Energy-Efficient Algorithms

Susanne Albers
University of Freiburg
Germany

Motivation

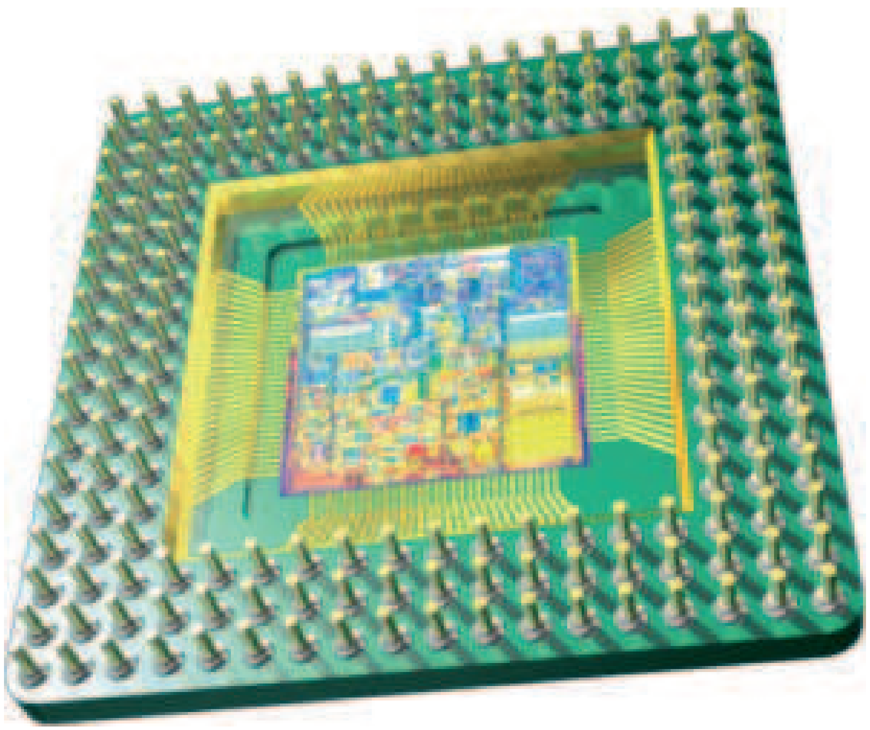
- Energy consumption grows exponentially in computing devices
  computers, embedded systems, portable devices, ...
- Performance of processors doubles every three years

- Critical in battery-operated devices
- Critical in terms of cost (computer centers)
- Critical since energy is converted into heat

Algorithmic techniques

- Power-down strategies: Put system into sleep state when idle.
- Dynamic speed scaling: Microprocessors can run at variable speed.

Dynamic speed scaling



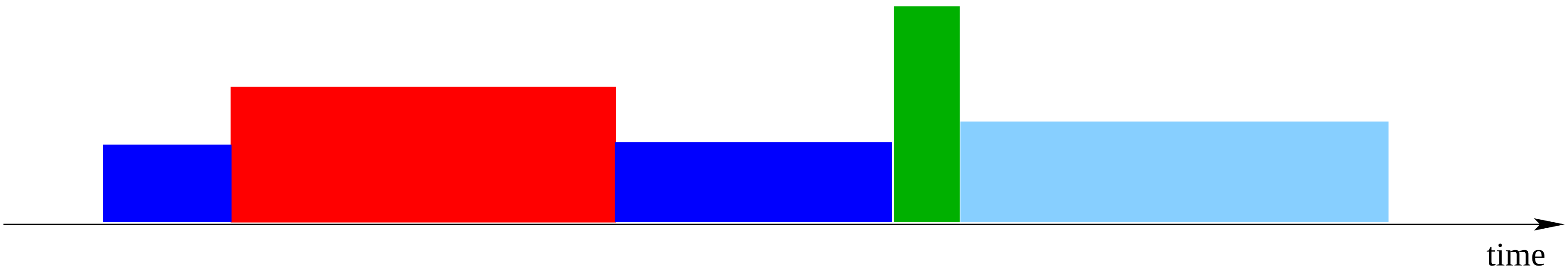
Microprocessors can run at variable speed

The higher the speed, the higher the power consumption

Speed s
Power consumption
$P(s) = s^{\alpha} \quad \alpha > 1$

Previous work

Deadline-based scheduling   1 processor



- Speed s Energy consumption $P(s) = s^\alpha$ $\alpha > 1$
- $\sigma = J_1, \ldots, J_n$

  J_i: a_i = arrival time

  b_i = deadline

  p_i = processing volume $t = p_i / s$
- Preemption allowed
- Construct feasible schedule minimizing total energy consumption.

Competitive analysis

Online problem: jobs arrive one by one

A: Online algorithm	OPT: Offline algorithm
$A(\sigma)$	$OPT(\sigma)$

A is c-competitive, if for all job sequences σ

$$A(\sigma) \leq c \cdot OPT(\sigma).$$

Previous work

Offline problem

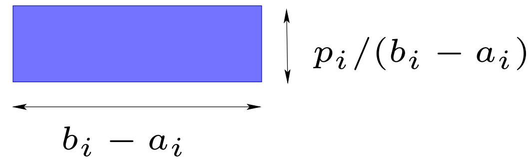
- polynomially solvable
  (Yao, Demers, Shenker 1995)

Online problem

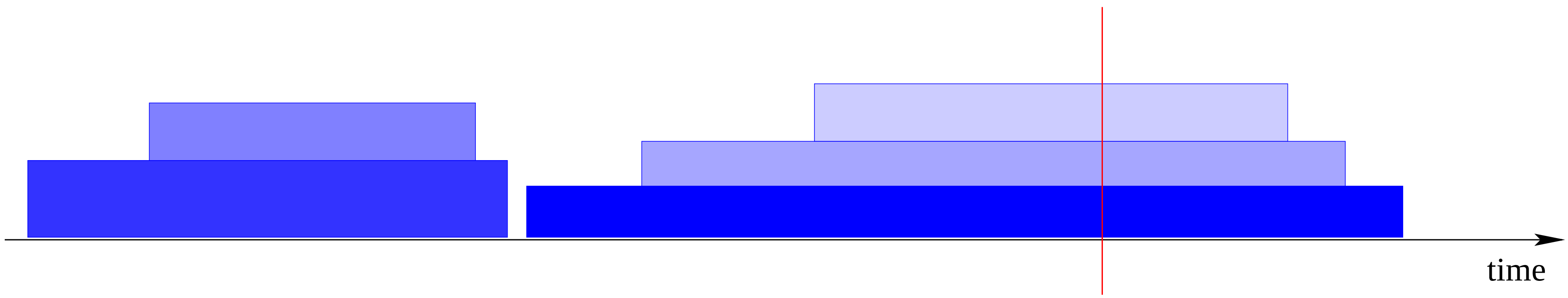
- Average Rate: $\alpha^\alpha \leq c \leq 2^\alpha \alpha^\alpha$ (Yao, Demers, Shenker 1995)
  Optimal Available: $c = \alpha^\alpha$ (Bansal, Kimbrel, Pruhs 2004)
- Upper bound $2(\alpha/(\alpha-1))^\alpha e^\alpha$
  Lower bound $\Omega((4/3)^\alpha)$
  (Bansal, Kimbrel, Pruhs 2004)

Average Rate

1. Job density $\delta_i = p_i/(b_i - a_i)$

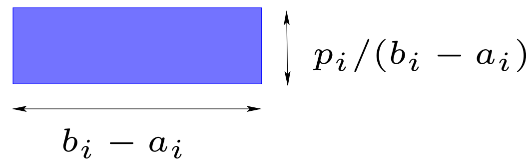


2. $s(t) = \sum_{i\,:\,t \in [a_i, b_i]} \delta_i$

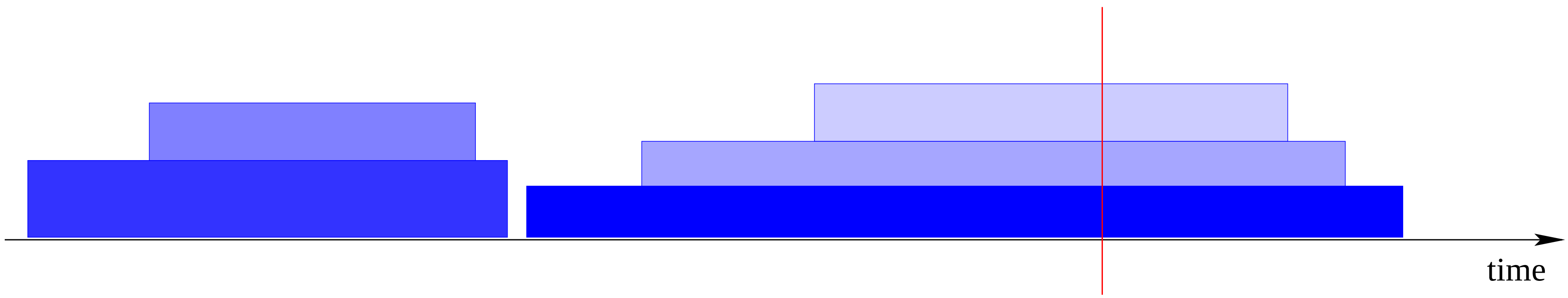


Average Rate

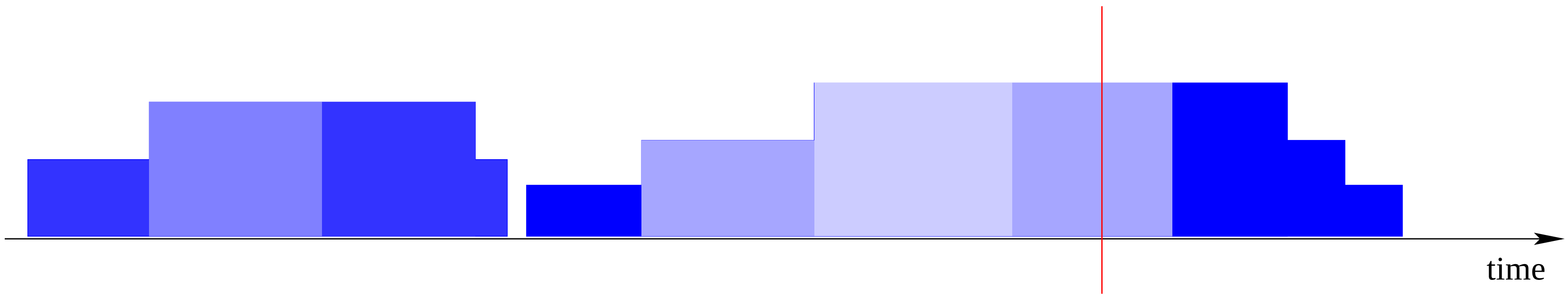
1. Job density $\delta_i = p_i/(b_i - a_i)$



2. $s(t) = \sum_{i\,:\,t \in [a_i, b_i]} \delta_i$

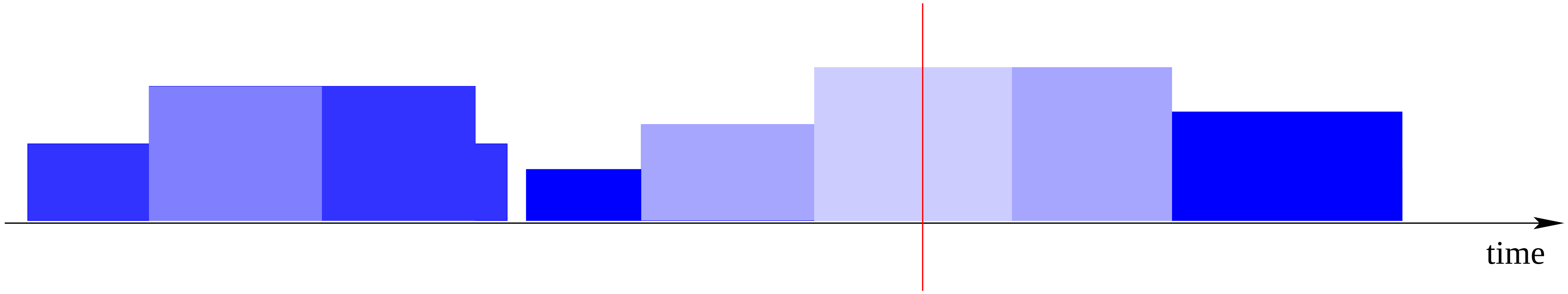


3. Use Earliest Deadline Policy



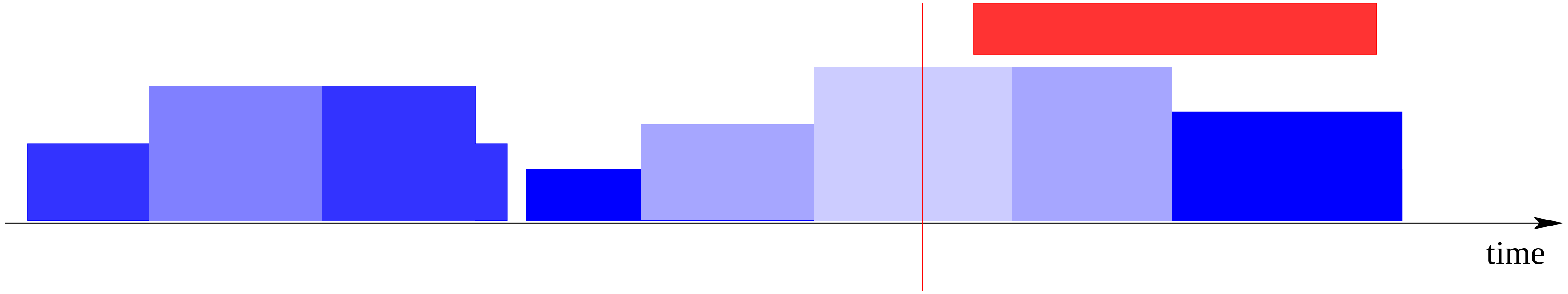
Optimal Available

At any time compute optimal schedule for remaining workload.

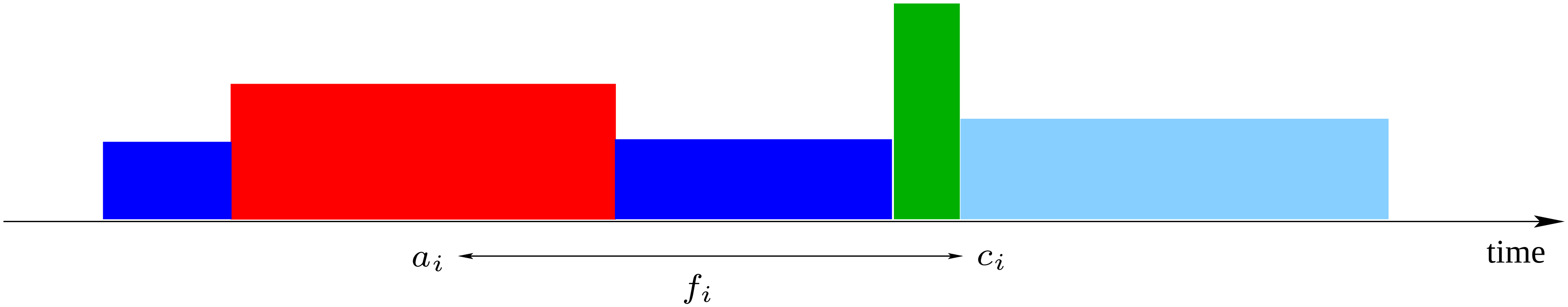


Optimal Available

At any time compute optimal schedule for remaining workload.



Flow time minimization



- Computer systems: jobs are not labeled with deadlines
- Users expect good response times
- Flow time of J_i: $f_i = c_i - a_i$

  c_i = completion time
- Energy and flow time minimization are orthogonal objectives

  low energy $\Longrightarrow$ low speed $\Longrightarrow$ high flow times

  small flow time $\Longrightarrow$ high speed $\Longrightarrow$ high energy

Previous work

Pruhs, Uthaisombut, Woeginger 2004

- Minimize flow time given fixed energy volume V
- $p_i = 1$ for all i
- Polynomial time offline algorithm computing optimal schedules simultaneously for all V.

Our approach

Albers, Fujiwara   STACS 2006

$$\min \left(\text{Energy} + \sum_{i=1}^{n} f_i \right)$$

- $\sigma = J_1, \ldots, J_n$
  J_i: a_i = arrival time
  p_i = processing volume
  preemption not allowed

- Combined objective functions for facility location, network design, TCP-acknowledgement, ...

Our results

p_i arbitrary

- Competitive ratio of $\Omega(n^{1-1/\alpha})$

$p_i = 1$

- Competitive ratio of $8e(1+\Phi)^\alpha$ $\quad \Phi = (1+\sqrt{5})/2 \approx 1.618$
- Offline problem polynomially solvable using dynamic programming; approach also solves problem of Pruhs, Uthaisombut, Woeginger

PhaseBalance

$S :=$ { jobs with $a_i = 0$ }

while $S \neq \emptyset$ **do**

  schedule jobs in S optimally;

  $S :=$ { jobs having arrived in the meantime };

endwhile

PhaseBalance

$S :=$ { jobs with $a_i = 0$ }

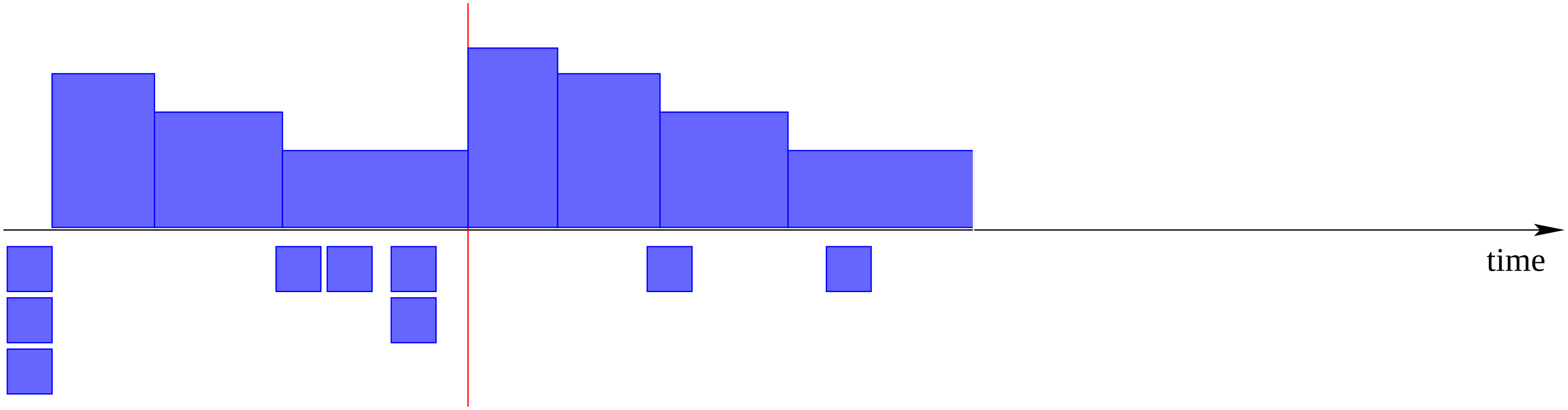
while $S \neq \emptyset$ **do**

  schedule jobs in S optimally;

  $S :=$ { jobs having arrived in the meantime };

endwhile

PhaseBalance



$S :=$ { jobs with $a_i = 0$ }

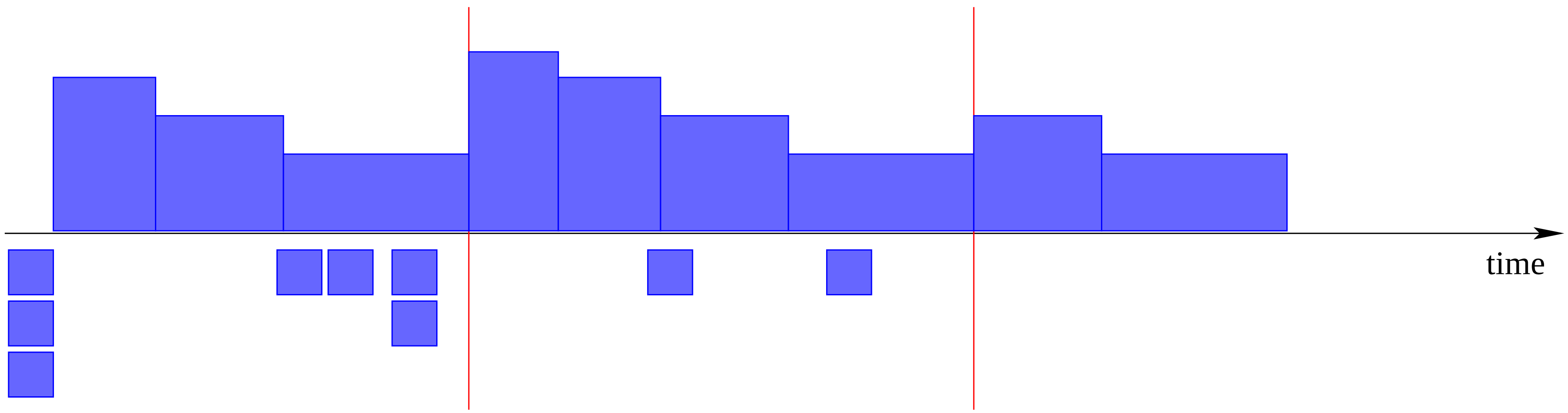
while $S \neq \emptyset$ **do**

  schedule jobs in S optimally;

  $S :=$ { jobs having arrived in the meantime };

endwhile

PhaseBalance



$S :=$ { jobs with $a_i = 0$ }

while $S \neq \emptyset$ **do**

  schedule jobs in S optimally;

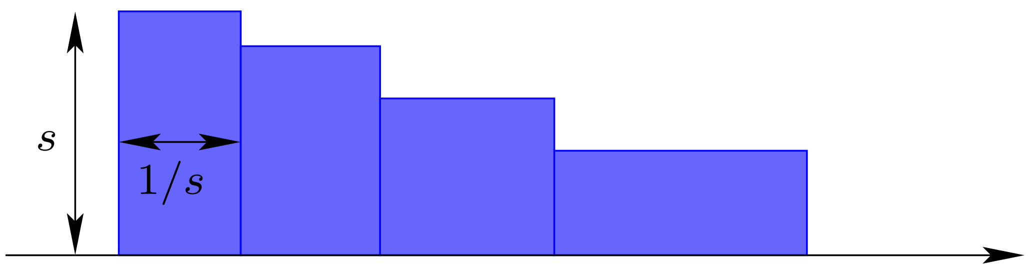
  $S :=$ { jobs having arrived in the meantime };

endwhile

Phase scheduling

- n_i jobs in phase i
- Speed sequence

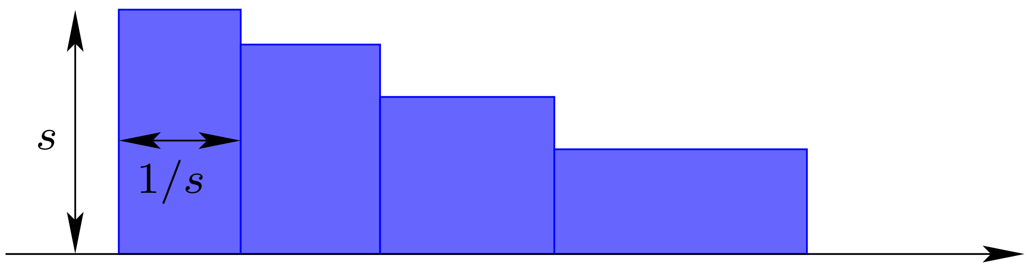
$$\sqrt[\alpha]{\frac{n_i}{\alpha - 1}},\ \sqrt[\alpha]{\frac{n_i - 1}{\alpha - 1}},\ \ldots,\ \sqrt[\alpha]{\frac{1}{\alpha - 1}}$$



Speed computation

First job of phase i

$$f(s) = s^{\alpha-1} + \frac{n_i}{s}$$



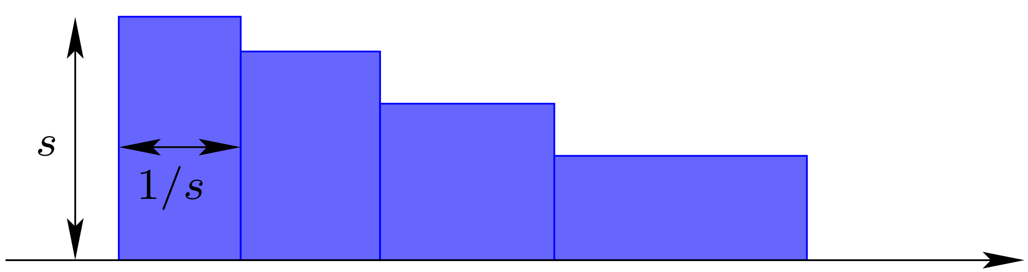
Speed computation

First job of phase i

$$\min f(s) = s^{\alpha-1} + \frac{n_i}{s}$$

$$f'(s) = (\alpha - 1)s^{\alpha-2} - \frac{n_i}{s^2}$$

$$\begin{aligned} f'(s) = 0 \quad &\Leftrightarrow \quad (\alpha - 1)s^{\alpha} = n_i \\ &\Leftrightarrow \quad s = \sqrt[\alpha]{\frac{n_i}{\alpha - 1}} \end{aligned}$$



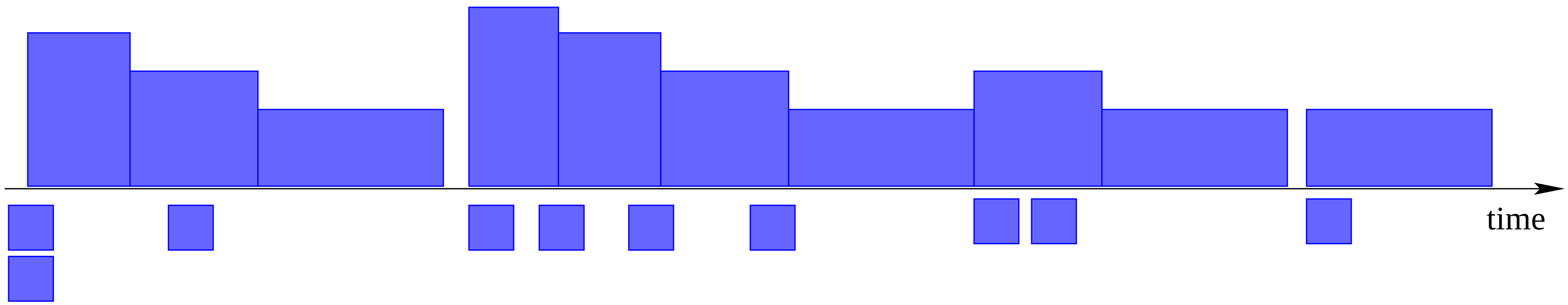
Analysis OPT

Lemma: If there are l unfinished jobs waiting

$$s \geq \sqrt[\alpha]{\frac{l}{\alpha - 1}}.$$

Lemma: Every job is finished at least as early as in the online schedule.

Offline algorithm



Sub-schedules $S_1, \ldots, S_m$.

S_j processes job with indices $j_1, \ldots, j_l$, where

$$c_i > a_{i+1} \qquad i = j_1, \ldots, j_l - 1$$

$$c_{j_l} \leq a_{j_l+1}$$

Speeds in subschedules

l jobs in interval of length T

- $s_i = \sqrt[\alpha]{\frac{l-i+1}{\alpha-1}}$ if $T \geq \sum_{i=1}^{l} 1/s_i$

- $s'_i = \sqrt[\alpha]{\frac{l-i+1+c}{\alpha-1}}$ if $T < \sum_{i=1}^{l} 1/s_i$

  c unique value with $\sum_{i=1}^{l} 1/s'_i = T$

Subproblems

$P[i, i+l]$ = subproblem with $J_i, \ldots, J_{i+l}$

$C[i, i+l]$ = optimal cost of $P[i, i+l]$ if J_{i+l} must be finished by a_{i+l+1}

Determine $C[1, n]$

Multi-processor speed scaling

- Server systems: several CPUs

  Google engineers: power costs overtake hardware costs

- Laptops: dual-processors

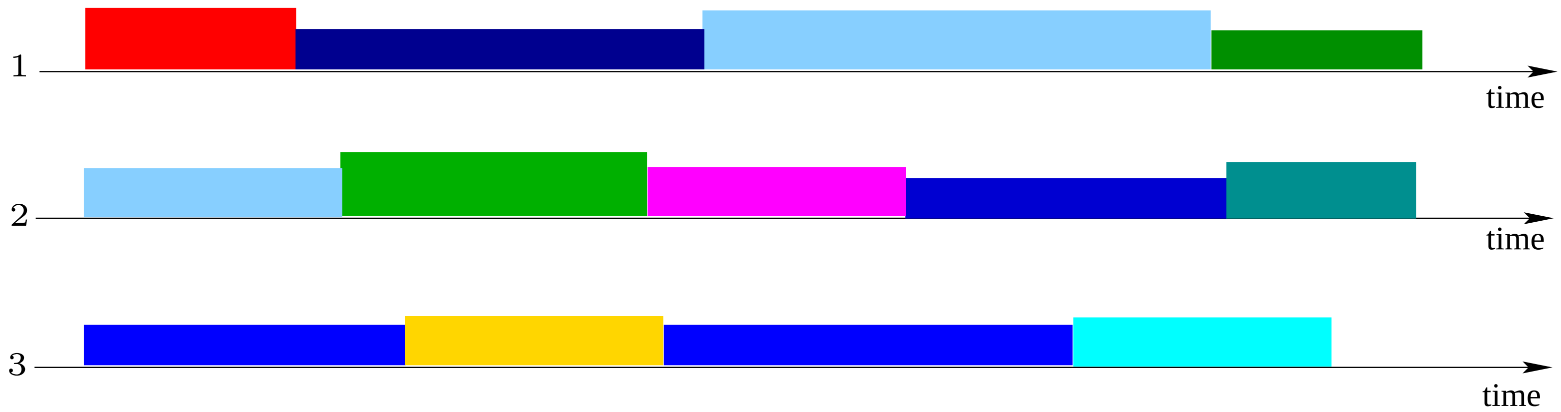
  AMD ”Quad-core design”

  Architectures with 8 CPUs are being developed

- Intel: experiments with 80 CPUs on one die

Multi-processor speed scaling

Albers, Müller, Schmelzer   SPAA 2007



- Each processor may run at individual speed s.
- Deadline based scheduling J_i: a_i, b_i, p_i
- Preemption allowed, migration disallowed
- Construct feasible schedule minimizing total energy consumption.

Unit size jobs, $p_i = 1$

Offline problem

- Agreeable deadlines

  $a_i < a_j \implies b_i \leq b_j$

  Polynomially solvable

- Arbitrary deadlines

  NP-hard

  Approximation factor $\alpha^\alpha 2^{4\alpha}$

Arbitrary size jobs

Offline problem

- Common release time or common deadline
  Approximation factor $2(2 - \frac{1}{m})^\alpha$

- Arbitrary deadlines
  Approximation factor $\alpha^\alpha 2^{4\alpha}$

Online setting

- $p_i = 1$, agreeable deadlines

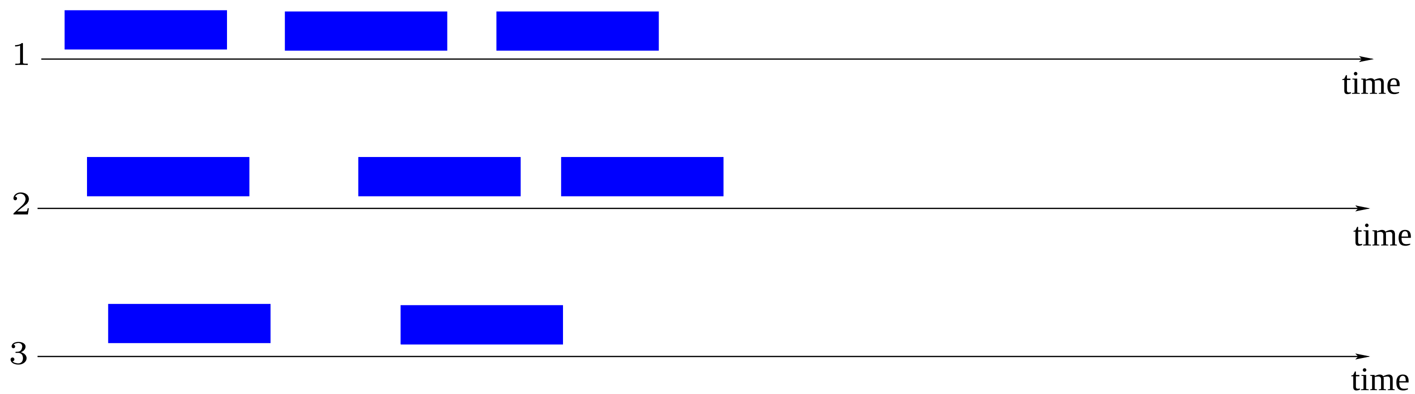
  Competitive factor $2(\alpha/(\alpha-1))^\alpha e^\alpha$

- $p_i = 1$, arbitrary deadlines

  p_i arbitrary, agreeable deadlines

  Competitive factor $\alpha^\alpha 2^{4\alpha}$

Unit size jobs, agreeable deadlines

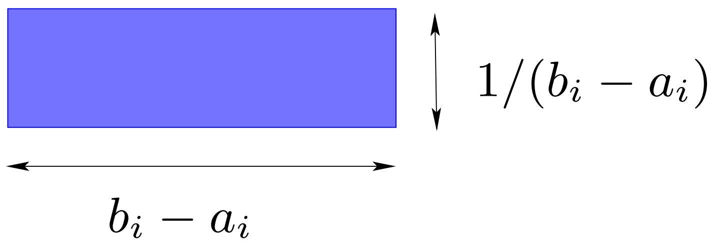


1. Sort jobs according to non-decreasing release dates.
2. Assign jobs to processors using Round Robin.
3. For each processor, compute optimal schedule.

Unit size jobs, arbitrary deadlines

1. Job density $\delta_i = 1/(b_i - a_i)$

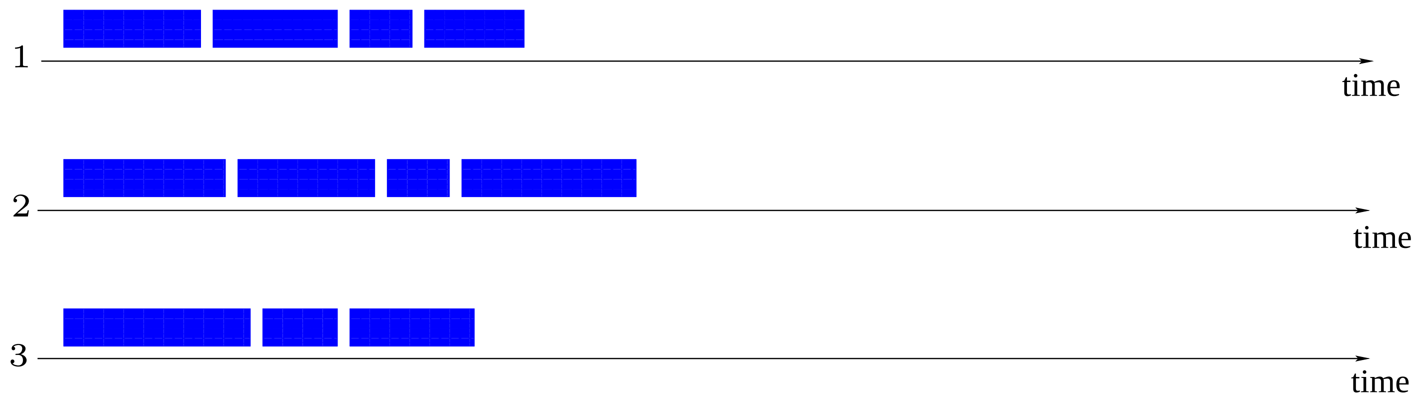
   $\Delta = \max_i \delta_i$



2. Job classes $C_k = [\Delta 2^{-k}, \Delta 2^{-(k-1)})$
3. Apply Round Robin to each class
4. For each processor, compute optimal schedule

Arbitrary size jobs

Common release time



1. Sort jobs according to non-decreasing deadlines.
2. Assign jobs to processors using List scheduling.
3. For each processor, compute optimal schedule.

Open problems

Flow time minimization

- Exact competitive ratio of PhaseBalance
- Analyze of following speed scaling algorithm:
  Speed $\sqrt[\alpha]{i}$ when there are i unfinished jobs waiting

Multi-processor setting

- Improve approximation guarantees
- Consider migration