Efficient Partitioning of Sporadic Real-Time Tasks with Shared Resources and Spin Locks

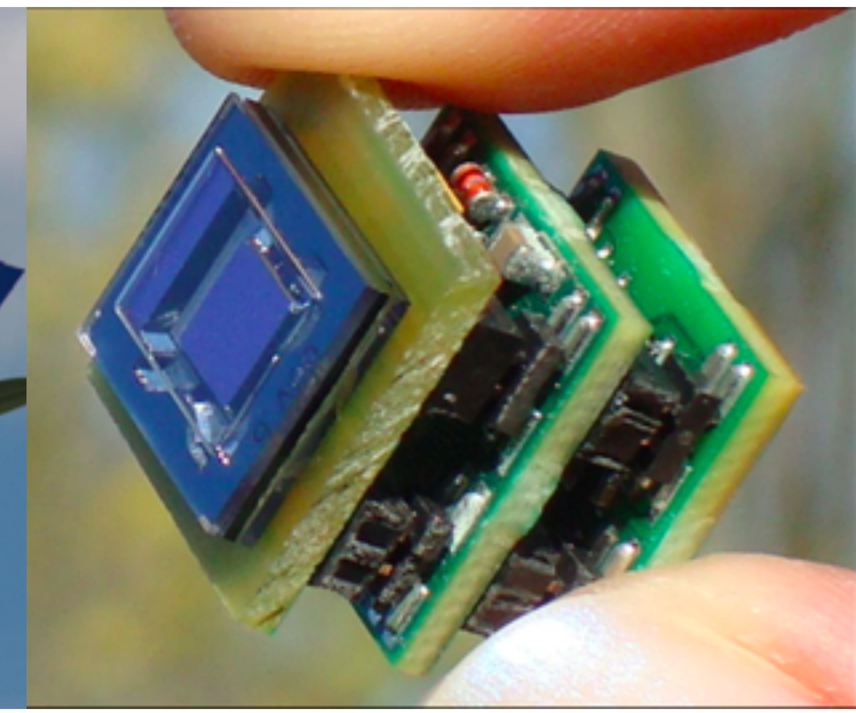
Alexander Wieder
Björn B. Brandenburg

Max Planck Institute for Software Systems
(MPI-SWS)

Max
Planck
Institute
for
Software Systems

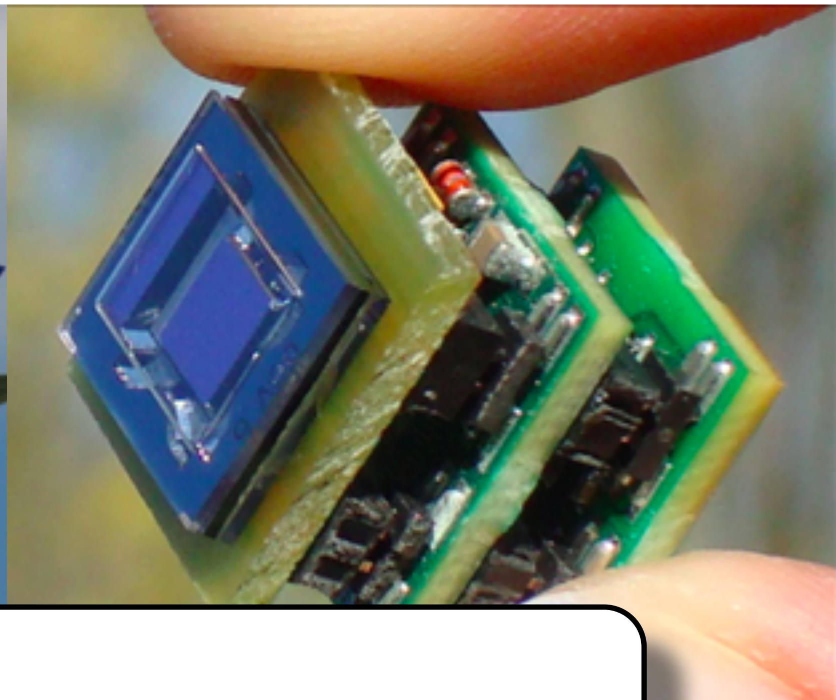
8th IEEE International Symposium on Industrial Embedded Systems
Porto, Portugal
19.06.2013

Motivation



- space, weight and power constraints
- tasks with real-time requirements
- multi-core architectures
- shared resources protected by spin locks

Motivation



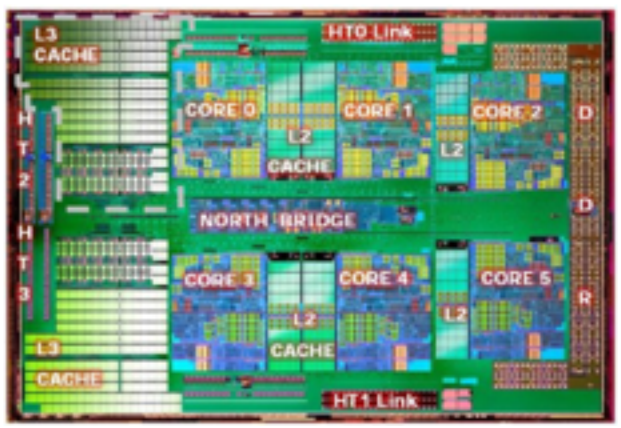
Our Focus: Processor Efficiency

Make best use of multi-core architectures despite shared resources protected by spin locks

Example: Autosar

AUTOSAR

multicore architectures:

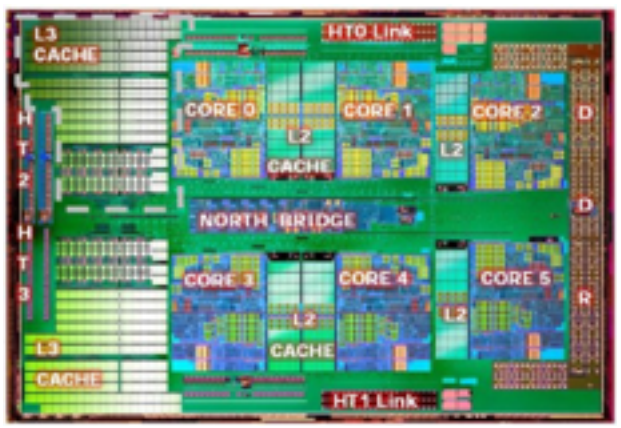


partitioned
fixed-priority
scheduling

Example: Autosar

AUTOSAR

multicore architectures:



partitioned
fixed-priority
scheduling

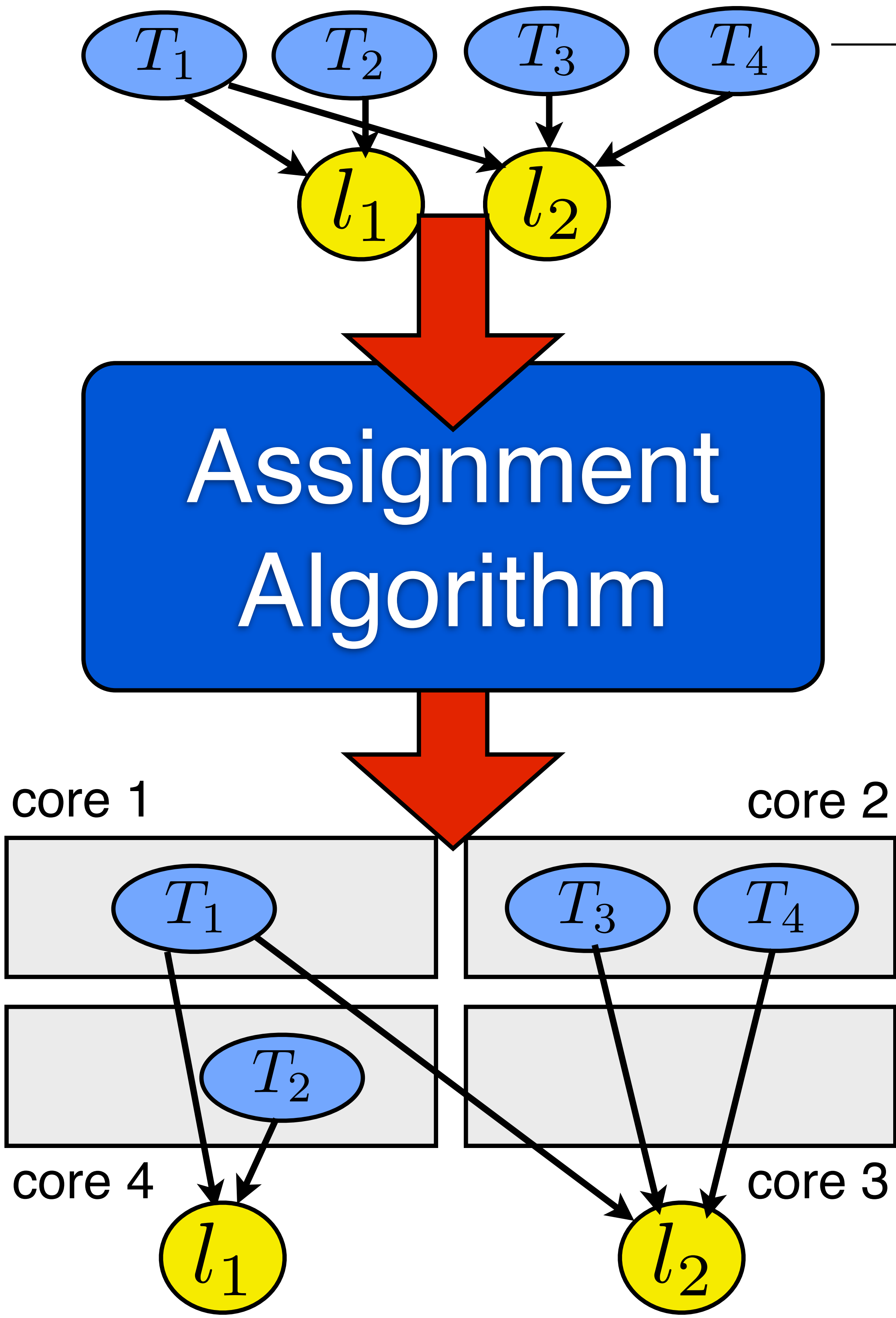
shared resources:

- sensors
- communication bus
- kernel objects

global resources:
non-preemptable spin locks

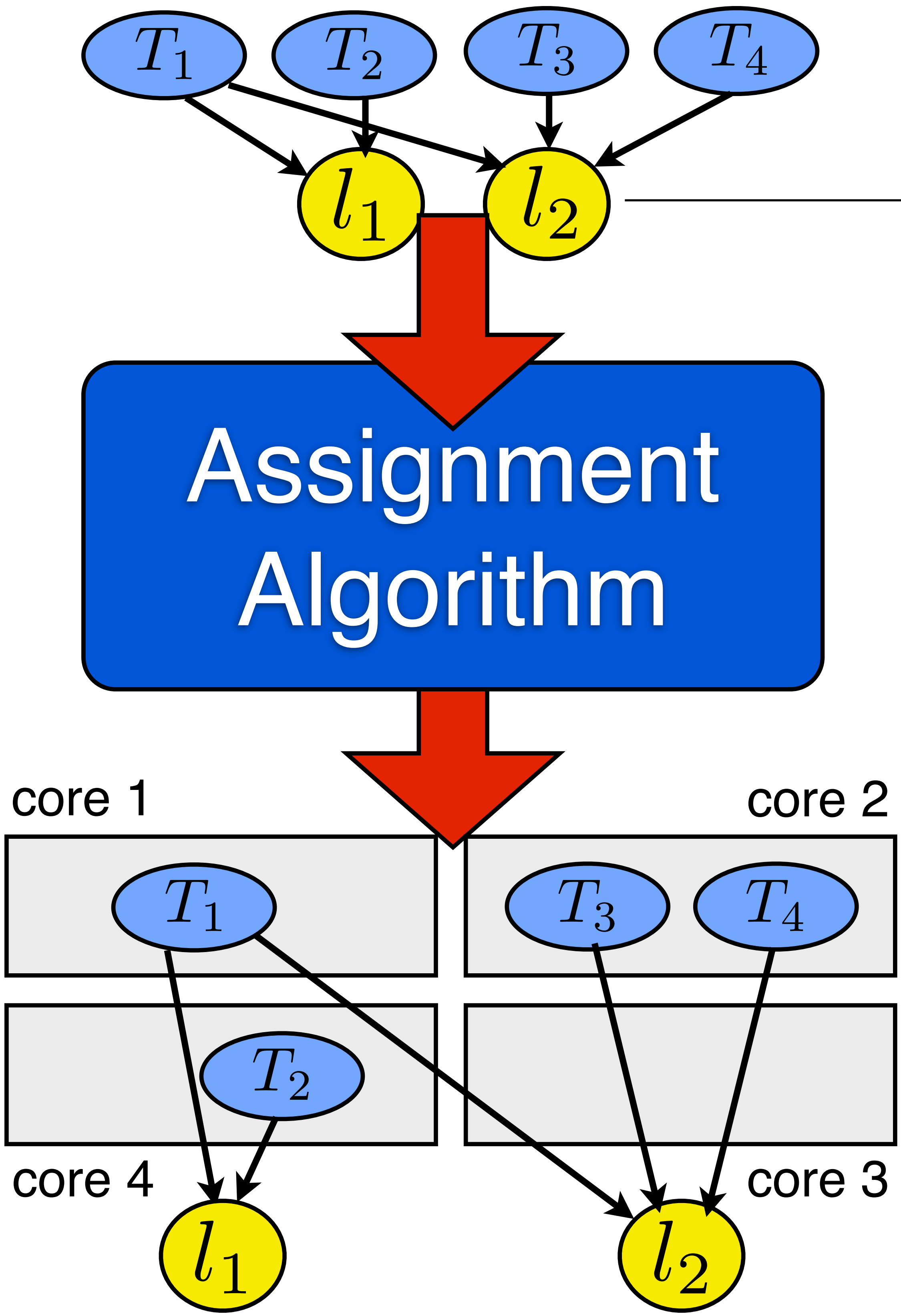
local resources:
Priority Ceiling Protocol (PCP)

The Task Assignment Problem



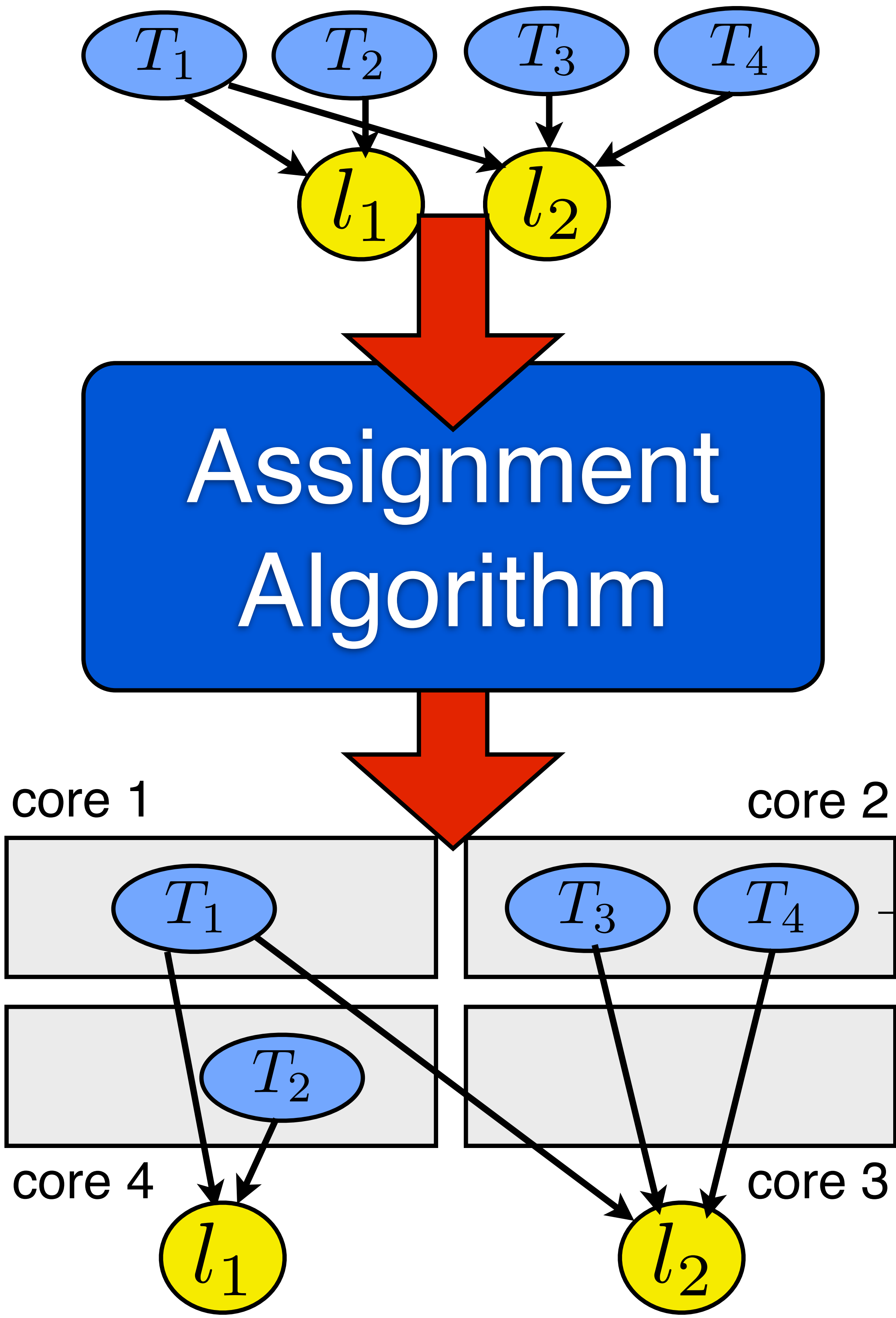
sporadic real-time tasks

The Task Assignment Problem



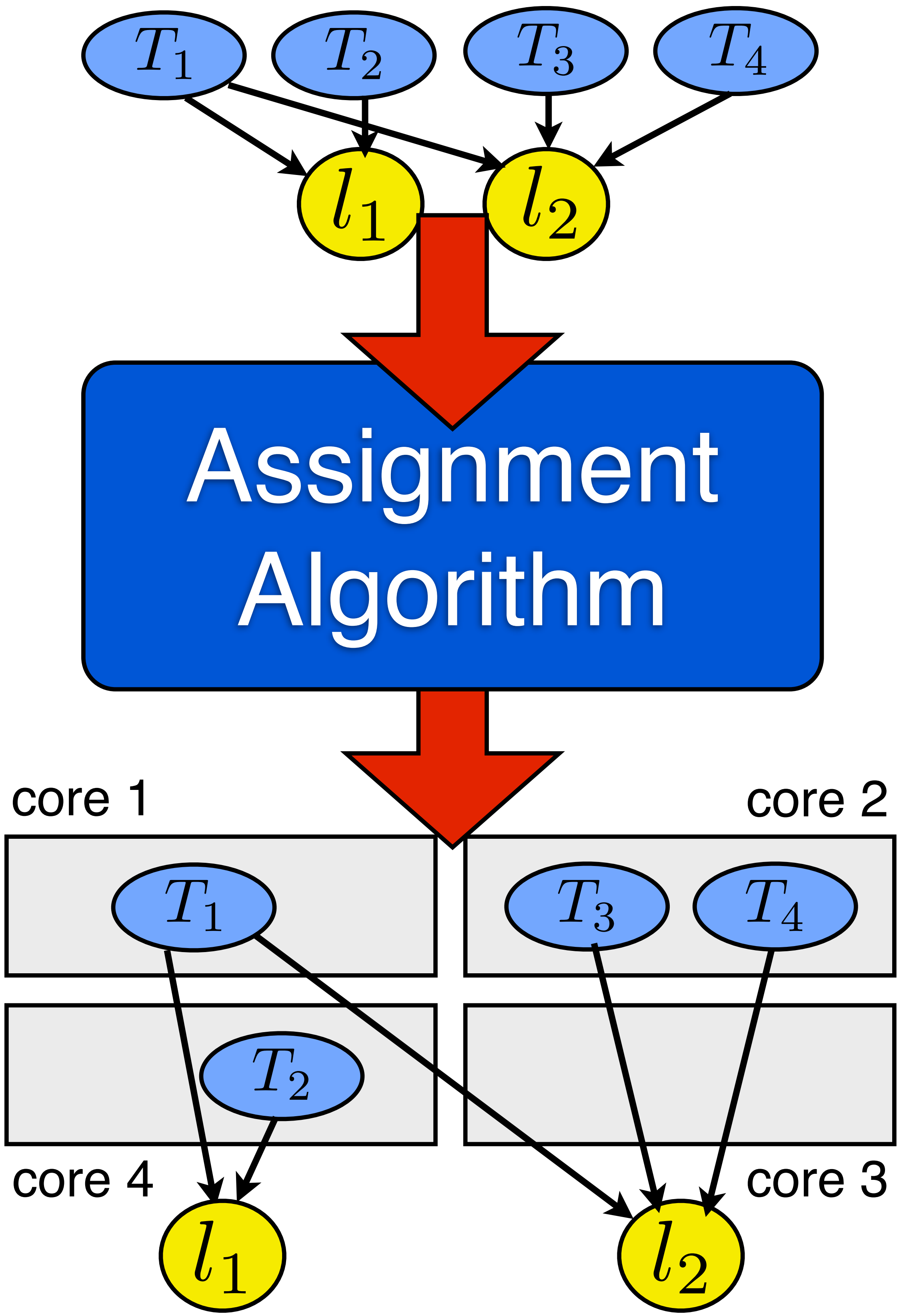
shared resources protected by **spin locks**

The Task Assignment Problem



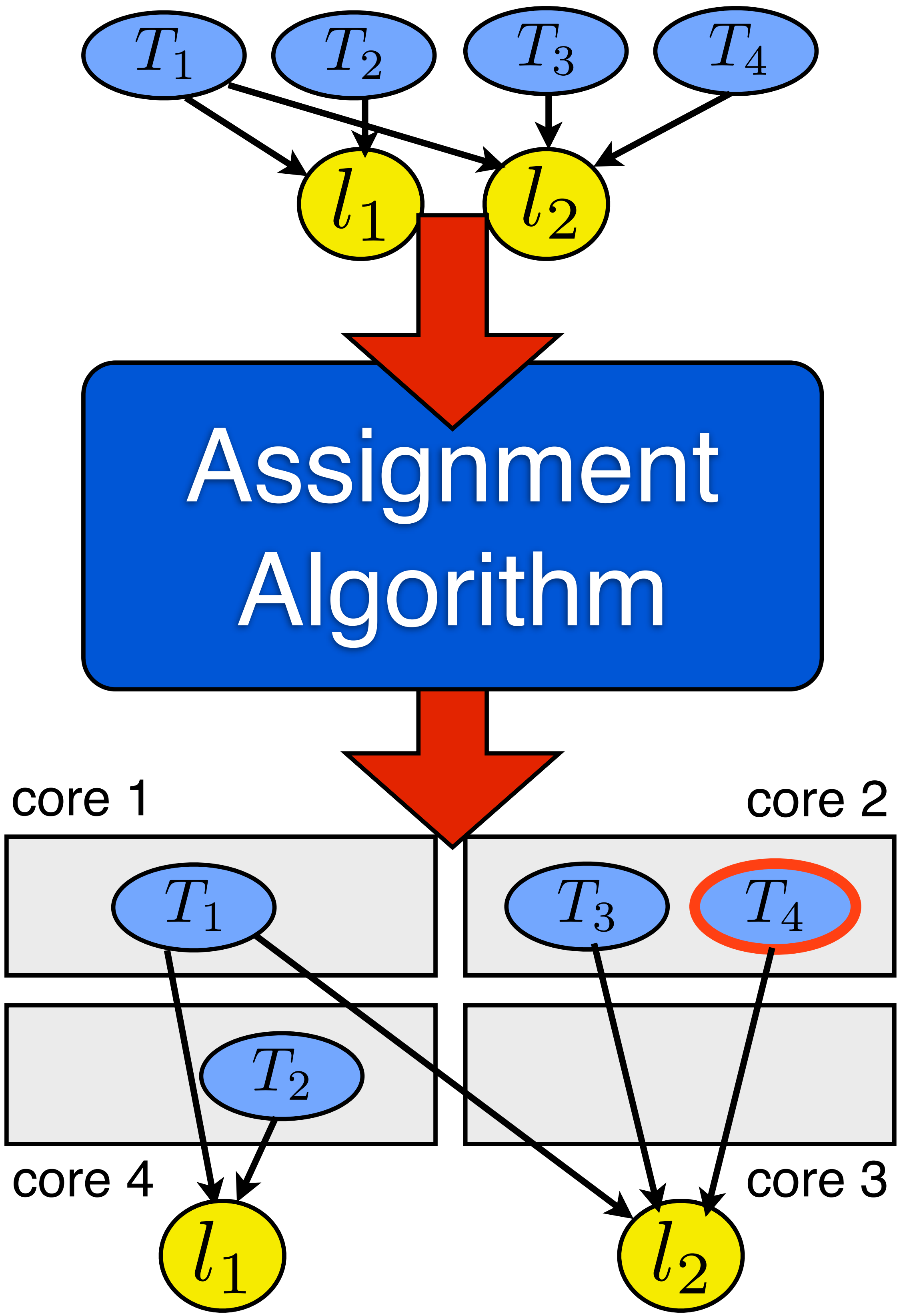
tasks mapped to processor cores s.t. **all deadlines met**

The Task Assignment Problem



Challenge:
Task sets using spin locks

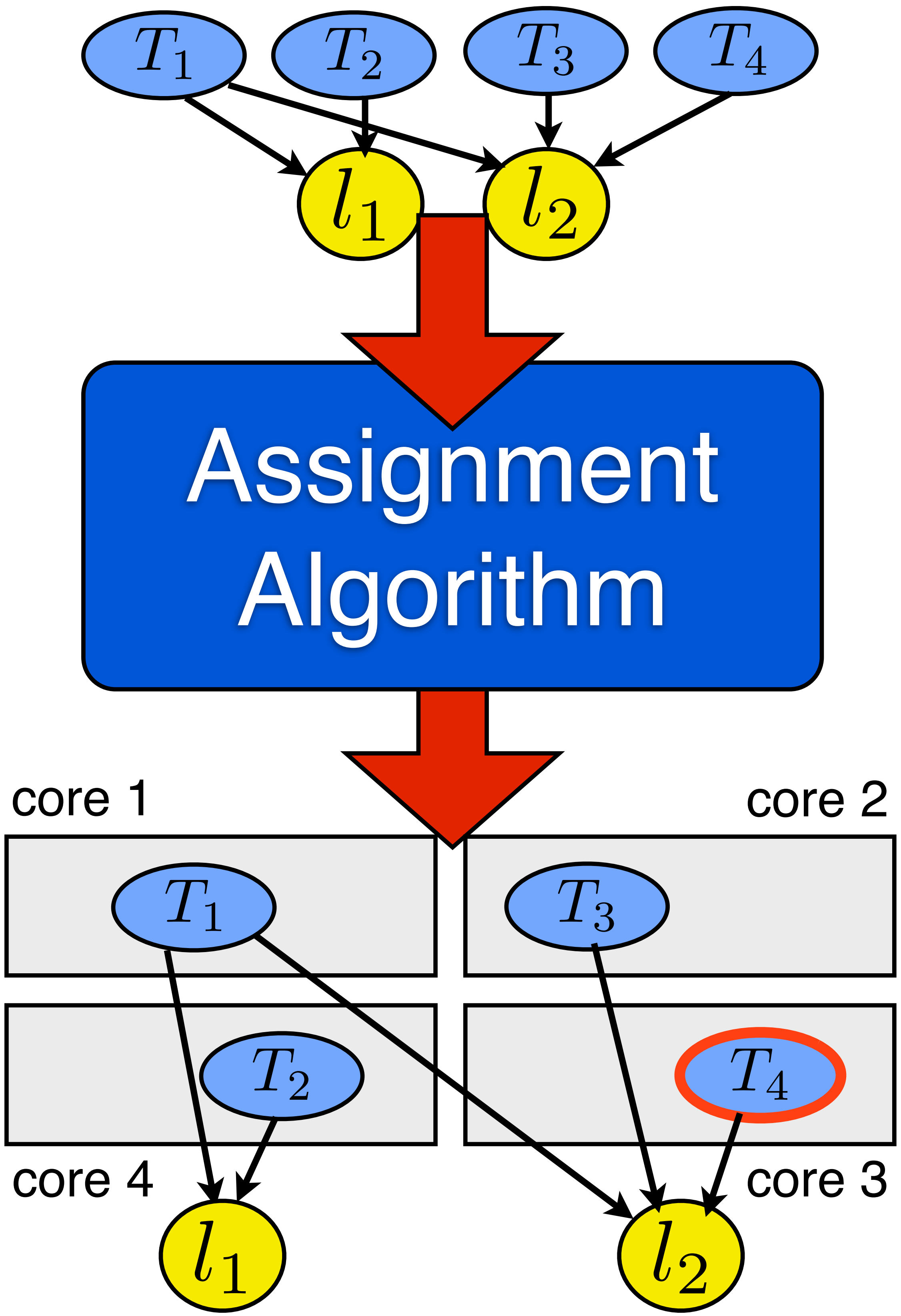
The Task Assignment Problem



Challenge:
Task sets using spin locks

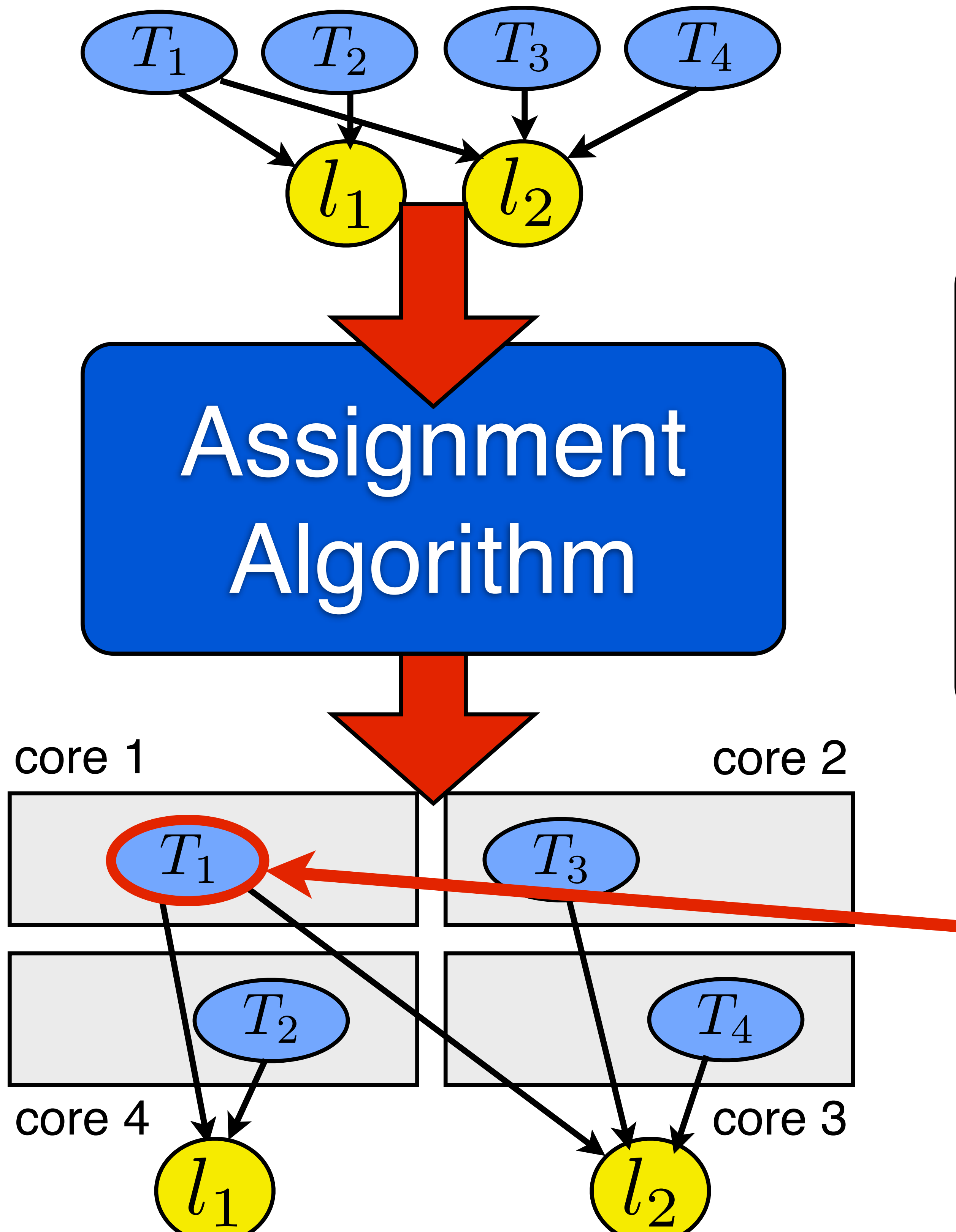
Suppose T_4 assigned to core 3 instead…

The Task Assignment Problem



Challenge: Task sets using spin locks

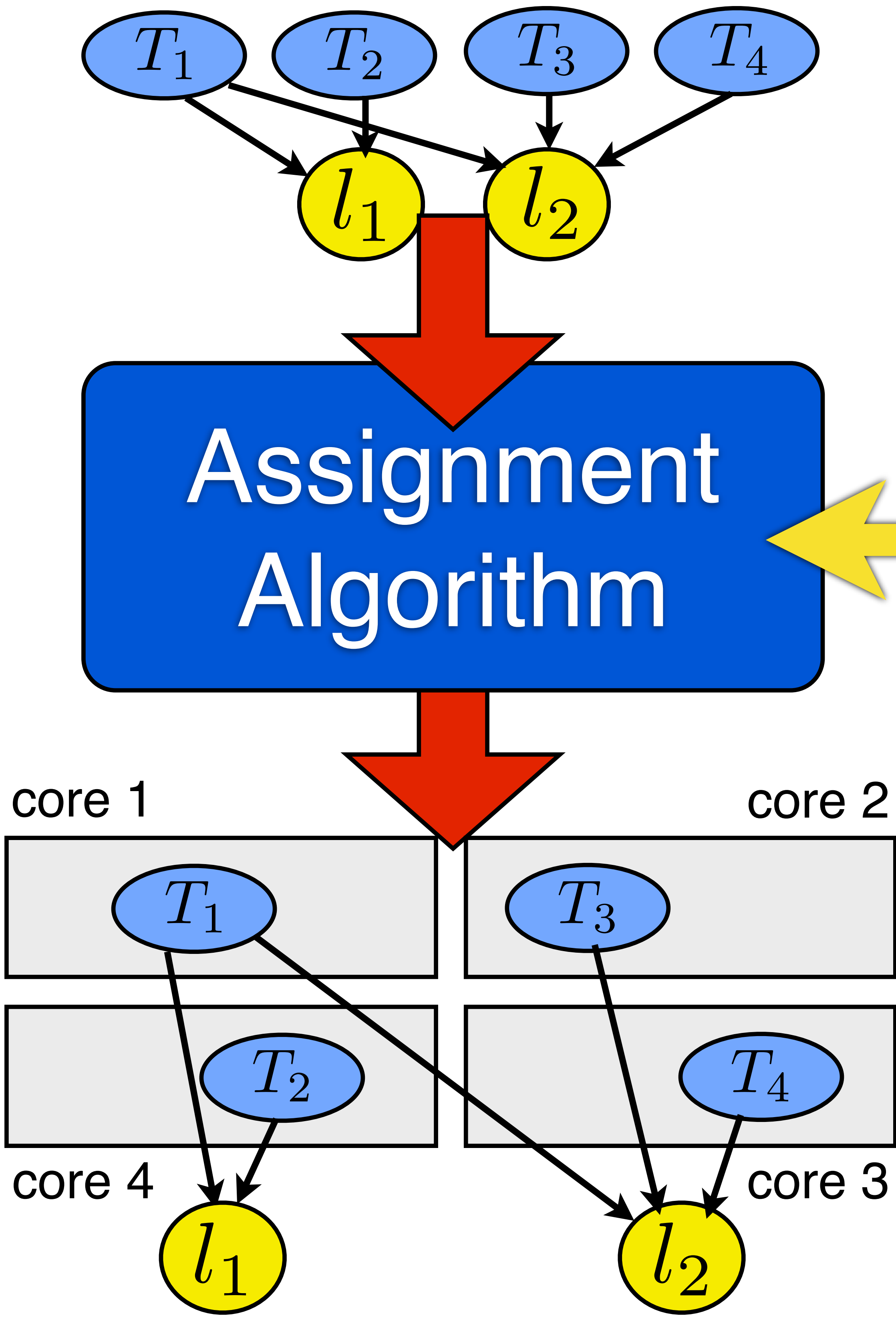
The Task Assignment Problem



Challenge: Task sets using spin locks

Remote task can miss deadline!

The Task Assignment Problem



How **efficient** are prior heuristics...?

This Paper

Observation

Part I

Optimality matters!
with shared resources, potential wasted by prior heuristics

Contribution

This Paper

<u>Observation</u>

Part I

Optimality matters! with shared resources, potential wasted by prior heuristics

→

<u>Contribution</u>

Part II

Optimal ILP-based partitioning scheme for task sets with shared resources

This Paper

Observation	Contribution
Part I	Part II
Optimality matters! with shared resources, potential wasted by prior heuristics	**Optimal** ILP-based partitioning scheme for task sets with shared resources
Prior sharing-aware heuristics are complicated and brittle	

This Paper

<u>Observation</u>

Part I

Optimality matters! with shared resources, potential wasted by prior heuristics

Prior sharing-aware heuristics are complicated and brittle

<u>Contribution</u>

Part II

Optimal ILP-based partitioning scheme for task sets with shared resources

Part III

Greedy Slacker: simple and robust heuristic

Task Model

- sporadic tasks: $T_i : (e_i, d_i, p_i)$
- constrained deadlines: $d_i \leq p_i$
- shared resources accessed in mutual exclusion
- coordinating resource access:
  - global: **non-preemptable FIFO spinlocks**
  - local: SRP (blocking equivalent to PCP)

Task Model

- sporadic tasks: $T_i : (e_i, d_i, p_i)$

- constrained deadlines: $d_i \leq p_i$

- shared resources accessed in mutual exclusion

- coordinating resource access:
  - global: **non-preemptable FIFO spinlocks**
  - local: SRP (blocking equivalent to PCP)

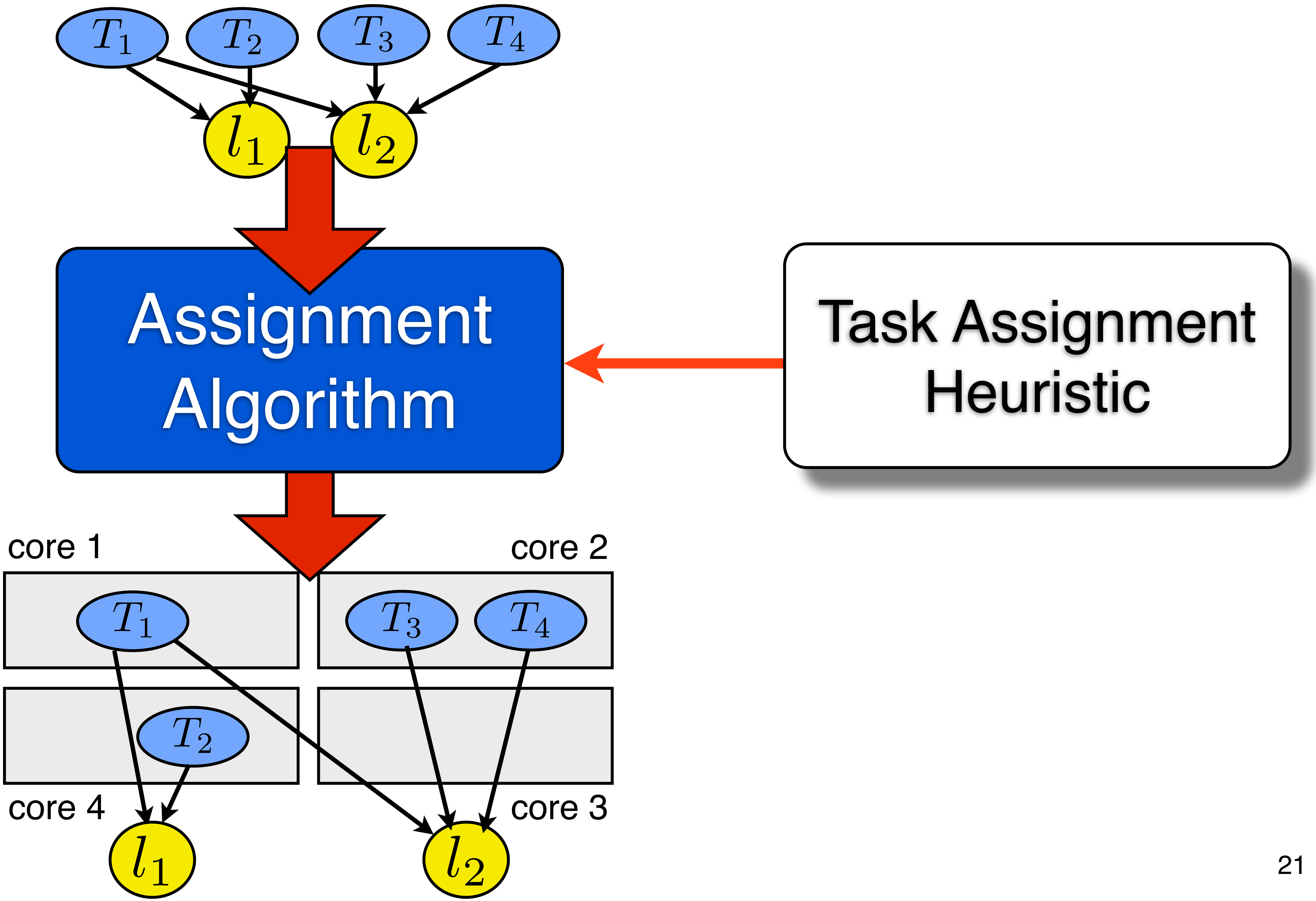
➡ Multiprocessor Stack Resource Policy [1]

[1] P. Gai, G. Lipari, and M. D. Natale, "Minimizing memory utilization of real-time task sets in single and multi-processor systems-on-a-chip," in Proc. RTSS, 2001.

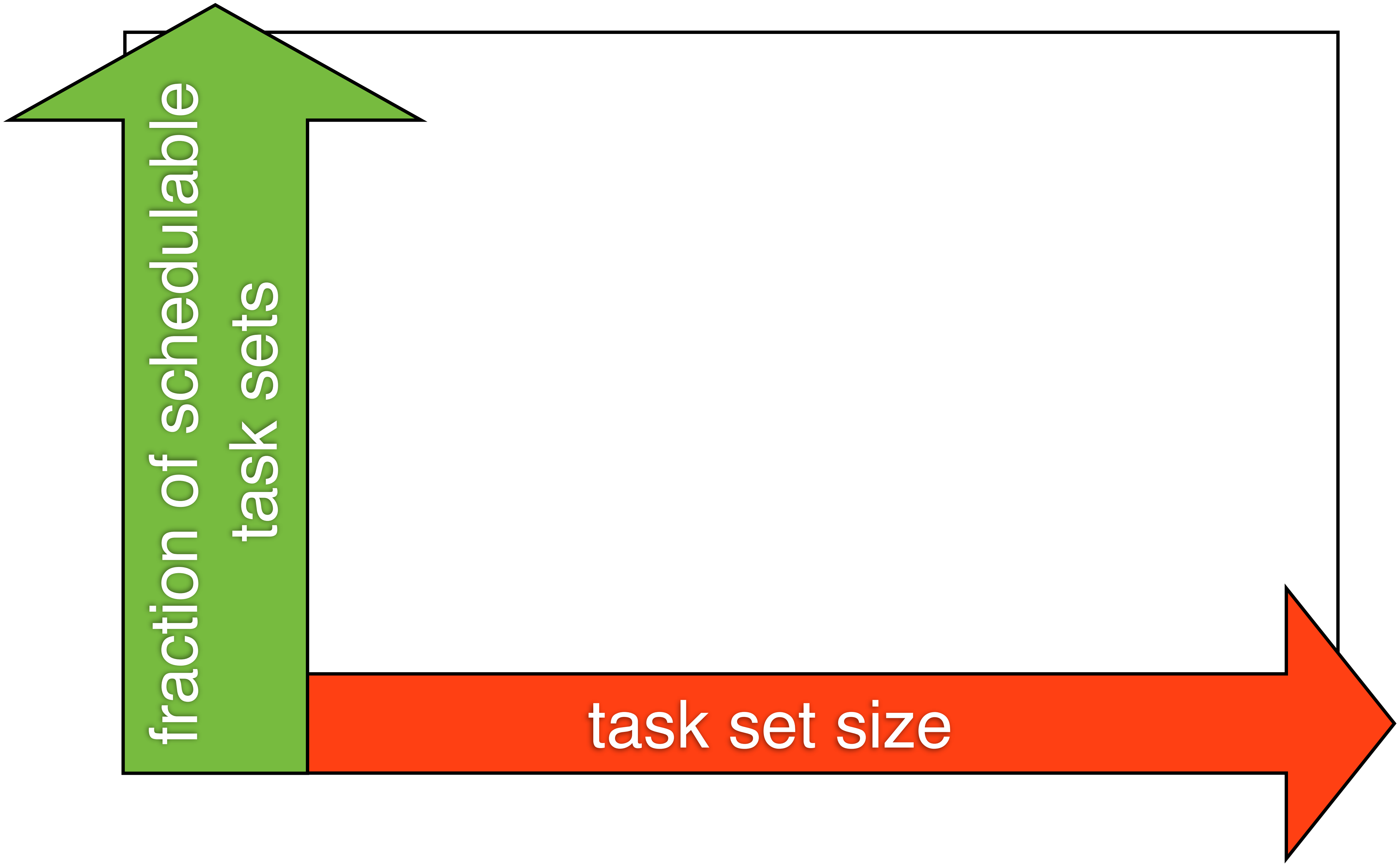
Part I

How efficient are prior heuristics?

Task Assignment with Heuristics

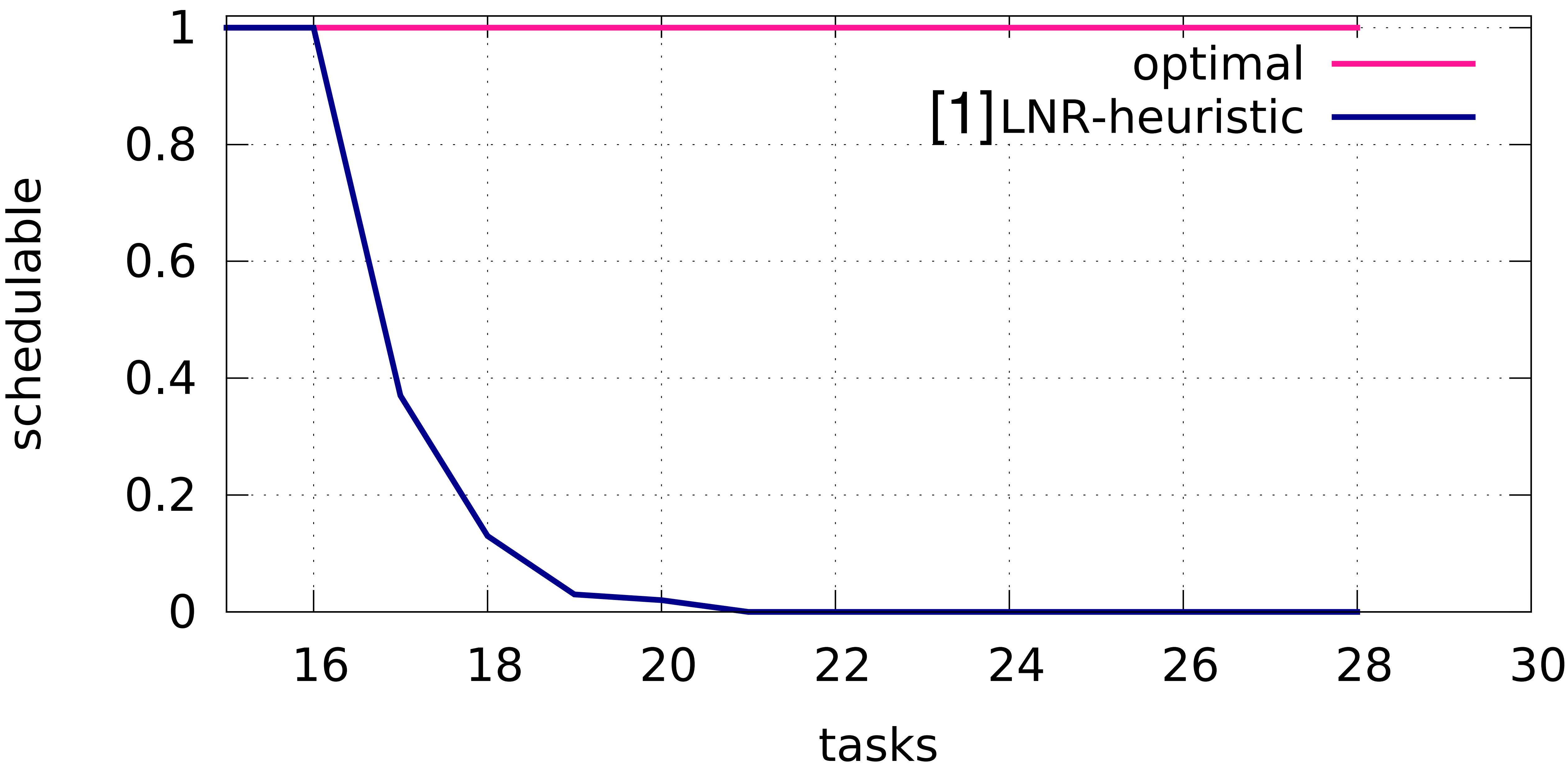


Schedulability Experiments



less idle time
more contention

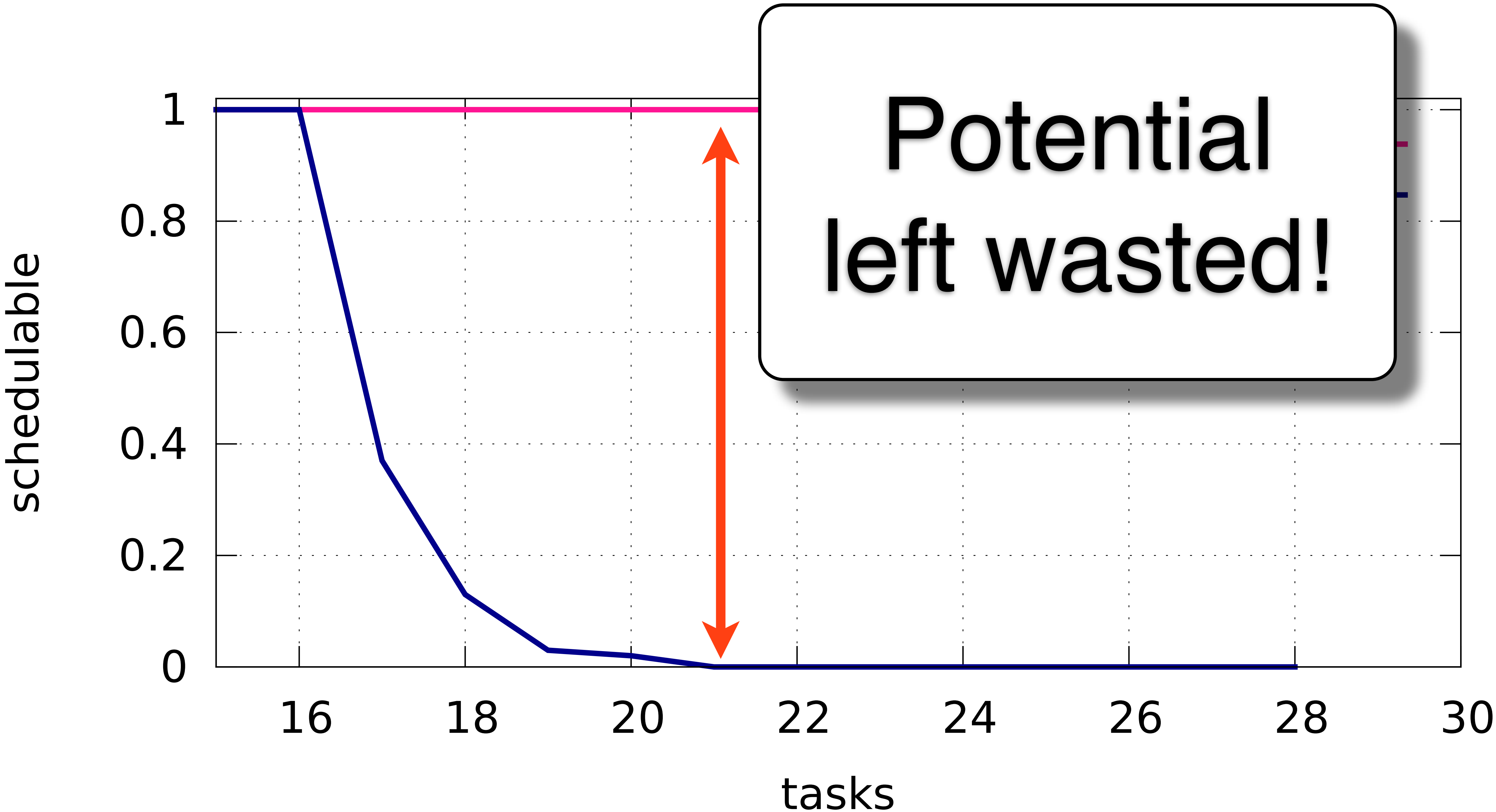
Exploring Wasted Potential



8 processors, 16 resources, critical section lengths in [1us,100us], periods in [3ms,33ms], 10% average task utilization

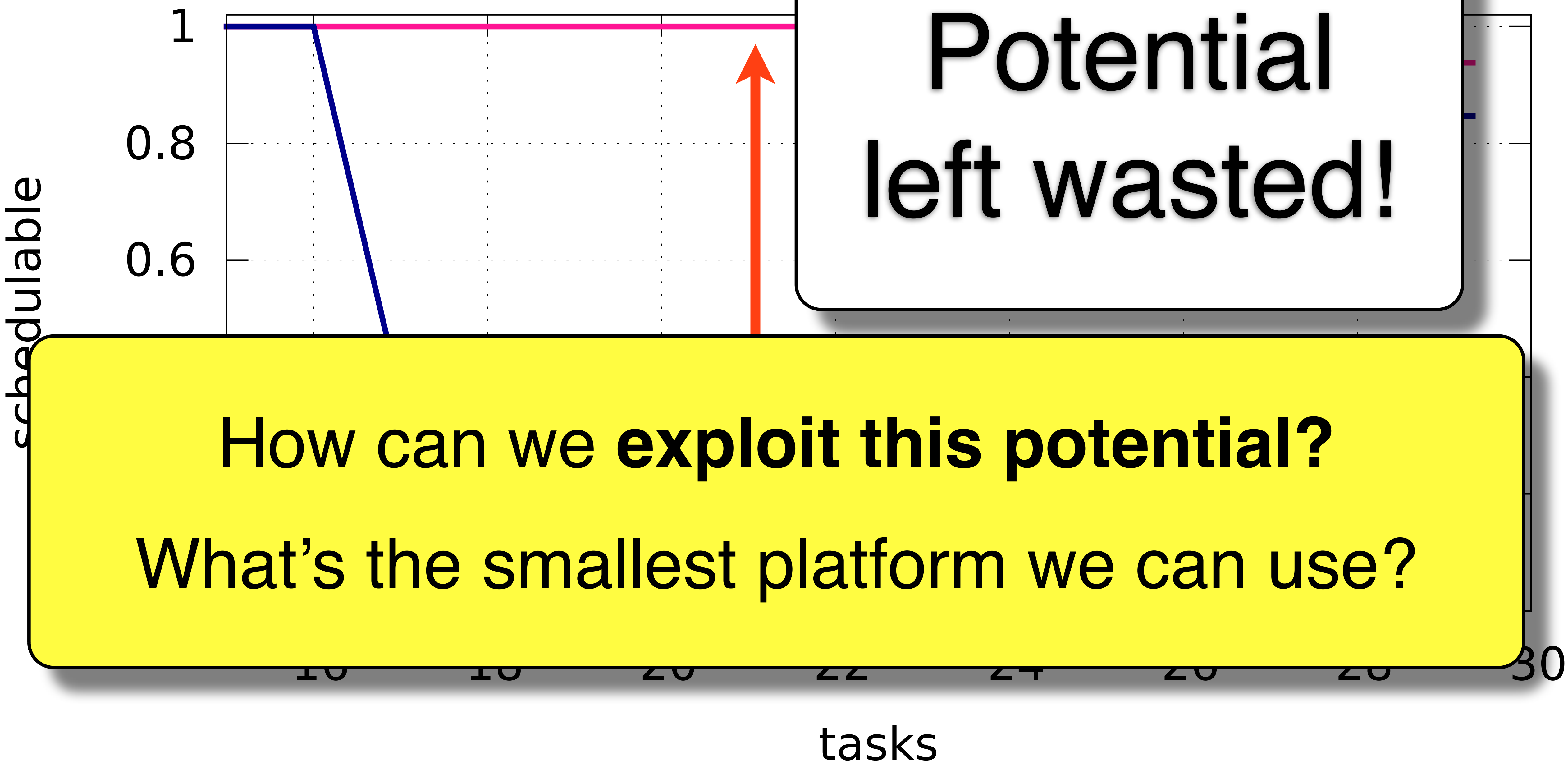
[1] K. Lakshmanan, D. de Niz, and R. Rajkumar, "Coordinated task scheduling, allocation and synchronization on multiprocessors," in Proc. RTSS, 2009.

Exploring Wasted Potential



8 processors, 16 resources, critical section lengths in [1us,100us], periods in [3ms,33ms], 10% average task utilization

Exploring Wasted Potential



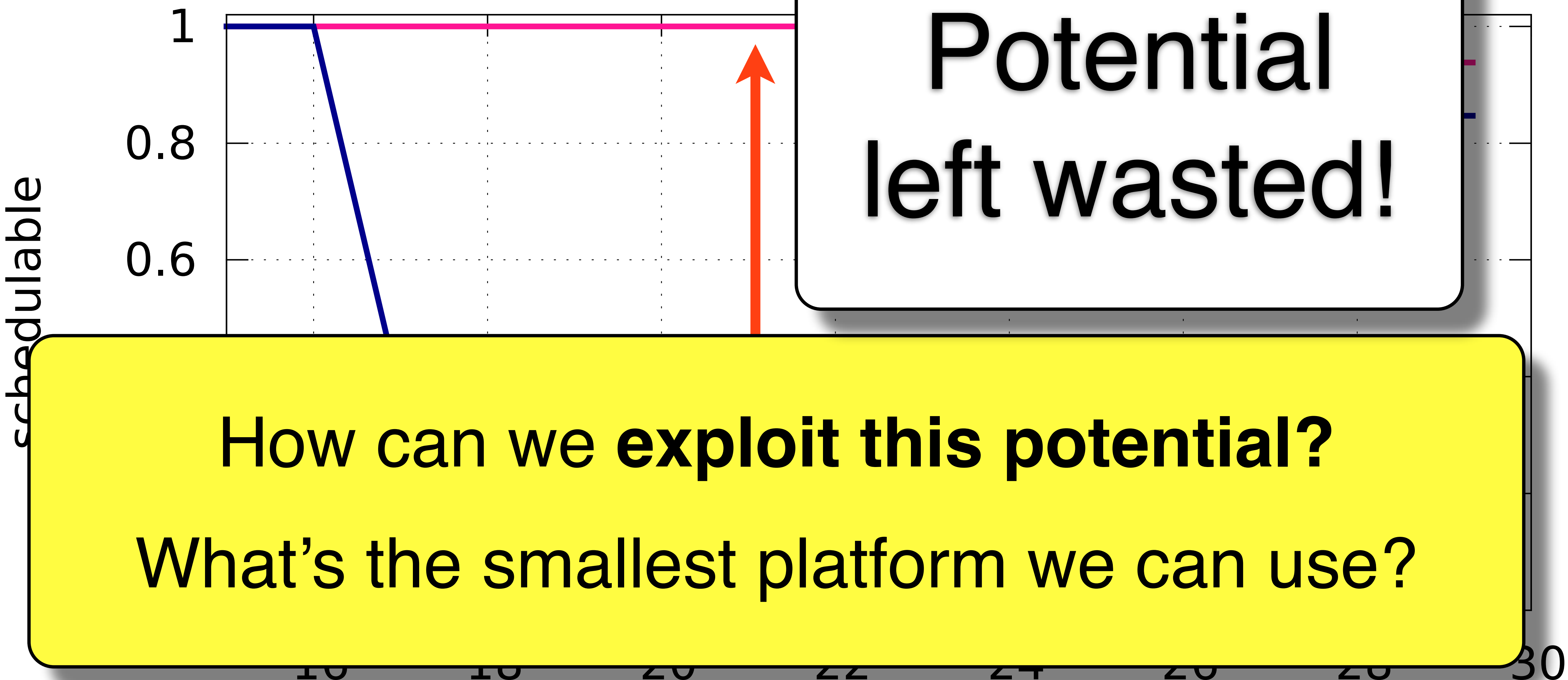
Potential left wasted!

How can we **exploit this potential?**

What's the smallest platform we can use?

8 processors, 16 resources, critical section lengths in [1us,100us], periods in [3ms,33ms], 10% average task utilization

Exploring Wasted Potential



Potential left wasted!

How can we **exploit this potential?**

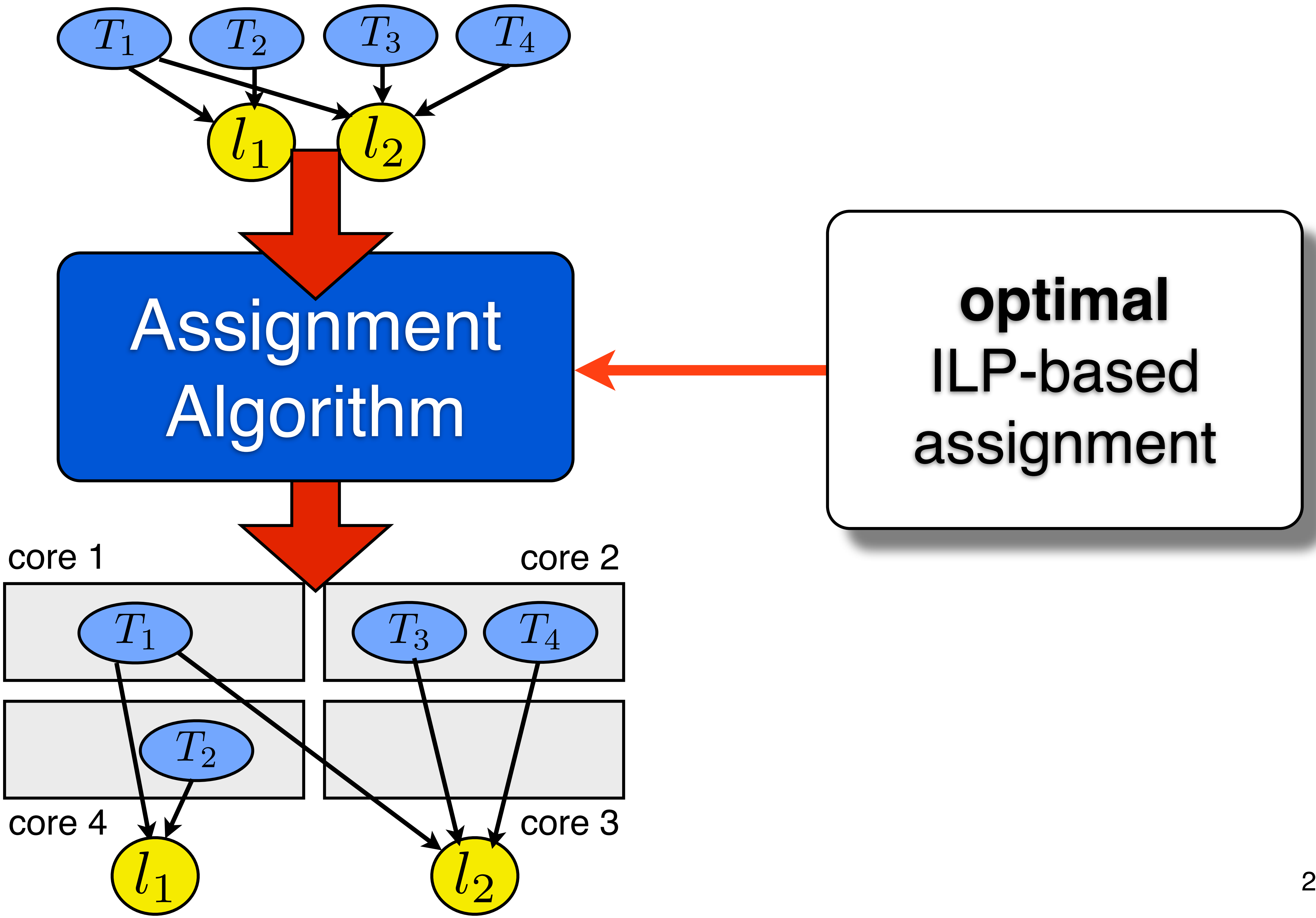
What’s the smallest platform we can use?

With shared resources,
optimal partitioning matters!

Part II

An Optimal ILP-based Partitioning Scheme

Optimal Task Assignment



Optimality

What is an optimal partitioning scheme?

If a valid partitioning
under a given analysis
exists,
a valid partitioning can be found.

Optimality

What is an optimal partitioning scheme?

If a valid partitioning
under a given analysis
exists,
a valid partitioning can be found.

all tasks claimed **schedulable** by analysis
(= no task misses a deadline)

Optimality

What is an optimal partitioning scheme?

> If a valid partitioning
> under a given analysis
> exists,
> a valid partitioning can be found.

We use the MSRP blocking analysis from Gai, Lipari, Di Natale (2001).

Basic ILP Model

Integer Linear Programming model encodes for a **fixed number of processors**:

- task assignment
- priority assignment
- constraints to enforce valid assignments

Basic ILP Model

Integer Linear Programming model encodes for a **fixed number of processors**:

- task assignment
- priority assignment
- constraints to enforce valid assignments

We need to encode the MSRP blocking analysis into the ILP!

Encoding Blocking in ILP

$R_i =$ own execution $+$

interference

Classic fixed-priority response-time analysis [1]

[1] S. Baruah and E. Bini, “Partitioned scheduling of sporadic task systems: An ILP based approach,” in Proc. DASIP, 2008.

Encoding Blocking in ILP

Blocking analysis from Gai et al.:

$R_i =$ own execution $+$ spinning $+$ own local blocking $+$ interference

Blocking under classic MSRP analysis from Gai et al.

Encoding Blocking in ILP

Blocking analysis from Gai et al.:

$R_i =$ own execution $+$

spinning $+$

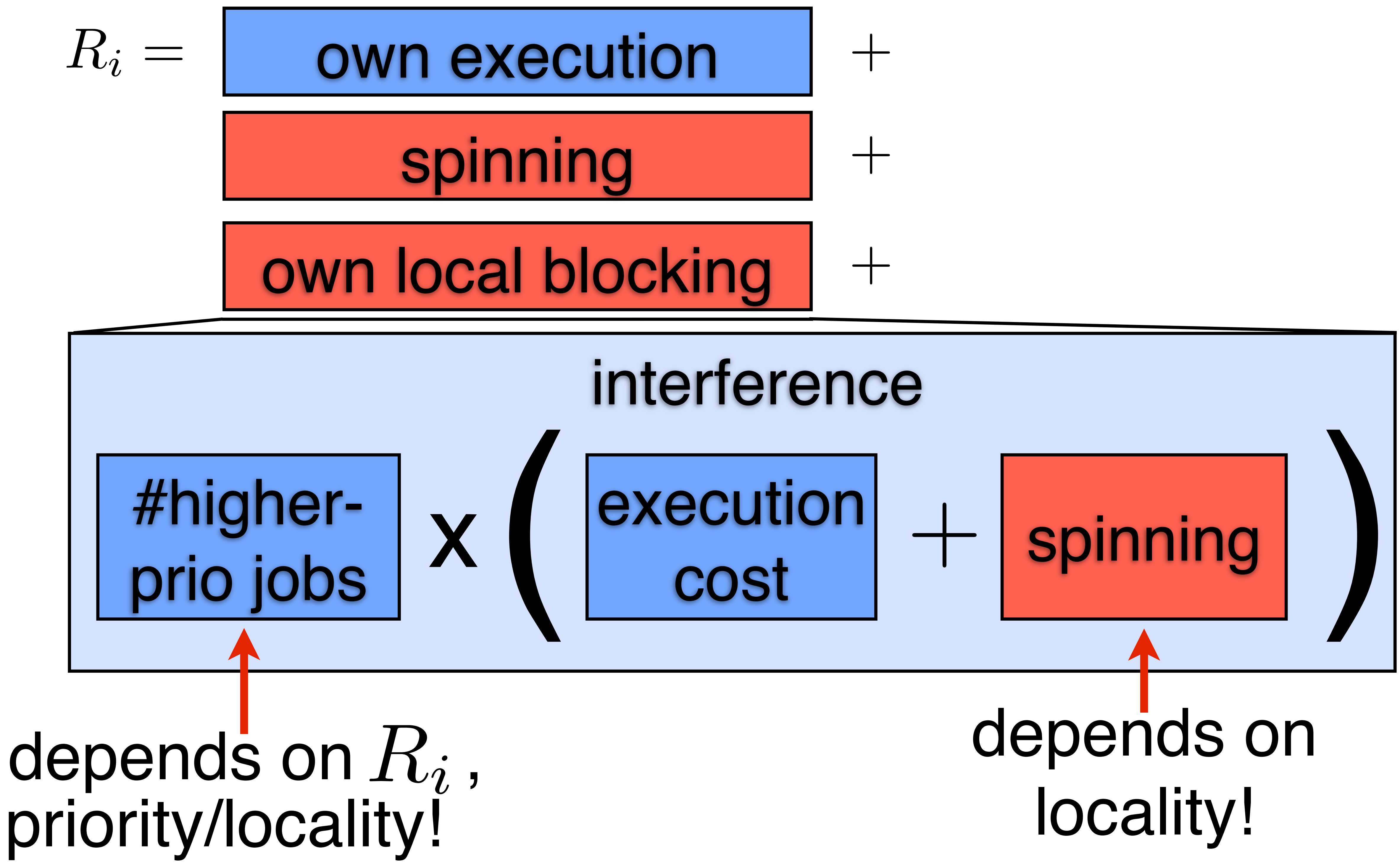
own local blocking $+$

interference

#higher-prio jobs x (execution cost $+$ spinning)

Encoding Blocking in ILP

Blocking analysis from Gai et al.:



Encoding Blocking in ILP

Blocking analysis from Gai et al.:

$R_i =$ own execution $+$
spinning $+$
own local blocking $+$

interference

#higher-prio jobs x (execution cost $+$ spinning)

variables

depends on R_i ,
priority/locality!

depends on
locality!

Encoding Blocking in ILP

How can we express blocking in purely linear terms?

interference

#higher-prio jobs x (execution cost + spinning)

variables

depends on R_i, priority/locality!

depends on locality!

Encoding Blocking in ILP

How can we express blocking in purely linear terms?

interference

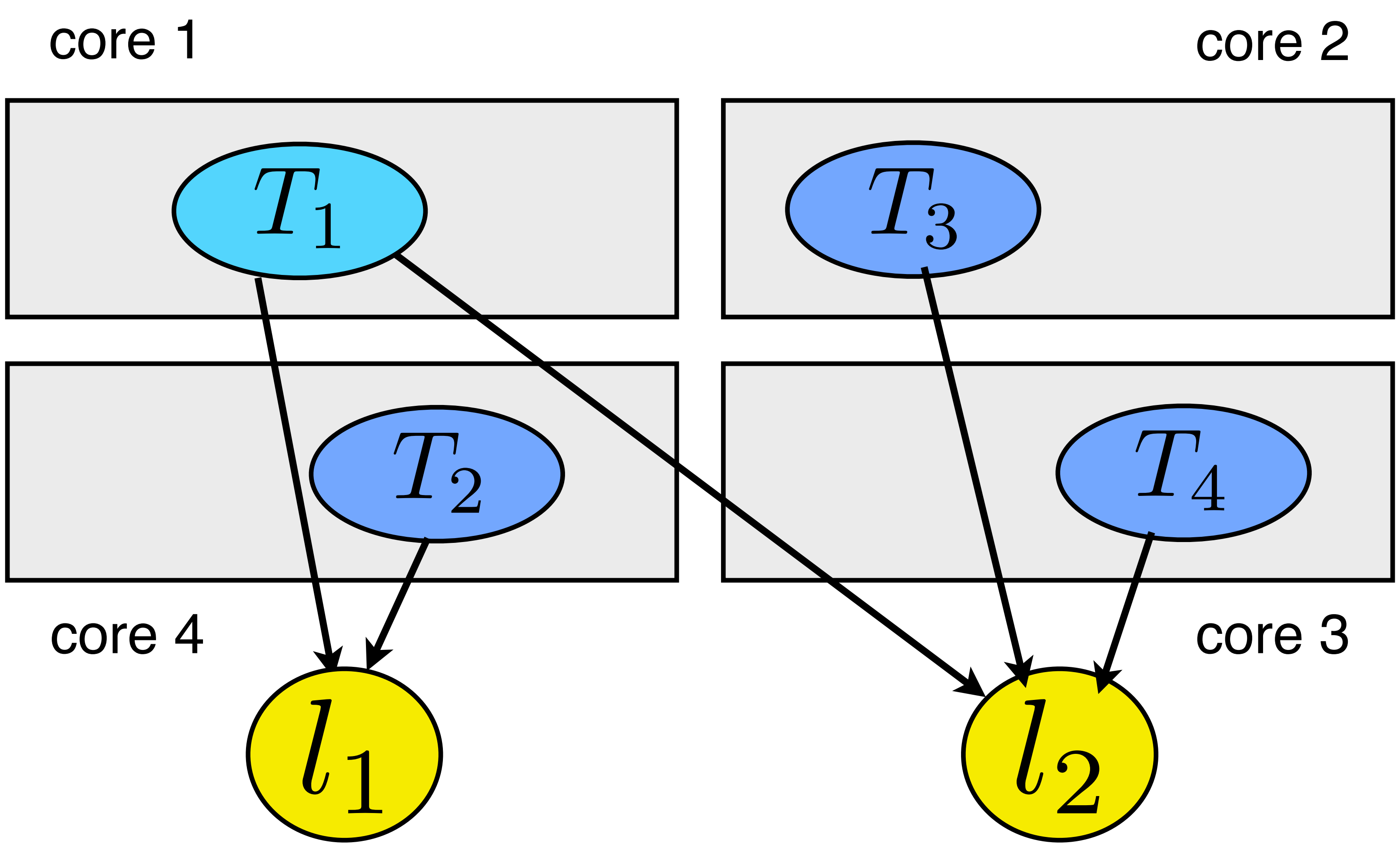
Transform to multiplication of variables with constants!

depends on n_i,
priority/locality!

locality!

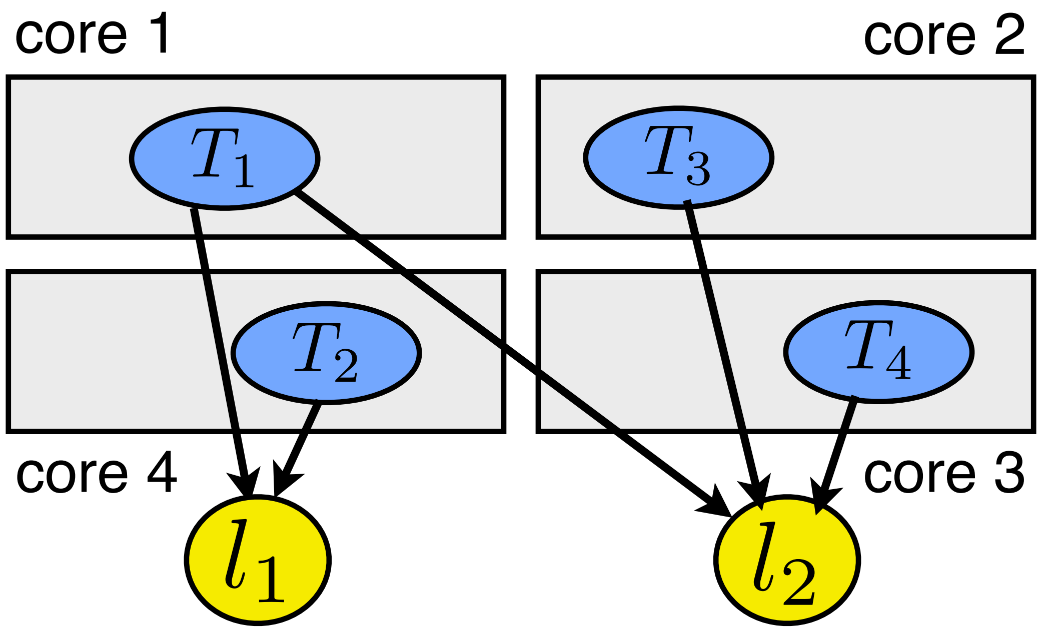
Encoding Spinning in Linear Terms

How long can T_1's job spin?



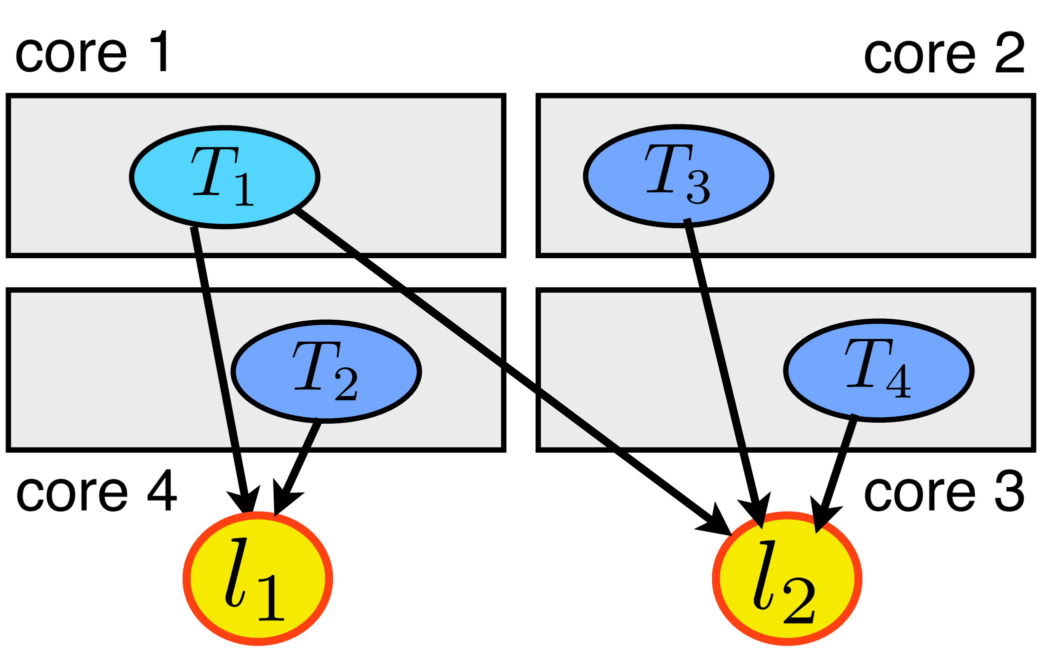
Encoding Spinning in Linear Terms

spinning =



Encoding Spinning in Linear Terms

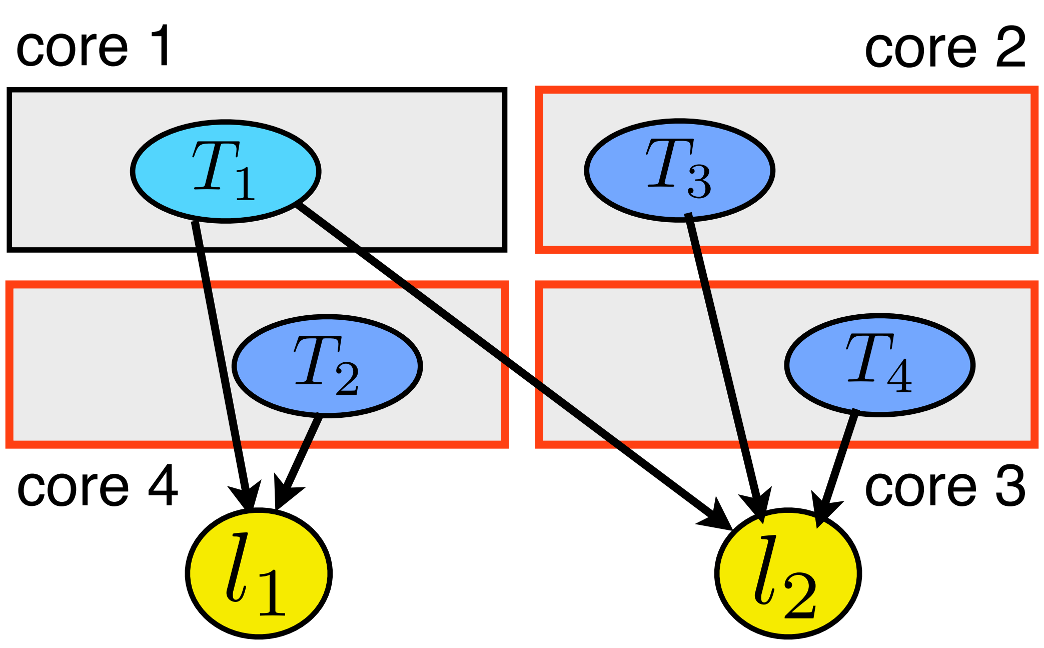
spinning = waiting for l_1 $+$ waiting for l_2



Encoding Spinning in Linear Terms

spinning = waiting for l_1 $+$ waiting for l_2

= waiting for core 4 $+$ waiting for core 2 $+$ waiting for core 3

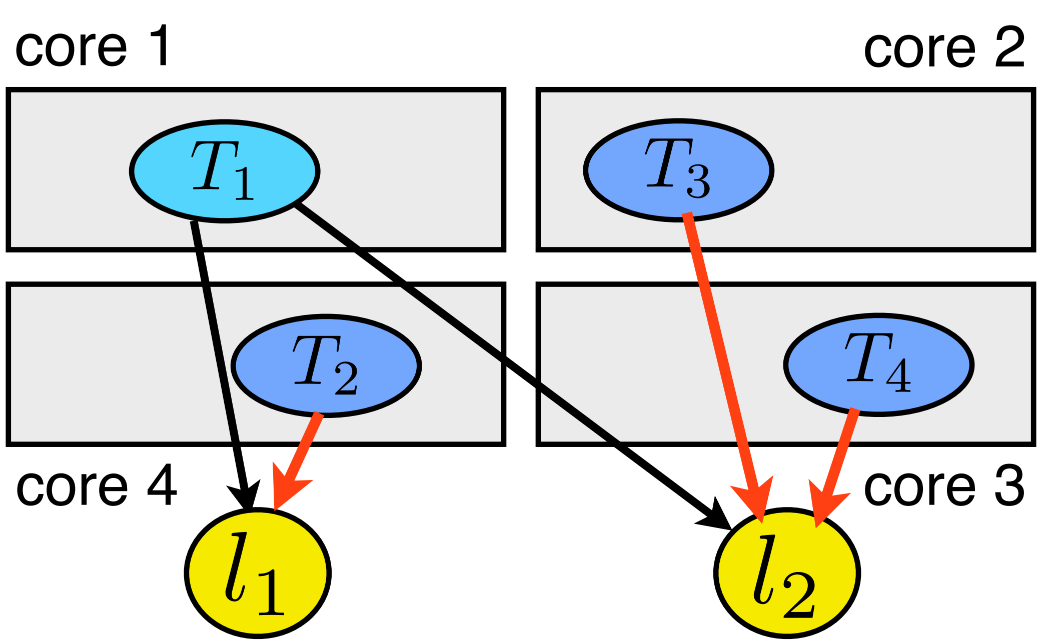


Encoding Spinning in Linear Terms

spinning = waiting for l_1 $+$ waiting for l_2

= waiting for core 4 $+$ waiting for core 2 $+$ waiting for core 3

$\geq$ T_2's request length $+$ T_3's request length $+$ T_4's request length



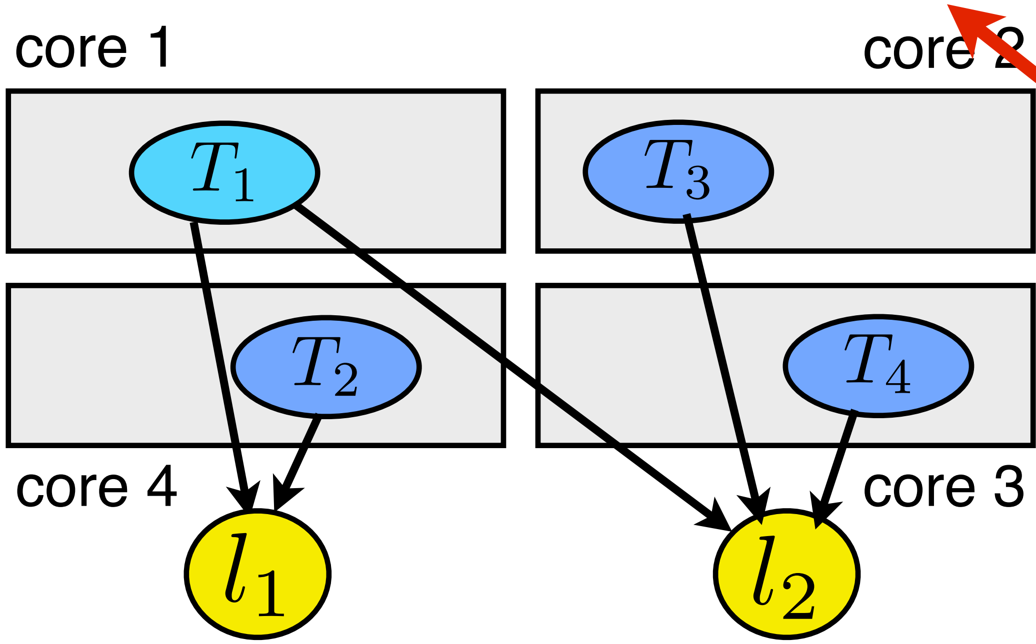
Encoding Spinning in Linear Terms

spinning = waiting for l_1 $+$ waiting for l_2

= waiting for core 4 $+$ waiting for core 2 $+$ waiting for core 3

$\geq$ T_2's request length $+$ T_3's request length $+$ T_4's request length

constants

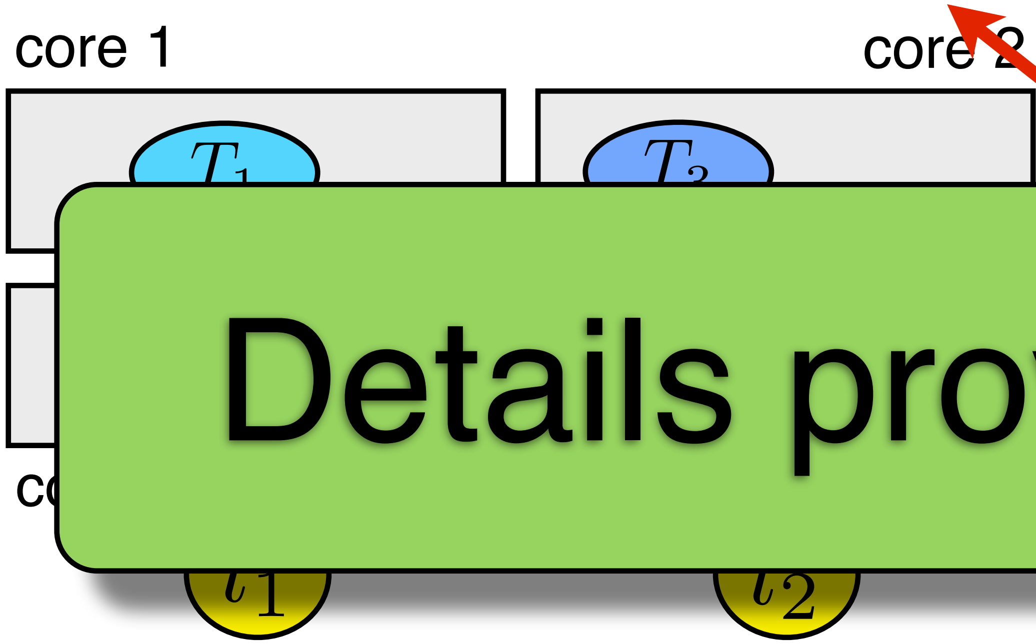


Encoding Spinning in Linear Terms

spinning $=$ waiting for l_1 $+$ waiting for l_2

$=$ waiting for core 4 $+$ waiting for core 2 $+$ waiting for core 3

$\geq$ T_2's request length $+$ T_3's request length $+$ T_4's request length



Details provided in the paper.

Making ILP-based Partitioning Practical

In the real world, we also want to...

minimize number of processors ⇒ objective function unused in basic ILP

Making ILP-based Partitioning Practical

In the real world, we also want to...

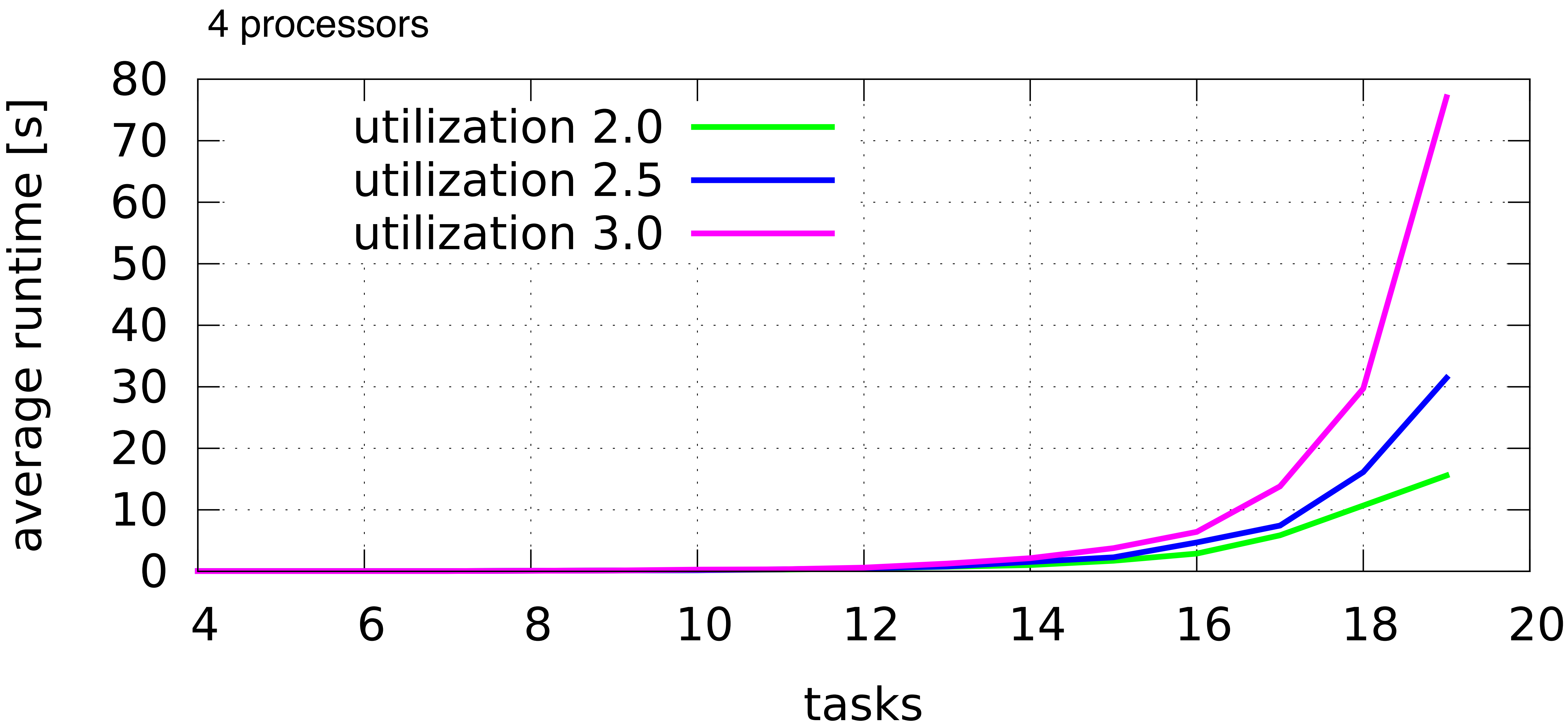
minimize number of processors	⇒	objective function unused in basic ILP
handle precedence constraints	⇒	one additional constraint per task

Making ILP-based Partitioning Practical

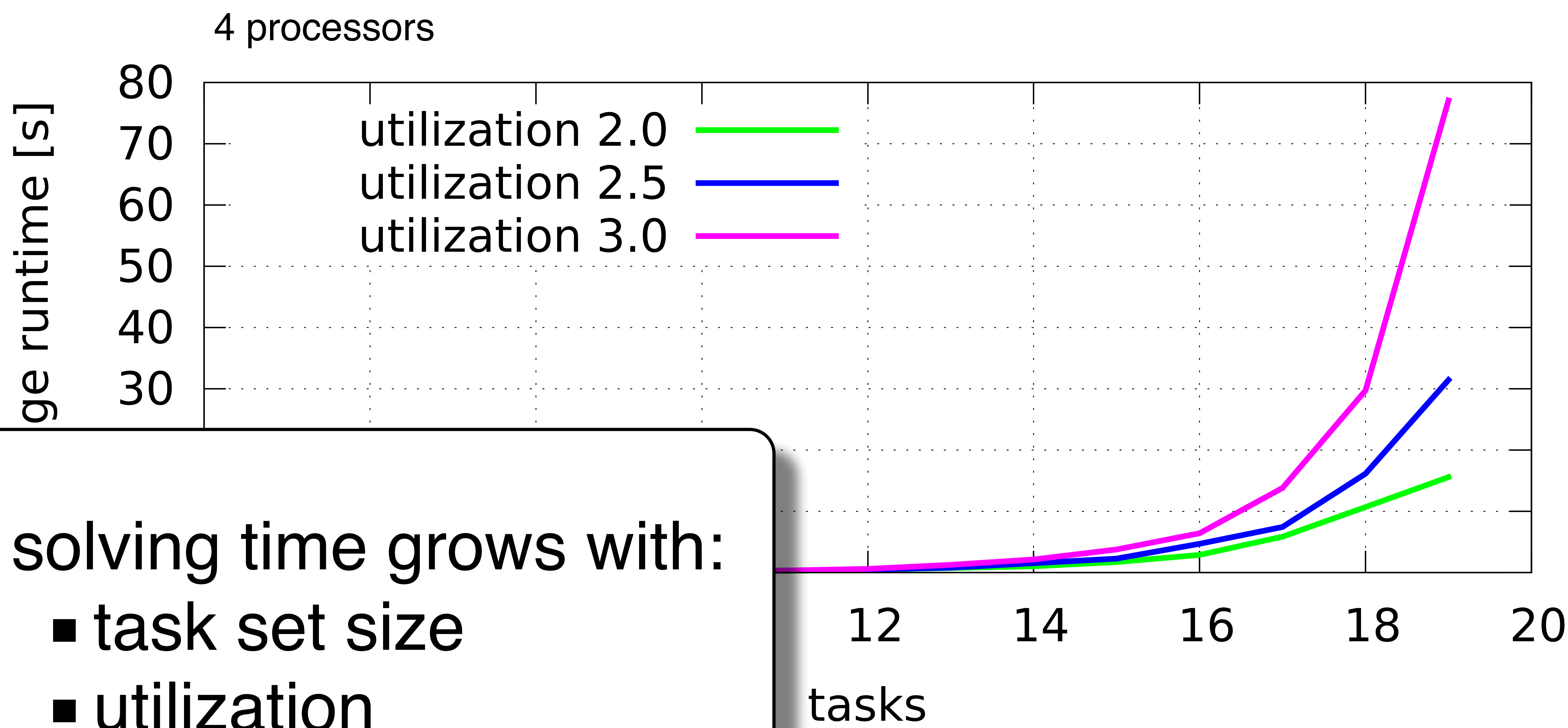
In the real world, we also want to...

minimize number of processors	⇒	objective function unused in basic ILP
handle precedence constraints	⇒	one additional constraint per task
incorporate partial specifications	⇒	constrain existing helper variables

ILP Solving Overhead



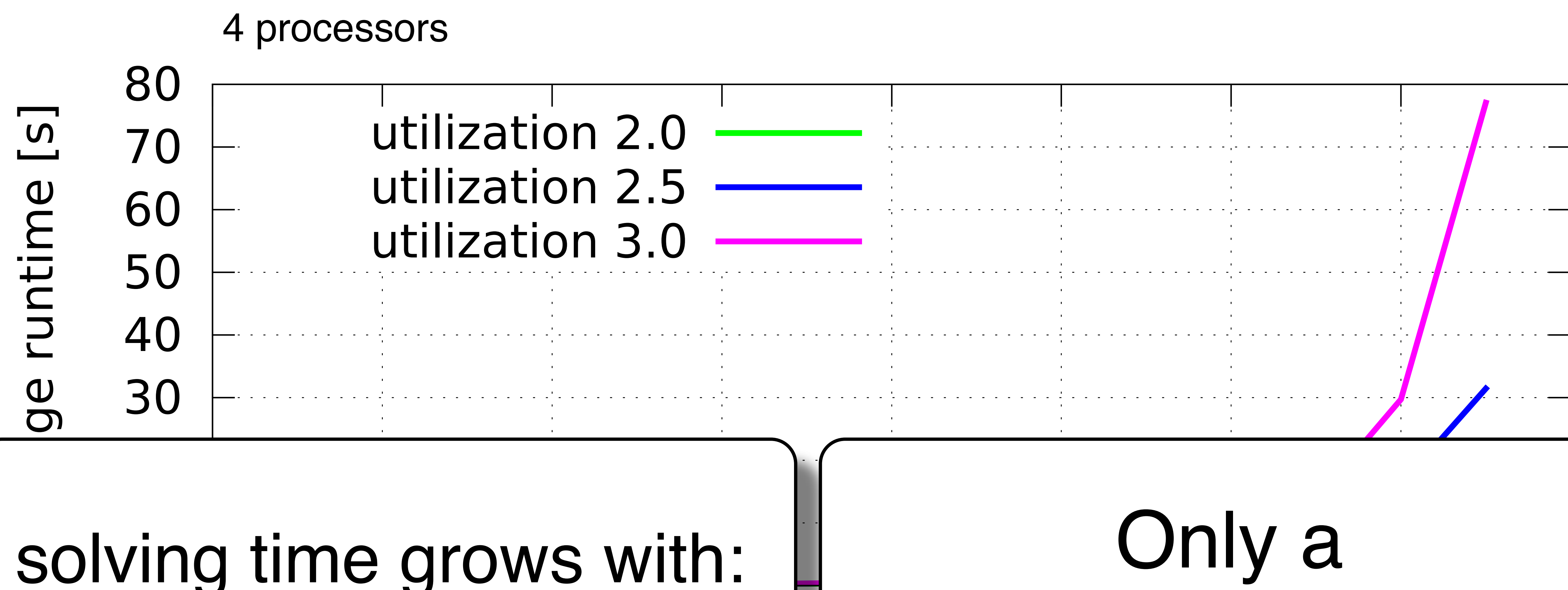
ILP Solving Overhead



solving time grows with:

- task set size
- utilization
- resource contention

ILP Solving Overhead

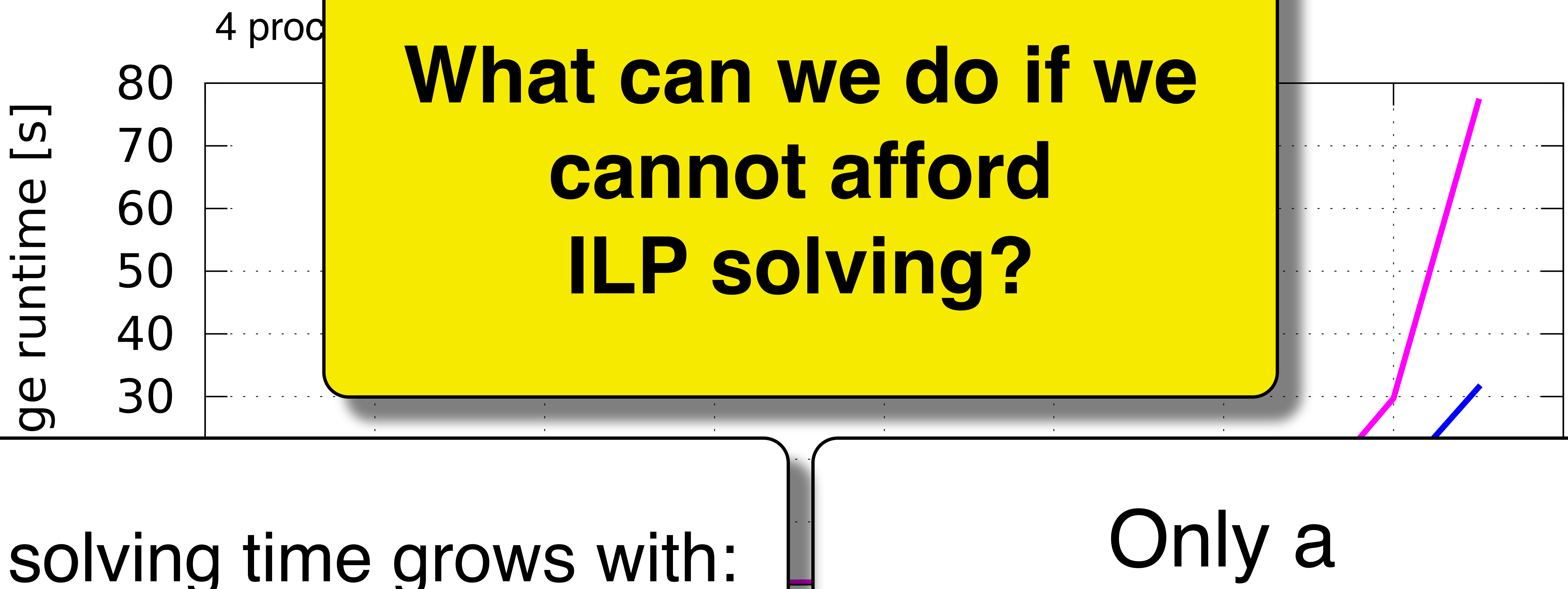


solving time grows with:

- task set size
- utilization
- resource contention

Only a **one-time cost** for exploiting wasted potential!

ILP Solving



What can we do if we cannot afford ILP solving?

solving time grows with:

- task set size
- utilization
- resource contention

Only a one-time cost for exploiting wasted potential!

Part III

A Simple Sharing-Aware Partitioning Heuristic

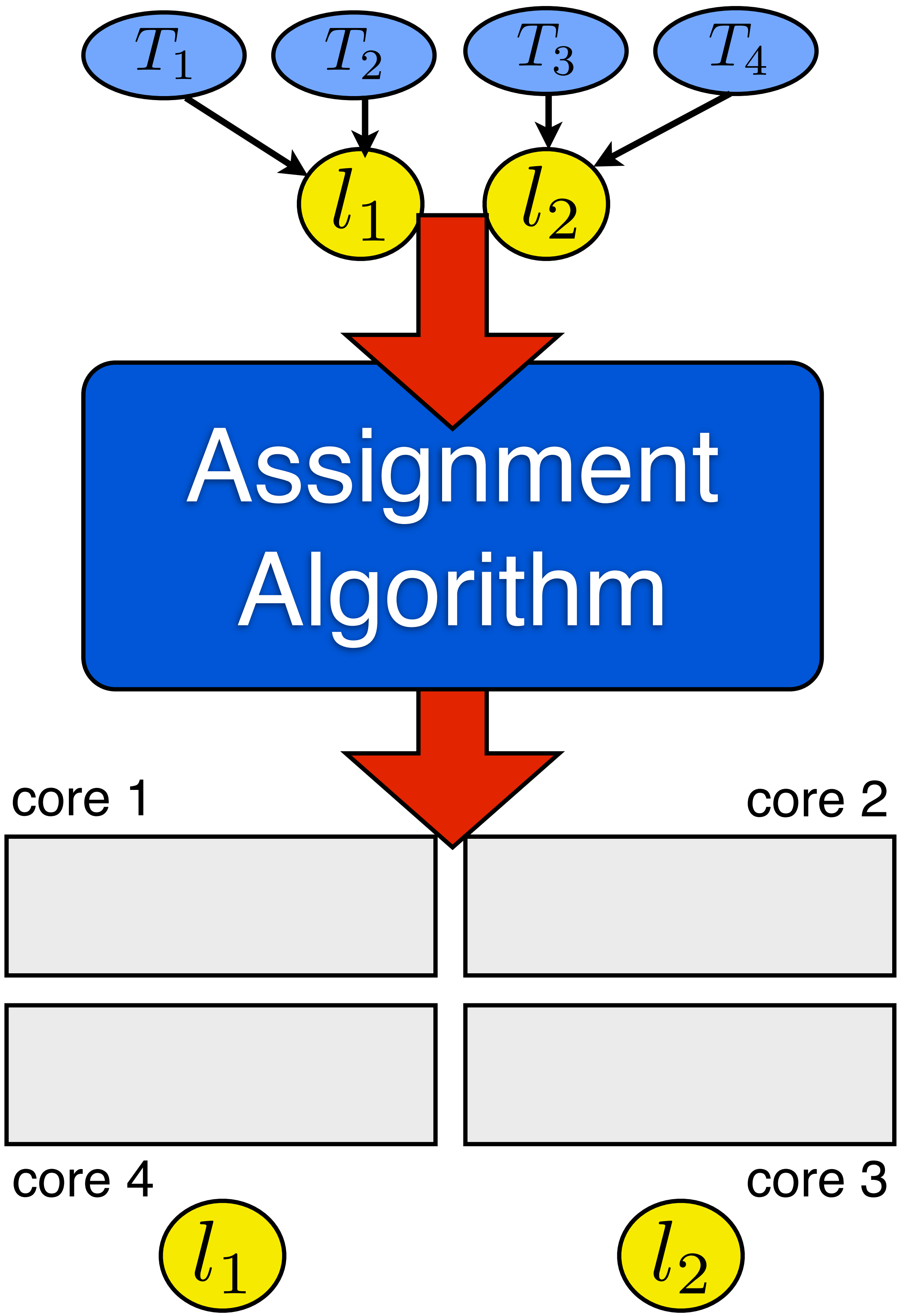
Sharing-Aware Partitioning Heuristics

Prior Sharing-Aware Heuristics:

- LNR-heuristic [1]
- Blocking-Aware Partitioning Algorithm (BPA) [2]

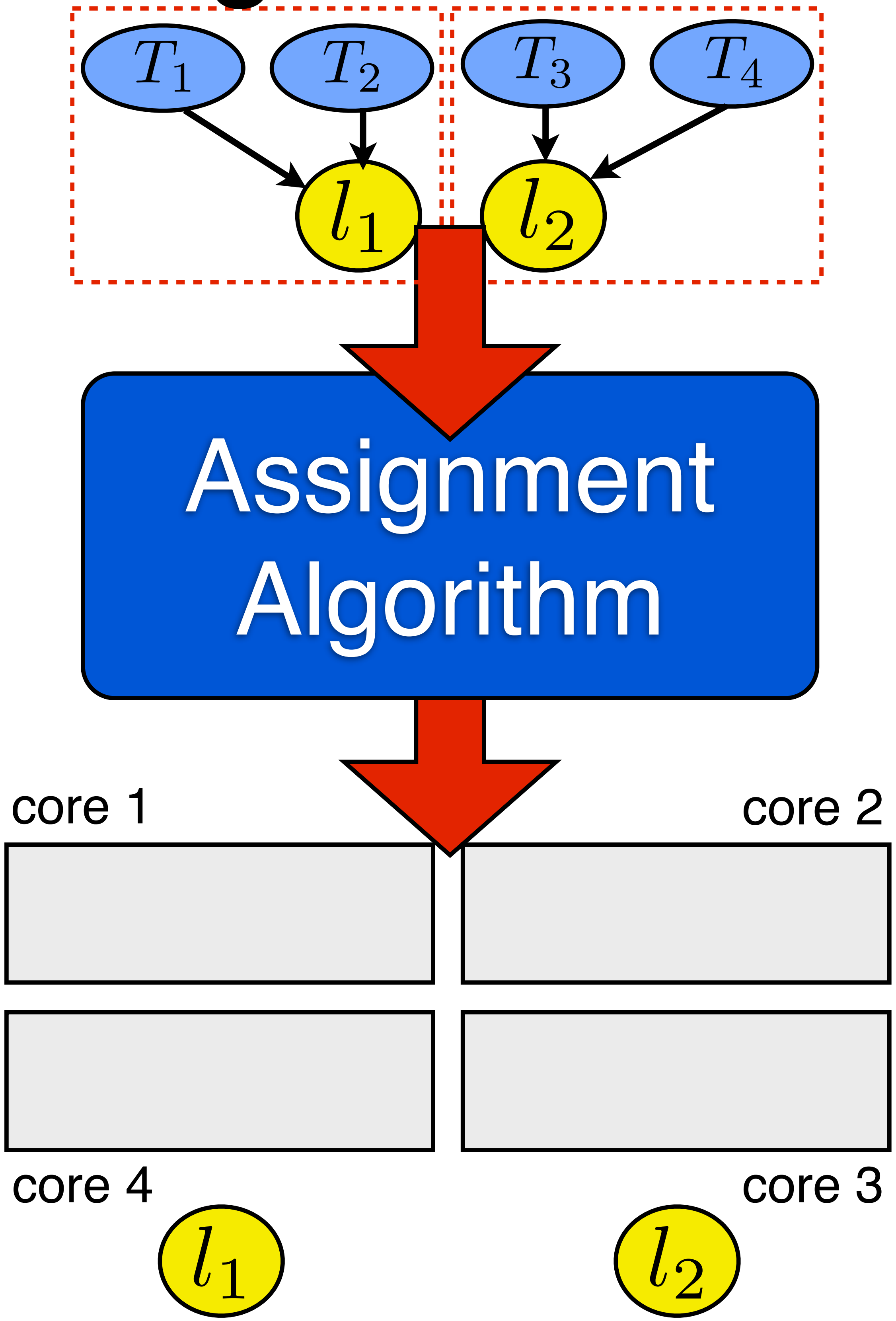
[1] K. Lakshmanan, D. de Niz, and R. Rajkumar, "Coordinated task scheduling, allocation and synchronization on multiprocessors," in Proc. RTSS, 2009.
[2] F. Nemati, T. Nolte, and M. Behnam, "Partitioning real-time systems on multiprocessors with shared resources," in Proc. OPODIS, 2010.

High-Level View of Sharing-Aware Partitioning Heuristics



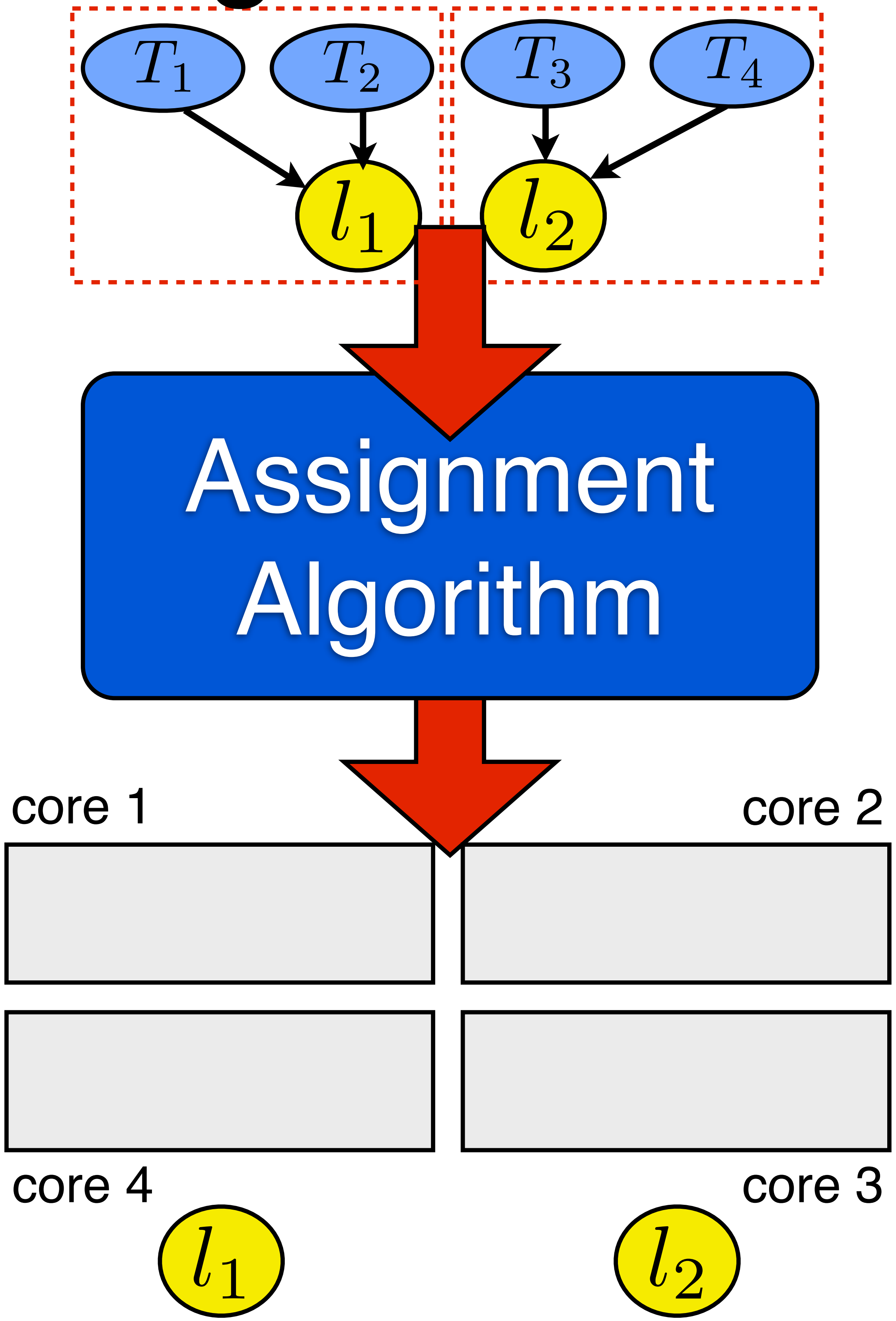
- identify connected components
- assign components
- if not possible, split
  - cost functions

High-Level View of Sharing-Aware Partitioning Heuristics



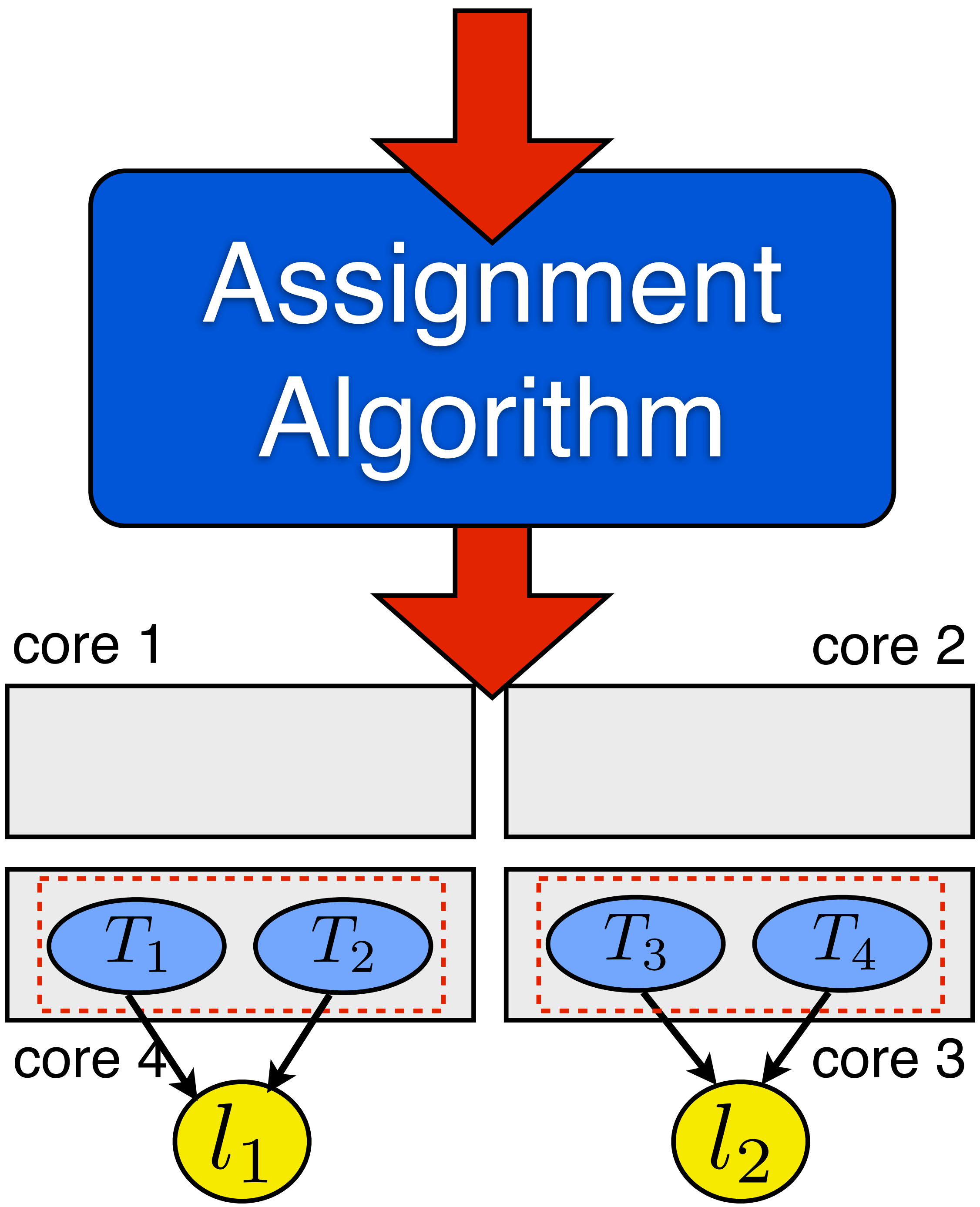
- identify connected components
- assign components
- if not possible, split
  - cost functions

High-Level View of Sharing-Aware Partitioning Heuristics



- identify connected components
- assign components
- if not possible, split
  - cost functions

High-Level View of Sharing-Aware Partitioning Heuristics

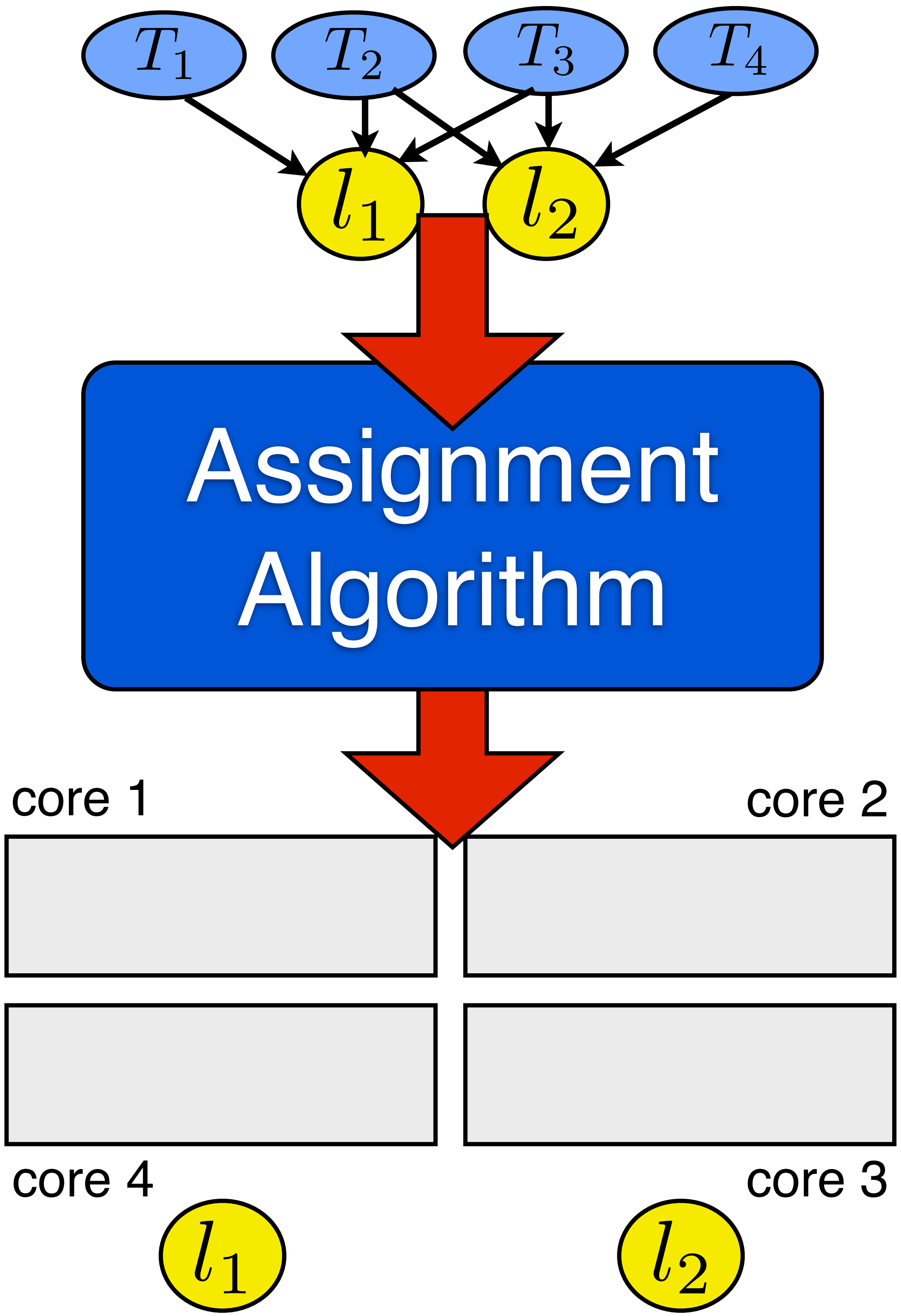


- identify connected components
- assign components
- if not possible, split
  - cost functions

High-Level View of Sharing-Aware Partitioning Heuristics

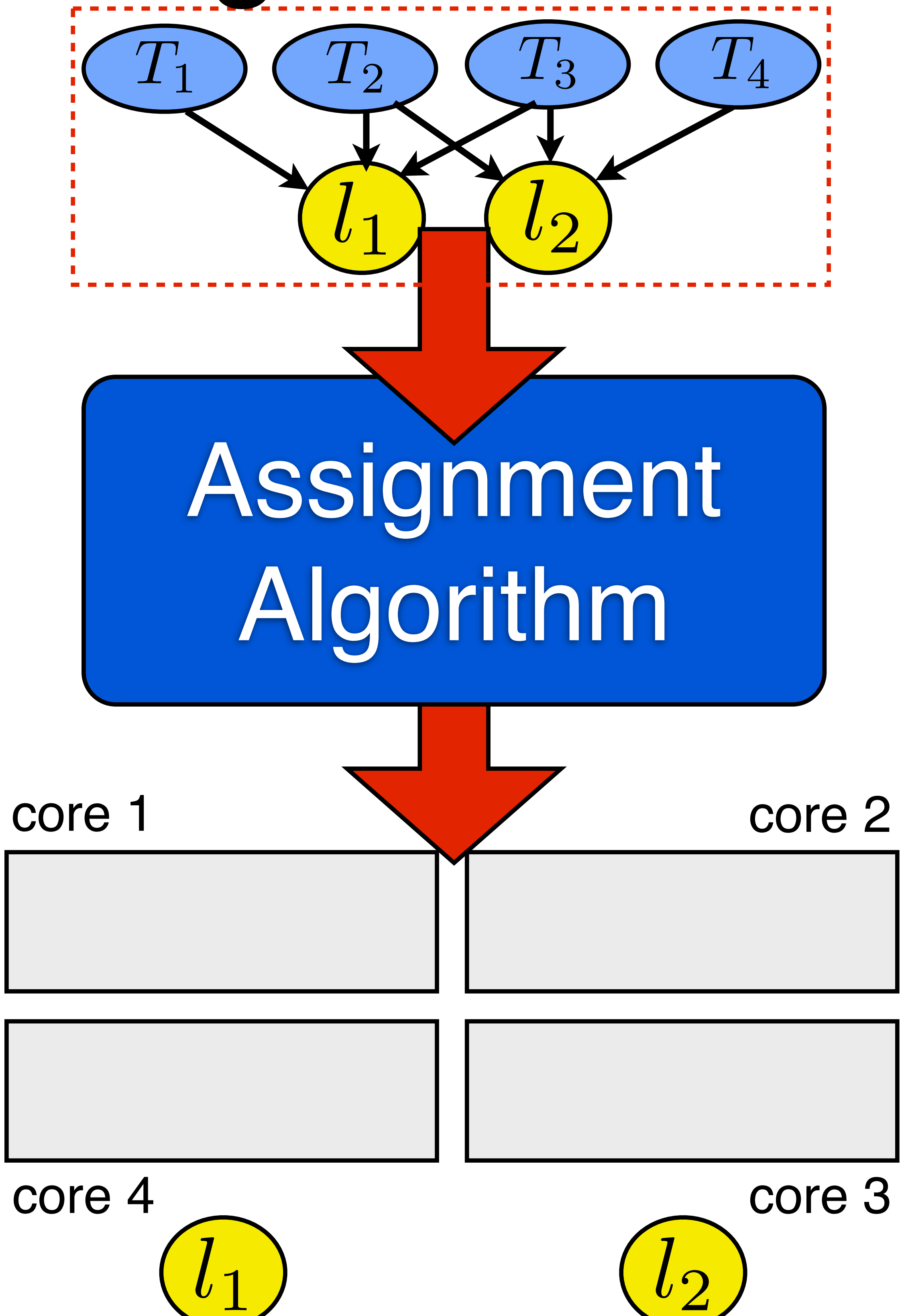
- identify connected components
- assign components
- if not possible, split
  - cost functions

High-Level View of Sharing-Aware Partitioning Heuristics



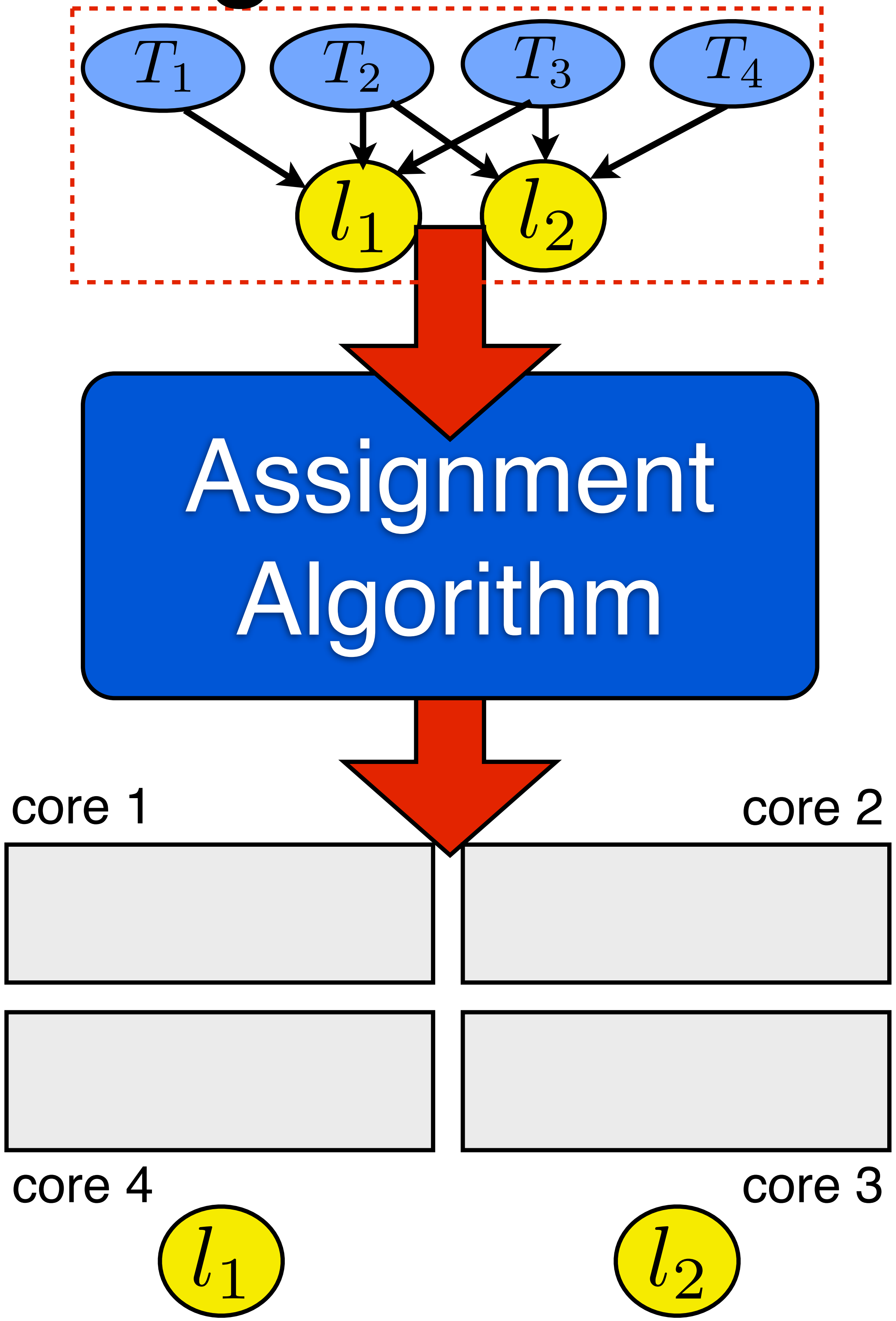
- identify connected components
- assign components
- if not possible, split
  - cost functions

High-Level View of Sharing-Aware Partitioning Heuristics



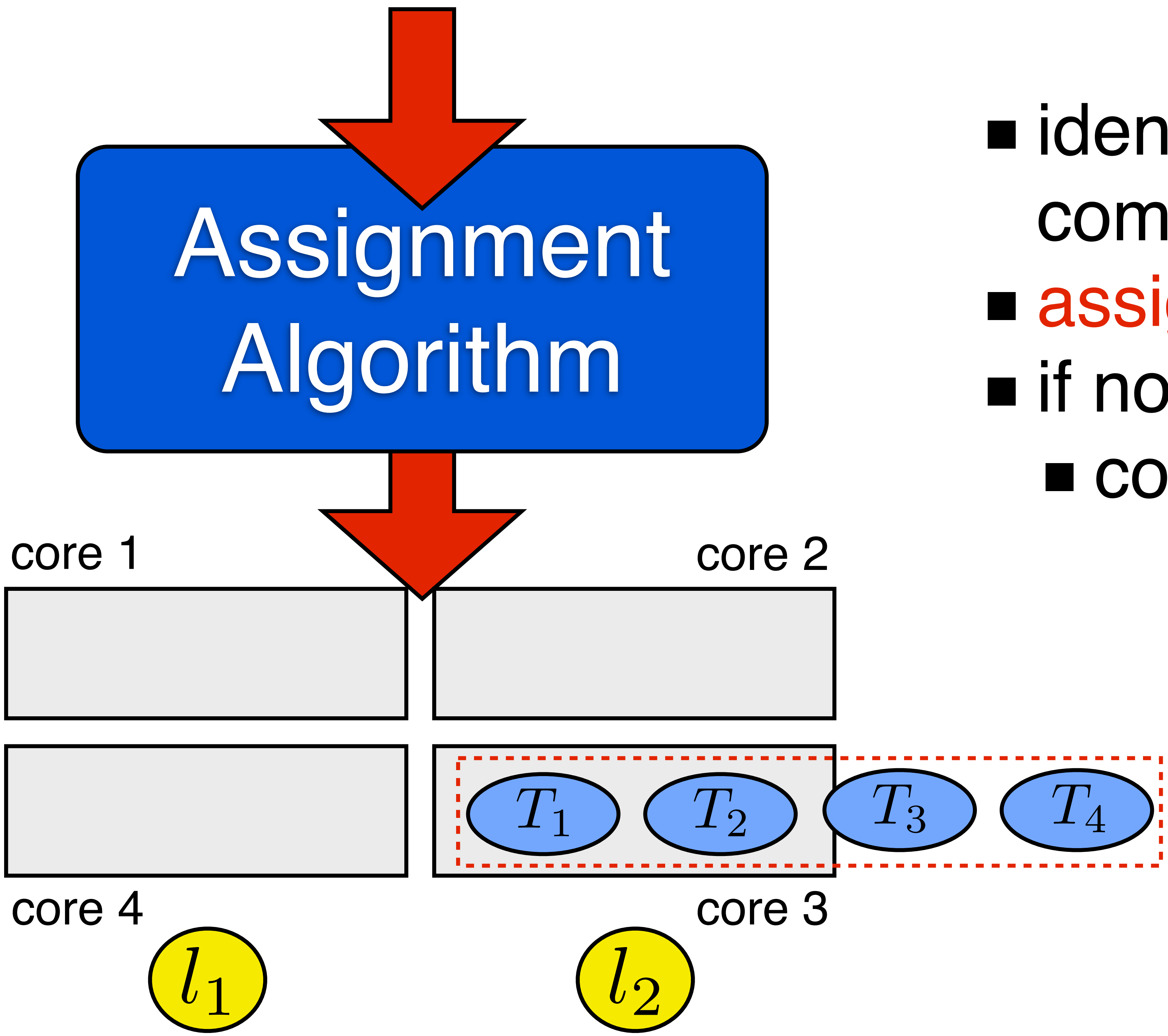
- identify connected components
- assign components
- if not possible, split
  - cost functions

High-Level View of Sharing-Aware Partitioning Heuristics



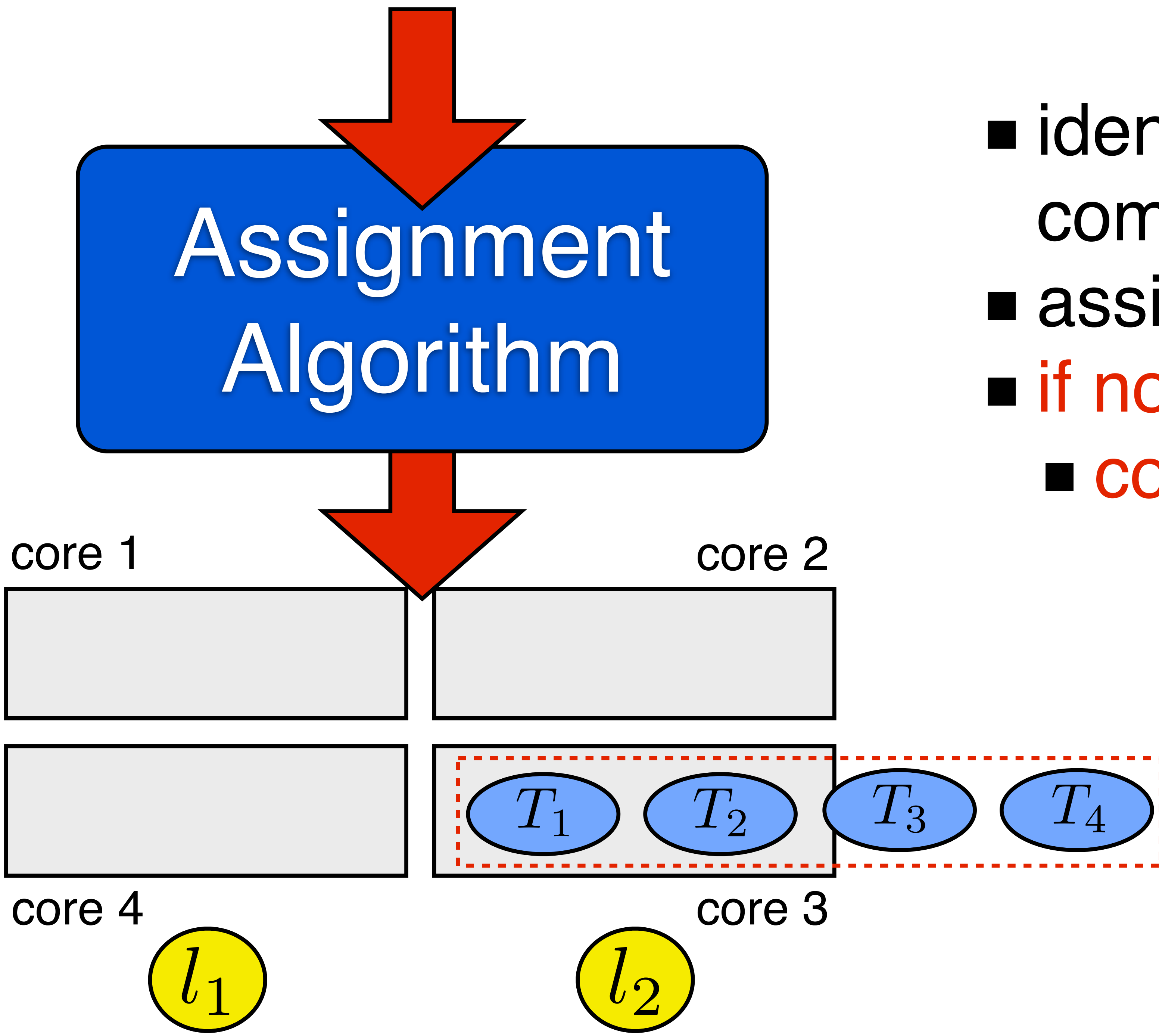
- identify connected components
- assign components
- if not possible, split
  - cost functions

High-Level View of Sharing-Aware Partitioning Heuristics



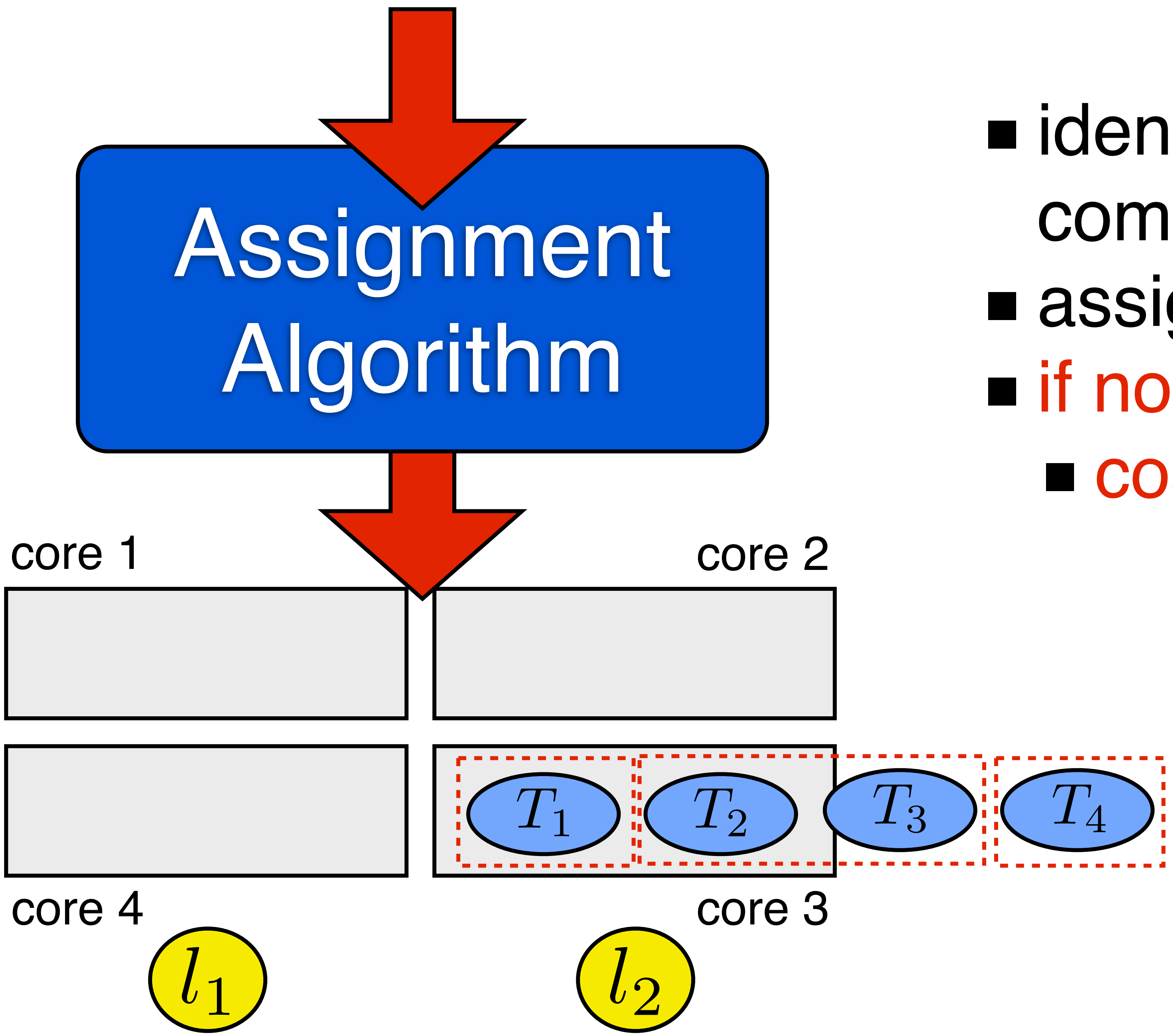
- identify connected components
- assign components
- if not possible, split
  - cost functions

High-Level View of Sharing-Aware Partitioning Heuristics



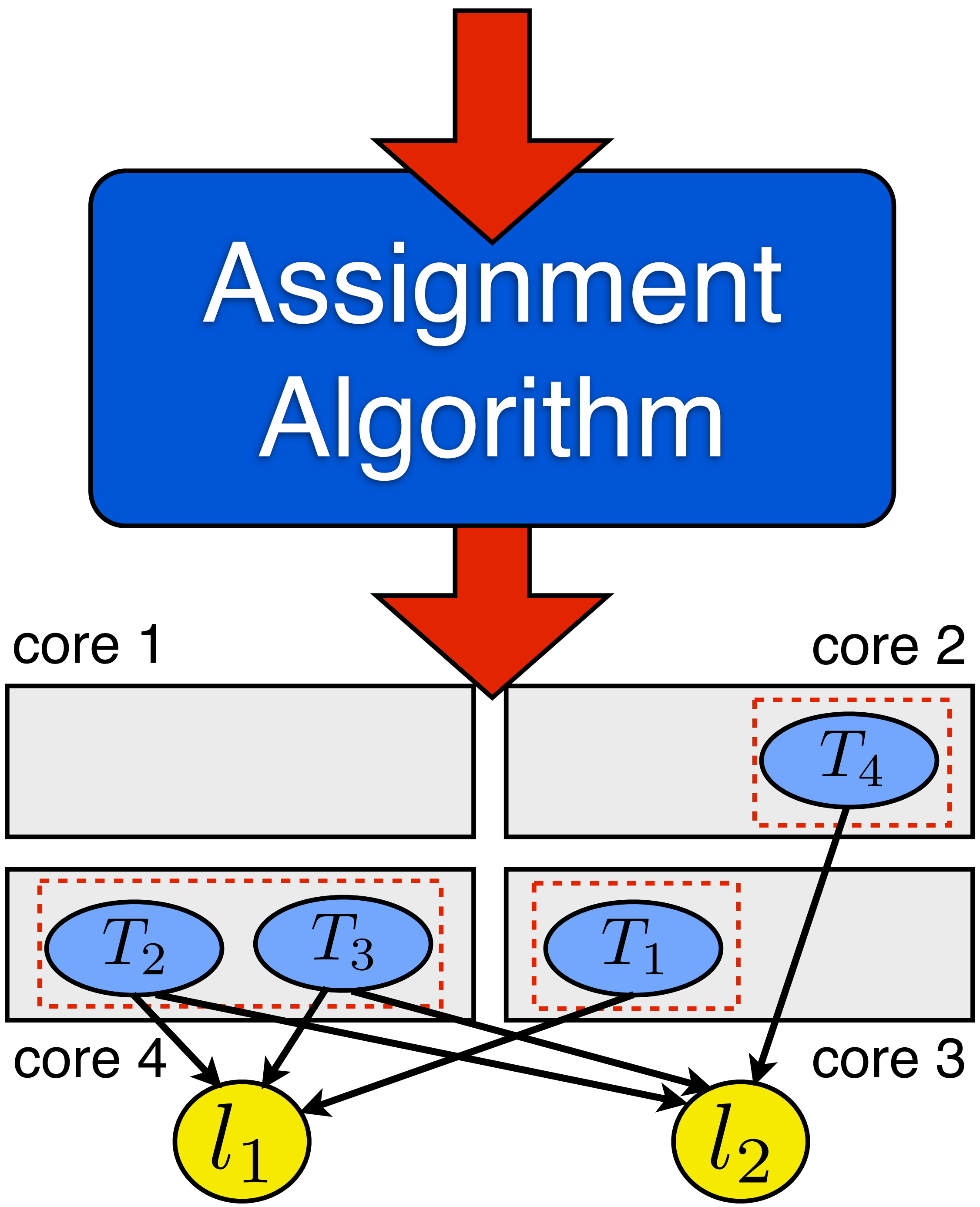
- identify connected components
- assign components
- if not possible, split
  - cost functions

High-Level View of Sharing-Aware Partitioning Heuristics



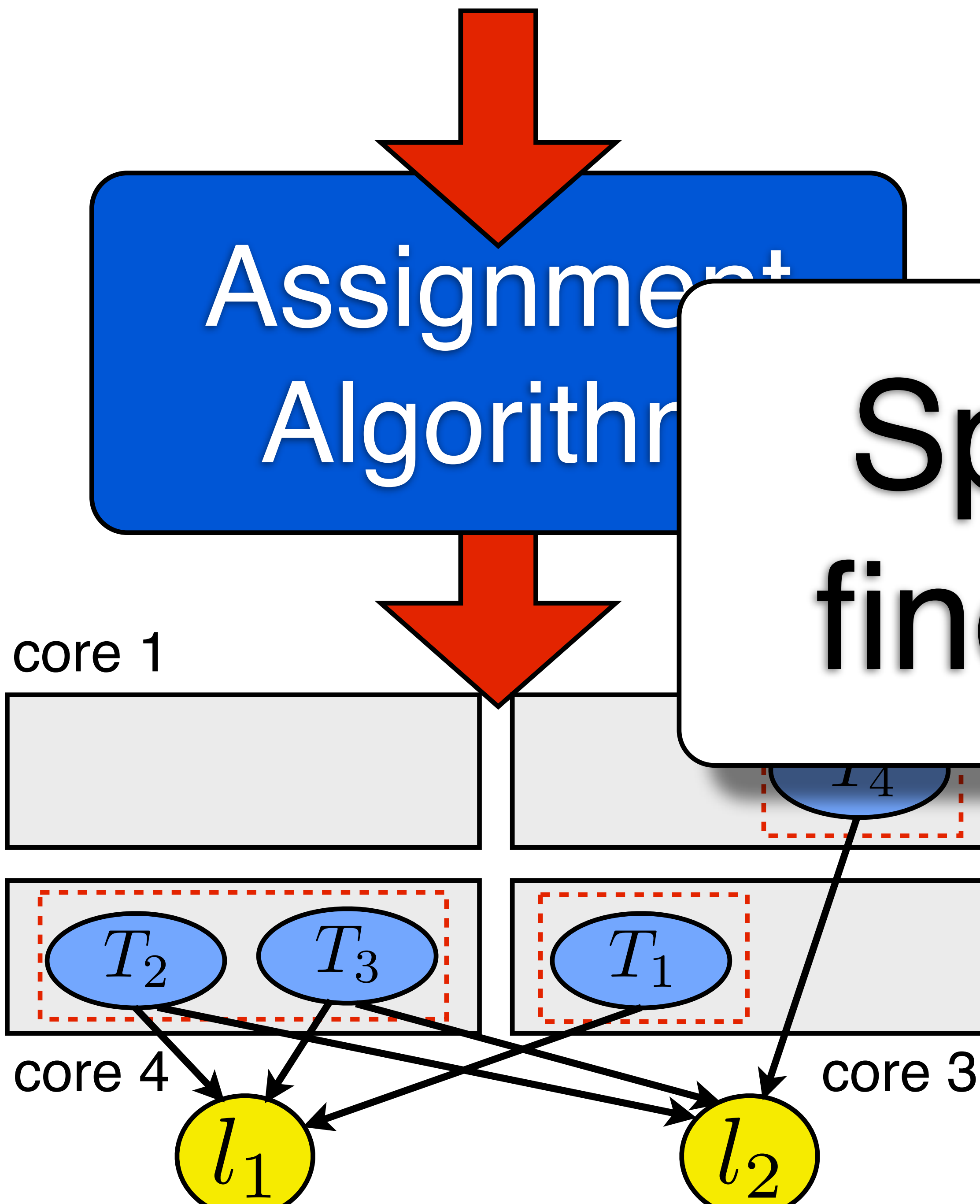
- identify connected components
- assign components
- if not possible, split
  - cost functions

High-Level View of Sharing-Aware Partitioning Heuristics



- identify connected components
- assign components
- if not possible, split
  - cost functions

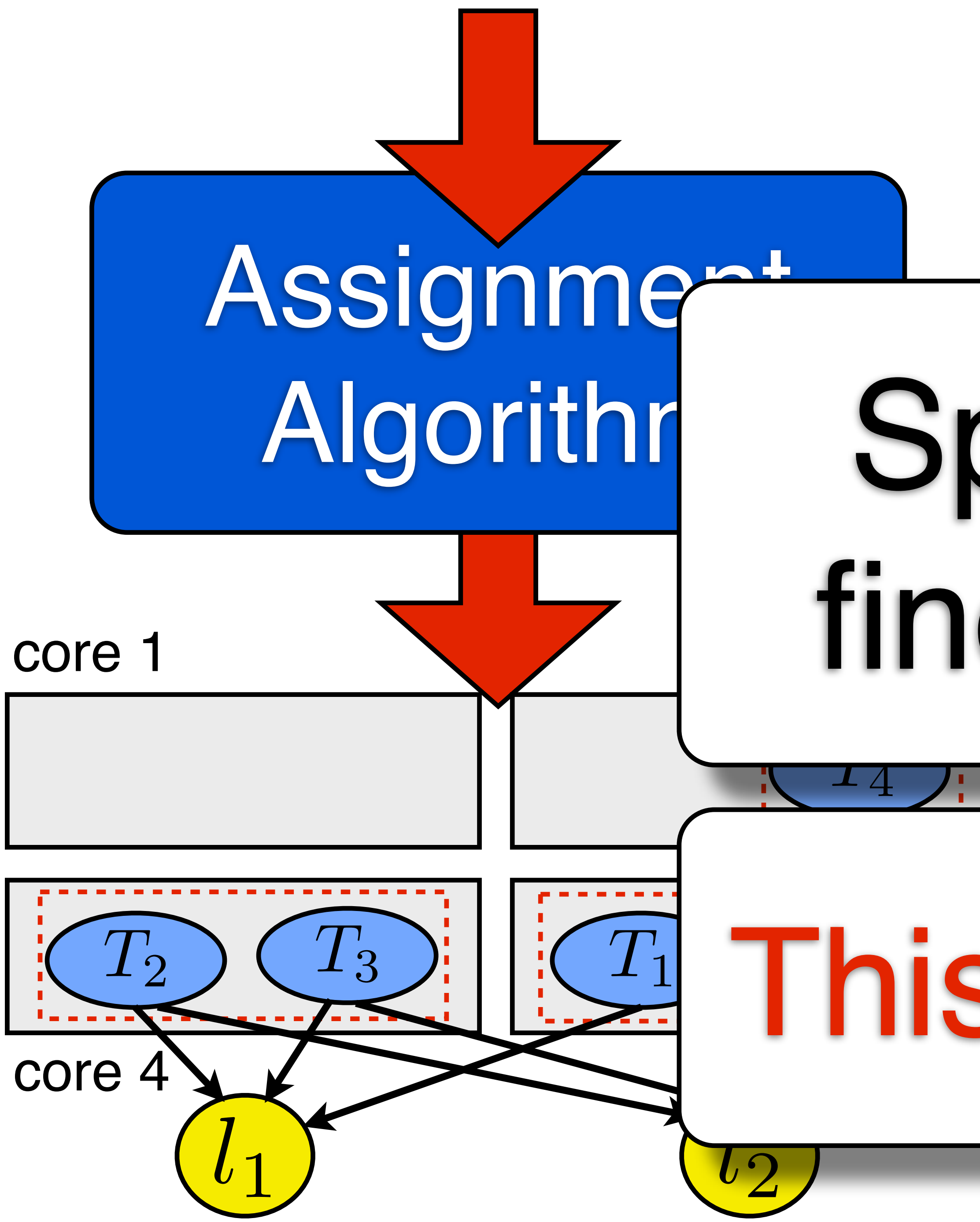
High-Level View of Sharing-Aware Partitioning Heuristics



- identify connected components

Splitting requires finding a *good* cut

High-Level View of Sharing-Aware Partitioning Heuristics



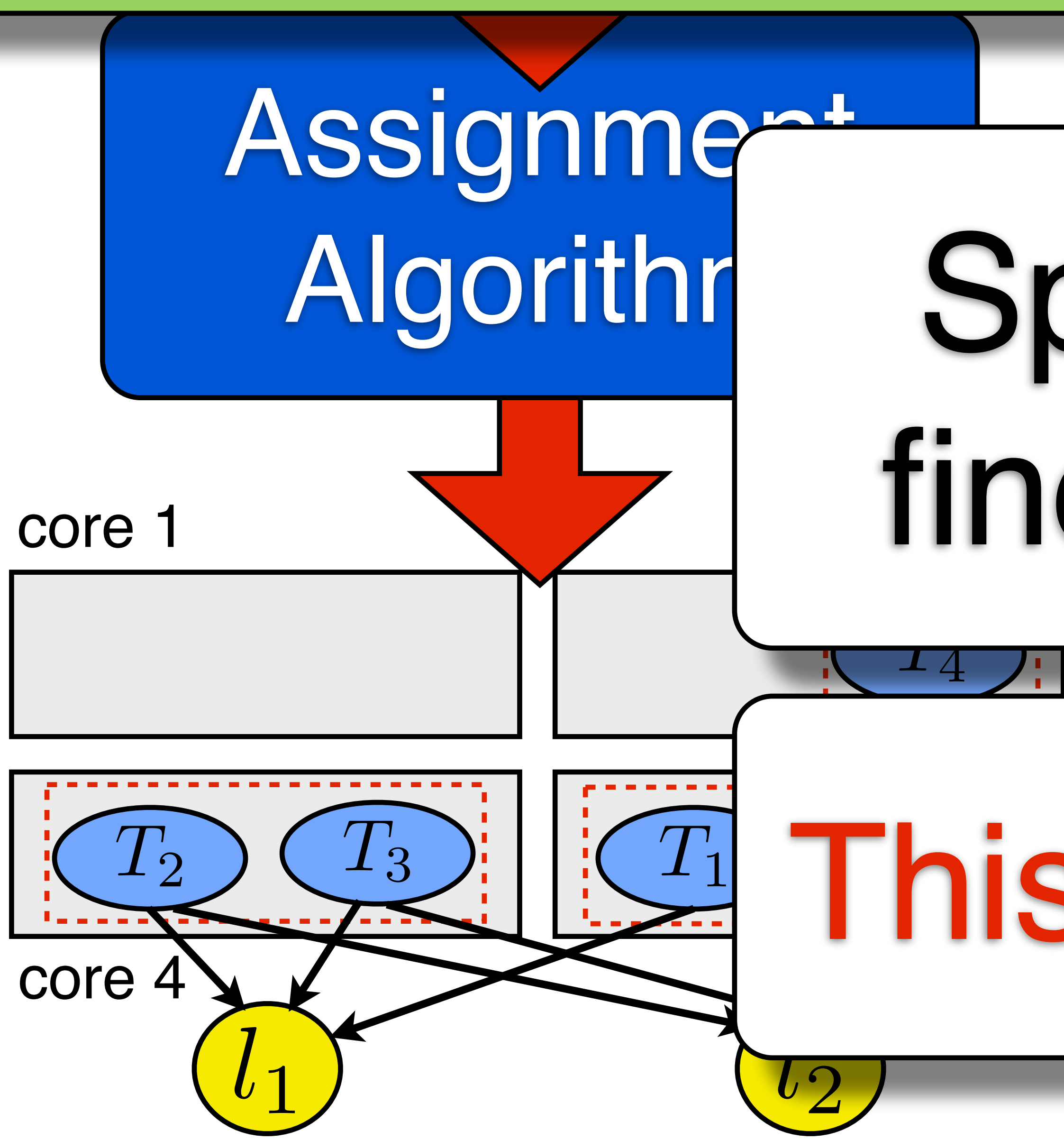
- identify connected components

Splitting requires finding a *good* cut

This is not so easy...

High-Level View of Sharing-Aware Partitioning Heuristics

Can we do something simpler?



components

Splitting requires finding a *good* cut

This is not so easy...

Greedy Slacker

embarrassingly simple:

- disregard graph structure
- greedily try to maximize minimum slack

Greedy Slacker

<u>embarrassingly simple:</u>

- disregard graph structure
- greedily try to maximize minimum slack

time until deadline is missed

Greedy Slacker

<u>embarrassingly simple:</u>

- disregard graph structure
- greedily try to maximize minimum slack

for each task T_i in order of increasing period:
  for each processor c_k:
    compute slack when T_i assigned to c_k
  if there is no c_r such that minimum slack $\geq$ 0:
    fail
  else:
    assign T_i to c_r s.t. minimum slack is maximized

Greedy Slacker

Works with **any** blocking analysis.

No cost functions!
Ignores graph structure!

- disregard graph structure
- greedily try to maximize minimum slack

for each task T_i in order of increasing period:
  for each processor c_k:
    compute slack when T_i assigned to c_k
  if there is no c_r such that minimum slack $\geq$ 0:
    fail
  else:
    assign T_i to c_r s.t. minimum slack is maximized

Greedy Slacker

Works with **any** blocking analysis.

No cost functions!
Ignores graph structure!

- disregard graph structure
- greedily try to maximize minimum slack

Can this possibly work?

else:

assign T_i **to** C_r **s.t. minimum slack is maximized**

Experimental Setup

Heuristics:

- sharing-oblivious (bin-packing)
- LNR-heuristic
- BPA
- Greedy Slacker

Experimental Setup

Heuristics:

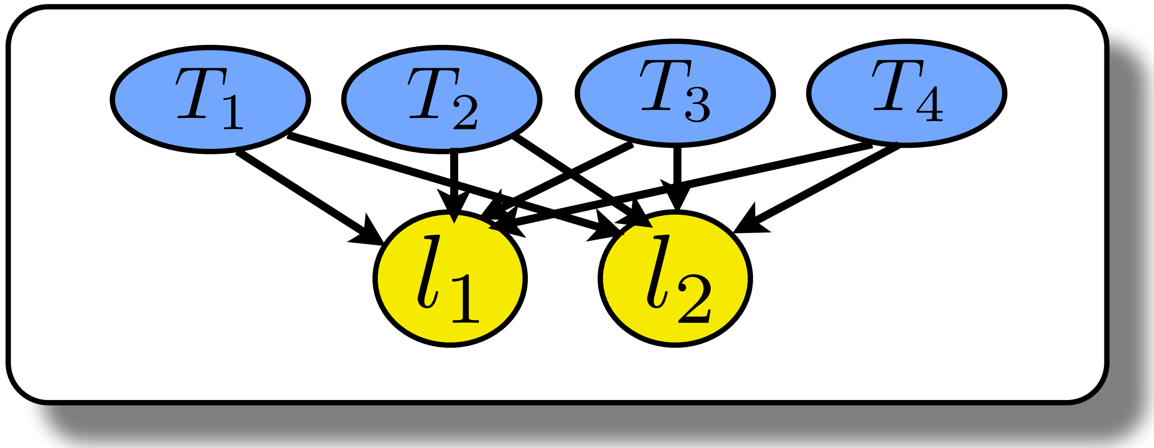
- sharing-oblivious (bin-packing)
- LNR-heuristic
- BPA
- Greedy Slacker

Configuration:

- 8 processors
- 4 shared resources
- 10% average task utilization
- each resource accessed by 25% of tasks
- 100 samples

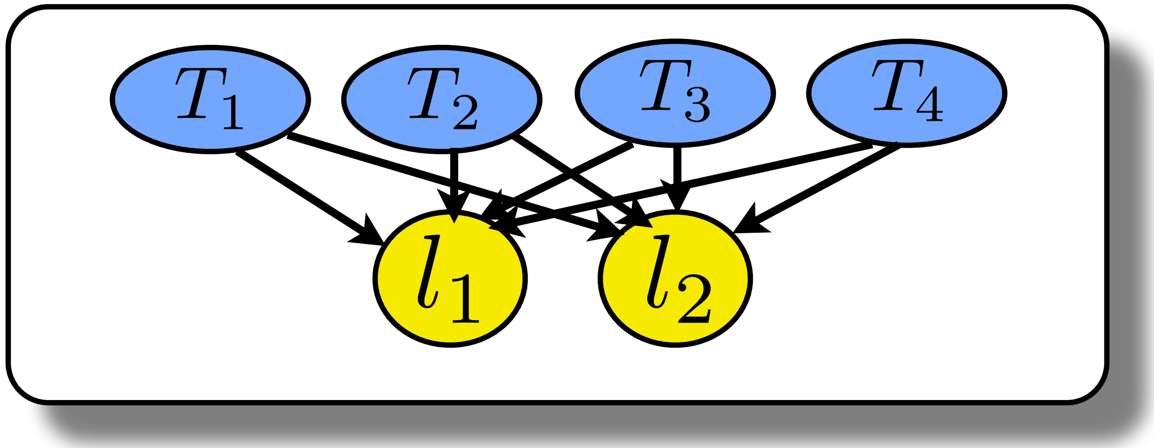
Resource Access Patterns

Unstructured

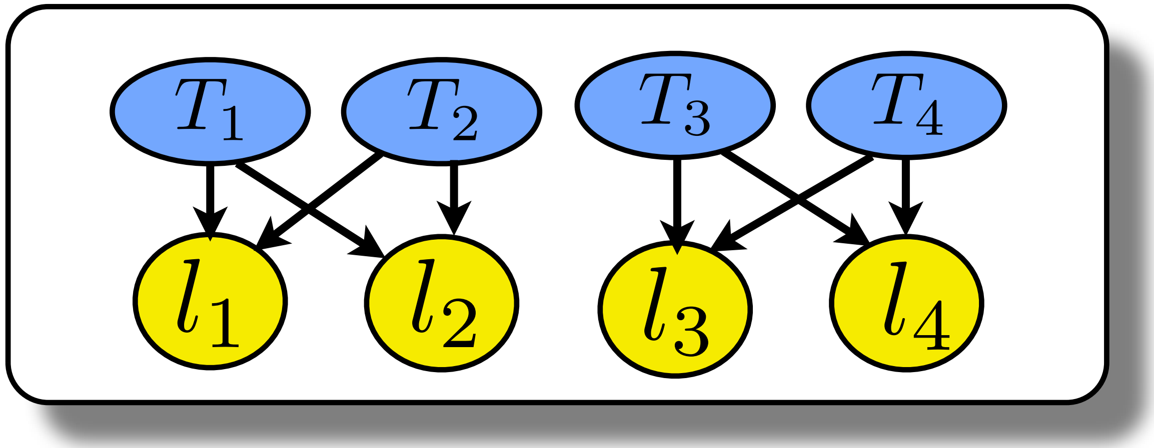


Resource Access Patterns

Unstructured



Structured

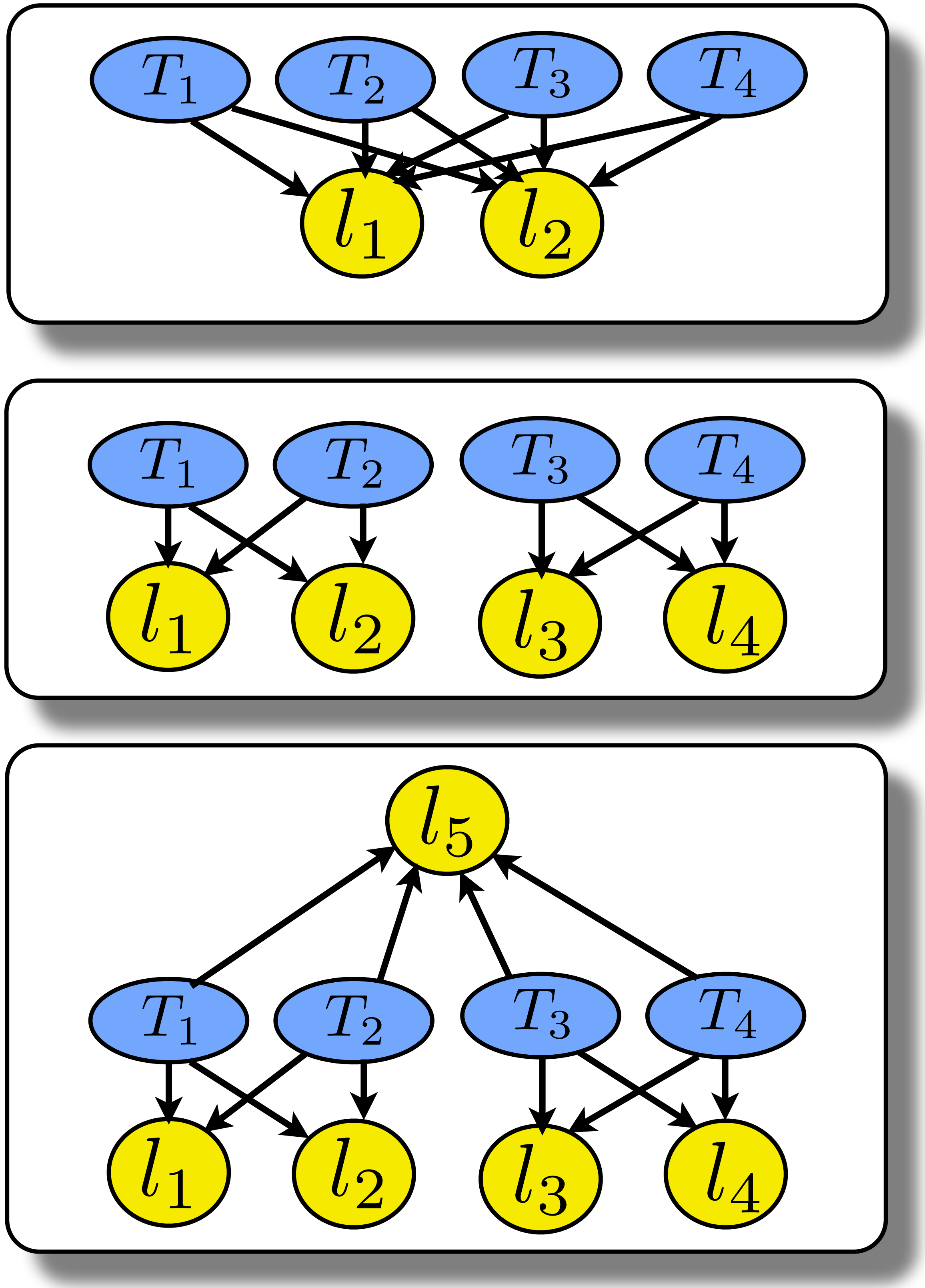


Resource Access Patterns

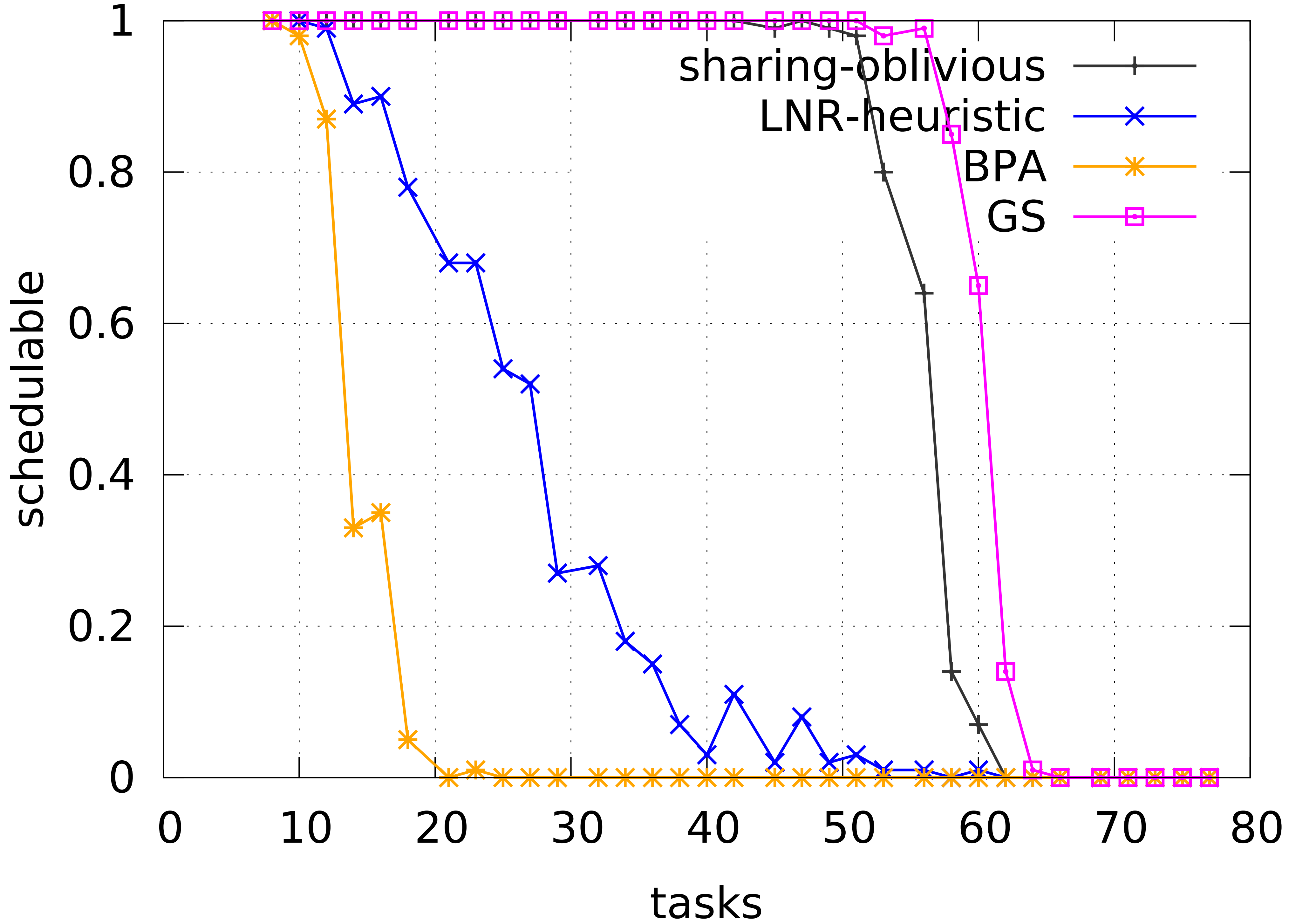
Unstructured

Structured

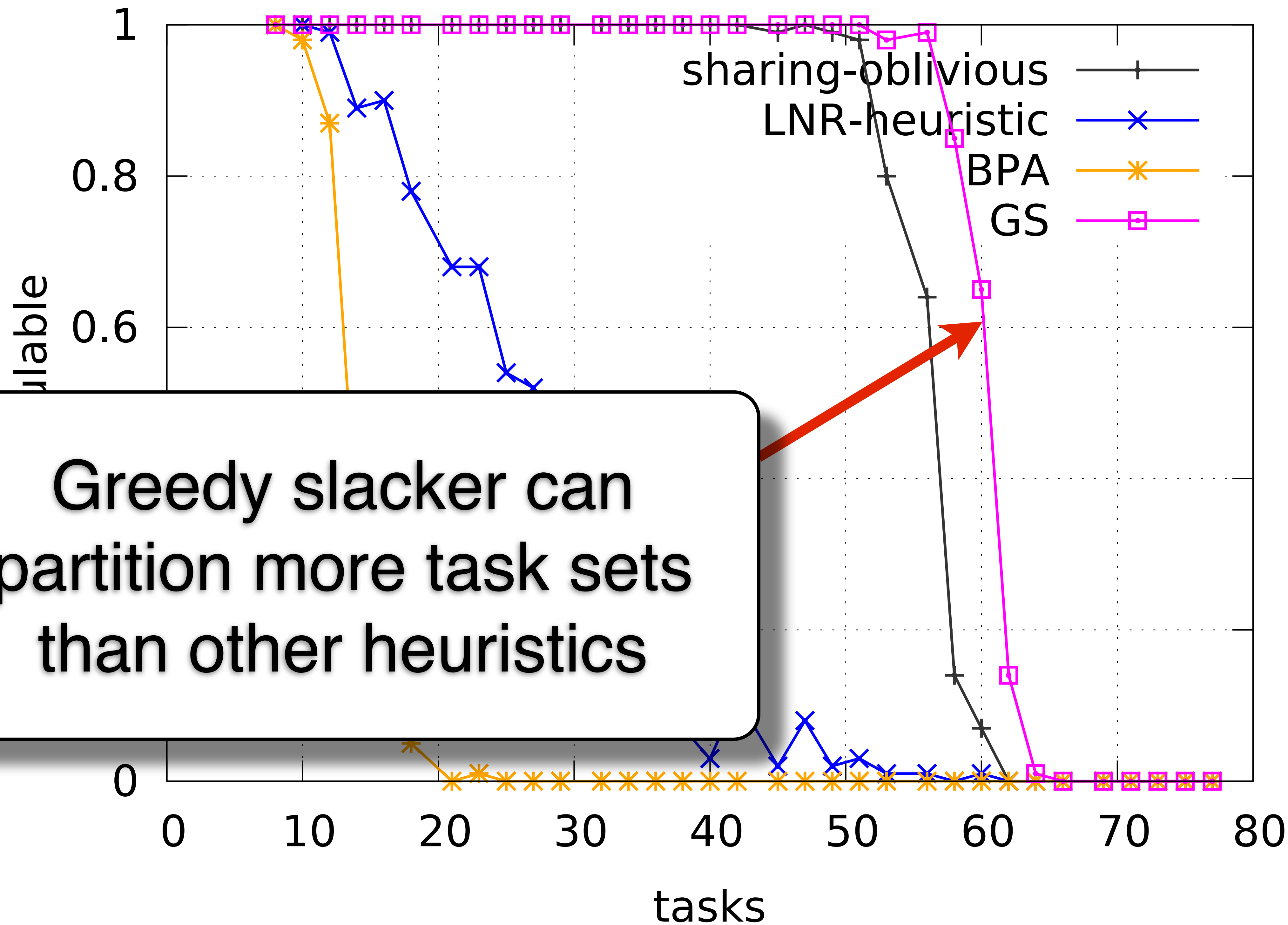
Structured
with
global resources



Unstructured Resource Accesses

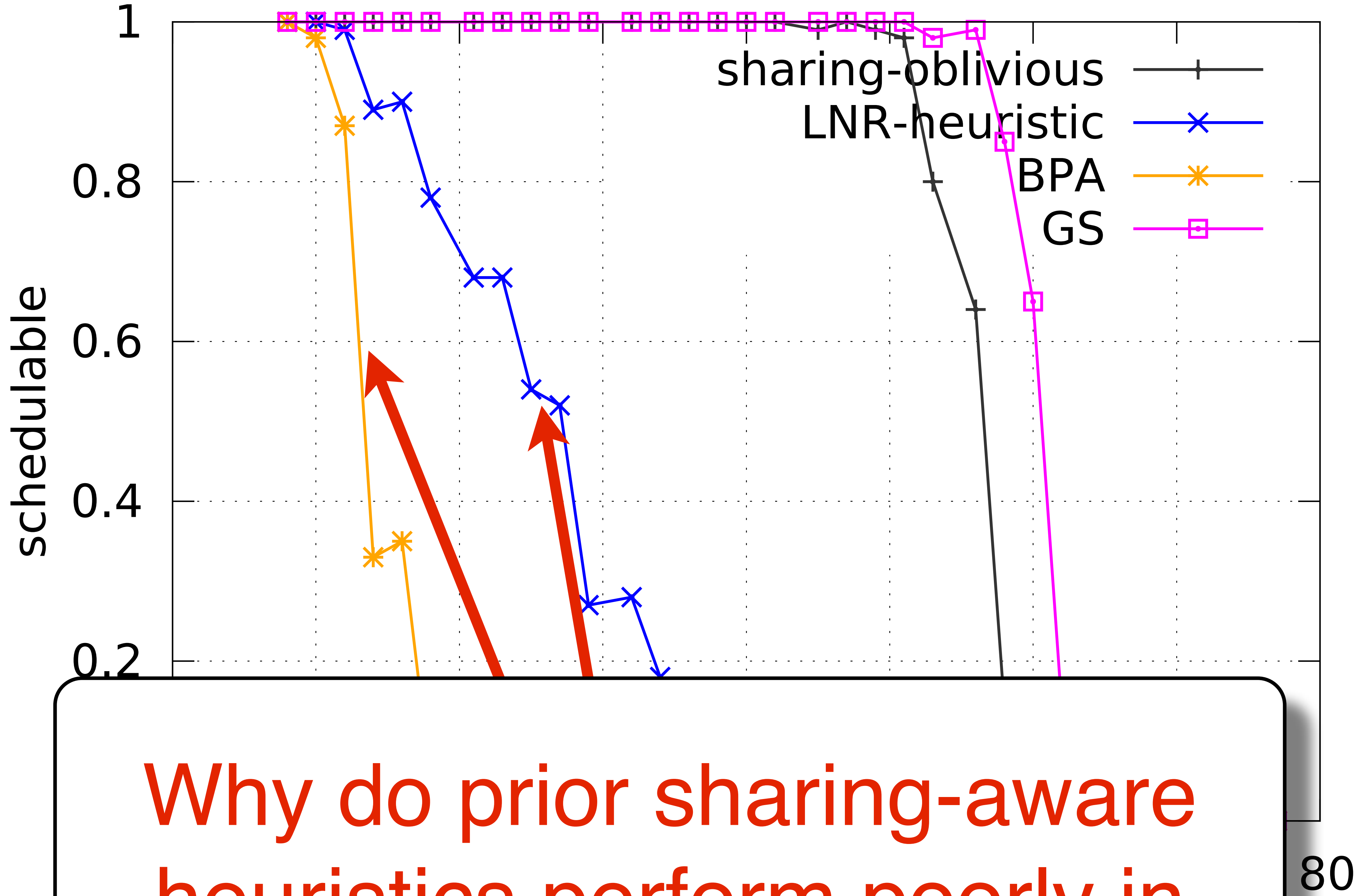


Unstructured Resource Accesses



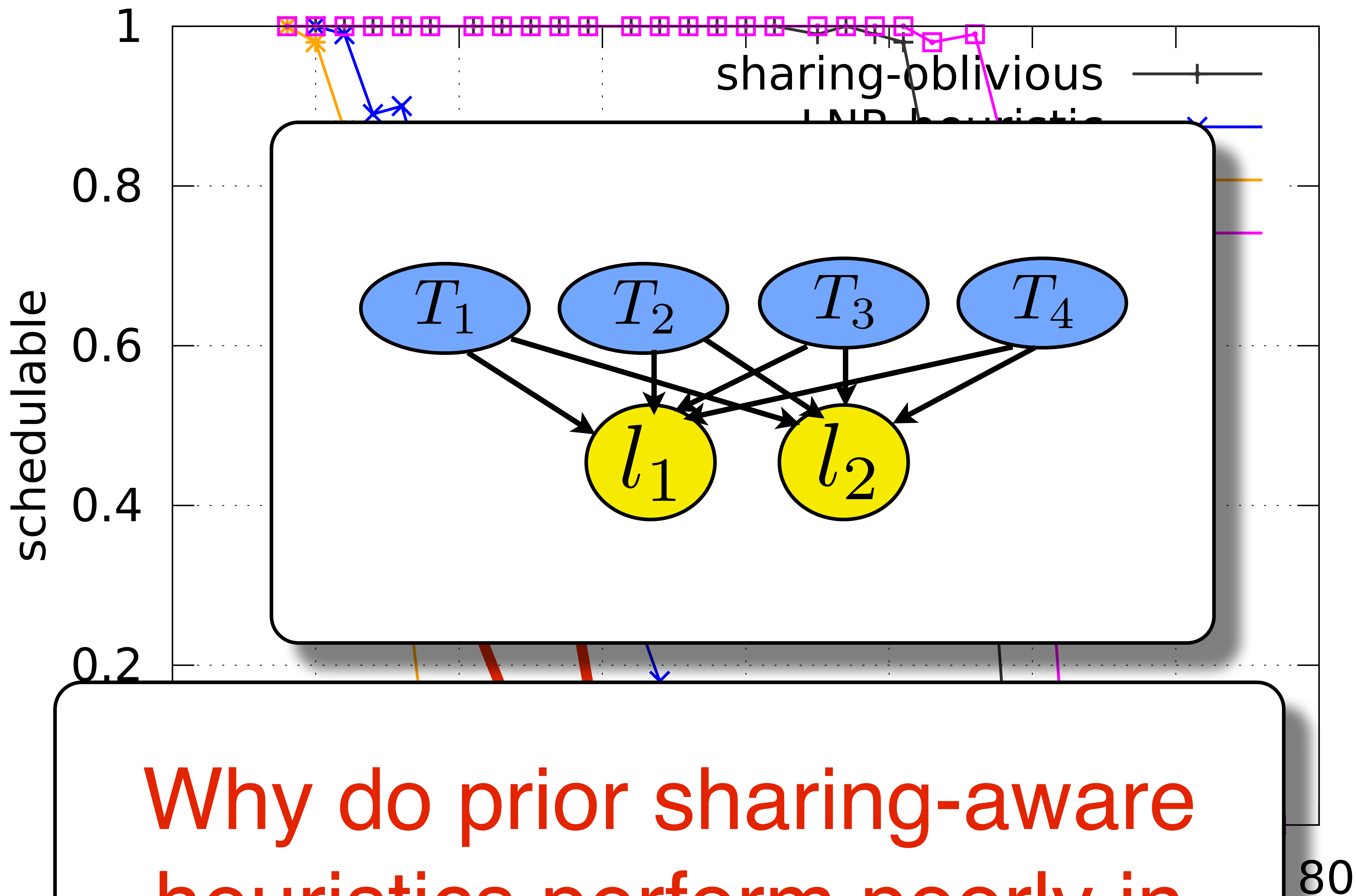
Greedy slacker can partition more task sets than other heuristics

Unstructured Resource Accesses



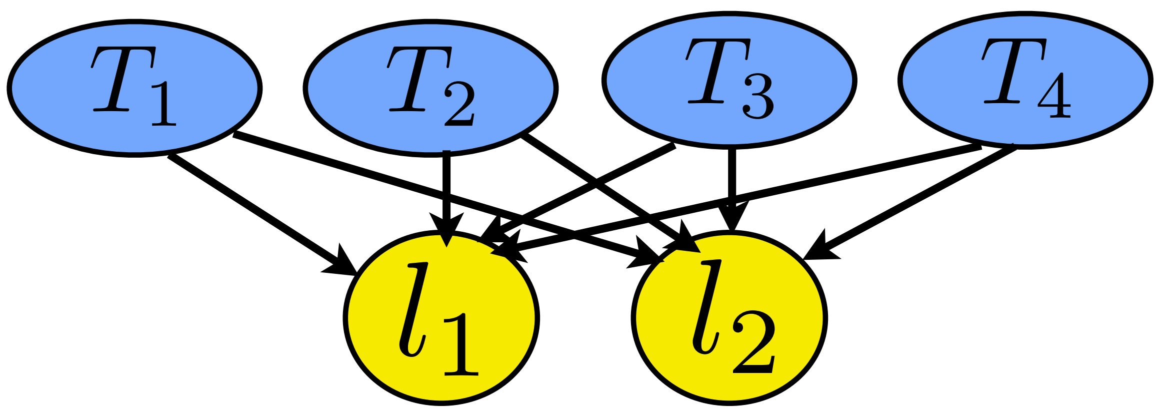
Why do prior sharing-aware heuristics perform poorly in this scenario?

Unstructured Resource Accesses



Why do prior sharing-aware heuristics perform poorly in this scenario?

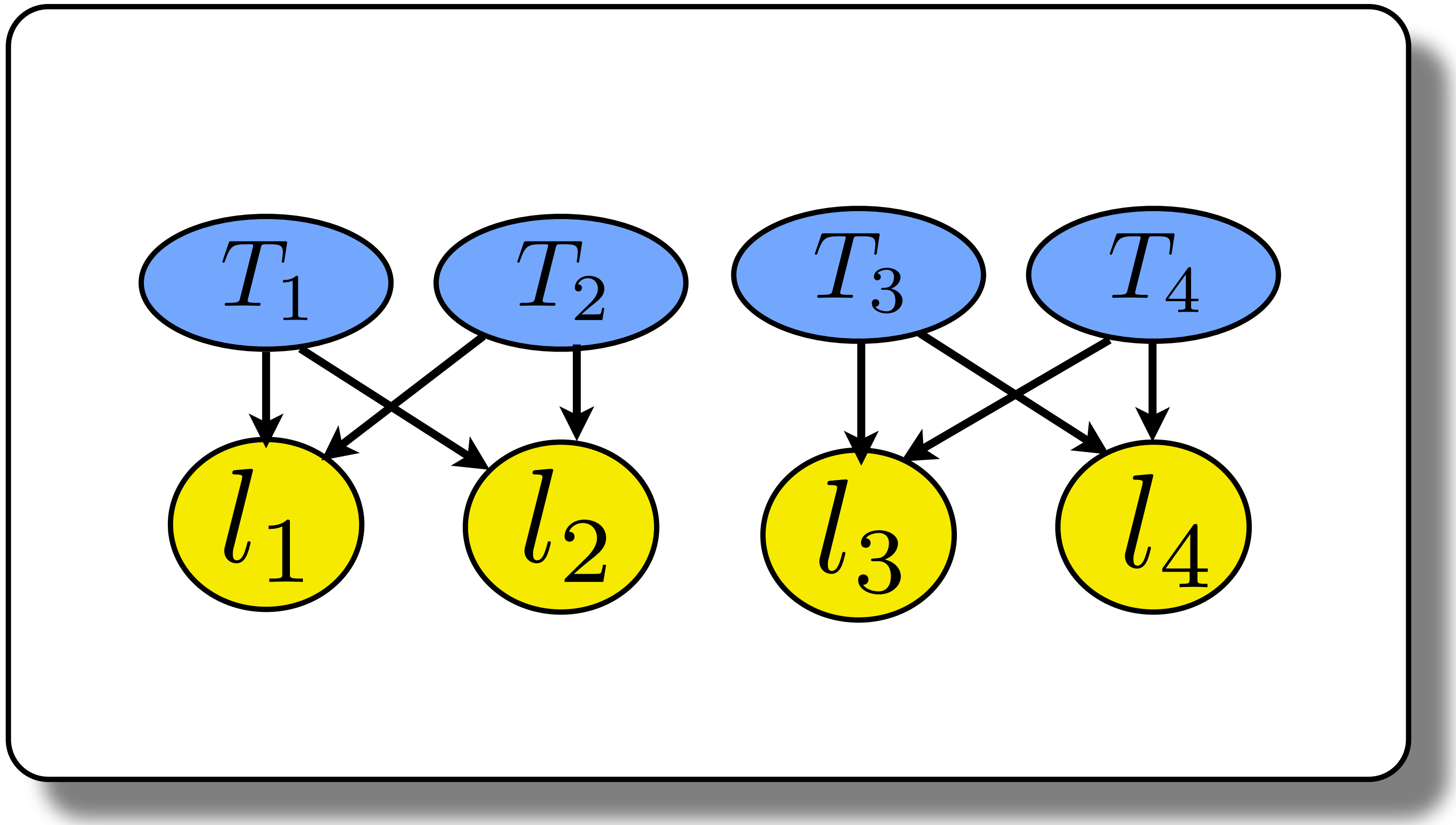
Unstructured Resource Accesses



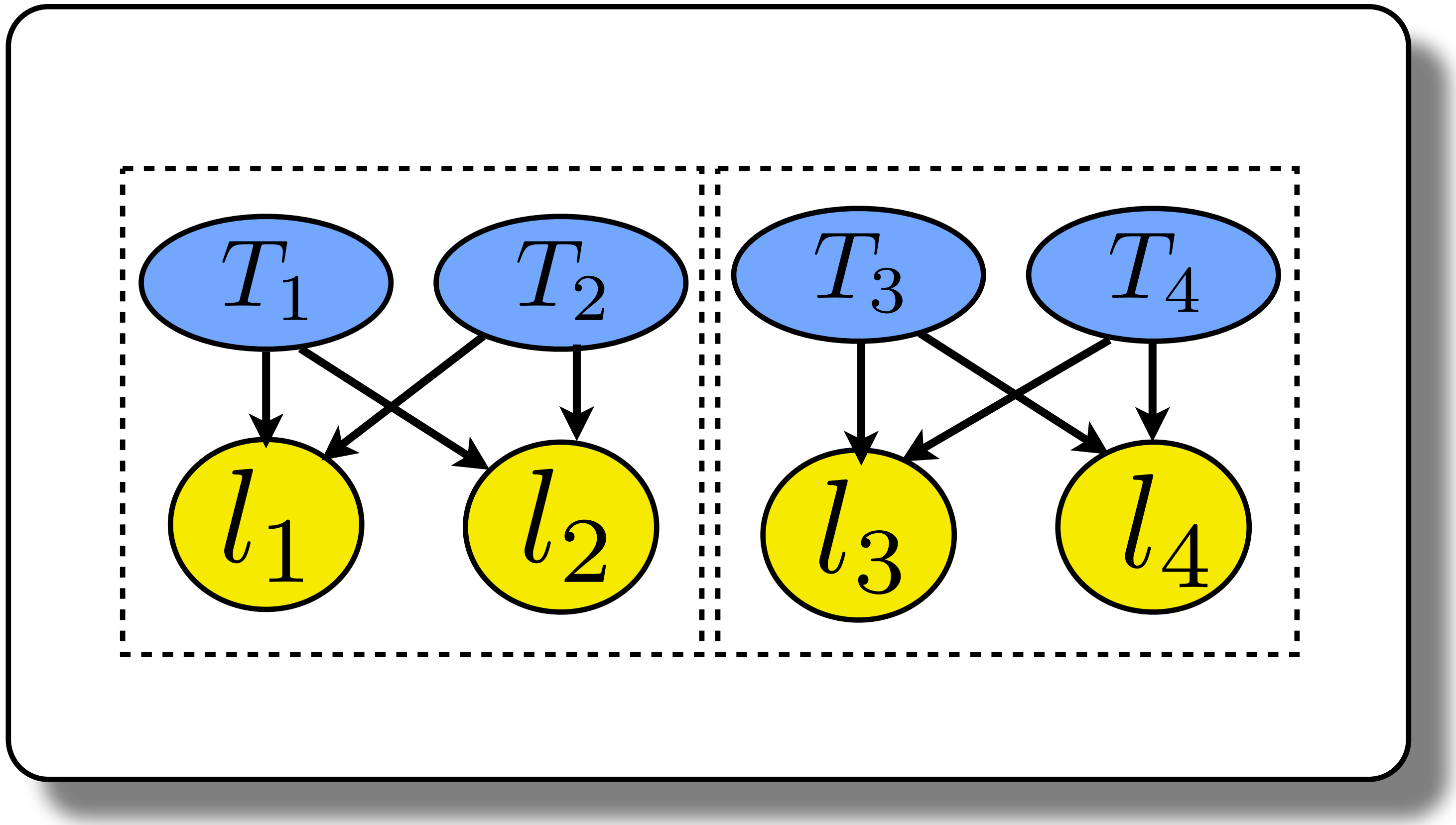
No convenient *structure* that can be exploited by heuristics!

Why do prior sharing-aware heuristics perform poorly in this scenario?

Structured Resource Accesses

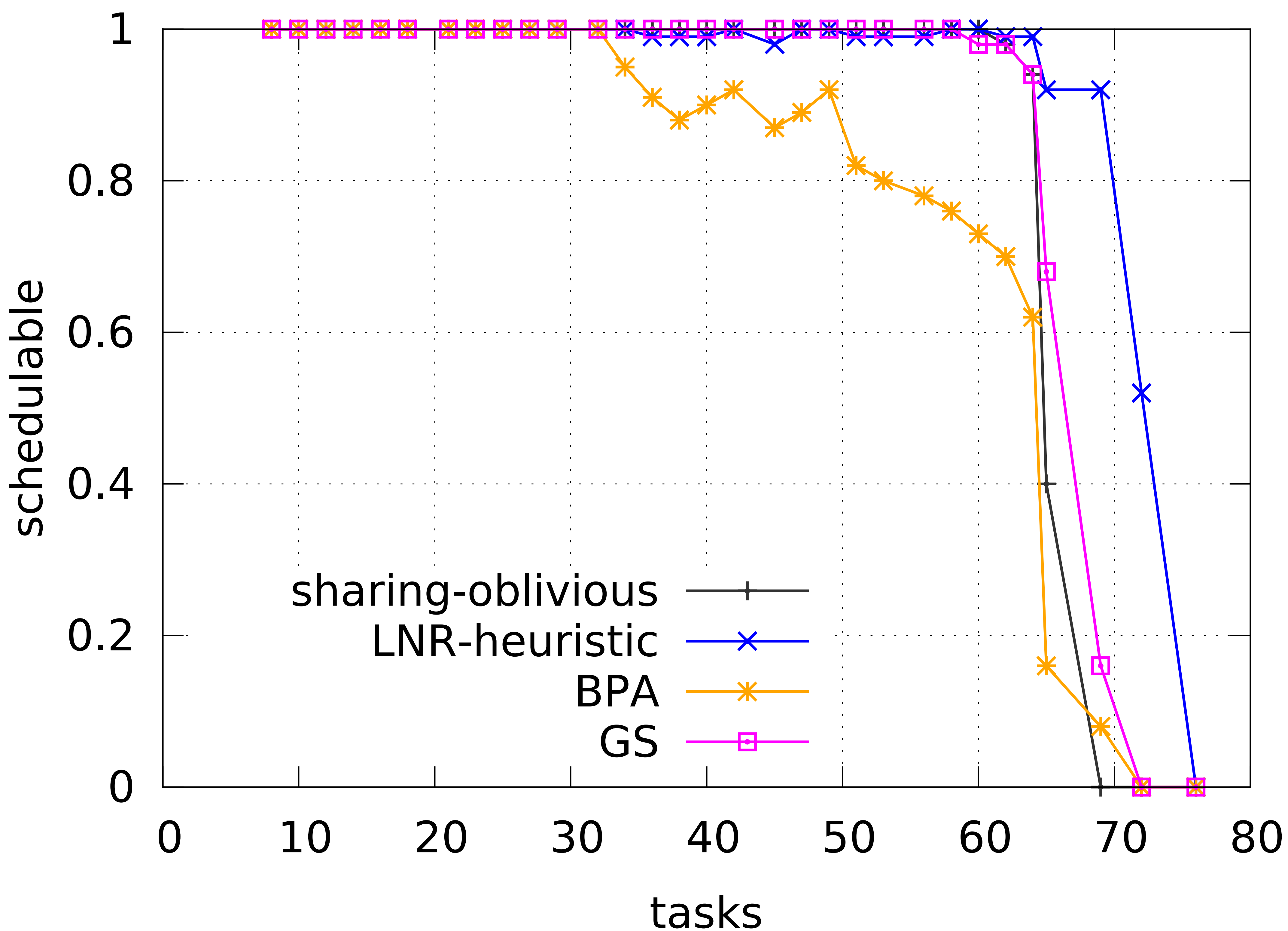


Structured Resource Accesses

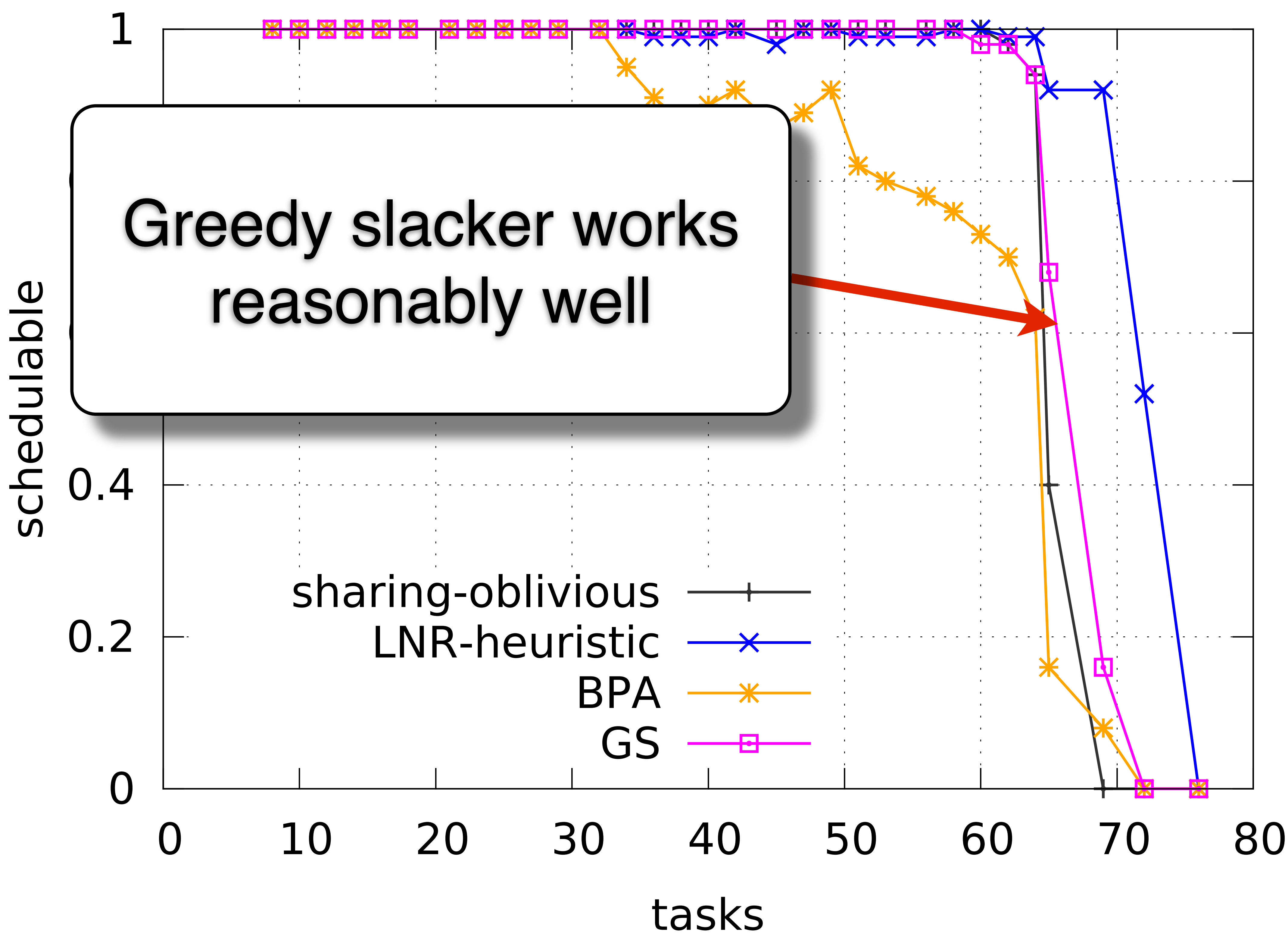


Grouping of tasks and resources into functional components

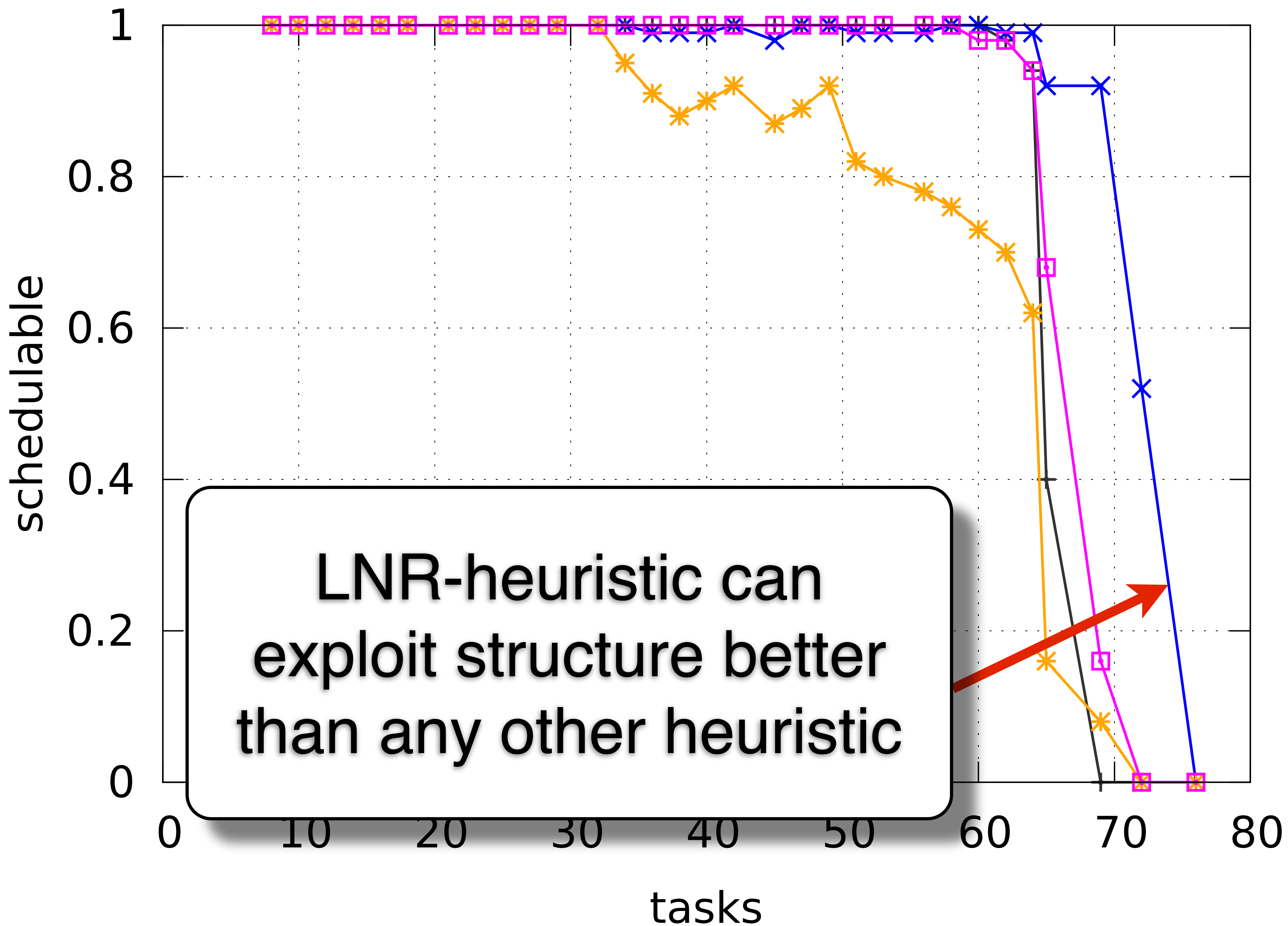
Structured Resource Accesses



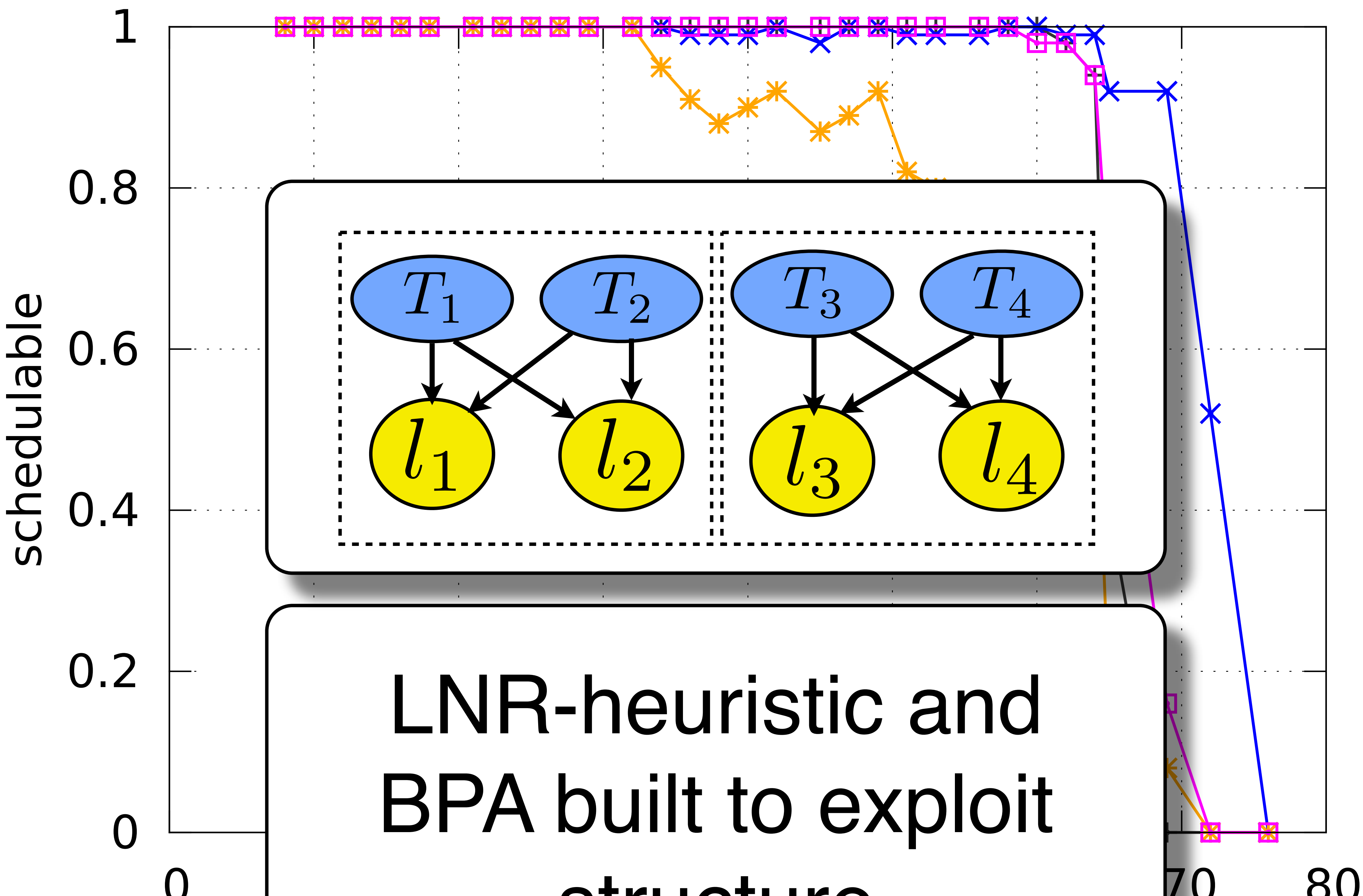
Structured Resource Accesses



Structured Resource Accesses

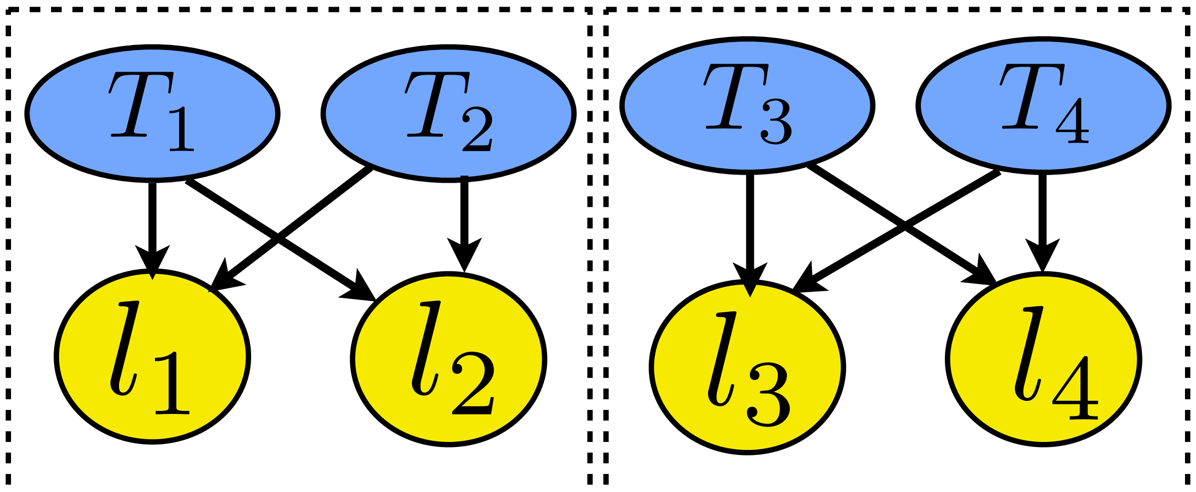


Structured Resource Accesses



LNR-heuristic and BPA built to exploit structure

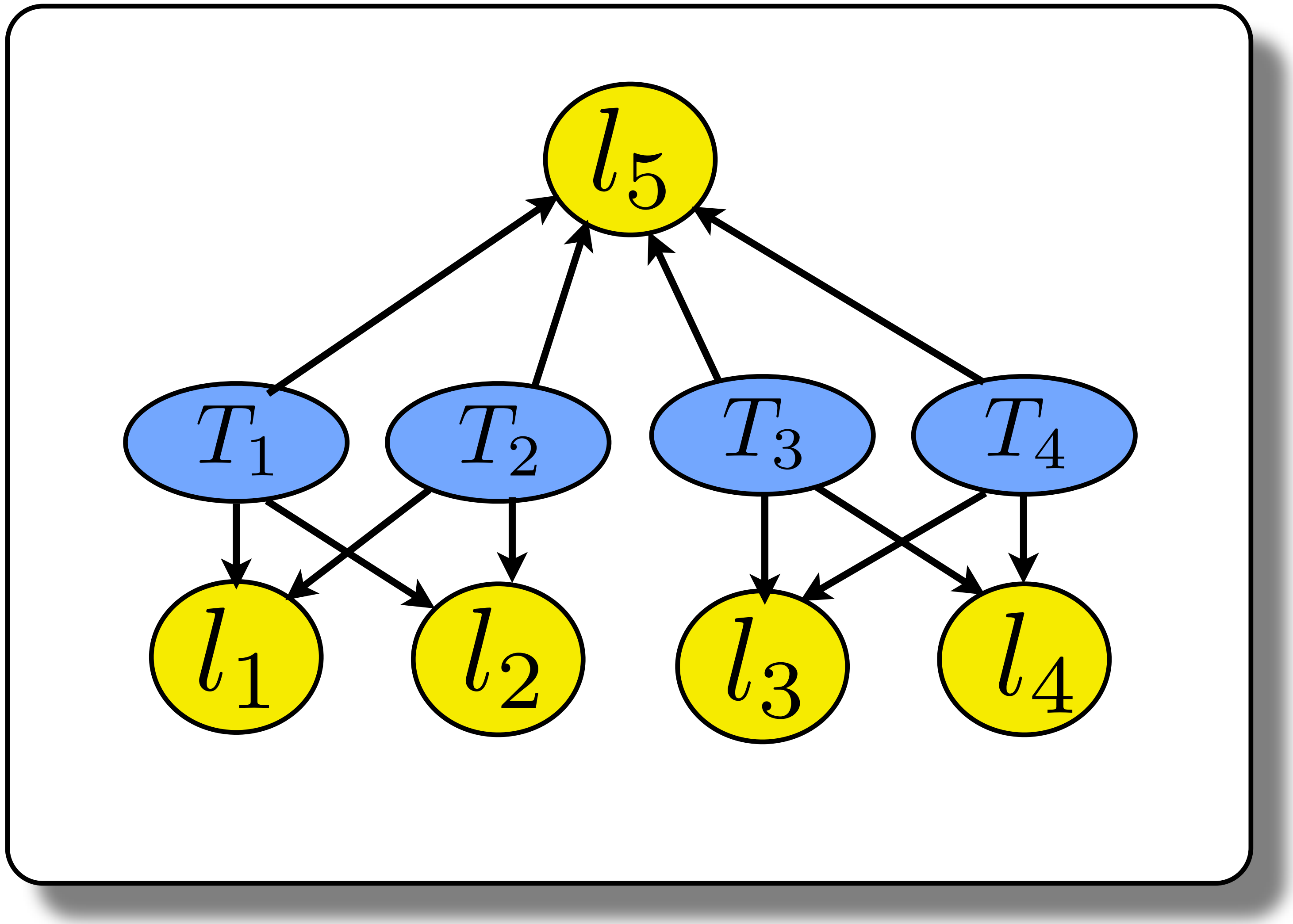
Structured Resource Accesses



In practice, some resources (e.g., kernel objects) are shared among **all** tasks.

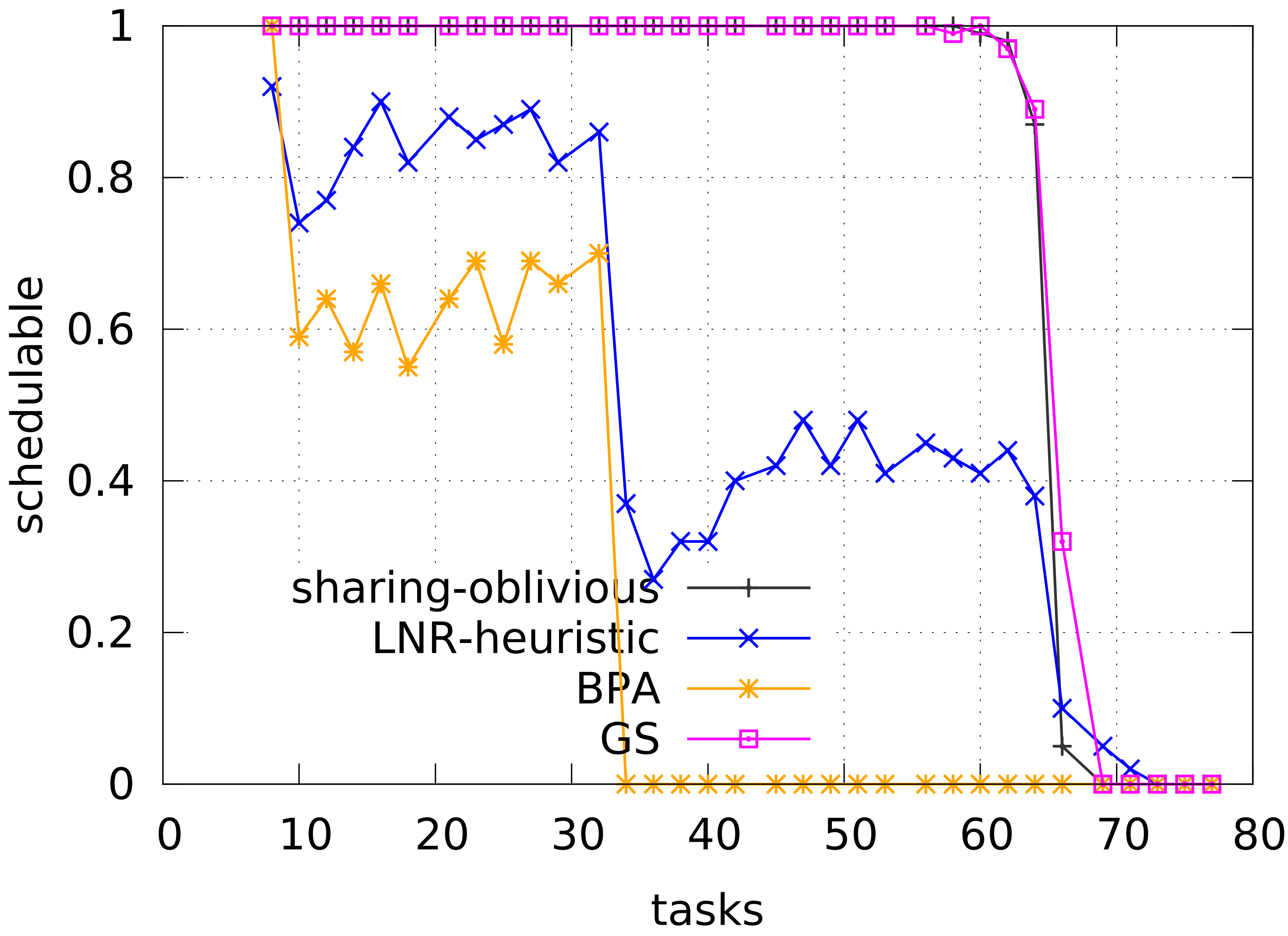
LNR-heuristic and BPA built to exploit structure

Structured Resource Accesses with global resources



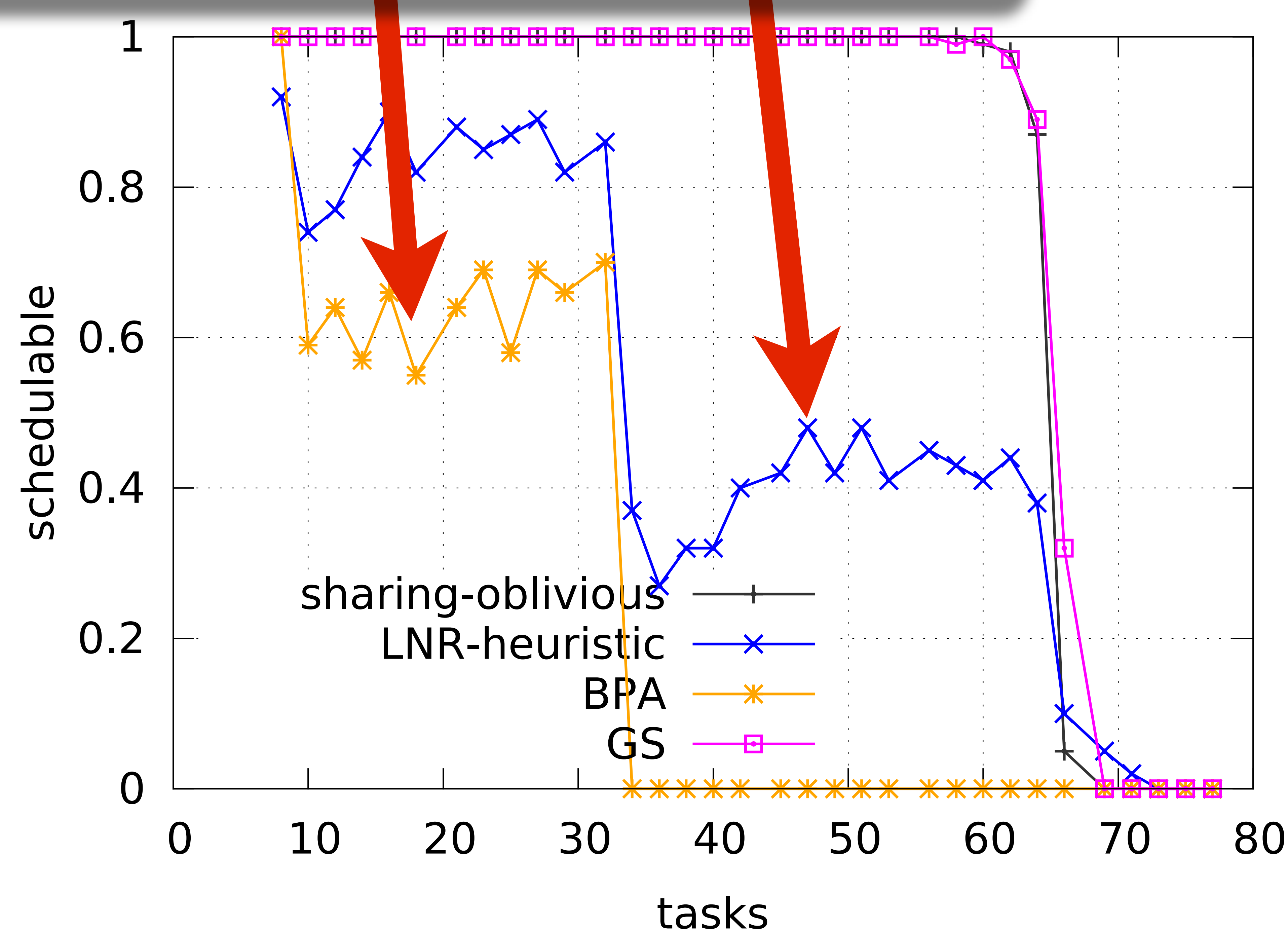
Grouping of tasks and resources
into functional components,
some resources access by all tasks

Structured Resource Accesses with global resources



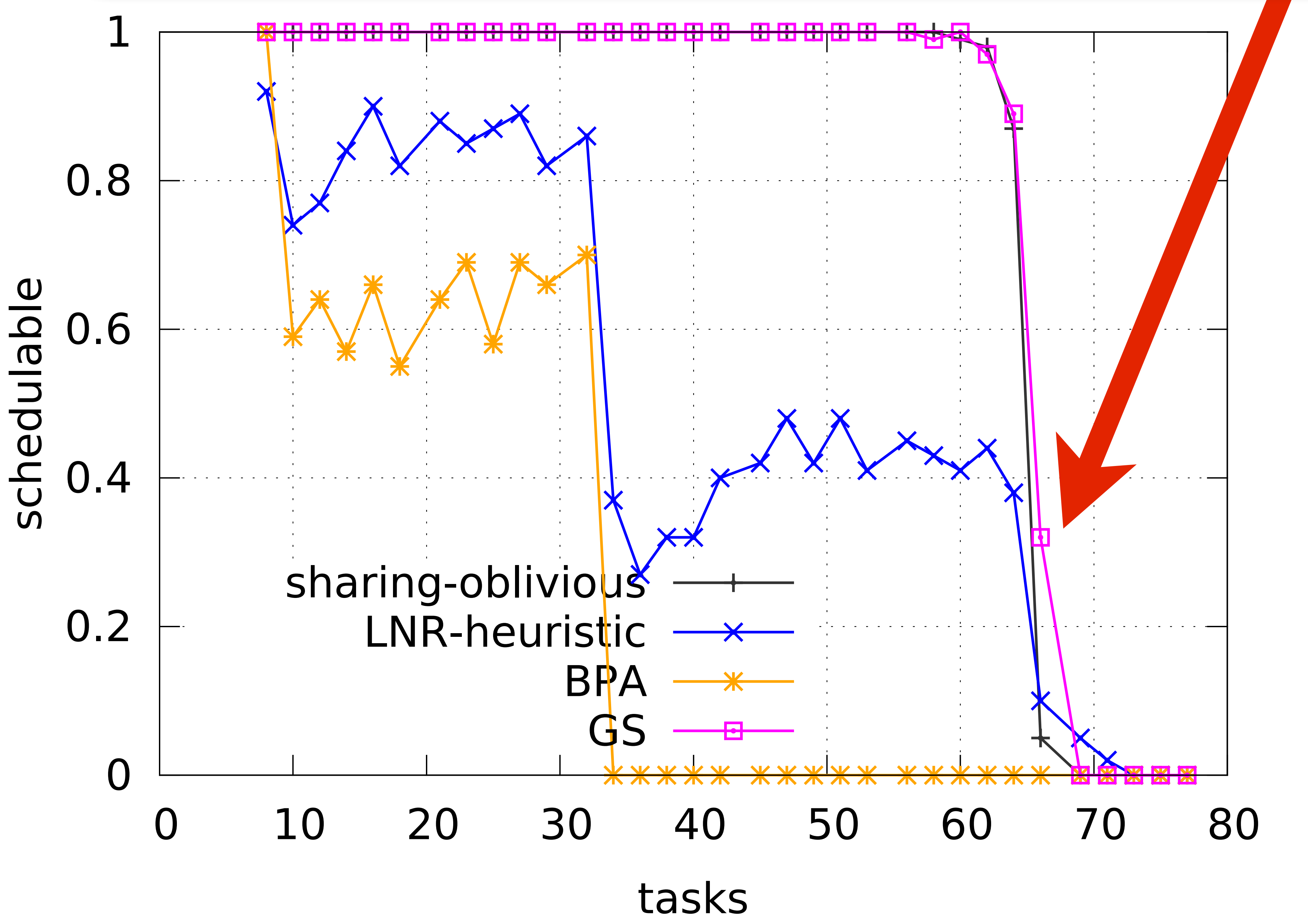
ccesses
ces

LNR-heuristic and BPA suffer from a single global resource

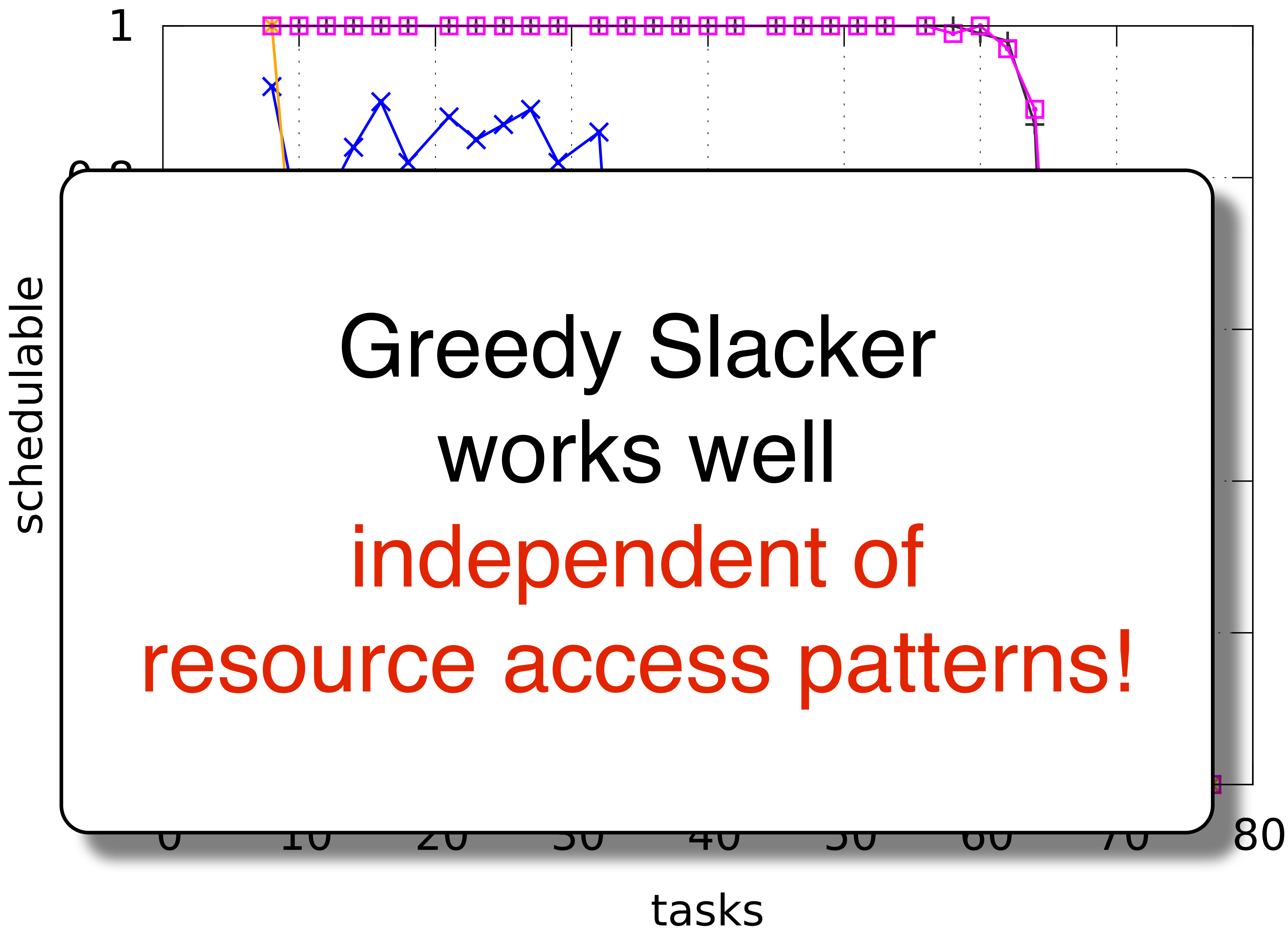


Stru

Greedy slacker can partition (slightly) more task sets than other heuristics



Structured Resource Accesses with global resources



Greedy Slacker
works well
independent of
resource access patterns!

Summary

Optimal partitioning matters
in the face of
shared resources
protected by spin locks.

Blocking due to spin locks in the MSRP
can be expressed
with purely linear expressions
which allows using ILP techniques.

Summary

Fast and robust sharing-aware partitioning heuristic can be embarrassingly simple.

Thanks!