Multiprocessor feasibility analysis of recurrent task systems with specified processor affinities

RTSS'13
December 5, 2013

Sanjoy Baruah (UNC Chapel Hill)

Björn Brandenburg (MPI-SWS)

THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

Max Planck Institute for Software Systems

Contribution

*A **polynomial-time algorithm** to decide whether a set of **implicit-deadline sporadic tasks** with **arbitrary processor affinities** is feasible.*

Contribution

*A **polynomial-time algorithm** to decide whether a set of **implicit-deadline sporadic tasks** with **arbitrary processor affinities** is feasible.*

This Talk

- ➡ What are **processor affinities** and **why do they matter**?
- ➡ Overview of the feasibility test.
- ➡ **What's left to do?**

Part 1

What are Processor Affinities?

Processor Affinities in Practice

Multiprocessor environment: users seek to
***restrict* on which processors a given task may run…**

Processor Affinities in Practice

Multiprocessor environment: users seek to
restrict on which processors a given task may run…

Why?

- ➡ cache conflicts
- ➡ fault tolerance
- ➡ covert channels
- ➡ resource sharing
- ➡ …

Processor Affinities in Practice

Multiprocessor environment: users seek to
restrict on which processors a given task may run…

Why?

- ➡ cache conflicts
- ➡ fault tolerance
- ➡ covert channels
- ➡ resource sharing
- ➡ …

How?

- ➡ Linux: **`sched_setaffinity()`**
- ➡ FreeBSD: **`cpuset_setaffinity()`**
- ➡ Windows: **`SetThreadAffinityMask()`**
- ➡ QNX: **`ThreadCtl(_NTO_TCTL_RUNMASK)`**
- ➡ VxWorks: **`taskCpuAffinitySet()`**

Processor Affinities in Practice

A task's ***processor affinity (PA)*** *is the* ***subset of processors*** *on which it* ***may be scheduled****.*

Why?

- cache conflicts
- fault tolerance
- covert channels
- resource sharing
- …

How?

- Linux: `sched_setaffinity()`
- FreeBSD: `cpuset_setaffinity()`
- Windows: `SetThreadAffinityMask()`
- QNX: `ThreadCtl(_NTO_TCTL_RUNMASK)`
- VxWorks: `taskCpuAffinitySet()`

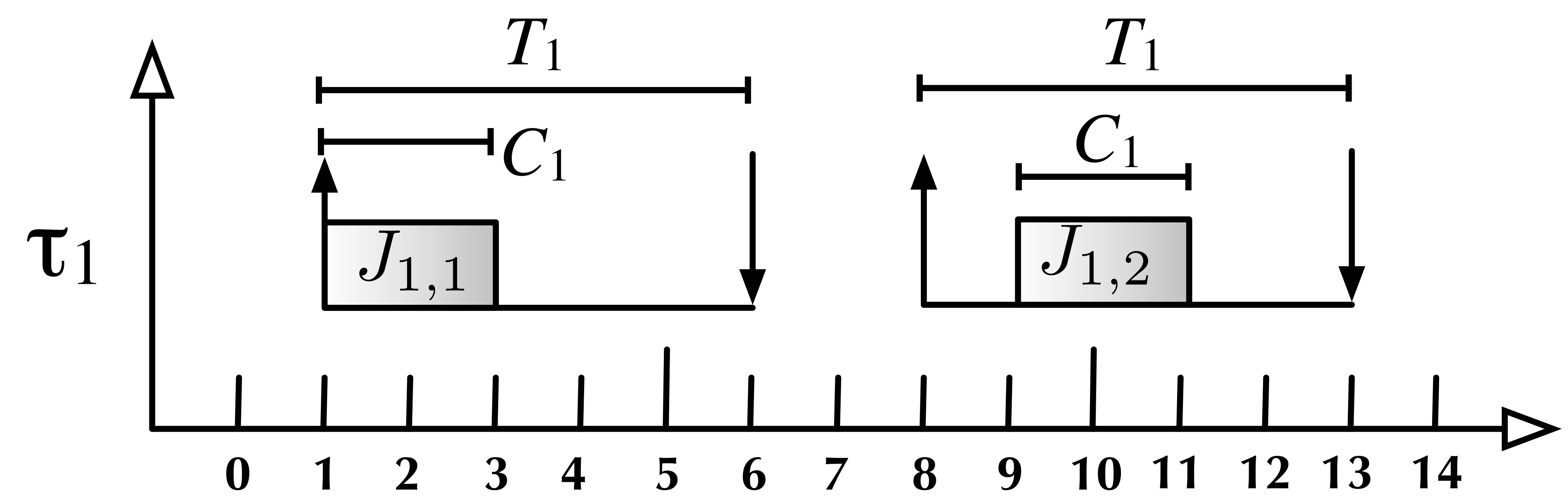
Formalizing Arbitrary Processor Affinities (APAs)

System Model

- ***n sporadic tasks*** $\tau_1, \ldots, \tau_n$
- ***m identical processors***

Task parameters

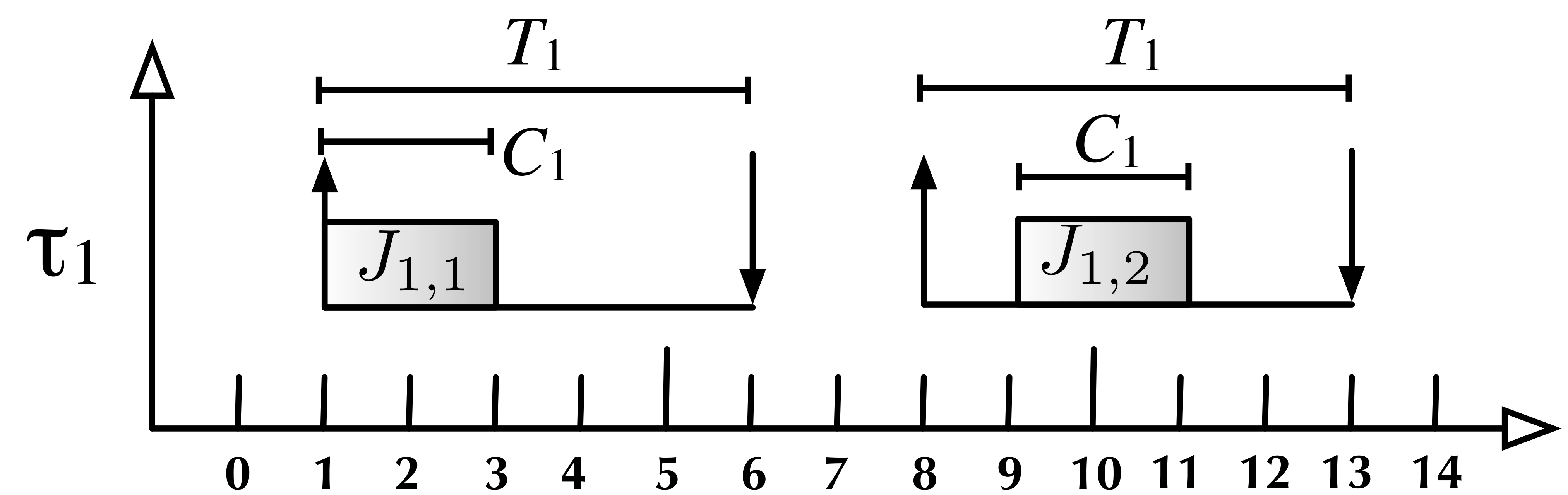
- **Implicit deadlines**
- execution cost C_i
- period T_i
- utilization $u_i = C_i / T_i$



Formalizing Arbitrary Processor Affinities (APAs)

System Model

- n sporadic tasks $\tau_1, \ldots, \tau_n$
- m identical processors

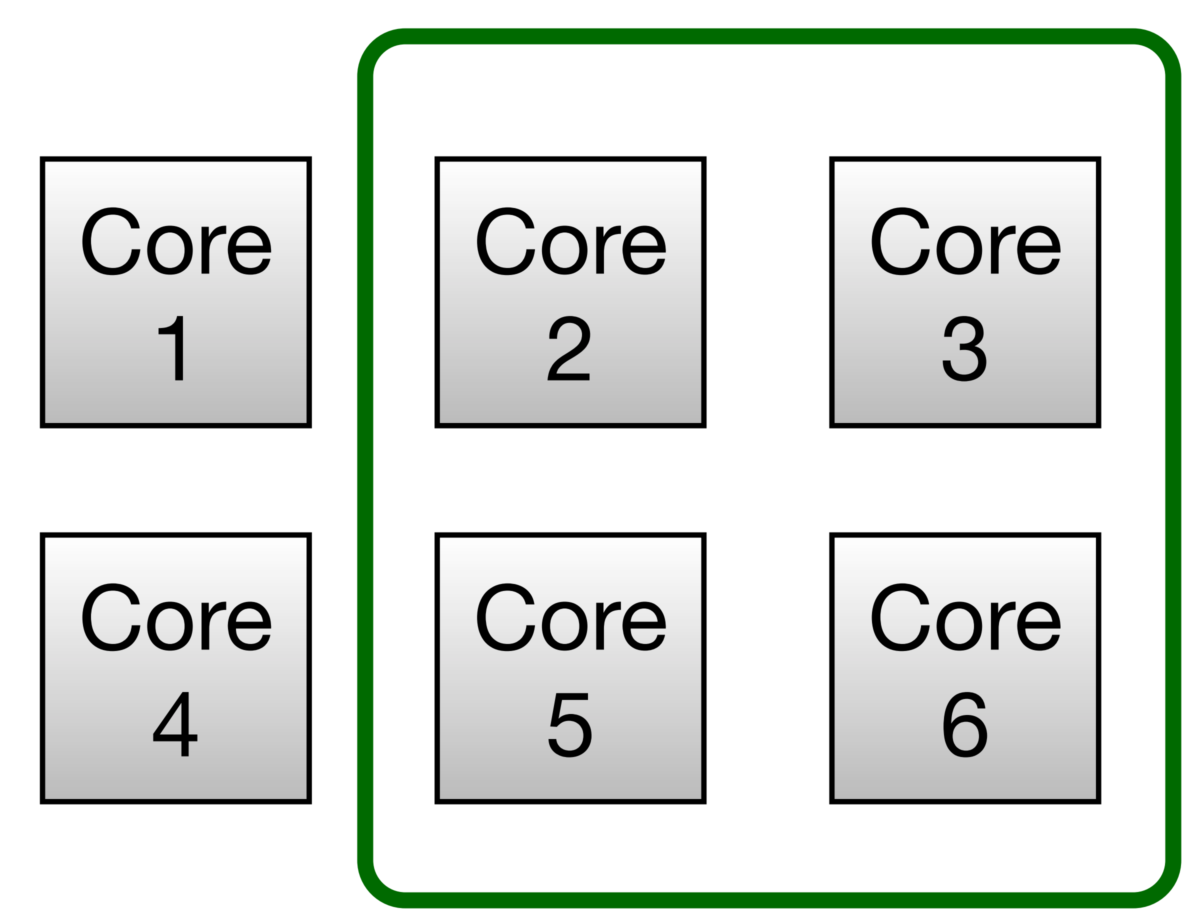


Task parameters

- Implicit deadlines
- execution cost C_i
- period T_i
- utilization $u_i = C_i / T_i$

Processor affinity

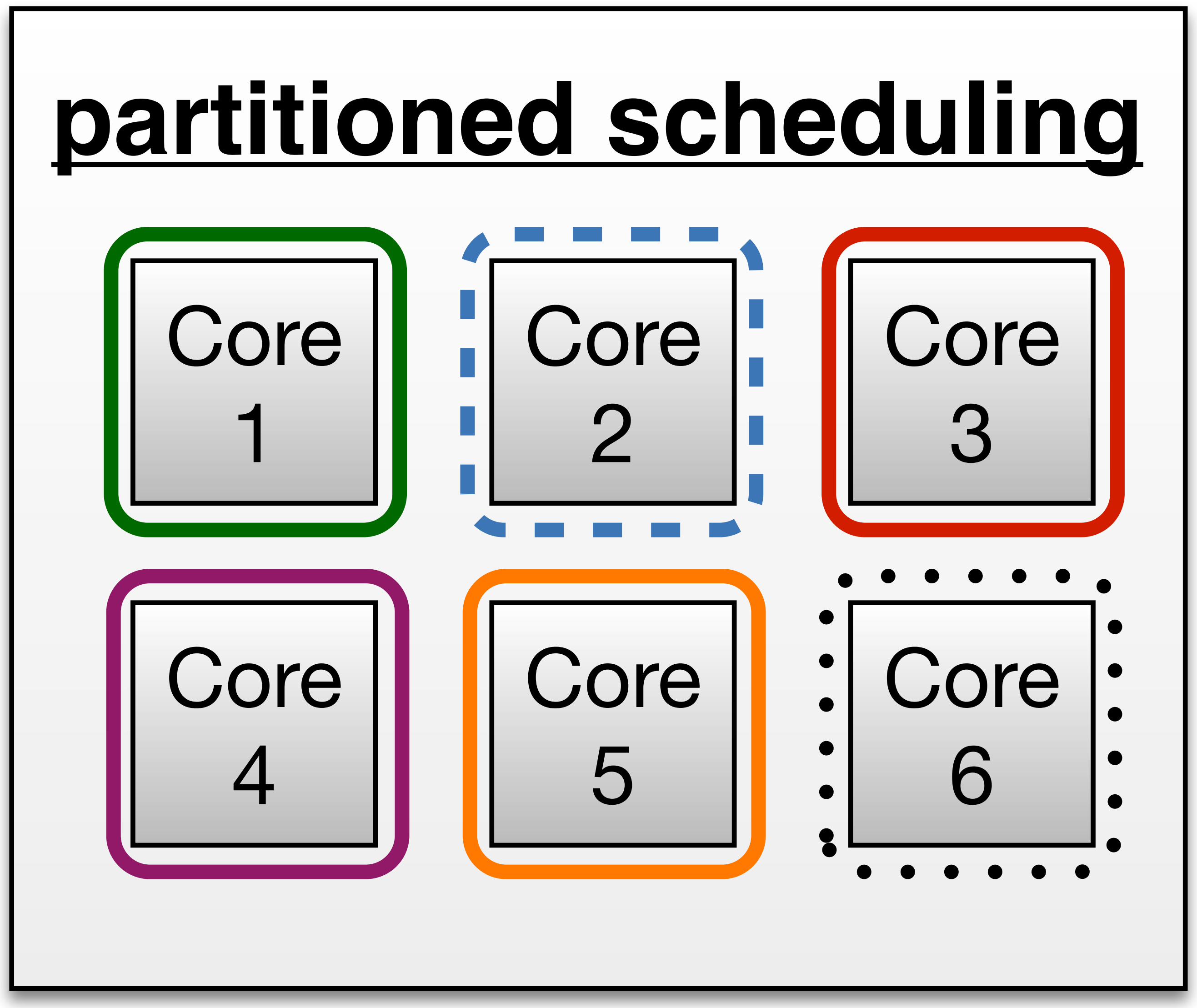
- $\alpha_i \subseteq \{1, 2, \ldots, m\}$

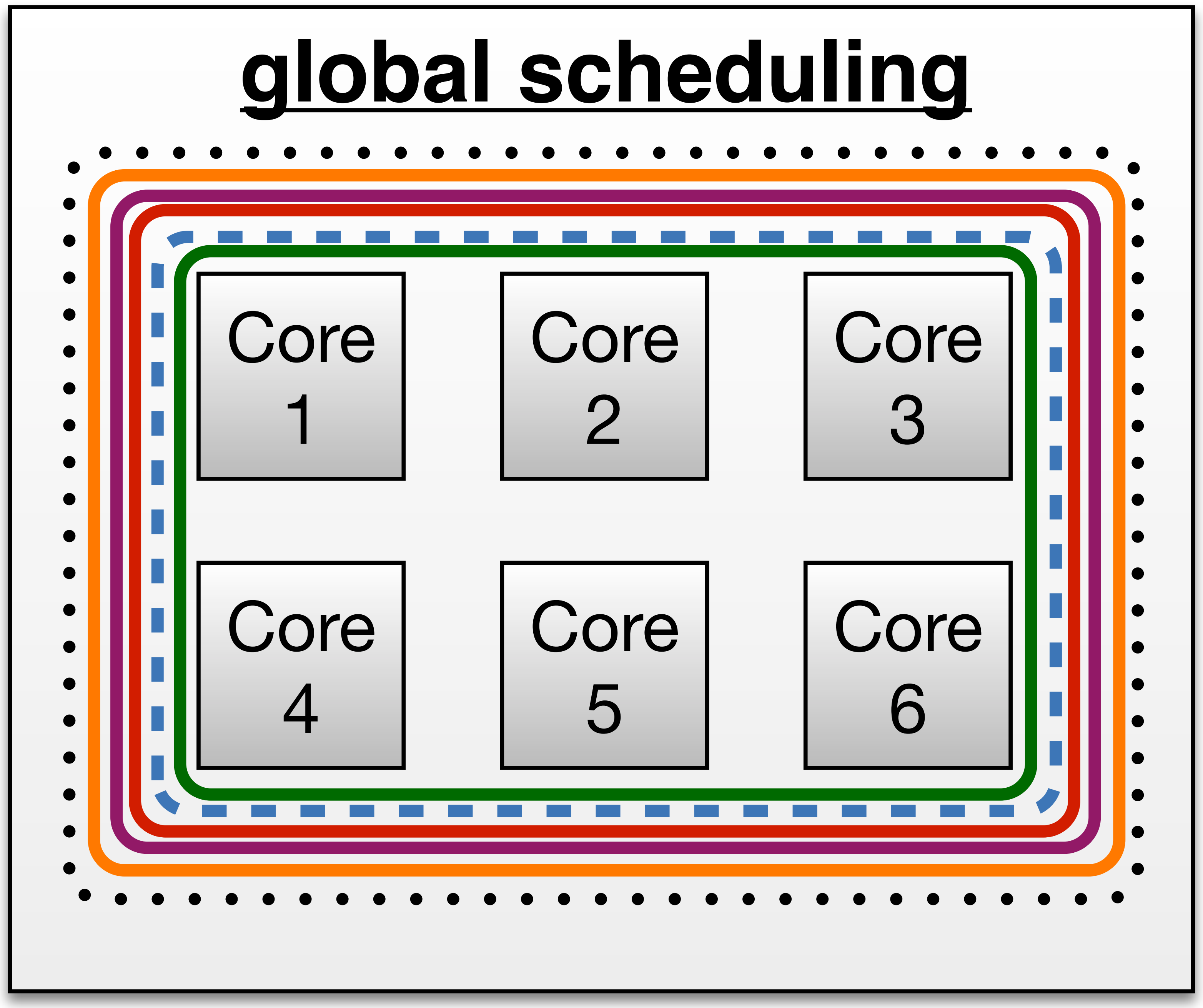


$\alpha_i = \{2, 3, 5, 6\}$

Why is APA Schedulability Analysis Challenging?

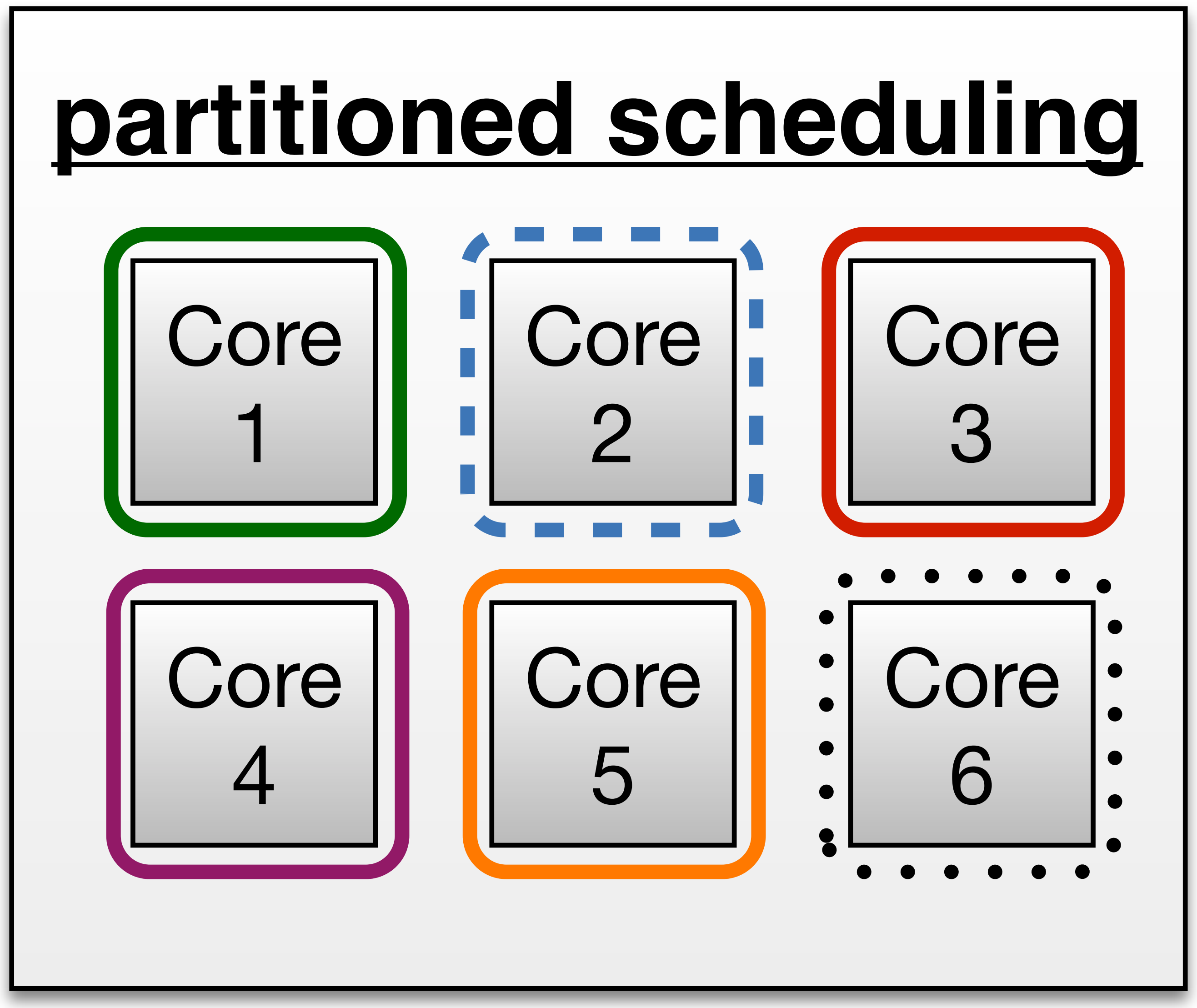
Why is APA Schedulability Analysis Challenging?

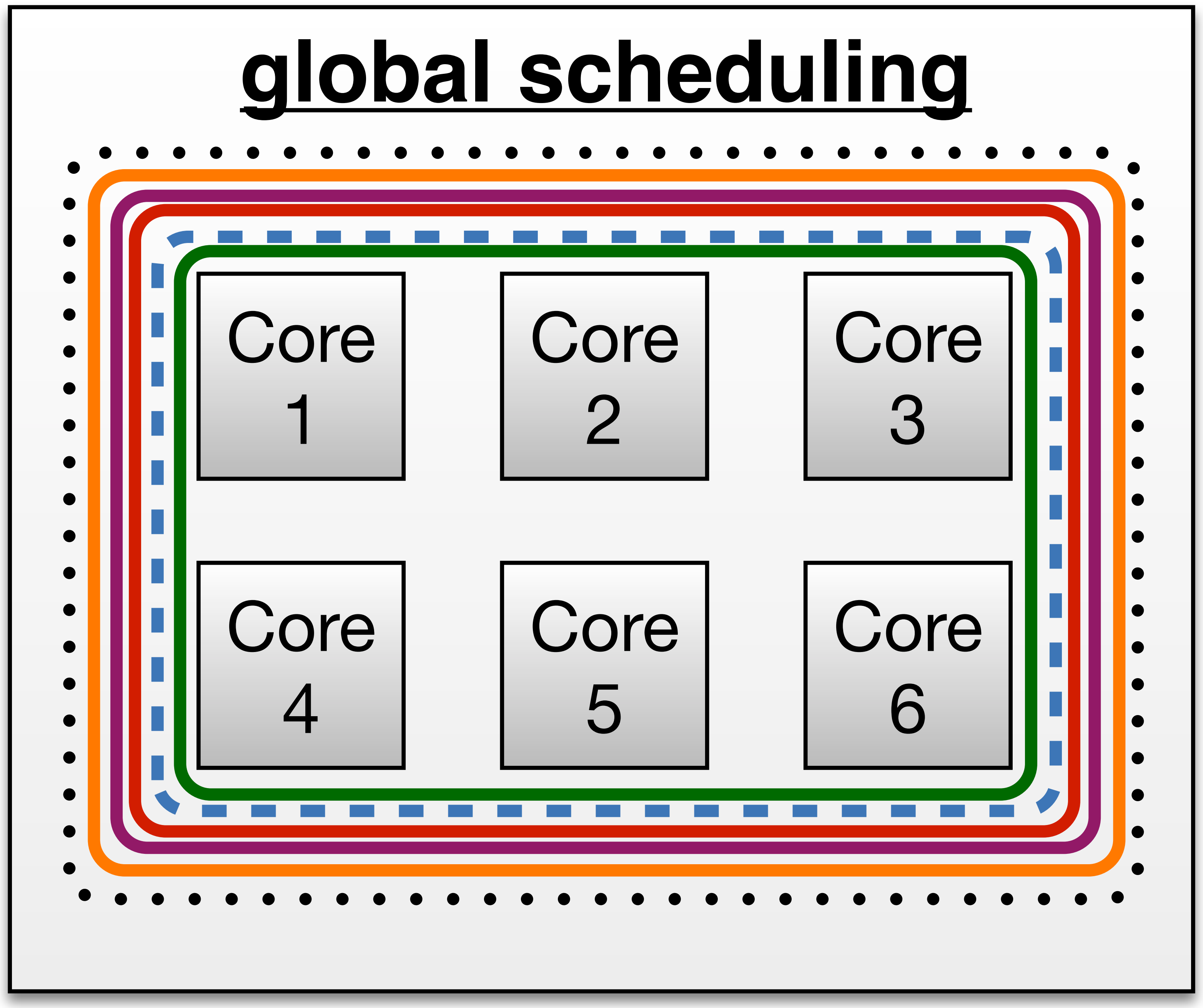


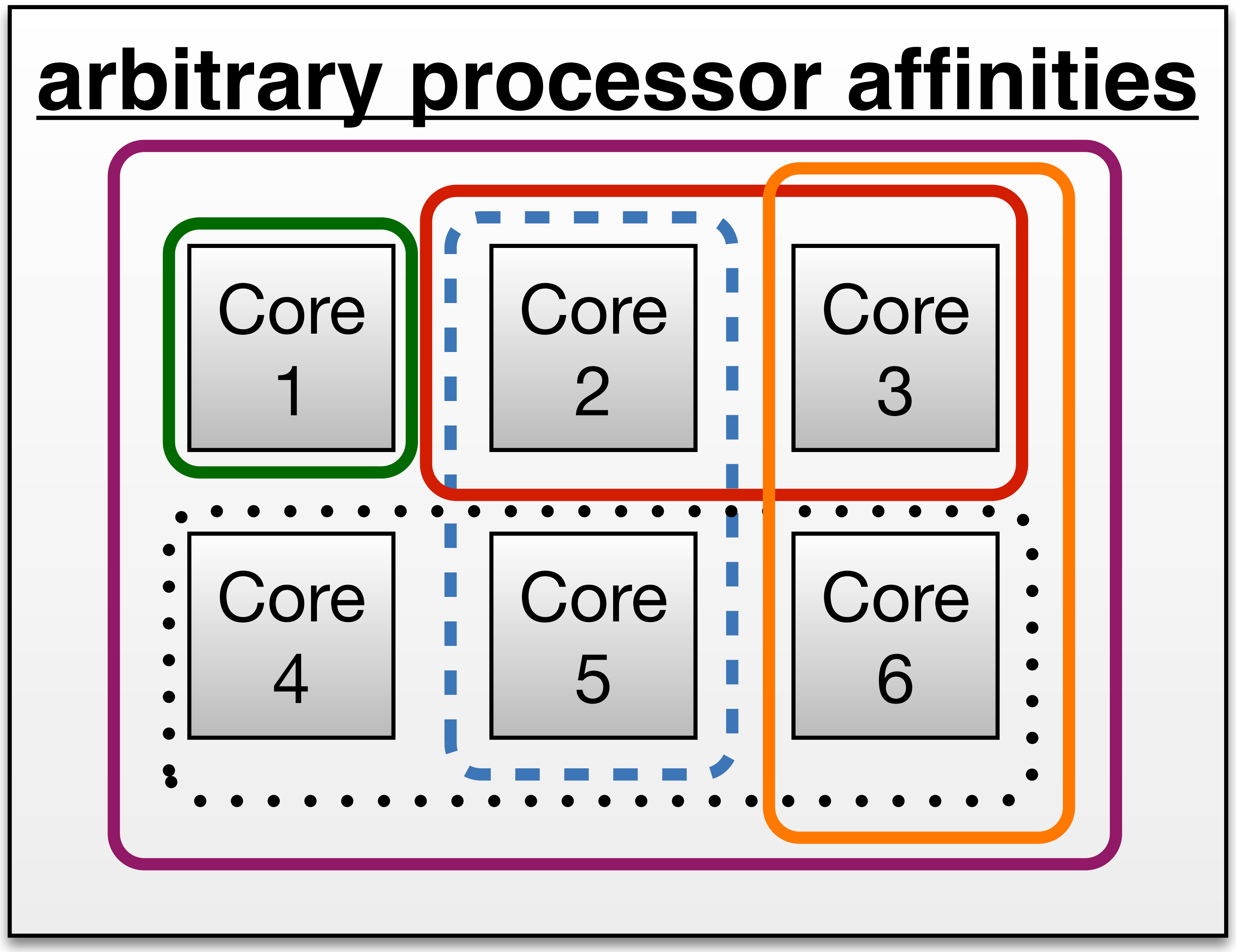


APA scheduling ***generalizes*** *partitioned, global, clustered…*

Why is APA Schedulability Analysis Challenging?

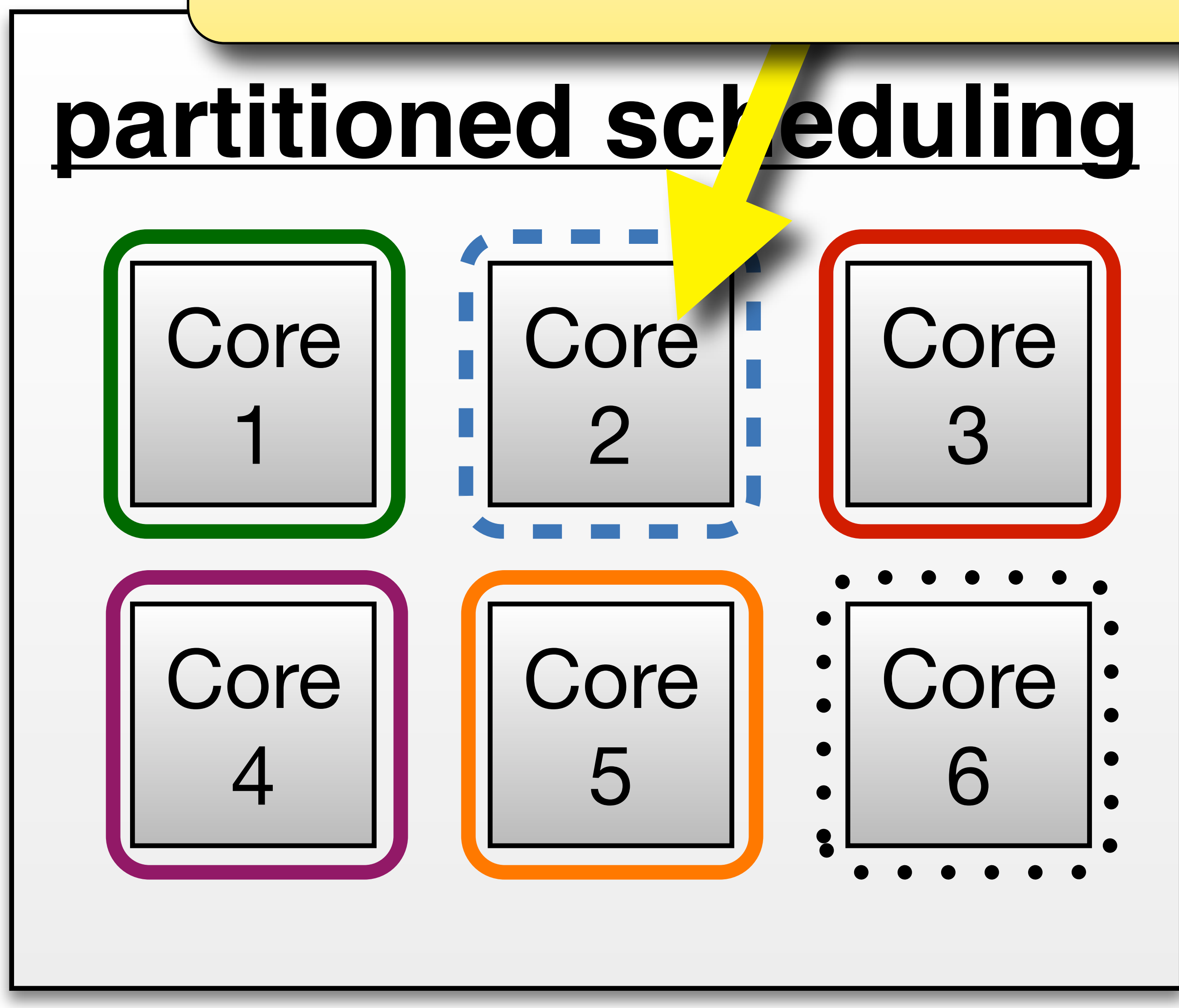




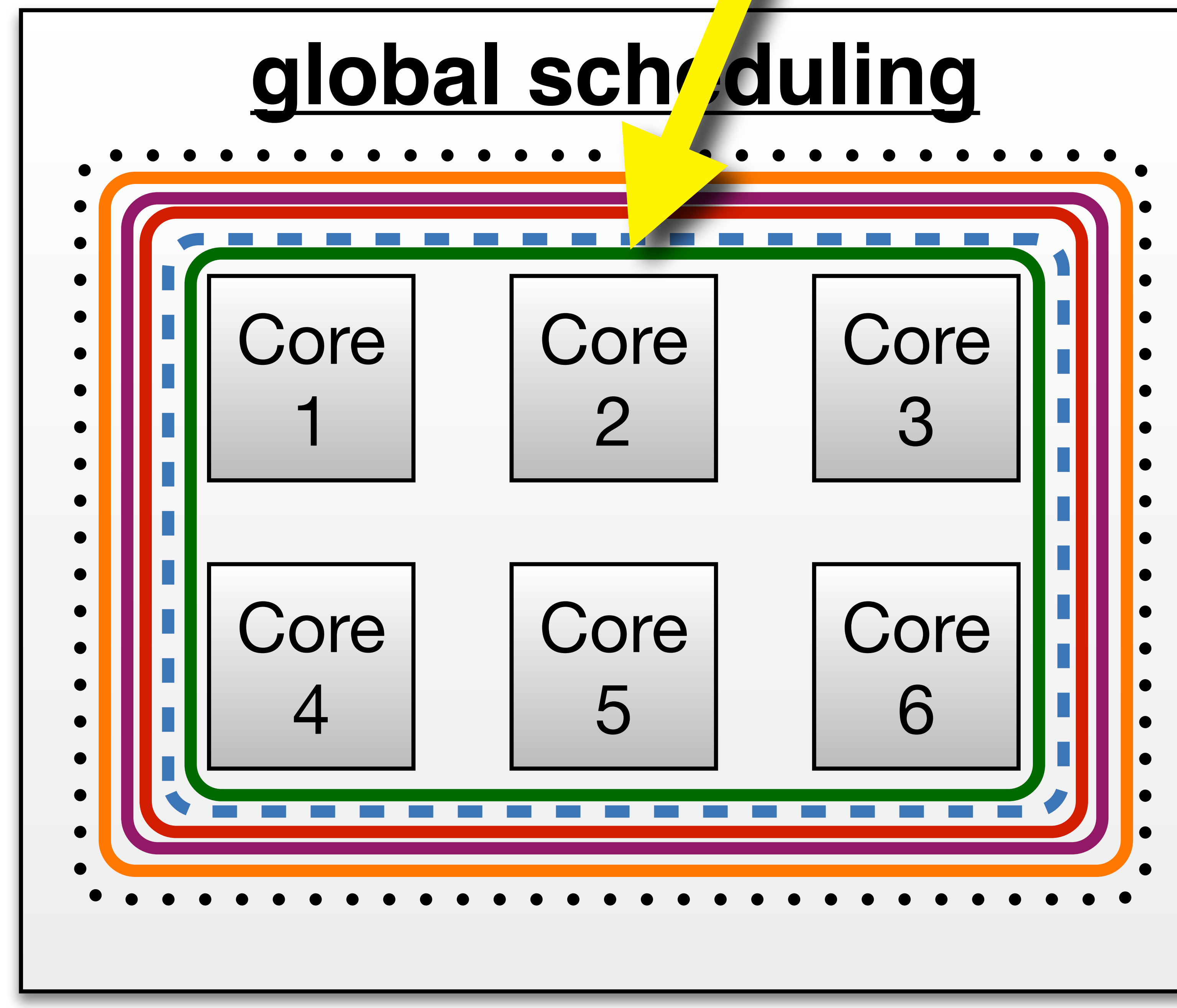


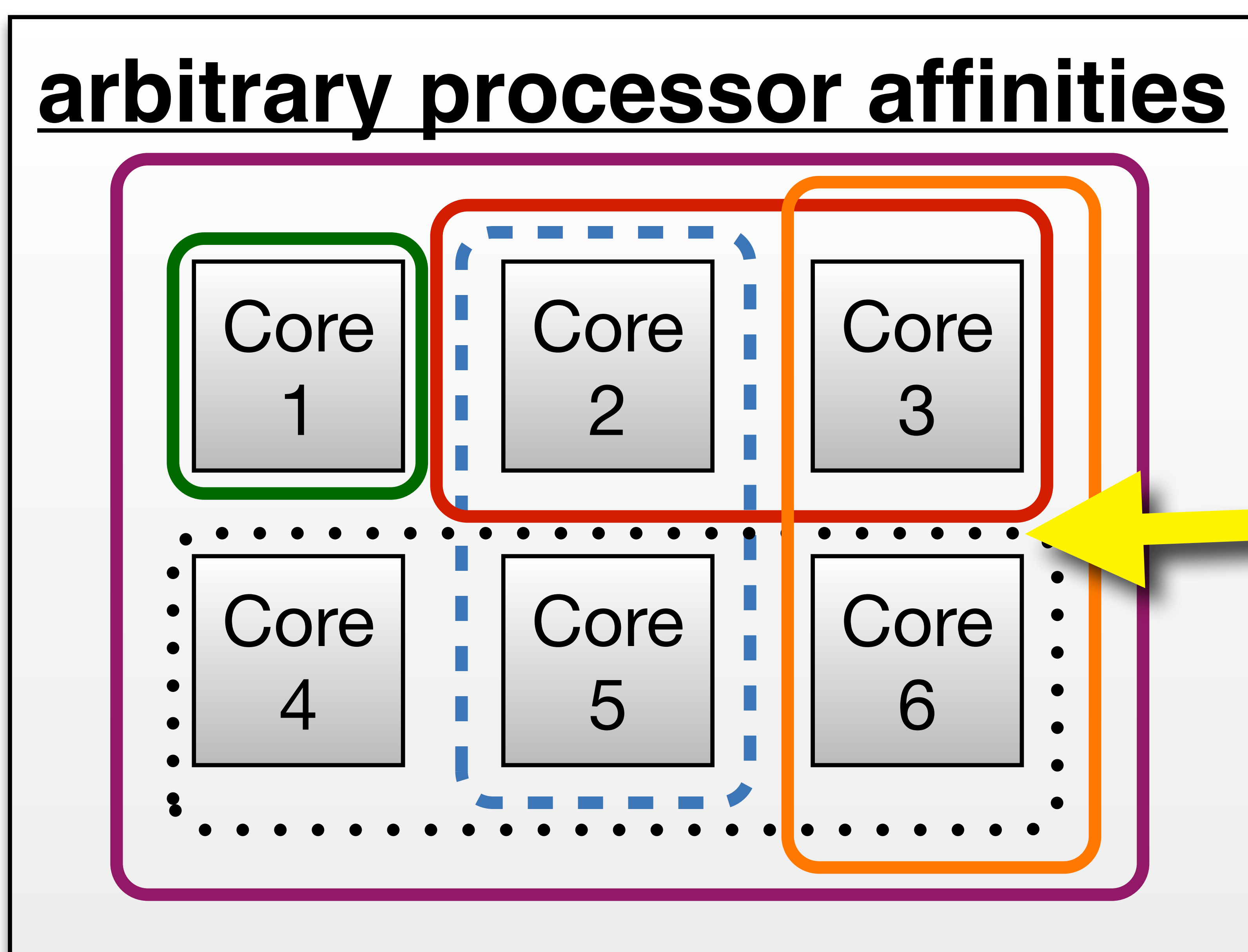
…bility An…

Singleton PAs, **no overlap**, difficult assignment problem.



Uniform PAs, **all processors** included, **symmetric overlap.**





Arbitrary PAs, **irregular** overlap, **transitive interference!**

Hardly any attention in prior work on schedulability analysis!

Prior Work: First Sufficient Analysis

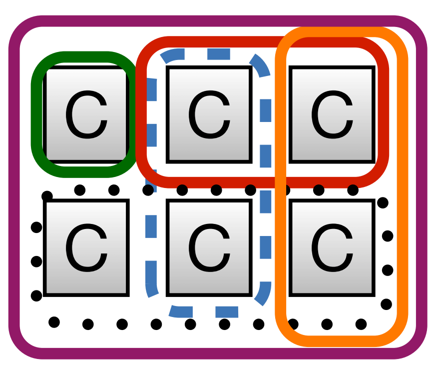
Is a given APA instance ***schedulable***
under ***Linux's global-like "push and pull"*** *scheduler?*

A. Gujarati, F. Cerqueira, and B. Brandenburg, *Schedulability Analysis of the Linux Push and Pull Scheduler with Arbitrary Processor Affinities*, ECRTS 2013.

Prior Work: First Sufficient Analysis

*Is a given APA instance **schedulable***
*under **Linux's global-like "push and pull"** scheduler?*

- $\tau_1, \ldots, \tau_n$
- $\alpha_1, \ldots, \alpha_n$

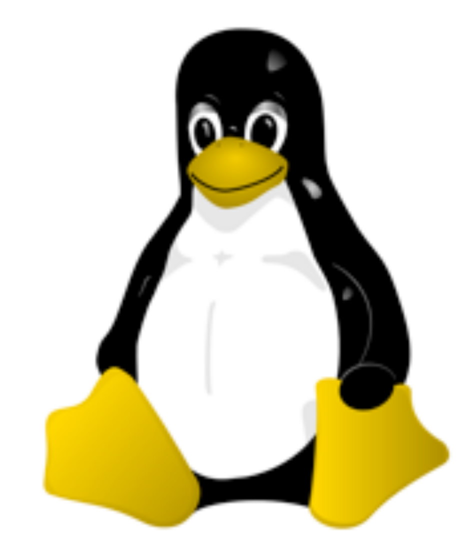


APA instance

Task set with designer-specified **PAs**.

Linux APA scheduling

OS scheduler enforces **PAs**



A. Gujarati, F. Cerqueira, and B. Brandenburg, *Schedulability Analysis of the Linux Push and Pull Scheduler with Arbitrary Processor Affinities*, ECRTS 2013.

Prior nalysis

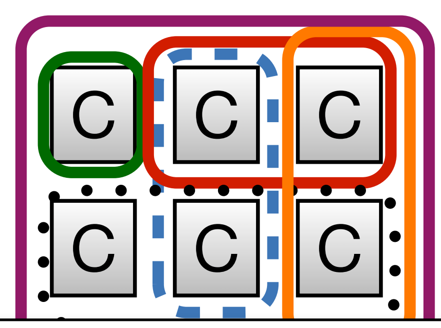
Figuring out appropriate PAs is tricky…

…system designer may include
“**more processors than necessary.**”

under heduler?

➡ $\tau_1,\ldots,\tau_n$

➡ $\alpha_1,\ldots,\alpha_n$



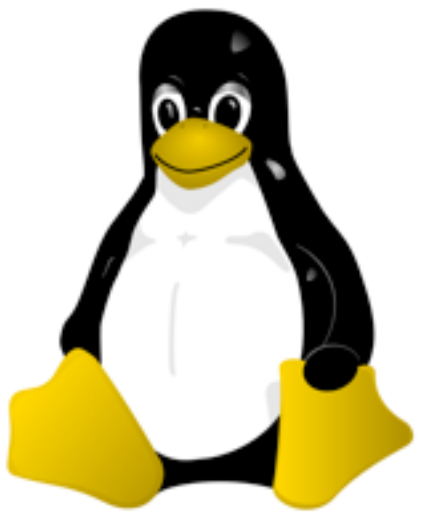
APA instance

Task set with designer-specified **PAs**.

Scheduled as given by user
on Linux.

Linux APA scheduling

OS scheduler enforces **PAs**



A. Gujarati, F. Cerqueira, and B. Brandenburg, *Schedulability Analysis of the Linux Push and Pull Scheduler with Arbitrary Processor Affinities*, ECRTS 2013.

Observation

“Maximal” PAs cause **unnecessary migrations** & **interference**.

➜ **Shrinking** PAs can actually **improve schedulability**.

APA instance

Task set with designer-specified **PAs**.

↓

Linux APA scheduling

OS scheduler enforces **PAs**

A. Gujarati, F. Cerqueira, and B. Brandenburg, *Schedulability Analysis of the Linux Push and Pull Scheduler with Arbitrary Processor Affinities*, ECRTS 2013.

The APA Scheduling Problem

APA instance

Task set with designer-specified **PAs**.

Offline PA optimization

Reduce **PAs** such that task set remains (or **becomes**) schedulable

Linux APA scheduling

OS scheduler enforces **PAs**

The APA Scheduling Problem

Designer gives set of **allowed processors**: **maximum flexibility.**

APA instance

Task set with designer-specified **PAs**.

Shrink PAs to **simplify online problem.**

Offline PA optimization

Reduce **PAs** such that task set remains (or **becomes**) schedulable

Linux APA scheduling

OS scheduler enforces **reduced PAs**

The APA Scheduling Problem

APA instance

Task set with designer-specified **PAs**.

Offline PA optimization

Reduce **PAs** such that task set remains (or **becomes**) schedulable

Linux's notion of APA scheduling is **not the most efficient**.

Online APA scheduling

OS scheduler enforces **reduced PAs**

This Paper: Feasibility Test and PA Reduction

APA instance

Task set with designer-specified **PAs**.

*Constructively determine if given APA instance is **feasible** (= given PAs permit schedule w/o deadline misses.)*

Offline PA optimization

Reduce PAs such that task set remains (or **becomes**) schedulable

***PA reduction**: if feasible, at most **m** tasks migrate.*

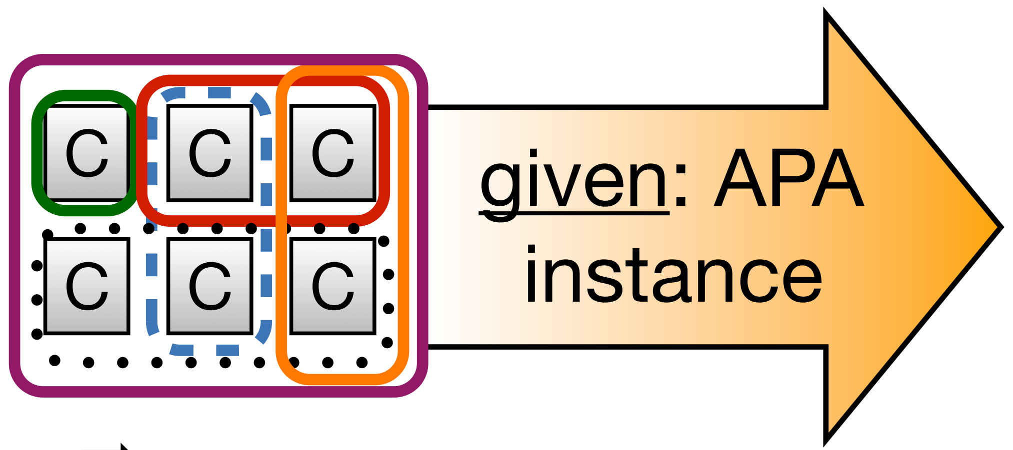
Online APA scheduling

OS scheduler enforces **reduced PAs**

*Construct **scheduling strategy** for instance.*

Part 2
A Feasibility Test for the APA Scheduling Problem

High-Level Overview: Iterative Template Construction



- $\tau_1, \ldots, \tau_n$
- $\alpha_1, \ldots, \alpha_n$

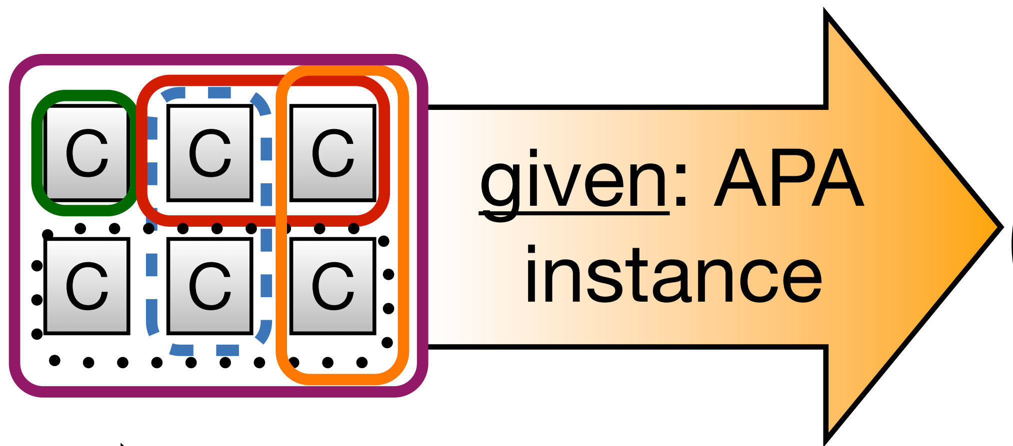
Offline

*Construct a **scheduling template** over the **unit interval [0,1]**.*

Online

scale & apply template

High-Level Overview: Iterative Template Construction



1 *construct & solve linear program*

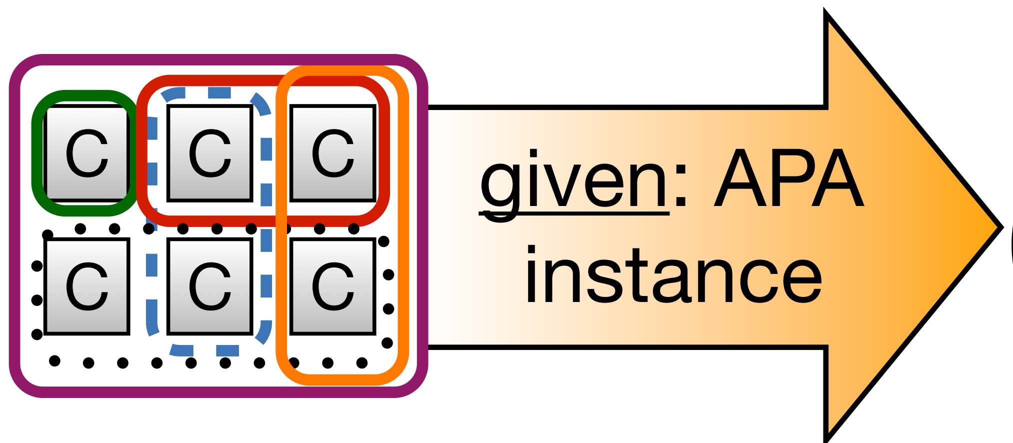
- ➡ $\tau_1,\ldots,\tau_n$
- ➡ $\alpha_1,\ldots,\alpha_n$

Feasibility Test

*If the linear program has **no solution**, then the **APA instance** is infeasible.*

The tricky part is to show existence of schedule if a solution exists…

High-Level Overview: Iterative Template Construction

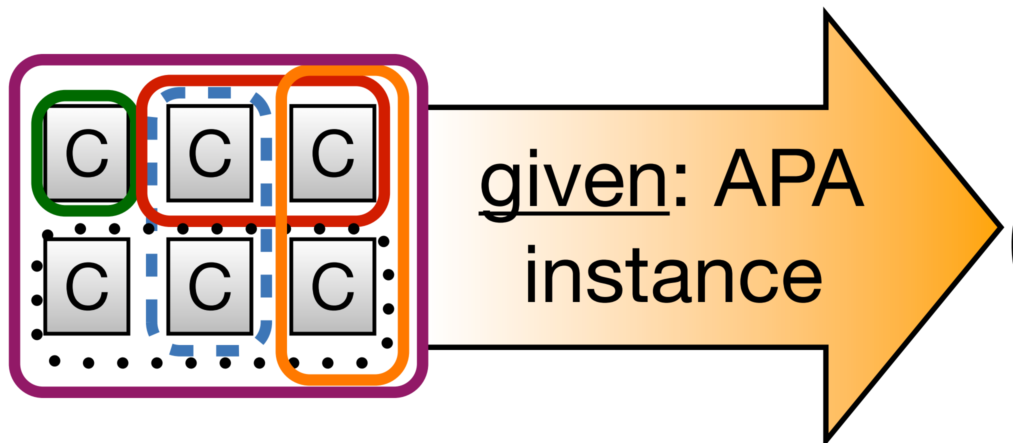


- ➡ $\tau_1,\ldots,\tau_n$
- ➡ $\alpha_1,\ldots,\alpha_n$

1. *construct & solve linear program*
2. *obtain **vertex solution***

(Needed to bound #migrations…)

High-Level Overview: Iterative Template Construction



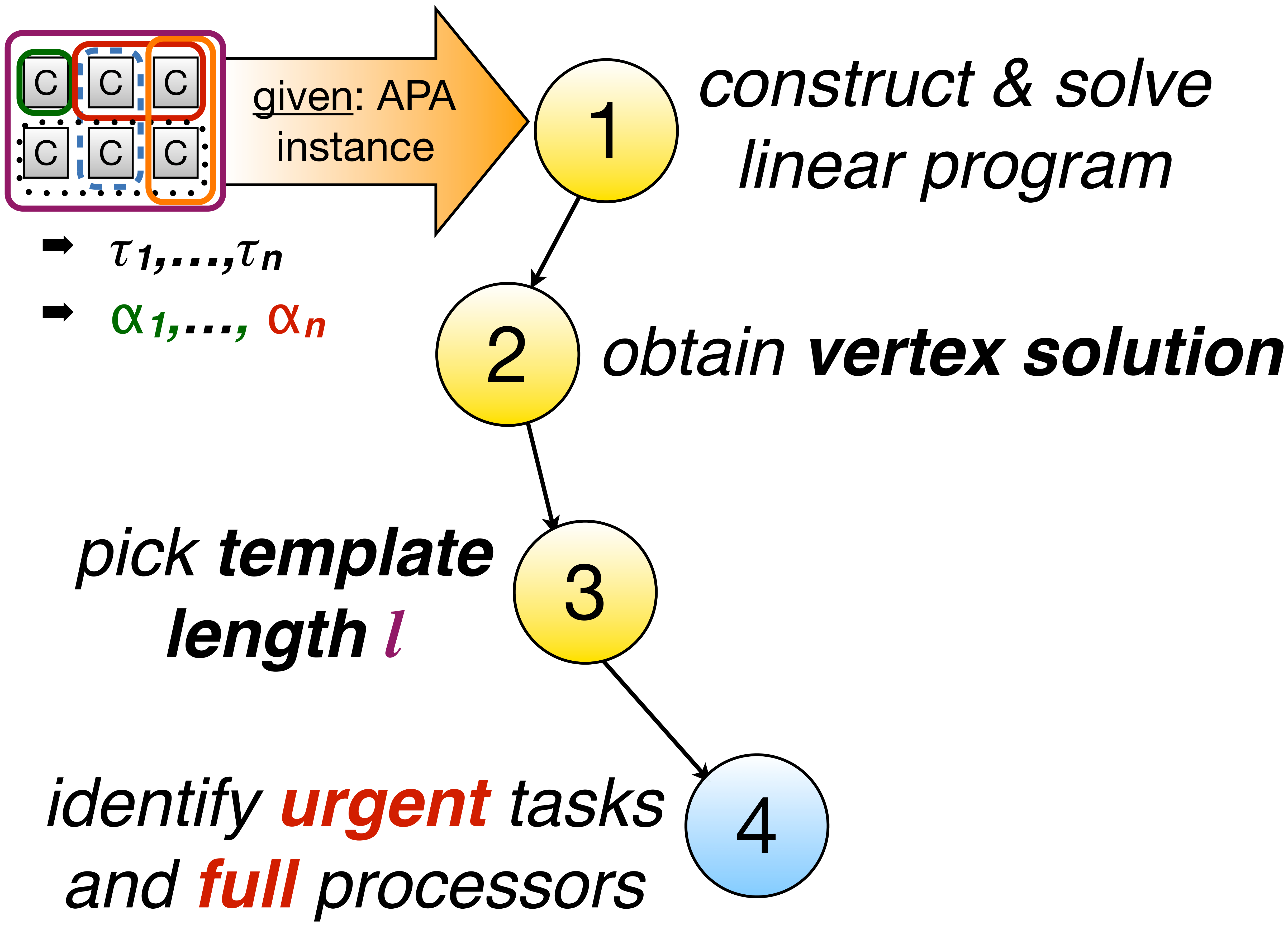
- ➡ $\tau_1,\ldots,\tau_n$
- ➡ $\alpha_1,\ldots,\alpha_n$

1 *construct & solve linear program*

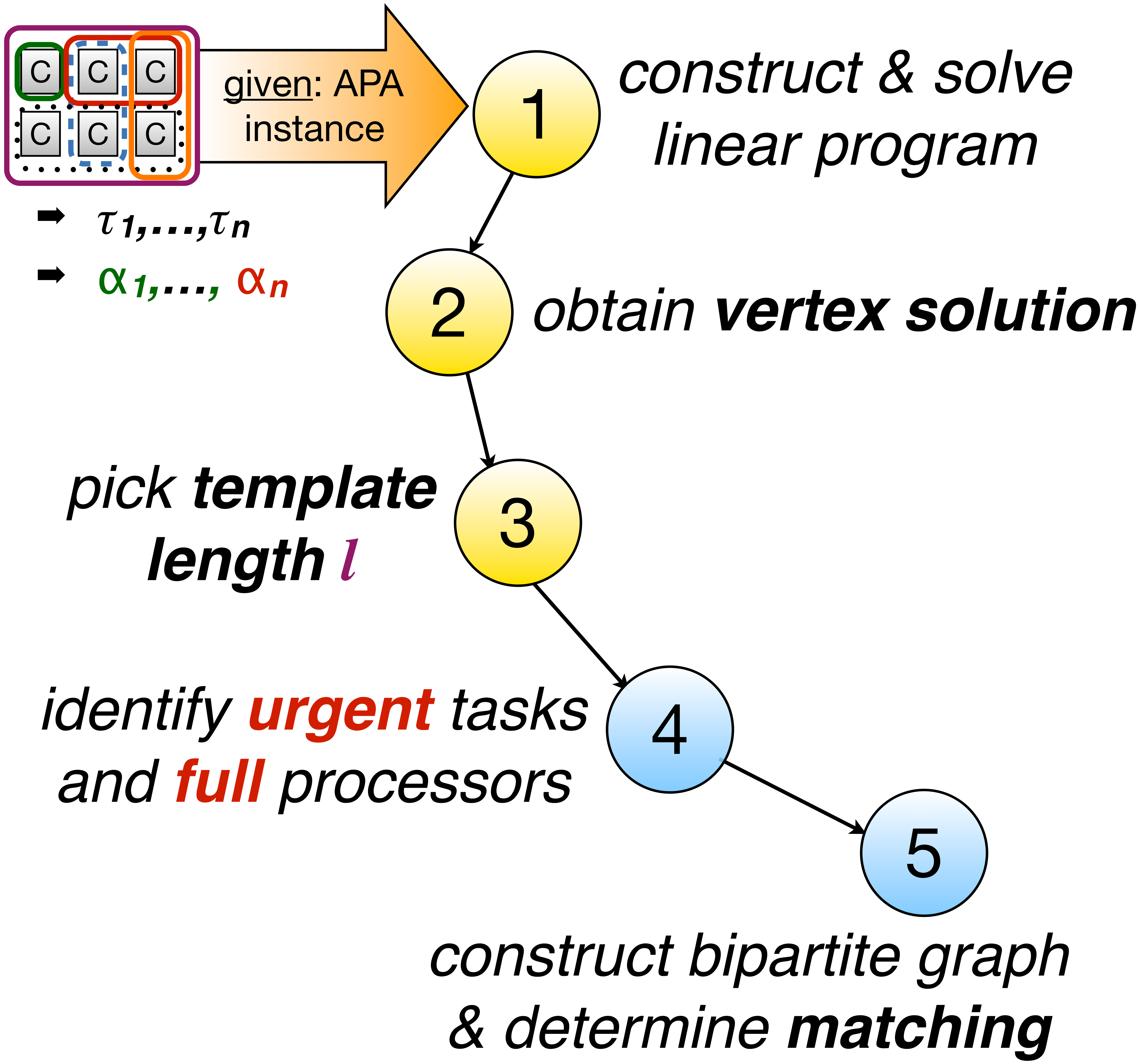
2 *obtain* ***vertex solution***

3 *pick* ***template length*** l

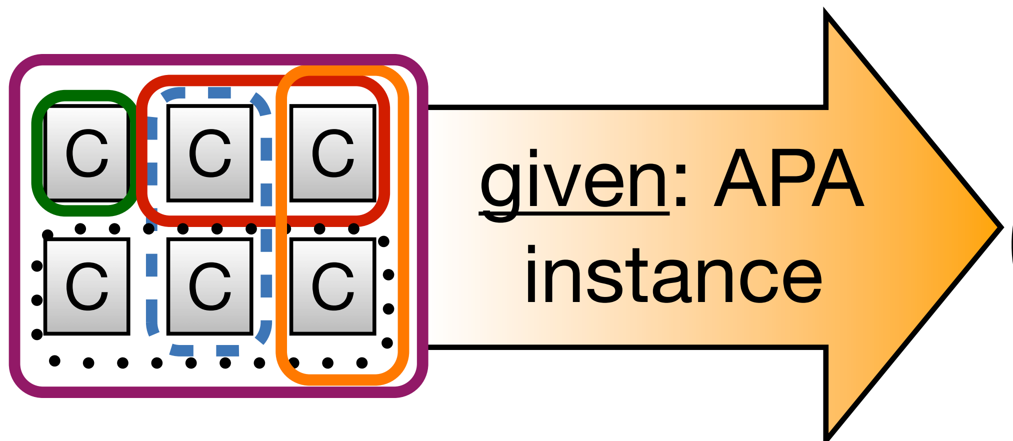
High-Level Overview: Iterative Template Construction



High-Level Overview: Iterative Template Construction



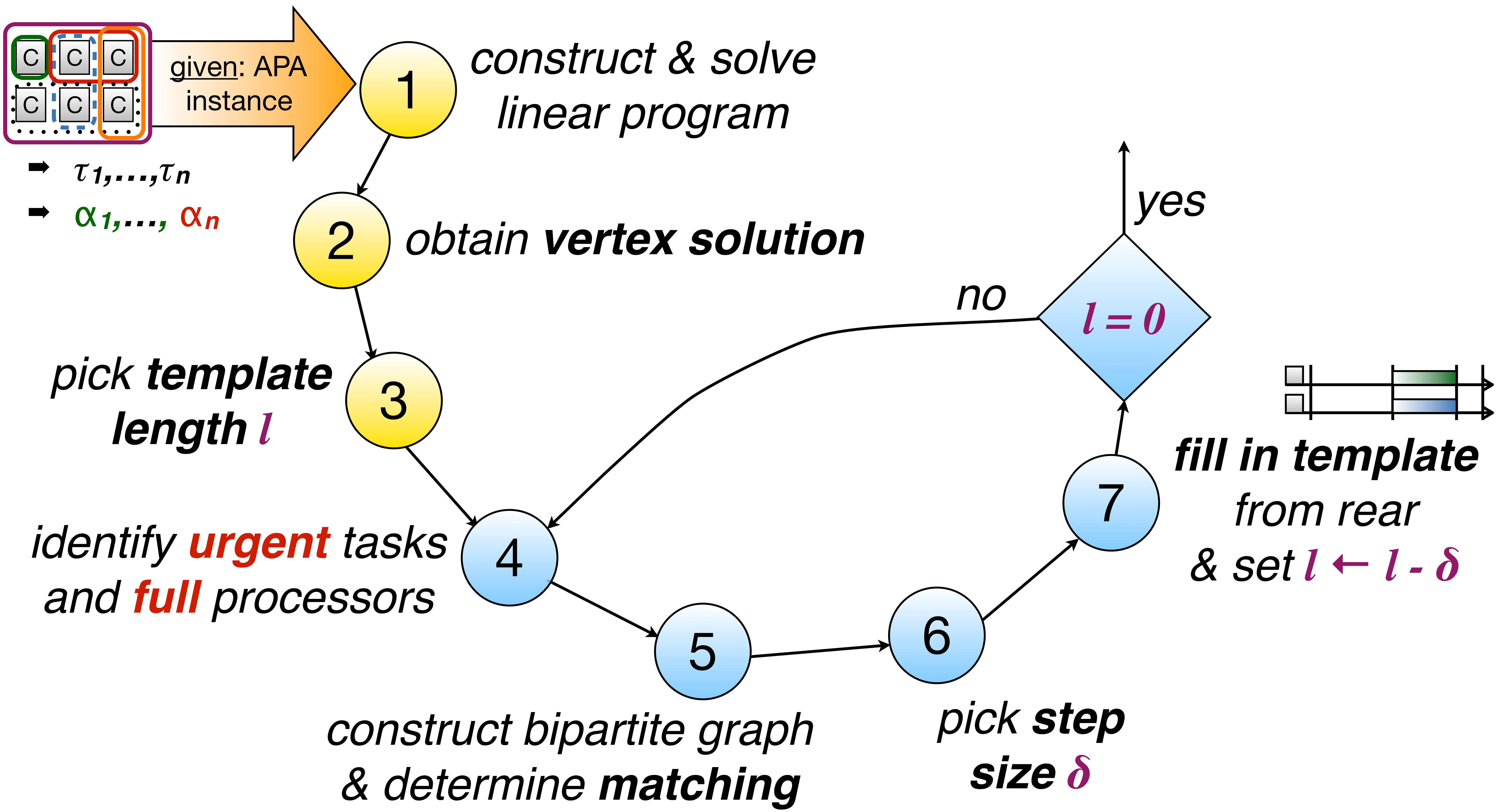
High-Level Overview: Iterative Template Construction



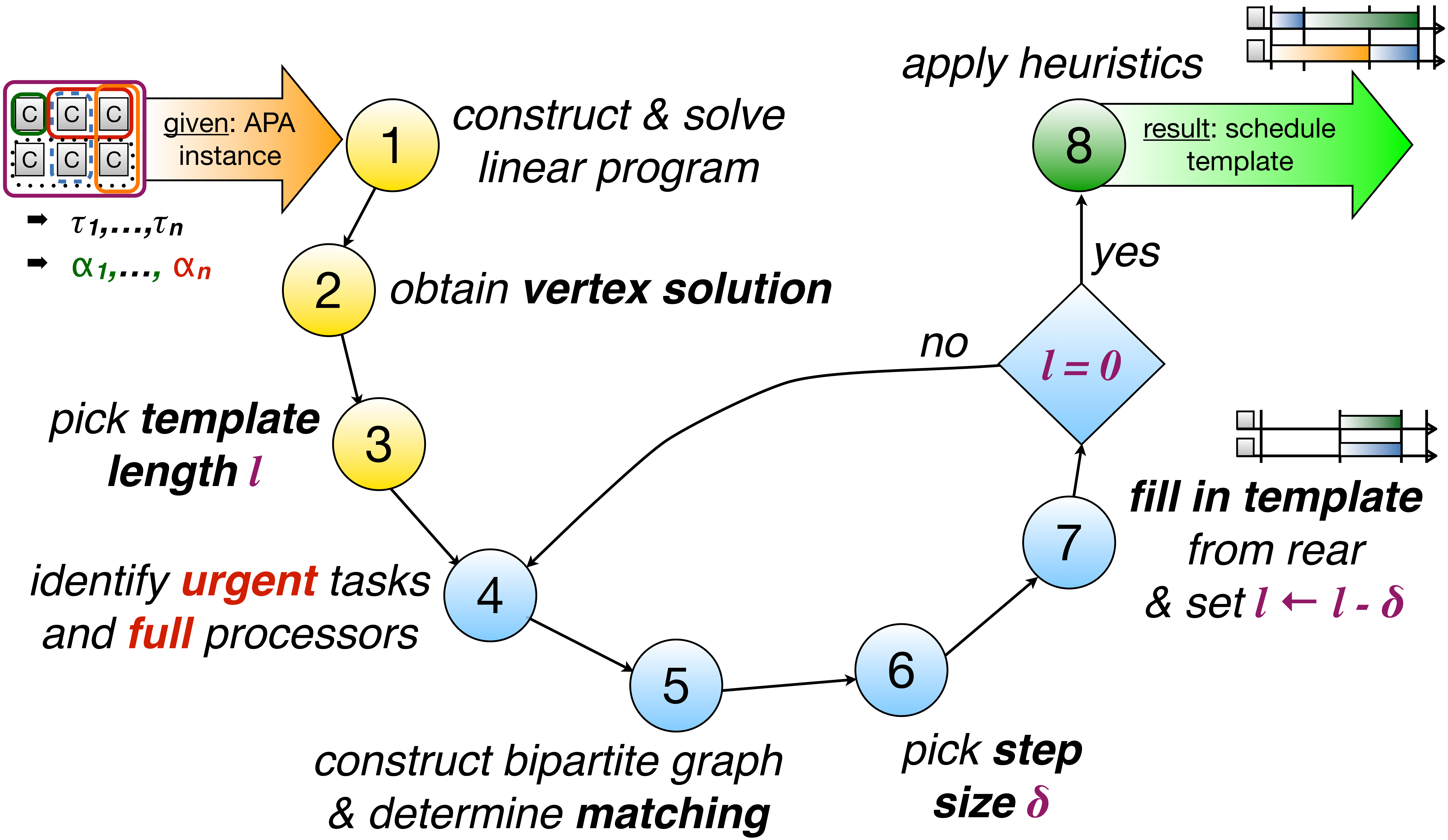
- ➡ $\tau_1,\ldots,\tau_n$
- ➡ $\alpha_1,\ldots,\alpha_n$

1. *construct & solve linear program*
2. *obtain **vertex solution***
3. *pick **template length** l*
4. *identify **urgent** tasks and **full** processors*
5. *construct bipartite graph & determine **matching***
6. *pick **step size** δ*

High-Level Overview: Iterative Template Construction



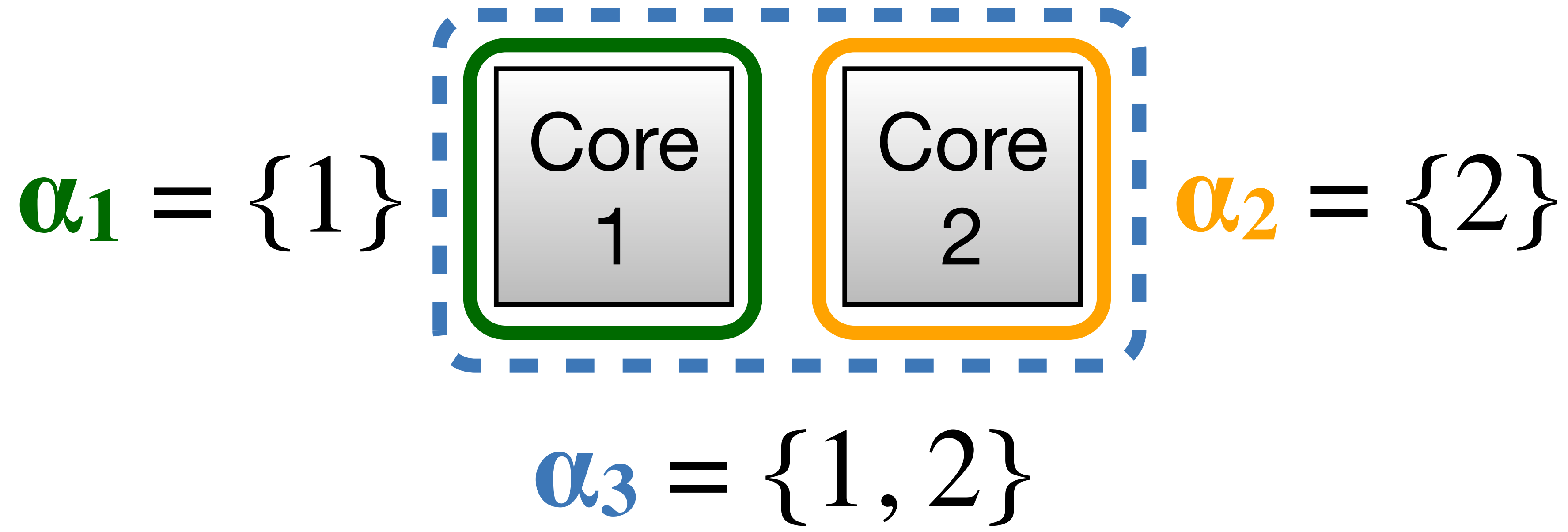
High-Level Overview: Iterative Template Construction



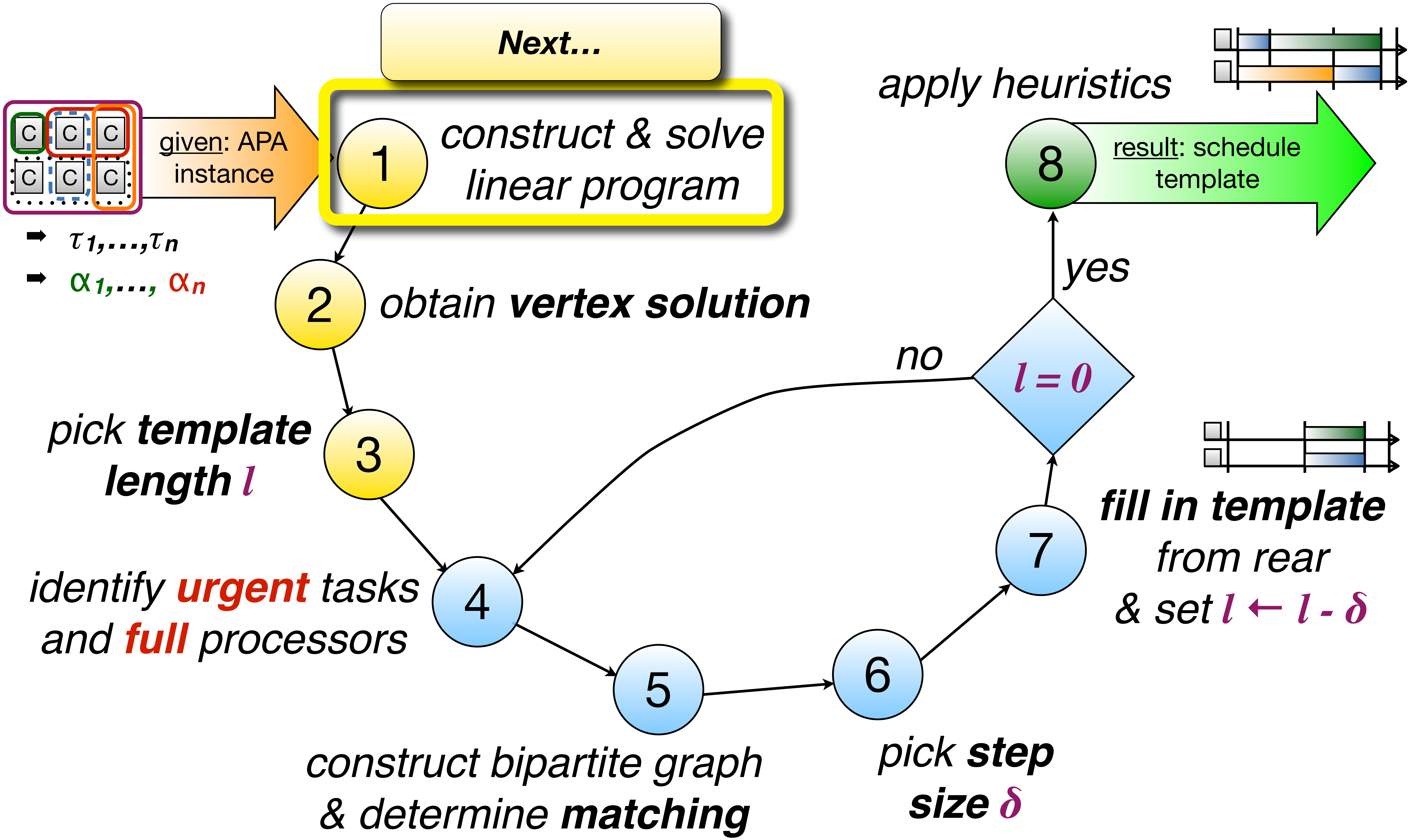
Example Task Set

n = 3 **m = 2**

τ_i	$\mathbf{C}_i$	$\mathbf{T}_i$	α_i	$\mathbf{u}_i$
τ_1	7	10	$\{1\}$	0.7
τ_2	6	10	$\{2\}$	0.6
τ_3	10	20	$\{1, 2\}$	0.5



High-Level Overview: Iterative Template Construction



Step 1: Linear Program

Construct linear program

➡ $n \times m$ variables and $n + m$ constraints

Variable x_{ij}

Fraction of utilization of **task τ_i** served by **processor j**.

τ_i	$\mathbf{C}_i$	$\mathbf{T}_i$	α_i	$\mathbf{u}_i$
τ_1	7	10	$\{1\}$	0.7
τ_2	6	10	$\{2\}$	0.6
τ_3	10	20	$\{1, 2\}$	0.5

Step 1: Linear Program

Construct linear program

➡ $n \times m$ variables and $n + m$ constraints

Variable x_{ij}

Fraction of utilization of **task τ_i** served by **processor j**.

τ_i	$\mathbf{C}_i$	$\mathbf{T}_i$	α_i	$\mathbf{u}_i$
τ_1	7	10	$\{1\}$	0.7
τ_2	6	10	$\{2\}$	0.6
τ_3	10	20	$\{1,2\}$	0.5

$$\begin{aligned} x_{11} &= 1 \\ x_{22} &= 1 \\ x_{31} + x_{32} &= 1 \end{aligned}$$

Each task is fully served: $$\sum_{j \in \alpha_i} x_{i,j} = 1$$

➡ n constraints

Step 1: Linear Program

Construct linear program

➡ $n \times m$ variables and $n + m$ constraints

Variable x_{ij}

Fraction of utilization of **task τ_i** served by **processor j**.

τ_i	$\mathbf{C}_i$	$\mathbf{T}_i$	α_i	$\mathbf{u}_i$
τ_1	7	10	$\{1\}$	0.7
τ_2	6	10	$\{2\}$	0.6
τ_3	10	20	$\{1,2\}$	0.5

Each task is fully served: $\sum_{j \in \alpha_i} x_{i,j} = 1$

➡ n constraints

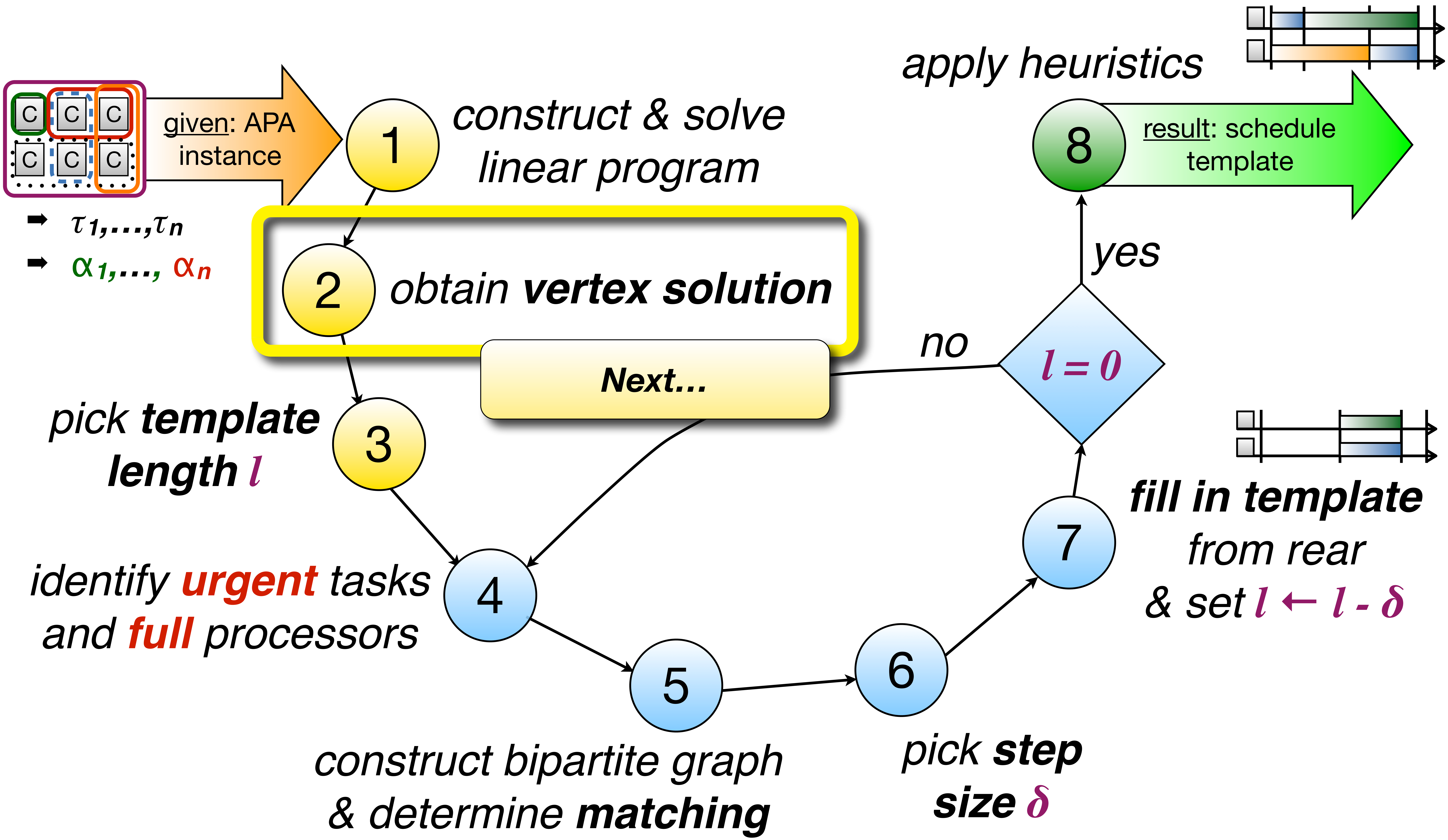
$$\begin{aligned} x_{11} &= 1 \\ x_{22} &= 1 \\ x_{31} + x_{32} &= 1 \end{aligned}$$

No processor is over-utilized: $\sum_{i=1}^{n} (u_i x_{i,j}) \leq 1$

➡ m constraints

$$\begin{aligned} 0.7\,x_{11} + 0.5\,x_{31} &\leq 1 \\ 0.6\,x_{22} + 0.5\,x_{32} &\leq 1 \end{aligned}$$

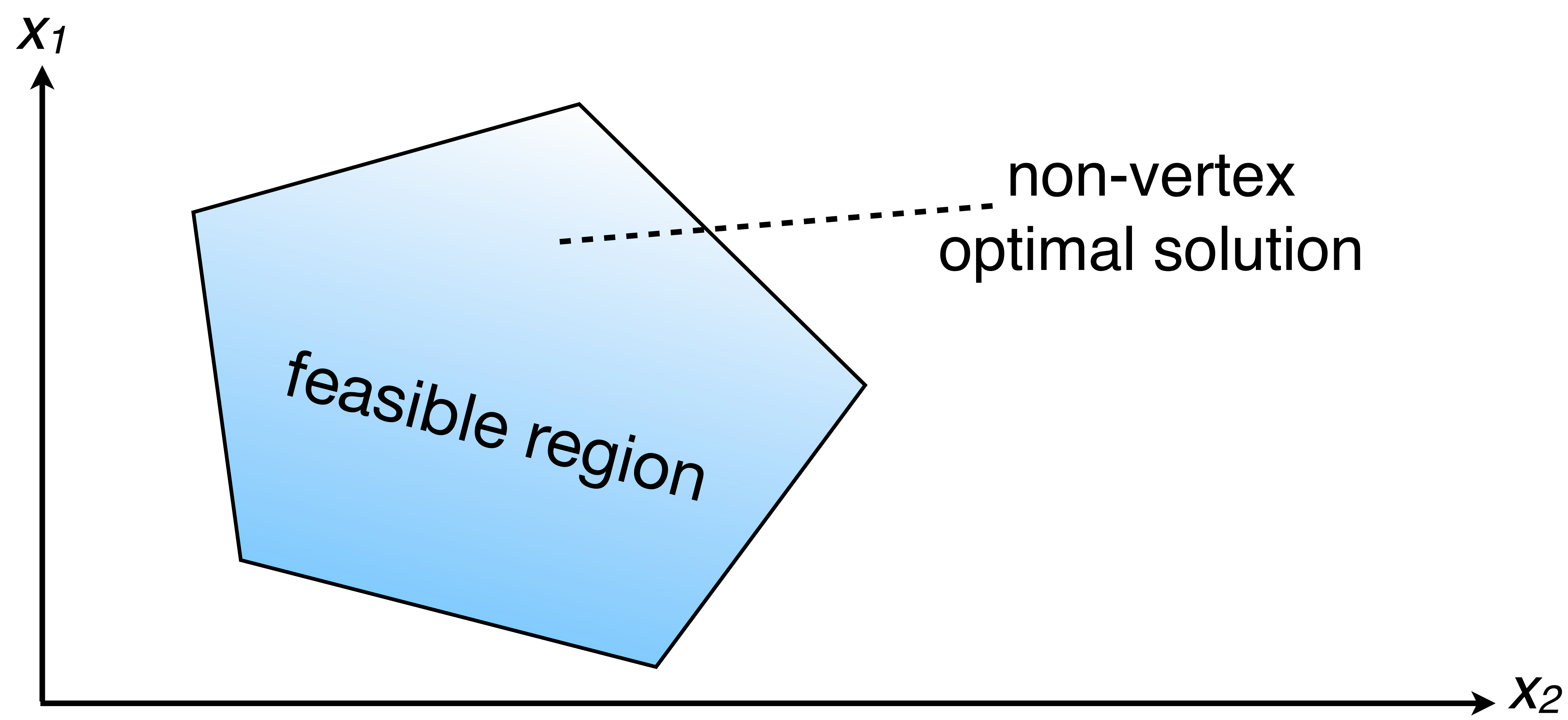
High-Level Overview: Iterative Template Construction



Step 2: Obtain Vertex Solution

Feasible region of linear program

➡ Solution exists within **convex high-dimensional polytope**



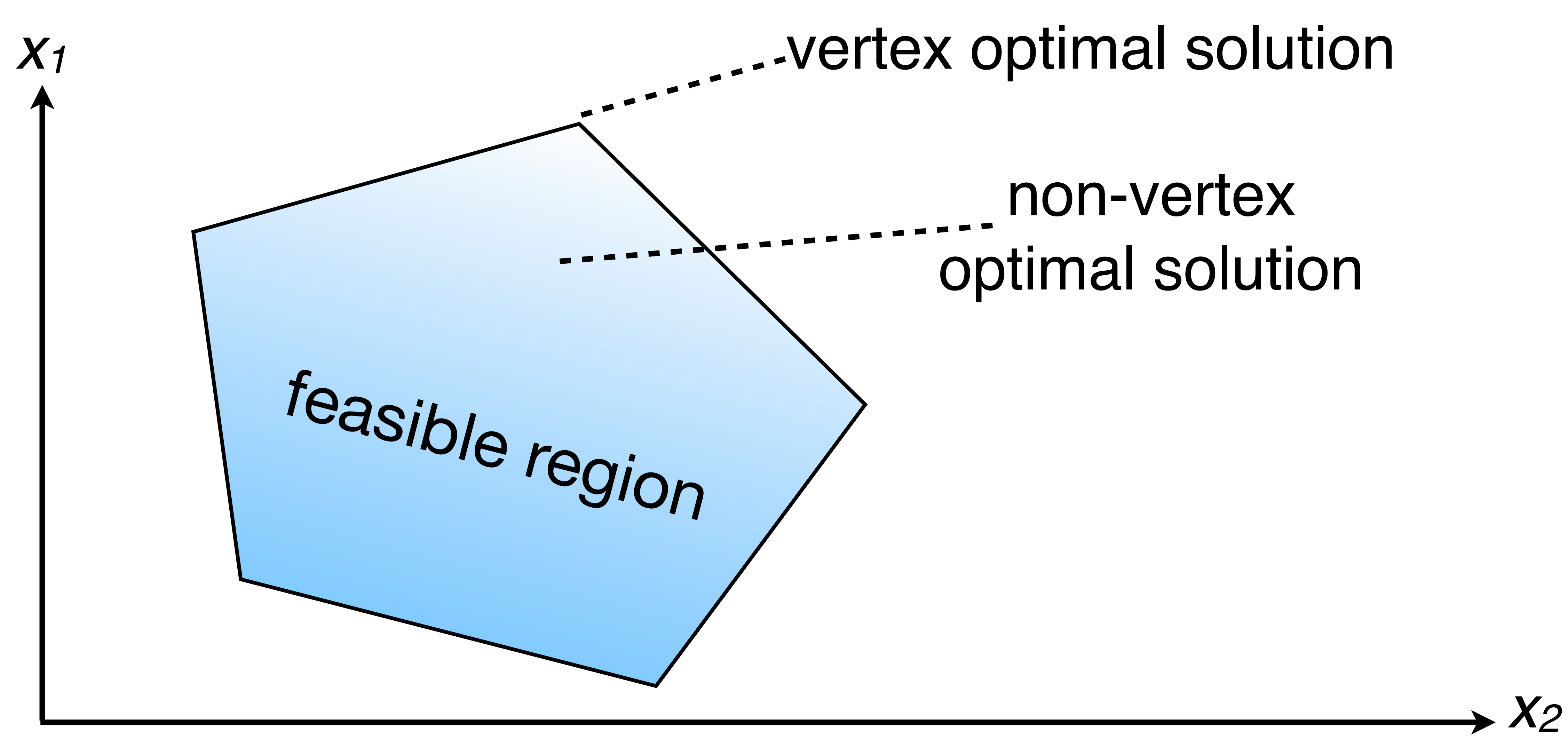
Step 2: Obtain Vertex Solution

Feasible region of linear program

➡ Solution exists within **convex high-dimensional polytope**

Need vertex optimal solution to limit task migrations (discussed later)

➡ Optimal solution given by solver **not necessarily at vertex of polytope** (but **vertex optimal solution** exists)



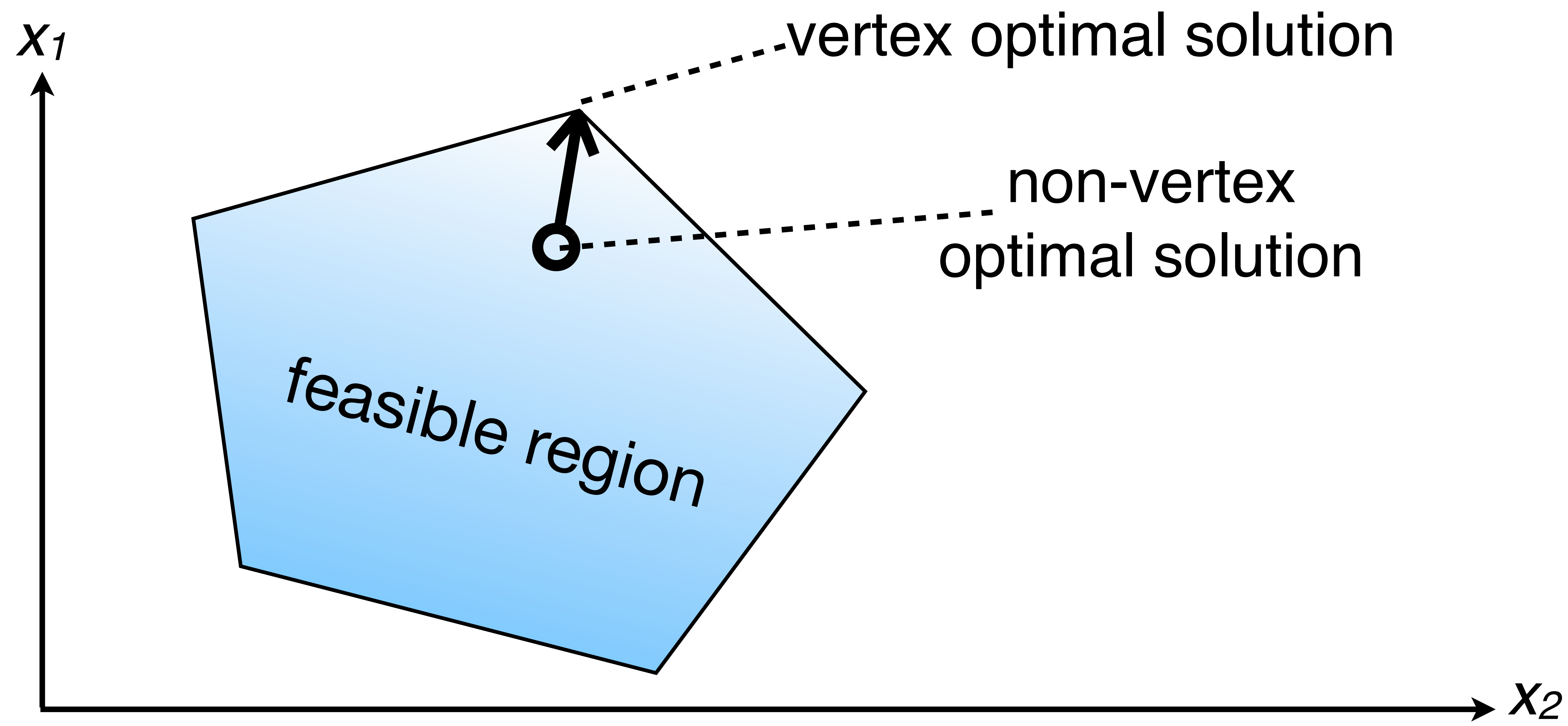
Step 2: Obtain Vertex Solution

Feasible region of linear program

➡ Solution exists within **convex high-dimensional polytope**

Need vertex optimal solution to limit task migrations (discussed later)

➡ Optimal solution given by solver **not necessarily at vertex of polytope** (but **vertex optimal solution** exists)



Obtain optimal vertex solution from optimal solution

➡ Can be done in polynomial time (standard OR techniques)

Step 2: Example Allocation

τ_i	$\mathbf{C}_i$	$\mathbf{T}_i$	α_i	$\mathbf{u}_i$
τ_1	7	10	$\{1\}$	0.7
τ_2	6	10	$\{2\}$	0.6
τ_3	10	20	$\{1, 2\}$	0.5

$$\begin{aligned} x_{11} &= 1 \\ x_{22} &= 1 \\ x_{31} + x_{32} &= 1 \end{aligned}$$

$$\begin{aligned} 0.7\, x_{11} + 0.5\, x_{31} &\leq 1 \\ 0.6\, x_{22} + 0.5\, x_{32} &\leq 1 \end{aligned}$$

Utilization	τ_1	τ_2	τ_3	Total
Core 1	0.7	—	0.2	0.9
Core 2	—	0.6	0.3	0.9

task allocation

Step 2: Example Allocation

τ_i	$\mathbf{C}_i$	$\mathbf{T}_i$	α_i	$\mathbf{u}_i$
τ_1	7	10	$\{1\}$	0.7
τ_2	6	10	$\{2\}$	0.6
τ_3	10	20	$\{1, 2\}$	0.5

Migratory task:
utilization split across cores.

Utilization	τ_1	τ_2	τ_3	Total
Core 1	0.7	—	0.2	0.9
Core 2	—	0.6	0.3	0.9

task allocation

Step 2: Example Allocation

τ_i	$\mathbf{C}_i$	$\mathbf{T}_i$	α_i	$\mathbf{u}_i$
τ_1	7	10	$\{1\}$	0.7
τ_2	6	10	$\{2\}$	0.6
τ_3	10	20	$\{1, 2\}$	0.5

$$\begin{aligned} x_{11} &= 1 \\ x_{22} &= 1 \\ x_{31} + x_{32} &= 1 \end{aligned}$$

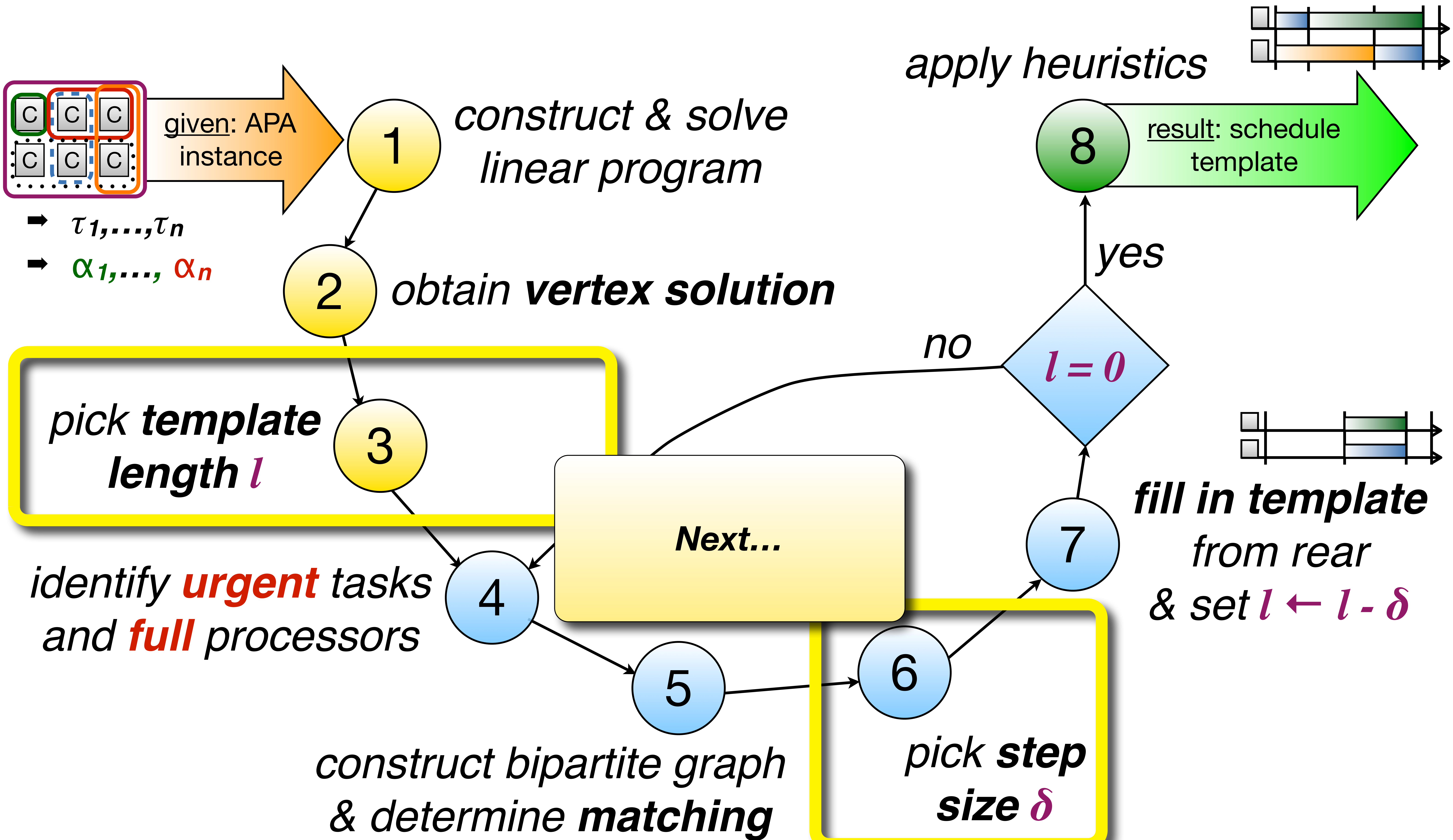
$$\begin{aligned} 0.7\,x_{11} + 0.5\,x_{31} &\leq 1 \\ 0.6\,x_{22} + 0.5\,x_{32} &\leq 1 \end{aligned}$$

total utilization on each core: 0.9

Utilization	τ_1	τ_2	τ_3	Total
Core 1	0.7	—	0.2	0.9
Core 2	—	0.6	0.3	0.9

task allocation

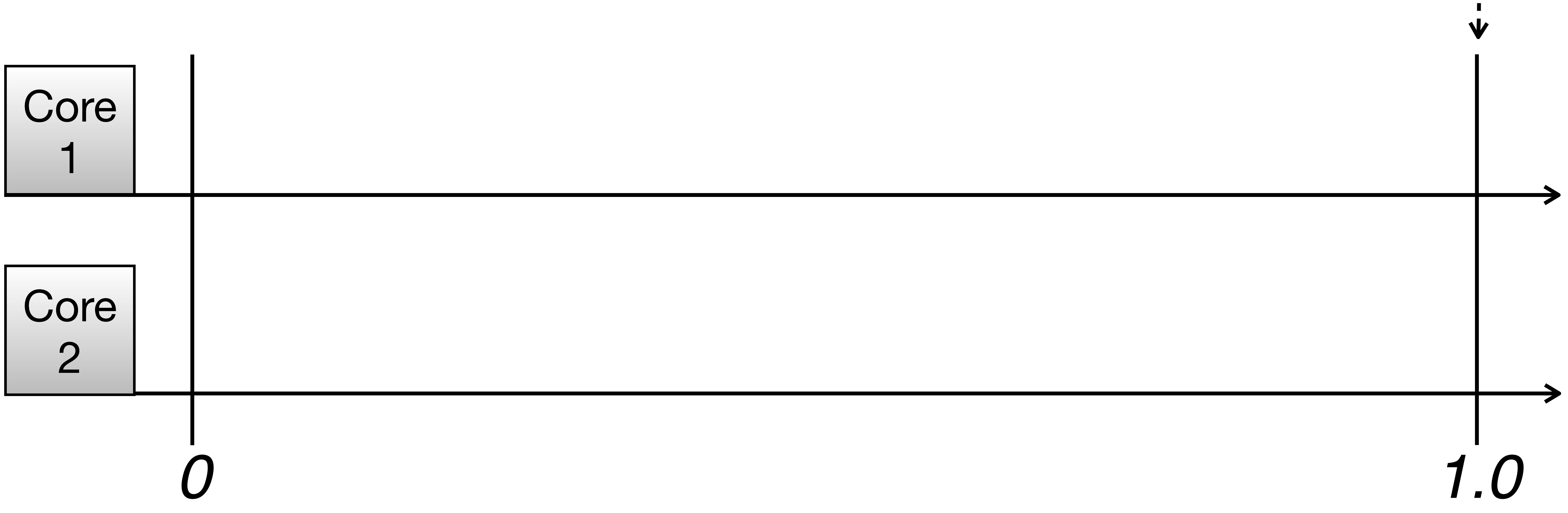
High-Level Overview: Iterative Template Construction



Steps 3 & 6: Template Length & Step Size

Iterative algorithm

- ➡ construct **schedule template**
- ➡ normalized for **interval [0, 1]**
- ➡ fill in template **from rear**



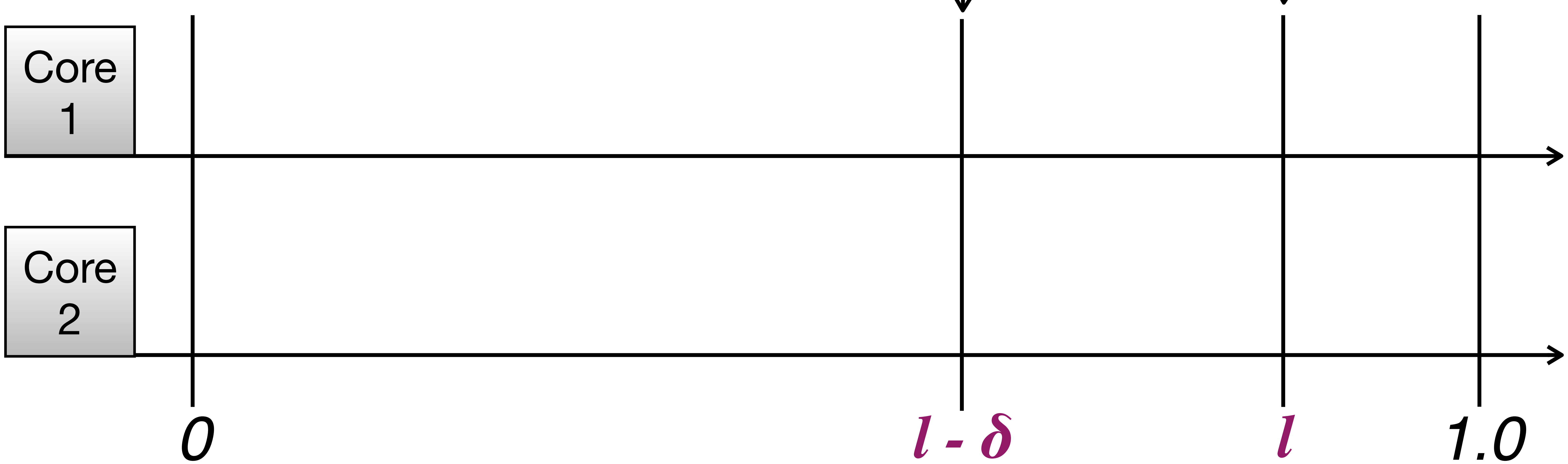
Steps 3 & 6: Template Length & Step Size

Iterative algorithm

- ➡ construct **schedule template**
- ➡ normalized for **interval [0, 1]**
- ➡ fill in template **from rear**

Key parameters

- ➡ template length: $0 < l \leq 1$
- ➡ iteration step size: $0 < \delta \leq l$



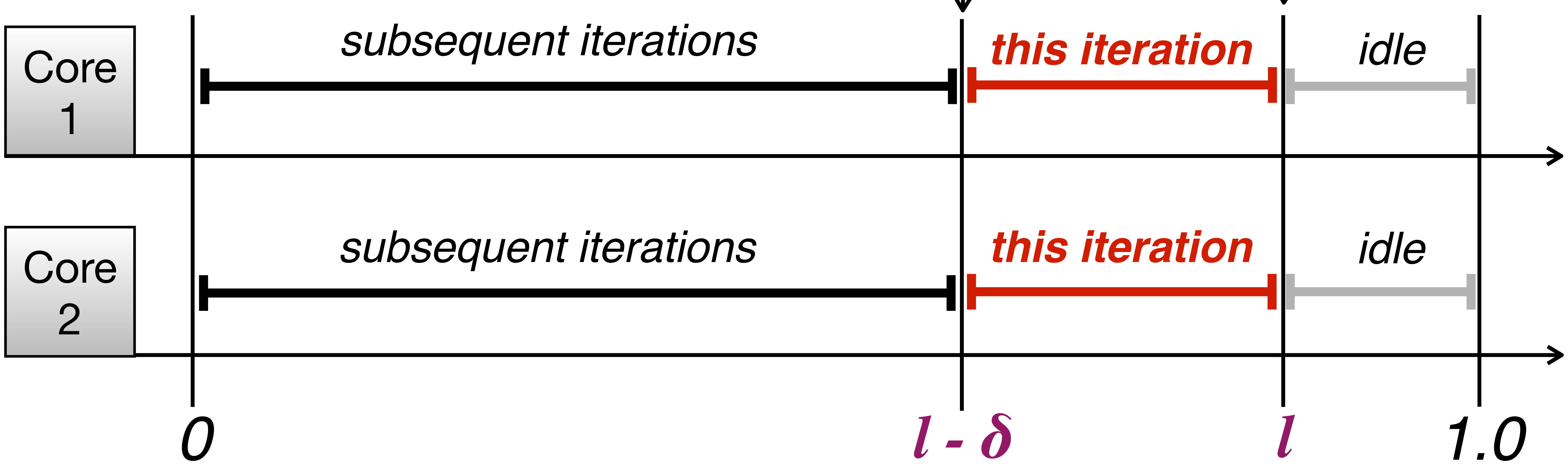
Steps 3 & 6: Template Length & Step Size

Iterative algorithm

- ➡ construct **schedule template**
- ➡ normalized for **interval [0, 1]**
- ➡ fill in template **from rear**

Key parameters

- ➡ template length: $0 < l \leq 1$
- ➡ iteration step size: $0 < \delta \leq l$



Steps 3 & 6: Template Length & Step Size

τ_i	$\mathbf{C}_i$	$\mathbf{T}_i$	α_i	$\mathbf{u}_i$
τ_1	7	10	$\{1\}$	0.7
τ_2	6	10	$\{2\}$	0.6
τ_3	10	20	$\{1, 2\}$	0.5

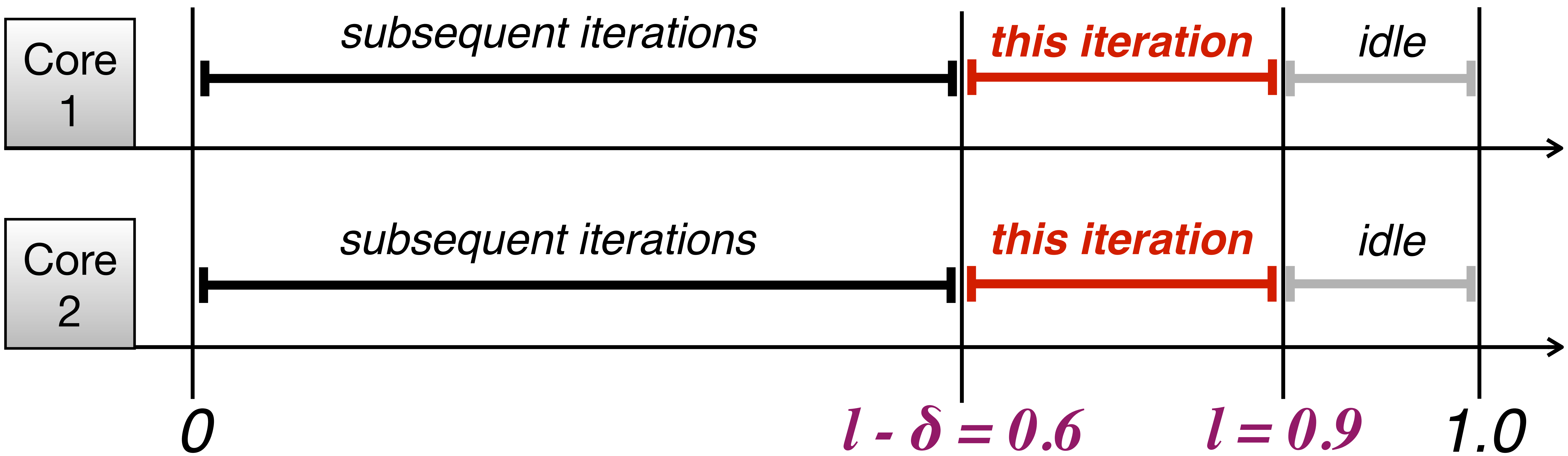
step size

$\delta = 0.3$

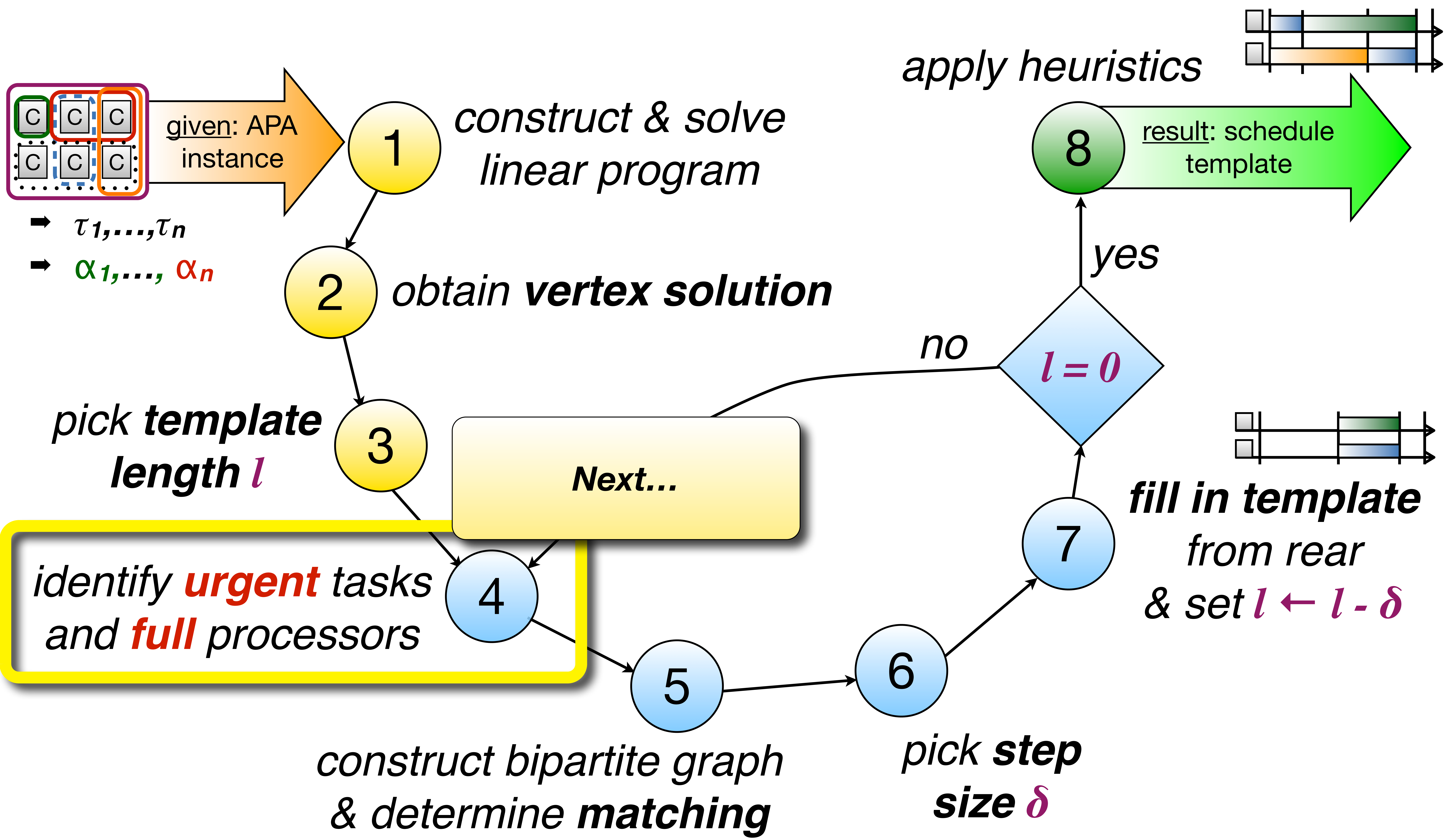
template length

$l = 0.9$

→ *assign tasks for interval (0.6, 0.9]*



High-Level Overview: Iterative Template Construction



Step 4: Urgent Tasks and Full Processors

Urgent task

no slack left

Full processor

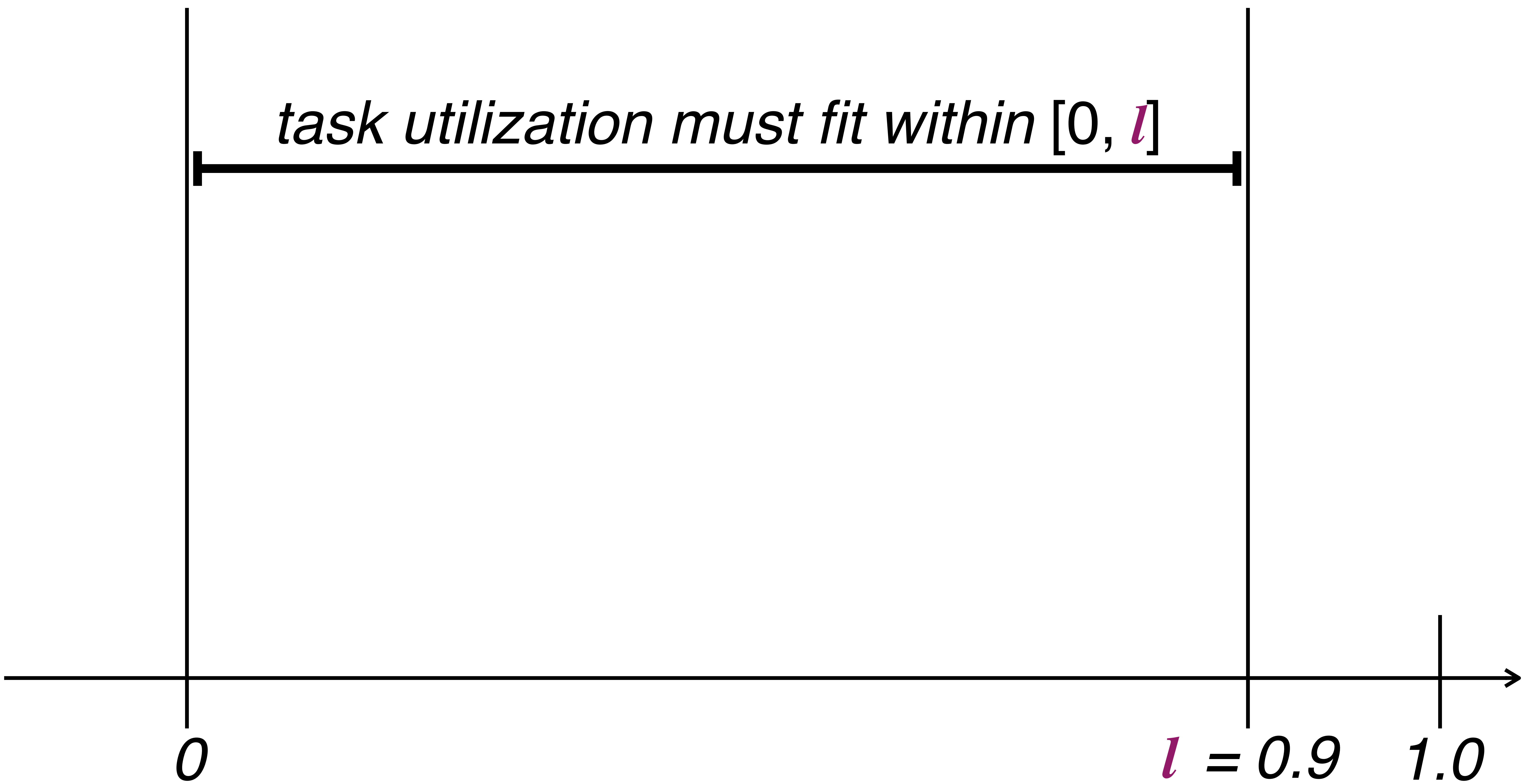
no idle time left

Step 4: Urgent Tasks and Full Processors

Urgent task

no slack left

τ_i	$\mathbf{C}_i$	$\mathbf{T}_i$	α_i	$\mathbf{u}_i$
τ_1	7	10	$\{1\}$	0.7
τ_2	6	10	$\{2\}$	0.6
τ_3	10	20	$\{1, 2\}$	0.5

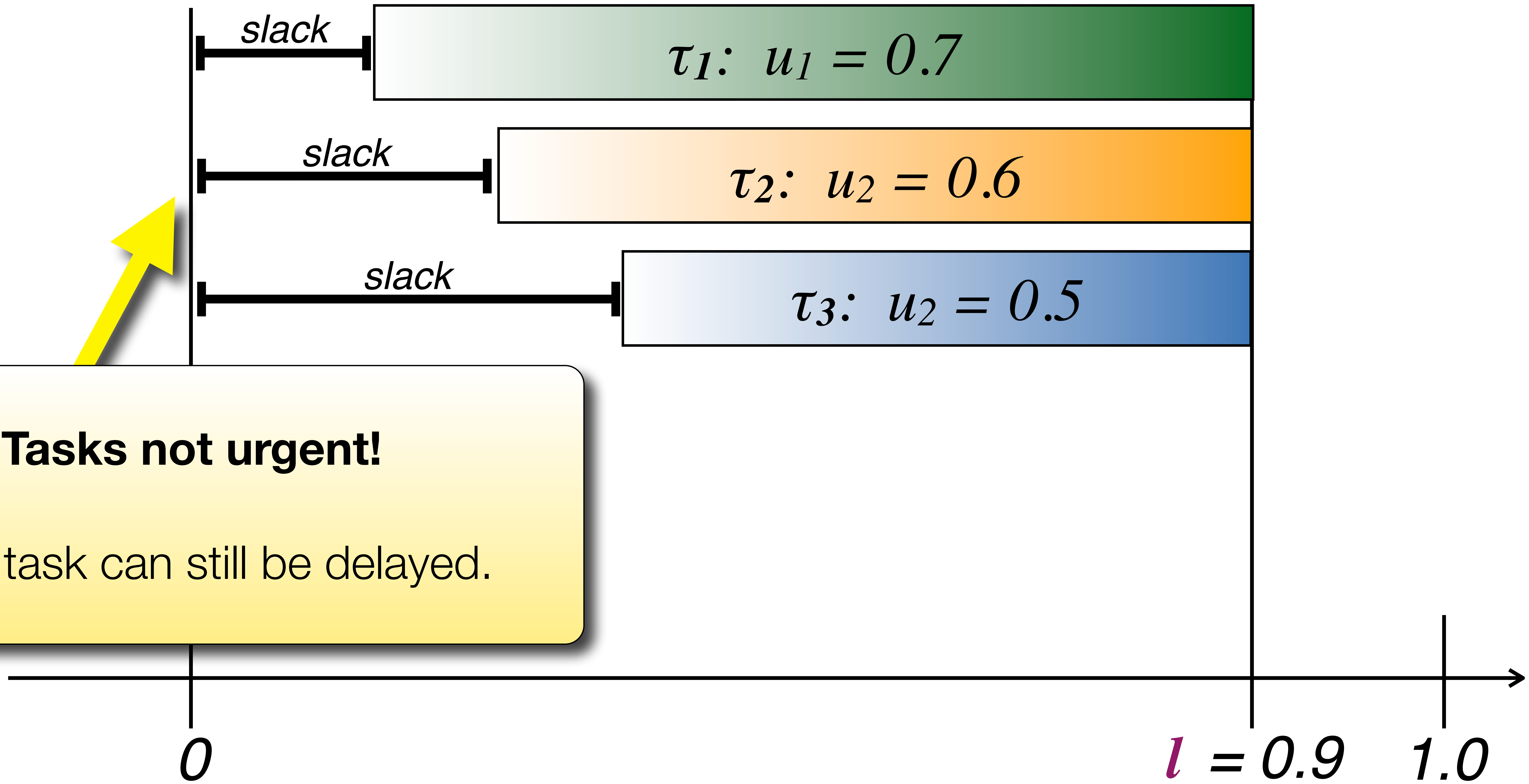


Step 4: Urgent Tasks and Full Processors

Urgent task

no slack left

τ_i	$\mathbf{C}_i$	$\mathbf{T}_i$	α_i	$\mathbf{u}_i$
τ_1	7	10	$\{1\}$	0.7
τ_2	6	10	$\{2\}$	0.6
τ_3	10	20	$\{1, 2\}$	0.5



Tasks not urgent!

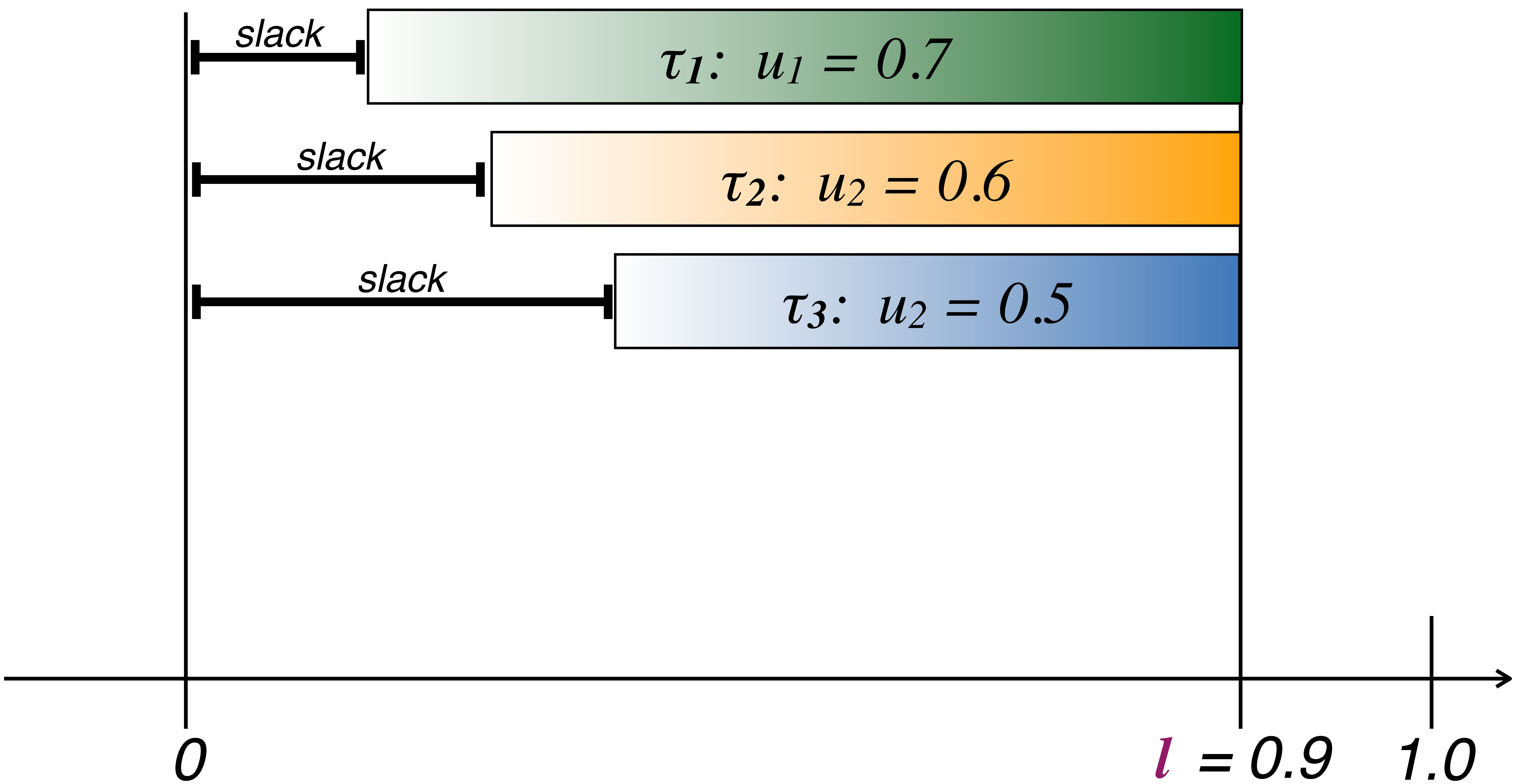
Each task can still be delayed.

Step 4: Urgent Tasks and Full Processors

Utilization	τ_1	τ_2	τ_3	Total
Core 1	0.7	—	0.2	0.9
Core 2	—	0.6	0.3	0.9

Full processor

no idle time left

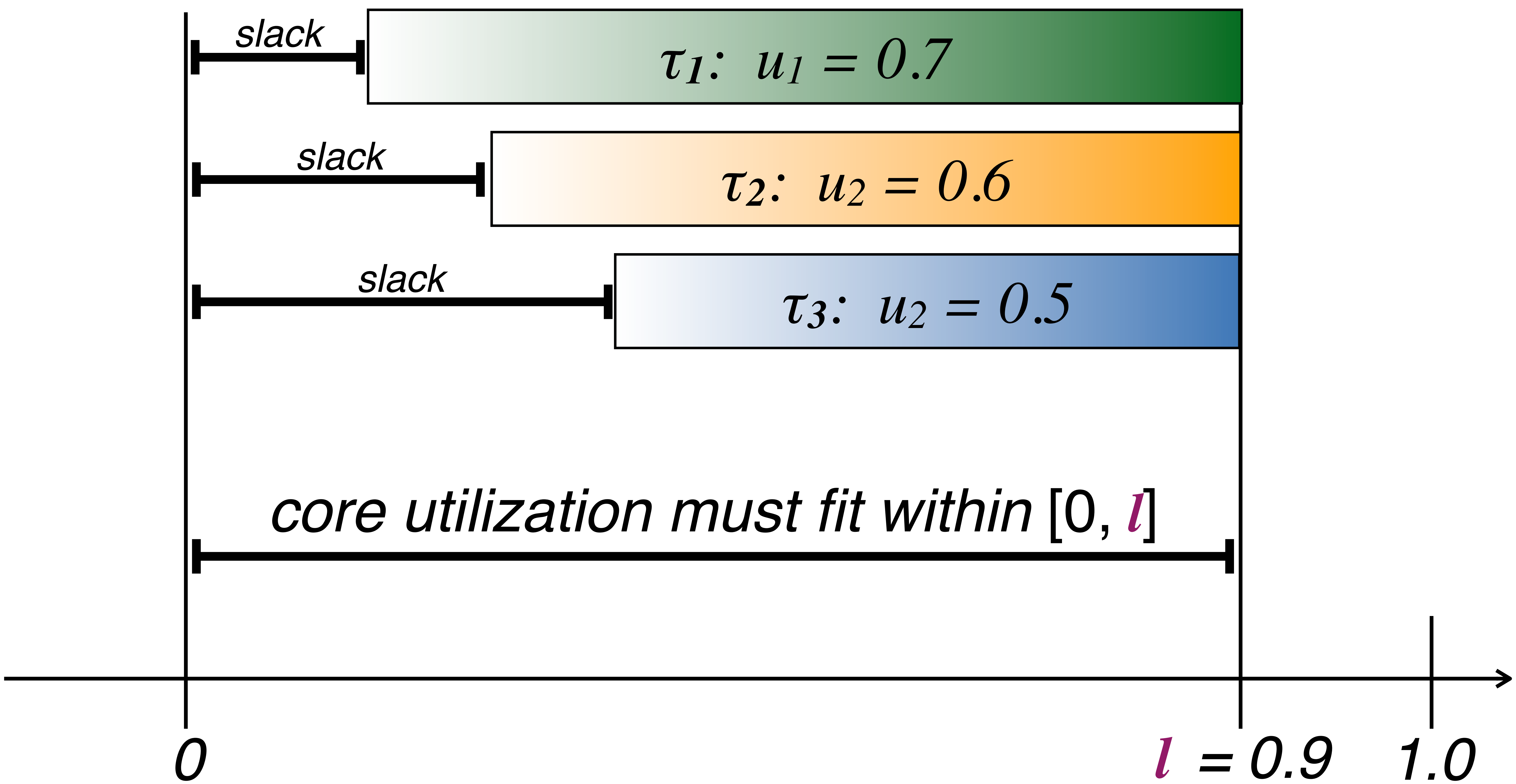


Step 4: Urgent Tasks and Full Processors

Utilization	τ_1	τ_2	τ_3	Total
Core 1	0.7	—	0.2	0.9
Core 2	—	0.6	0.3	0.9

Full processor

no idle time left



Both processors urgent!

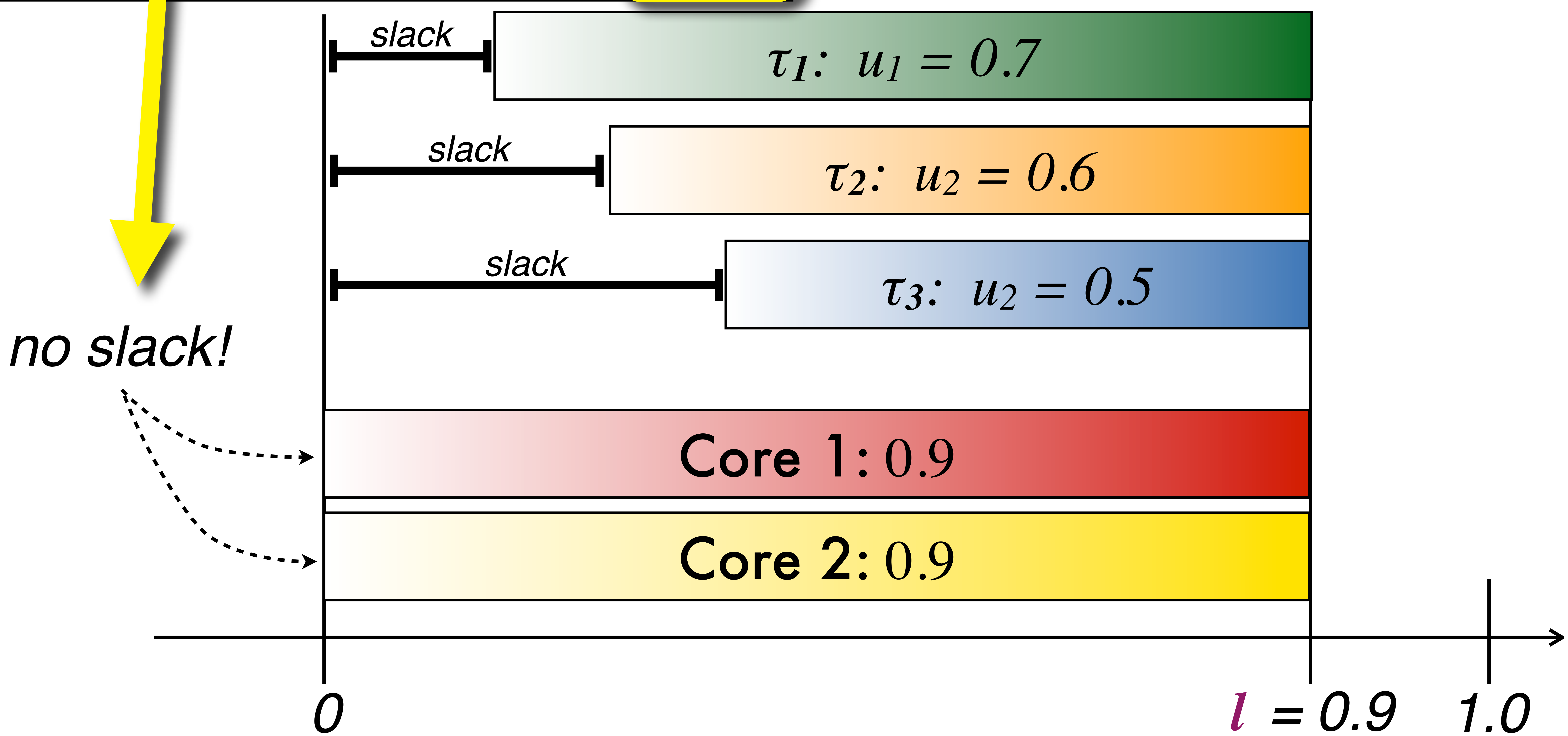
Must schedule *some* task.

d Full Processors

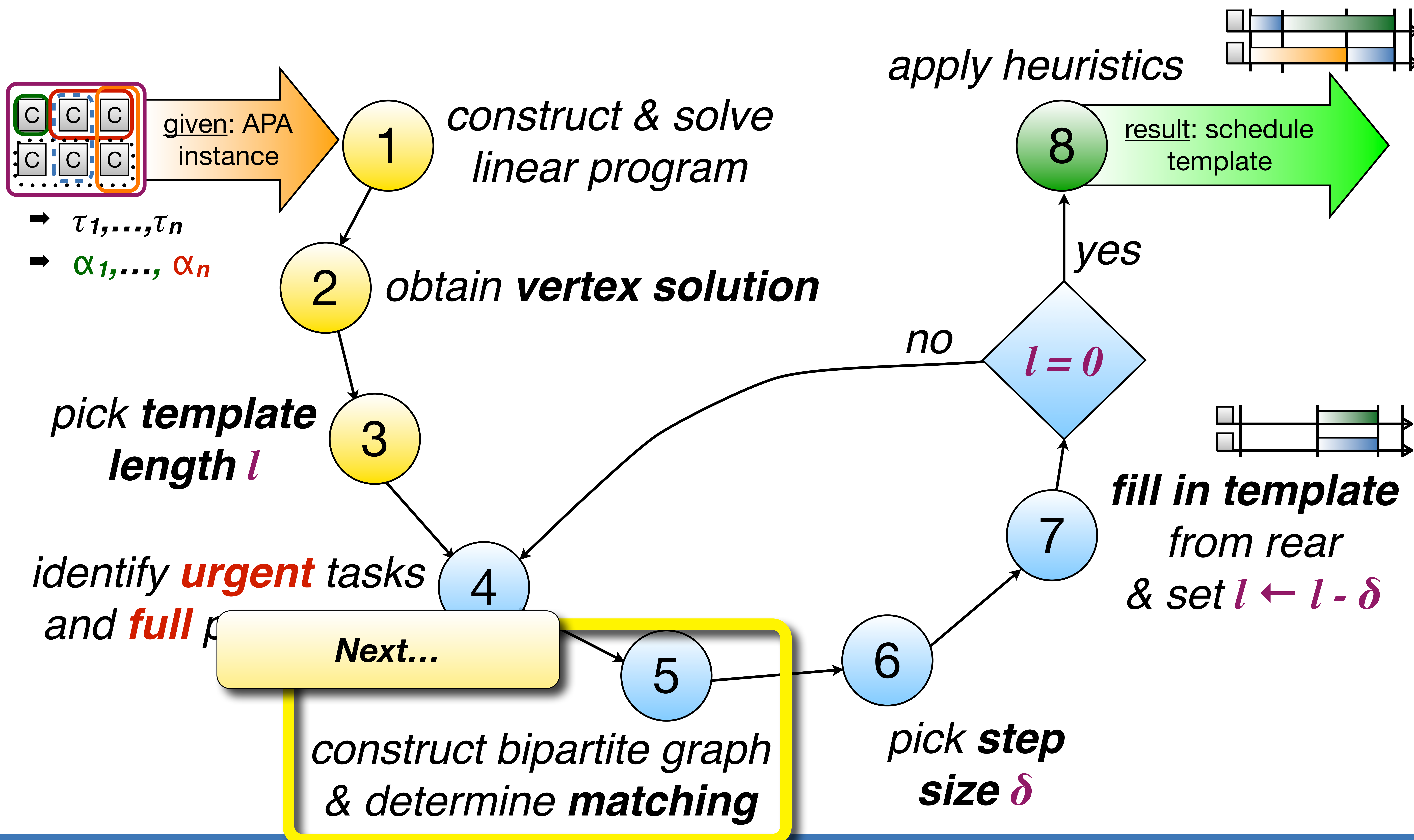
Utilization	τ_1	τ_2	τ_3	Total
Core 1	0.7	—	0.2	0.9
Core 2	—	0.6	0.3	0.9

Full processor

no idle time left



High-Level Overview: Iterative Template Construction



Step 5: Matching Tasks to Processors

*All **full** processors and all **urgent** tasks must be **matched**.*

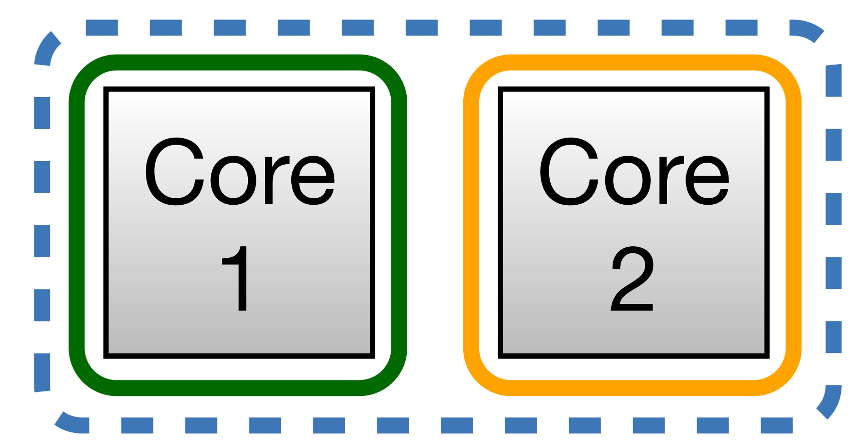
Step 5: Matching Tasks to Processors

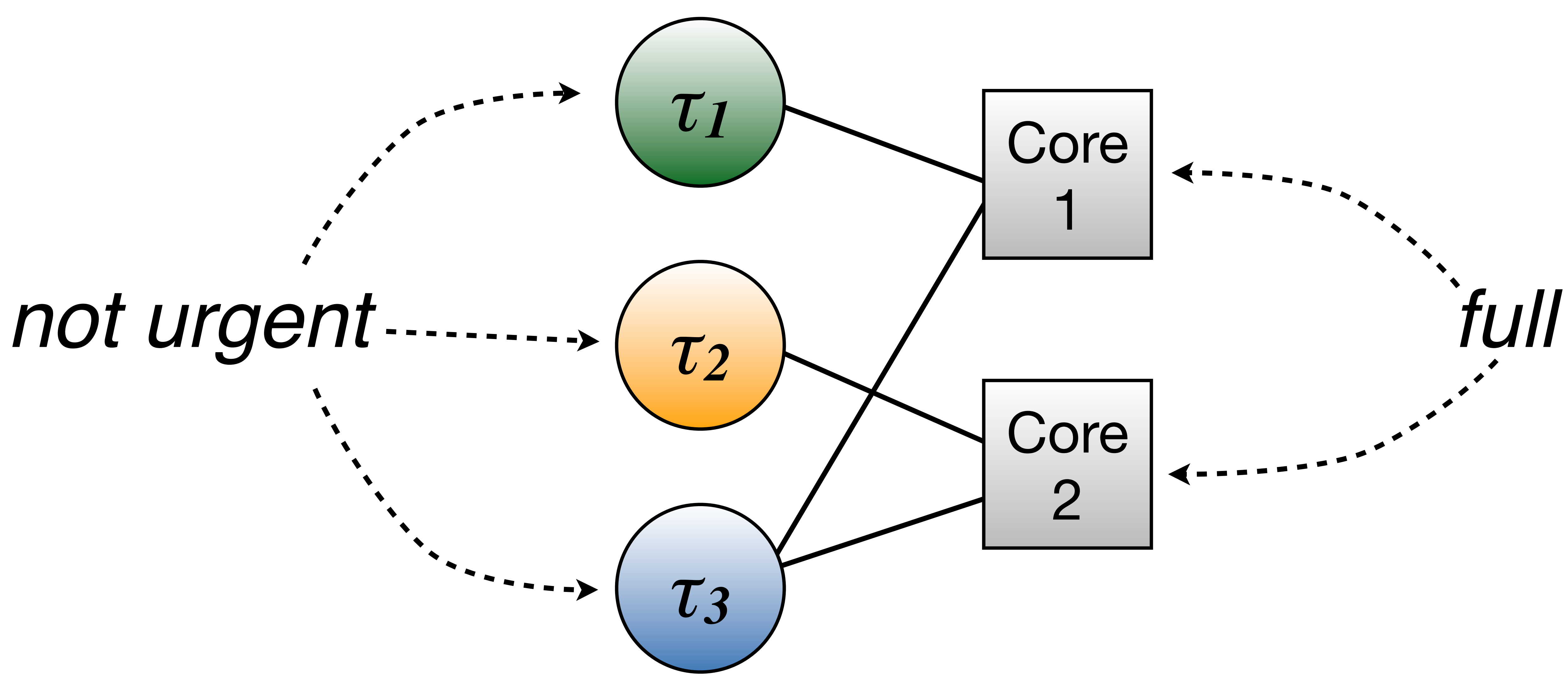
*All **full** processors and all **urgent** tasks must be **matched**.*

Define bipartite graph & find matching

- vertices: tasks & cores
- edges according to affinity

τ_i	$\mathbf{C}_i$	$\mathbf{T}_i$	α_i	$\mathbf{u}_i$
τ_1	7	10	$\{1\}$	0.7
τ_2	6	10	$\{2\}$	0.6
τ_3	10	20	$\{1, 2\}$	0.5





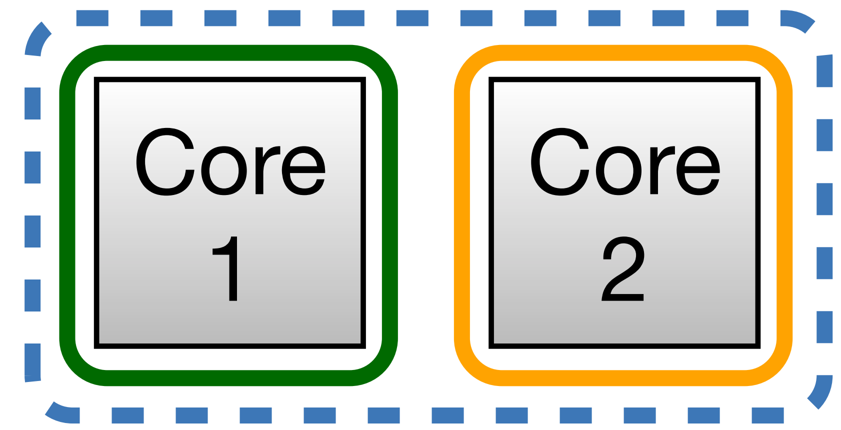
Step 5: Matching Tasks to Processors

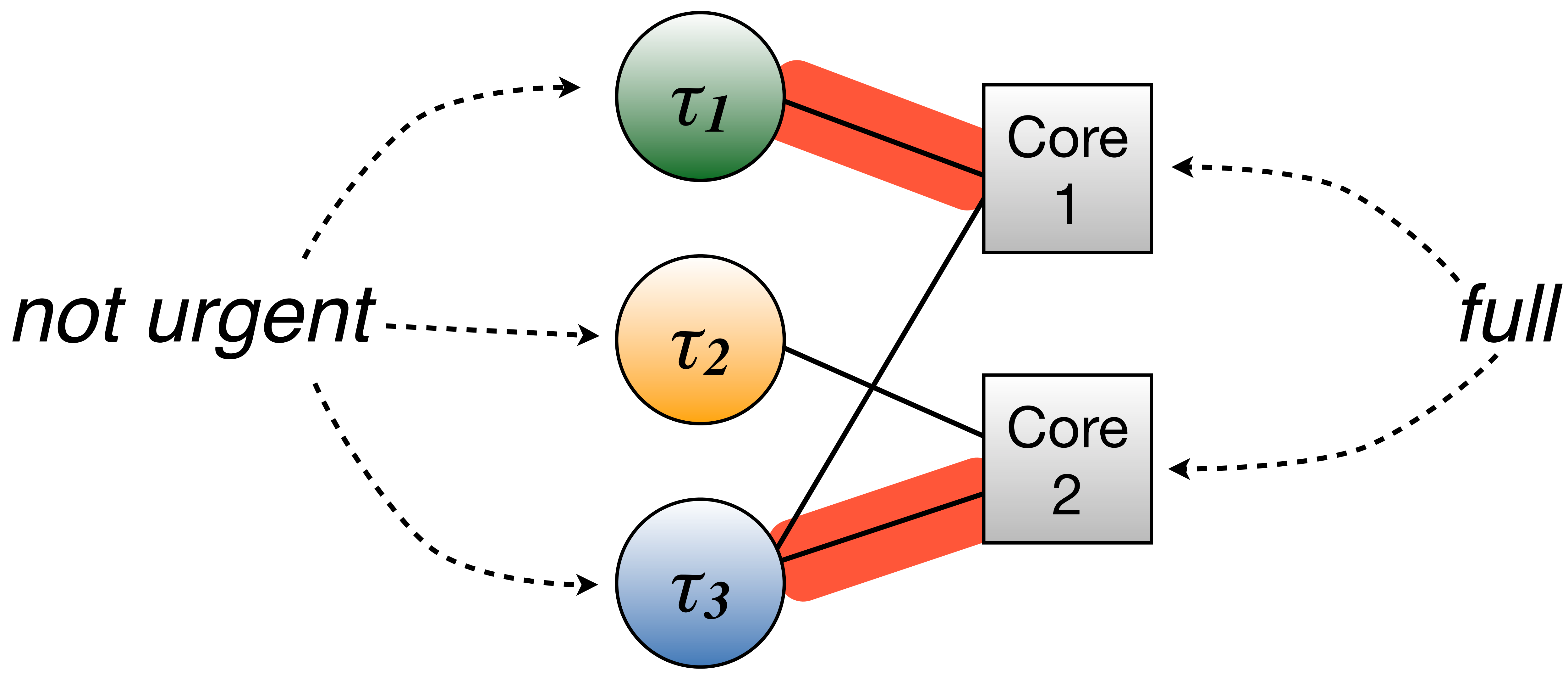
*All **full** processors and all **urgent** tasks must be **matched**.*

Define bipartite graph & find matching

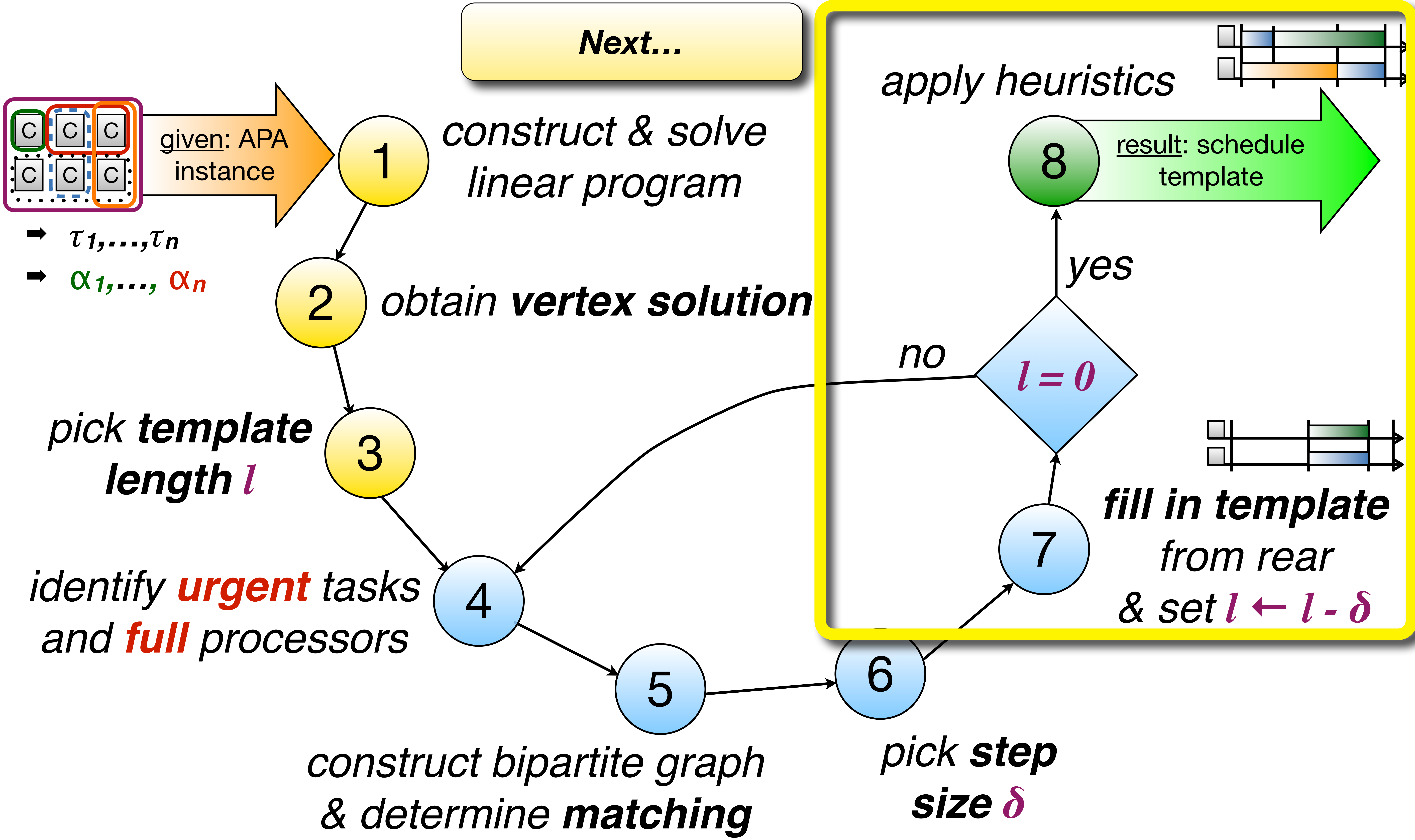
- vertices: tasks & cores
- edges according to affinity
- **construct matching (in three steps)**
- existence follows from ***Hall's theorem***

τ_i	$\mathbf{C}_i$	$\mathbf{T}_i$	α_i	$\mathbf{u}_i$
τ_1	7	10	$\{1\}$	0.7
τ_2	6	10	$\{2\}$	0.6
τ_3	10	20	$\{1, 2\}$	0.5

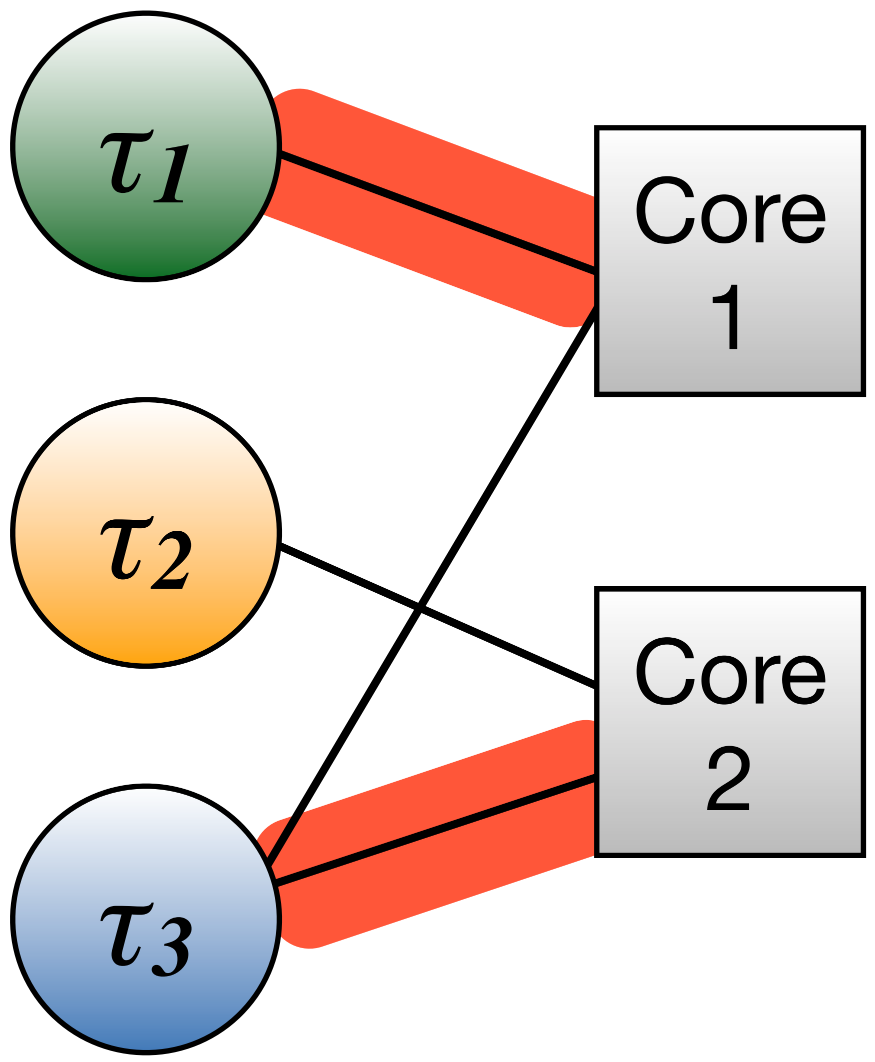




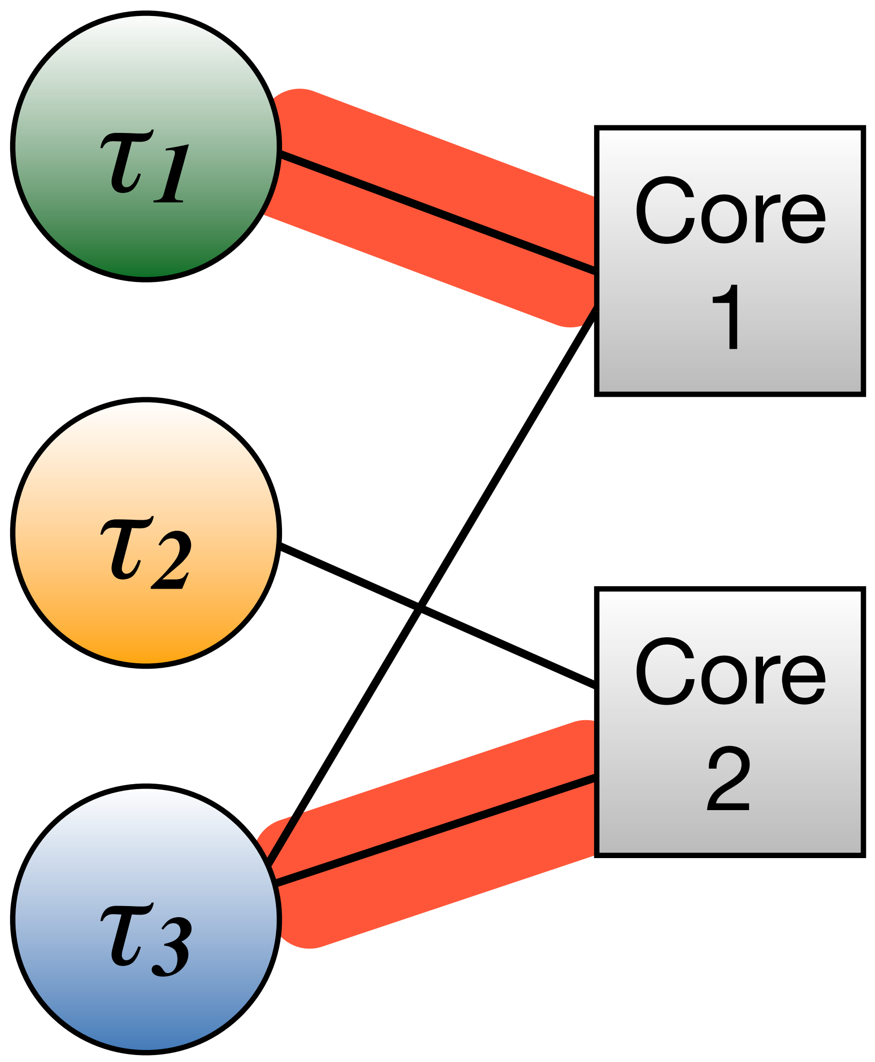
High-Level Overview: Iterative Template Construction



Step 7: Schedule Construction

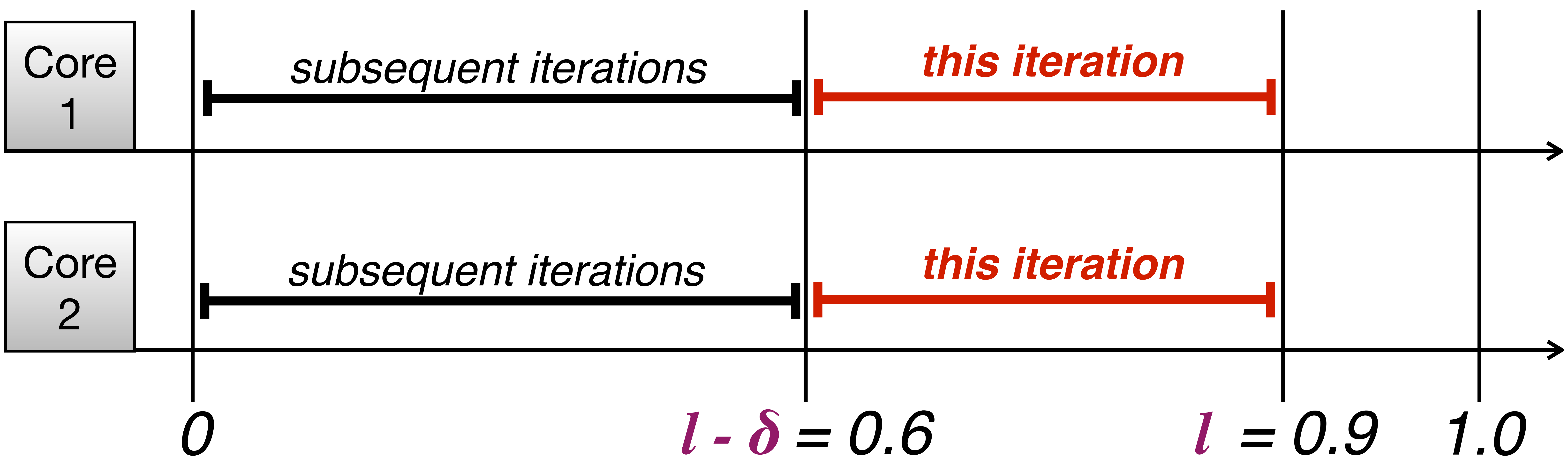


Step 7: Schedule Construction

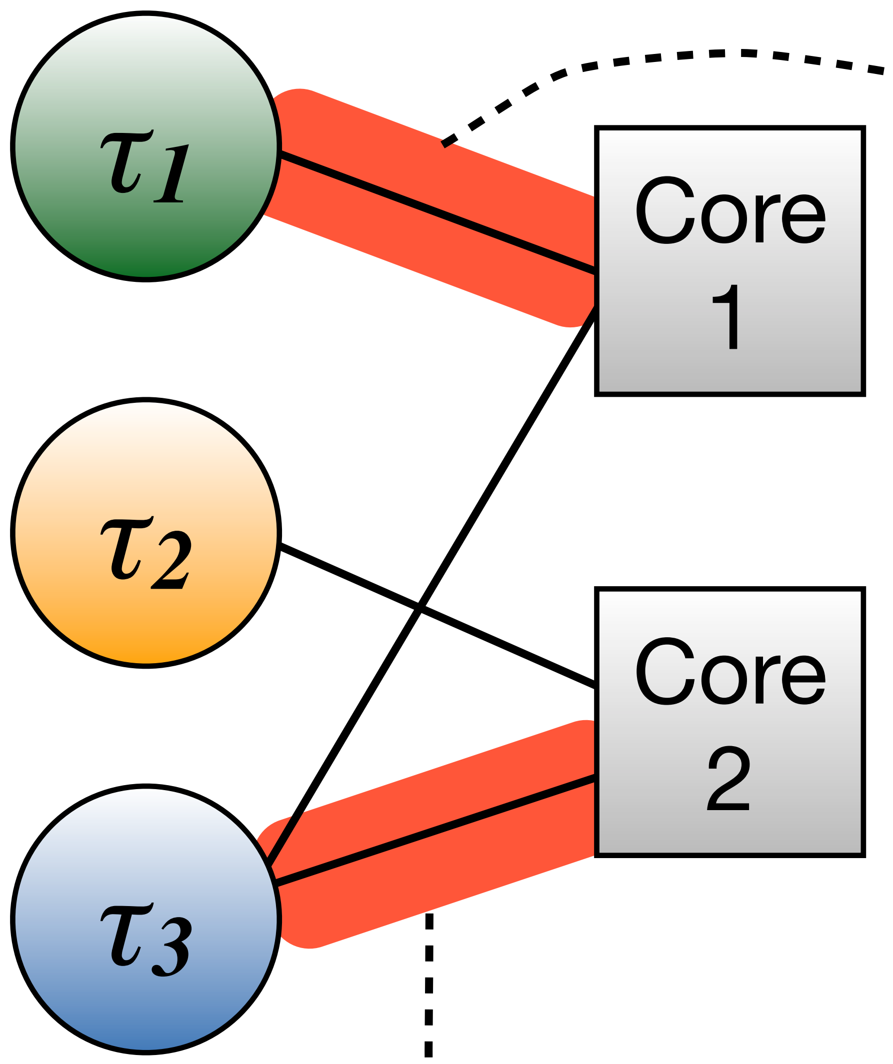


template length	step size
$l = 0.9$	$\delta = 0.3$

→ *assign tasks for interval (0.6, 0.9]*



Step 7: Schedule Construction



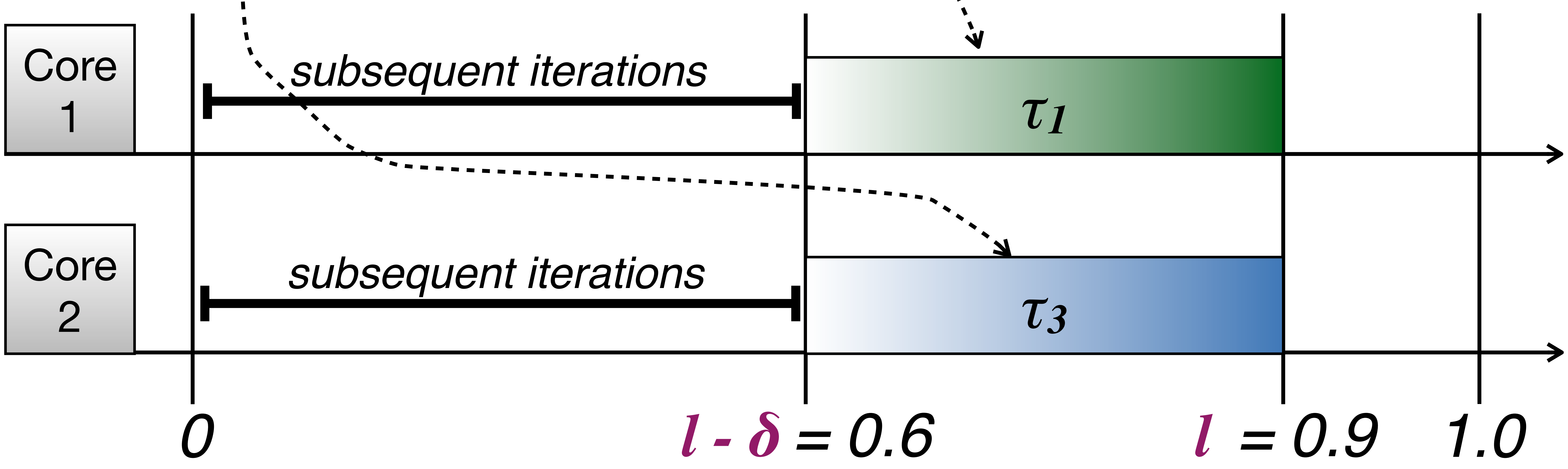
template length

$l = 0.9$

step size

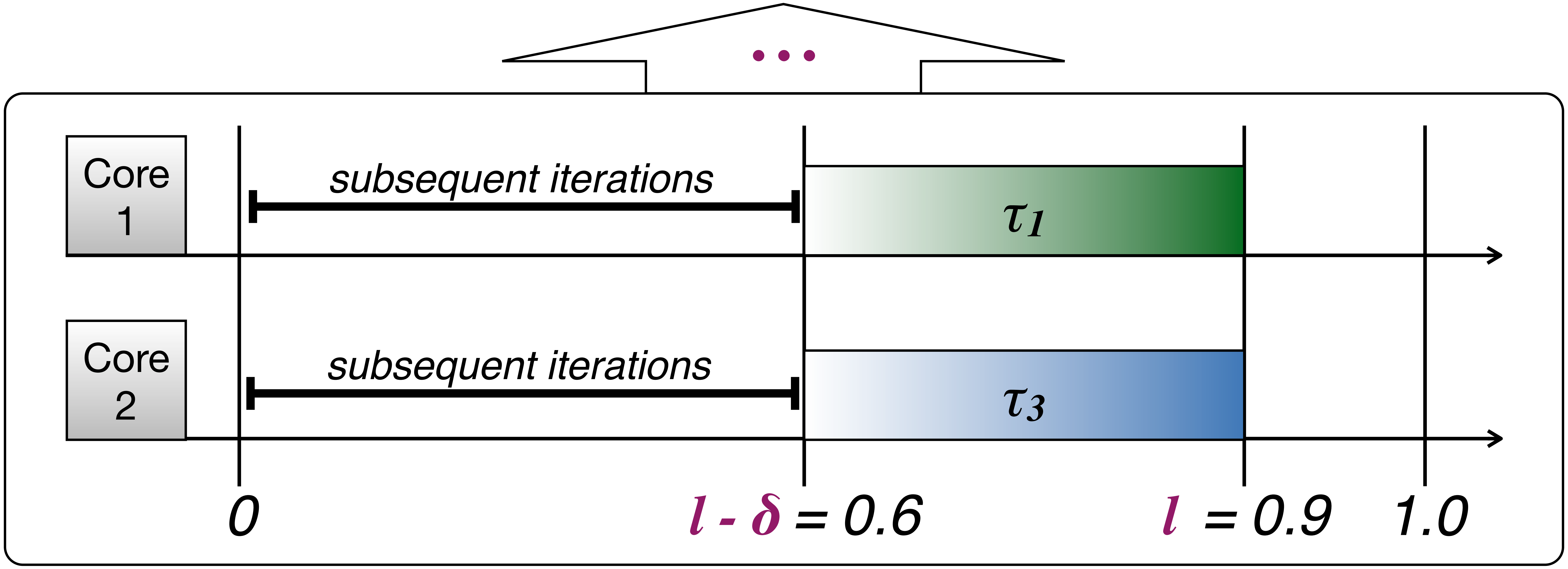
$\delta = 0.3$

→ *assign tasks for interval (0.6, 0.9]*

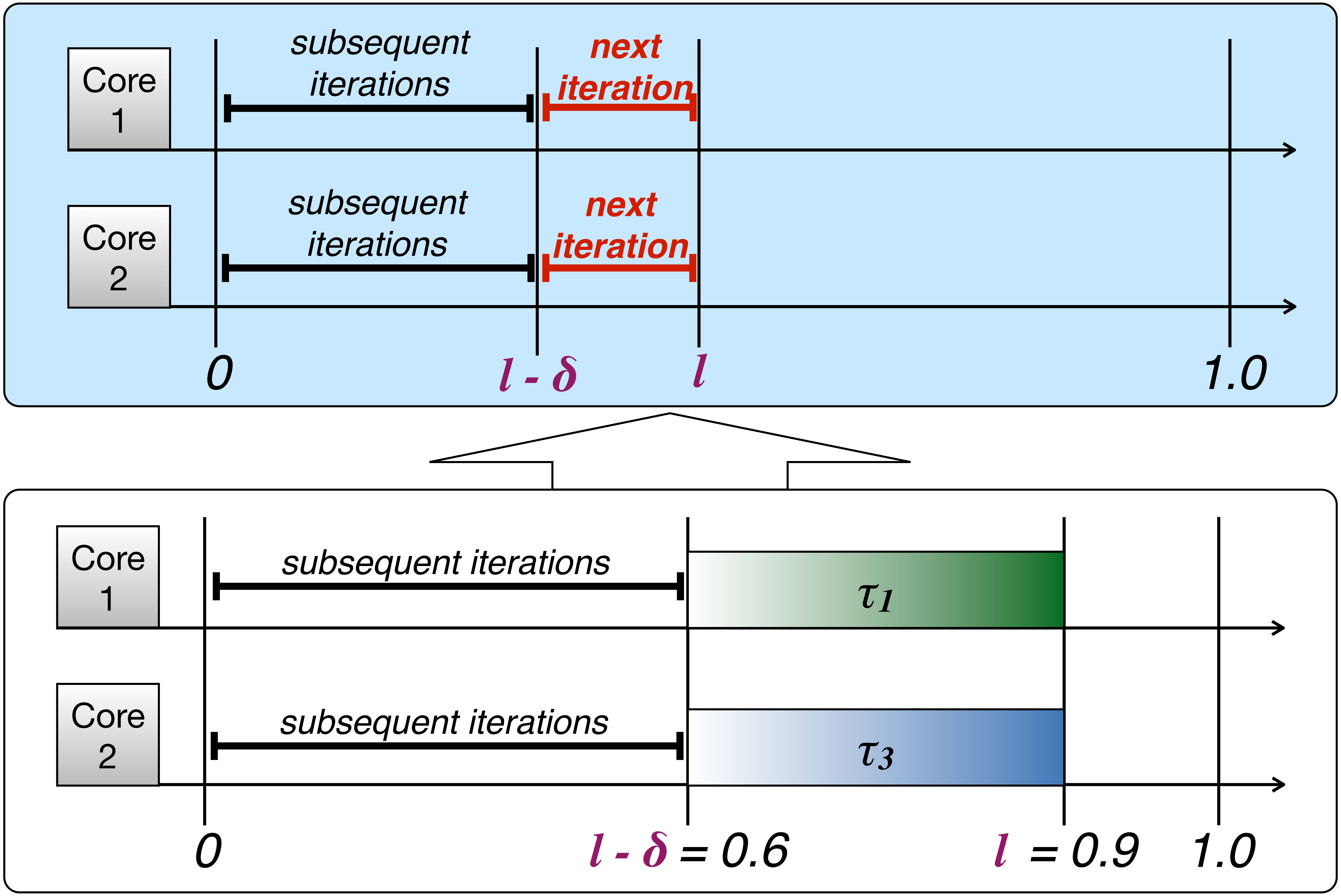


Finally, Update & Repeat

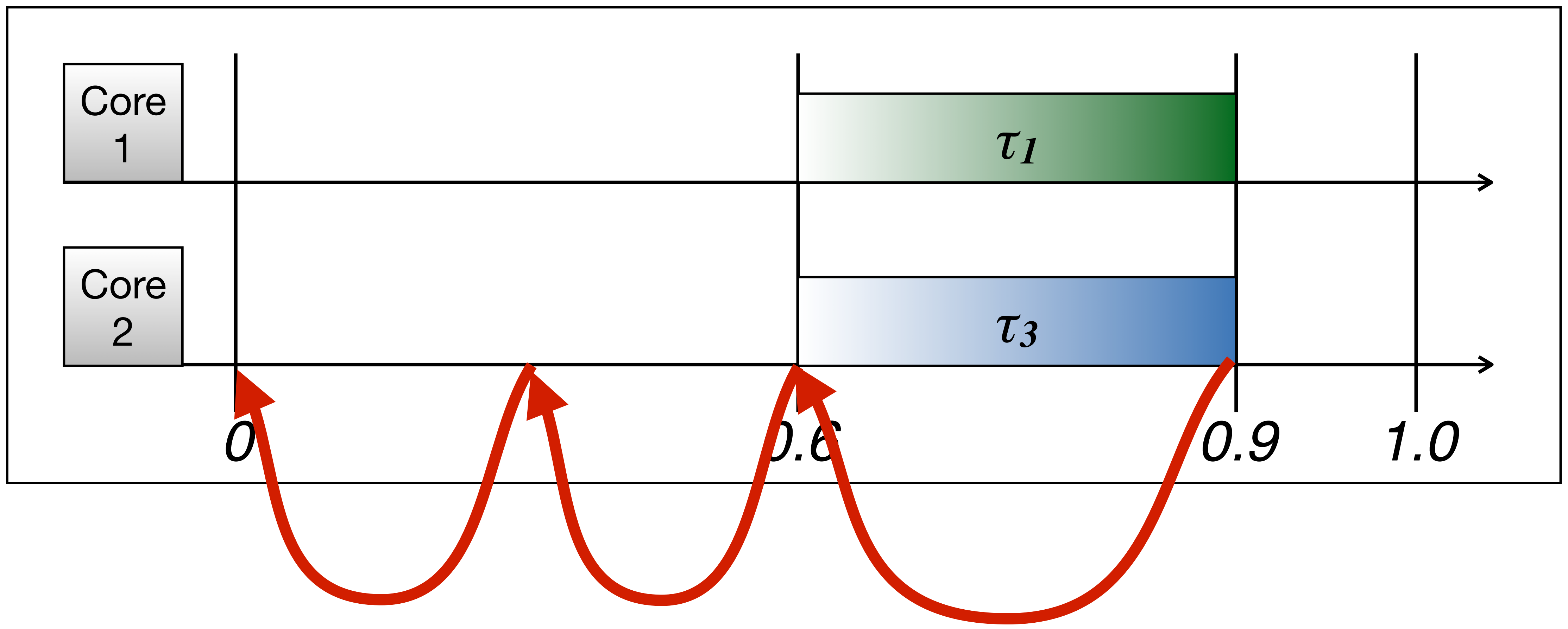
$l \leftarrow l - \delta$



Finally, Update & Repeat



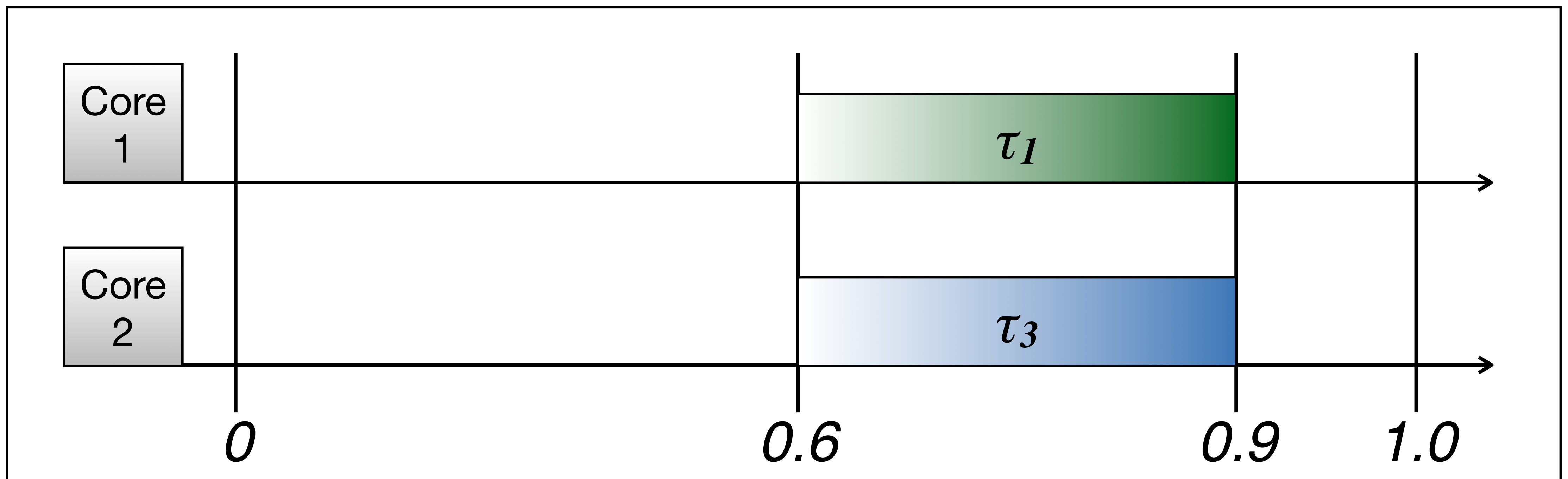
Iterate until *l=0*



τ_i	$\mathbf{C}_i$	$\mathbf{T}_i$	α_i	$\mathbf{u}_i$
τ_1	7	10	$\{1\}$	0.7
τ_2	6	10	$\{2\}$	0.6
τ_3	10	20	$\{1, 2\}$	0.5

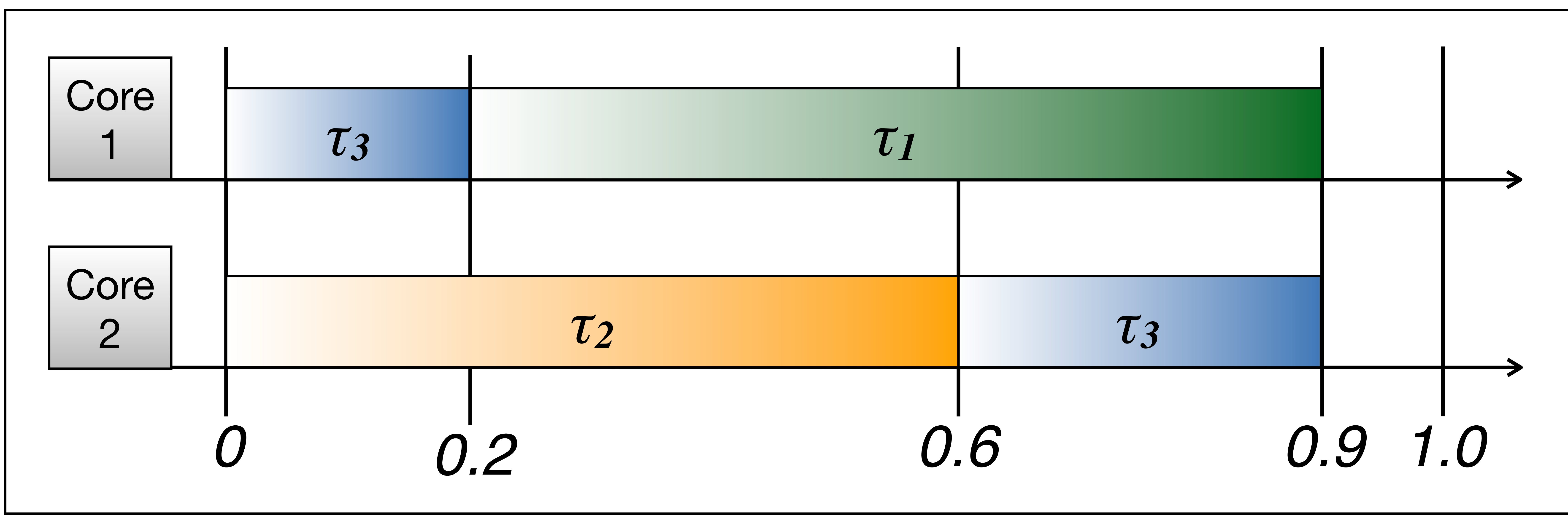
Fill in template from back to front…

Iterate until l=0



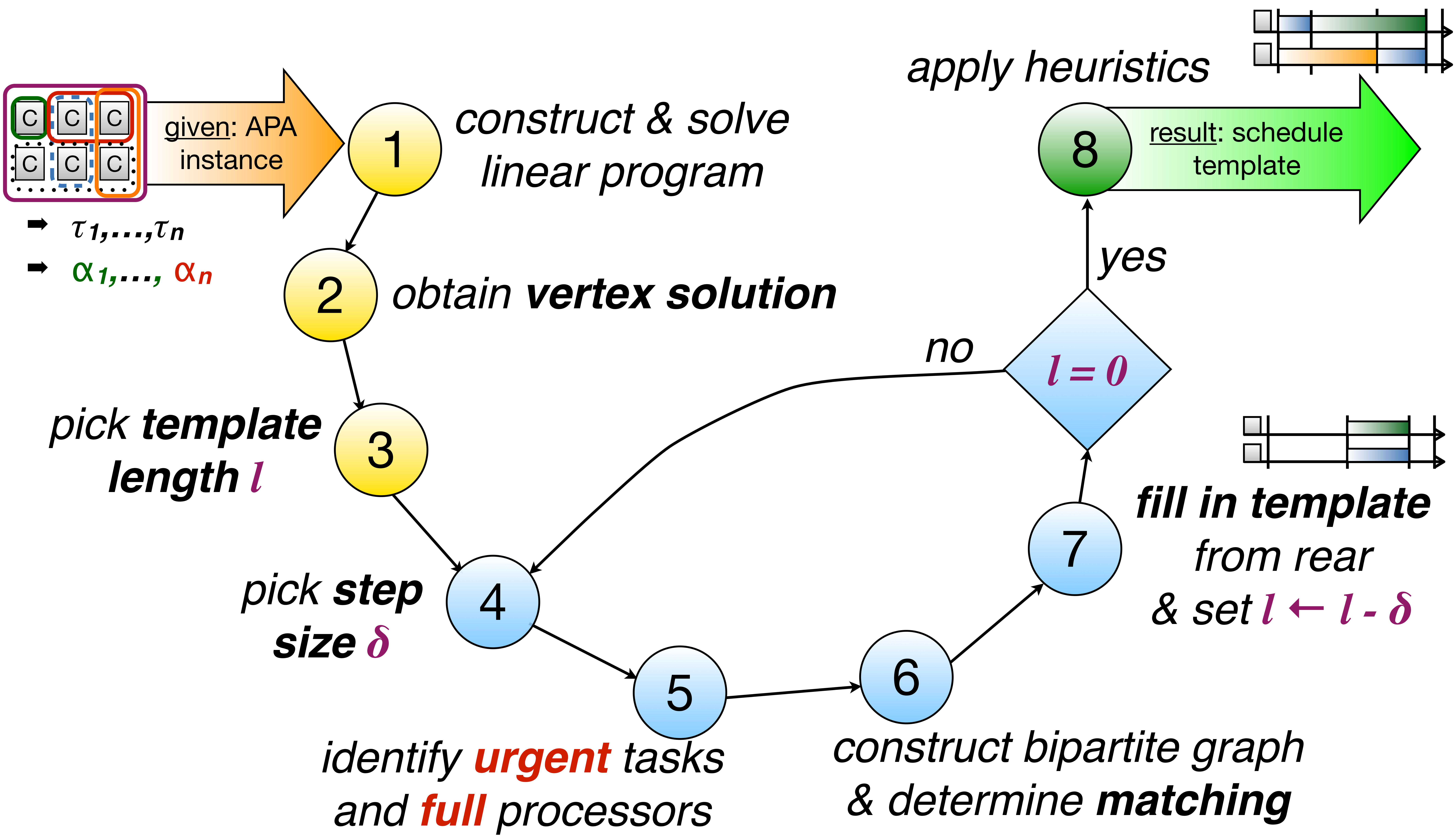
τ_i	$\mathbf{C}_i$	$\mathbf{T}_i$	α_i	$\mathbf{u}_i$
τ_1	7	10	$\{1\}$	0.7
τ_2	6	10	$\{2\}$	0.6
τ_3	10	20	$\{1,2\}$	0.5

after two more iterations
& heuristics to avoid preemptions…

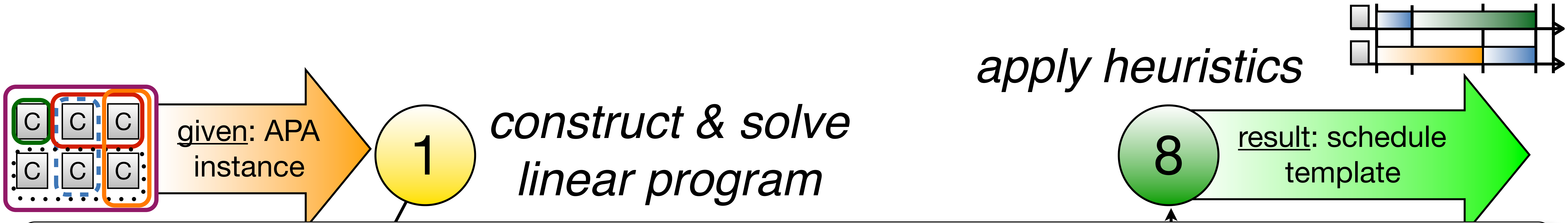


final template

High-Level Overview: Iterative Template Construction



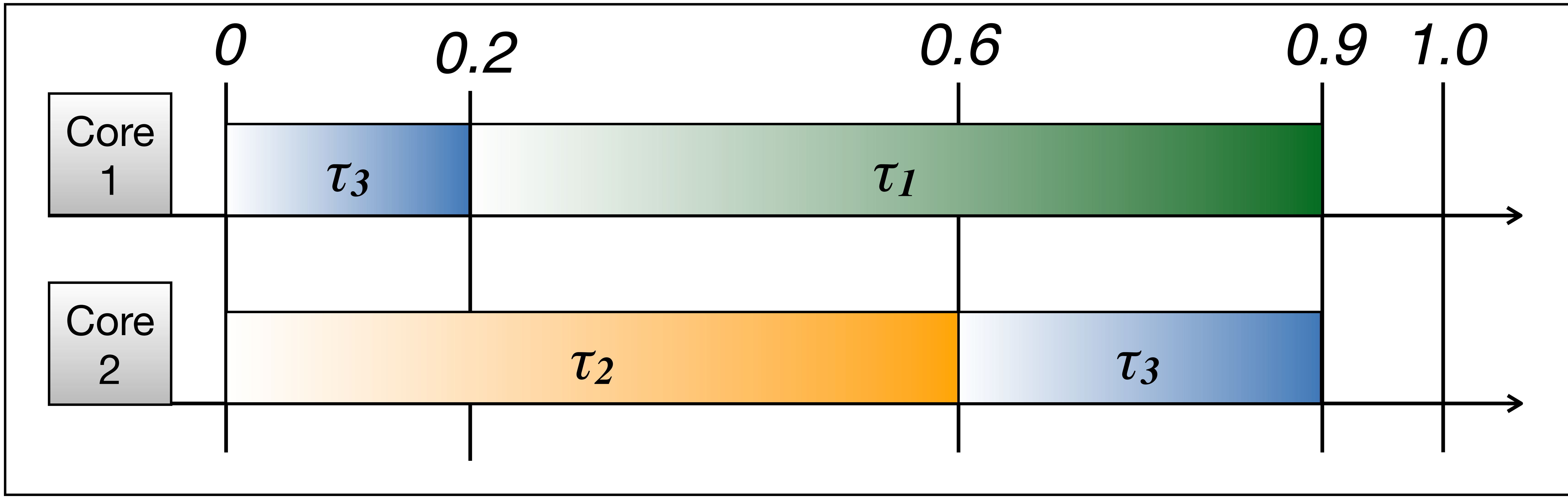
High-Level Overview: Iterative Template Construction



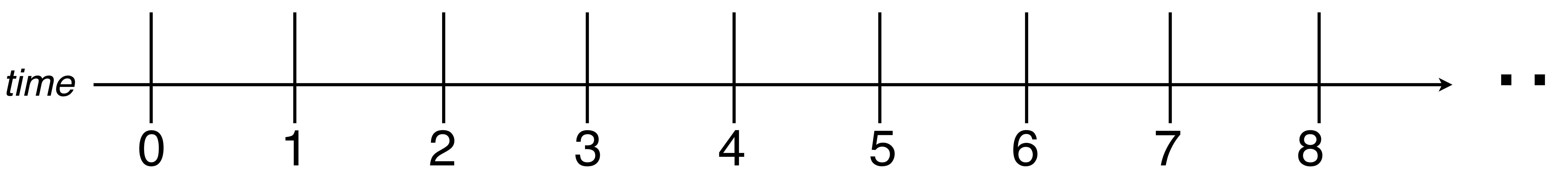
How to *use* the template?

How many tasks *migrate*?

How to use the template?

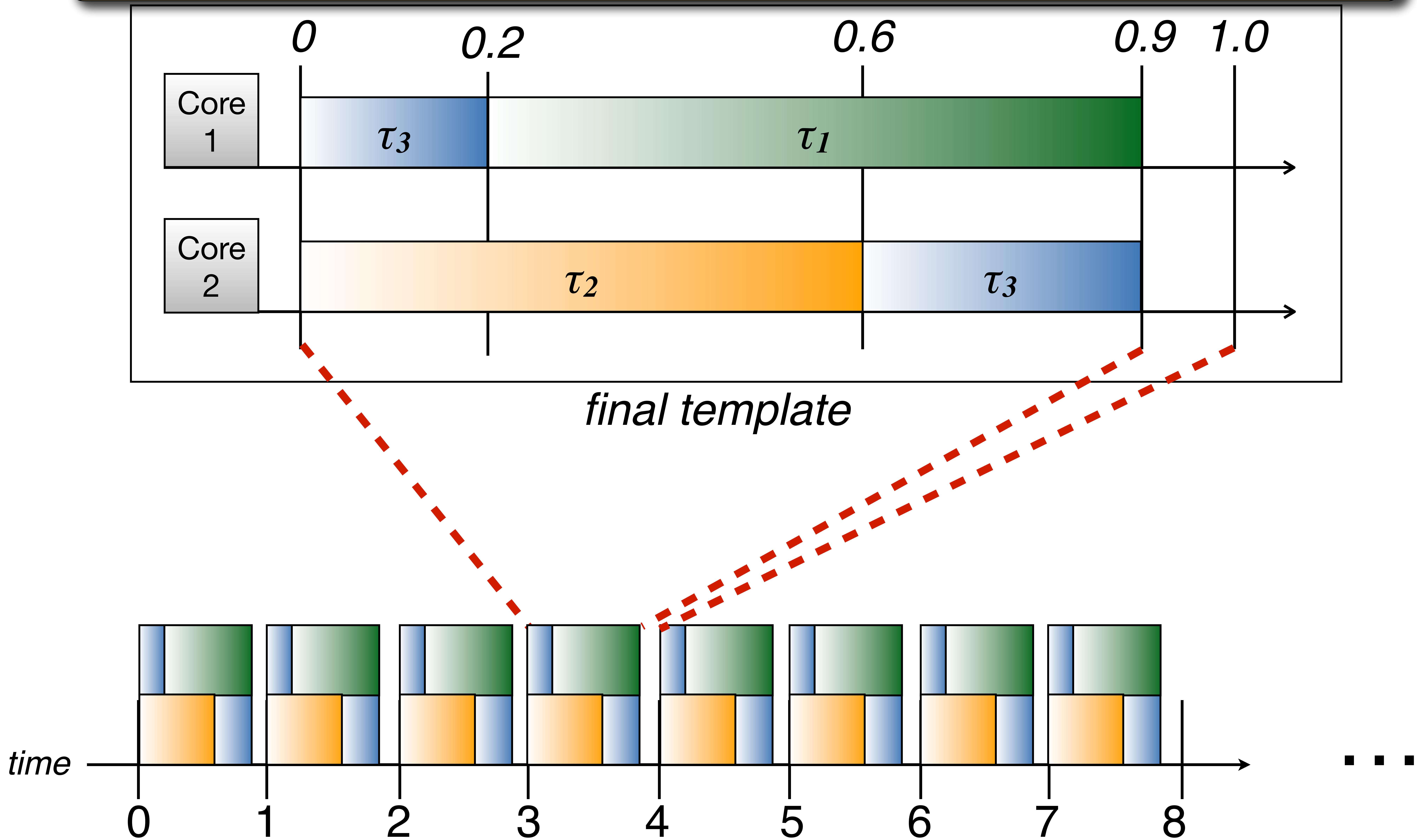


final template



Naïve approach: instantiate for each quantum (**impractical**, but shows **existence**).

*See paper for sketch of more practical **EDF-based** scheduler…*



Degree of Migration

Schedule template not very useful
*in practice if **"too many"** tasks migrate…*

Degree of Migration

Migration Bound

At most m tasks migrate.

*Other tasks are statically assigned to a single processor compliant with their PA: ➜ **semi-partitioning**.*

Degree of Migration

Migration Bound

At most m tasks migrate.

Why?

Vertex solution & **fewer constraints than variables** in LP

(details in paper…)

Open Questions

Open Questions

Why study APA scheduling?

Open Questions

Why study APA scheduling?

Relevance

Virtually **every** major (real-time) **operating system** supports PAs.
Practitioners have to (and **want** to) deal with it.

Open Questions

Why study APA scheduling?

Relevance

Virtually **every** major (real-time) **operating system** supports PAs.
Practitioners have to (and **want** to) deal with it.

Generality

Every solution assuming the **APA model** immediately solves the same problem for **global** / **partitioned** / **clustered** scheduling as well.

Open Problem 1/3
Feasibility with Constrained Deadlines

Requires reasoning about demand

➡ More than polynomial number of constraints (if done naively).

$$\forall t \geq 0 \quad \sum_{i=1}^{n} DBF(T_i, t) \leq t,$$

No longer “few” migrating tasks

➡ LP structure essential.

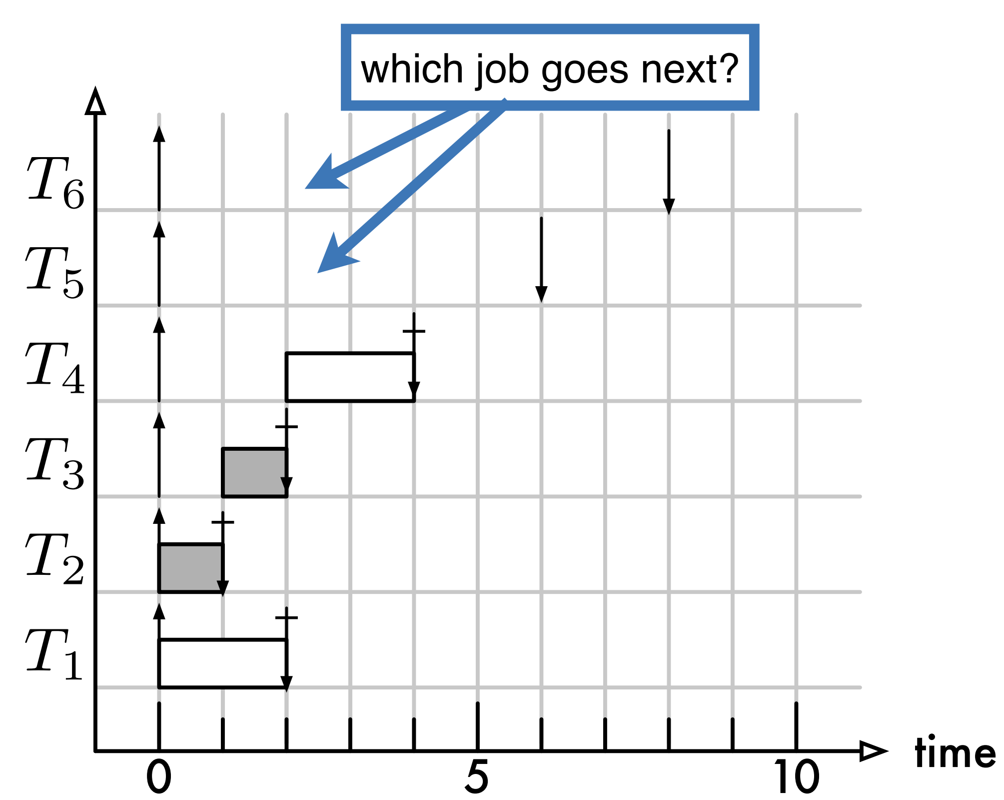
Open Problem 1/3
Feasibility with Constrained Deadlines

Requires reasoning about demand
➡ More than polynomial number of constraints (if done naively).

$$\forall t \geq 0 \quad \sum_{i=1}^{n} DBF(T_i, t) \leq t,$$

No longer "few" migrating tasks
➡ LP structure essential.

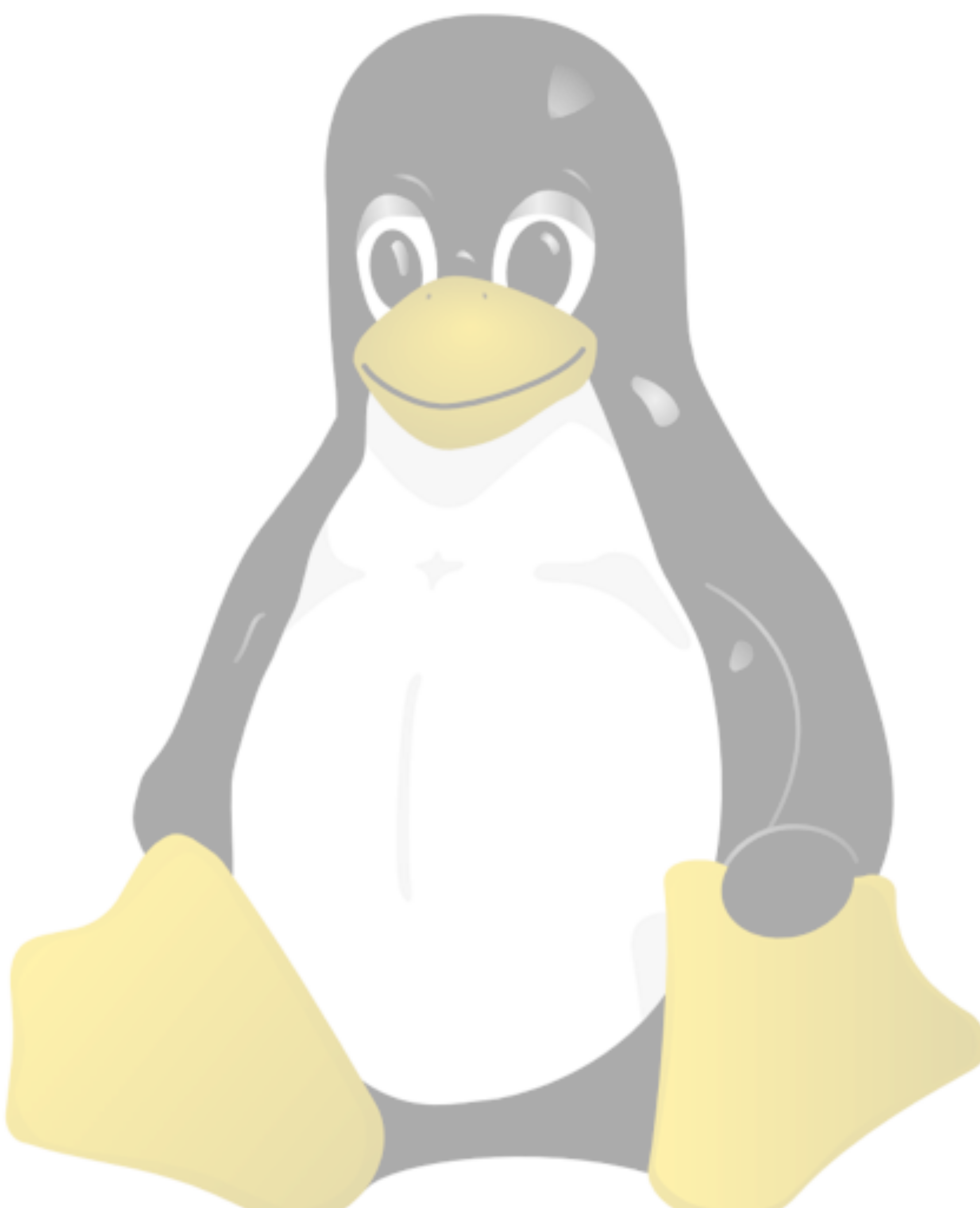
Task	WCET	Deadline	Period
T_1	2	2	5
T_2	1	1	5
T_3	1	2	6
T_4	2	4	100
T_5	2	6	100
T_6	4	8	100



Fisher, Goossens, Baruah (2010), Optimal online multiprocessor scheduling of sporadic real-time tasks is impossible. *Real-Time Systems*, volume 45, pp 26-71.

Optimal online scheduling with constrained deadlines
➡ Our test constructs online strategy.
➡ **Such a strategy does not always exists** if $d_i < p_i$.

Open Problem 2/3
Efficient Online APA Scheduling



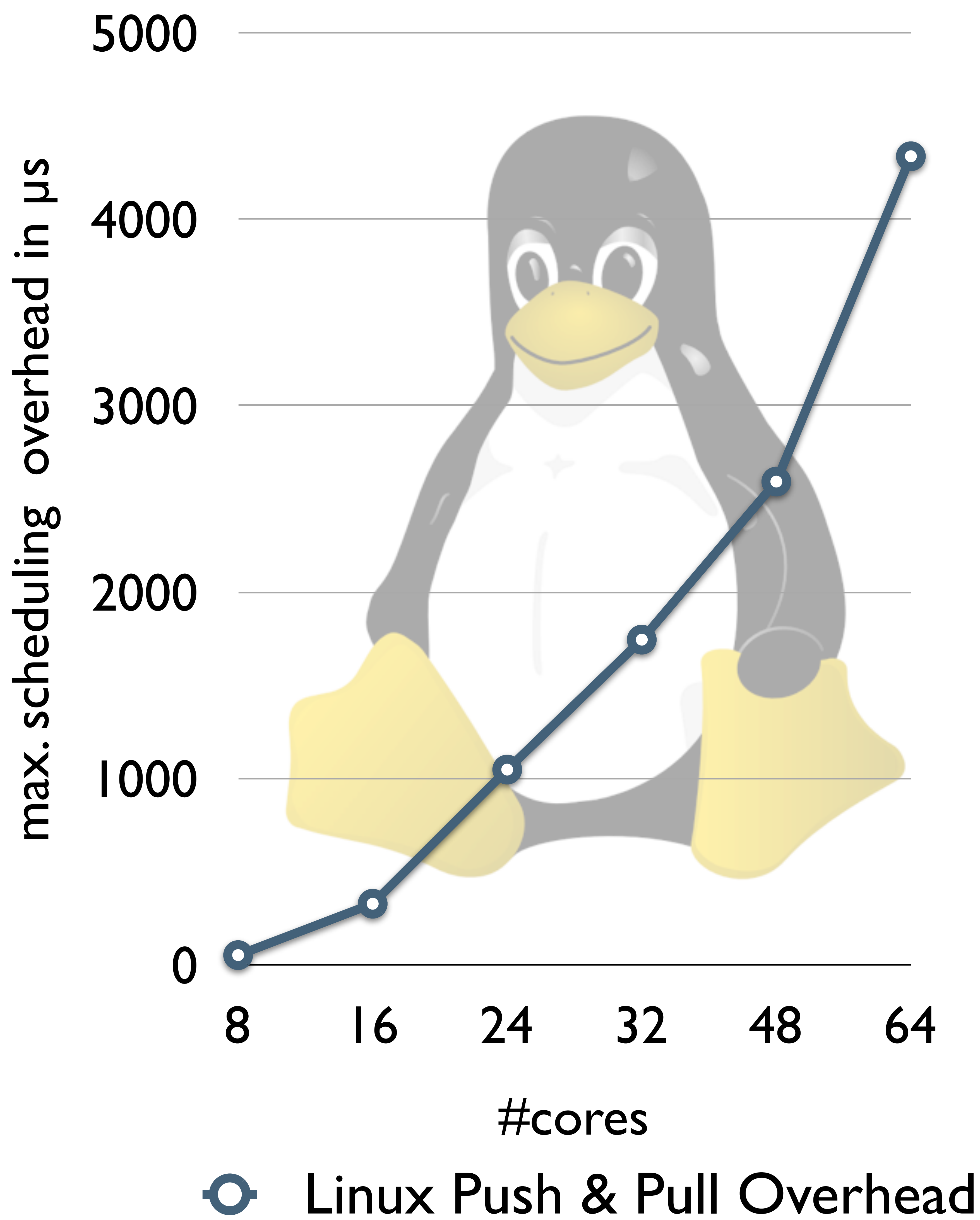
Open systems

➡ Cannot solve LP on `fork()`…

Why not use Linux's approach?

➡ well-known APA implementation

Open Problem 2/3
Efficient Online APA Scheduling



Open systems

➡ Cannot solve LP on `fork()`…

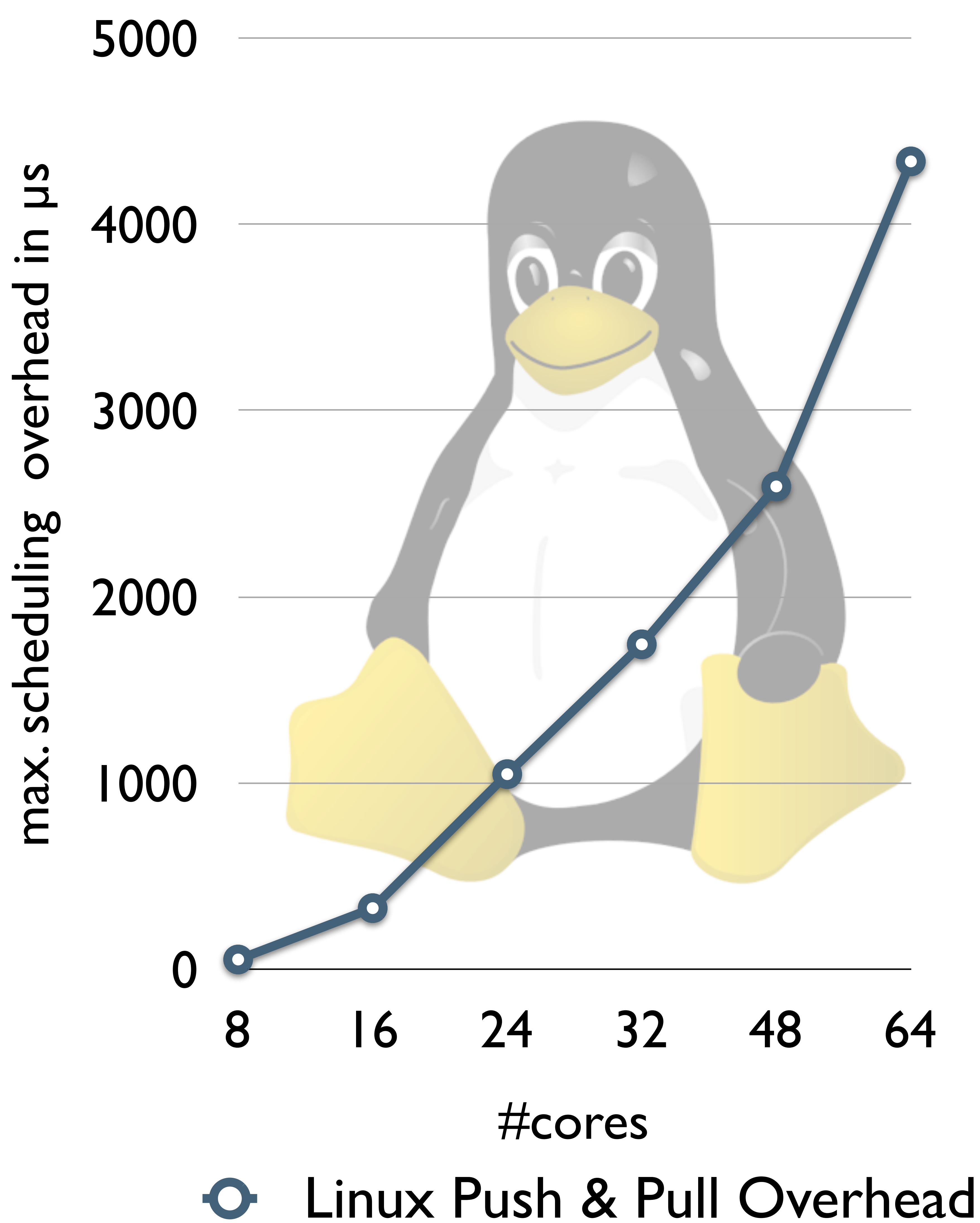
Why not use Linux's approach?

➡ well-known APA implementation

➡ inefficient

- **high maximum overheads**
- also w.r.t. **schedulability**

Open Problem 2/3
Efficient Online APA Scheduling



Open systems
- Cannot solve LP on `fork()`…

Why not use Linux's approach?
- well-known APA implementation
- inefficient
  - **high maximum overheads**
  - also w.r.t. **schedulability**

Goal
- Fast scheduler
- Fast task admission
- Efficient APA semantics
- **Semi-partitioned**?

Open Problem 3/3
JLFP/FP Feasibility and Priority Assignment

This paper: feasibility in general

- ➡ Solution: “non-standard” dynamic priority scheduler
- ➡ Industry prefers simpler fixed-priority scheduling

WIND RIVER FreeBSD Microsoft QNX

Open Problem 3/3

JLFP/FP Feasibility and Priority Assignment

This paper: feasibility in general

- ➡ Solution: “non-standard” dynamic priority scheduler
- ➡ Industry prefers simpler fixed-priority scheduling

Open problem: FP feasibility

- ➡ Exact schedulability test …also for global scheduling.

Open Problem 3/3
JLFP/FP Feasibility and Priority Assignment

This paper: feasibility in general

- ➡ Solution: “non-standard” dynamic priority scheduler
- ➡ Industry prefers simpler fixed-priority scheduling

WIND RIVER FreeBSD Microsoft QNX

Open problem: FP feasibility

- ➡ Exact schedulability test …also for global scheduling.

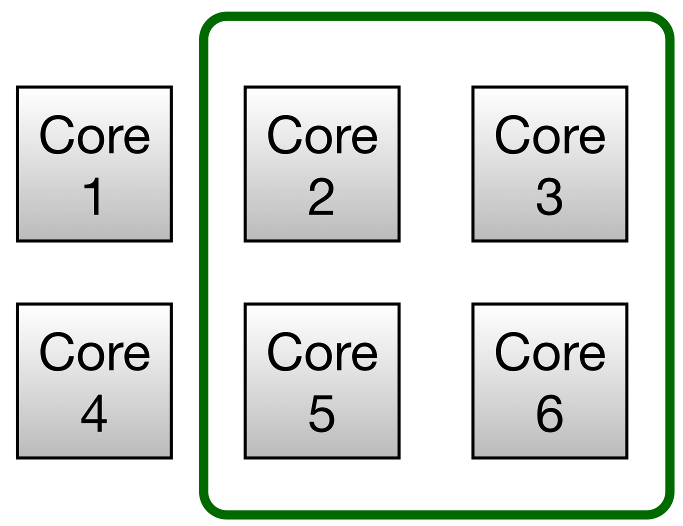
Joint priority and affinity assignment

- ➡ “Good” **priority** assignment can mask “bad” processor **affinities**.
- ➡ “Good” processor **affinities** can mask “bad” **priority** assignment.

Conclusion

APA

arbitrary processor affinity



α_i = {2, 3, 5, 6}

QNX

WIND RIVER

FreeBSD

Microsoft

THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

Max Planck Institute for Software Systems

Conclusion

APA
arbitrary processor affinity

Core 1	Core 2	Core 3
Core 4	Core 5	Core 6

α_i = {2, 3, 5, 6}

A polynomial-time feasibility test for the **APA scheduling problem**.

The APA Scheduling Problem

APA instance
Task set with designer-specified **APAs**.

Offline APA optimization
Reduce **APAs** such that task set remains (or **becomes**) schedulable

Online APA scheduling
OS scheduler enforces **reduced APAs**

Conclusion

APA
arbitrary processor affinity

Core 1	Core 2	Core 3
Core 4	Core 5	Core 6

α_i = {2, 3, 5, 6}

A polynomial-time feasibility test for the **APA scheduling problem**.

Constructs a **semi-partitioned** schedule template with **at most *m* migrating tasks**.

The APA Scheduling Problem

APA instance
Task set with designer-specified **APAs**.

Offline APA optimization
Reduce APAs such that task set remains (or **becomes**) schedulable

Online APA scheduling
OS scheduler enforces **reduced APAs**

QNX

WIND RIVER

FreeBSD

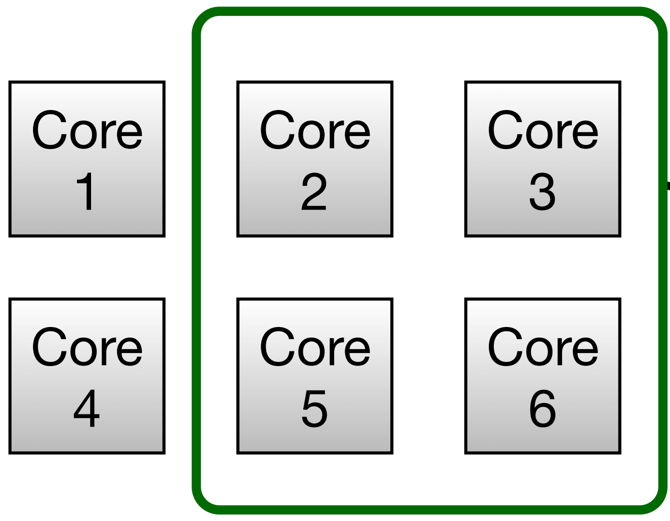
Microsoft

THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

Max Planck Institute for Software Systems

Conclusion

APA
arbitrary processor affinity



α_i = {2, 3, 5, 6}

A polynomial-time feasibility test for the **APA scheduling problem**.

Constructs a **semi-partitioned** schedule template with **at most *m* migrating tasks**.

A **general model** and **conceptual framework** that **originates in practice** with plenty of open problems!

The APA Scheduling Problem

APA instance
Task set with designer-specified **APAs**.

Offline APA optimization
Reduce APAs such that task set remains (or **becomes**) schedulable

Online APA scheduling
OS scheduler enforces **reduced APAs**

THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

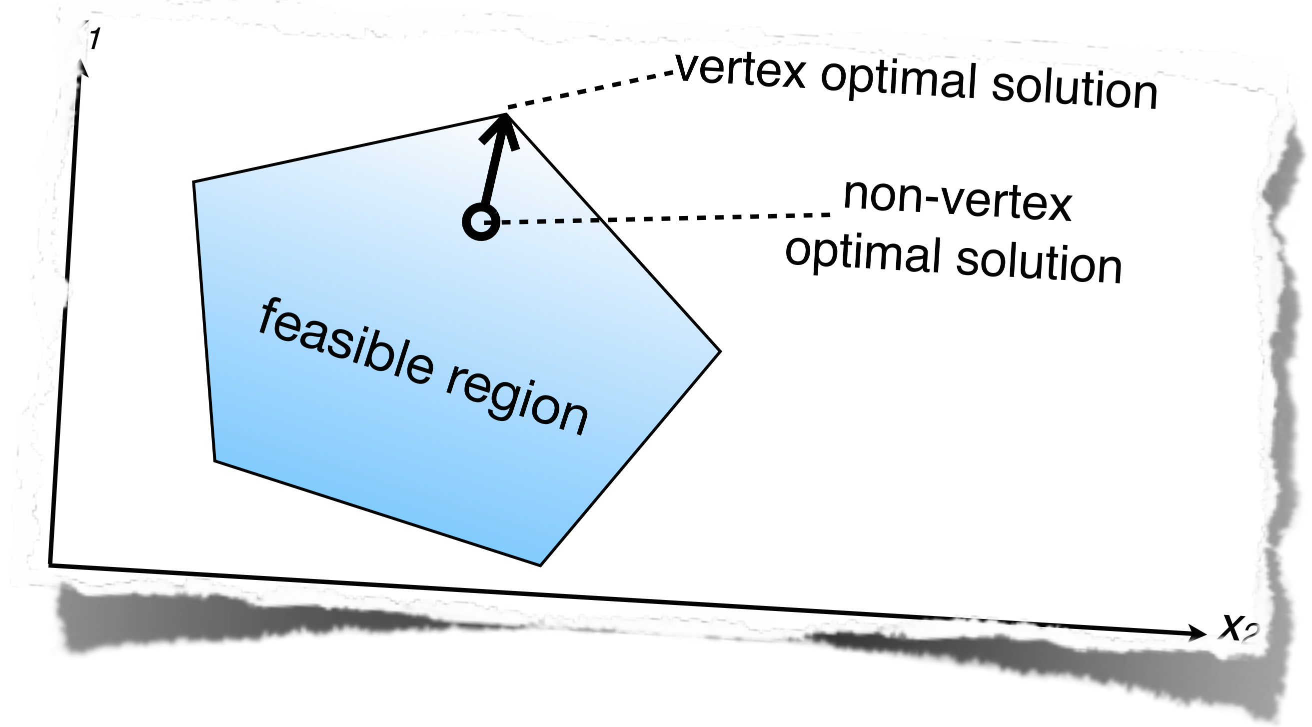
Max Planck Institute for Software Systems

Appendix

Degree of Migration – Explanation

Construct linear program
➡ $n \times m$ variables and $n + m$ constraints

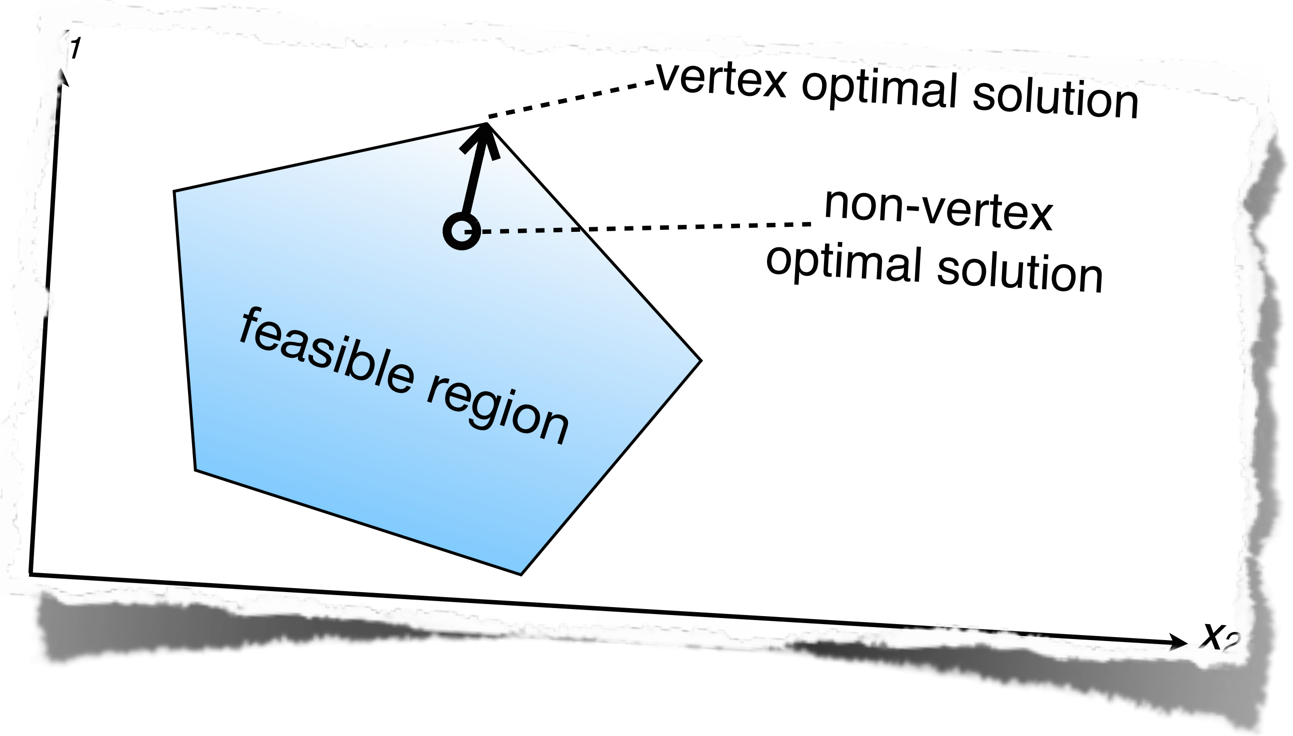
+



Degree of Migration – Explanation

Construct linear program
➡ $n \times m$ variables and $n + m$ constraints

+



A basic fact about linear programs…

+

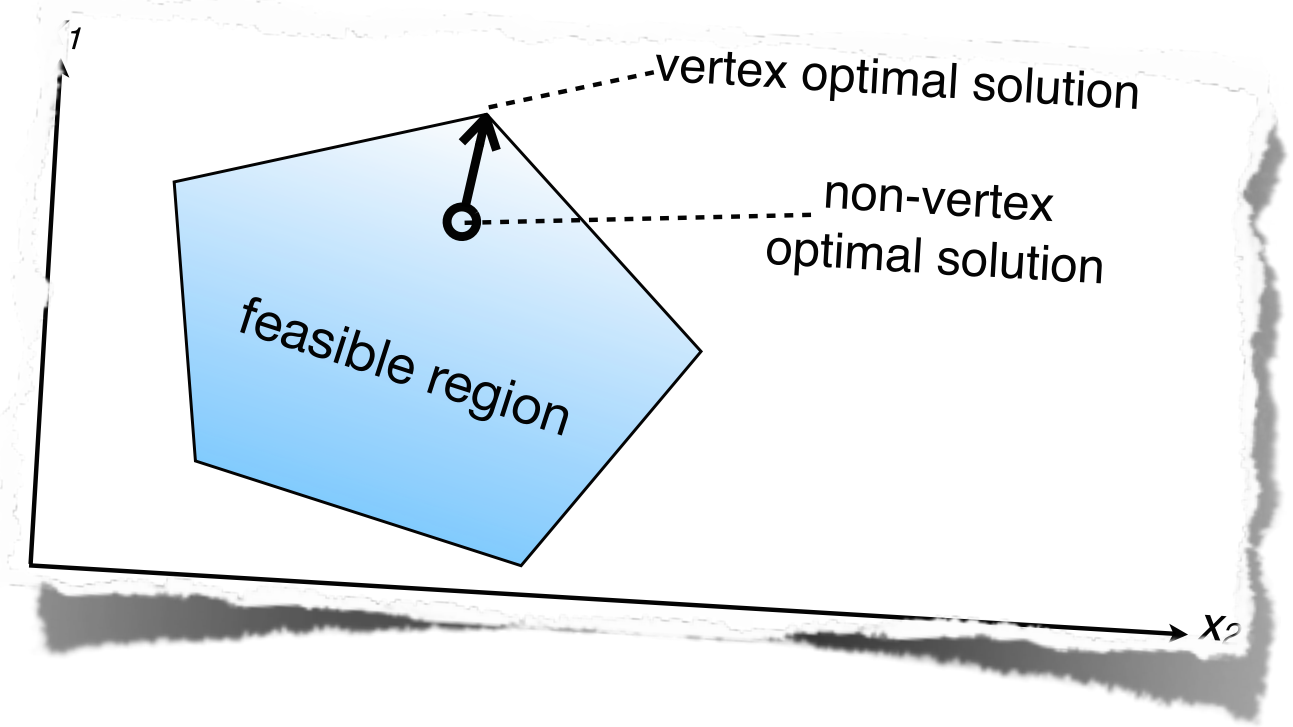
A linear program with…
➡ *X* non-negative variables (= $n \times m$),
➡ *C* additional constraints (= $n + m$).

If *C* < *X*, then…
➡ at most *C* non-zero values (= $n + m$)
➡ at each **vertex of the feasible region**.

Degree of Migration – Explanation

Construct linear program
➡ $n \times m$ variables and $n + m$ constraints

+



A basic fact about linear programs…

+

A linear program with…
➡ *X* non-negative variables (= $n \times m$),
➡ *C* additional constraints (= $n + m$).

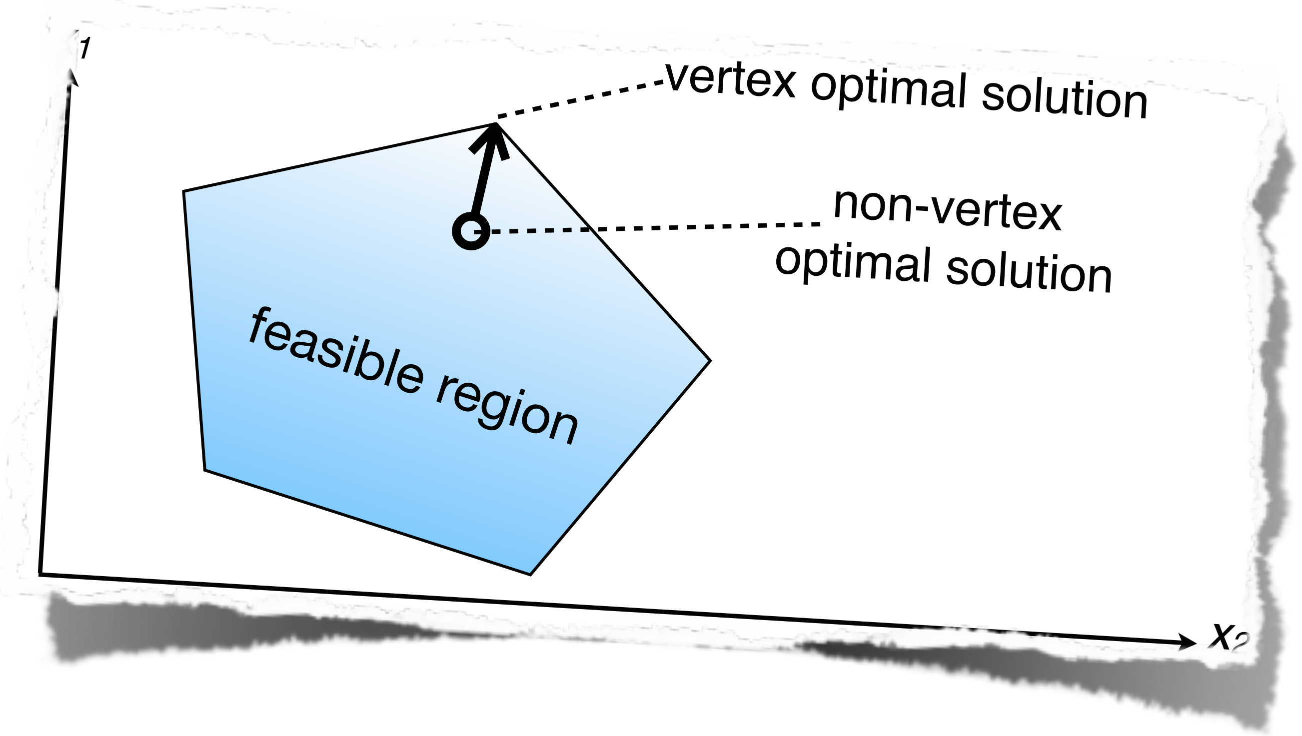
If *C* < *X*, then…
➡ at most *C* non-zero values (= $n + m$)
➡ at each **vertex of the feasible region**.

⟶ At most $n + m$ utilization fractions x_{ij} are non-zero.

Degree of Migration — Explanation

Construct linear program
➡ $n \times m$ variables and $n + m$ constraints

+



A basic fact about linear programs…

+

A linear program with…
➡ ***X*** non-negative variables (= $n \times m$),
➡ ***C*** additional constraints (= $n + m$).

If *C* < *X*, then…
➡ at most ***C*** non-zero values (= $n + m$)
➡ at each **vertex of the feasible region**.

→ At most $n + m$ utilization fractions x_{ij} are non-zero.

➜ at most m of the n tasks are associated with more than one non-zero variable (= at most m migrate).