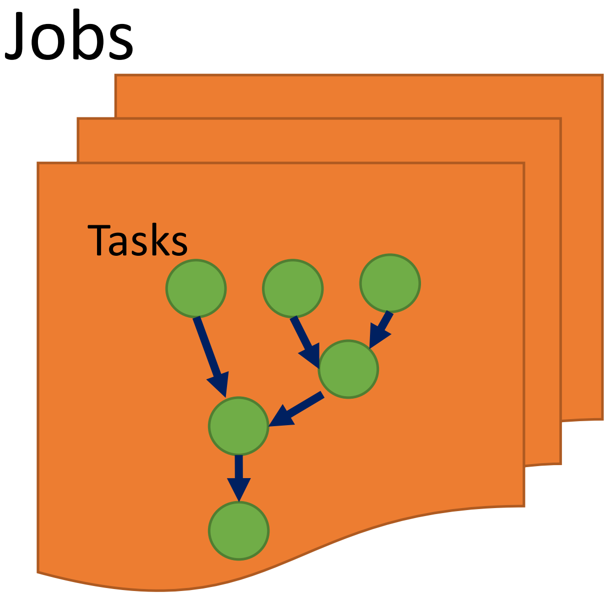
Packing Tasks with Dependencies

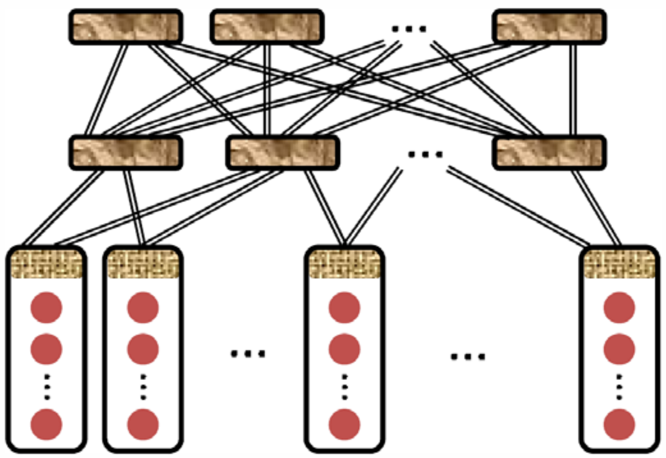
Robert Grandl, Srikanth Kandula,
Sriram Rao, Aditya Akella, Janardhan Kulkarni

Microsoft
WISCONSIN
UNIVERSITY OF WISCONSIN–MADISON

The Cluster Scheduling Problem

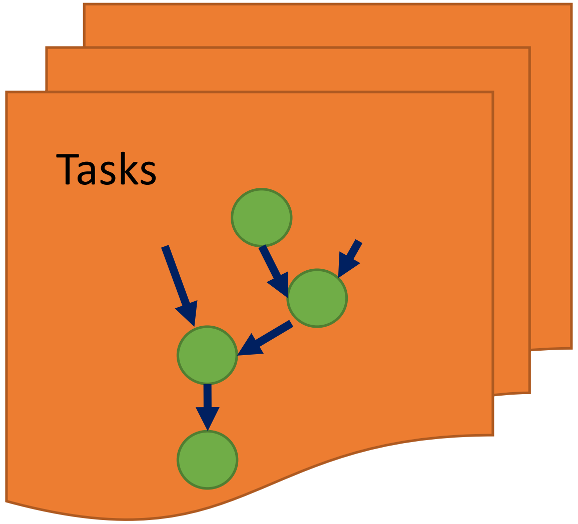


Goal: match tasks to resources

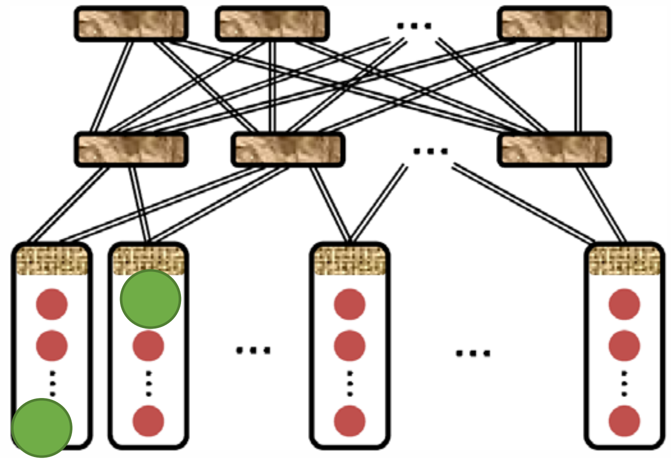


The Cluster Scheduling Problem

Jobs

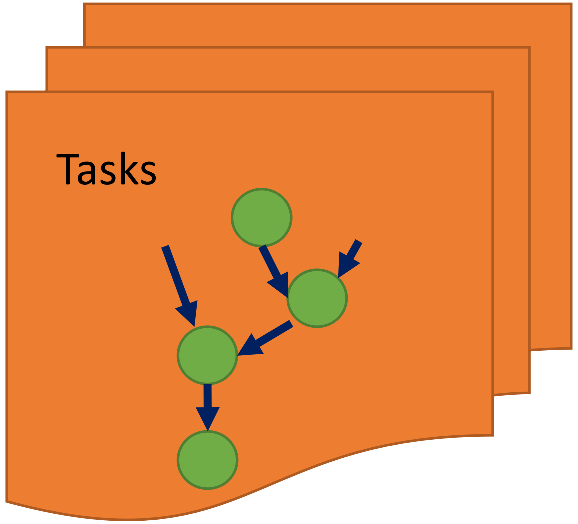


Goal: match tasks to resources



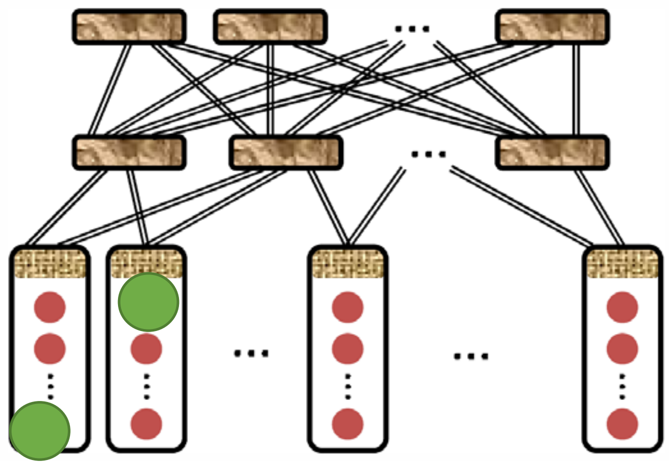
The Cluster Scheduling Problem

Jobs

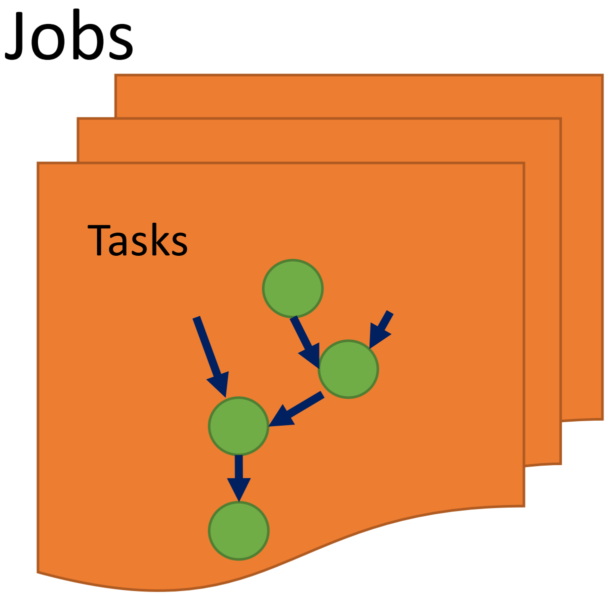


Goal: match tasks to resources
to achieve

- High cluster utilization
- Fast job completion
- Guarantees (deadlines, fair shares)

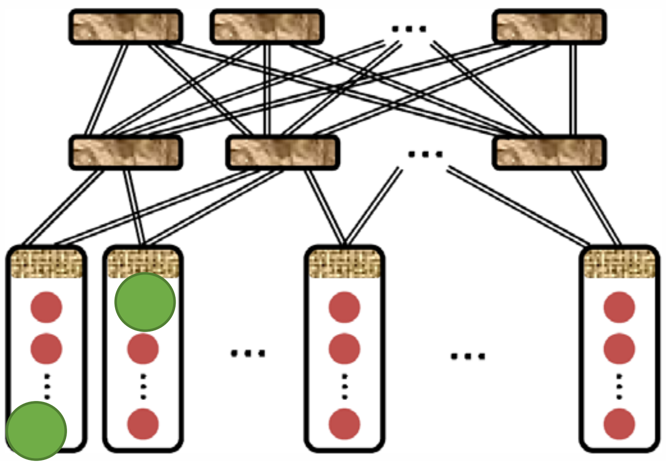


The Cluster Scheduling Problem



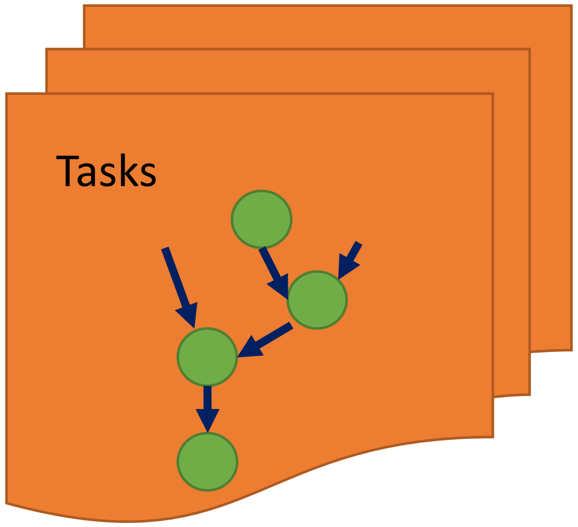
Goal: match tasks to resources to achieve
- High cluster utilization
- Fast job completion
- Guarantees (deadlines, fair shares)

Constraints
- Scale $\Rightarrow$ fast twitch



The Cluster Scheduling Problem

Jobs

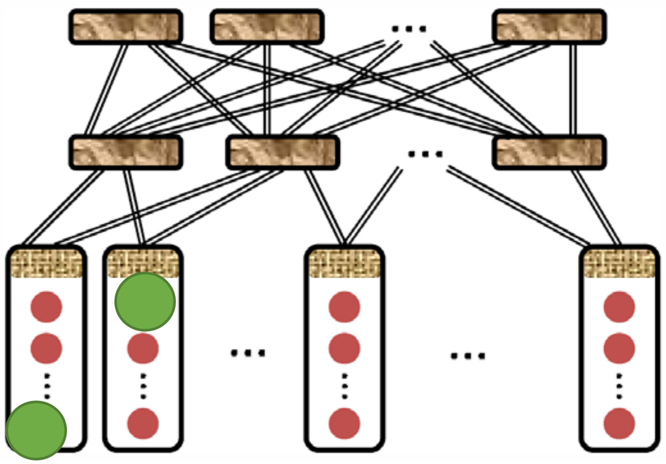


Goal: match tasks to resources to achieve

- High cluster utilization
- Fast job completion
- Guarantees (deadlines, fair shares)

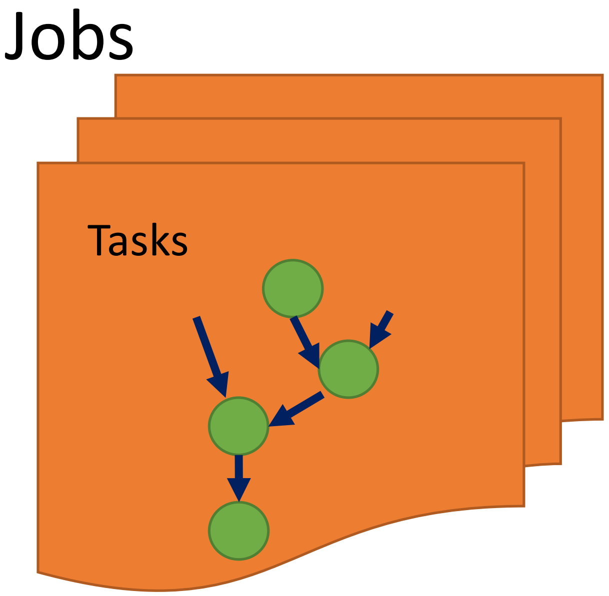
Constraints

- Scale $\Rightarrow$ fast twitch



- Large and high-value deployments
  - E.g., Spark, Yarn*, Mesos*, Cosmos

The Cluster Scheduling Problem

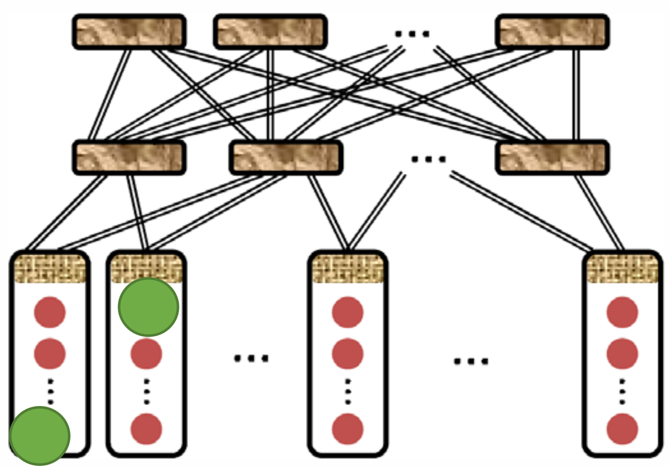


Goal: match tasks to resources to achieve

- High cluster utilization
- Fast job completion
- Guarantees (deadlines, fair shares)

Constraints

- Scale $\Rightarrow$ fast twitch



- Large and high-value deployments
  - E.g., Spark, Yarn*, Mesos*, Cosmos
- Today, schedulers are simple and (as we show) performance can improve a lot

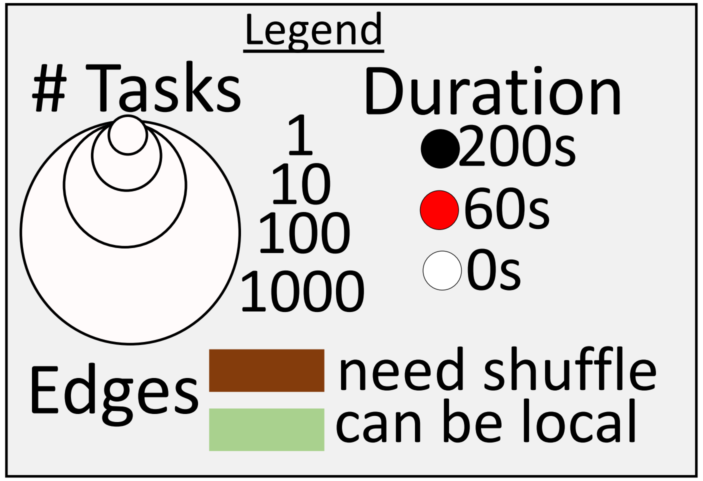
Jobs have heterogeneous **DAGs**

User queries $\rightarrow$ Query optimizer $\rightarrow$ Job DAG

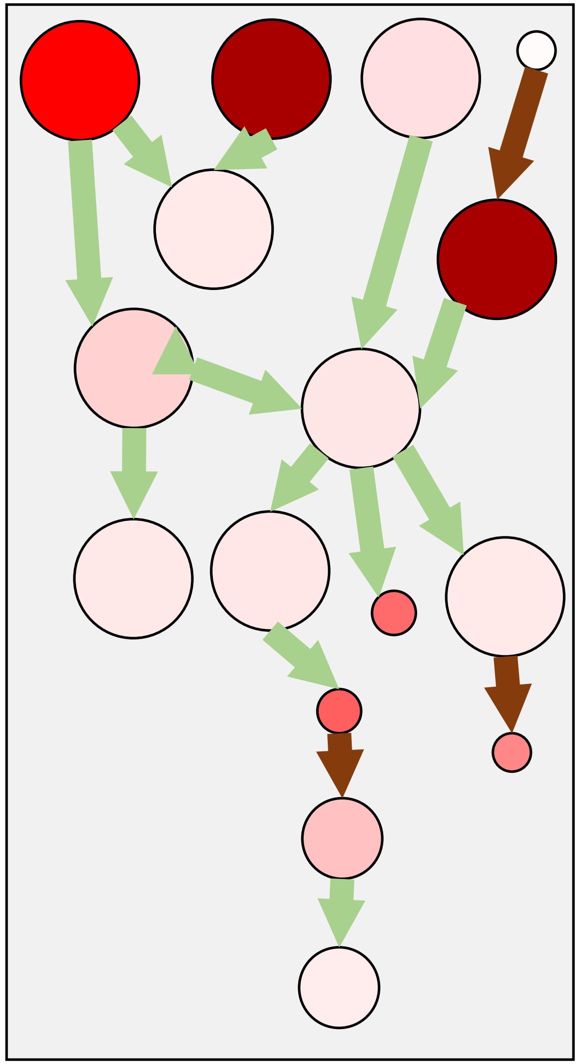
(Dryad, Spark-SQL, Hive,…)

Jobs have heterogeneous DAGs

User queries → Query optimizer → Job DAG
(Dryad, Spark-SQL, Hive,...)



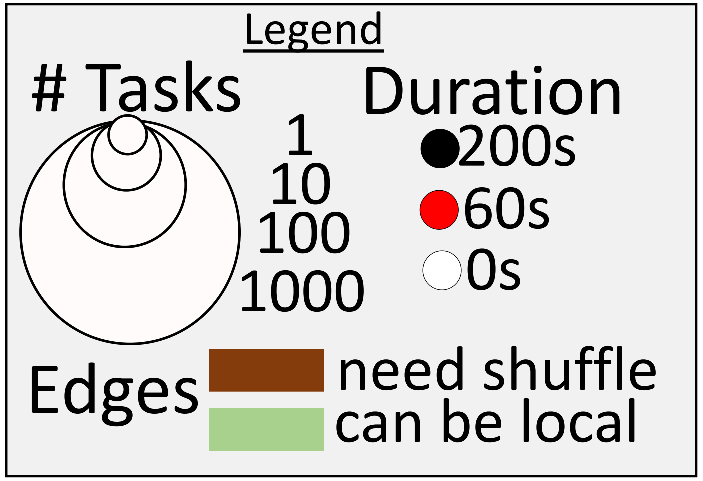
Example DAG



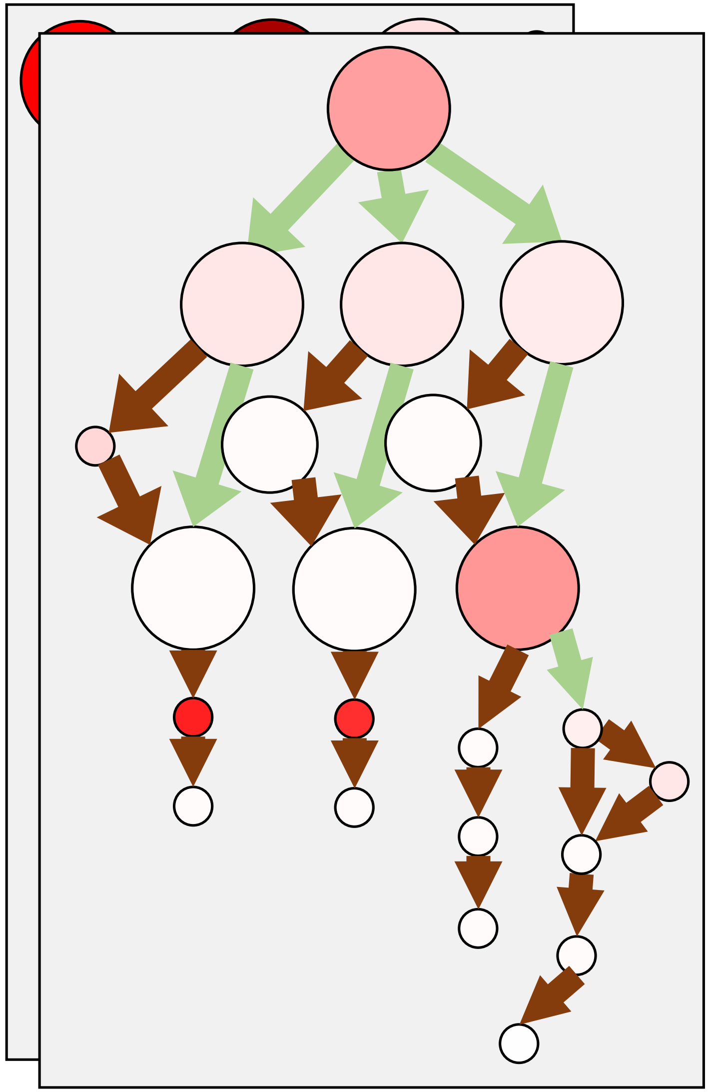
Jobs have heterogeneous DAGs

User queries → Query optimizer → Job DAG

(Dryad, Spark-SQL, Hive,…)



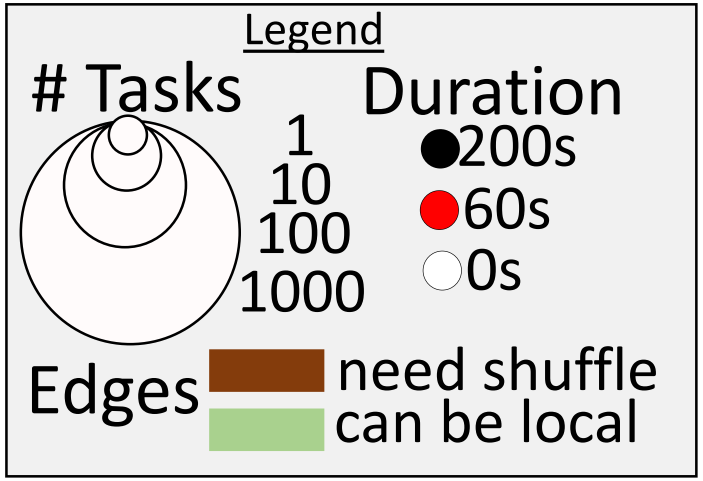
Example DAG



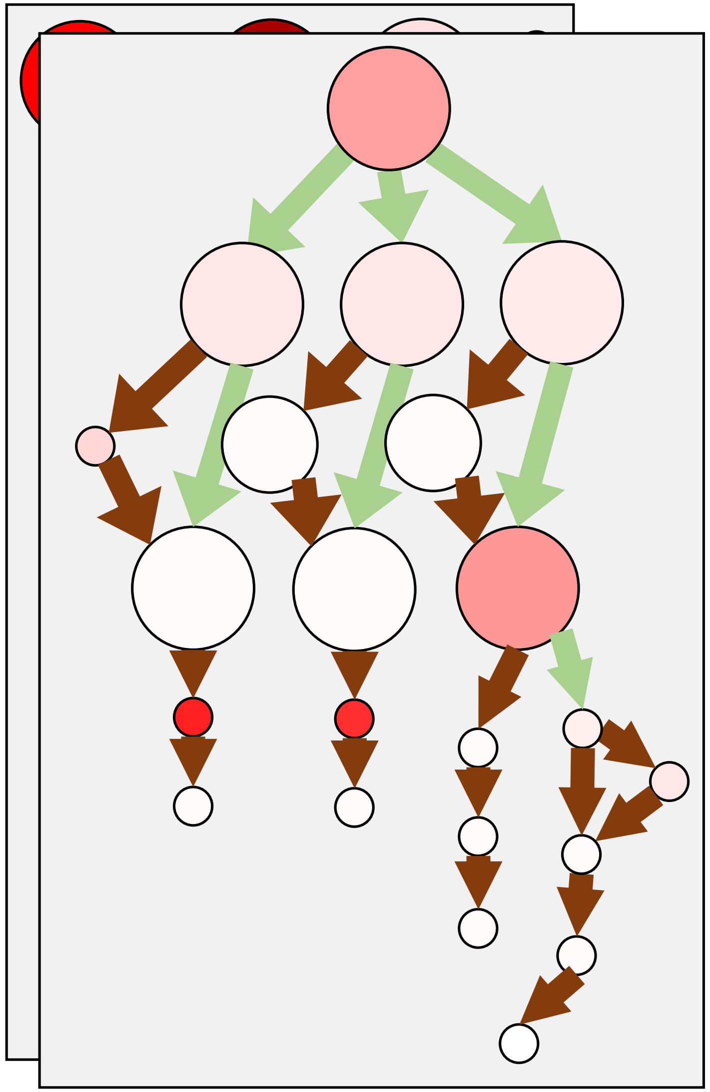
Jobs have heterogeneous DAGs

User queries → Query optimizer → Job DAG
(Dryad, Spark-SQL, Hive,…)

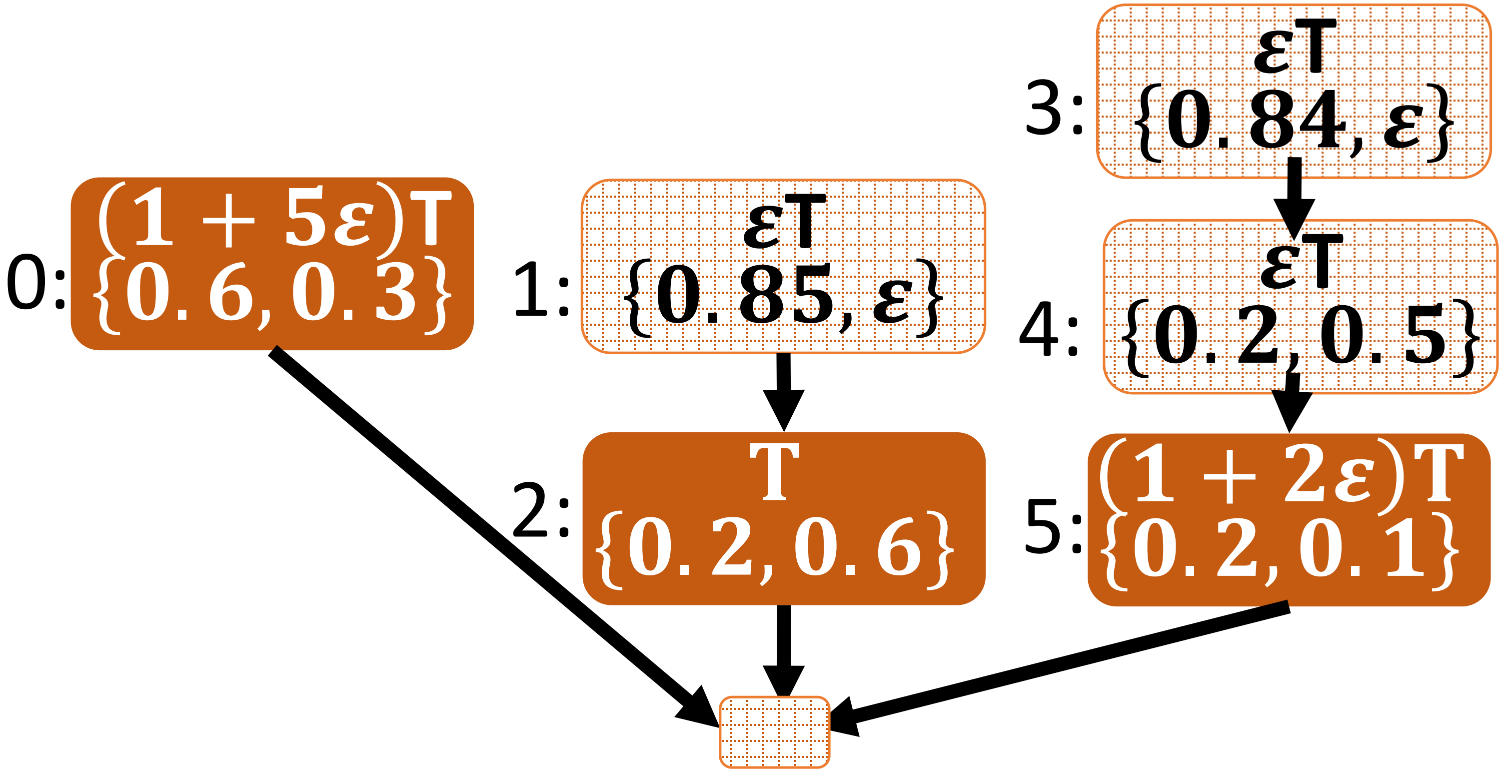
- DAGs have deep and complex structures
- Task durations range from <1s to >100s
- Tasks use different amounts of resources



Example DAG

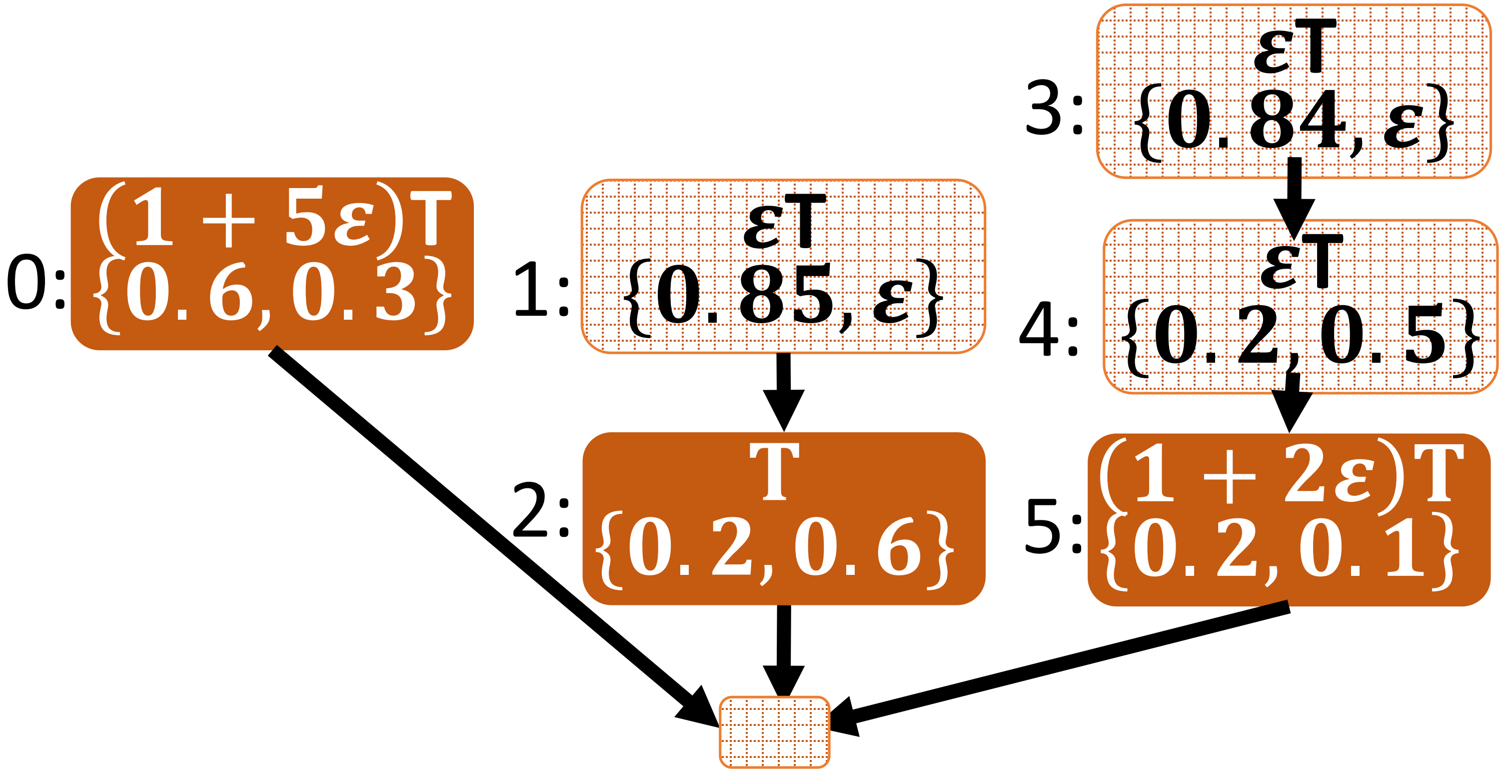


Challenges in scheduling heterogeneous DAGs



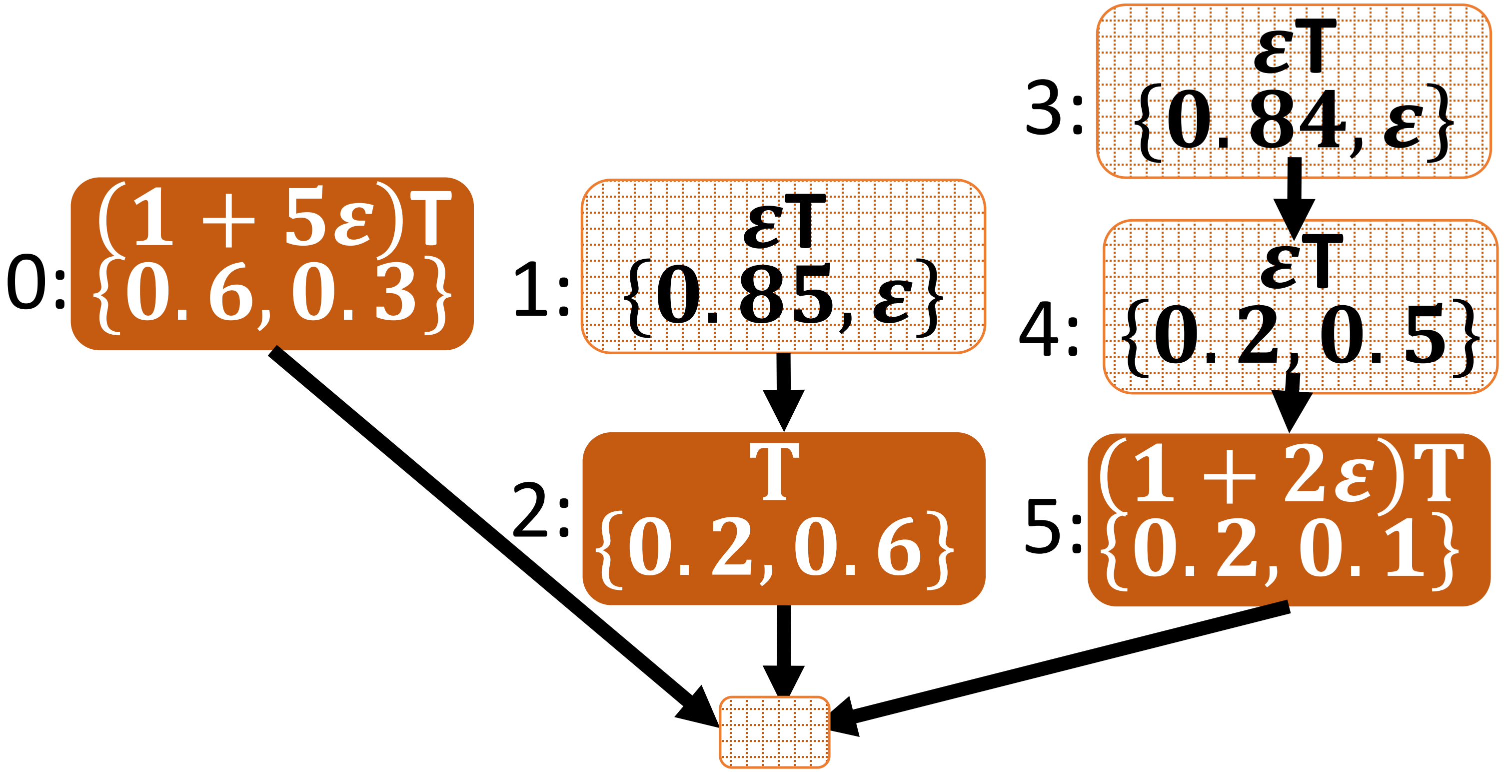
Challenges in scheduling heterogeneous DAGs

Technique	**Execution Order**	**Time**
OPT	$t_1 \rightarrow t_3 \rightarrow \{t_4, t_0\} \rightarrow \{t_0, t_2, t_5\}$	T



Challenges in scheduling heterogeneous DAGs

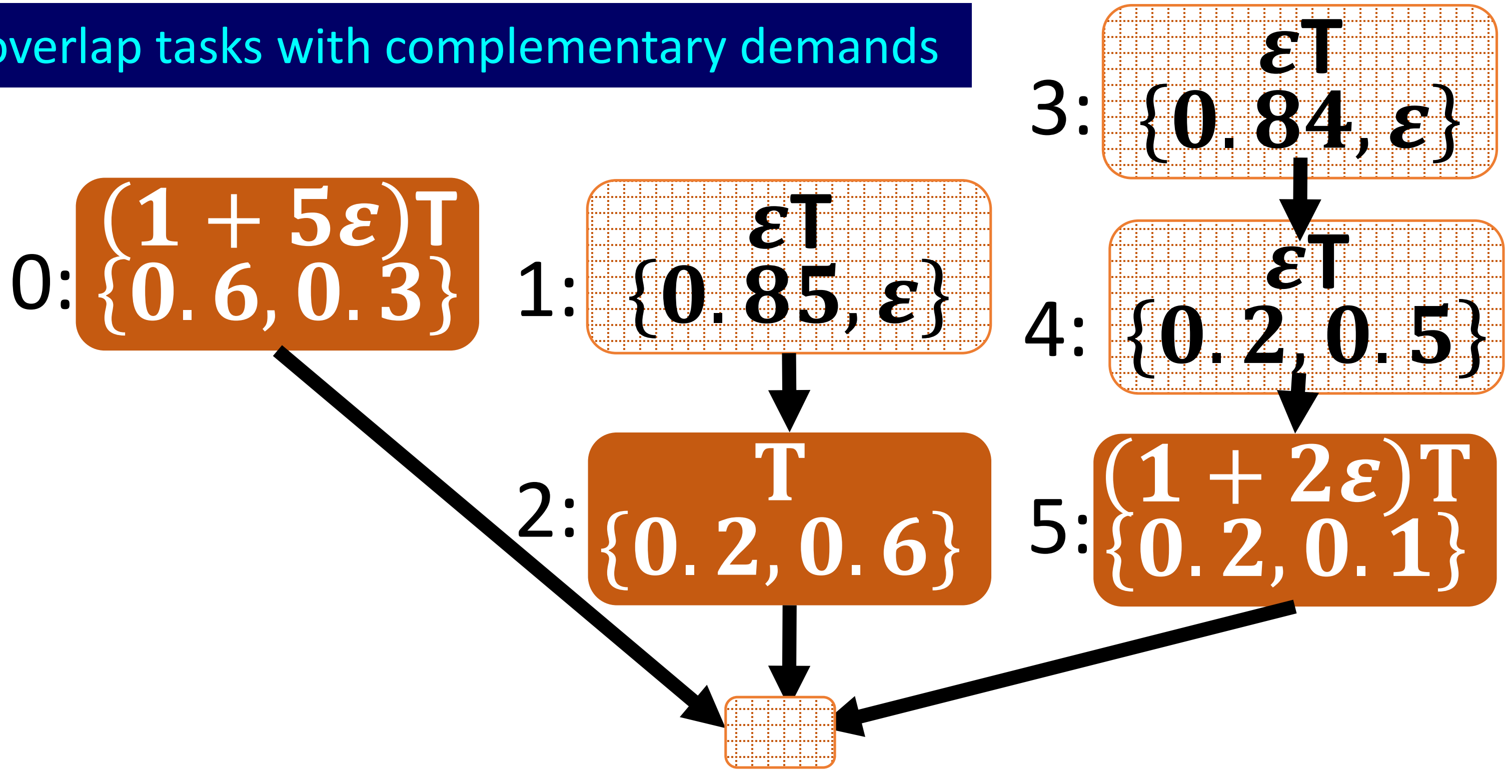
Technique	Execution Order	Time
OPT	$t_1 \to t_3 \to \{t_4, t_0\} \to \{t_0, t_2, t_5\}$	T
CPSched	$t_0 \to t_3 \to t_4 \to t_5 \to t_1 \to t_2$	$3T$



Challenges in scheduling heterogeneous DAGs

Technique	Execution Order	Time
OPT	$t_1 \rightarrow t_3 \rightarrow \{t_4, t_0\} \rightarrow \{t_0, t_2, t_5\}$	T
CPSched	$t_0 \rightarrow t_3 \rightarrow t_4 \rightarrow t_5 \rightarrow t_1 \rightarrow t_2$	$3T$

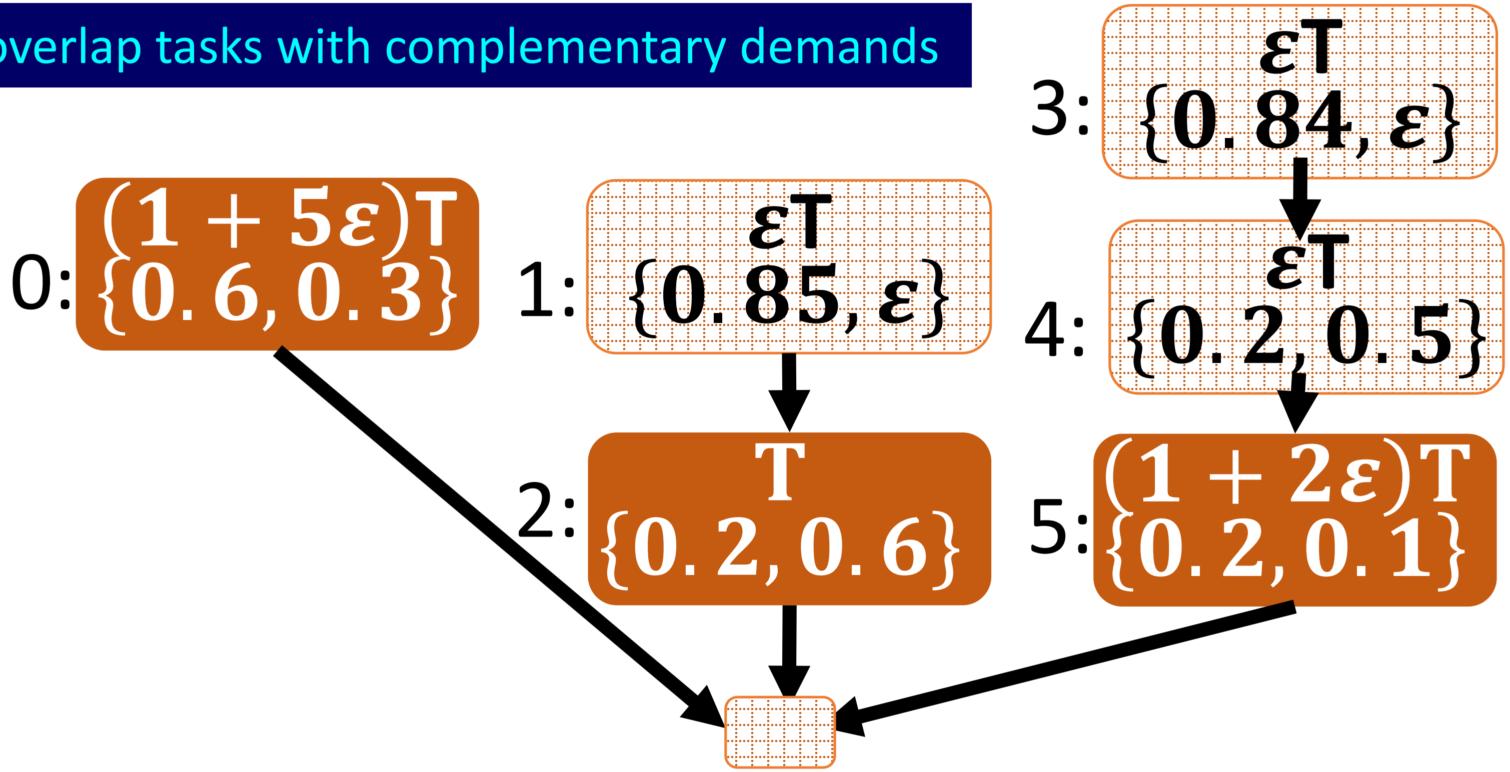
CPSched cannot overlap tasks with complementary demands



Challenges in scheduling heterogeneous DAGs

Technique	Execution Order	Time
OPT	$t_1 \rightarrow t_3 \rightarrow \{t_4, t_0\} \rightarrow \{t_0, t_2, t_5\}$	T
CPSched	$t_0 \rightarrow t_3 \rightarrow t_4 \rightarrow t_5 \rightarrow t_1 \rightarrow t_2$	$3T$
Packers[1]	$t_0 \rightarrow t_1 \rightarrow t_3 \rightarrow t_2 \rightarrow t_4 \rightarrow t_5$	$3T$

CPSched cannot overlap tasks with complementary demands



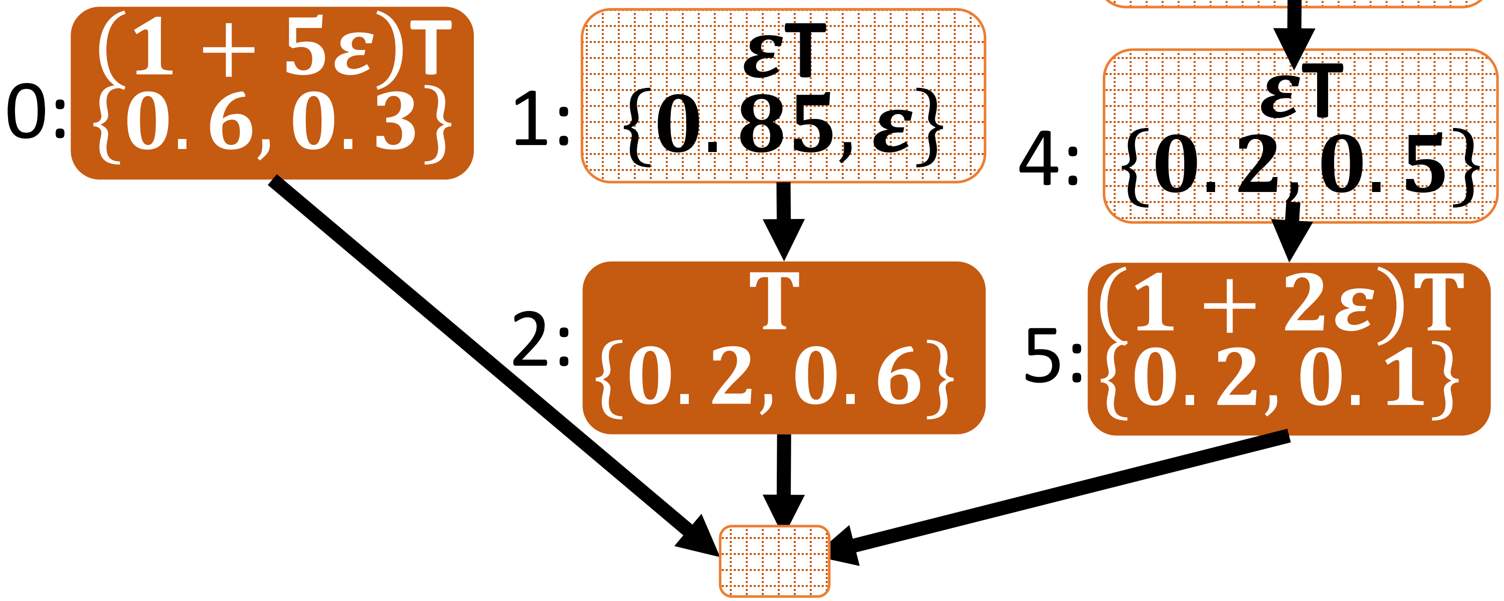
[1] Tetris: Multi-resource Packing for Cluster Schedulers, SIGCOMM'14

Challenges in scheduling heterogeneous DAGs

Technique	Execution Order	Time
OPT	$t_1 \rightarrow t_3 \rightarrow \{t_4, t_0\} \rightarrow \{t_0, t_2, t_5\}$	T
CPSched	$t_0 \rightarrow t_3 \rightarrow t_4 \rightarrow t_5 \rightarrow t_1 \rightarrow t_2$	$3T$
Packers[1]	$t_0 \rightarrow t_1 \rightarrow t_3 \rightarrow t_2 \rightarrow t_4 \rightarrow t_5$	$3T$

CPSched cannot overlap tasks with complementary demands

Packers do not handle dependencies



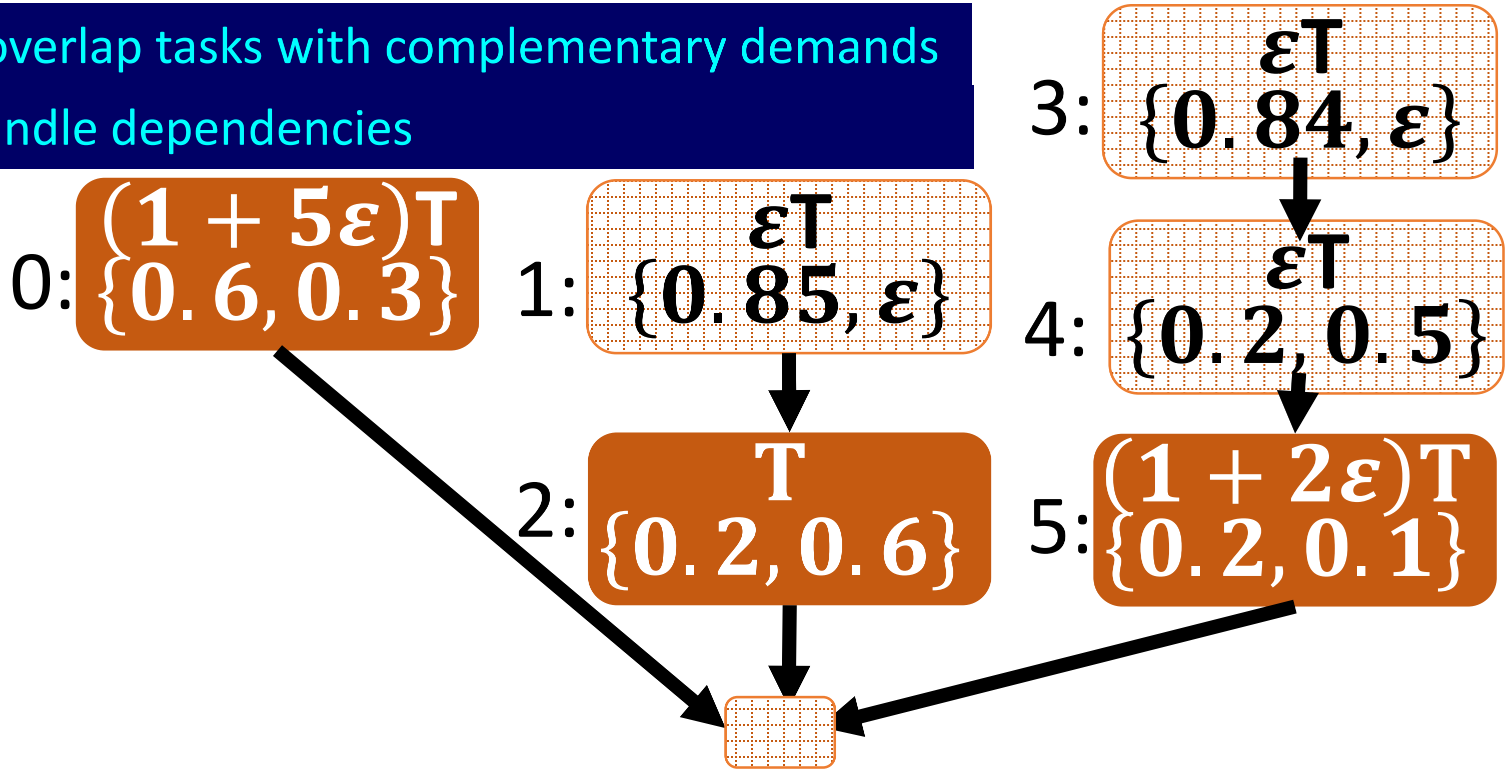
[1] Tetris: Multi-resource Packing for Cluster Schedulers, SIGCOMM'14

Challenges in scheduling heterogeneous DAGs

Technique	Execution Order	Time	Worst-case
OPT	$t_1 \to t_3 \to \{t_4, t_0\} \to \{t_0, t_2, t_5\}$	T	–
CPSched	$t_0 \to t_3 \to t_4 \to t_5 \to t_1 \to t_2$	$3T$	$O(n) \times$ OPT n tasks
Packers[1]	$t_0 \to t_1 \to t_3 \to t_2 \to t_4 \to t_5$	$3T$	$O(d) \times$ OPT d resources

CPSched cannot overlap tasks with complementary demands

Packers do not handle dependencies



[1] Tetris: Multi-resource Packing for Cluster Schedulers, SIGCOMM'14

Challenges in scheduling heterogeneous DAGs …

Challenges in scheduling heterogeneous DAGs …

1. Simple heuristics lead to poor schedules

Challenges in scheduling heterogeneous DAGs …

1. Simple heuristics lead to poor schedules
2. Production DAGs are roughly 50% slower than lower bounds

Challenges in scheduling heterogeneous DAGs …

1. Simple heuristics lead to poor schedules
2. Production DAGs are roughly 50% slower than lower bounds
3. Simple variants of “Packing dependent tasks” are NP-hard problems

Challenges in scheduling heterogeneous DAGs …

1. Simple heuristics lead to poor schedules
2. Production DAGs are roughly 50% slower than lower bounds
3. Simple variants of “Packing dependent tasks” are NP-hard problems
4. Prior analytical solutions miss some practical concerns

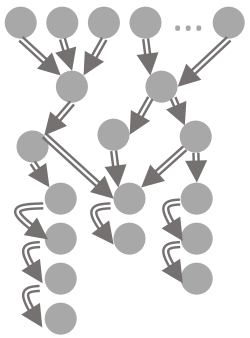
Challenges in scheduling heterogeneous DAGs …

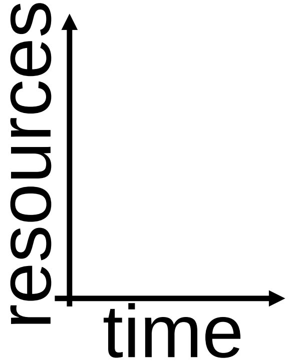
1. Simple heuristics lead to poor schedules
2. Production DAGs are roughly 50% slower than lower bounds
3. Simple variants of “Packing dependent tasks” are NP-hard problems
4. Prior analytical solutions miss some practical concerns
    - Multiple resources
    - Complex dependencies
    - Machine-level fragmentation
    - Scale; Online; …

Given an annotated DAG
and available resources,
compute a good schedule

+ practical model

Main ideas for one DAG

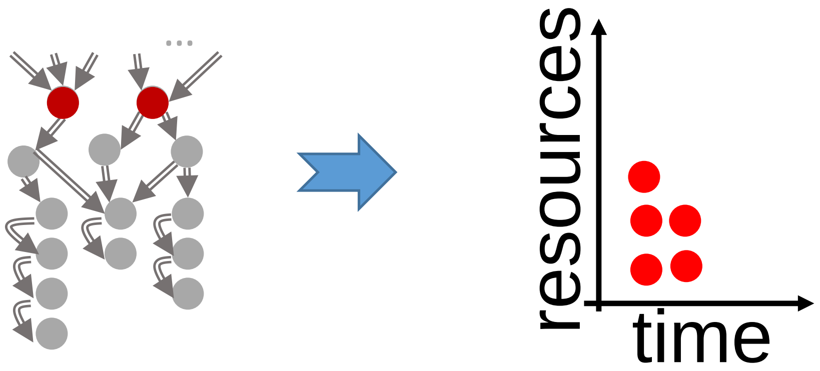




Existing schedulers:

A task is schedulable *after* all its parents have *finished*

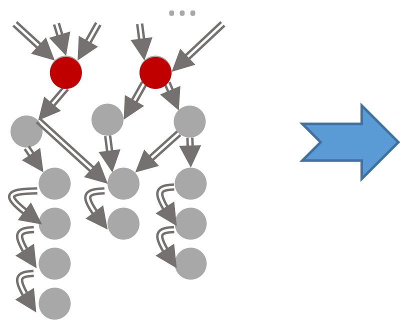
Main ideas for one DAG

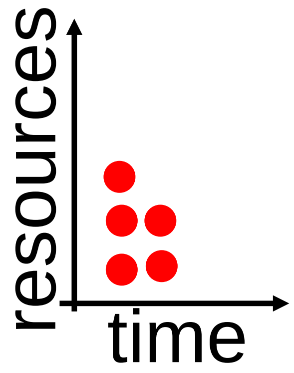


Existing schedulers:

A task is schedulable *after* all its parents have *finished*

Main ideas for one DAG





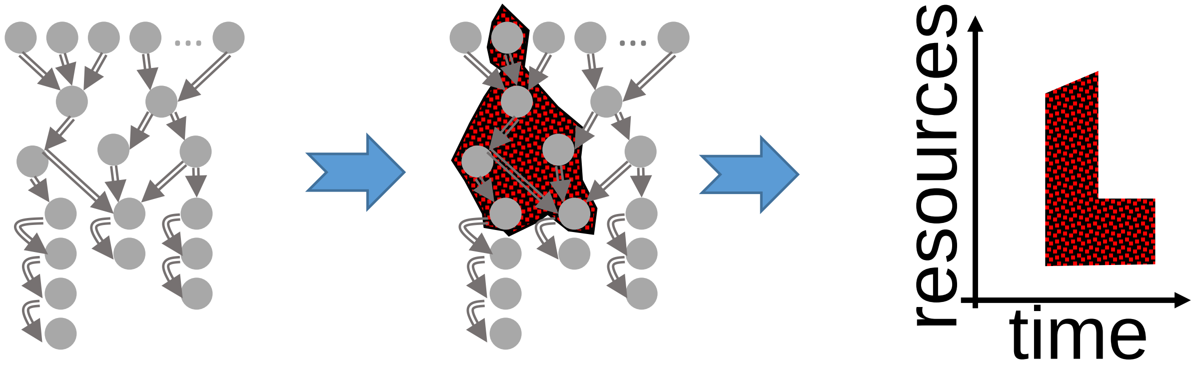
Existing schedulers:

A task is schedulable *after* all its parents have *finished*

Graphene:

Identifies *troublesome tasks* and places them *first*

Main ideas for one DAG



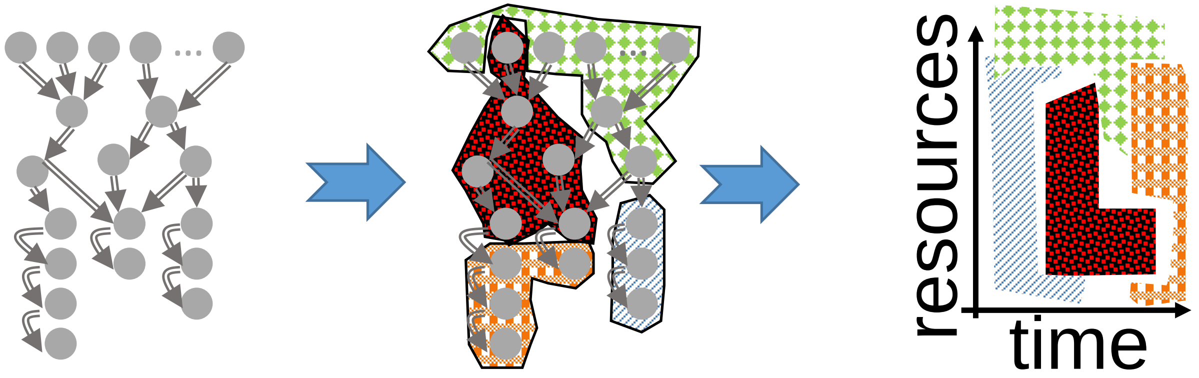
Existing schedulers:

A task is schedulable *after* all its parents have *finished*

Graphene:

Identifies *troublesome tasks* and places them *first*

Main ideas for one DAG



Existing schedulers:

A task is schedulable *after* all its parents have *finished*

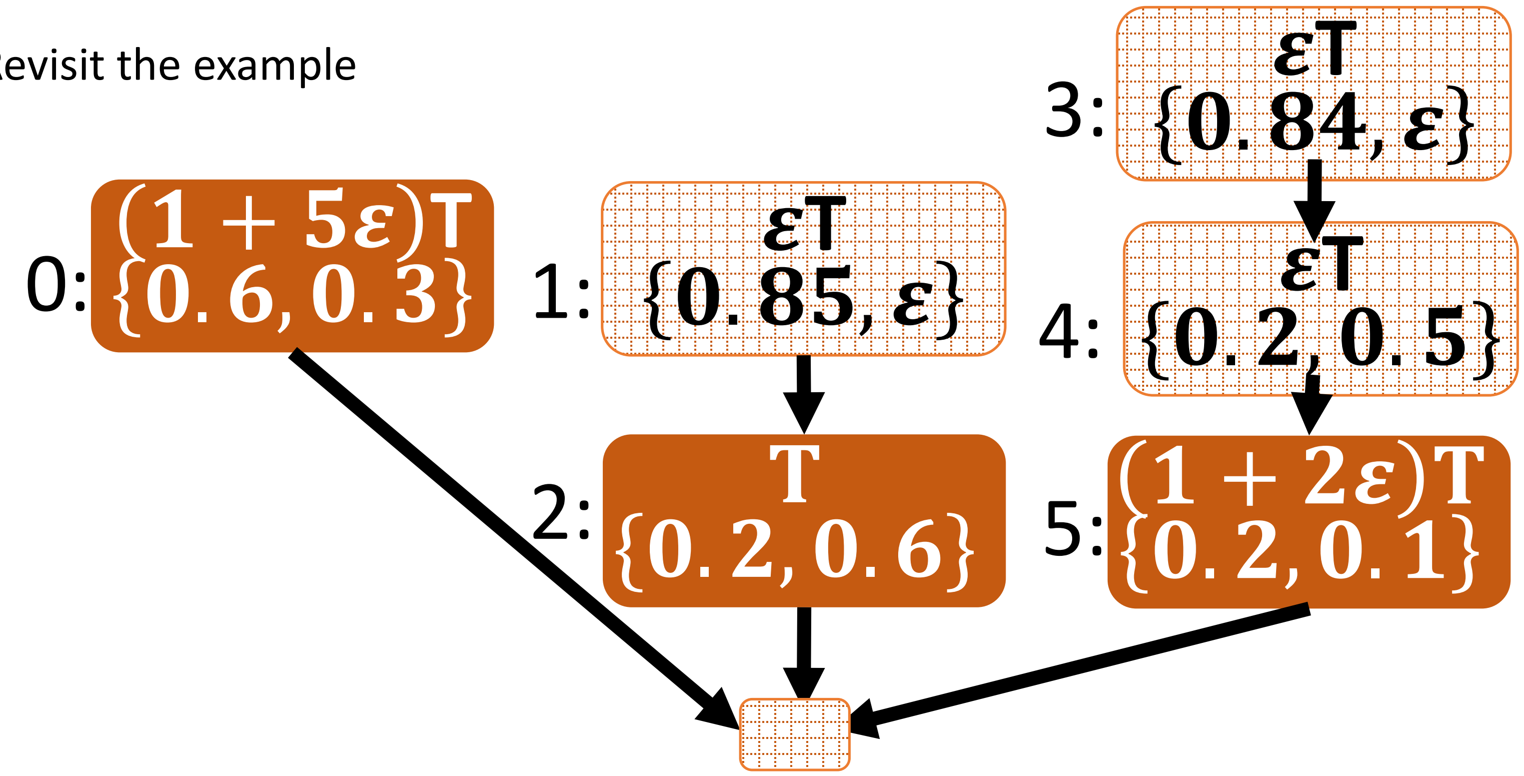
Graphene:

Identifies *troublesome tasks* and places them *first*

Place other tasks around trouble

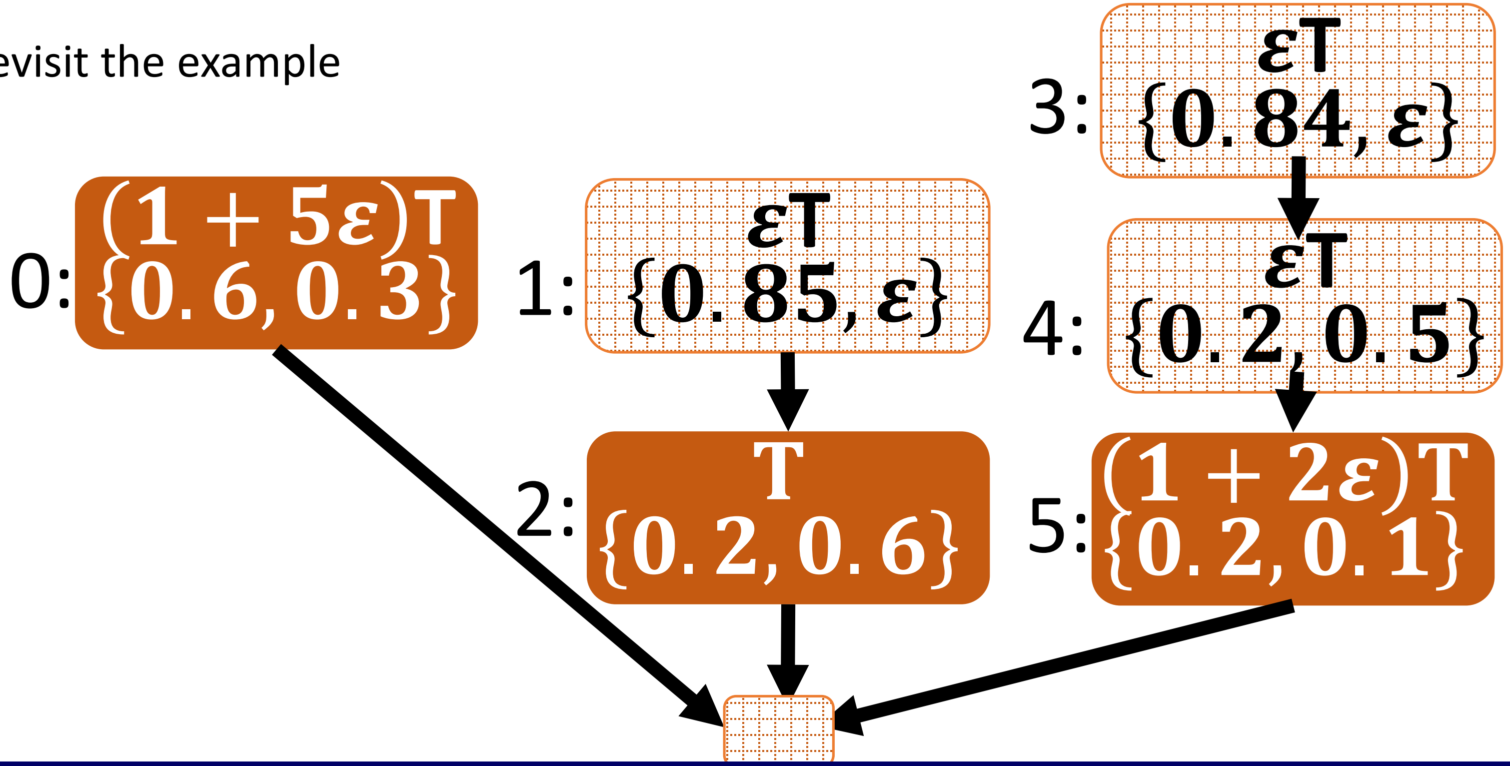
Does placing troublesome tasks first help?

Revisit the example



Does placing troublesome tasks first help?

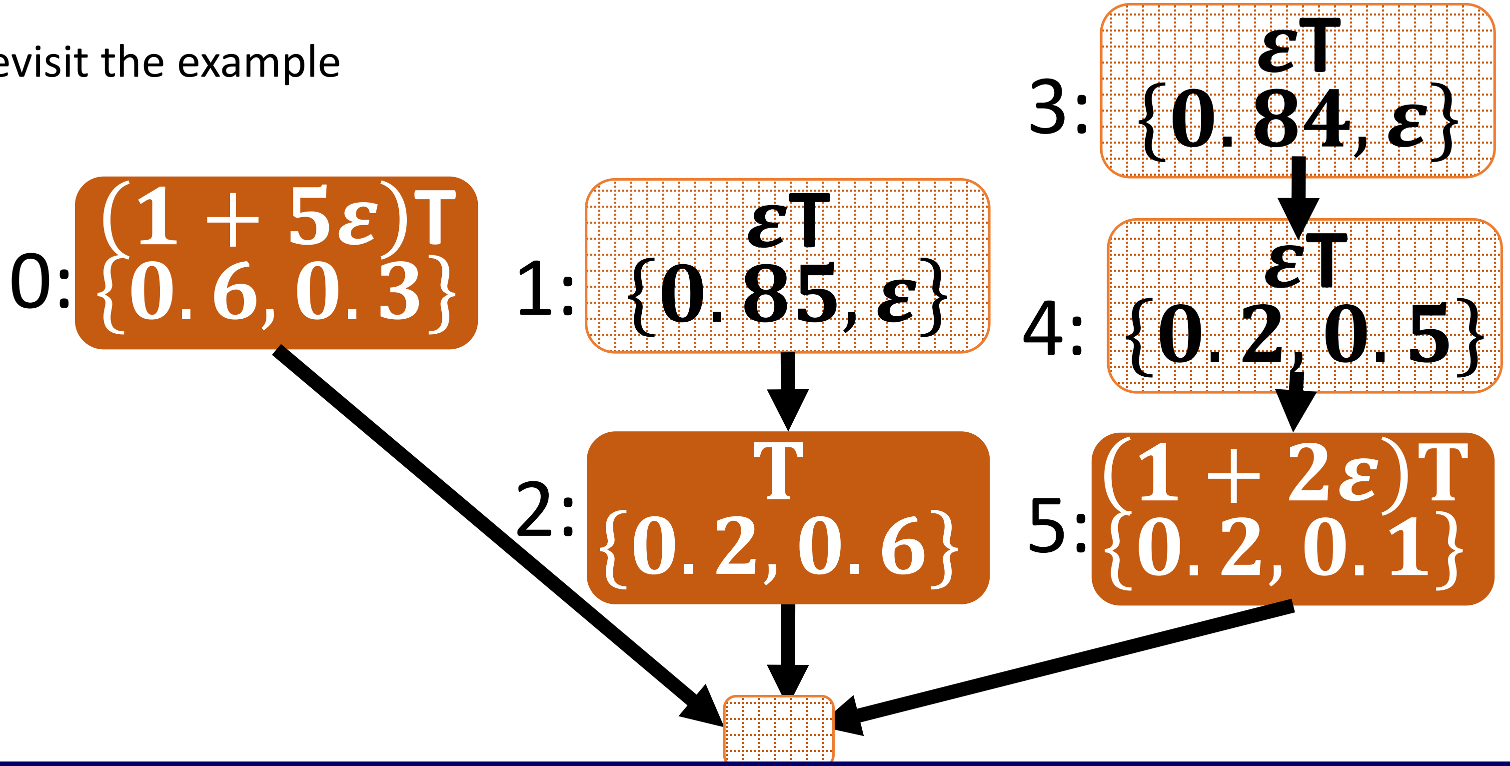
Revisit the example

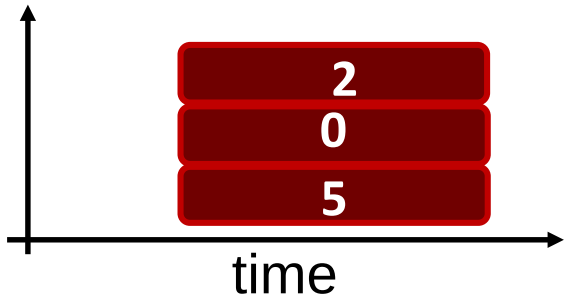


If troublesome tasks $\supseteq$ long-running tasks, Graphene $\equiv$ OPT

Does placing troublesome tasks first help?

Revisit the example

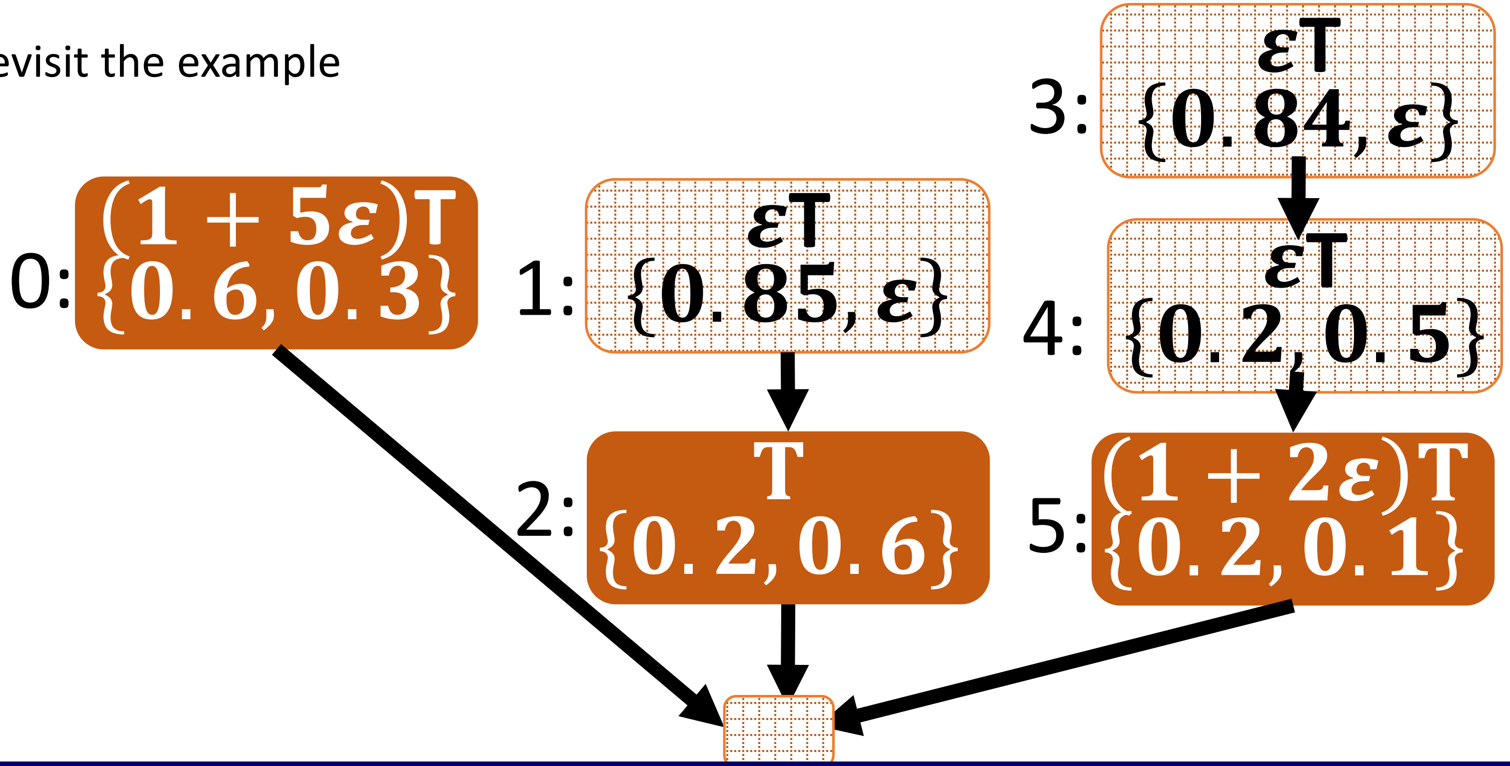


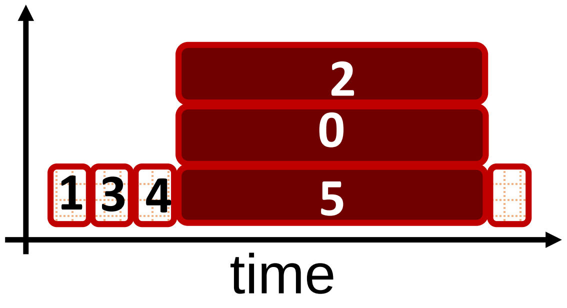


If troublesome tasks $\supseteq$ long-running tasks, Graphene $\equiv$ OPT

Does placing troublesome tasks first help?

Revisit the example





If troublesome tasks $\supseteq$ long-running tasks, Graphene $\equiv$ OPT

How to choose troublesome tasks T?

$\text{frag} \geq f$

How to choose troublesome tasks T?

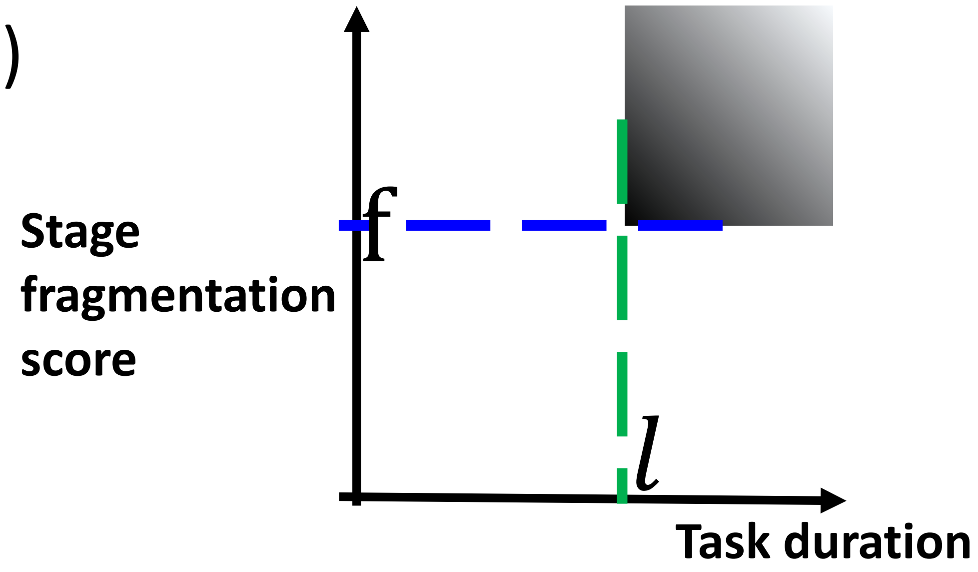
Optimal choice is intractable (recall: NP-Hard)

$$\text{frag} \geq f$$

How to choose troublesome tasks T?

Optimal choice is intractable (recall: NP-Hard)

$$\text{frag} \geq f$$



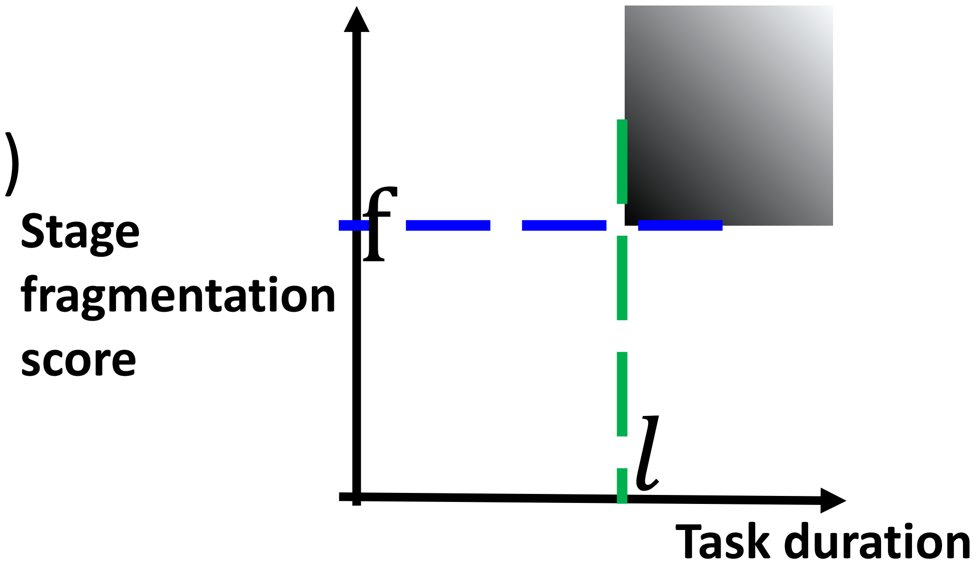
How to choose troublesome tasks T?

ff

Optimal choice is intractable (recall: NP-Hard)

Graphene:

`BuildSchedule(T)` $\left\{ \begin{matrix} \text{and} \\ \text{or} \end{matrix} \right\}$



How to choose troublesome tasks T?

$f f$

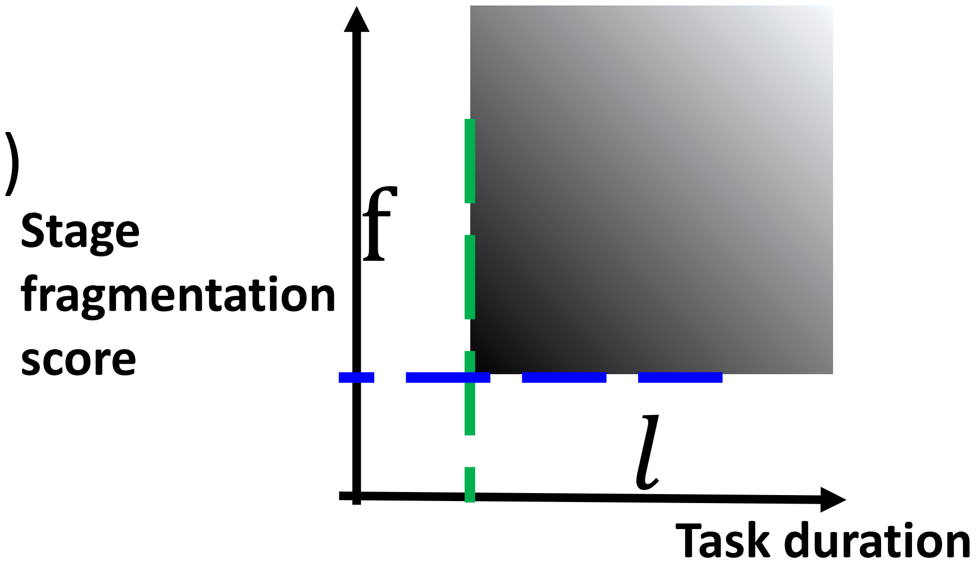
Optimal choice is intractable (recall: NP-Hard)

Vary l, f

Graphene: and / or

`BuildSchedule(T)`

Pick the most compact schedule



How to choose troublesome tasks T?

ff

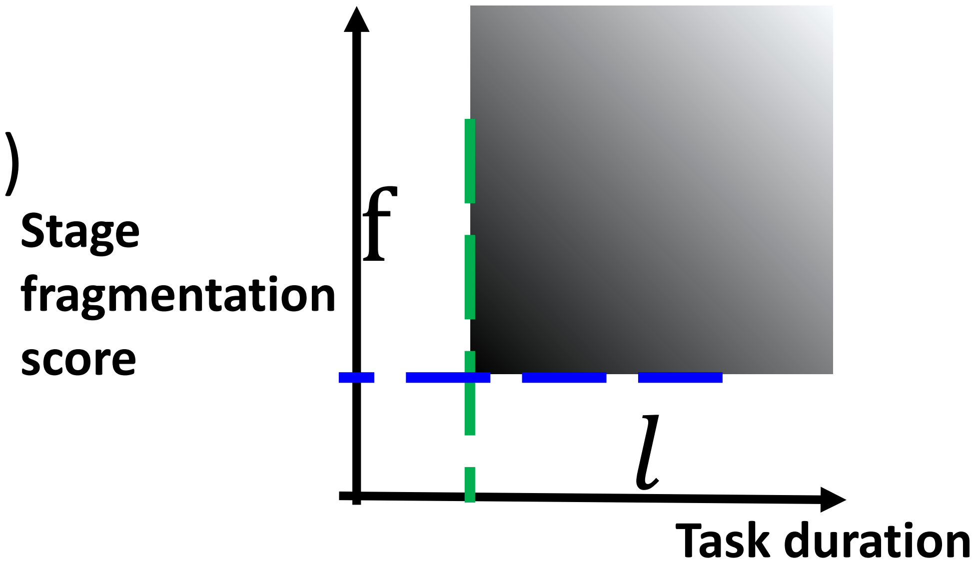
Optimal choice is intractable (recall: NP-Hard)

Vary l, f

Graphene:

`BuildSchedule(T)` $\left\{\begin{array}{c}\text{and}\\ \text{or}\end{array}\right\}$

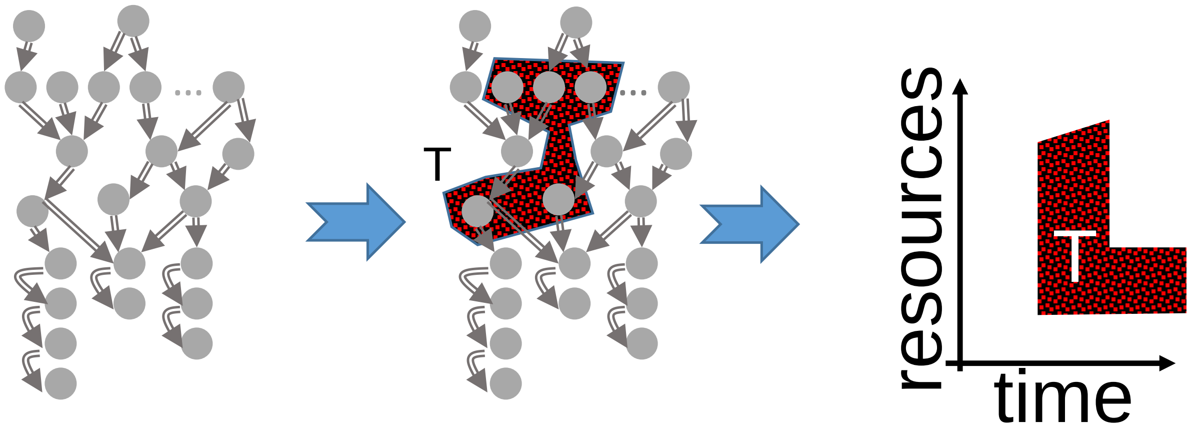
Pick the most compact schedule



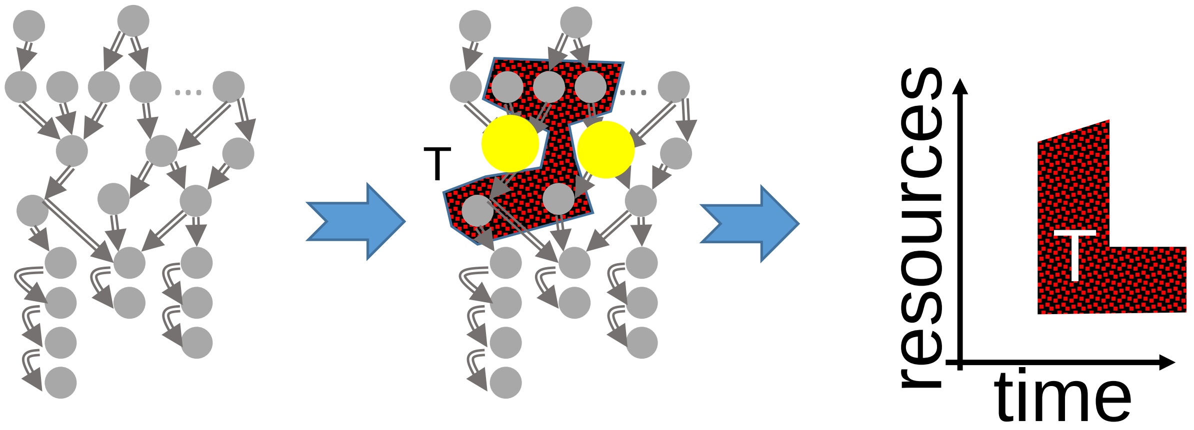
Extensions

1) Explore different choices of T in parallel
2) Recurse
3) Memoize …

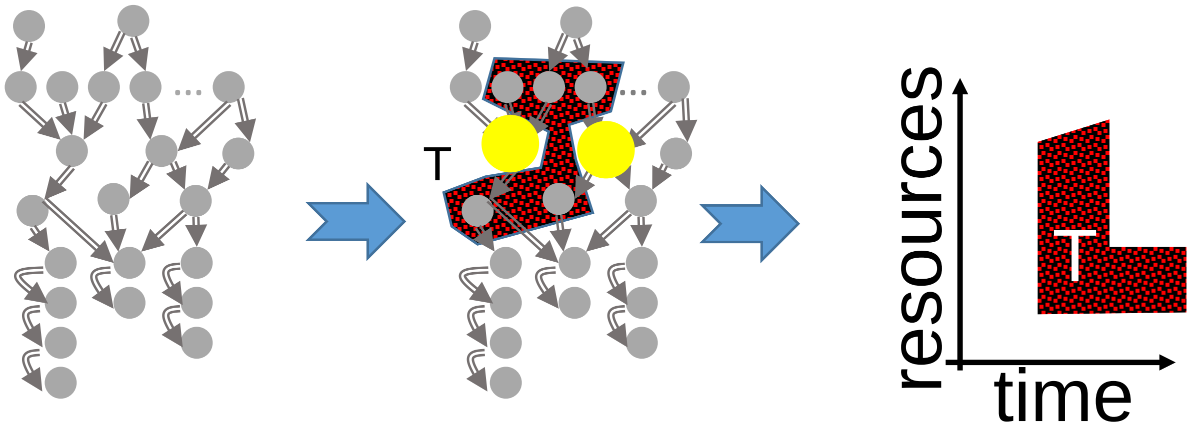
Schedule dead-ends



Schedule dead-ends

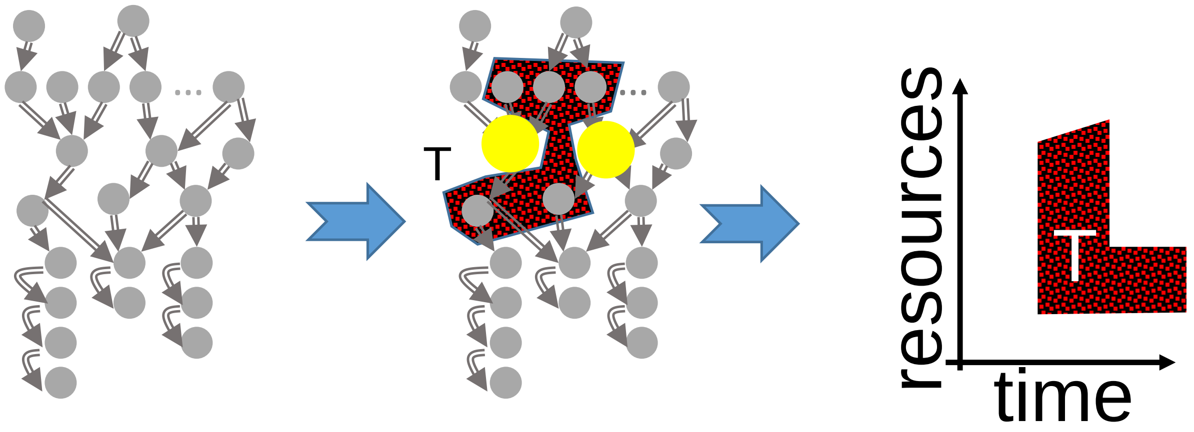


Schedule dead-ends



1) Since some parents and children of are already *placed* with T, may not be able to place

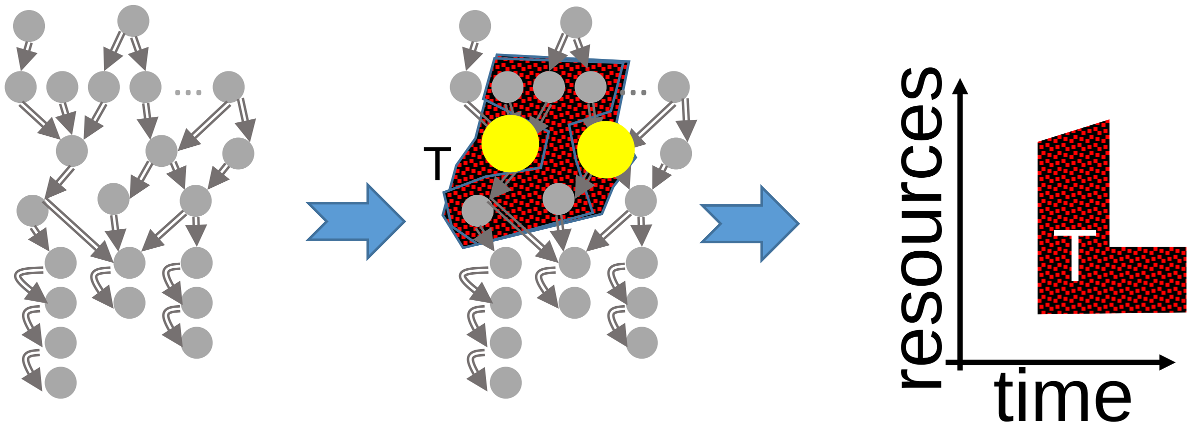
Schedule dead-ends



1) Since some parents and children of are already *placed* with T, may not be able to place

```
T ← TransitiveClosure(T)
```

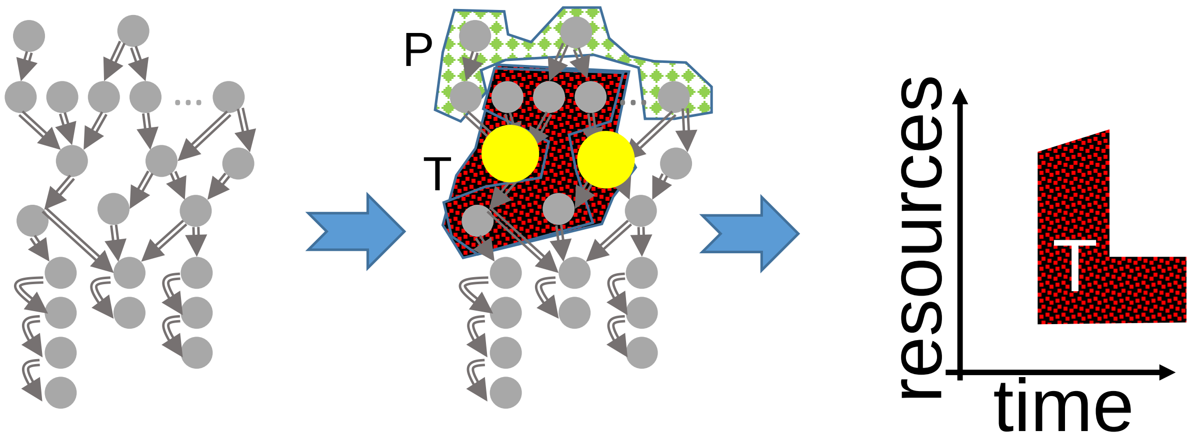
Schedule dead-ends



1) Since some parents and children of are already *placed* with T, may not be able to place

```
T ← TransitiveClosure(T)
```

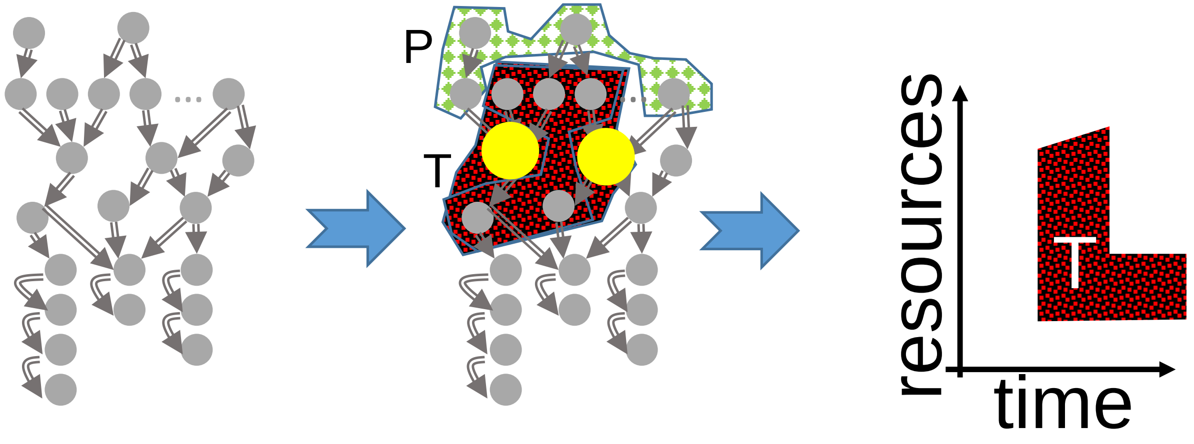
Schedule dead-ends



1) Since some parents and children of are already *placed* with T, may not be able to place

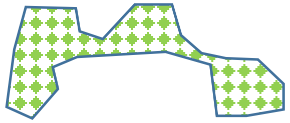
```
T ← TransitiveClosure(T)
```

Schedule dead-ends

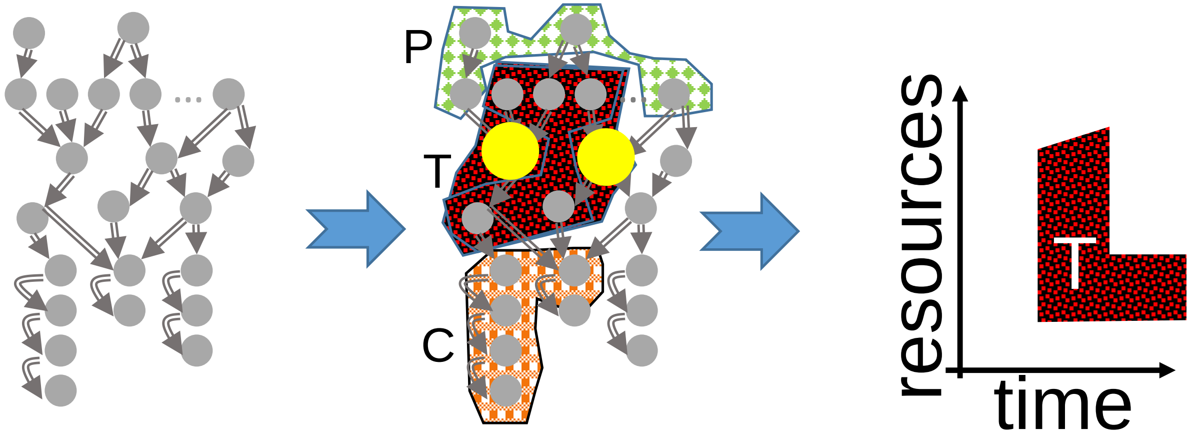


1) Since some parents and children of are already *placed* with T, may not be able to place

`T ← TransitiveClosure(T)`

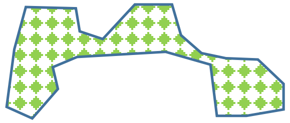
2) When placing tasks in , P, have to go *backwards* (place task after all children are placed)

Schedule dead-ends

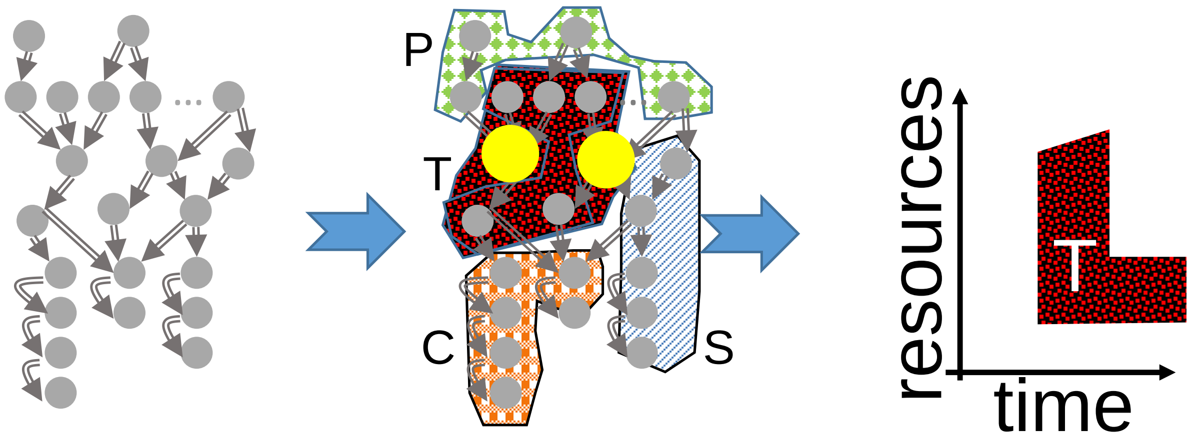


1) Since some parents and children of are already *placed* with T, may not be able to place

`T ← TransitiveClosure(T)`

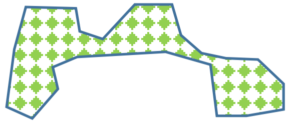
2) When placing tasks in , P, have to go *backwards* (place task after all children are placed)

Schedule dead-ends



1) Since some parents and children of are already *placed* with T, may not be able to place

T ← TransitiveClosure(T)

2) When placing tasks in , P, have to go *backwards* (place task after all children are placed)

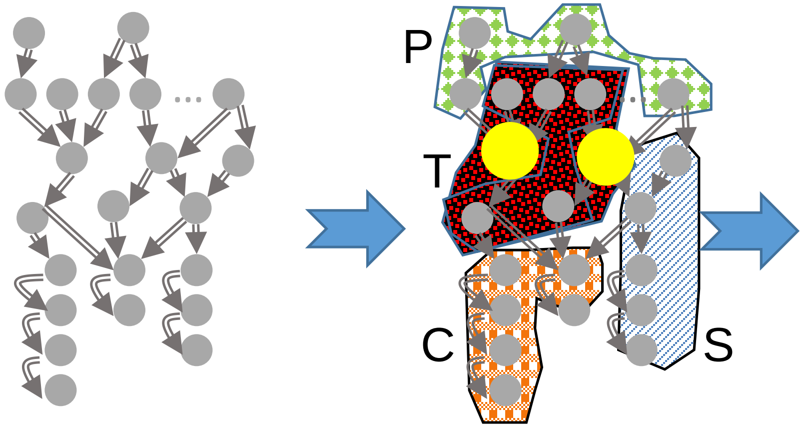
Schedule dead-ends

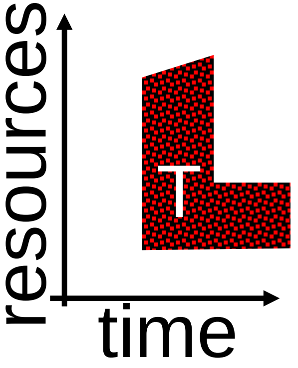
Which of these orders are legit?

$T_{fb}P_bS_fC_f$

$T_{fb}P_bC_fS_f$

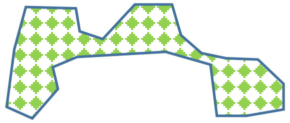
$T_{fb}S_{fb}P_bC_f$





1) Since some parents and children of are already *placed* with T, may not be able to place

`T ← TransitiveClosure(T)`

2) When placing tasks in , P, have to go *backwards* (place task after all children are placed)

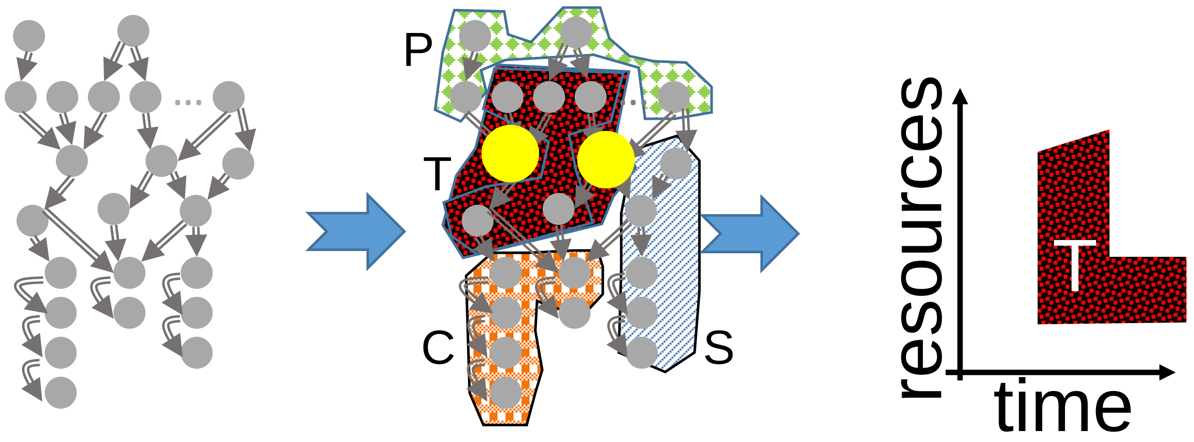
Schedule dead-ends

Which of these orders are legit?

$T_{fb}P_bS_fC_f$ ✓

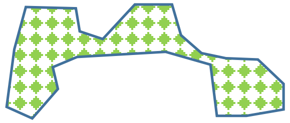
$T_{fb}P_bC_fS_f$ ✗

$T_{fb}S_{fb}P_bC_f$ ✓



1) Since some parents and children of are already *placed* with T, may not be able to place

`T ← TransitiveClosure(T)`

2) When placing tasks in  , P, have to go *backwards* (place task after all children are placed)

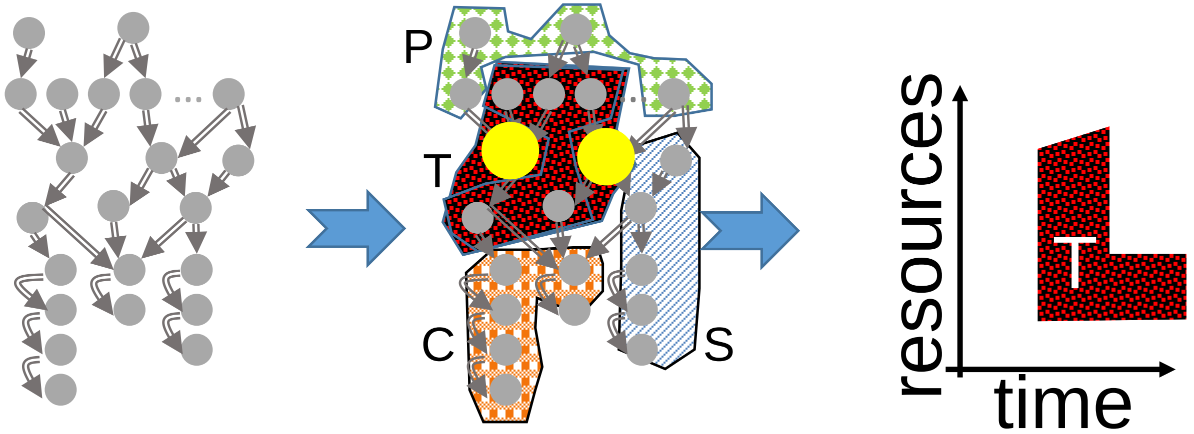
Schedule dead-ends

Which of these orders are legit?

$T_{fb}P_bS_fC_f$ ✓

$T_{fb}P_bC_fS_f$ ✗

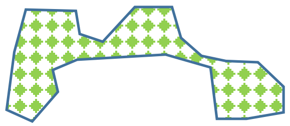
$T_{fb}S_{fb}P_bC_f$ ✓



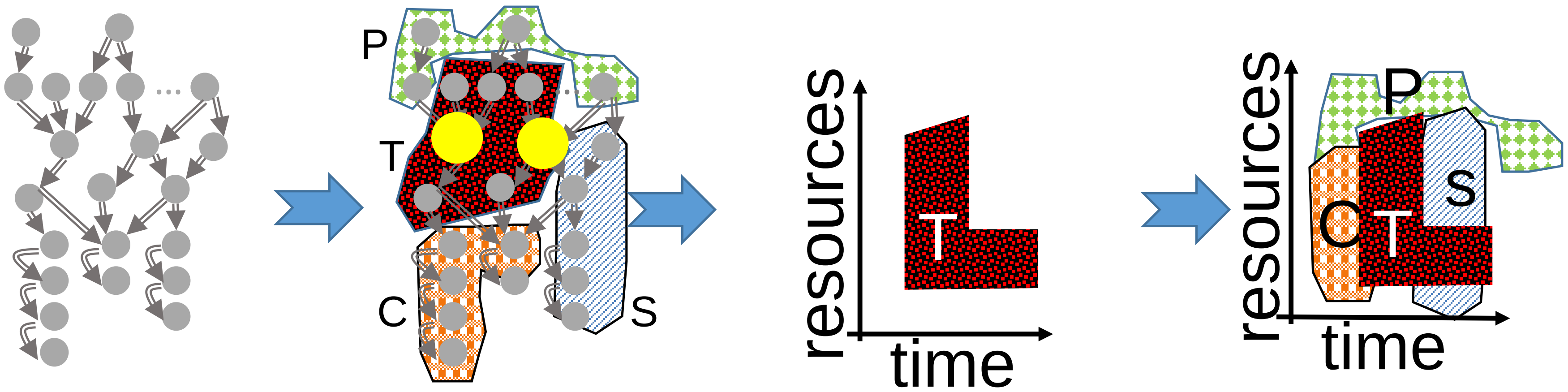
Graphene explores all orders and avoids dead-ends

1) Since some parents and children of are already *placed* with T, may not be able to place

$T \leftarrow$ `TransitiveClosure(T)`

2) When placing tasks in , P, have to go *backwards* (place task after all children are placed)

Main ideas for one DAG



1. Identify *troublesome tasks* and place them *first*
2. Systematically place tasks to avoid dead-ends

Computed offline schedule for

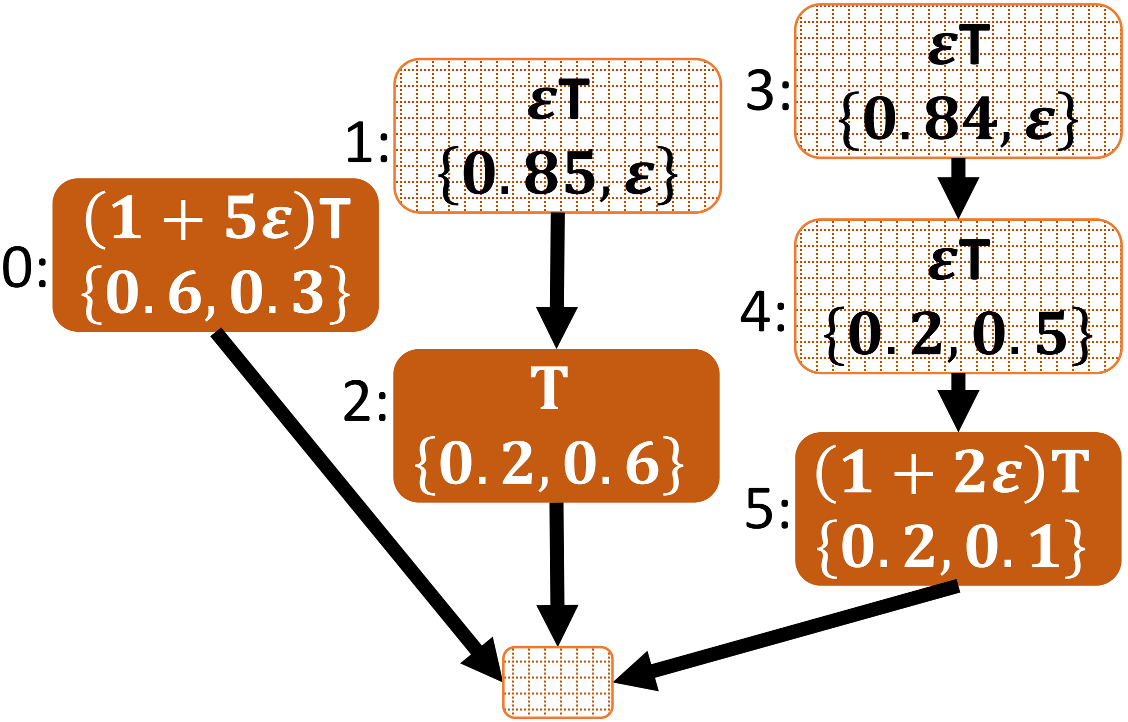
One DAG

Production clusters have

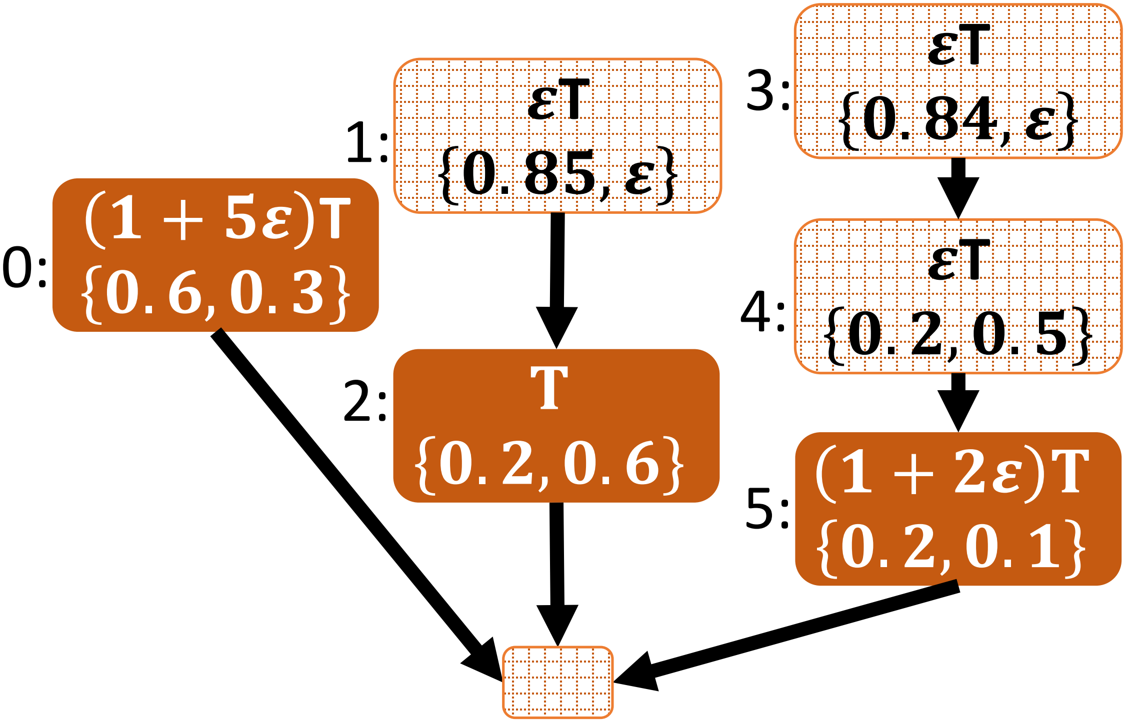
Multiple DAGs

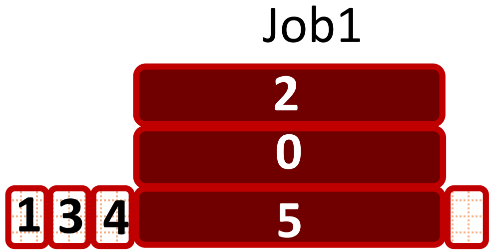
Convert offline schedule to priority order on tasks

Convert offline schedule to priority order on tasks

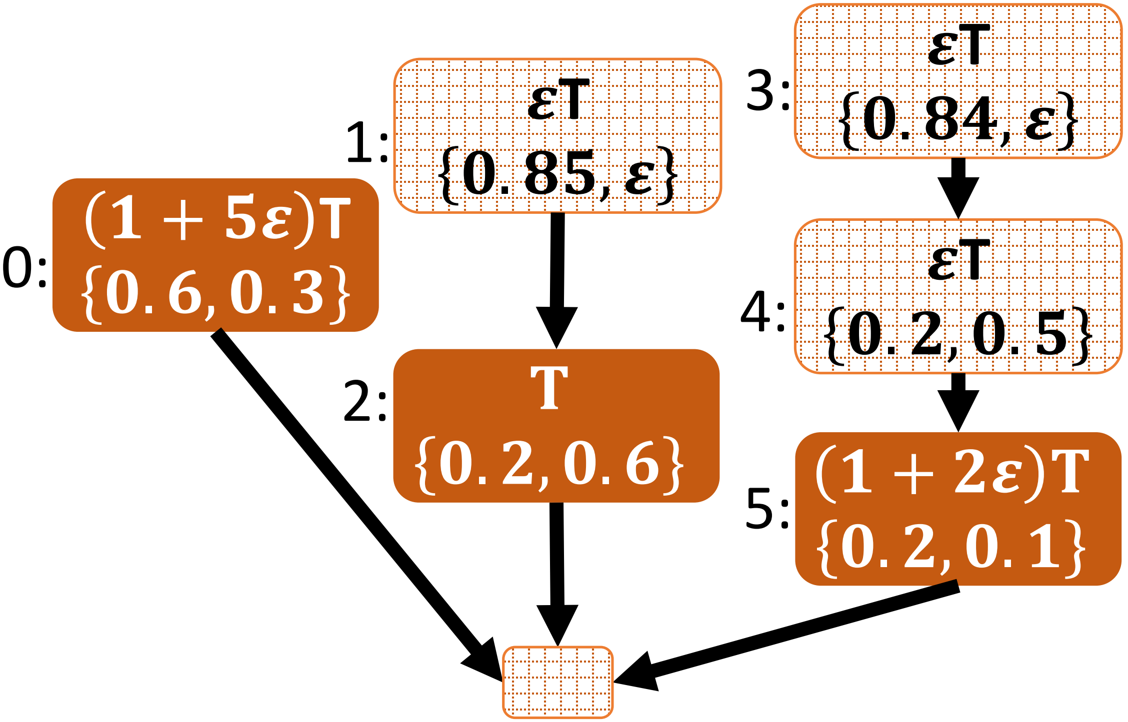


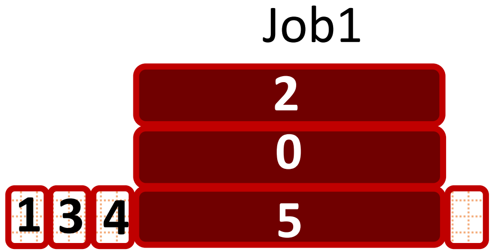
Convert offline schedule to priority order on tasks





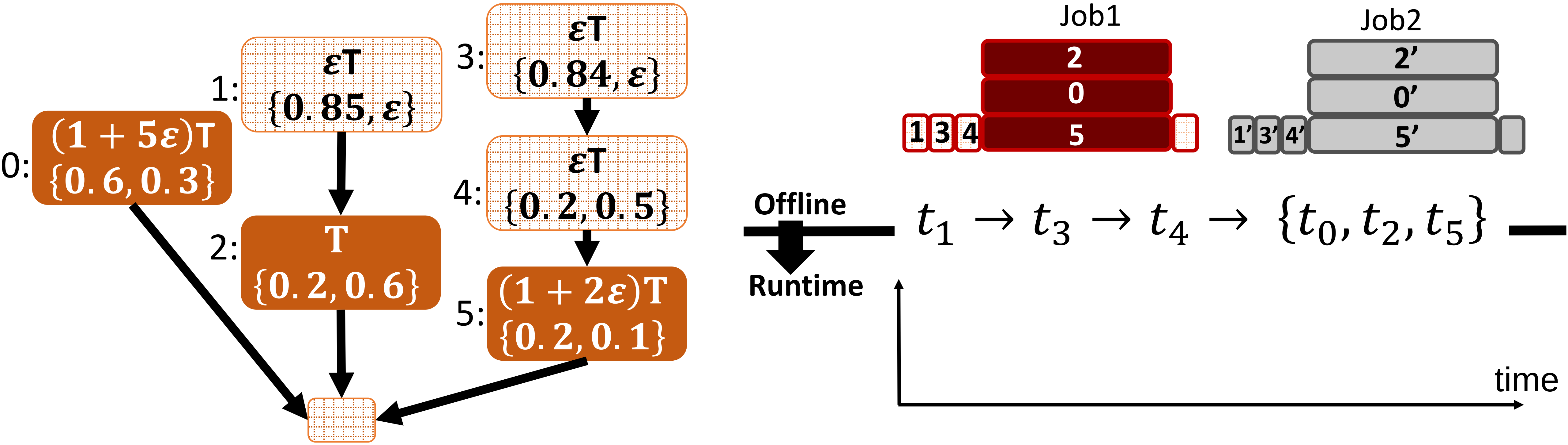
Convert offline schedule to priority order on tasks



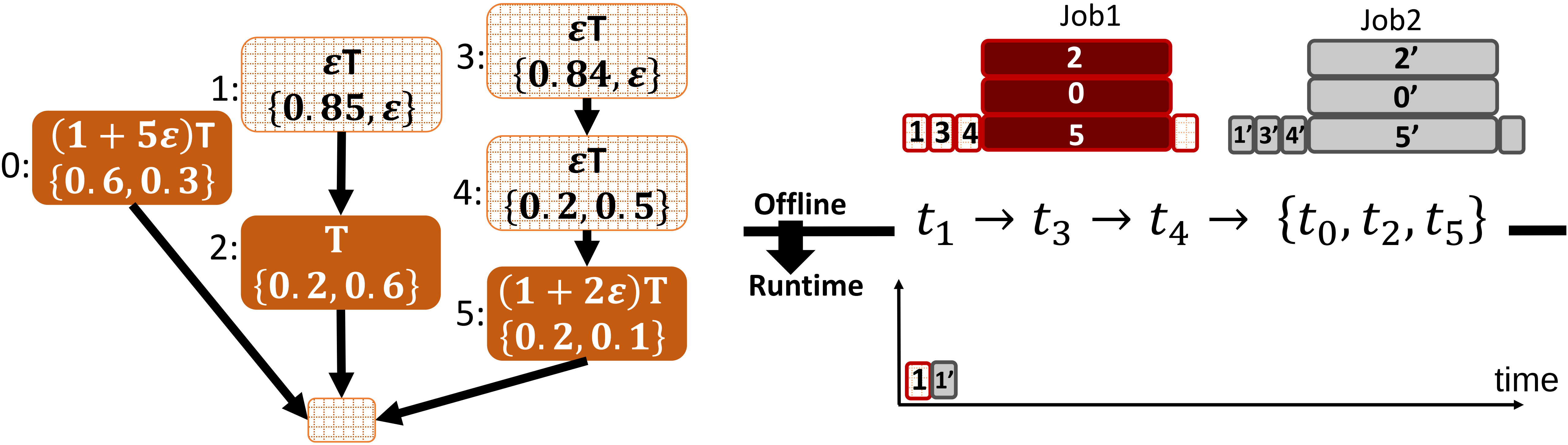


$$t_1 \rightarrow t_3 \rightarrow t_4 \rightarrow \{t_0, t_2, t_5\}$$

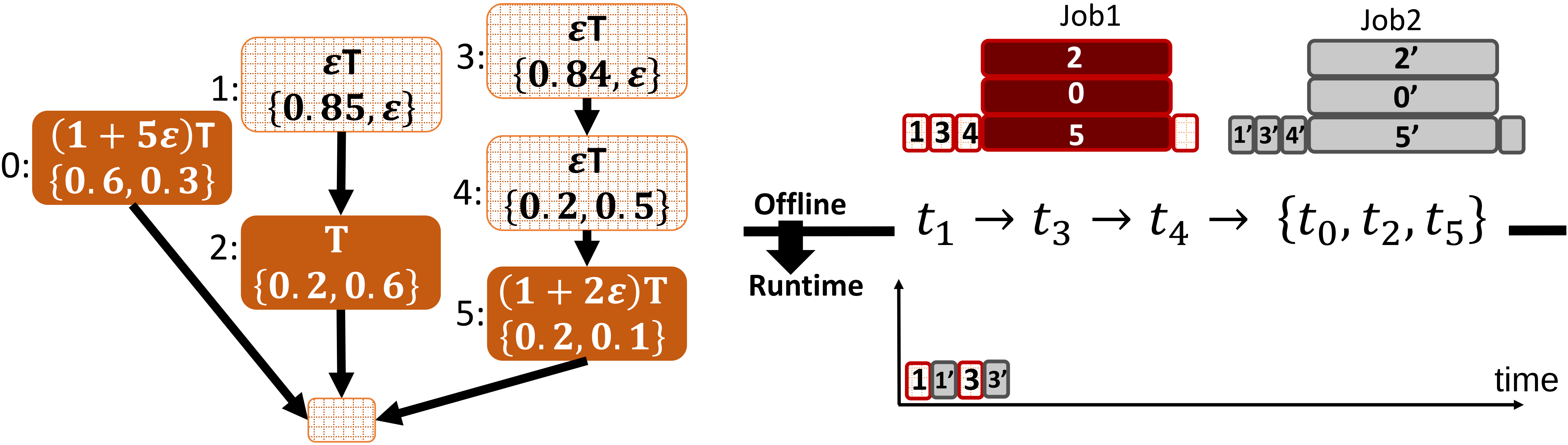
Convert offline schedule to priority order on tasks



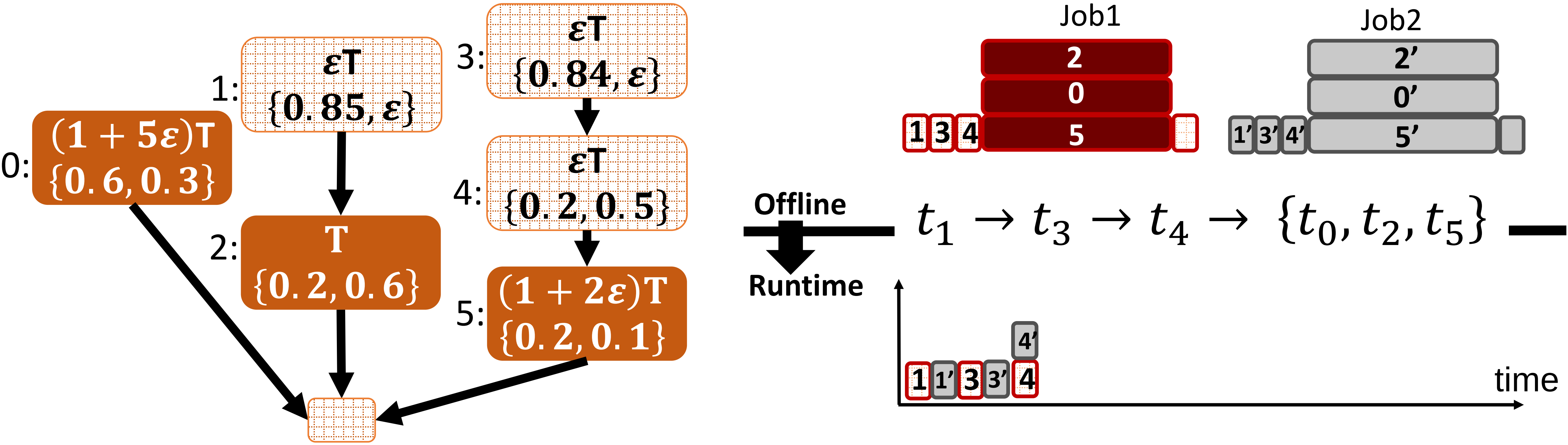
Convert offline schedule to priority order on tasks



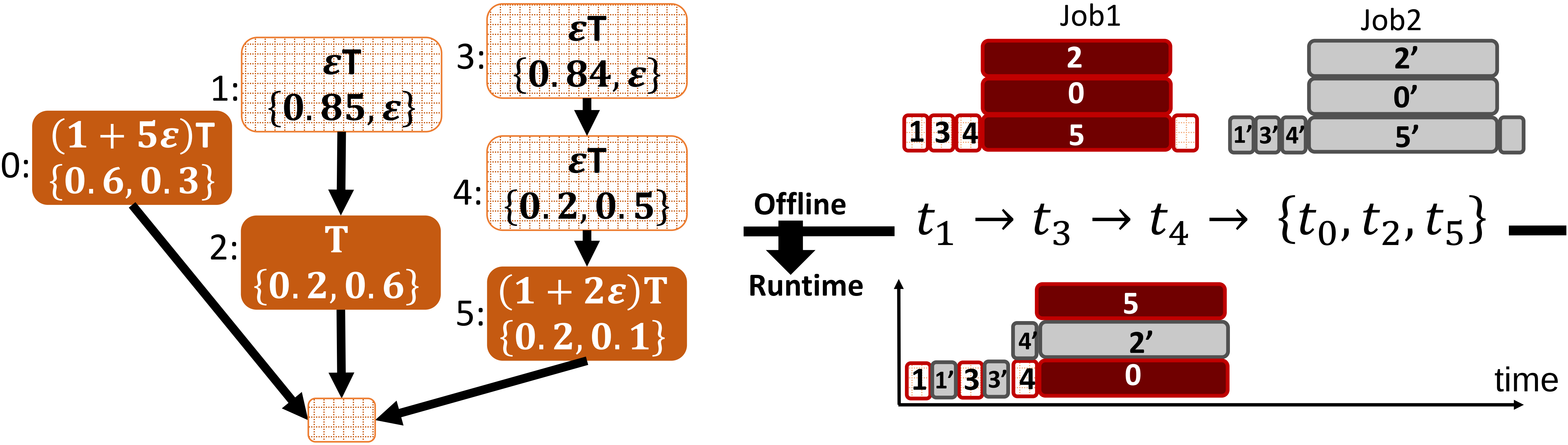
Convert offline schedule to priority order on tasks



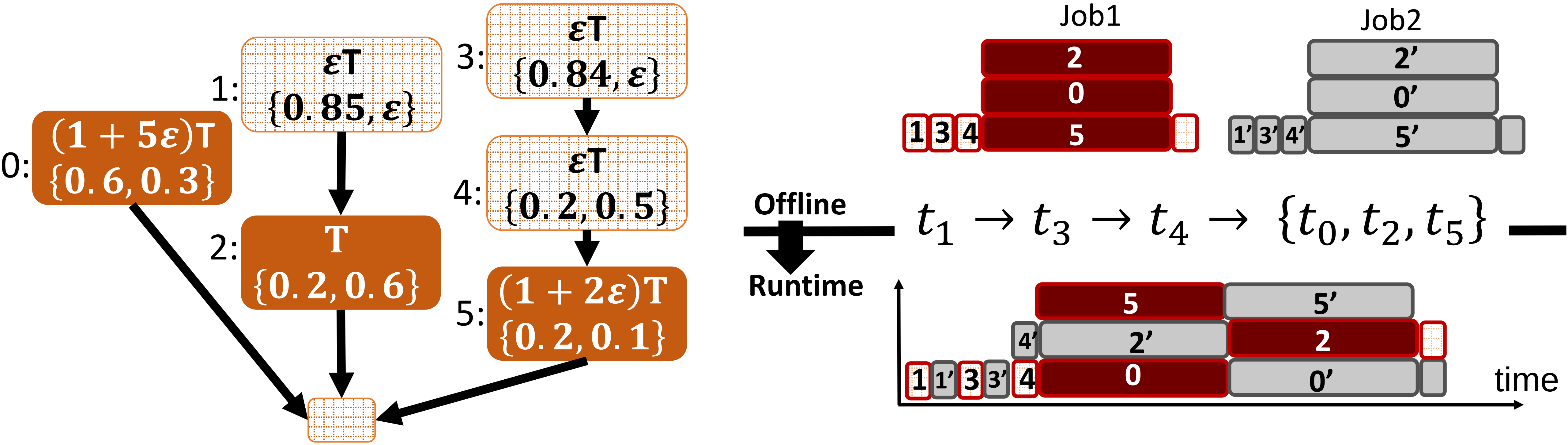
Convert offline schedule to priority order on tasks



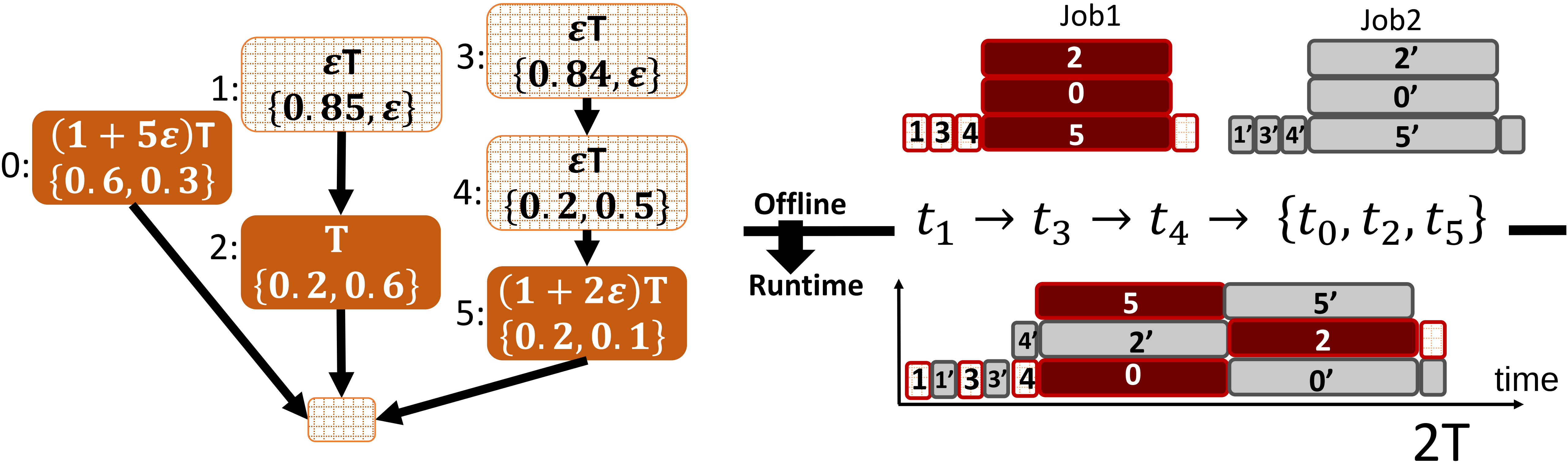
Convert offline schedule to priority order on tasks



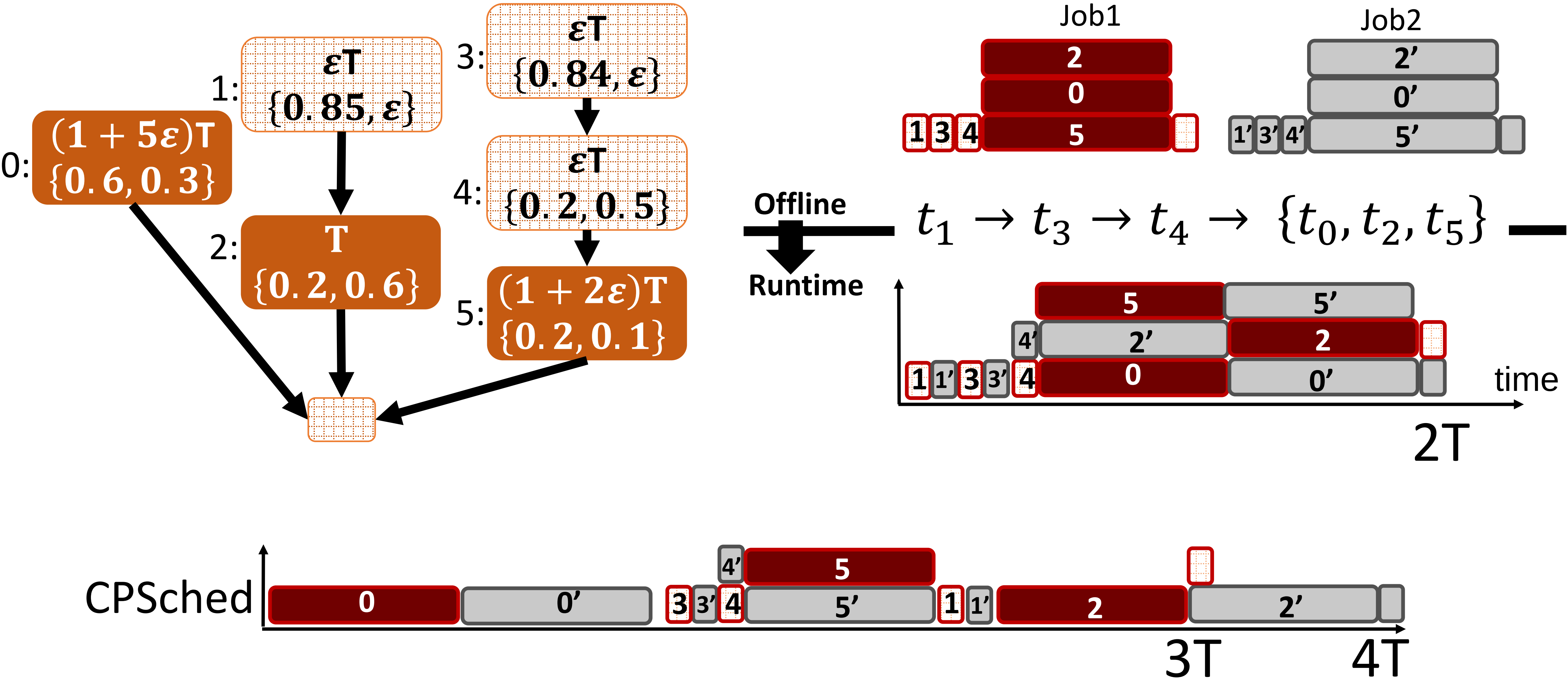
Convert offline schedule to priority order on tasks



Convert offline schedule to priority order on tasks



Convert offline schedule to priority order on tasks



Main ideas for multiple DAGs

1) Convert offline schedule to priority order on tasks

Main ideas for multiple DAGs

1) Convert offline schedule to priority order on tasks
2) Online, enforce schedule priority along with heuristics for
   (a) Multi-resource packing
   (b) “SRPT” to lower average job completion time
   (c) Bounded amount of unfairness
   (d) Overbooking …

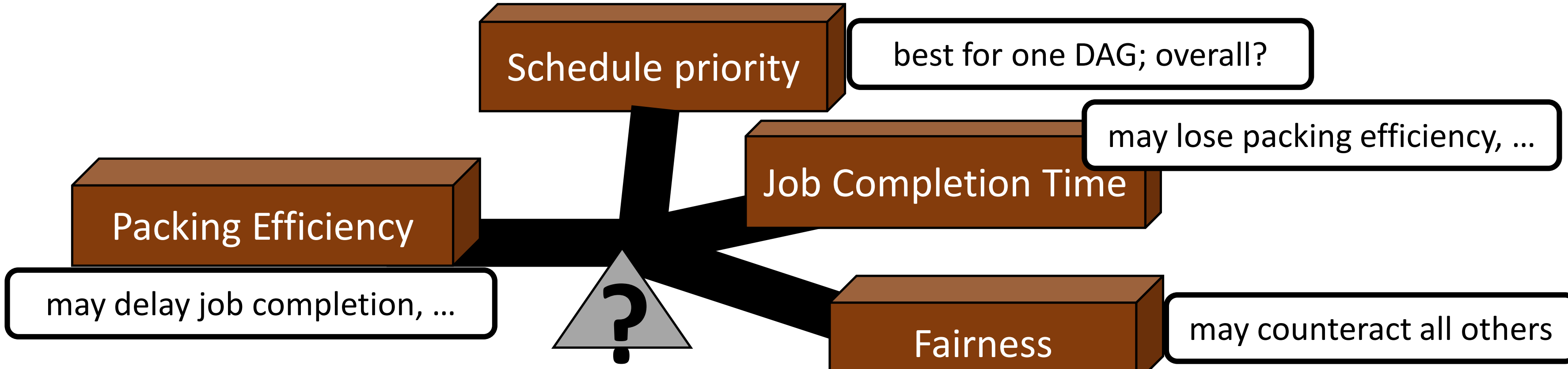
Main ideas for multiple DAGs

1) Convert offline schedule to priority order on tasks
2) Online, enforce schedule priority along with heuristics for
   (a) Multi-resource packing
   (b) “SRPT” to lower average job completion time
   (c) Bounded amount of unfairness
   (d) Overbooking …

Main ideas for multiple DAGs

1) Convert offline schedule to priority order on tasks
2) Online, enforce schedule priority along with heuristics for
   (a) Multi-resource packing
   (b) "SRPT" to lower average job completion time
   (c) Bounded amount of unfairness
   (d) Overbooking …

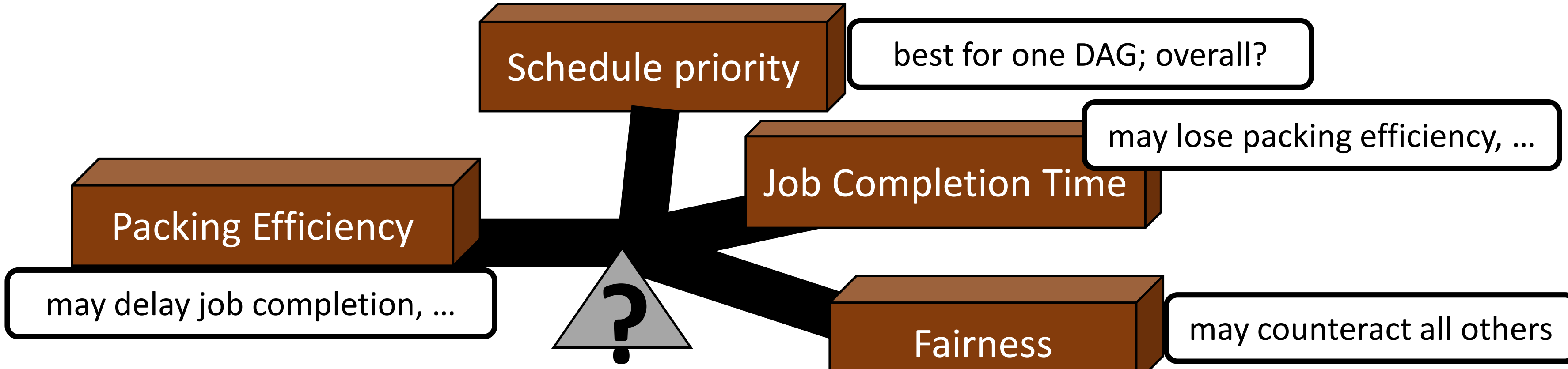
Trade-offs:



Main ideas for multiple DAGs

1) Convert offline schedule to priority order on tasks
2) Online, enforce schedule priority along with heuristics for
   (a) Multi-resource packing
   (b) "SRPT" to lower average job completion time
   (c) Bounded amount of unfairness
   (d) Overbooking …

Trade-offs:



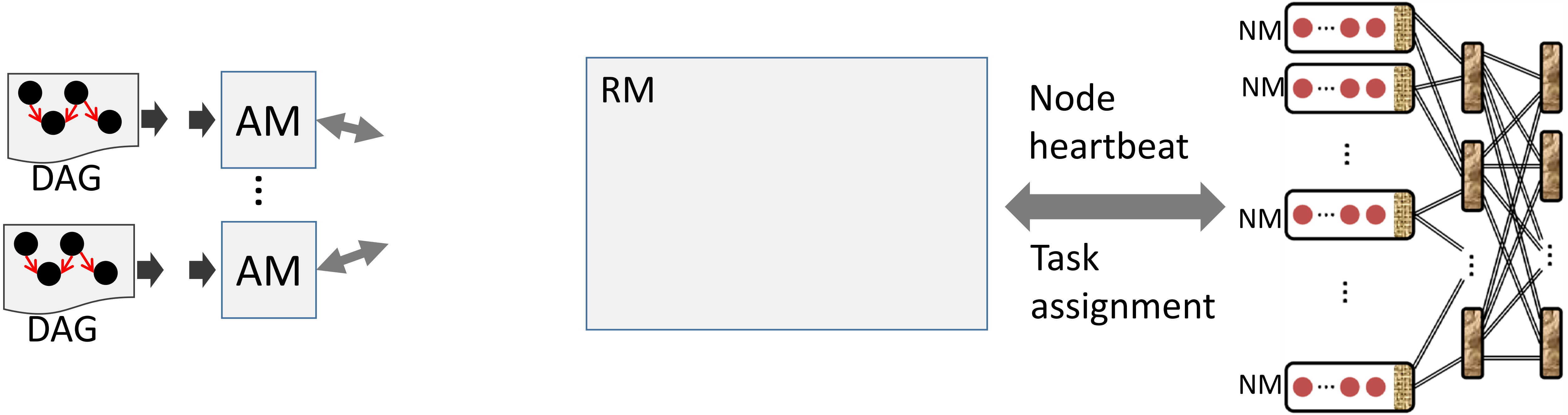
We show that:
{best "perf" |bounded unfairness} ~ best "perf"

Graphene summary & implementation

1) Offline, schedule each DAG by placing troublesome tasks first
2) Online, enforce priority over tasks along with other heuristics

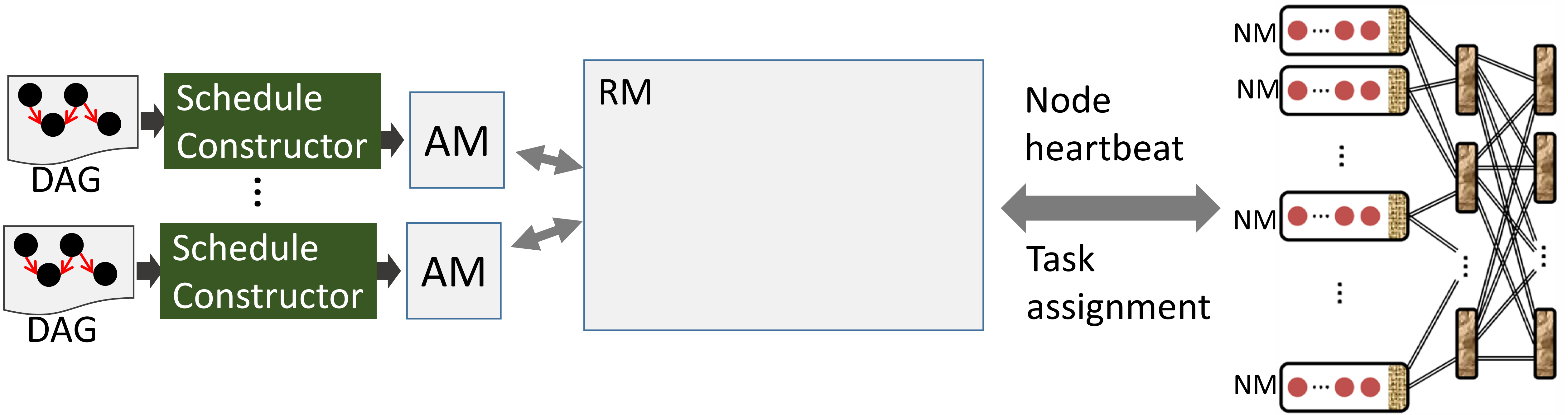
Graphene summary & implementation

1) Offline, schedule each DAG by placing troublesome tasks first
2) Online, enforce priority over tasks along with other heuristics



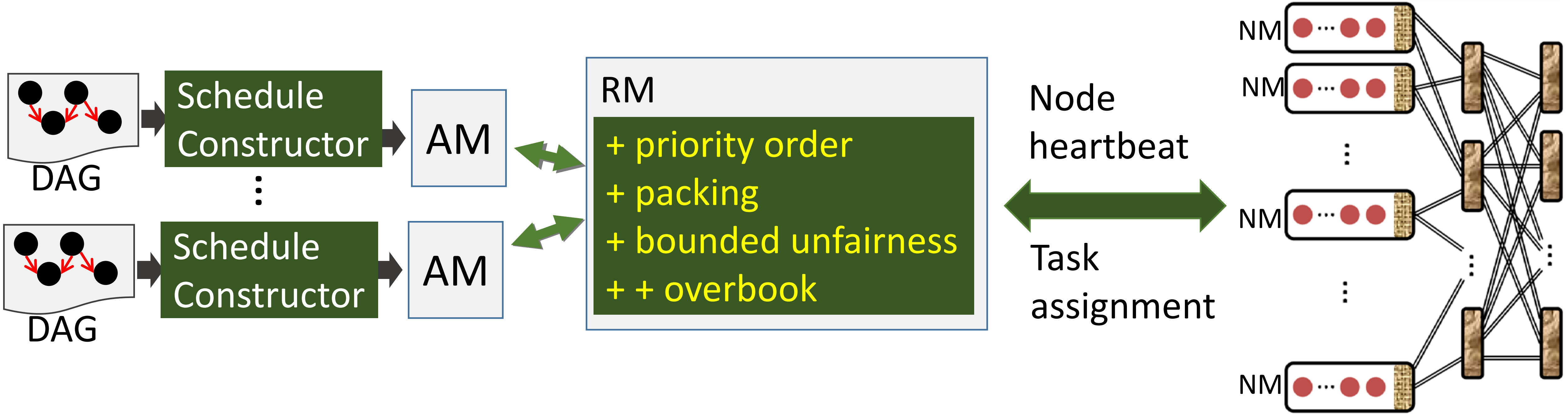
Graphene summary & implementation

1) Offline, schedule each DAG by placing troublesome tasks first
2) Online, enforce priority over tasks along with other heuristics



Graphene summary & implementation

1) Offline, schedule each DAG by placing troublesome tasks first
2) Online, enforce priority over tasks along with other heuristics



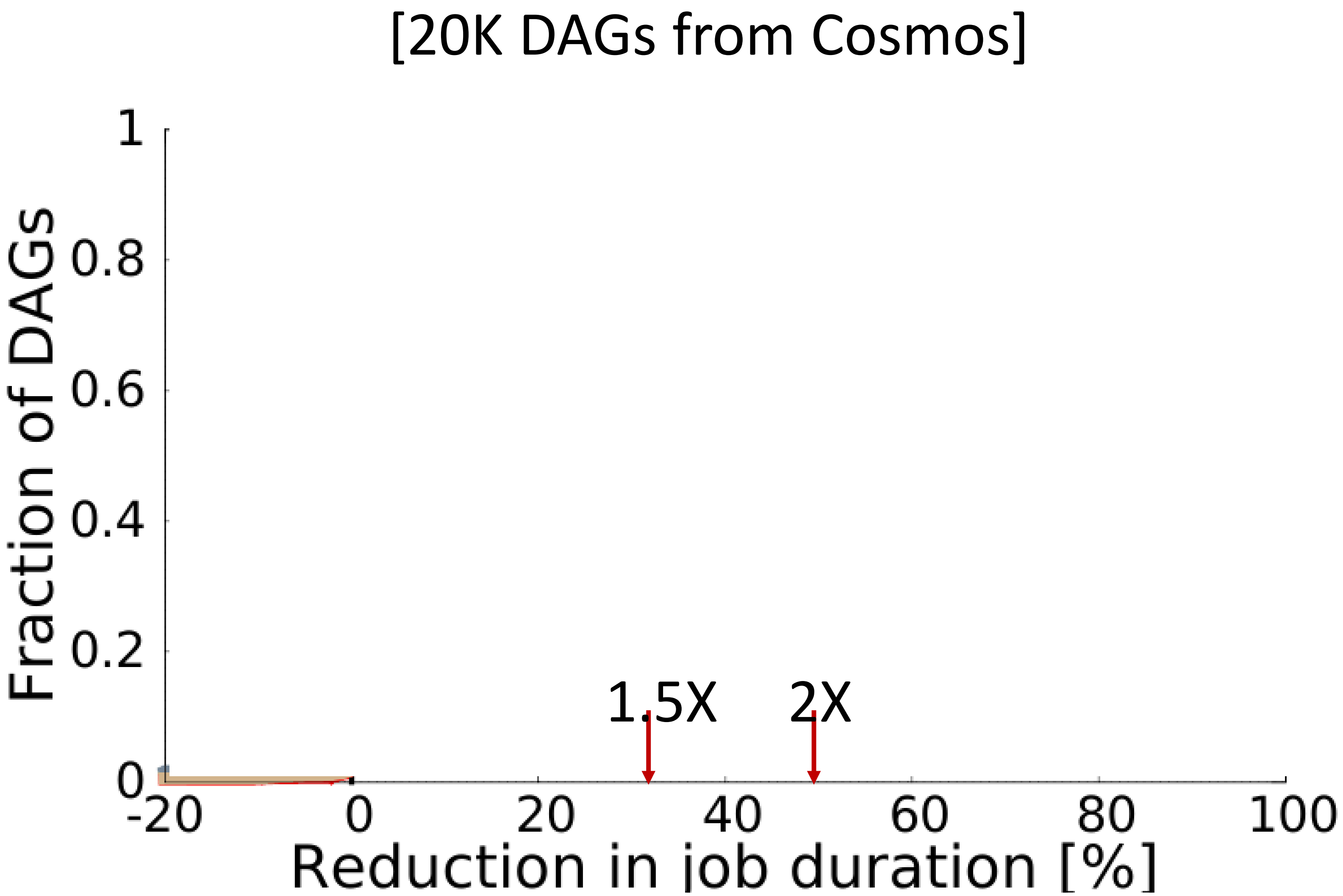
Implementation details

- DAG annotations
- Bundling: improve schedule quality w/o killing scheduling latency
- Co-existence with (many) other scheduler features

Evaluation

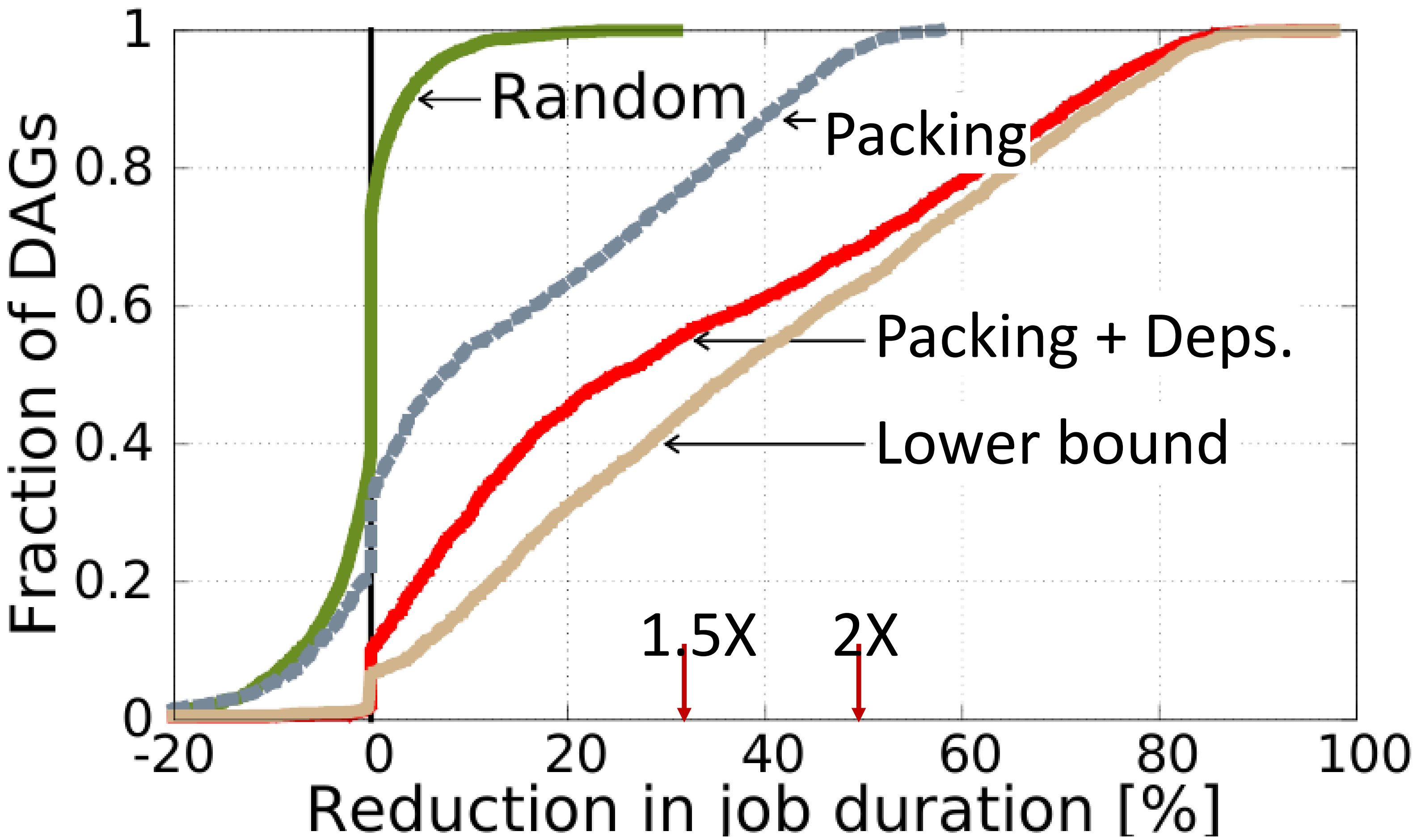
- Prototype
  - 200 server multi-core cluster
  - TPC-DS, TPC-H, …, GridMix to replay traces
  - Jobs arrive online
- Simulations
  - Traces from production Microsoft Cosmos and Yarn clusters
  - Compare with many alternatives

Results - 1


[20K DAGs from Cosmos]
Fraction of DAGs
1
0.8
0.6
0.4
0.2
0
-20
0
20
40
60
80
100
Reduction in job duration [%]
1.5X
2X

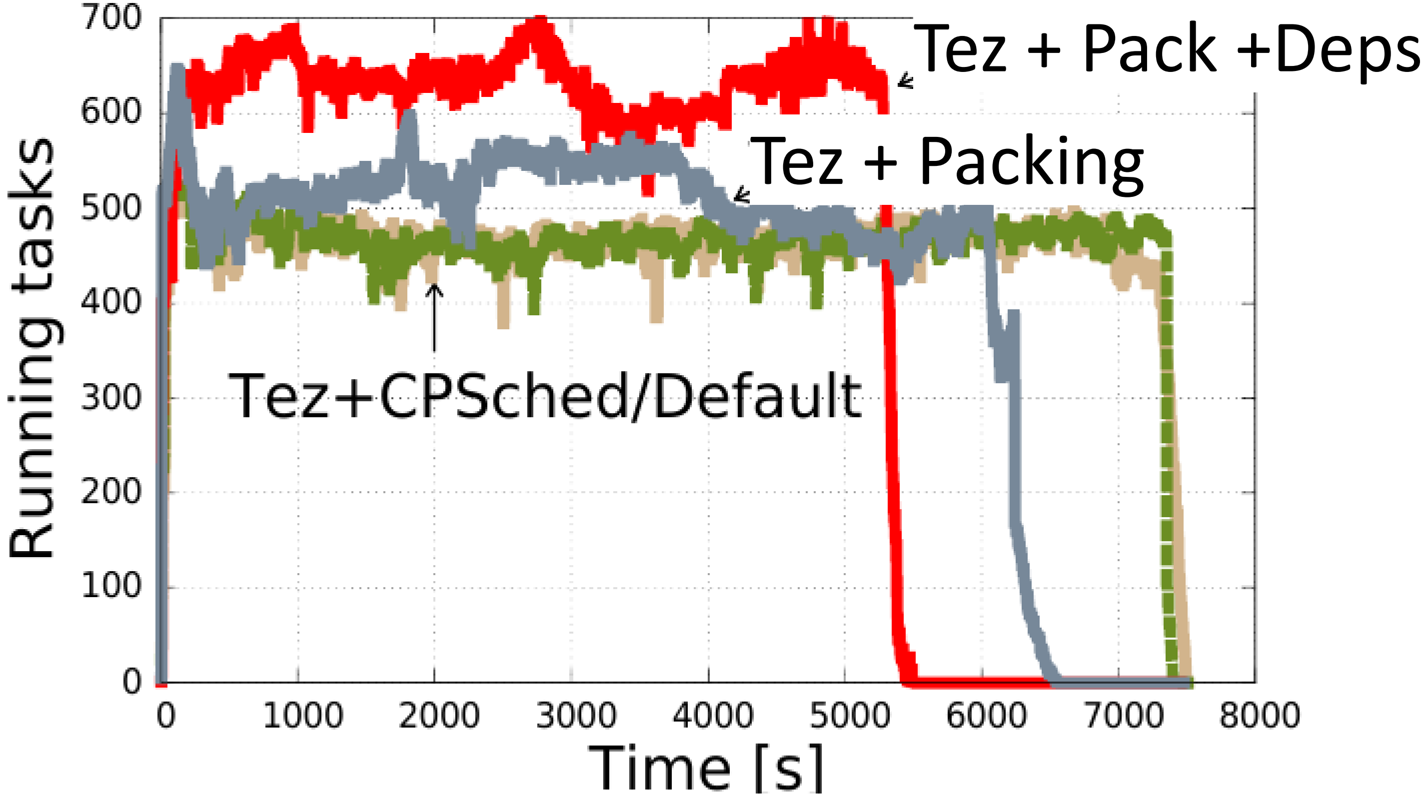
Results - 1

[20K DAGs from Cosmos]



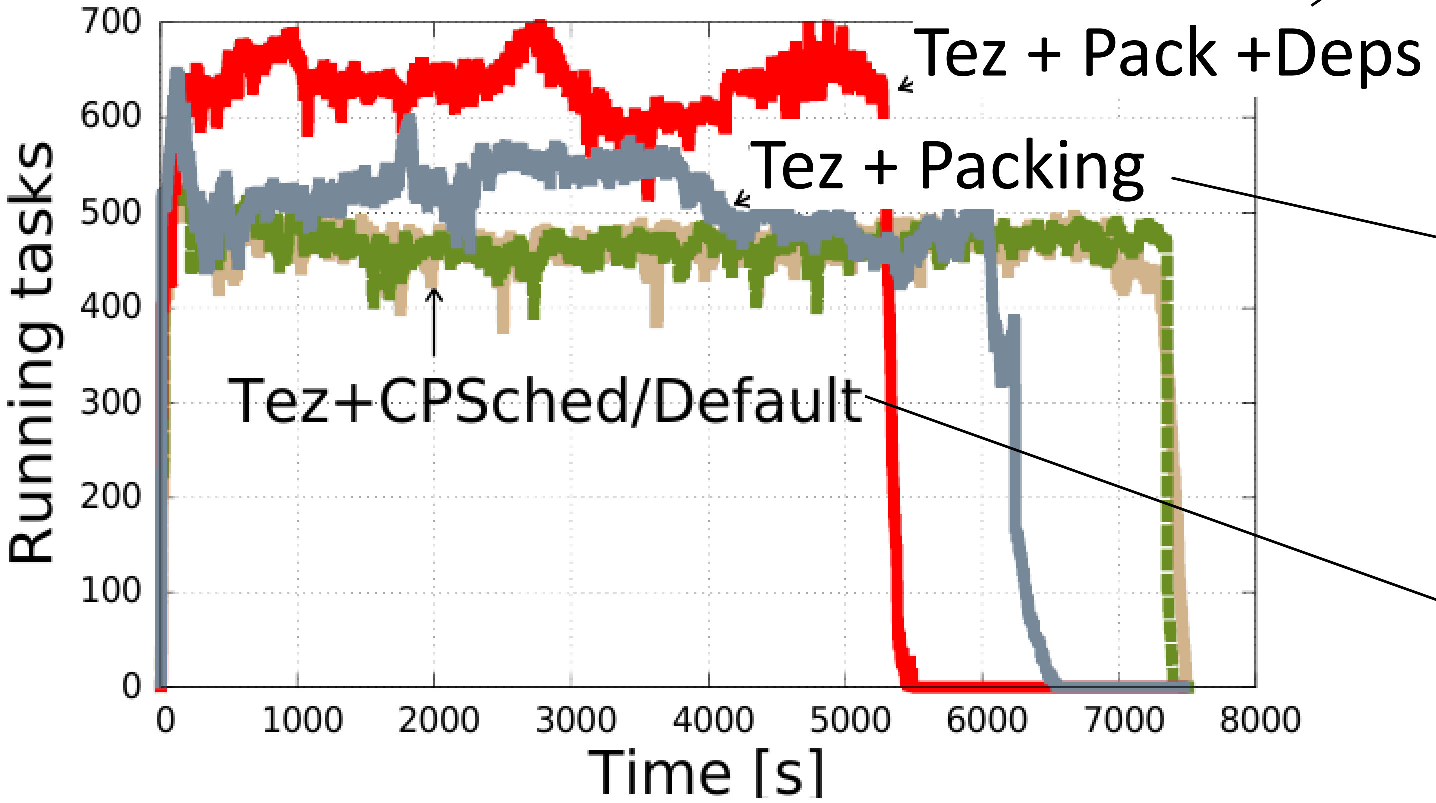
Results - 2

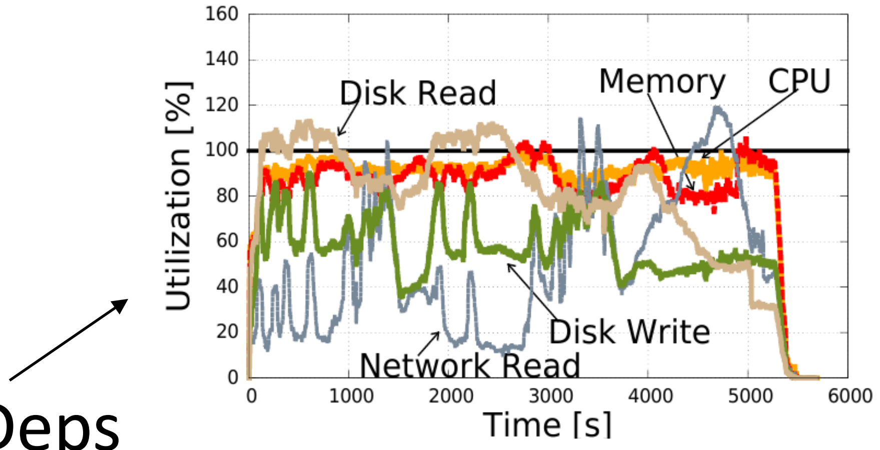
[200 jobs from TPC-DS, 200 server cluster]

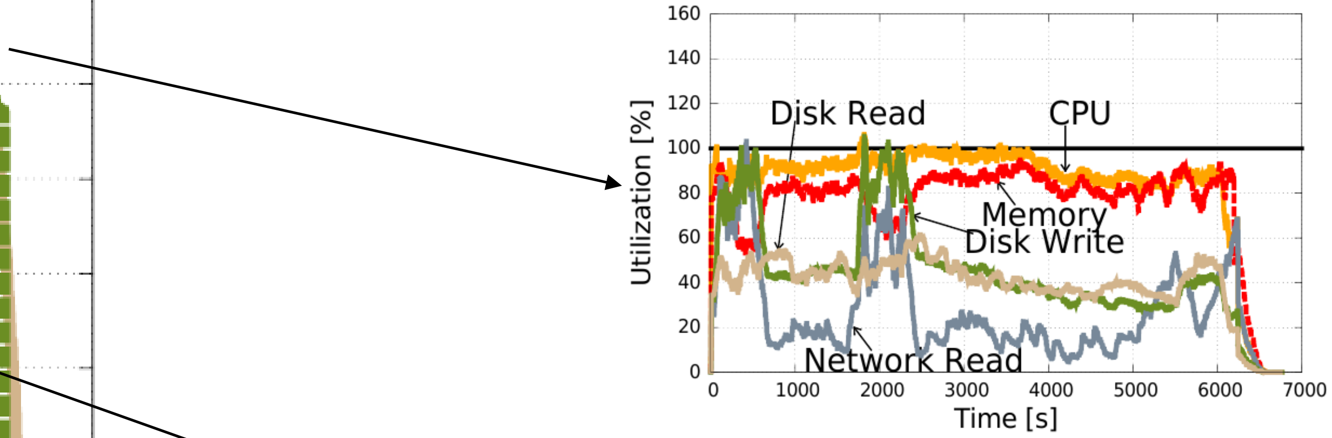


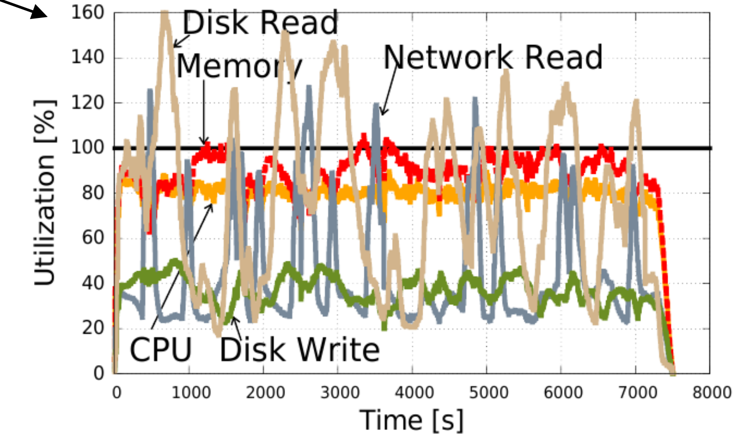
Results - 2

[200 jobs from TPC-DS, 200 server cluster]









Scheduling heterogeneous DAGs well requires an online solution that handles multiple resources and dependencies

Scheduling heterogeneous DAGs well requires an online solution that handles multiple resources and dependencies

Graphene

- Offline, construct per-DAG schedule by placing troublesome tasks first
- Online, enforce schedule priority along with other heuristics
- New lower bound shows nearly optimal for half of the DAGs

Scheduling heterogeneous DAGs well requires an online solution that handles multiple resources and dependencies

Graphene

- Offline, construct per-DAG schedule by placing troublesome tasks first
- Online, enforce schedule priority along with other heuristics
- New lower bound shows nearly optimal for half of the DAGs

Experiments show gains in job completion time, makespan, …

Scheduling heterogeneous DAGs well requires an online solution that handles multiple resources and dependencies

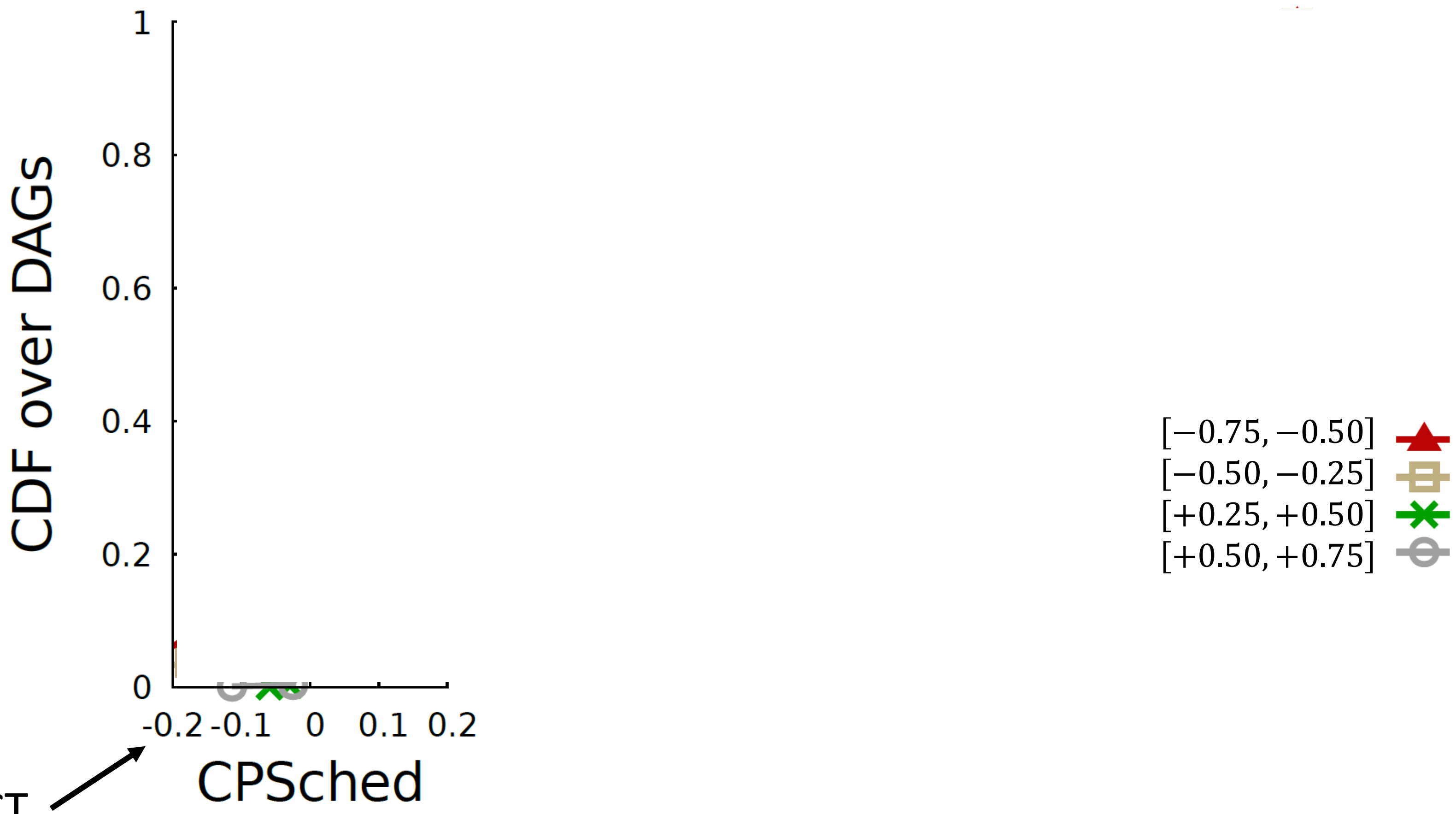
Graphene

- Offline, construct per-DAG schedule by placing troublesome tasks first
- Online, enforce schedule priority along with other heuristics
- New lower bound shows nearly optimal for half of the DAGs

Experiments show gains in job completion time, makespan, …

Graphene generalizes to DAGs in other settings

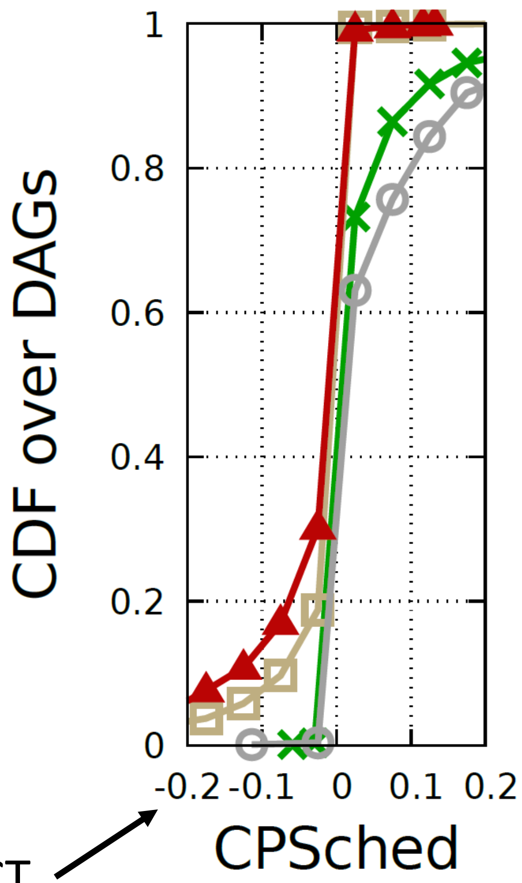
When scheduler works with erroneous task profiles



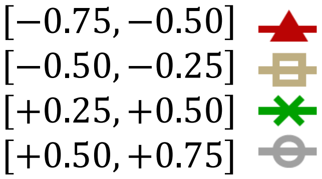
Amount of error

Fractional change in JCT

When scheduler works with erroneous task profiles

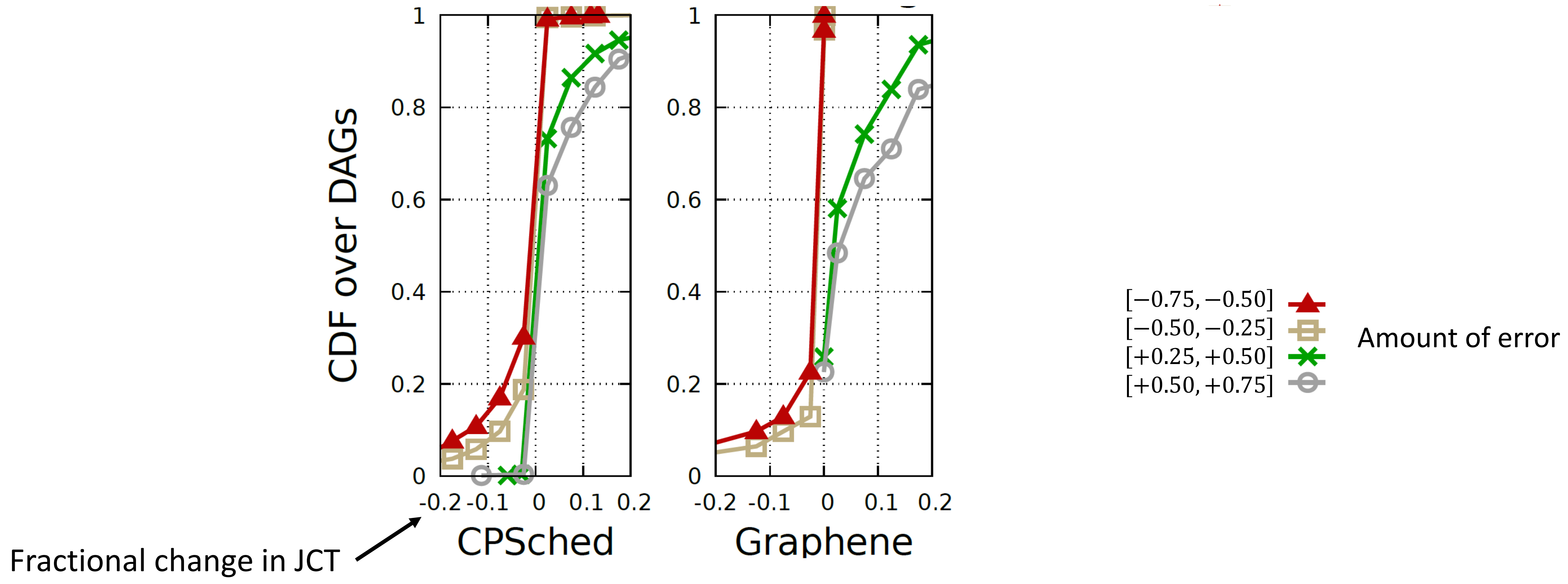


Fractional change in JCT

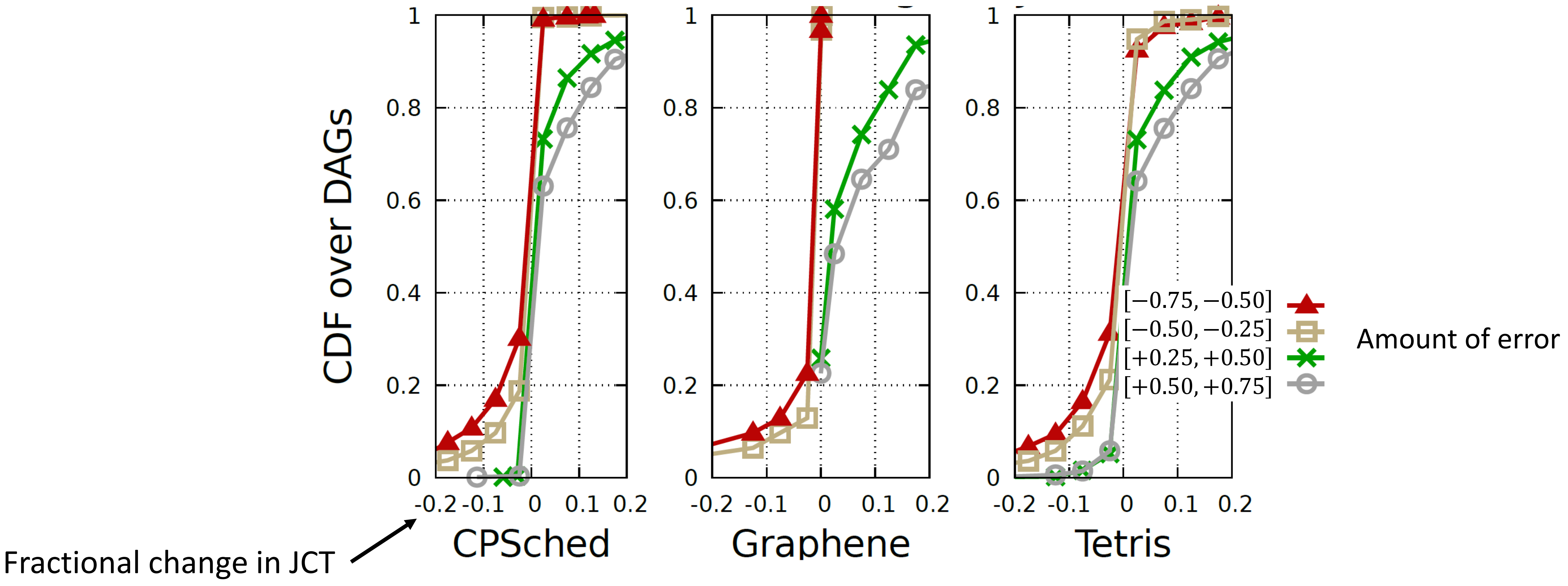


Amount of error

When scheduler works with erroneous task profiles



When scheduler works with erroneous task profiles



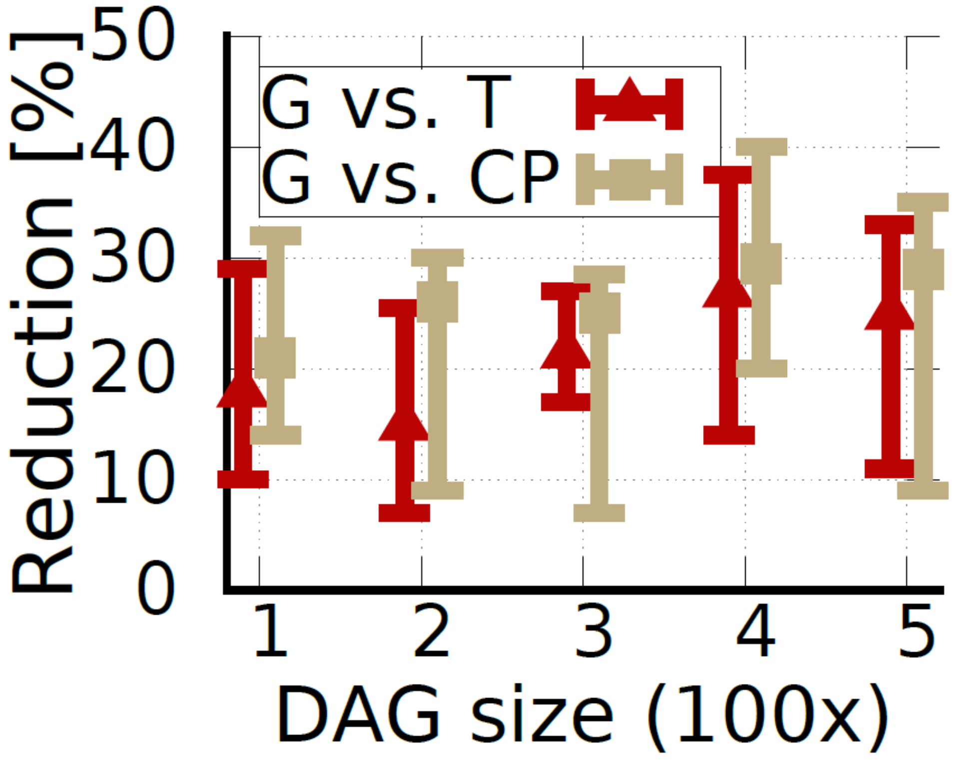
DAG annotations

G uses per-stage average duration and demands of {cpu, mem, net. disk}

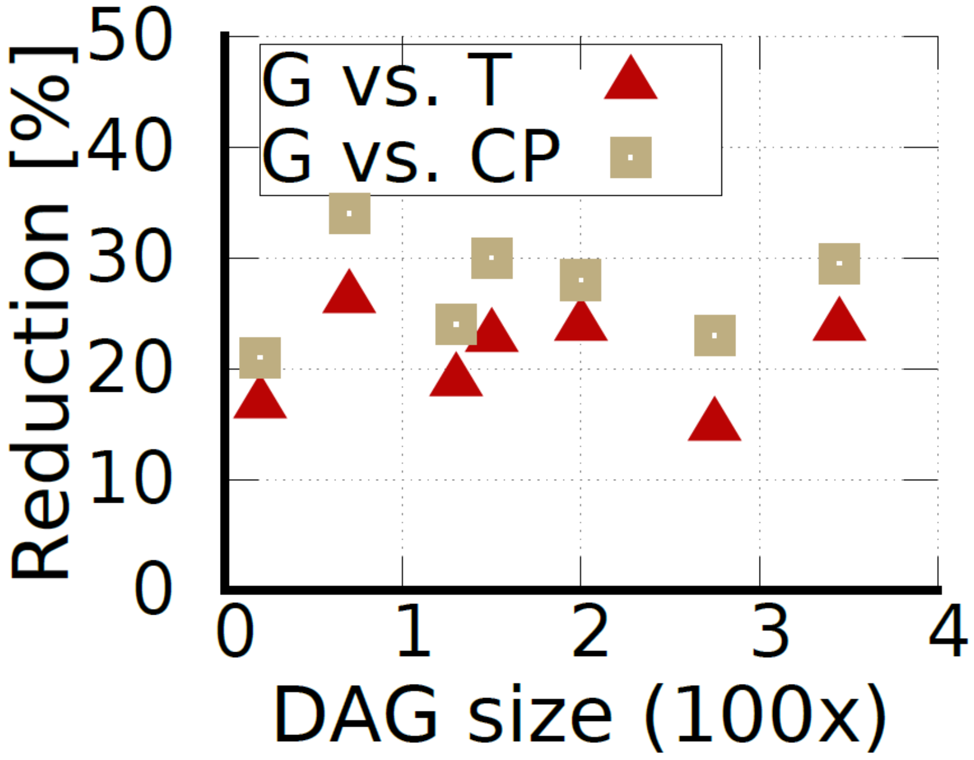
1) Almost all frameworks have user's annotate cpu and mem
2) Recurring jobs[1] have predictable profiles (correcting for input size)
3) Ongoing work on building profiles for ad-hoc jobs
    - Sample and project[2]
    - Program analysis[3]

[1] RoPE, NSDI'12; …
[2] Perforator, SOCC'16; …
[3] SPEED, POPL'09; …

Using Graphene to schedule other DAGs

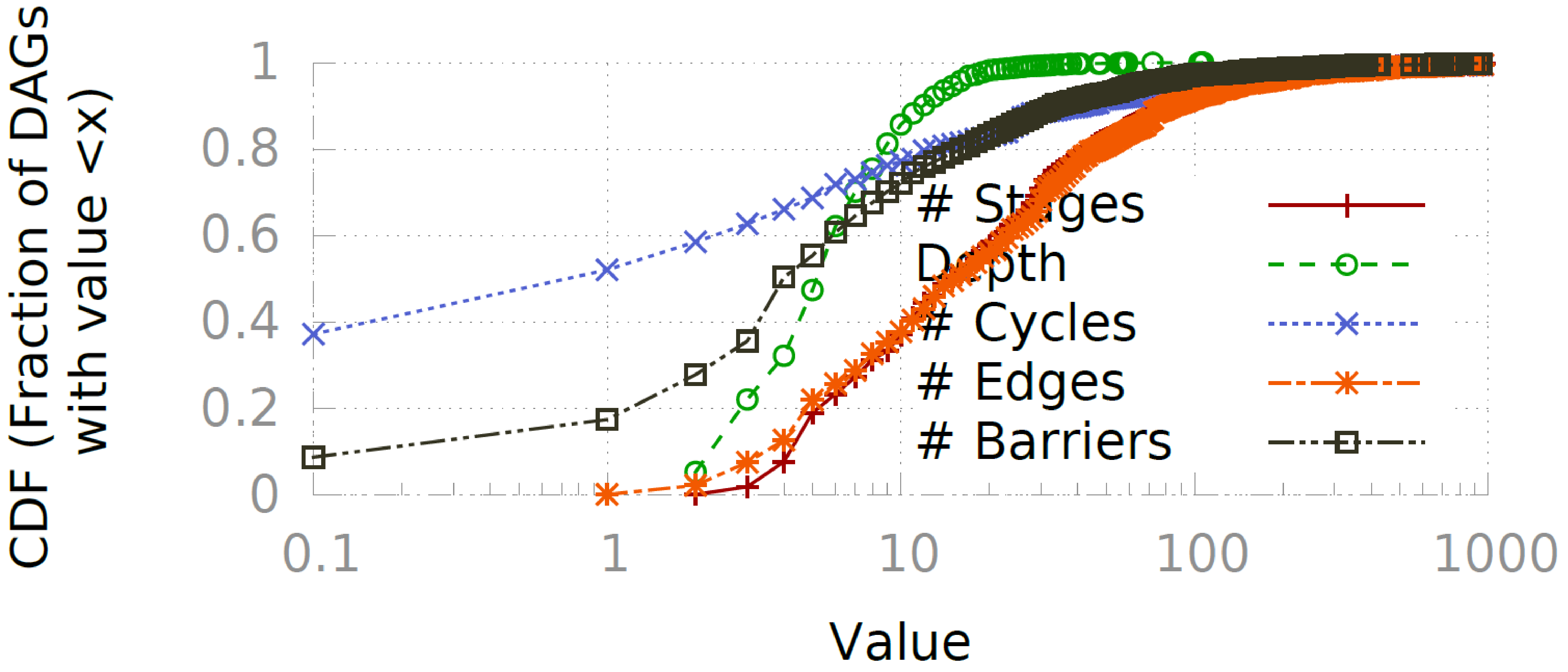


(a) Distributed Build Systems: Compilation time

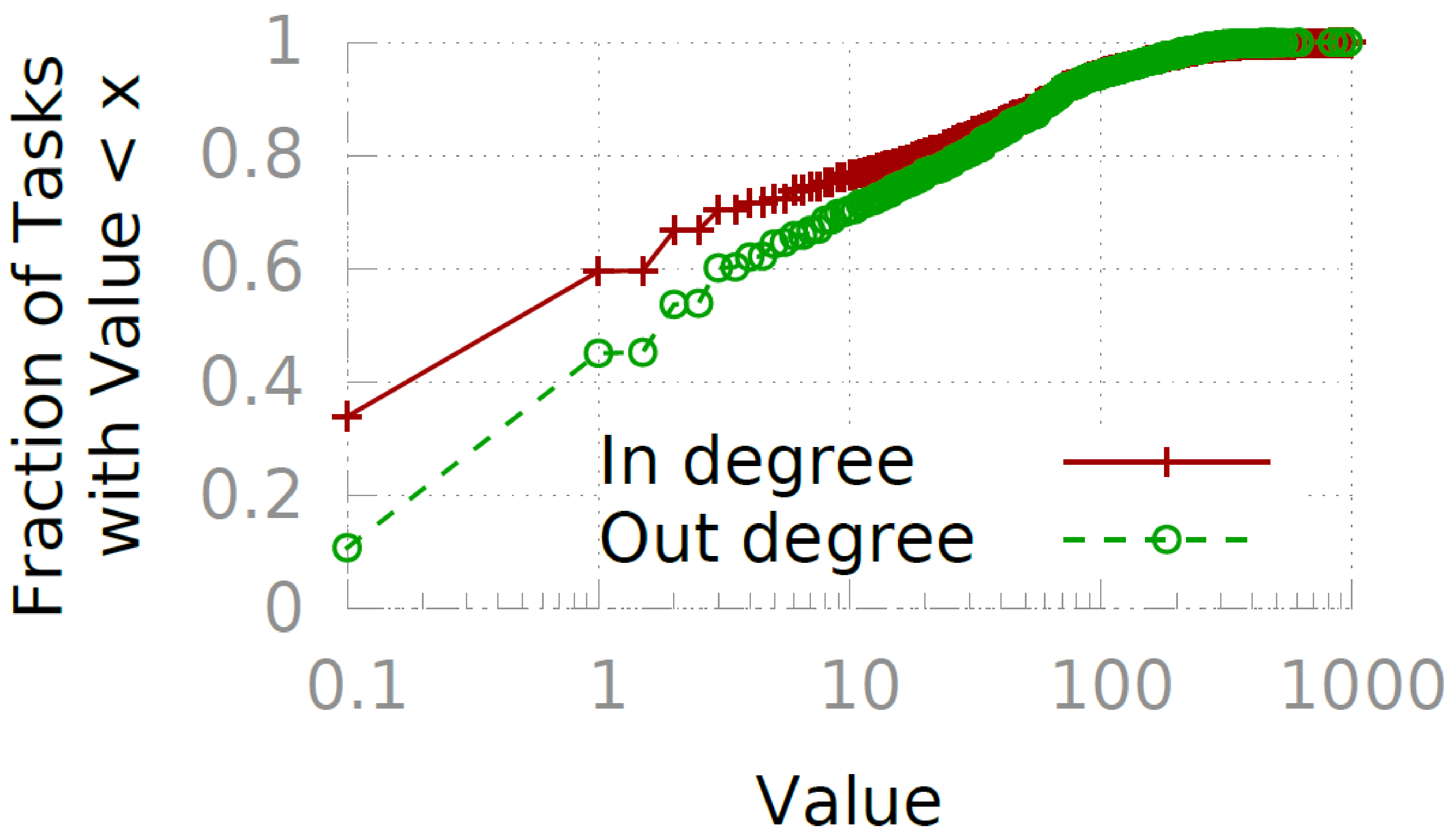


(b) Request-response workflows: Query latency

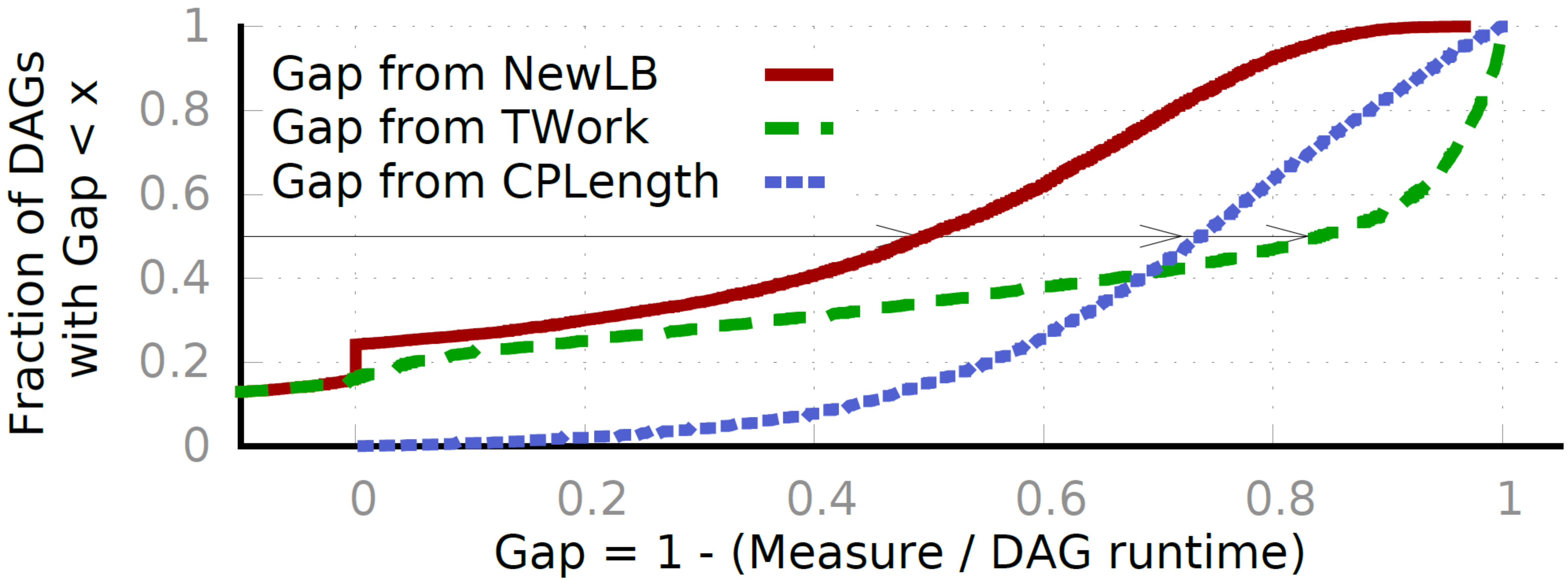
Characterizing DAGs in Cosmos clusters



Characterizing DAGs in Cosmos clusters – 2



Runtime of production DAGs



Job completion times on different workloads

Workload	50th percentile G	50th percentile T+C	50th percentile T+T	75th percentile G	75th percentile T+C	75th percentile T+T
TPC-DS	**27.8**	4.1	6.5	**45.7**	8.9	16.6
TPC-H	**30.5**	3.8	8.9	**48.3**	7.7	15.0
BigBench	**25.0**	6.4	6.2	**33.3**	21.7	18.5
E-Hive	**19.0**	1.0	5.8	**29.7**	4.5	14.2

G stands for GRAPHENE. T+C and T+T denote `Tez + CP` and `Tez + Tetris` respectively (see §7.1). The improvements are relative to `Tez`.

Makespan

Workload	Tez+CP	Tez+Tetris	GRAPHENE
TPC-DS	+2.1%	+8.2%	+30.9%
TPC-H	+4.3%	+9.6%	+27.5%

Fairness

Workload	Scheme	2Q vs. 1Q Perf. Gap	Jain's fairness index		
			10s	60s	240s
TPC-DS	Tez	−13%	0.82	0.86	0.88
	Tez+DRF	−12%	0.85	0.89	0.90
	Tez+Tetris	−10%	0.77	0.81	0.92
	GRAPHENE	+2%	0.72	0.83	0.89

Comparison with other alternatives

		25^{th}	50^{th}	75^{th}	90^{th}
GRAPHENE		**7**	**25**	**57**	**74**
Random		−2	0	1	4
Crit.Path	Fit cpu/mem	−2	0	2	1
	Fit all	1	4	13	16
Tetris	Fit all	0	7	29	42
Strip Part.	Fit all	0	1	12	27
Coffman-Graham.	Fit all	0	1	12	26
	Fit cpu/mem	−2	0	0	2

Online Pseudocode

Func: FindAppropriateTasksForMachine:
Input: $\mathbf{m}$: vector of available resources at machine; $\mathcal{J}$: set of jobs with task details$\{t_{\text{duration}}, \mathbf{t}_{\text{demands}}, t_{\text{priScore}}\}$; `deficit`: counters for fairness;
Parameters: κ: unfairness bound; rp: remote penalty
Output: S, the set of tasks to be allocated on the machine

```
S ← ∅
while true do
    foreach task t do
        {pScore_t, oScore_t} ← {0, 0}
        rPenalty_t ← t is locality sensitive ? rp : 1
        if t_demands ≤ m // fits? then
            pScore_t ← (m · t_demands) rPenalty_t // dot product
        else
            // what-if analysis: "overbook or wait".
            ∀tasks t′ affected by t running at m, let before(t′),
            after(t′) be expected completion times before and
            after placing t at m
            benefit = nextSchedOpp + t_duration − after(t)
            cost = Σ_{aff. tasks t′} (after(t′) − before(t′))
            if benefit > cost then oScore_t = benefit − cost;
        job j ∋ t, srpt_j ← Σ_{pending u∈j} u_duration * |u_demands|
        ┌──────────────────────────────────────────────────────────┐
        │ perfScore_t ← t_priScore {pScore_t, oScore_t} − η srpt_j │
        └──────────────────────────────────────────────────────────┘
    t^best ← arg max{perfScore_t | t} // task with highest perf score
    if t^best = ∅ then break // no new task can be scheduled on this machine;
    g′ ← jobgroup with highest deficit counter
    ┌──────────────────────────────────────────────────────────────────────┐
    │ if deficit_g′ ≥ κC then t^best ← arg max{perfScore_t | t ∈ g′};      │
    └──────────────────────────────────────────────────────────────────────┘
    S ← S ∪ t^best
    m ← [m − t^best_demands]_0+
    deficit_g ← deficit_g +
    ┌──────────────────────────────────────────────────────────────────────┐
    │ factor(t^best_demands) * { fairShare_g − 1   t ∈ jobgroup g          │
    │                          { fairShare_g       otherwise               │
    └──────────────────────────────────────────────────────────────────────┘
```