Multiprocessor Global Scheduling on Frame-Based DVFS Systems

Vandy BERTEN

Université Libre de Bruxelles, Belgium
Fonds National de la Recherche Scientifique

ULB

Outline

- Motivations & Context
- Formal Model
- Scheduling Algorithms
- Some Simulations
- Conclusions

Motivations & Context

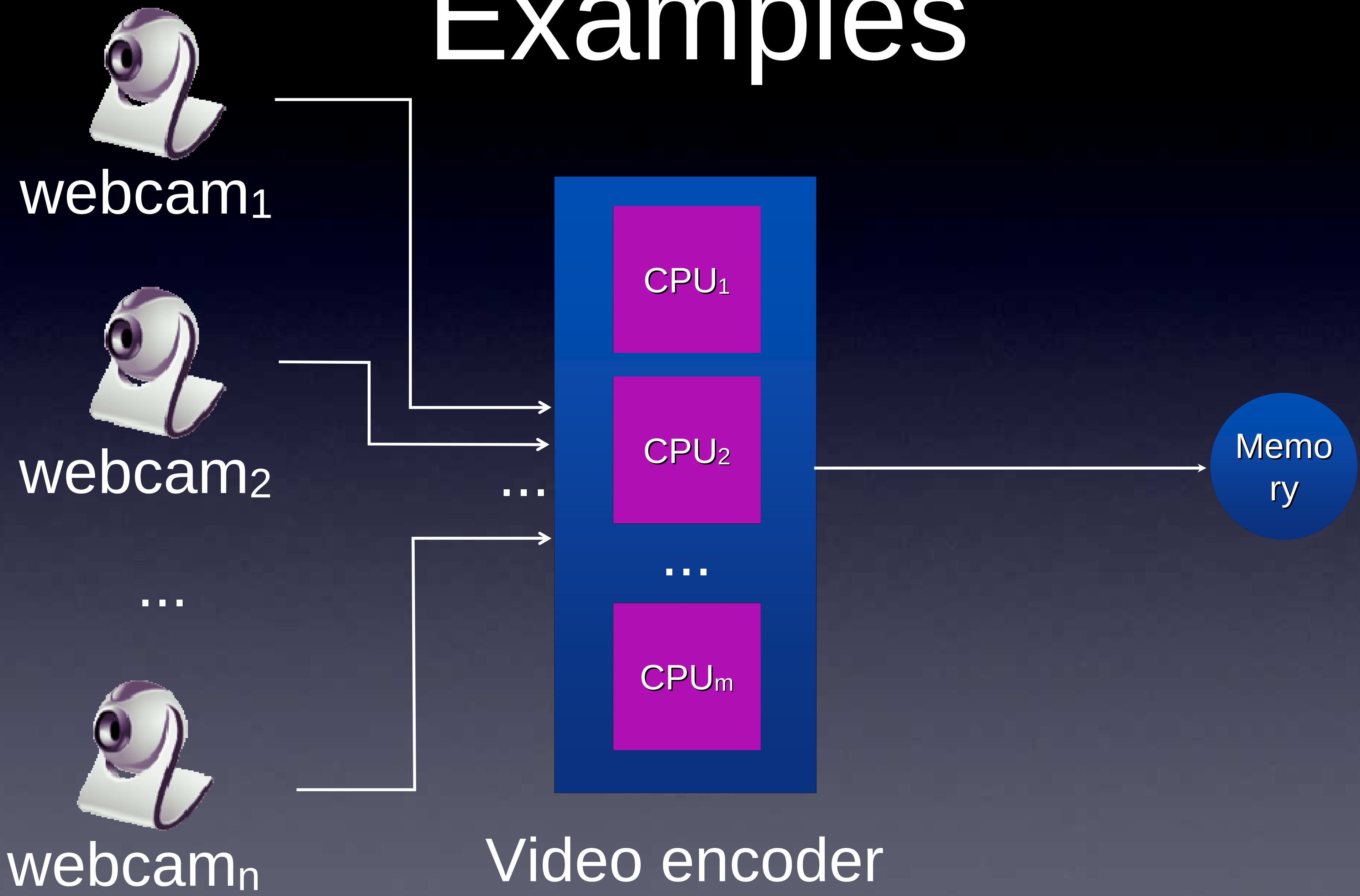
Motivations

- Many embedded, multimedia, communication, ... devices have common characteristics:
  - They have real-time constraints → RT Scheduling
  - They are battery powered → Energy-efficiency
  - Execution lengths are not known in advance → Stochastic models
  - Contain already 2 or 4 CPUs, and very soon several hundreds → Multiprocessor systems

Motivations (con't)

- This talk is about Real-time scheduling algorithms for Energy-efficient systems with Stochastic tasks on Multiprocessor Platforms
- We are interested in a specific task model: Frame-based systems (all tasks share the same period/deadline)

Examples


webcam1
webcam2
...
webcamn
...
CPU1
CPU2
...
CPUm
Video encoder
Memory

Formal Model

Frame-Based System

- We consider a n tasks $\{\tau_1, \tau_2, \ldots, \tau_n\}$
- Frame-Based: all tasks share the same deadline/period ($T_i = D_i = D$)
- Every multiple of D, a bunch of n jobs arrives ...
- ... and should be finished before the next arrival
- The task order is given (or chosen beforehand)

Stochastic Models

- The execution length of a job is not known before the end
- We know the execution length distribution of each task ...
- ... and the Worst Case Execution number of Cycles (WCEC) : w_i

Energy Efficiency

- DVFS platforms (Dynamic Voltage & Frequency Scaling) allow to change the frequency on-the-fly
- DVFS scheduling algorithms aim at selecting the right frequency in order to:
  - meet deadlines
  - minimize energy consumption

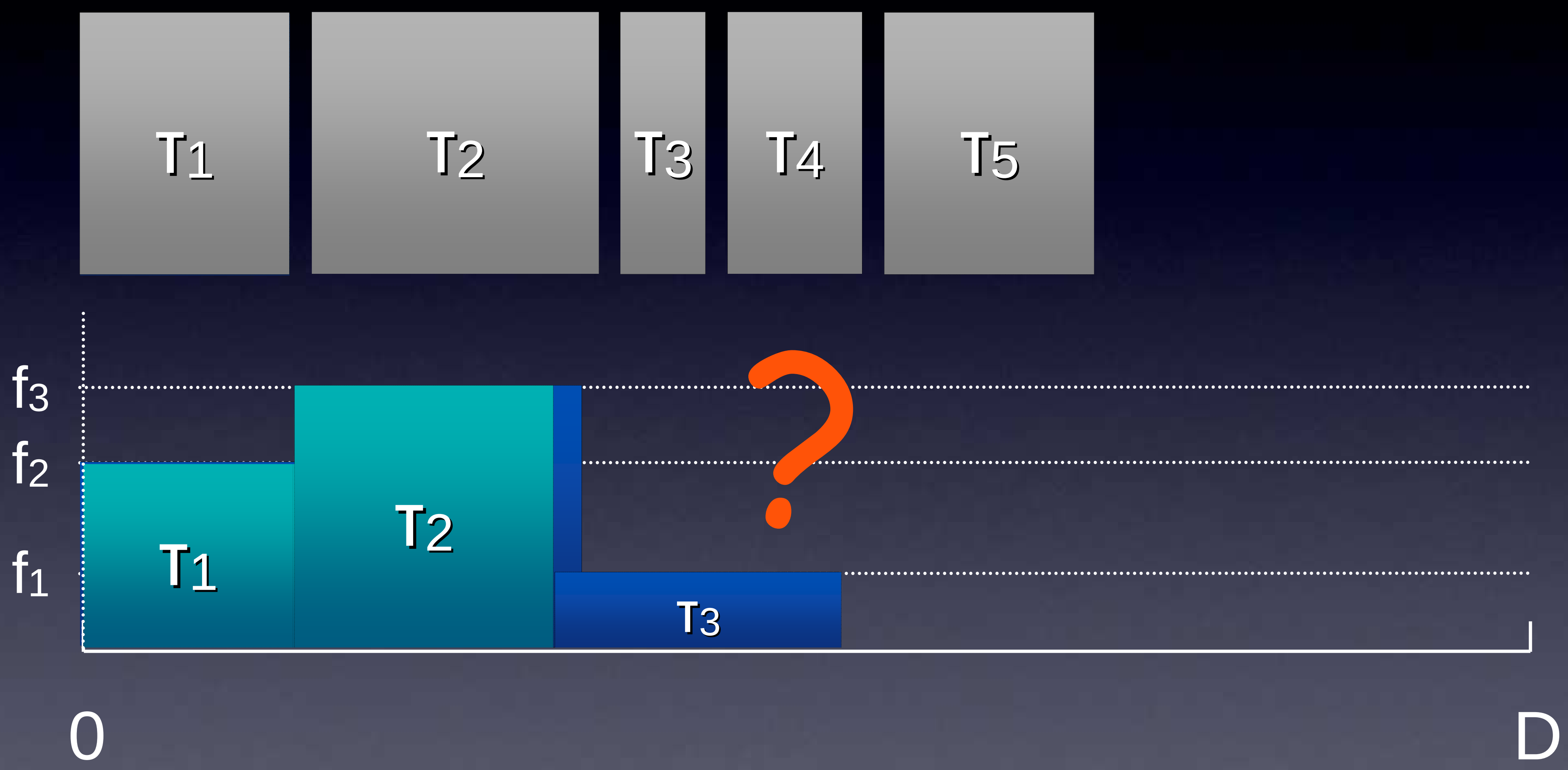
Energy Efficiency (cont'd)

- We consider models with M frequencies $f_1 < ... < f_M$
- For each frequency, we know the consumption
- To simplify: changing frequency is “free”
- One frequency per job

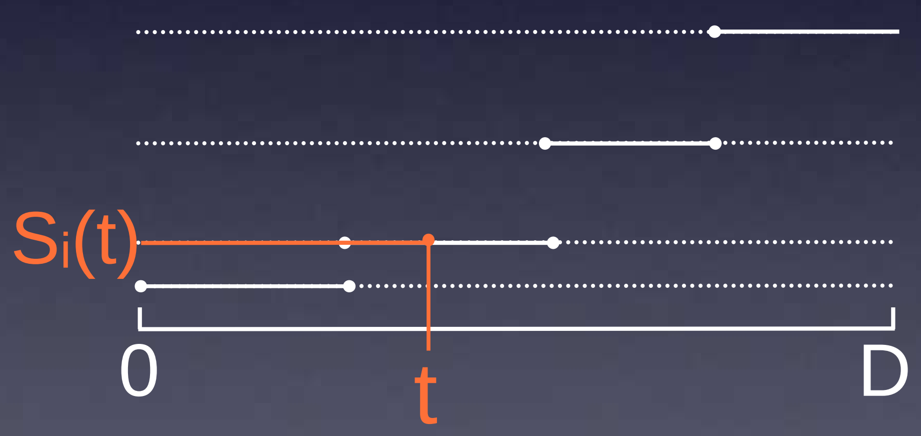
Scheduling Algorithms

Single CPU case

- With only one CPU: lots of results already
- Offline phase: uses length distribution to “prepare” the scheduling
- Online phase: uses the remaining time
- Scheduling: consists in choosing the best frequency


T1
T2
T3
T4
T5
f3
f2
f1
T1
T2
T3
0
D

Single CPU case (cont'd)



- Offline phase: compute a set of n functions S_i (one for each task) - can be complex
- Online phase: when task τ_i has to start at time t, use frequency $S_i(t)$ - must be quick
- Several very good ways of computing S-functions are available

Multiprocessor case

- If several (identical) CPUs are available: much more complex ...
- Not a lot of results in the literature
- We’d like to take advantage of the good results we obtained in the single CPU case

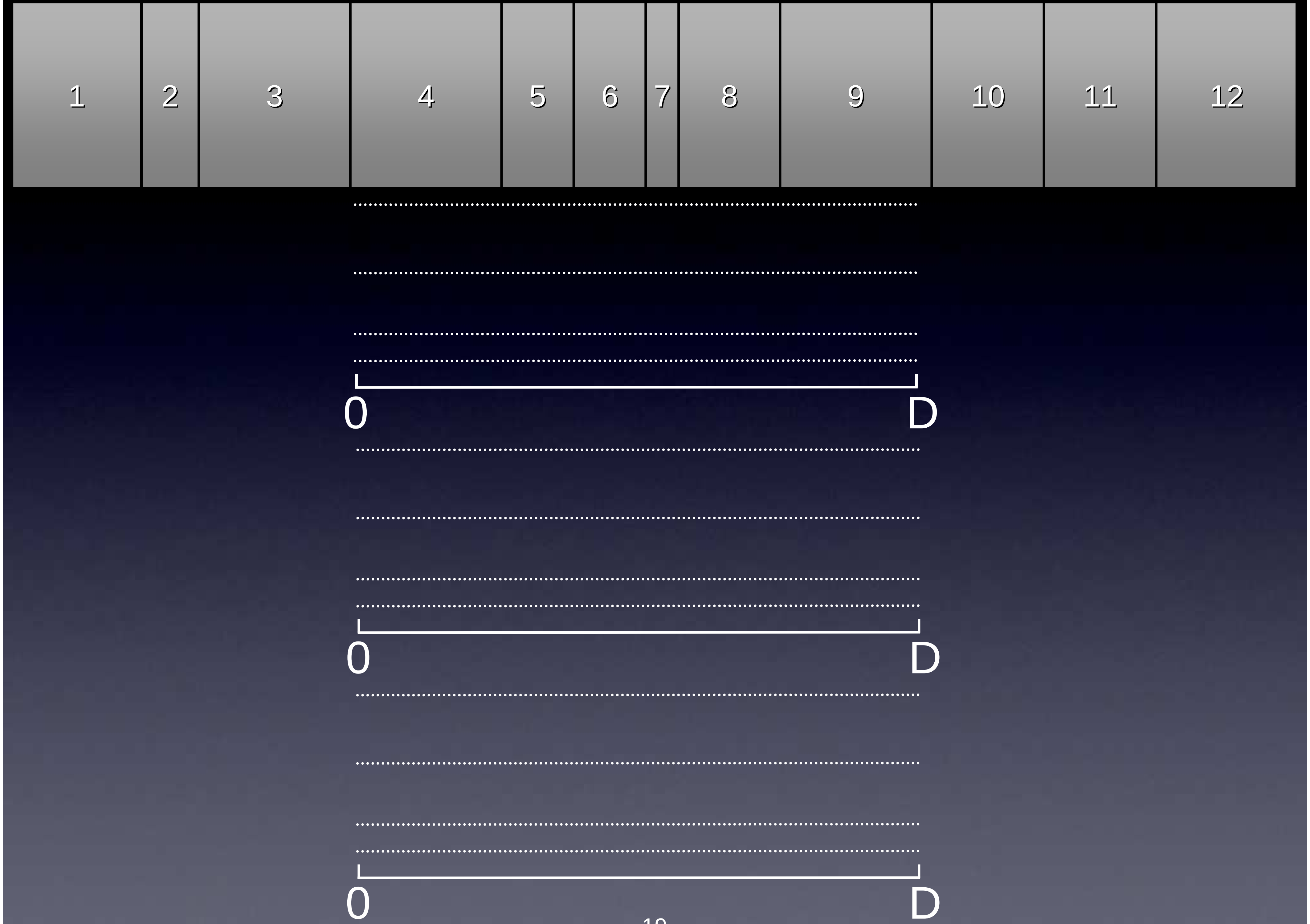
Multiprocessor case

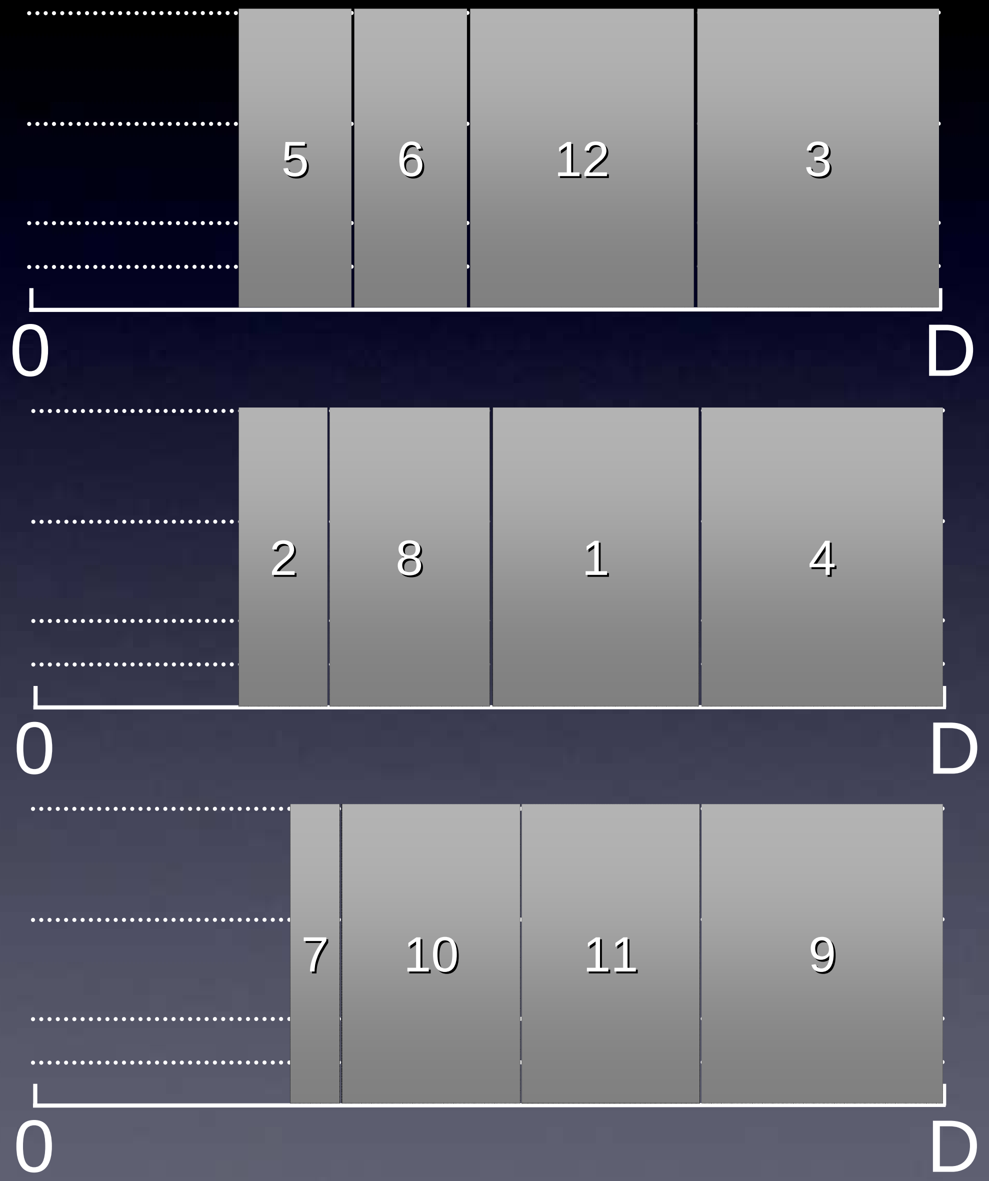
(cont'd)

- When several tasks need to be scheduled on several CPUs, mainly 2 solutions:
  - Partitioning: each task is statically assigned to a CPU. We then run single CPU methods on each CPU. Easier, but less efficient
  - Global scheduling: tasks can move between CPUs (but usually jobs cannot). Much more complex, but often more efficient
- We want to do something in between ...

Virtual Static Partitioning

- Offline phase: virtual static partitioning, each task is assigned to a CPU
- Online phase: we dynamically update this partitioning (re-assign tasks having not started yet), such as most task could feel as on a single CPU

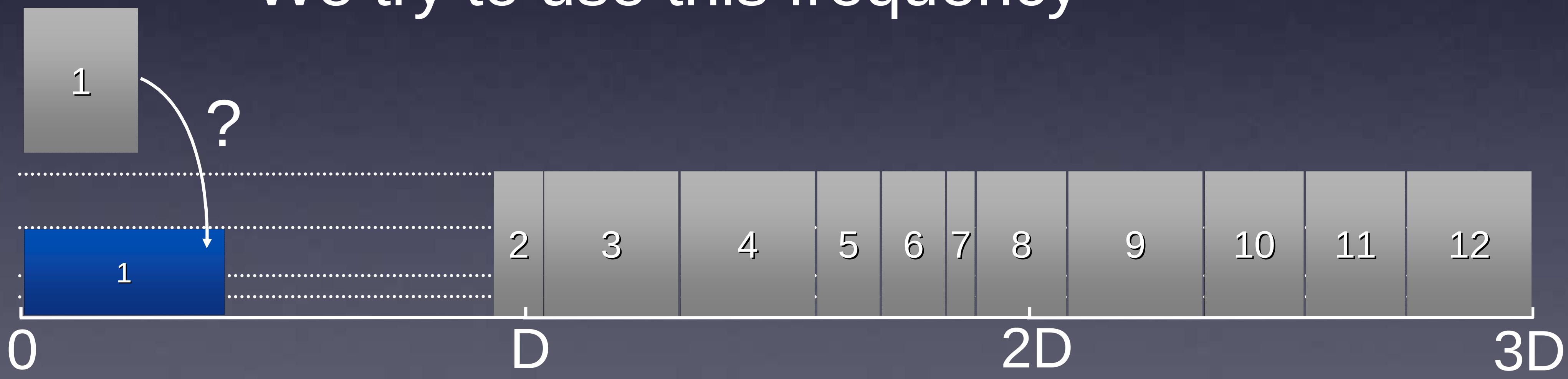

1
2
3
4
5
6
7
8
9
10
11
12
0
D
0
D
0
D

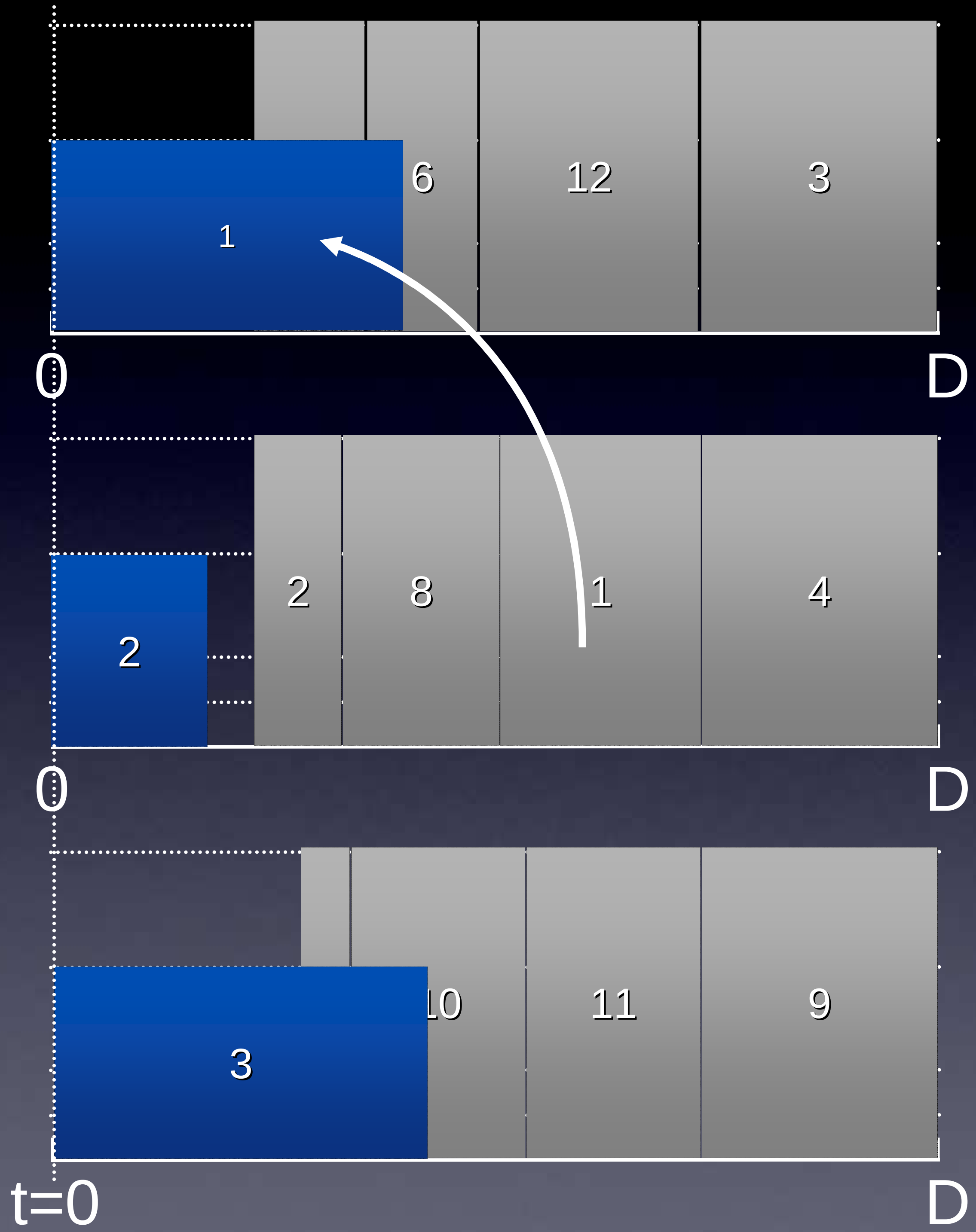


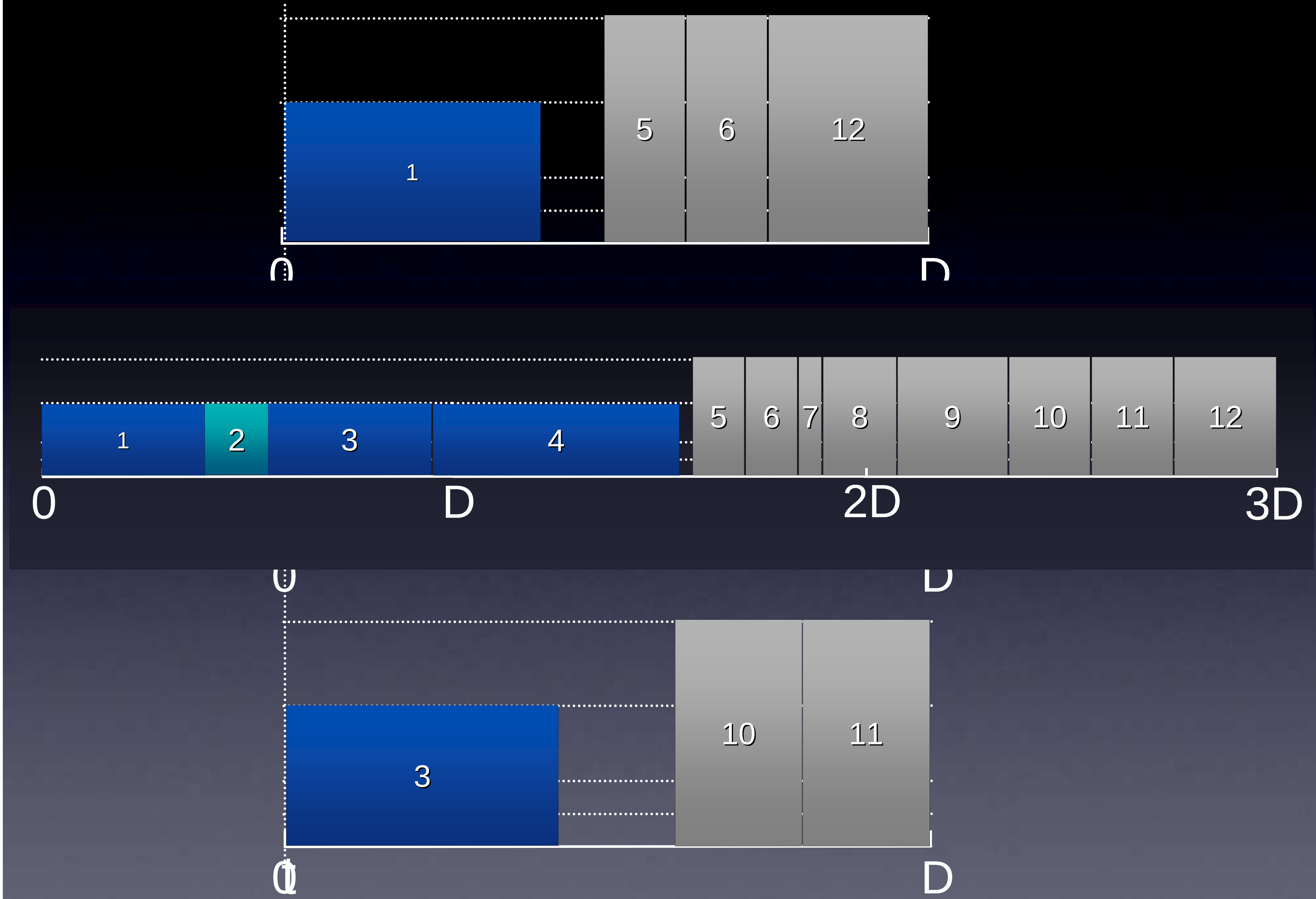
- We have to keep the task order
- Task 1 is then the first to start, for instance on CPU 1
- We want that Task 1 feels as on a single CPU

Online updating

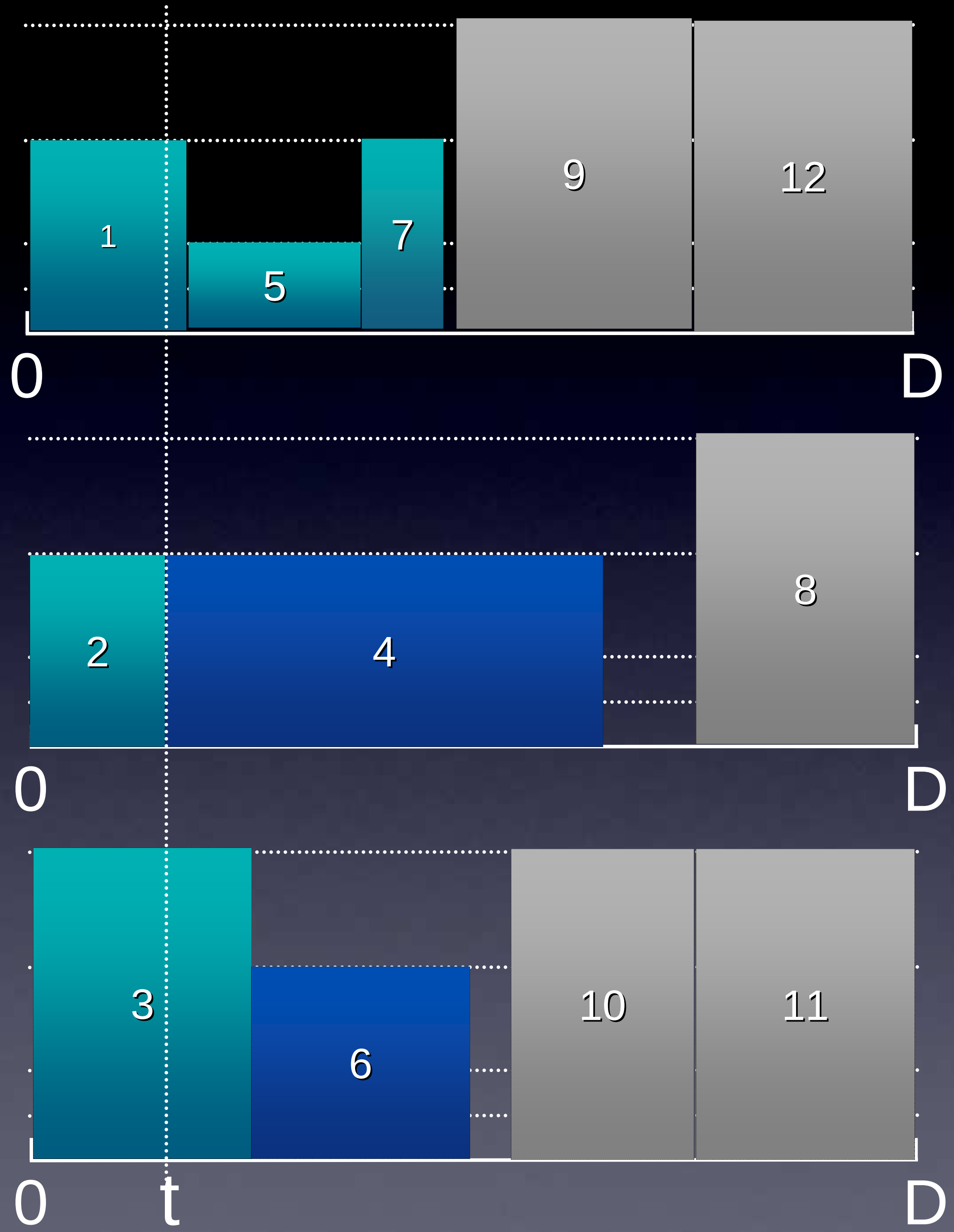
- What frequency would we choose in such a case?
- We try to use this frequency



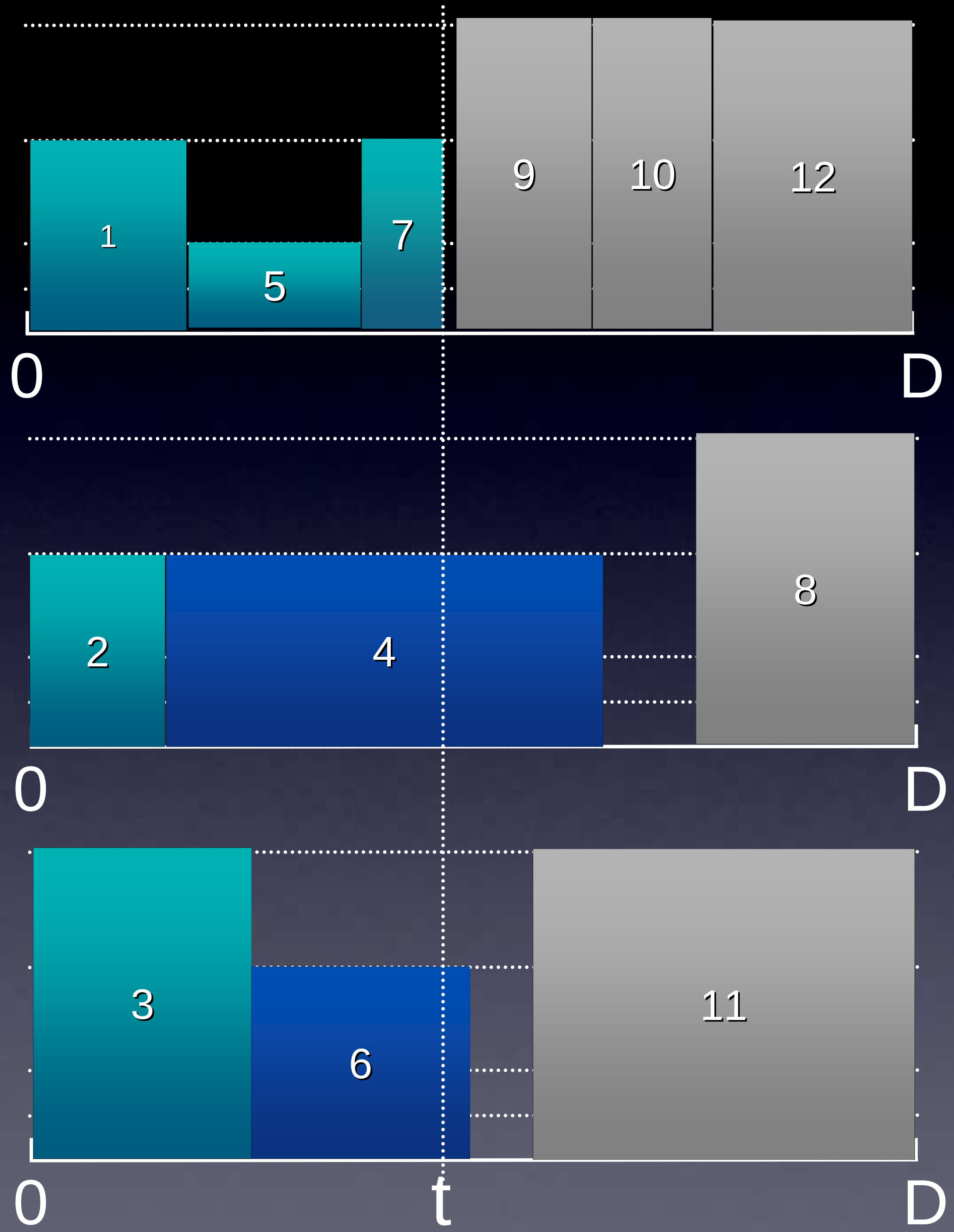

6
12
3
1
0
D
2
8
1
4
2
0
D
10
11
9
3
t=0
D

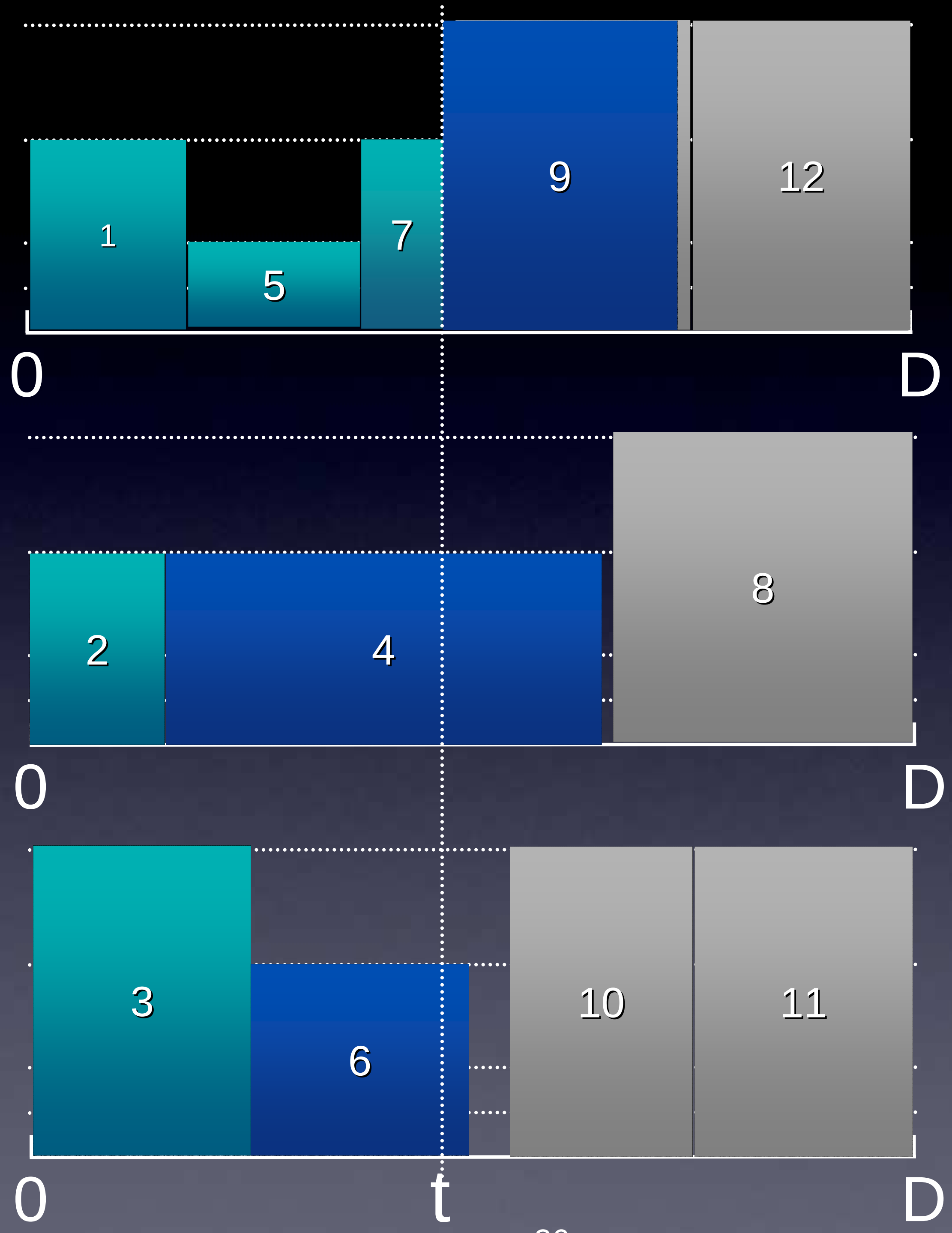


5
6
12
1
1
2
3
4
5
6
7
8
9
10
11
12
0
D
2D
3D
10
11
3
0
D



1
5
7
9
12
0
D
2
4
8
0
D
3
6
10
11
0
t
D


1
5
7
9
10
12
0
D
2
4
8
0
D
3
6
11
0
t
D



1
5
7
9
12
0
D
2
4
8
0
D
3
6
10
11
0
t
D

Online Updating

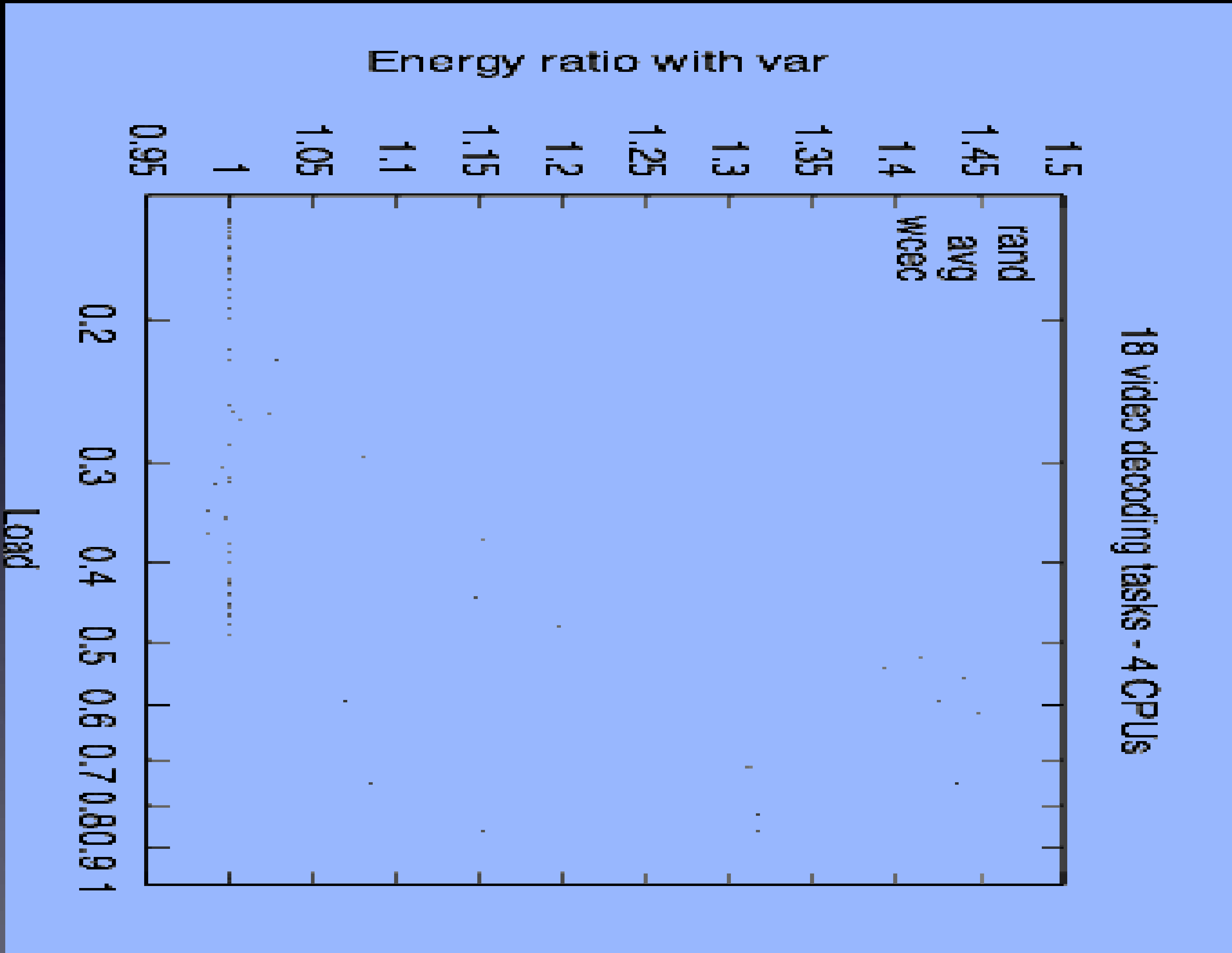
- Moving tasks is a complex problem, especially at high load. Probably close to bin-backing problem
- If we accept to change the order:
  Static partition found

  $\not\Leftarrow$

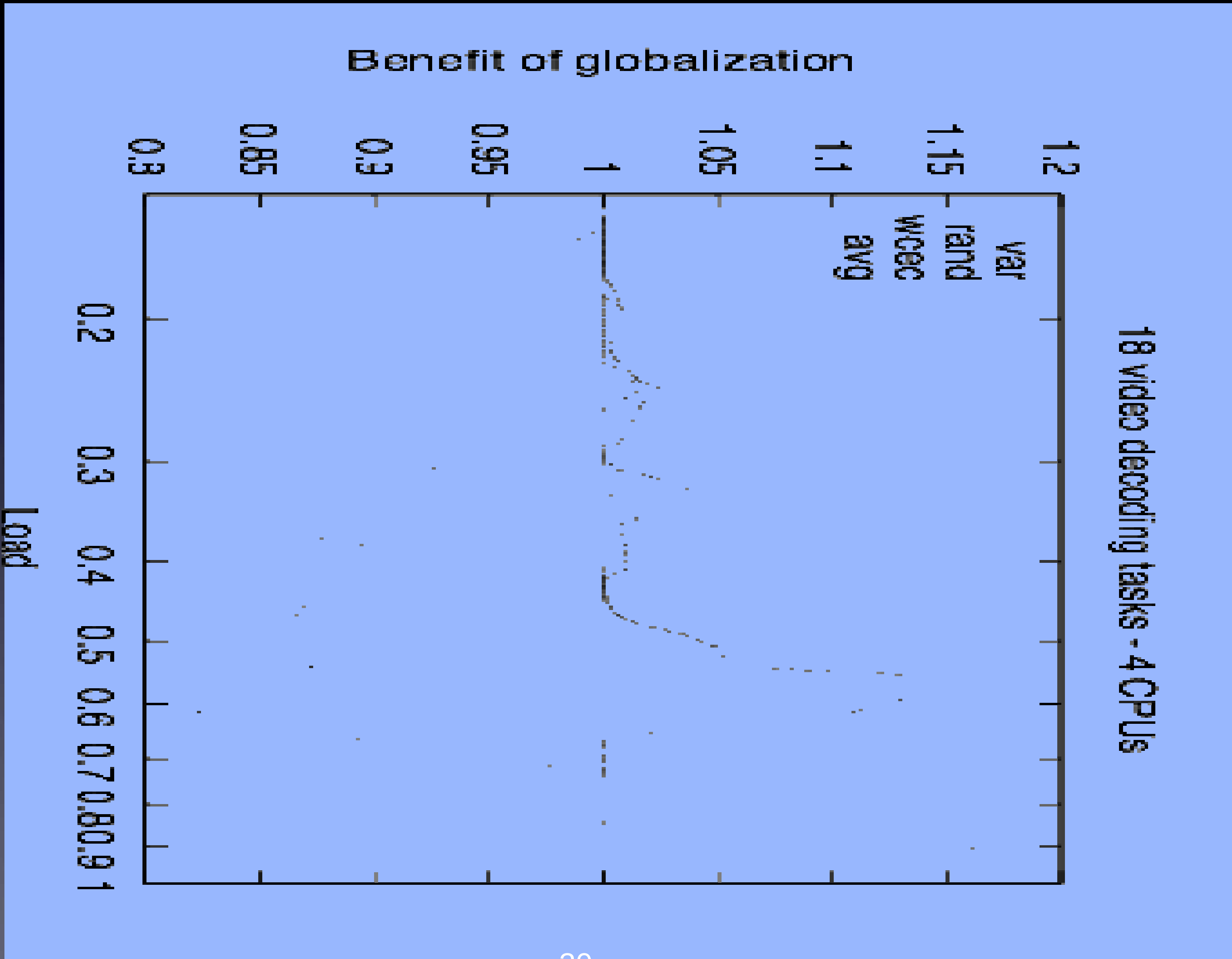
  Schedulable (meet all deadlines)

Some Simulations

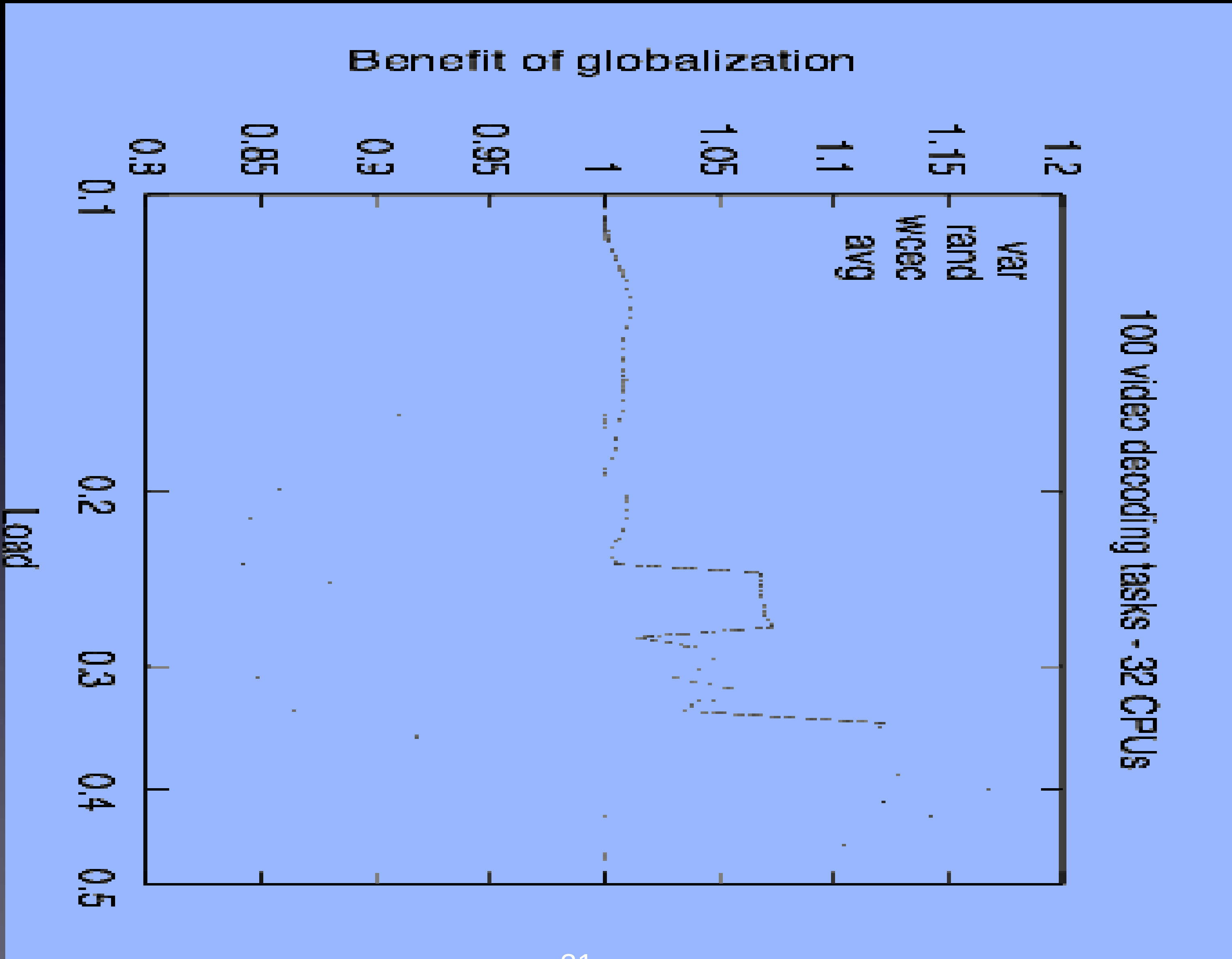
18 Tasks on 4 CPUs



18 Tasks on 4 CPUs



100 Tasks on 32 CPUs



Conclusions

- We have extended a uniprocessor algorithm to a multiprocessor one, keeping real-time constraint guarantees
- When the task order is efficient, global scheduling helps to save energy
- Scheduler rather simple, fast online phase
- ECRTS10 is the next place to submit!

Questions?