Compositional Virtual Timeline: Verifying Dynamic-Priority Partitions with Algorithmic Temporal Isolation

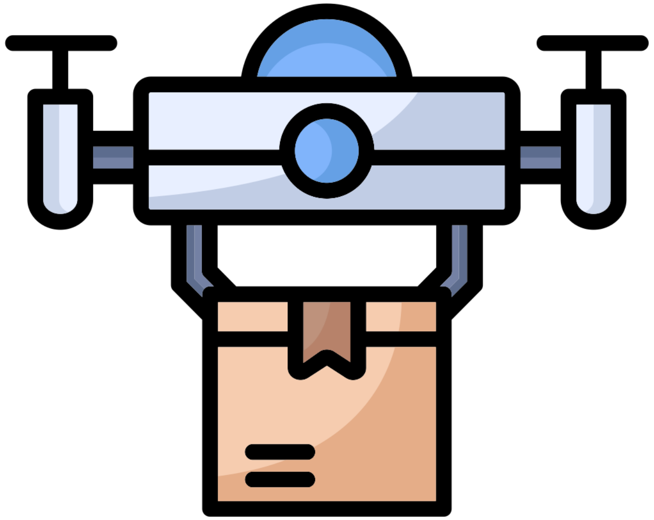
Mengqi Liu[1], **Zhong Shao**, Hao Chen, Man-Ki Yoon[2], Jung-Eun Kim[2]

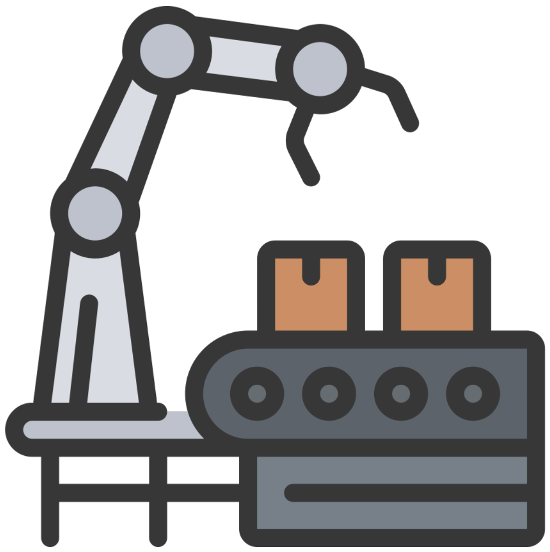
Yale University

1. Mengqi Liu is now at Alibaba
2. Man-Ki Yoon and Jung-Eun Kim are now at North Carolina State University

Real-Time Systems and Temporal Isolation

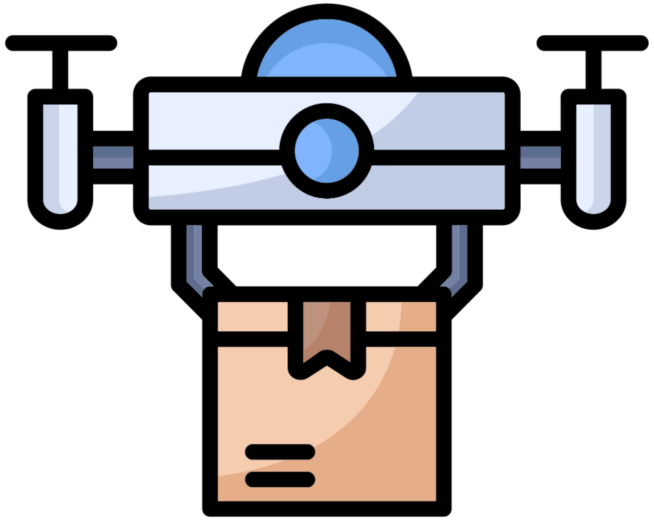
Real-time systems power our world today





Real-Time Systems and Temporal Isolation

Modern systems may integrate components from various vendors

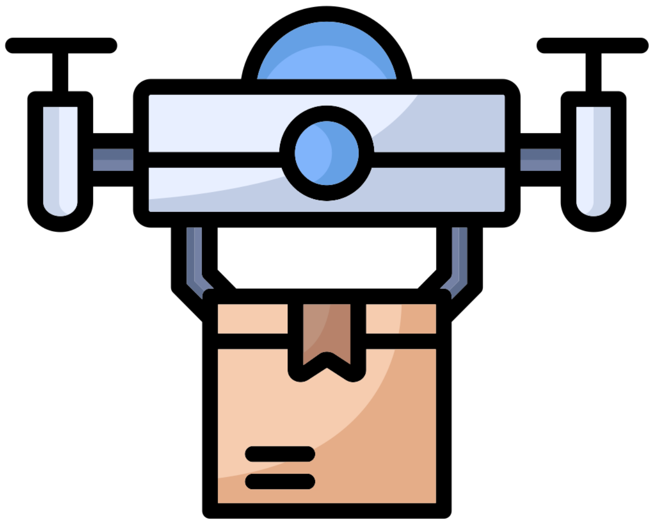


- Flight Controller —— In-House
- Visual Recognition —— Vendor A
- Navigation —— Vendor B
- Payload Handling —— Vendor C
- ……

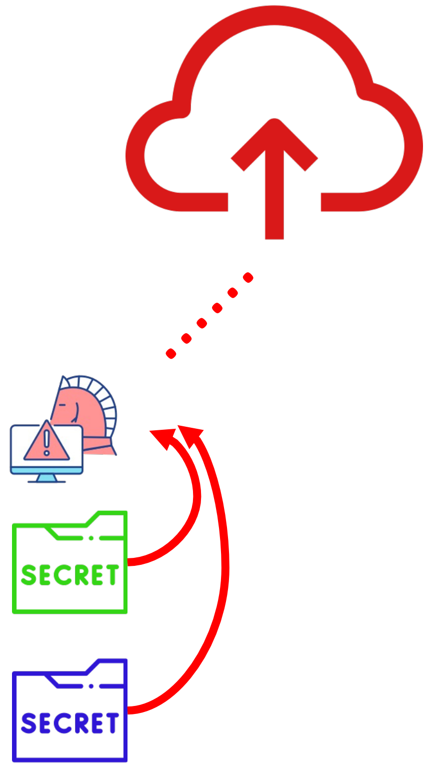
Real-Time Systems and Temporal Isolation

Integration opens the door to security vulnerability

through real-time scheduling



- Flight Controller —— In-House
- Visual Recognition (Mal) —— Vendor A
- Navigation —— Vendor B
- Payload Handling —— Vendor C
- ……



Our Contributions

- **Braided virtual timeline**: a novel language-based abstraction for the formal reasoning of dynamic-priority schedulers
- A fully **verified real-time OS kernel** with budget-enforcing EDF partitions
- A mechanized proof of **temporal isolation between partitions** (no information leakage through real-time scheduling)
- Artifact available: https://flint.cs.yale.edu/publications/compvtl.html

Real-Time Systems and Temporal Isolation

Isolation is key in ensuring the security of modern real-time systems

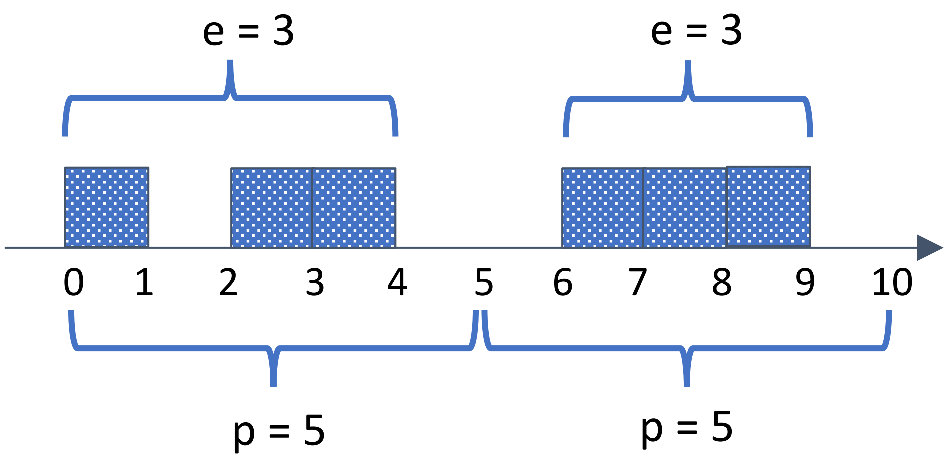
- **Memory resource**: isolation through virtual memory
- **Time resource**: isolation through real-time scheduling
  - Real-time **task**: requires budget (**e**) amount of time within every period (**p**)
  - Real-time **partition**: hosts *a group of tasks* within the partition budget (**C**) and period (**T**)

Background 1: Task-Level Scheduling

Real-time periodic scheduling

- CPU time divided into units of time slots
- Period **p**: a task repeats its execution during each period
- Budget **e**: a task executes for (at most) **e** time slots during each period

A valid schedule of task
$\tau = (e := 3, p := 5)$

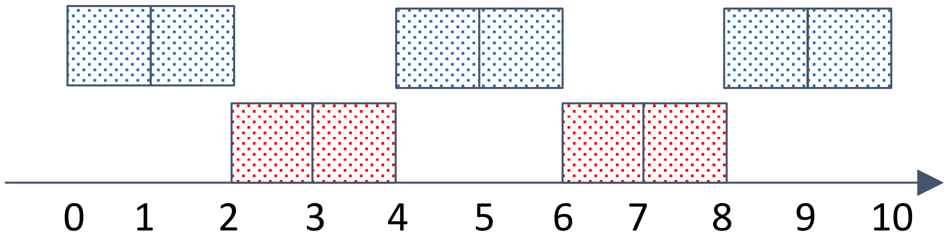


Background 1: Hierarchical Scheduling

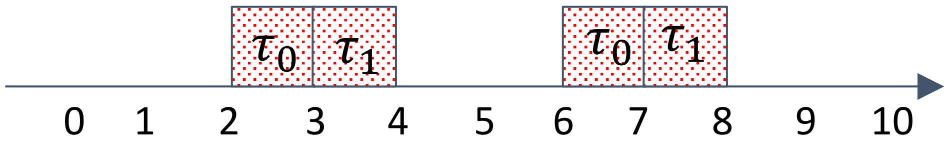
- CPU time is divided among partitions
- A partition’s time slots are further divided among tasks

$$\Pi_0 = (2,5) \begin{cases} \tau_0 = (1,6) \\ \tau_1 = (2,15) \end{cases}$$

$$\Pi_1 = (2,4)$$



Partition-level scheduling:
earliest-deadline-first (EDF)



Task-level scheduling in Π_0

Challenge 1: Information Flow Vulnerability

Though access to global time is prohibited, Π_0 (from Vendor A) learns about Π_1 (from Vendor B) by observing Π_0's own tasks.

$$\Pi_0 = (2,5) \begin{cases} \tau_0 = (1,6) \\ \tau_1 = (2,15) \end{cases}$$

$$\Pi_1 = (2,4)$$

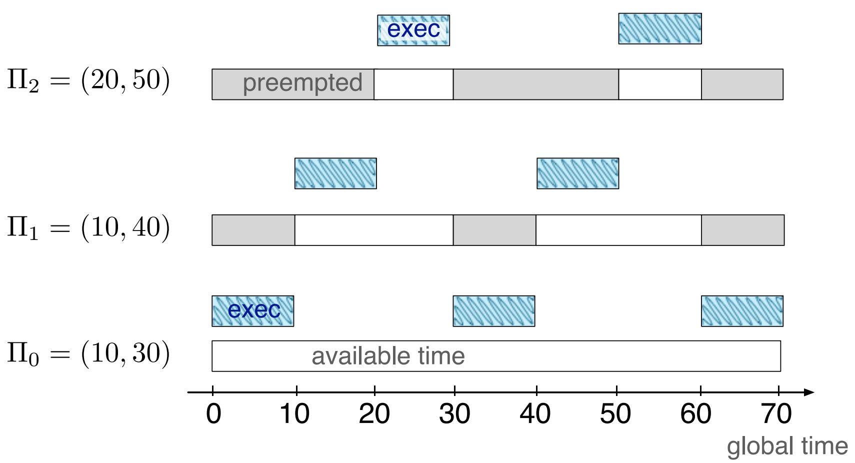
Π_1 is turned on

Π_1 is turned off

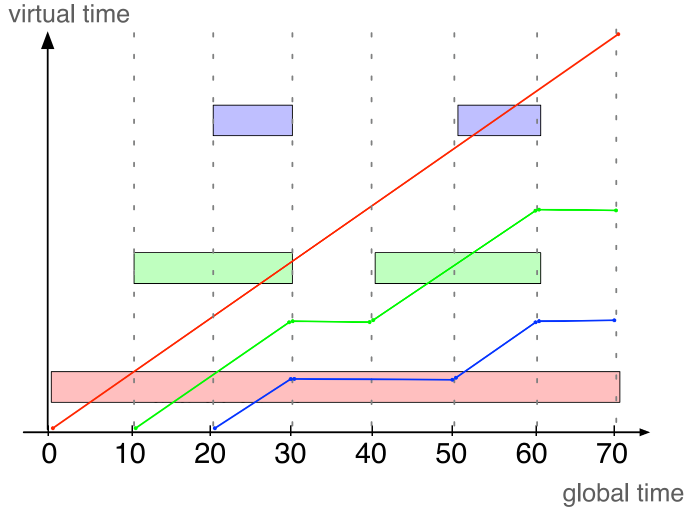
Background 2: Static Virtual Timeline [Liu et al POPL20]

- Characterized by a time map $\pi: Z \rightarrow Z$
- $\pi_i(t)$ means the amount of "available" time to Π_i within global time duration [0, t)
- Can be "statically" computed from highest-priority to lowest-priority

(a). FP schedule & Virtual Timeline



(b). Virtual Time Map



EDF Scheduling is Compositional

N partitions are schedulable under EDF iff.

$$\sum \frac{C_i}{T_i} \leq 1$$

Fixed-priority scheduling (e.g., RMS) is NOT compositional.

N partitions are schedulable iff.

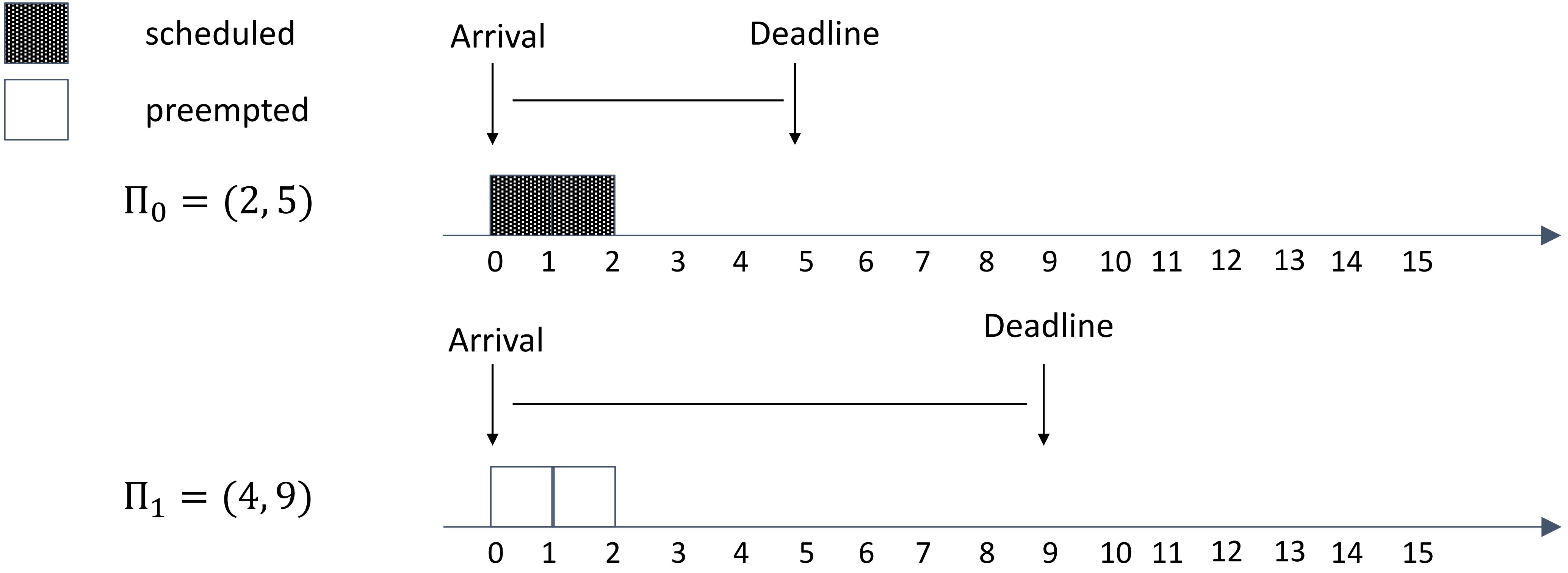
$$\forall i, k, r_i^k \leq T_i$$

where

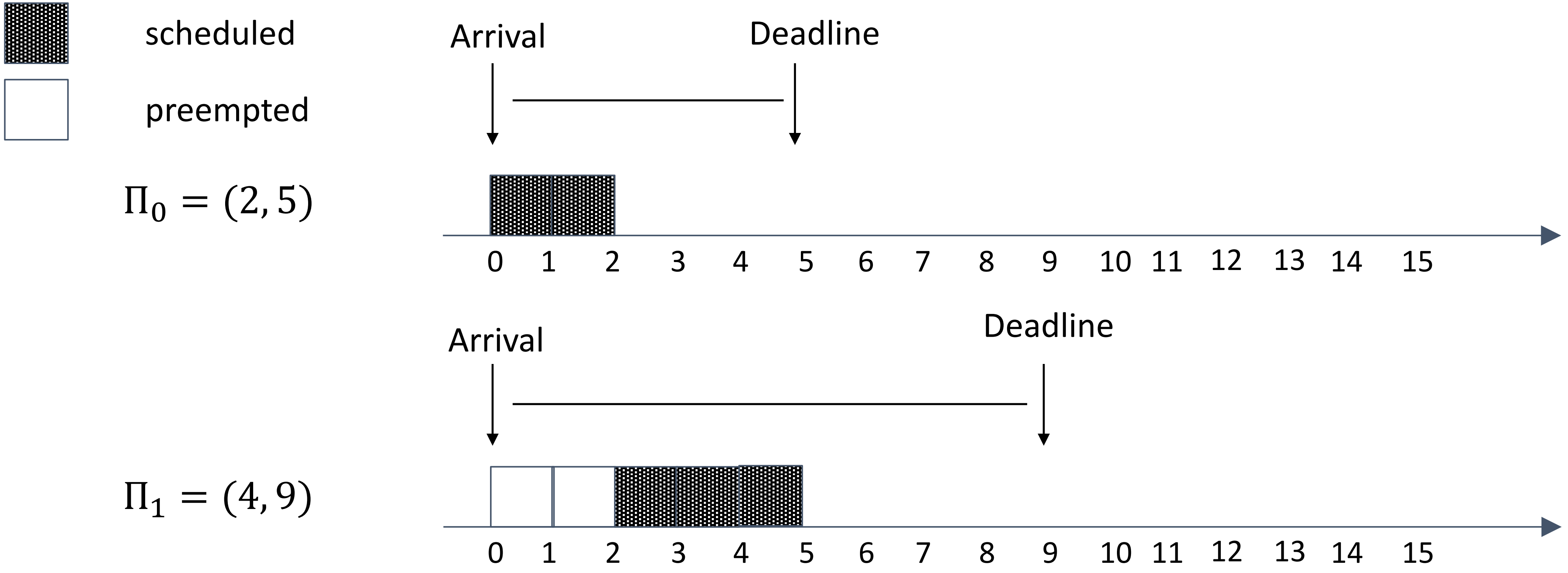
$$r_i^{k+1} = C_i + \sum_{j=1}^{i-1} \lceil \frac{r_i^k}{T_j} \rceil C_j$$

$$r_i^0 = \sum_{j=1}^{i} C_j$$

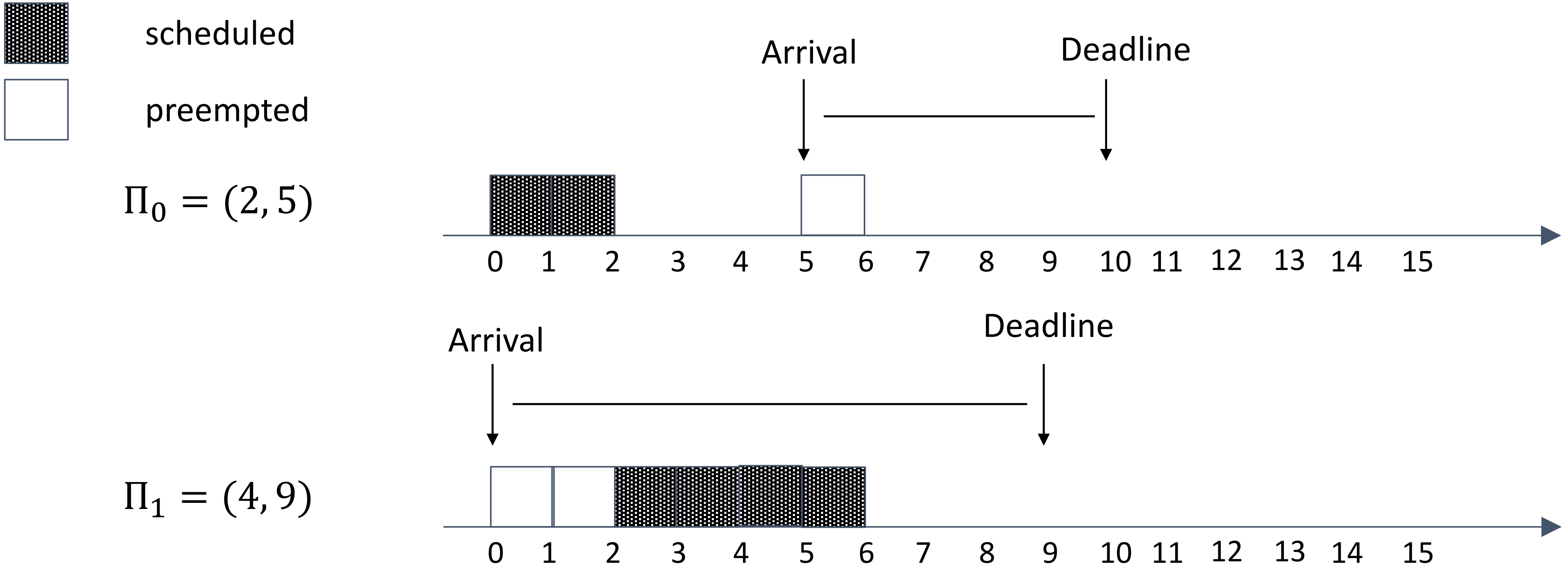
Earliest Deadline First (EDF) scheduling



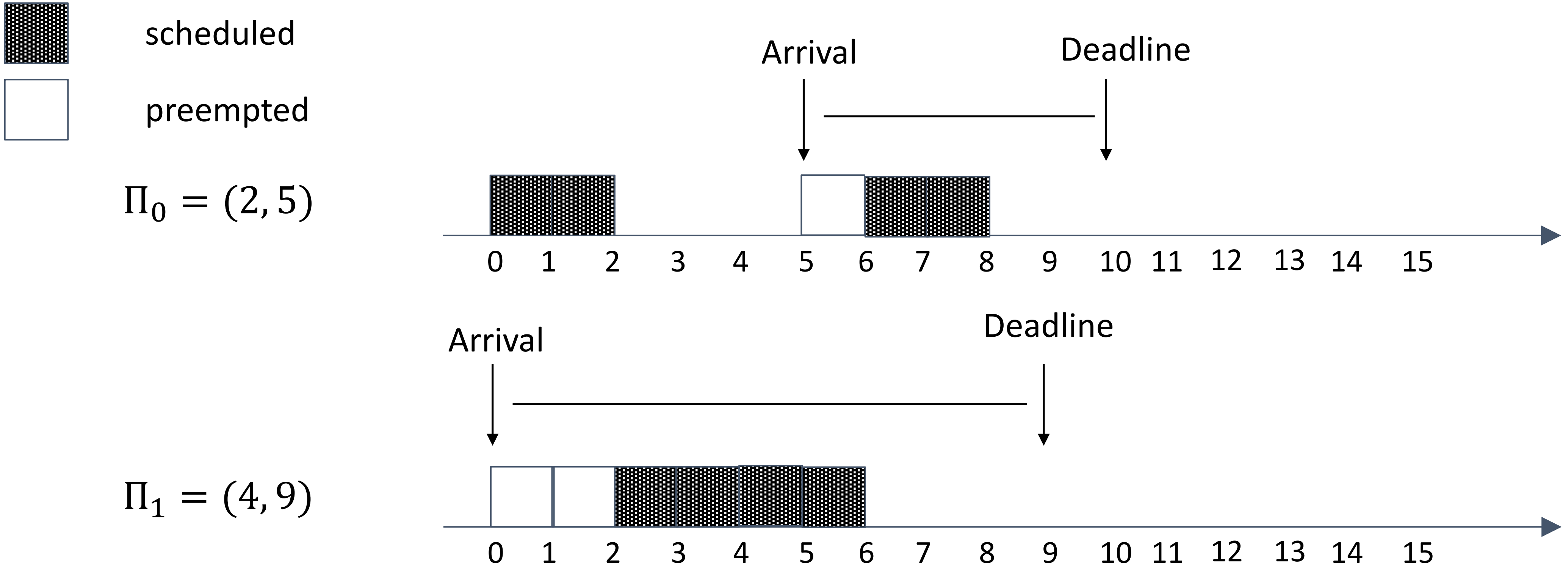
Earliest Deadline First (EDF) scheduling



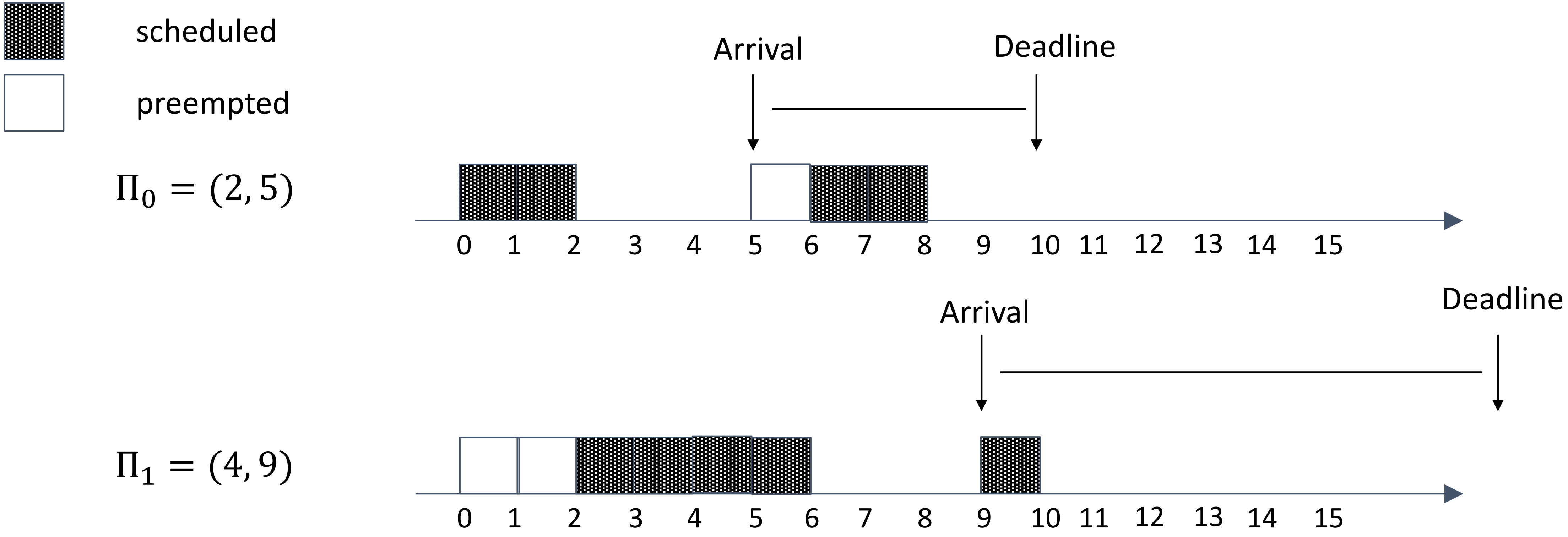
Earliest Deadline First (EDF) scheduling



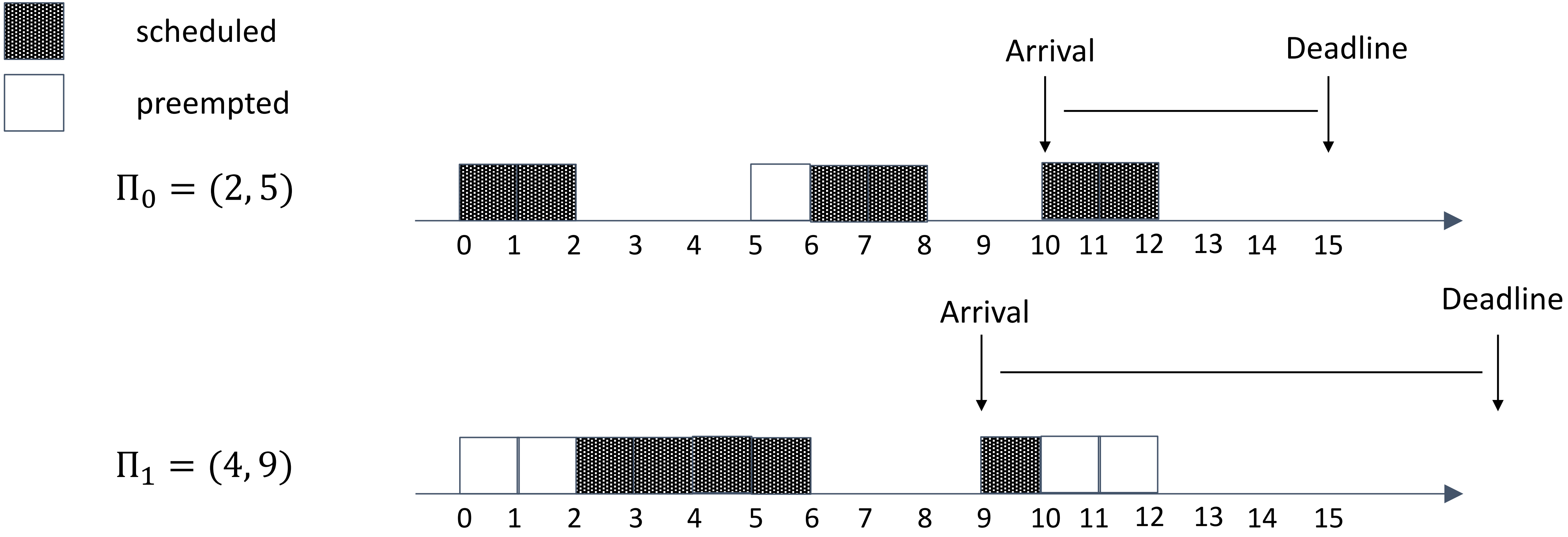
Earliest Deadline First (EDF) scheduling



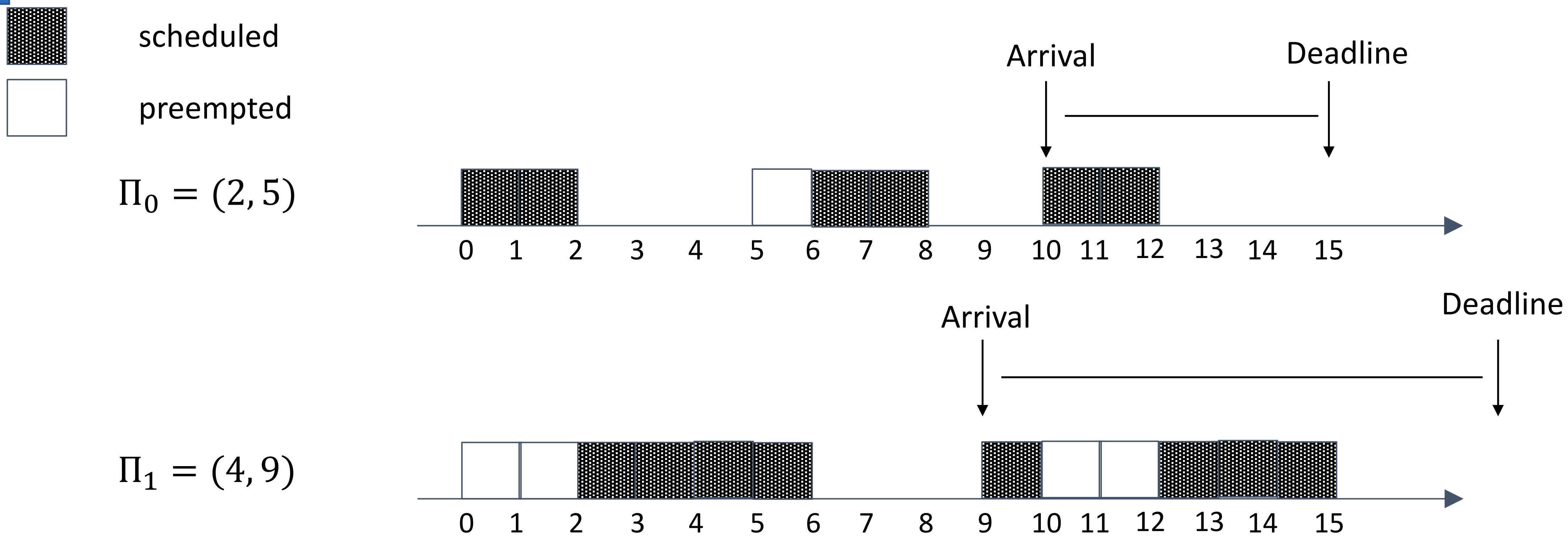
Earliest Deadline First (EDF) scheduling



Earliest Deadline First (EDF) scheduling

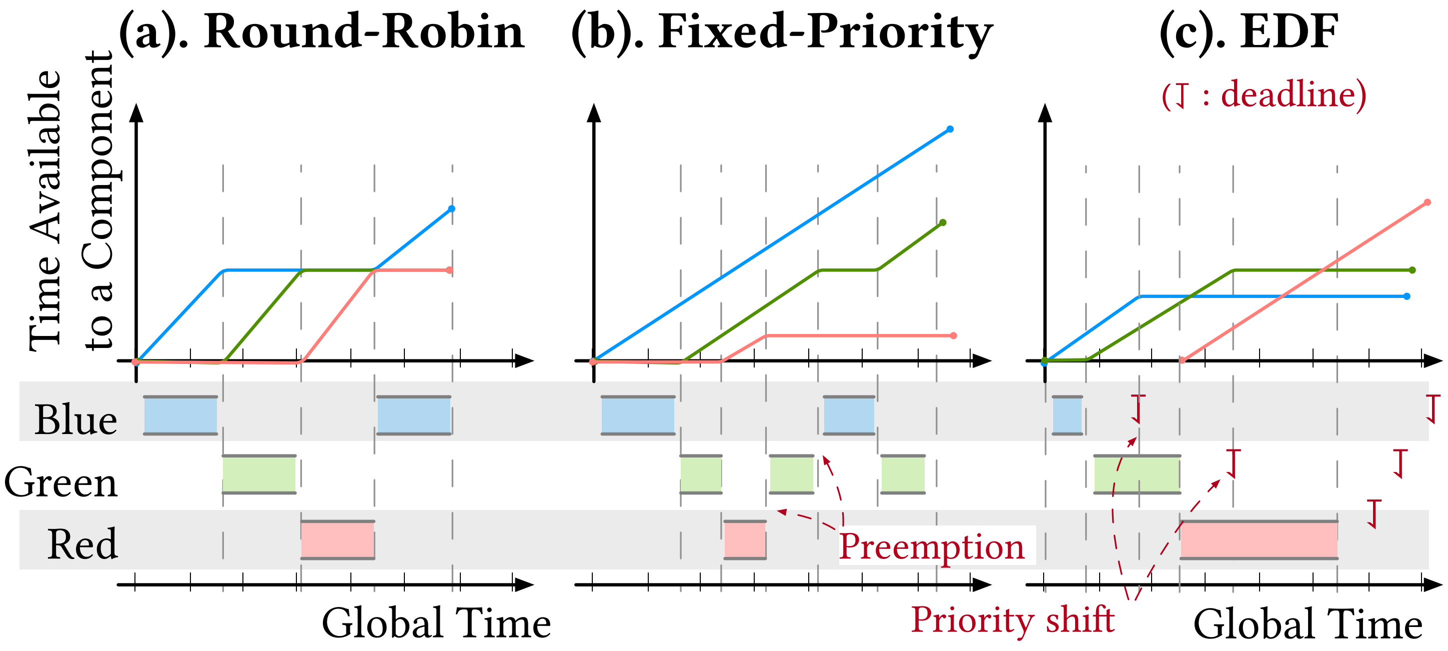


Earliest Deadline First (EDF) scheduling



Challenge 2: Limitations of Static Virtual Timeline

- EDF is compositional and fully utilizes CPU time
- Virtual timeline for EDF scheduling can only be calculated dynamically



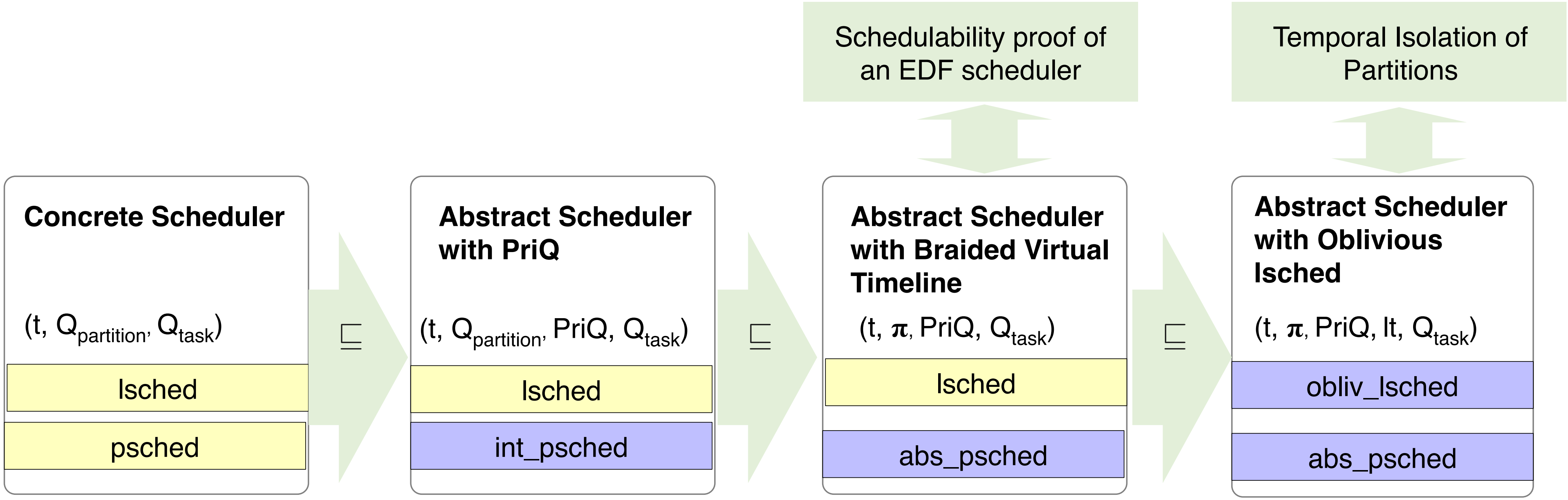
Our Approach: EDF Partitions w/ Bound Tasks

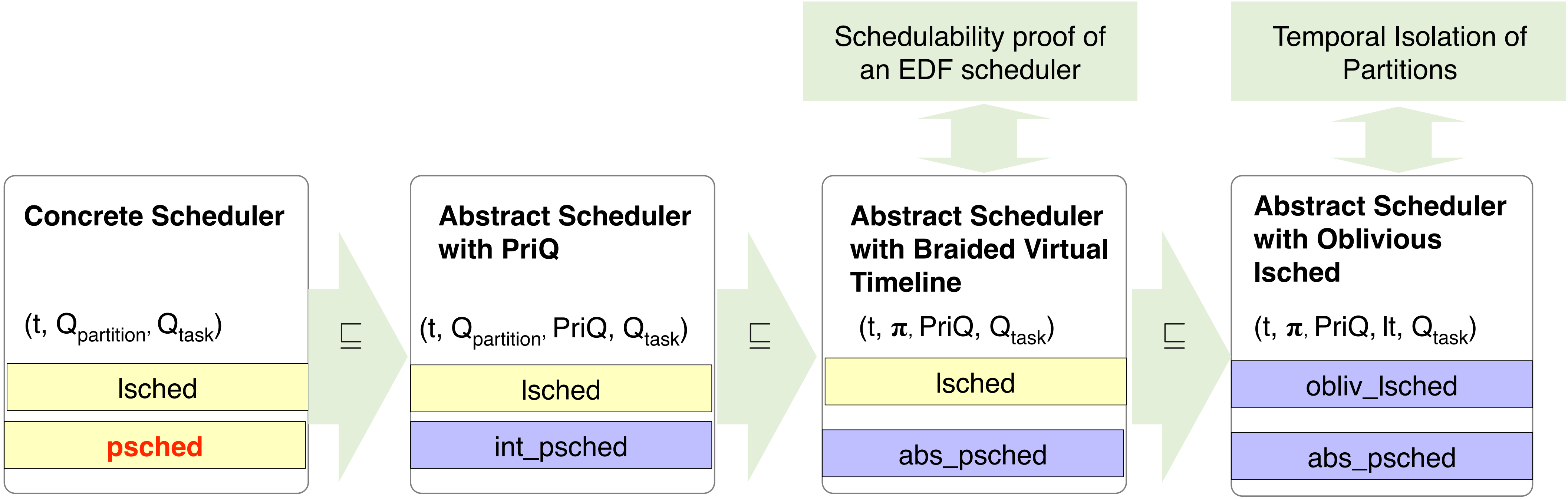
- **EDF partitions**: only considers the utilization ($\frac{C_i}{T_i}$) when adding/removing a partition. Can schedule partitions that are not schedulable under fixed-priority policy.

- **Bound tasks**: for each task in Π_i, the task period is a multiple of T_i

- This is sufficient for ensuring temporal isolation of partitions. BUT we still have to formally prove it.

Verification Overview

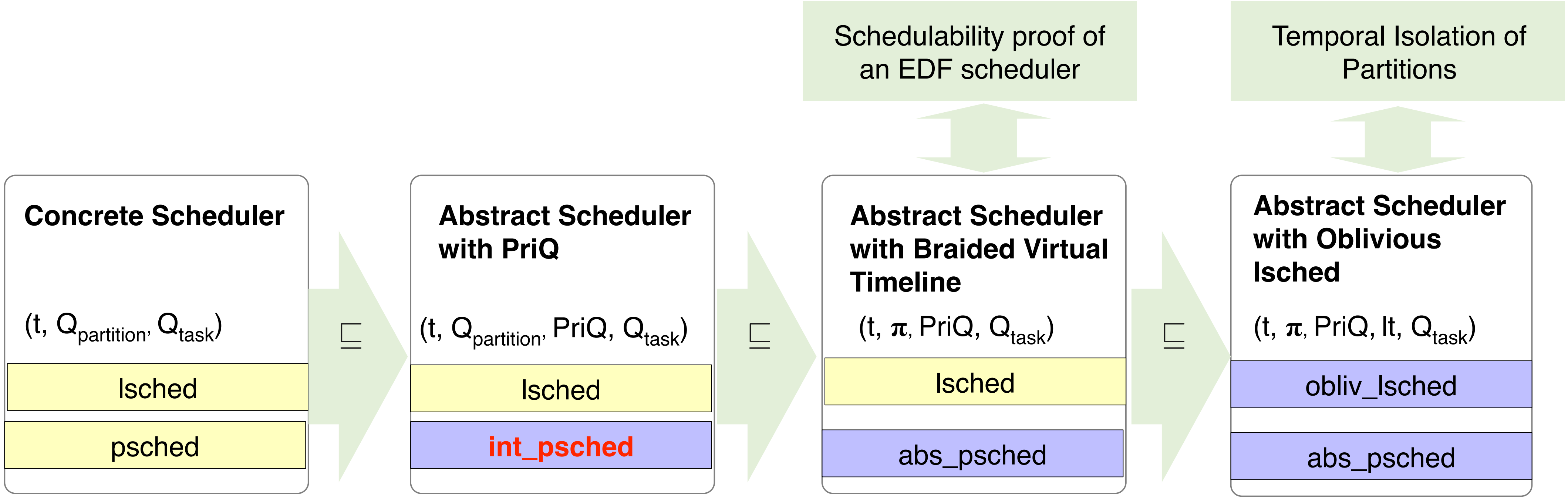



Schedulability proof of an EDF scheduler
Temporal Isolation of Partitions
Concrete Scheduler
(t, Q_partition, Q_task)
Isched
psched
⊑
Abstract Scheduler with PriQ
(t, Q_partition, PriQ, Q_task)
Isched
int_psched
⊑
Abstract Scheduler with Braided Virtual Timeline
(t, π, PriQ, Q_task)
Isched
abs_psched
⊑
Abstract Scheduler with Oblivious Isched
(t, π, PriQ, It, Q_task)
obliv_Isched
abs_psched

Partition-Level EDF Scheduler

```
int psched(){
  t++;
  // refill partition budgets
  for(int i = 0; i < N; i++){
    if (t % Ti == 0){
      Qpartition[i] = Ci;
    }
  }

  // scheduling
  int pid = N;
  int min_ddl = INT_MAX;
  for(int i = 0; i < N; i++){
    if (Qpartition[i] > 0){
      int ddl = t / Ti * Ti + Ti;
      if (pid == N || ddl < min_ddl){
        pid = i;
        min_ddl = ddl;
      }
    }
  }
  if (pid < N){
    Qpartition[pid]--;
  }
  return pid;
}
```

t	The current global time
N	The number of partitions
T_i	Pre-specified period of partition Π_i
C_i	Pre-specified budget of partition Π_i
$Q_{partition}[i]$	The remaining budget of partition Π_i

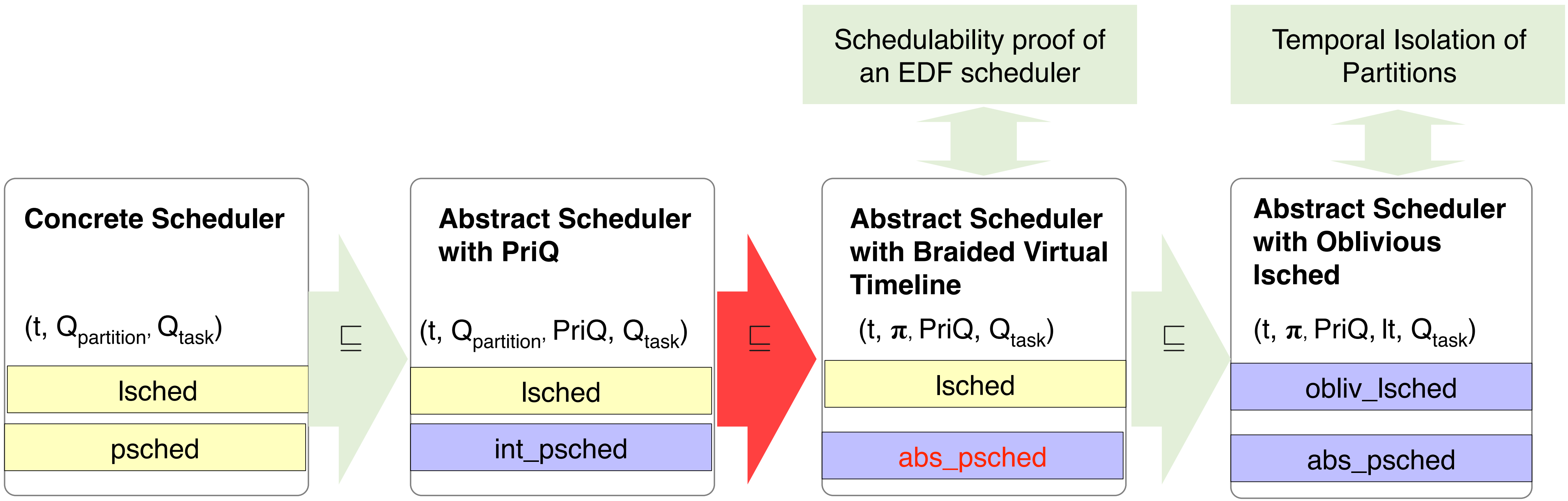


Schedulability proof of an EDF scheduler
Temporal Isolation of Partitions
Concrete Scheduler
(t, $Q_{partition}$, Q_{task})
lsched
psched
$\sqsubseteq$
Abstract Scheduler with PriQ
(t, $Q_{partition}$, PriQ, Q_{task})
lsched
int_psched
$\sqsubseteq$
Abstract Scheduler with Braided Virtual Timeline
(t, π, PriQ, Q_{task})
lsched
abs_psched
$\sqsubseteq$
Abstract Scheduler with Oblivious lsched
(t, π, PriQ, lt, Q_{task})
obliv_lsched
abs_psched

Abstraction: Intermediate Scheduler

```
int int_psched(){
  t++;
  // refill partition budgets
  for(int i = 0; i < N; i++){
    if (t % T_i == 0){
      Q_partition[i] = C_i;
    }
  }

  // scheduling
  int pid = N;
  for(int p = 0; p < N; p++){
    int i = PriQ_t(p);
    if (Q_partition[i] > 0){
      pid = i;
      break;
    }
  }
  if (pid < N){
    Q_partition[pid]--;
  }
  return pid;
}
```

$PriQ_t$ (p)	Mapping from priority level p to partition ID at time instant t

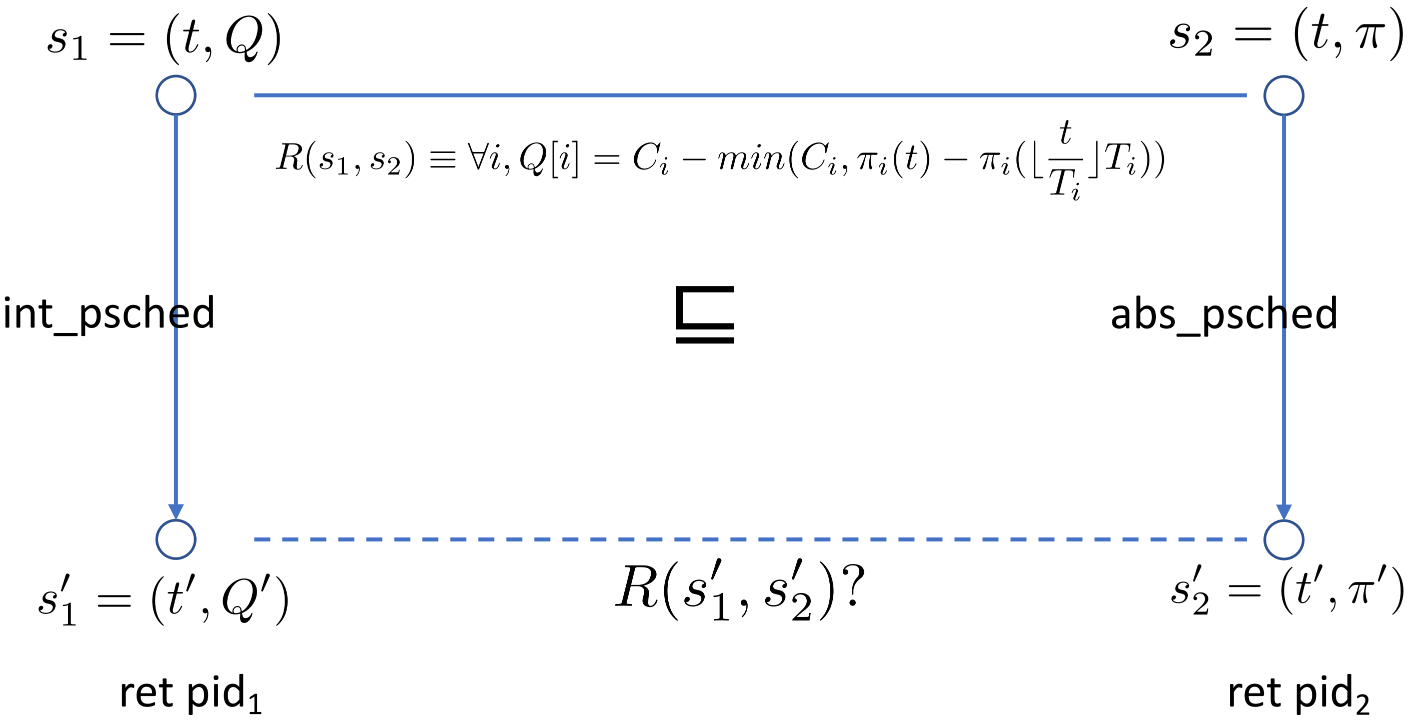

Schedulability proof of an EDF scheduler
Temporal Isolation of Partitions
Concrete Scheduler
(t, $Q_{partition}$, Q_{task})
lsched
psched
$\sqsubseteq$
Abstract Scheduler with PriQ
(t, $Q_{partition}$, PriQ, Q_{task})
lsched
int_psched
$\sqsubseteq$
Abstract Scheduler with Braided Virtual Timeline
(t, π, PriQ, Q_{task})
lsched
abs_psched
$\sqsubseteq$
Abstract Scheduler with Oblivious lsched
(t, π, PriQ, lt, Q_{task})
obliv_lsched
abs_psched

Further Abstraction: Braided Virtual Timeline

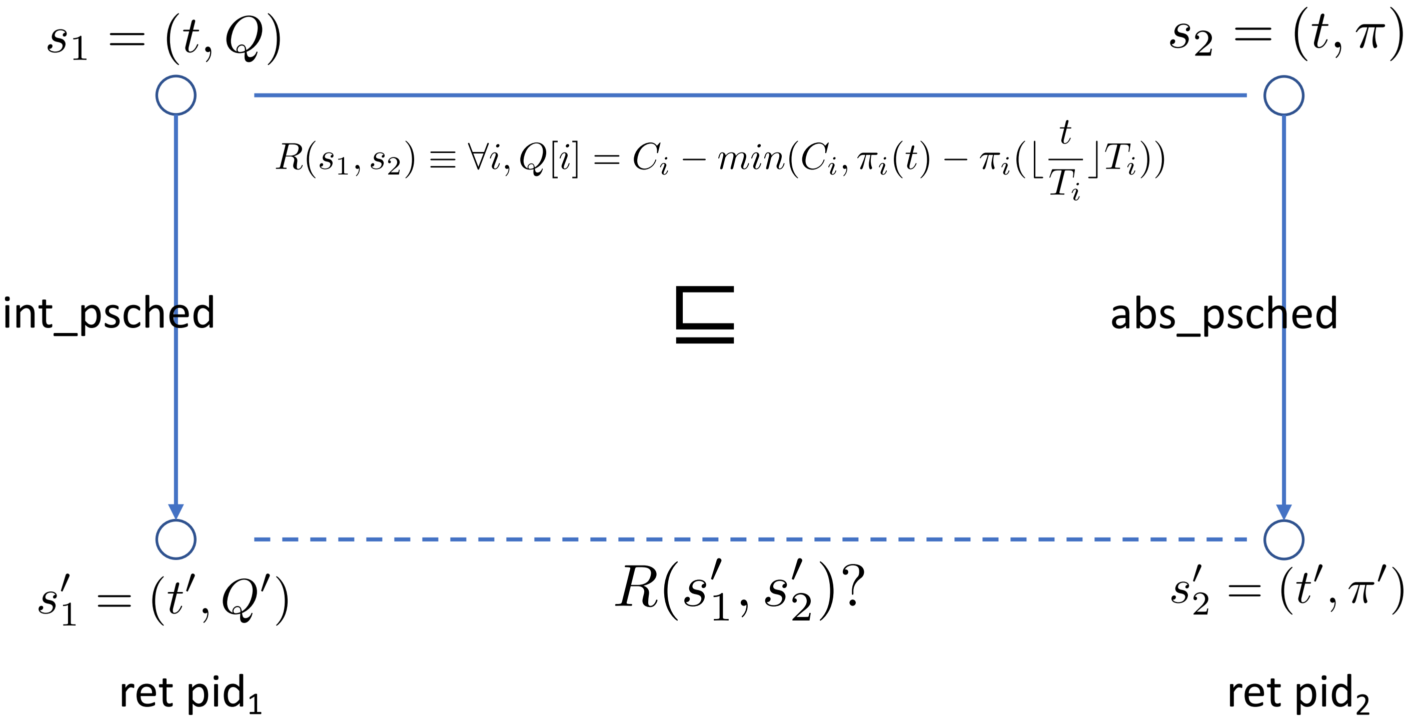
```
int abs_psched(){
  t++;
  int pid = N;
  for (p = 0; p < N; p++) {
    int i = PriQt(p);
    if (pid == N){
      πi = λx.(x ≥ t)? πi(t − 1) + 1: πi(x);
      if (πi(t) − πi(⌊t/Ti⌋ Ti) < Ci){
        pid = i;
      }
    }else{
      // Interference incurred
      Ii(⌊t/Ti⌋)++;
    }
  }
  return pid;
}
```

π_i	Virtual Timeline for partition Π_i: mapping from global time to accumulative available time
$I_i(k)$	The amount of temporal interference incurred on Π_i in the k-th period

Contextual Refinement Proof



Contextual Refinement Proof

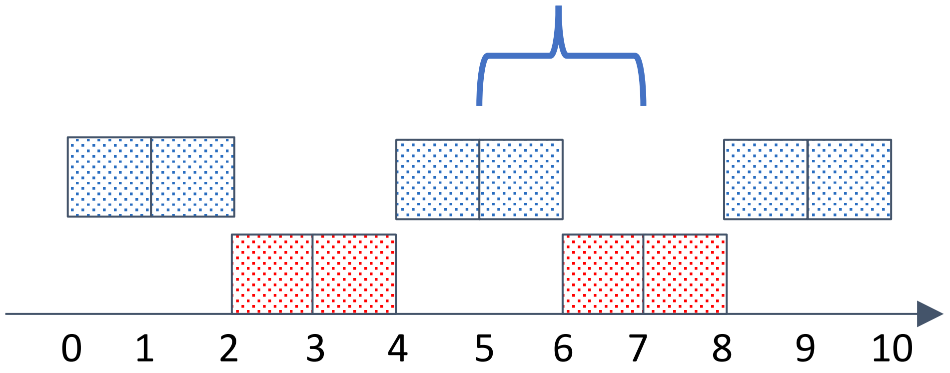


$\Pi_0 = (2, 5)$

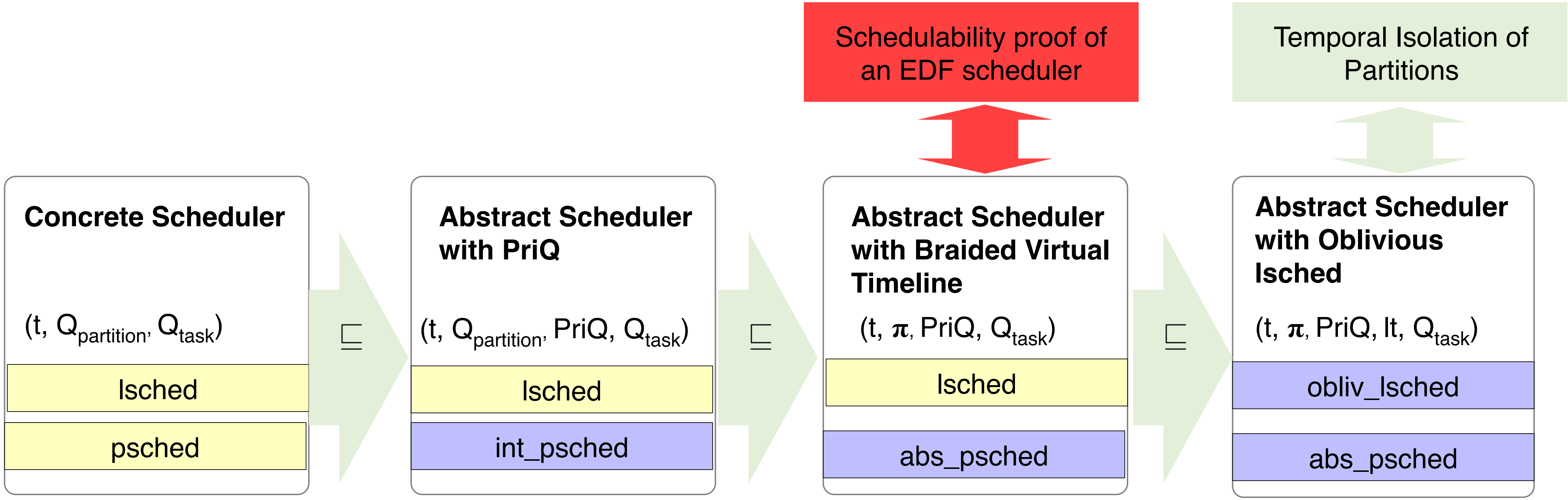
$\Pi_1 = (2, 4)$

t = 7
i = 0

$Q[0] = 2 - min(2, \pi_0(7) - \pi_0(5))$



Partition-level scheduling:
earliest-deadline-first (EDF)


Schedulability proof of an EDF scheduler
Temporal Isolation of Partitions
Concrete Scheduler
(t, Q_partition, Q_task)
Isched
psched
⊑
Abstract Scheduler with PriQ
(t, Q_partition, PriQ, Q_task)
Isched
int_psched
⊑
Abstract Scheduler with Braided Virtual Timeline
(t, π, PriQ, Q_task)
Isched
abs_psched
⊑
Abstract Scheduler with Oblivious Isched
(t, π, PriQ, It, Q_task)
obliv_Isched
abs_psched

Schedulability Proof of EDF

Proof goal: for each partition Π_i, the amount of available time within each period is greater than or equal to its budget.

$$\forall k \geq 0, \pi_i((k+1)T_i) - \pi_i(kT_i) \geq C_i$$

- Abstract enough to facilitate the proof
- Proof carries down to the scheduler implementation

Schedulability Proof of EDF

Break down the proof obligation into smaller steps

[Wilding, CAV'98]

Partition	Budget	Period	Util
Π_0	10	40	25%
Π_1	10	30	33%
Π_2	20	50	40%

LCM(40, 30, 50) = 600

GCD(40, 30, 50) = 10

LCM / GCD = 60

Lift each (C_i, T_i) by LCM / T_i

Then enlarge a partition by T_i / GCD one at a time

Then shrink all by LCM/GCD in a single step

{(150, 600), (200, 600), (240, 600)} is schedulable

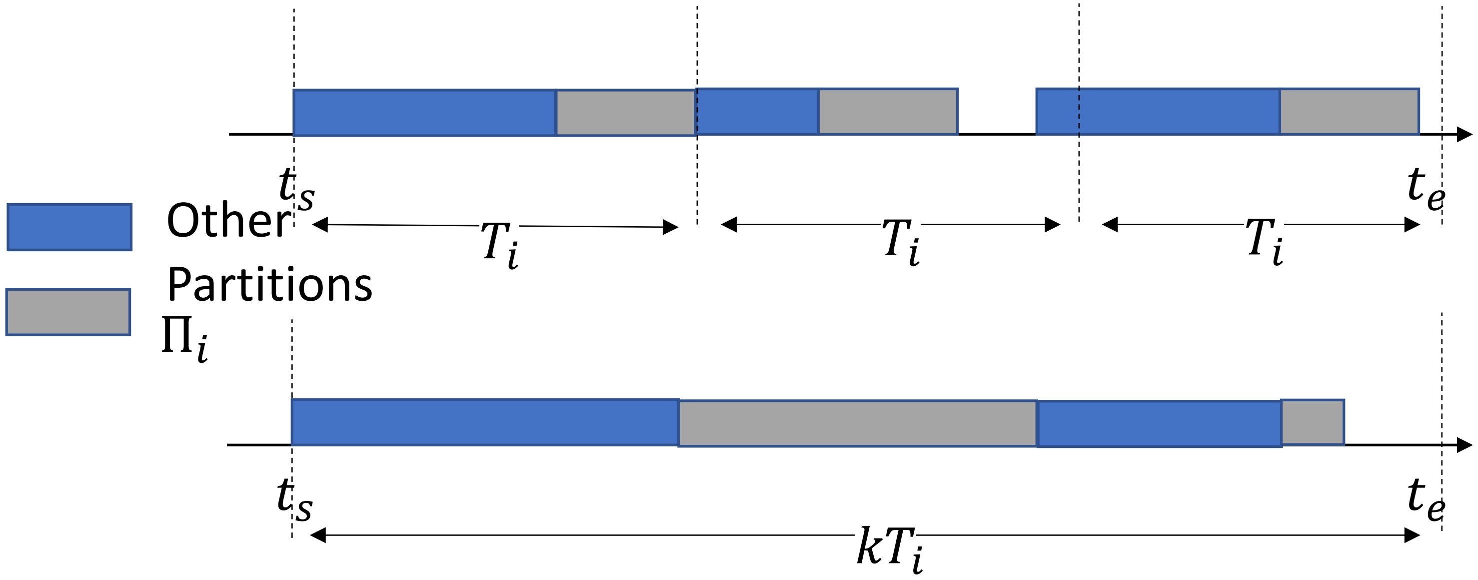
=> {(150, 600), (600, 1800), (240, 600)} is schedulable

=> {(600, 2400), (600, 1800), (240, 600)} is schedulable

=> {(600, 2400), (600, 1800), (1200, 3000)} is schedulable

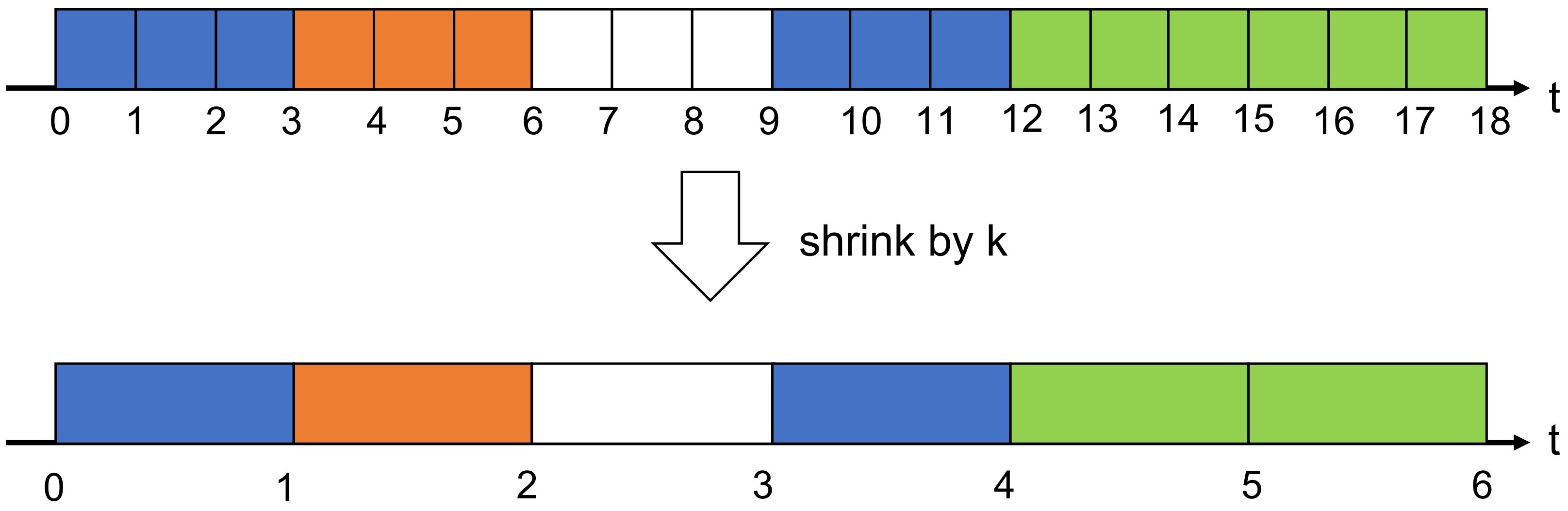
=> {(10, 40), (10, 30), (20, 50)} is schedulable

Schedulability Proof: Enlarging One Partition

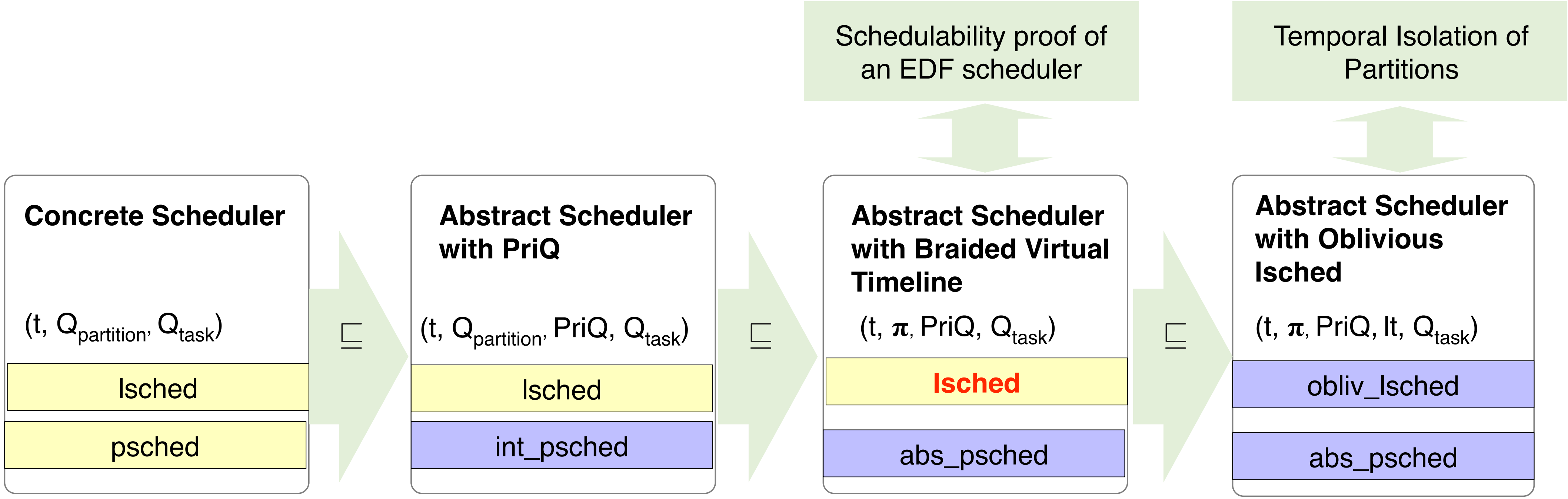


- The total interference from other partitions does not increase
- Thus, Π_i is still schedulable after enlarged by k

Schedulability Proof: Shrinking All Partitions



- If {(kC, kT), …} is schedulable, {(C, T), …} is also schedulable

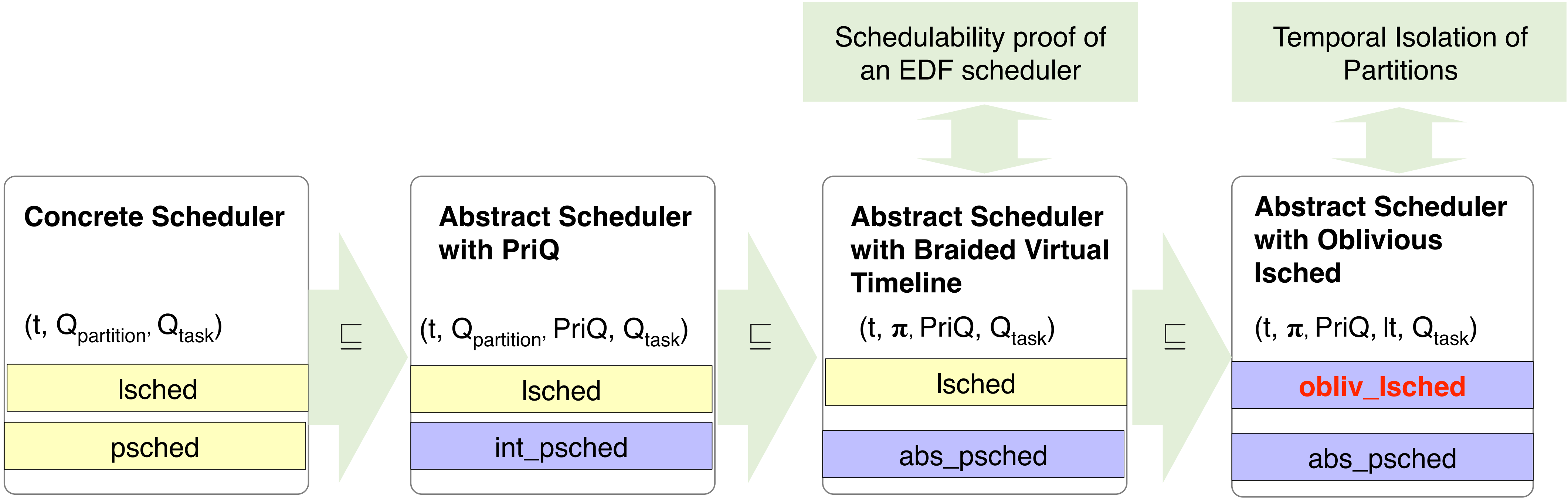


Schedulability proof of an EDF scheduler
Temporal Isolation of Partitions
Concrete Scheduler
(t, Q_partition, Q_task)
lsched
psched
⊑
Abstract Scheduler with PriQ
(t, Q_partition, PriQ, Q_task)
lsched
int_psched
⊑
Abstract Scheduler with Braided Virtual Timeline
(t, π, PriQ, Q_task)
lsched
abs_psched
⊑
Abstract Scheduler with Oblivious lsched
(t, π, PriQ, lt, Q_task)
obliv_lsched
abs_psched

Local Scheduler for Tasks

```
int lsched(){
  for(int j = 0; j < M; j++){
    if (t % pj == 0){
      Qtask[j] = ej;
    }
  }

  int tid = M;
  for(int j = 0; j < M; j++){
    if(Qtask[j] > 0){
      tid = j;
      Qtask[j]--;
      break;
    }
  }

  return tid;
}
```

t	The current global time
M	The number of tasks
p_j	Pre-specified period of task τ_j
e_j	Pre-specified budget of task τ_j
$Q_{task}[j]$	The remaining budget of task τ_j



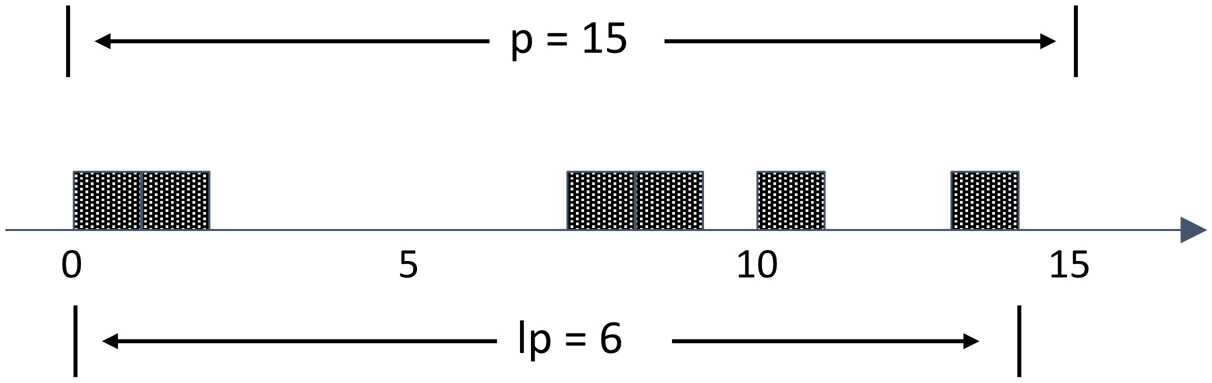
Schedulability proof of an EDF scheduler
Temporal Isolation of Partitions
Concrete Scheduler
(t, Q_partition, Q_task)
lsched
psched
⊑
Abstract Scheduler with PriQ
(t, Q_partition, PriQ, Q_task)
lsched
int_psched
⊑
Abstract Scheduler with Braided Virtual Timeline
(t, π, PriQ, Q_task)
lsched
abs_psched
⊑
Abstract Scheduler with Oblivious lsched
(t, π, PriQ, lt, Q_task)
obliv_lsched
abs_psched

Abstraction: Oblivious Local Scheduler

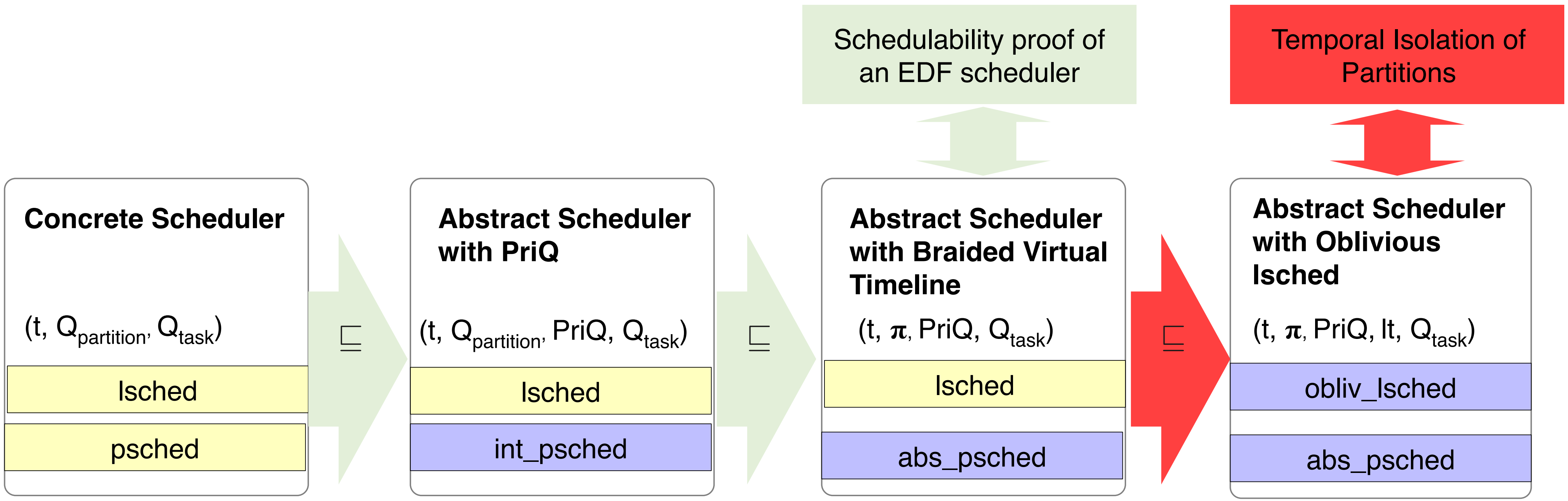
```
int obliv_lsched(){
  for(int j = 0; j < M; j++){
    if (lt % lp_j == 0){
      Q_task[j] = e_j;
    }
  }

  int tid = M;
  for(int j = 0; j < M; j++){
    if(Q_task[j] > 0){
      tid = j;
      Q_task[j]--;
      break;
    }
  }

  return tid;
}
```

lt	The local time experienced by the enclosing partition
lp_j	Equals p_j multiplied by the enclosing partition's utilization (budget / period)

$$\Pi = (2, 5), \tau = (2, 15)$$

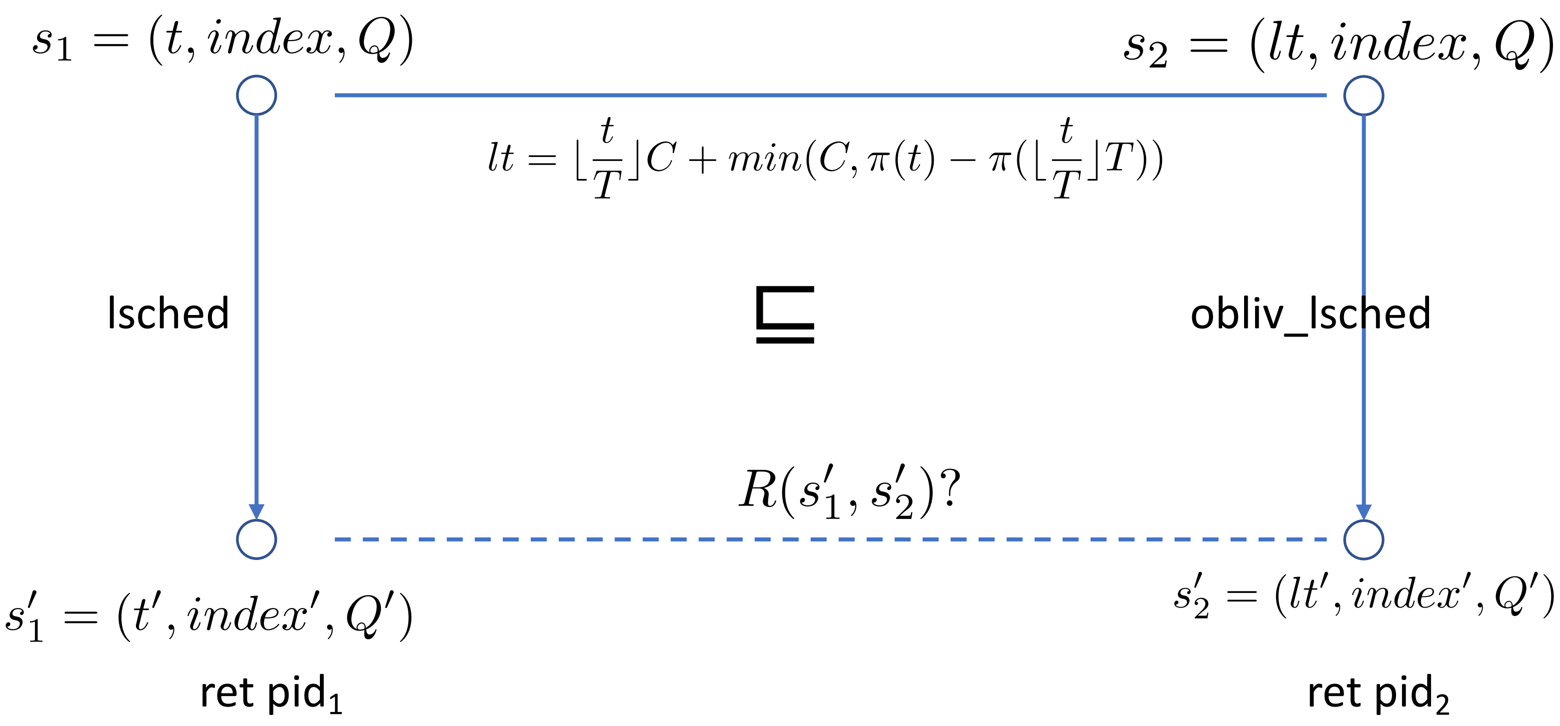


The portion of p that is visible to Π

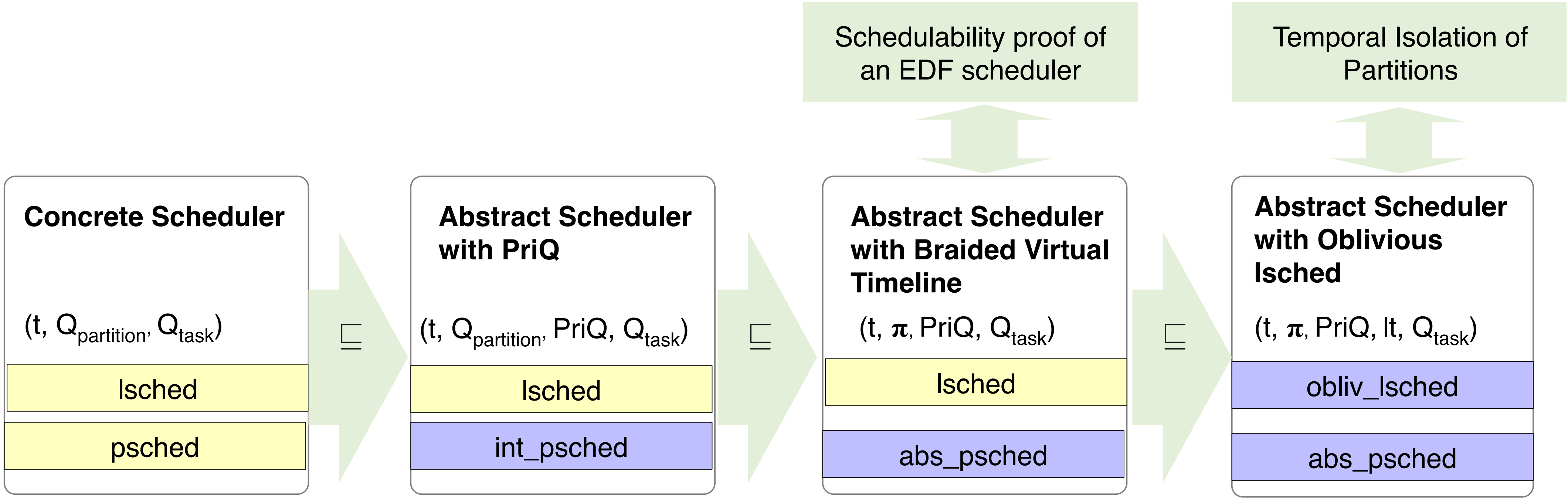


Schedulability proof of an EDF scheduler
Temporal Isolation of Partitions
Concrete Scheduler
(t, $Q_{partition}$, Q_{task})
Isched
psched
$\sqsubseteq$
Abstract Scheduler with PriQ
(t, $Q_{partition}$, PriQ, Q_{task})
Isched
int_psched
$\sqsubseteq$
Abstract Scheduler with Braided Virtual Timeline
(t, π, PriQ, Q_{task})
Isched
abs_psched
$\sqsubseteq$
Abstract Scheduler with Oblivious Isched
(t, π, PriQ, It, Q_{task})
obliv_Isched
abs_psched

Contextual Refinement Proof



Verification Overview

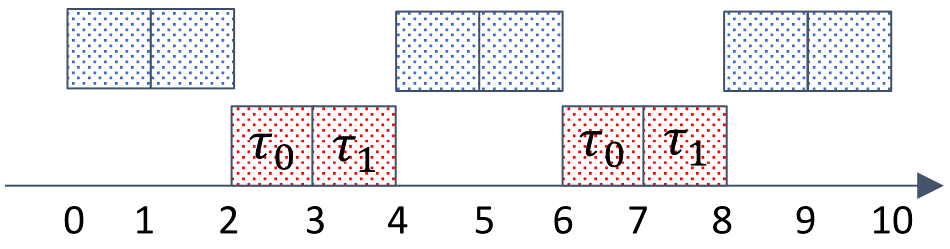


Challenge 1: Information Flow Vulnerability

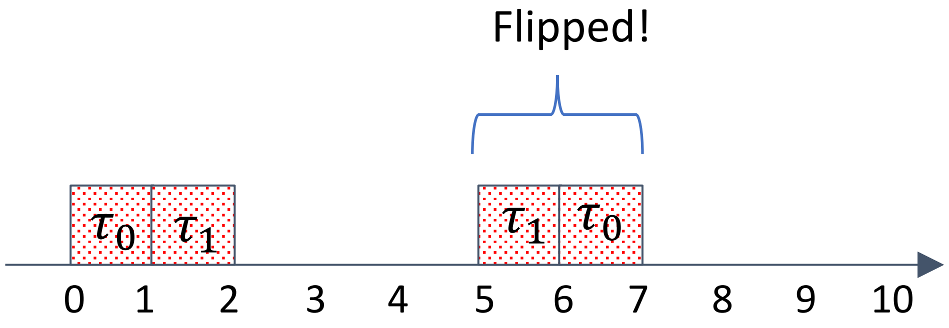
Though access to global time is prohibited, Π_0 (from Vendor A) learns about Π_1 (from Vendor B) by observing Π_0's own tasks.

$$\Pi_0 = (2,5) \begin{cases} \tau_0 = (1,6) \\ \tau_1 = (2,15) \end{cases}$$

$$\Pi_1 = (2,4)$$

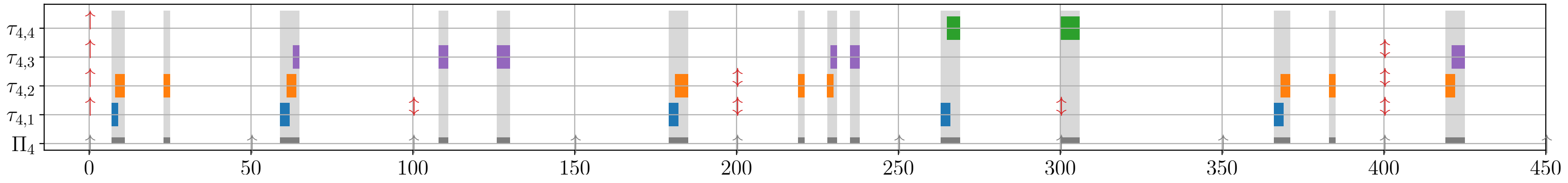


Π_1 is turned on

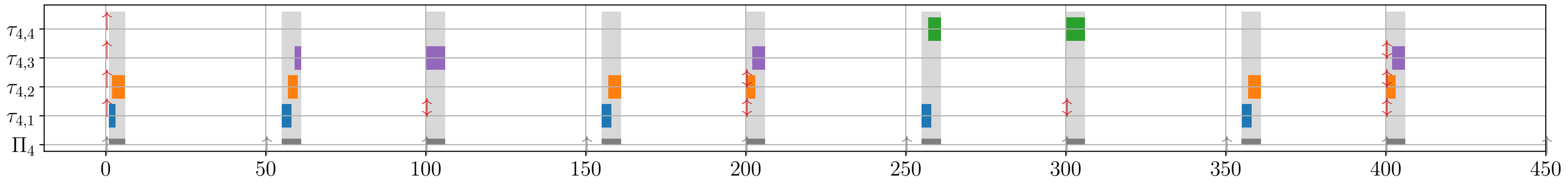


Π_1 is turned off

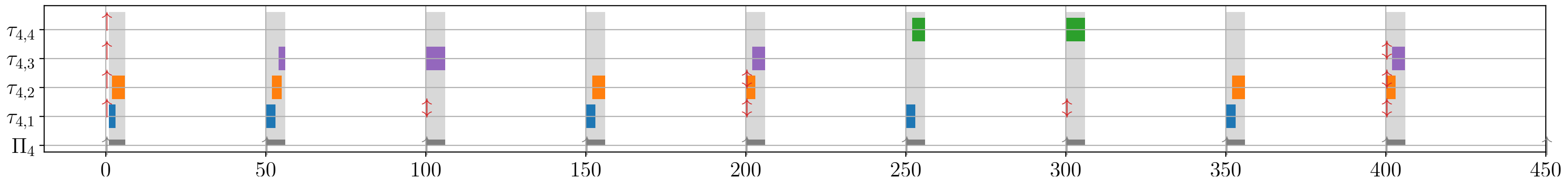
Evaluation: Temporal Isolation



(c) w/3 other partitions



(b) w/1 other partition



(a) solo partition

Scheduling traces of partition $\{\tau_{4,1}, \ldots, \tau_{4,4}\}$ are equivalent when run with different other partitions

Proof Efforts

Item	LOC in Coq
Formalization of the dynamic virtual time map and related lemmas	2,102
The schedulability proof on top of the braided virtual timelines	16,170
Functional correctness proof for the partition-level EDF scheduler's C code	2,876
Connecting the schedulability proof with the partition-level EDF scheduler	4,876
Functional correctness proof for the local task scheduler's C code	2,963
Contextual refinement proof between the local task scheduler and its oblivious abstraction	7,594
Grand Total	36,581

Conclusions

- **Braided virtual timeline**: a novel language-based abstraction for the formal reasoning of dynamic-priority schedulers
- A fully **verified real-time OS kernel** with budget-enforcing EDF partitions
- A mechanized proof of **temporal isolation between partitions** (no information leakage through real-time scheduling)
- Artifact available: https://flint.cs.yale.edu/publications/compvtl.html

Thank you!