Scheduling Input

**Input:** \( n \) jobs, each with

- release time: Earliest time the job can start
- processing time: How long the job runs
Scheduling Input

**Input:** $n$ jobs, each with

- release time: Earliest time the job can start
- processing time: How long the job runs
Dynamic Voltage Scaling

Many modern processors can enter lower-power states to conserve energy

**AMD Athlon 64 power consumption**

<table>
<thead>
<tr>
<th>Speed (MHz)</th>
<th>Power consumed (W)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2000</td>
<td>89</td>
</tr>
<tr>
<td>1800</td>
<td>66</td>
</tr>
<tr>
<td>800</td>
<td>35</td>
</tr>
</tbody>
</table>

(from AMD Athlon 64 Processor Power and Thermal Data Sheet)
Continuous model

- Standard algorithmic model: power $= \text{speed}^\alpha$ for constant $\alpha$
  
  - Continuity approached as processors allow more speeds (also observed when overclocking)
  
  - Experimentally, $\alpha \approx 3$
• Standard algorithmic model: $\text{power} = \text{speed}^\alpha$ for constant $\alpha$
  
  – Continuity approached as processors allow more speeds
    (also observed when overclocking)
  
  – Experimentally, $\alpha \approx 3$

• Initially assume power function is continuous & strictly convex
**Power-aware scheduling**

**Input:** $n$ jobs, each with

- **release time:** Earliest time the job can start
- **work requirement:** Number of cycles the job needs

![Diagram](image)

Area = work requirement
Quality measure: Makespan

- The “going home” problem
  - Computer operator has list of jobs to run
  - Wants to finish last job as early as possible
Quality measure: Makespan

- The “going home” problem
  - Computer operator has list of jobs to run
  - Wants to finish last job as early as possible

- Modern examples:
  - System installation: assign workers to parts of task
  - Rendering a multi-part webpage
Makespan scheduling

**Makespan**: Completion time of rightmost job
Power-aware makespan scheduling

Can vary speed to improve the makespan:
Properties of the optimal schedule

1. One speed per job

2. No idle time

3. First-come first-served

4. One speed per block

5. Non-decreasing block speeds
Linear-time algorithm

Maintain list of tentative blocks, initially empty

For $i = 1$ to $n$
    Add job $i$ to end of list as its own block
    While last block runs slower than its predecessor,
        Merge last 2 blocks
Linear-time algorithm

Maintain list of tentative blocks, initially empty

For $i = 1$ to $n$

    Add job $i$ to end of list as its own block

    While last block runs slower than its predecessor,
        Merge last 2 blocks
Linear-time algorithm

Maintain list of tentative blocks, initially empty
For $i = 1$ to $n$
    Add job $i$ to end of list as its own block
    While last block runs slower than its predecessor,
        Merge last 2 blocks
Linear-time algorithm

Maintain list of tentative blocks, initially empty

For $i = 1$ to $n$

Add job $i$ to end of list as its own block

While last block runs slower than its predecessor,

Merge last 2 blocks
Linear-time algorithm

Maintain list of tentative blocks, initially empty

For $i = 1$ to $n$

Add job $i$ to end of list as its own block

While last block runs slower than its predecessor,

Merge last 2 blocks
Linear-time algorithm

Maintain list of tentative blocks, initially empty

For $i = 1$ to $n$

Add job $i$ to end of list as its own block

While last block runs slower than its predecessor,

Merge last 2 blocks
Linear-time algorithm

Maintain list of tentative blocks, initially empty

For $i = 1$ to $n$
  Add job $i$ to end of list as its own block
  While last block runs slower than its predecessor,
    Merge last 2 blocks
Linear-time algorithm

Maintain list of tentative blocks, initially empty

For $i = 1$ to $n$
    Add job $i$ to end of list as its own block
    While last block runs slower than its predecessor,
        Merge last 2 blocks

Med energy:
Linear-time algorithm

Maintain list of tentative blocks, initially empty

For $i = 1$ to $n$

  Add job $i$ to end of list as its own block

  While last block runs slower than its predecessor,
    Merge last 2 blocks

Low energy:
Linear-time algorithm

Maintain list of tentative blocks, initially empty

For $i = 1$ to $n$

- Add job $i$ to end of list as its own block
- While last block runs slower than its predecessor,
  Merge last 2 blocks