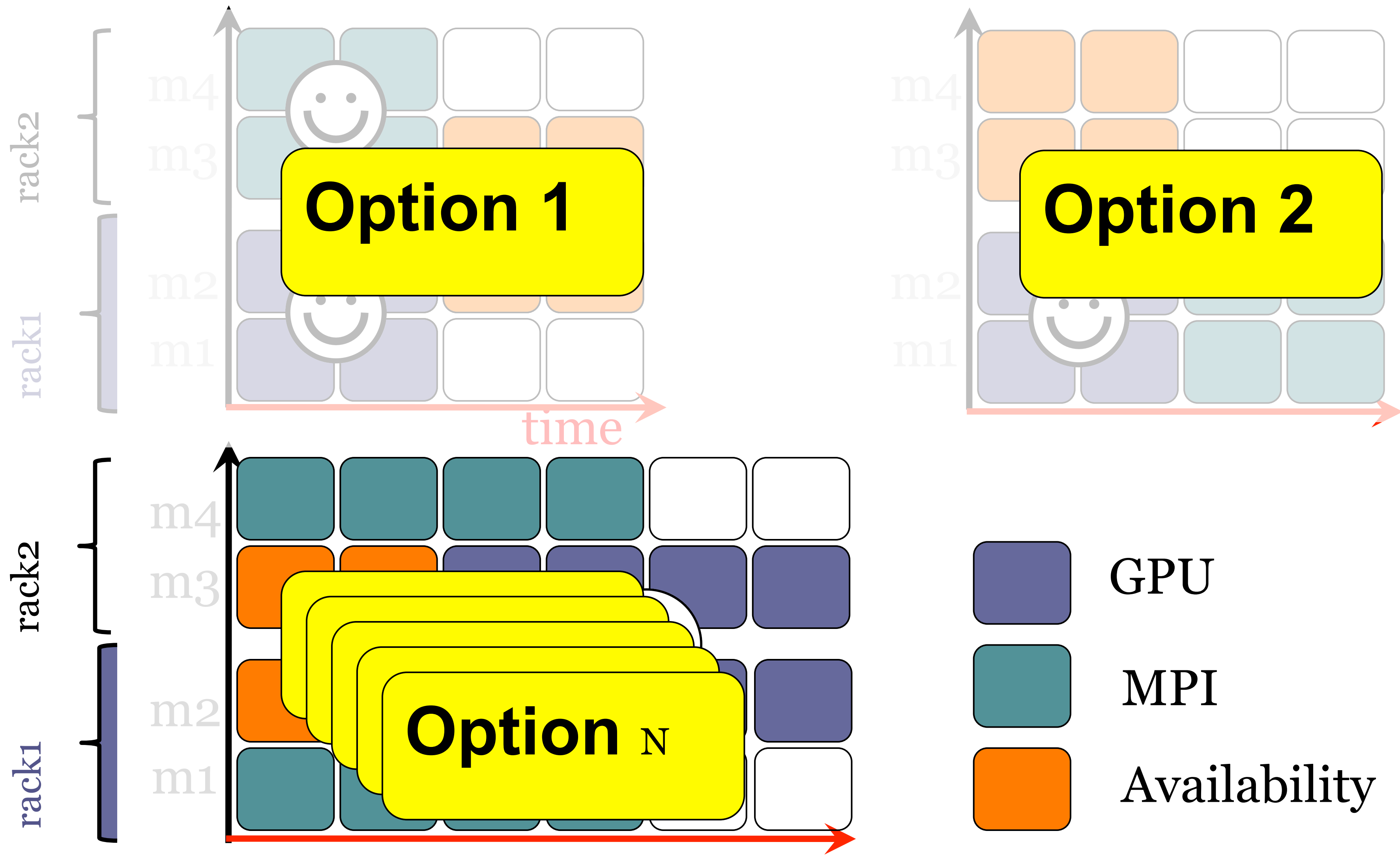


And, diverse specialized servers

- Have a mix of platform types, purposefully
  - Providing a mix of capabilities and features
  - Then, match work to platform during scheduling
    - goal: assign right work to right place at right time

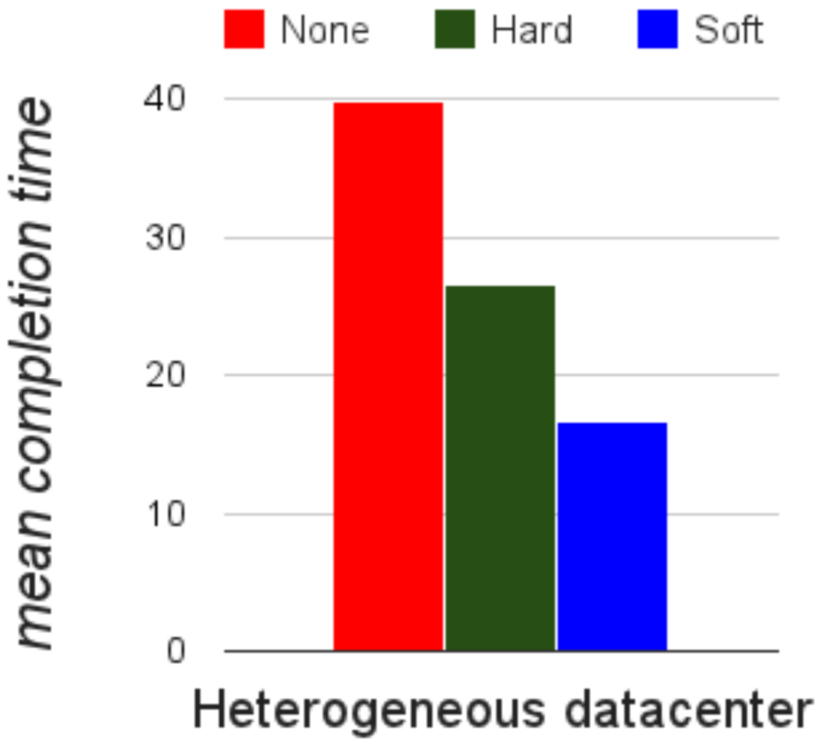


Challenge: Explosion of Options + Tradeoffs



Need to exploit per-job flexibility

- Problem: most schedulers don't
  - usually, preferred option treated as only option
  - a few (Mesos) expose choice, but don't control it
- But, large benefits to doing so
  - better for resource usage AND application service



What do we need to do it

- Informed exploitation of flexibility needs ability to
  - *Quantify* tradeoffs among acceptable options
  - *Express* options and tradeoffs (concisely)
  - *Exploit* this knowledge to improve resource assignments

  - … all in a practical manner

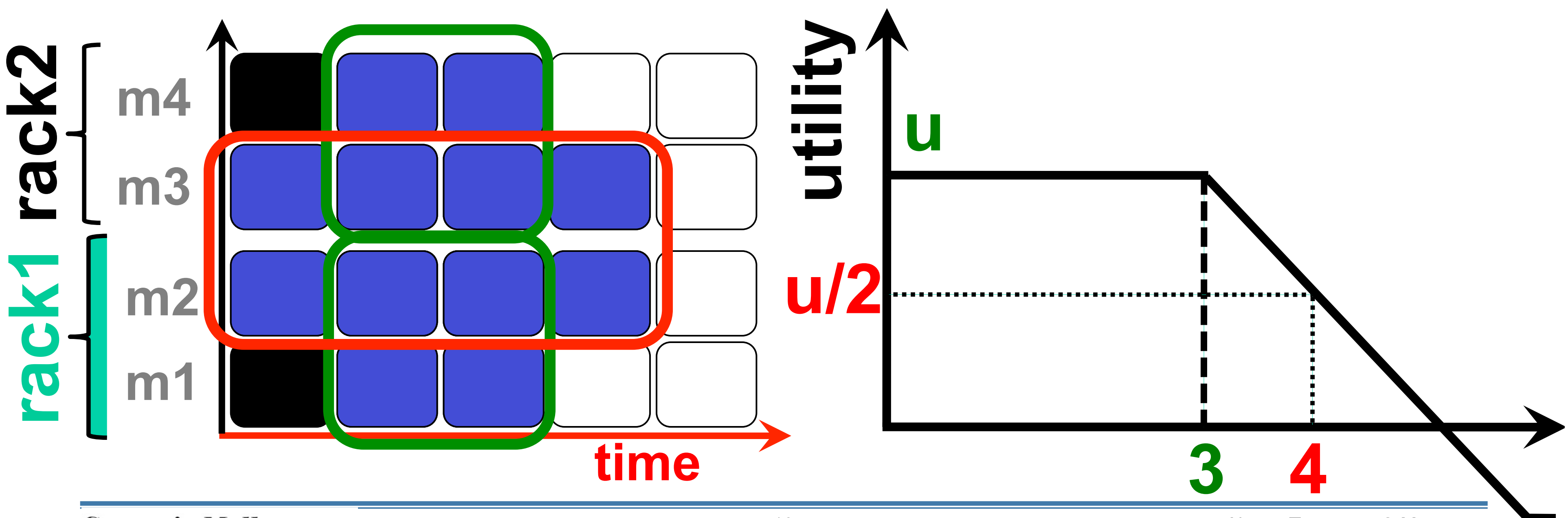
STRL Generator: quantify -> express

- Translates high-level objectives to STRL (our language)
- Adapts to new forms of heterogeneity

Space-Time Request Language

- Utility u(p,t): placement p@t → utility u

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



STRL Expression Composition

- Utility u(p,t): placement p@t → utility u

max

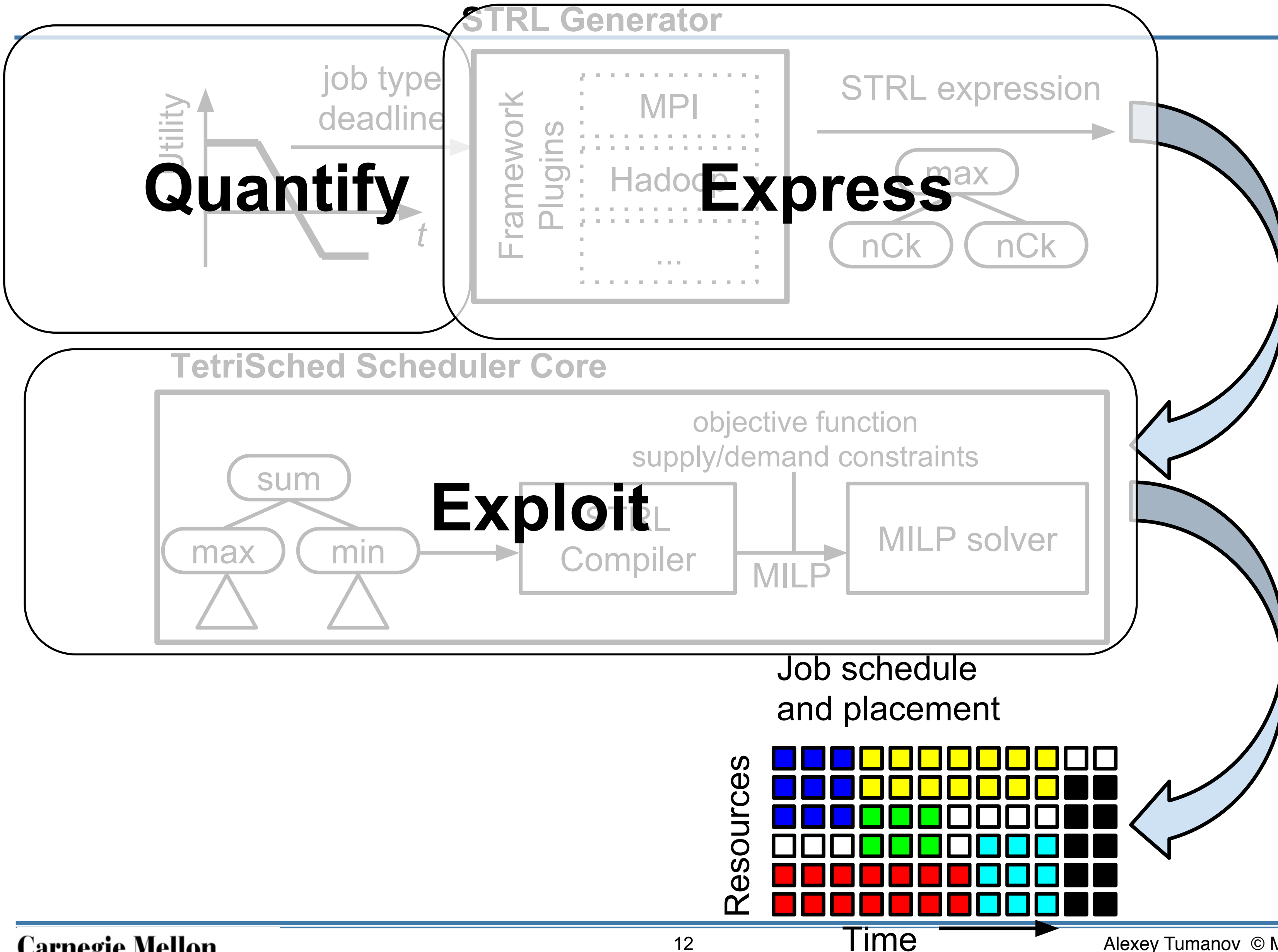
OR

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

rack2: nCk ($m_i \in \text{rack2}$, k=2, s=1, d=2, u)

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

TetriSched



STRL Generator
Quantify
Utility
job type
deadline
t
Framework
Plugins
MPI
Hadoop
...
Express
STRL expression
max
nCk
nCk
TetriSched Scheduler Core
Exploit
objective function
supply/demand constraints
sum
max
min
STRL
Compiler
MILP
MILP solver
Job schedule
and placement
Resources
Time

Latest efforts

- Pushing for wide external use
  - working with Apache Hadoop YARN committers
  - incremental pushing of the concepts, over summer/fall
- Integrating with resource reservation (Rayon)
  - toward heterogeneity-aware resource reservation
  - allow exploiting flexibility in space and time
- Scalability characterization and enhancement
  - e.g., heuristics in place of full MILP optimization
  - e.g., separate best-eff short jobs from demanding ones

Takeaways

- Problem: current schedulers don't cope with
  - increased heterogeneity in datacenters
  - explosion of tradeoffs and choices
- Solution: Tetrisched ☺
  - exploits concisely expressed options and tradeoffs
- End result:
  - better schedules of heterogeneous mixes
  - easier adoption of specialized hardware
- Current steps:
  - integrating into mainline YARN (Apache Hadoop)
  - enhancing scalability and coupling with reservations