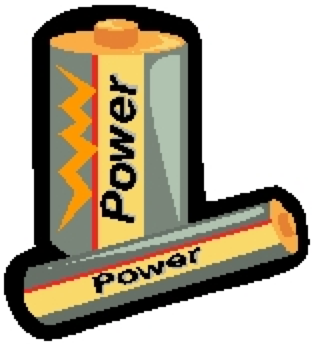
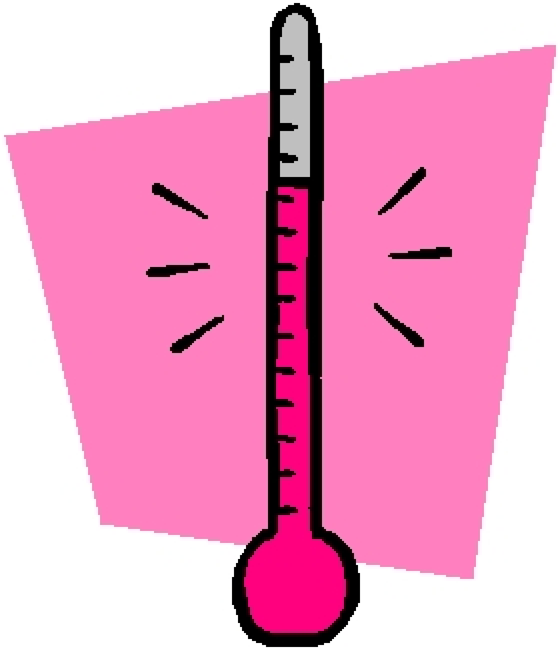
PPT to PDF 1.4

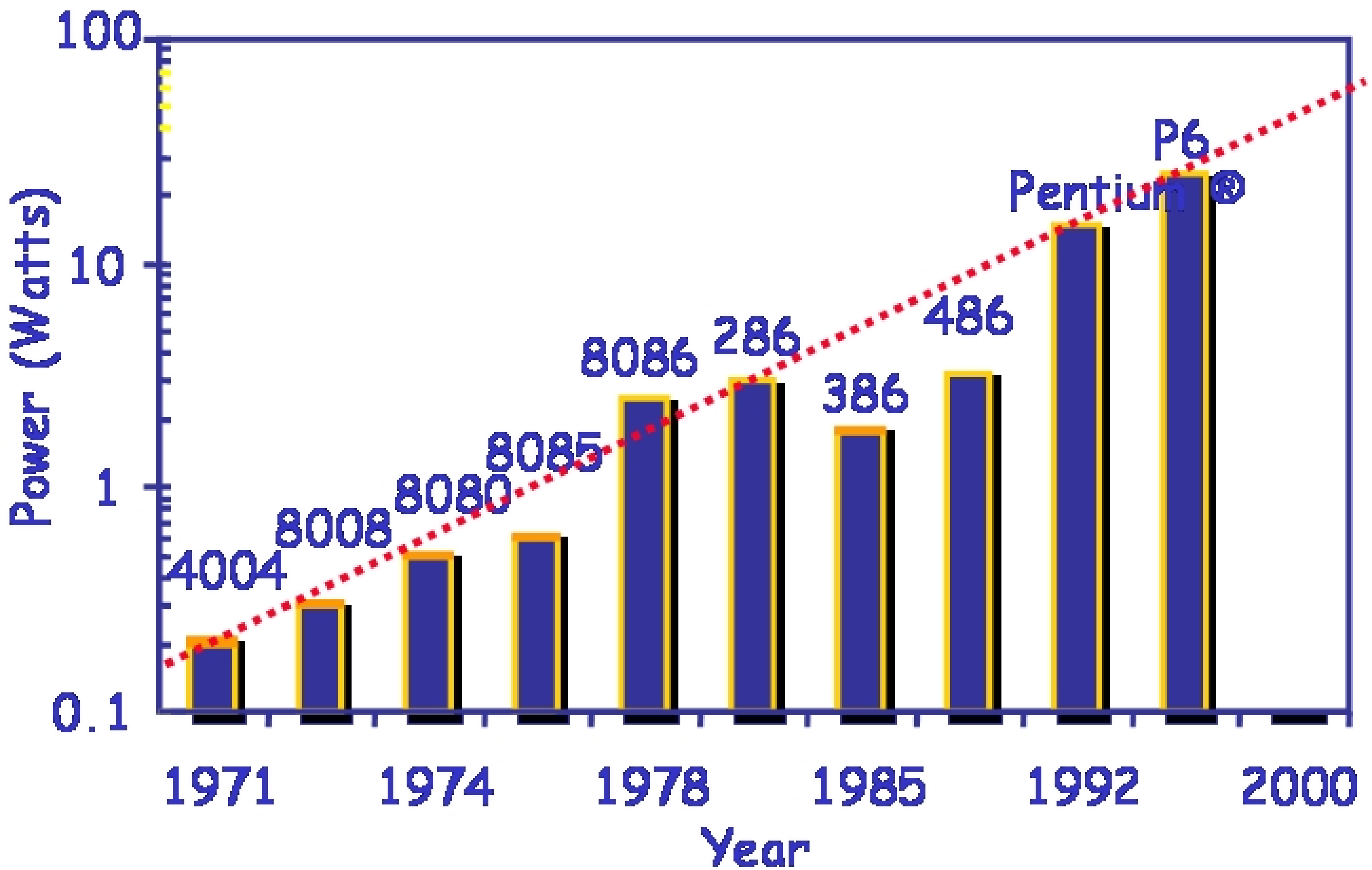
Speed Scaling Algorithms For Power Management





Kirk Pruhs
University of Pittsburgh

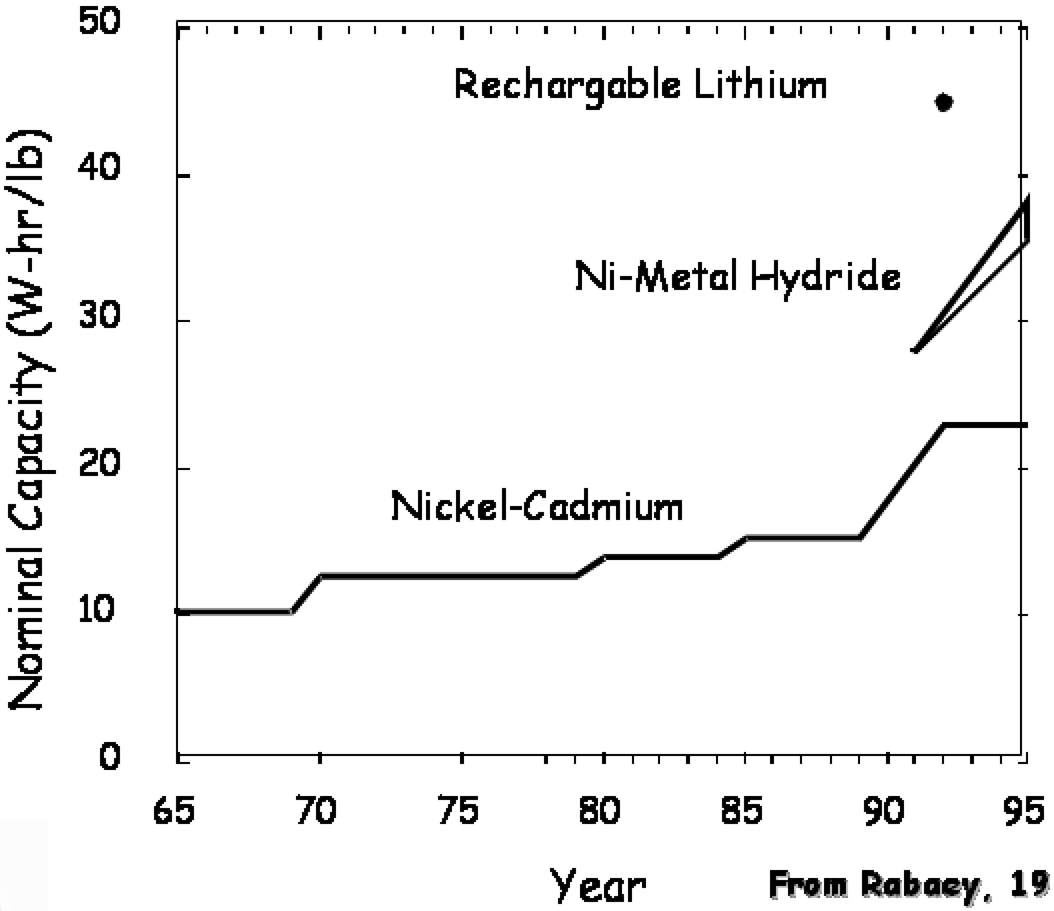
Microprocessor Power Increasing Exponentially



Source: Borkar, De Intel®

PPT to PDF 1.4

Why worry about power ?
Most Obvious Answer: Battery capacity increasing linearly



From Rabaey, 1995

PPT to PDF 1.4

Why worry about power ?
Less Obvious Answer 2: Chips get hot

Intel Hits "Thermal Wall"

Reuters Friday May 7, 2004

SAN FRANCISCO, May 7 (Reuters) - Intel Corp. said on Friday it has scrapped the development of two new computer chips (code-named Tejas and Jayhawk) for desktop/server systems in order to rush to the marketplace a more efficient chip technology more than a year ahead of schedule. Analysts said the move showed how eager the world's largest chip maker was to cut back on the heat its chips generate. Intel's method of cranking up chip speed was beginning to require expensive and noisy cooling systems for computers.

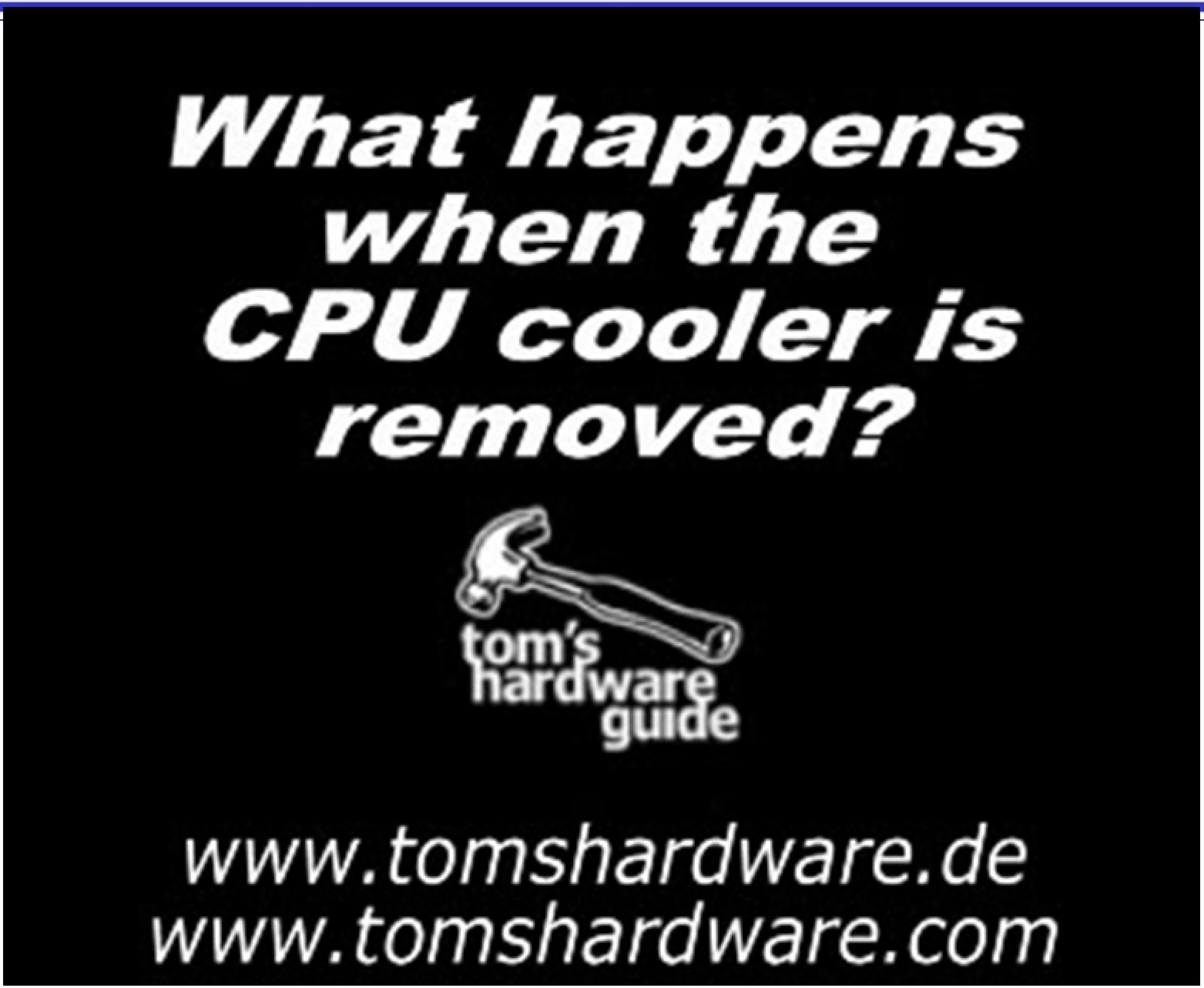
PPT to PDF 1.4

Laptops may damage male fertility

- Reuters: December 9, 2004

  Men should keep their laptops off their laps because they could damage fertility, an expert said on Thursday. Laptops, which reach high internal operating temperatures, can heat up the scrotum which could affect the quality and quantity of men's sperm. "The increase in scrotal temperature is significant enough to cause changes in sperm parameters," said Dr Yefim Sheynkin, an associate professor of urology at the State University of New York at Stony Brook.

Speed Scaling to Manage Power/Heat



Outline

- Introduction
  - Importance of power management for energy and temperature
  - Speed scaling power management technique
  - Modeling energy and temperature
  - Brief review of scheduling
  - Brief history of the literature
- Algorithmic results
  - Offline optimal speed scaling algorithms
    - Deadline feasibility and energy
    - Deadline feasibility and temperature
    - Flow time and energy
  - Online speed scaling algorithms
    - Flow time and energy
    - Deadline feasibility and energy
    - Deadline feasibility and temperature

PPT to PDF 1.4

Power and Energy Design Space

	Constant Throughput/Latency		Variable Throughput/Latency
Energy	Design Time	Non-active Modules	Run Time
Active	Logic Design Reduced V_{DD} Sizing Multi-V_{DD}	Clock Gating	DFS, DVS (Dynamic Freq, Voltage Scaling)
Leakage	+ Multi-V_T Stack effect	Sleep Transistors Multi-V_{DD} Variable V_T + Input control	+ Variable V_T

Standard Fixed Speed Processors

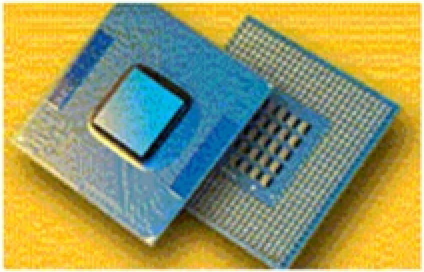
- Historical Intel Pentium processor speeds

CPU	CPU speed
8086	4.77 MHz
80286	12 MHz
80386DX	25 MHz
486 DX2-66	66 MHz
5x86-133	133 MHz
Pentium 75	75 MHz
Pentium 90	90 MHz
Pentium 100	100 MHz
Pentium 133	133 MHz
Pentium 166	166 MHz
Pentium 200	200 MHz

PPT to PDF 1.4

Intel Pentium 4 with Speed Scaling

Mobile Intel® Pentium® 4 Processor - M



Built on 0.13-micron process technology and Intel® NetBurst™ microarchitecture, the Mobile Intel® Pentium® 4 Processor - M provides innovative capabilities for graphics-intensive multimedia applications. It's also excellent for processor-intensive background computing tasks, such as compression, encryption, and virus scanning.

Enhanced Intel SpeedStep® technology helps to optimize application performance and power consumption, and Deeper Sleep Alert State, a dynamic power management mode, adjusts voltage during brief periods of inactivity—even between keystrokes—for longer battery life.

Mobile Intel® Pentium® 4 Processor – M Features

Available Speeds	**2.60 GHz**, 2.50 GHz, 2.40 GHz, 2.20 GHz, 2.0 GHz, 1.80 GHz, 1.70 GHz, 1.60 GHz, 1.50 GHz, 1.40 GHz
Chipset	Mobile Intel® 845 Chipset Family
Cache	512 KB On-Die Level 2 (L2) Cache
RAM	up to 1GB DDR SDRAM
System Frequency Bus	400 MHz

Product I
- Processor
- Specs Upc
- Datasheet
- Performan
- Applicatio
- Design Gu
- Frequently
- Processor
- Technical
- Boxed Mo Processor

Whe

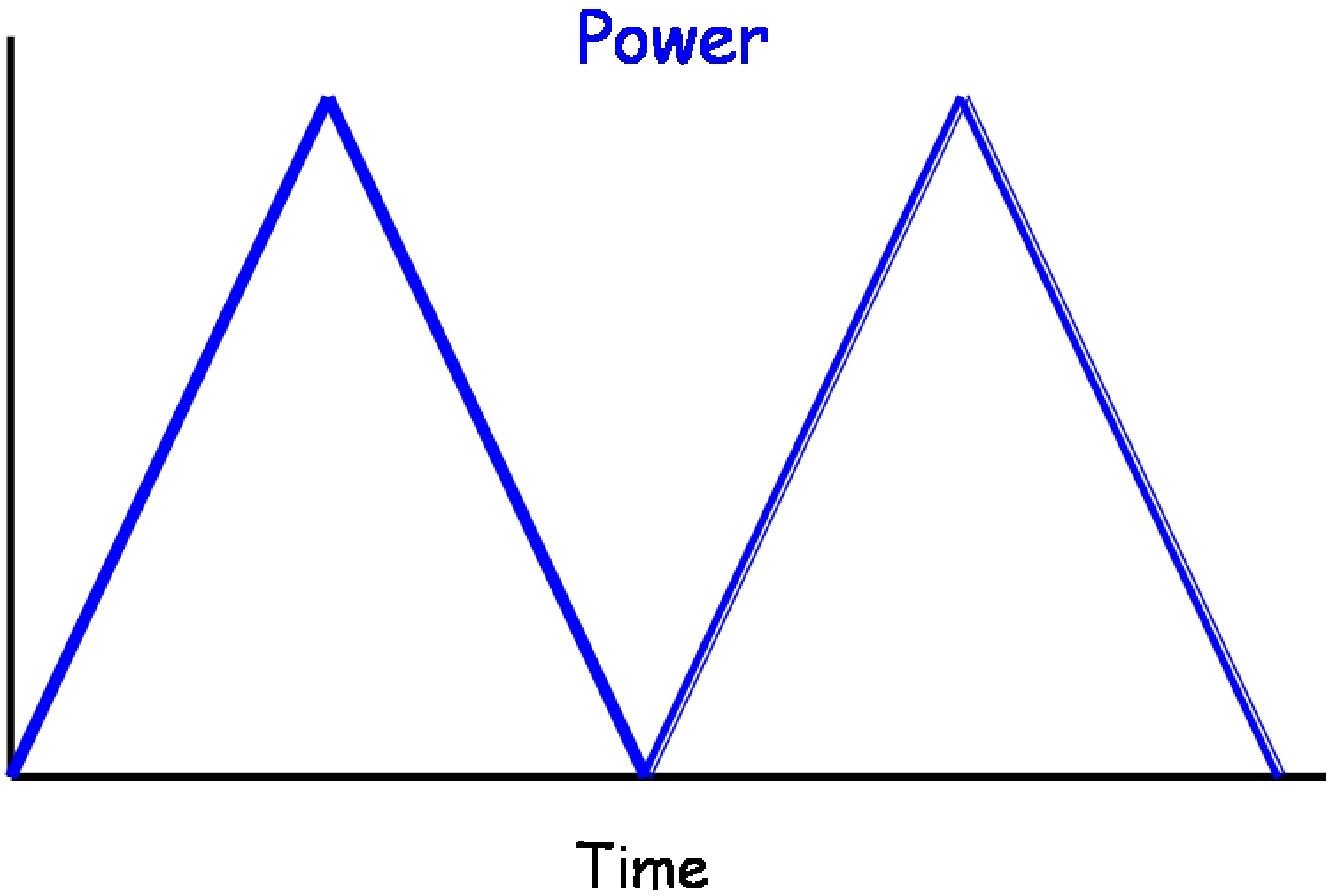
Tools
- Find the R
- Intel® Pro Benchmar
- Compare I

Technic

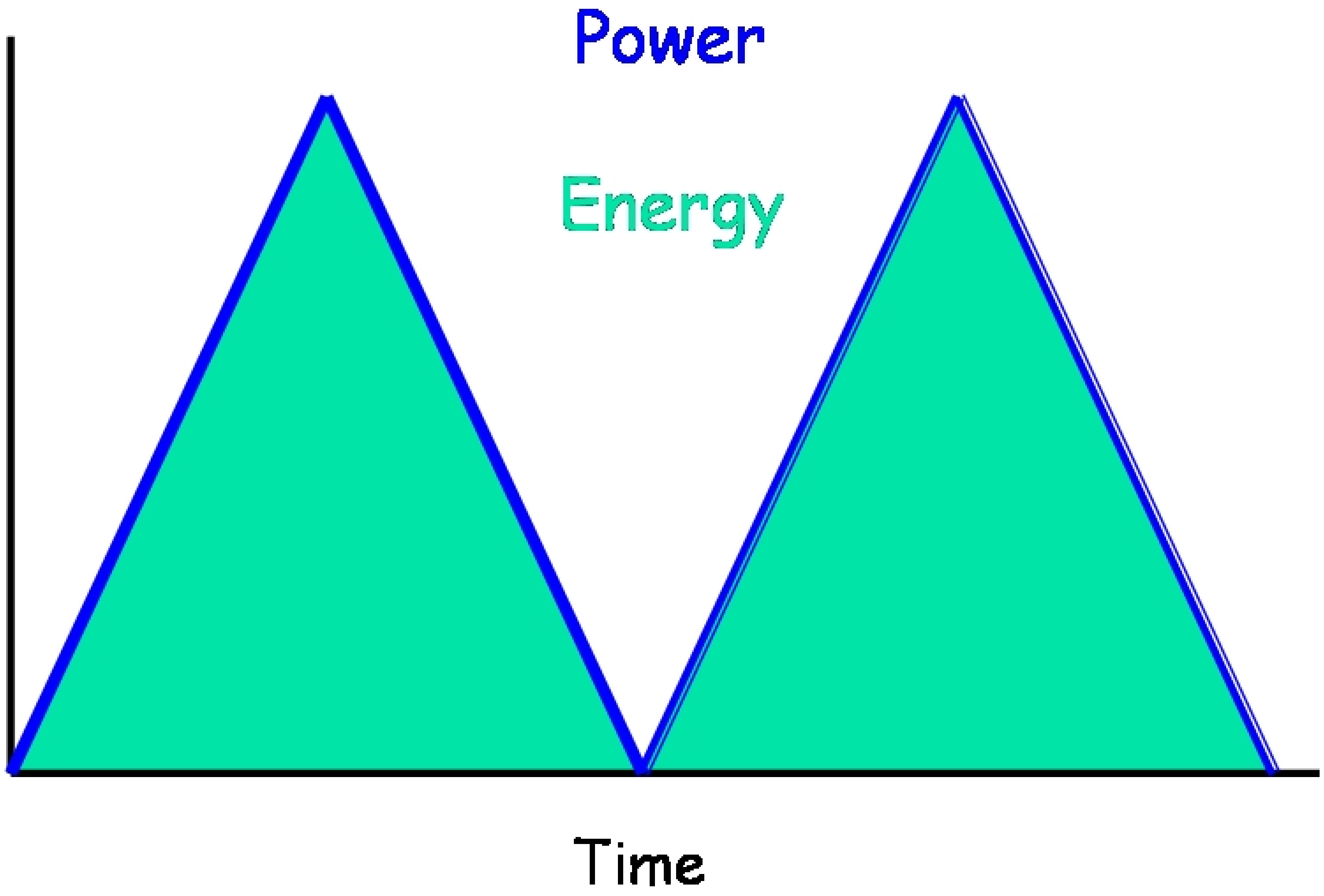
Outline

- Introduction
  - Importance of power management for energy and temperature
  - Speed scaling power management technique
  - Modeling energy and temperature
  - Brief review of scheduling
  - Brief history of the literature

Relationship of Power, Energy, and Speed

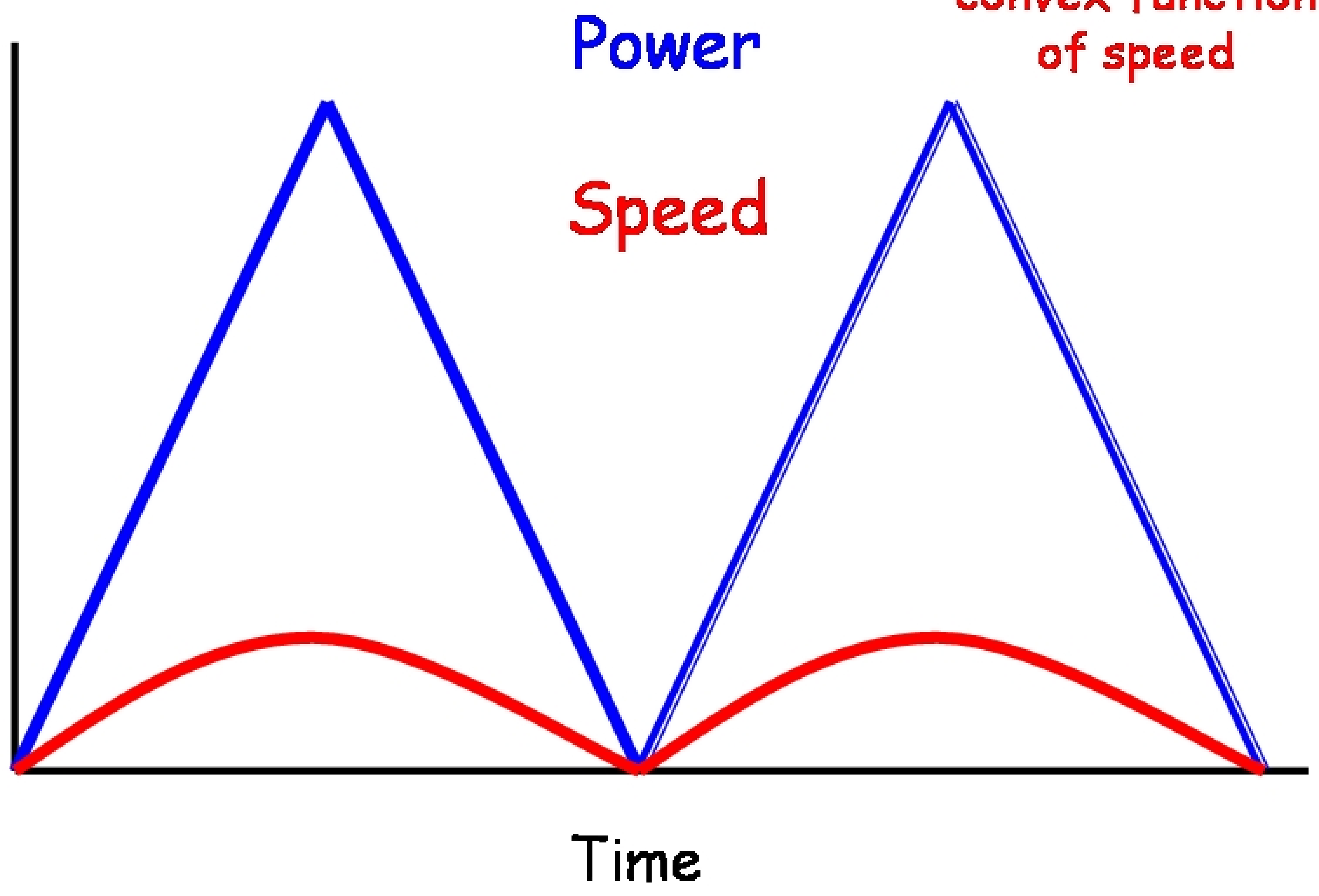


Relationship of Power, Energy, and Speed

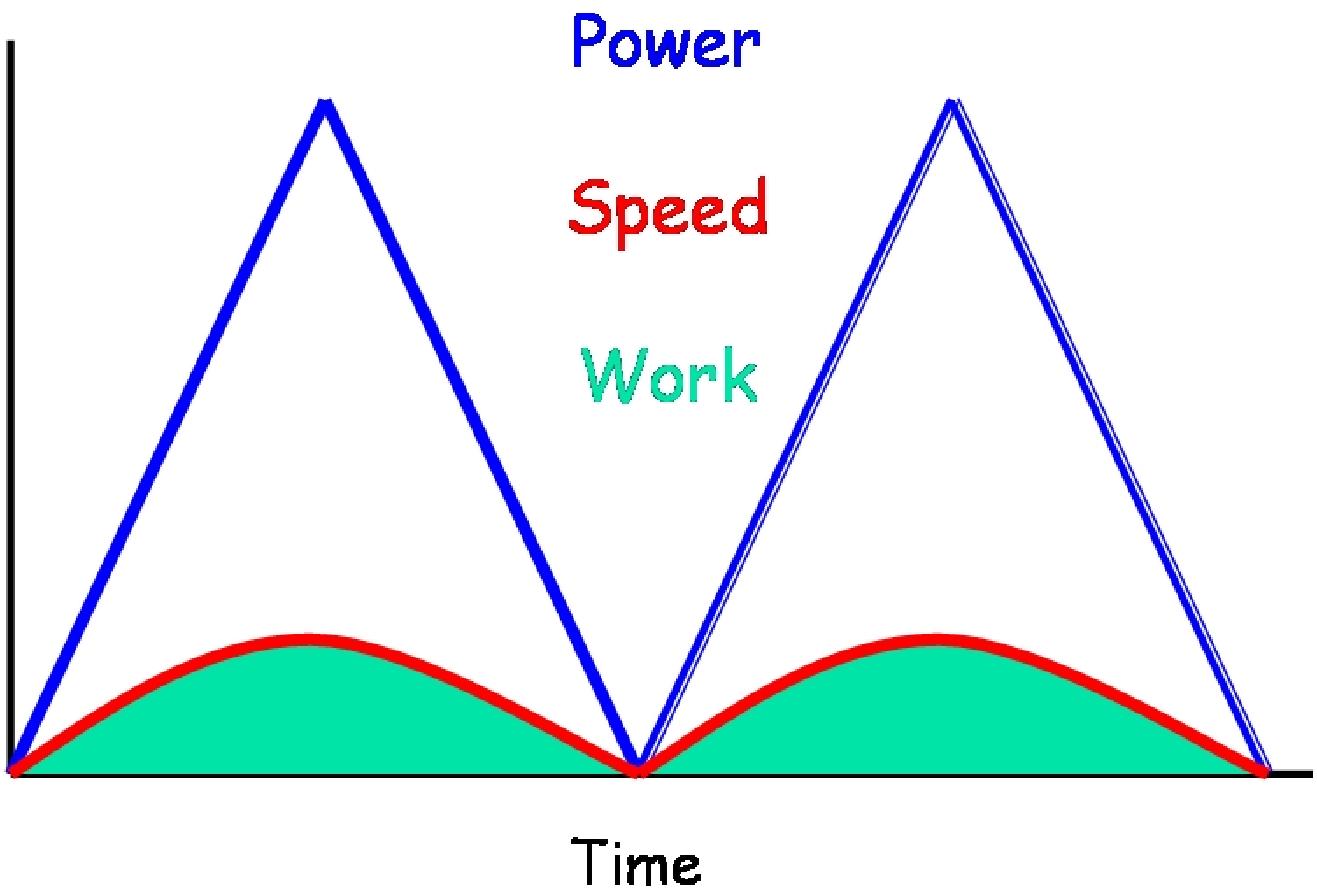


Relationship of Power, Energy, and Speed

- Power is strictly convex function of speed

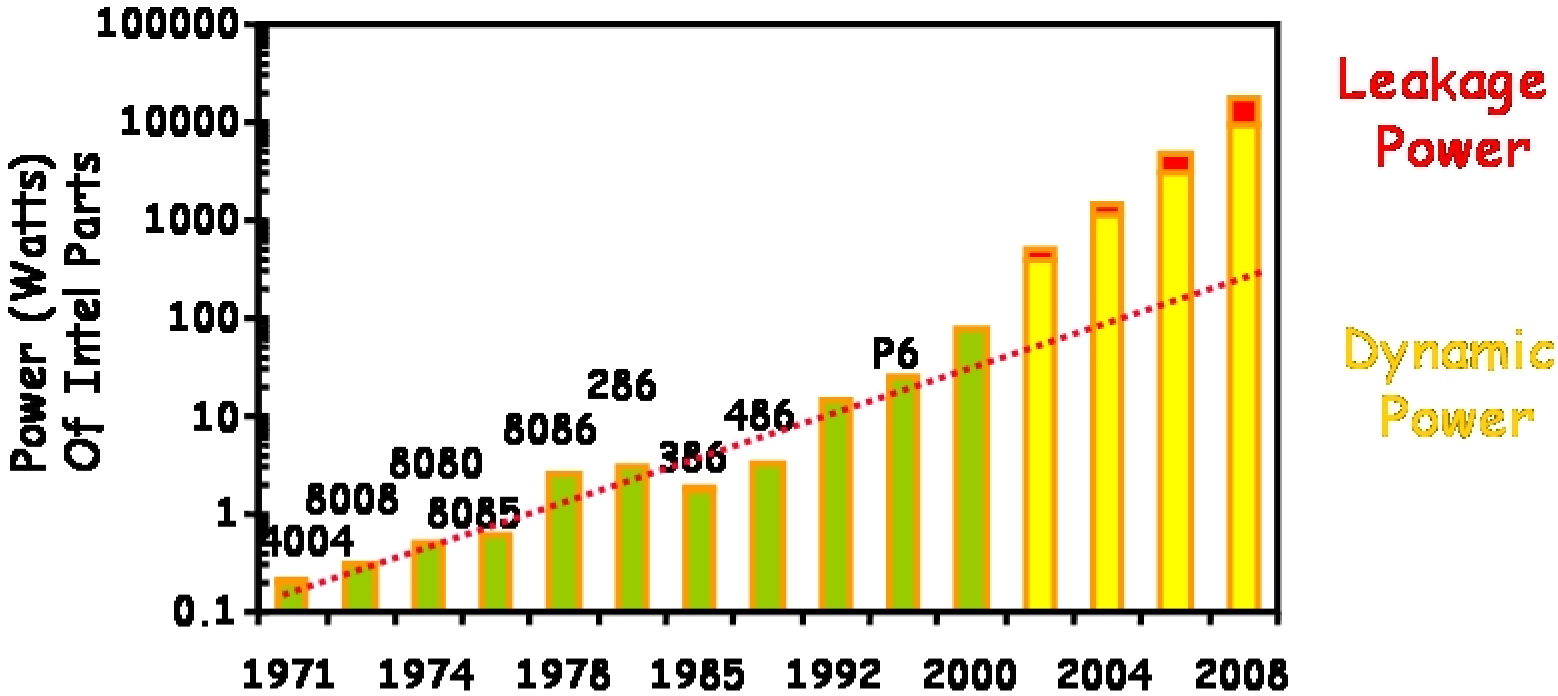


Relationship of Power, Energy, and Speed



Cube Root Rule in CMOS Technologies(1)

- Power P = Energy used per unit time
  - = dynamic power + ~~leakage power~~ → 0
    - Leakage power = power used when idling



Cube Root Rule in CMOS Technologies(2)

- Dynamic Power $P = c\,V^2\,s$
  - V = voltage
  - s = frequency = processor speed
  - c = some constant
- There is a minimum voltage V required to run the processor at speed s, and V is roughly linear in s.
- $P = c\,s^3$
  - Speed is cube root of power

Newton's Law of Cooling(1)

- Key Assumption: fixed ambient temperature T_a
- Newton's Law: rate of cooling is proportional to the temperature difference
- Equation

$$dT/dt = P - b\,(T - T_a) = P - b\,T$$

- T = Temperature
- t = time
- P = supplied power
- b is constant cooling parameter
- For simplicity rescale so that $T_a = 0$

Newton's Law of Cooling(2)

- If supplied power P =0, then temperature decays exponentially with half-life θ(1/b)

$$dT/dt = - b\, T$$

- Theorem: Maximum temperature = Θ(maximum over time intervals I of length 1/b of energy used during I)

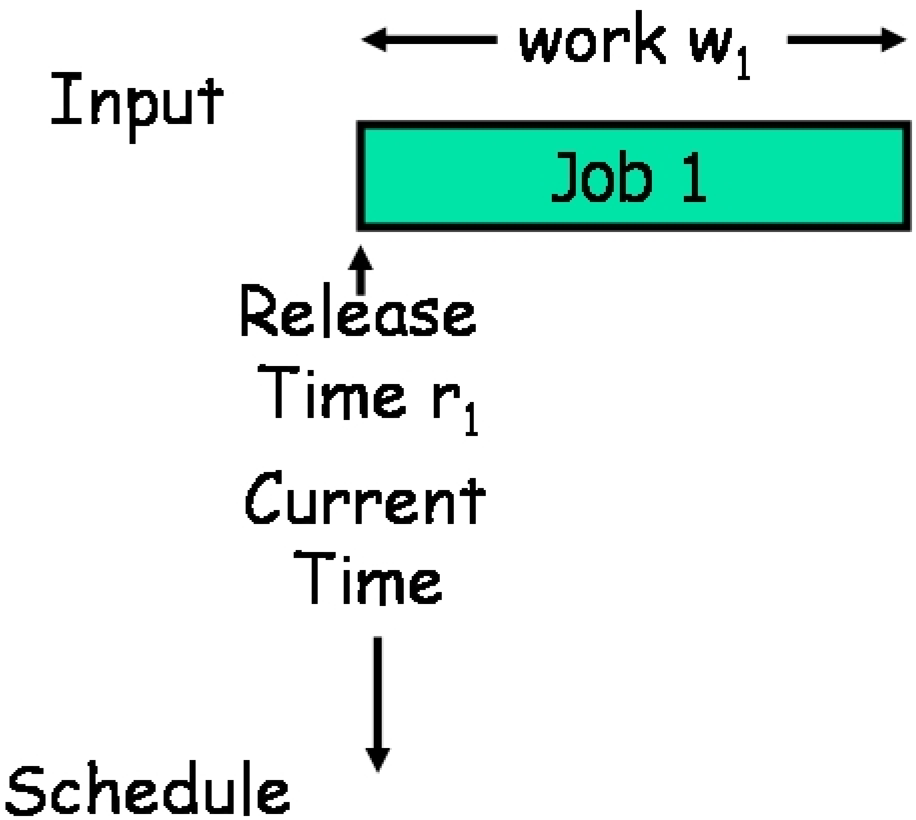
Understanding dT/dt = P - bT

- Theorem: Maximum temperature ~ maximum energy over an interval of length 1/b
- If b=0, then maximum temperature = total energy
- If b=∞, then maximum temperature = maximum power
- Definition: An algorithm is cooling oblivious if O(1)-approximate for temperature for all b
- Theorem: A cooling oblivious algorithm is O(1)-approximate for total energy, maximum power, and maximum speed

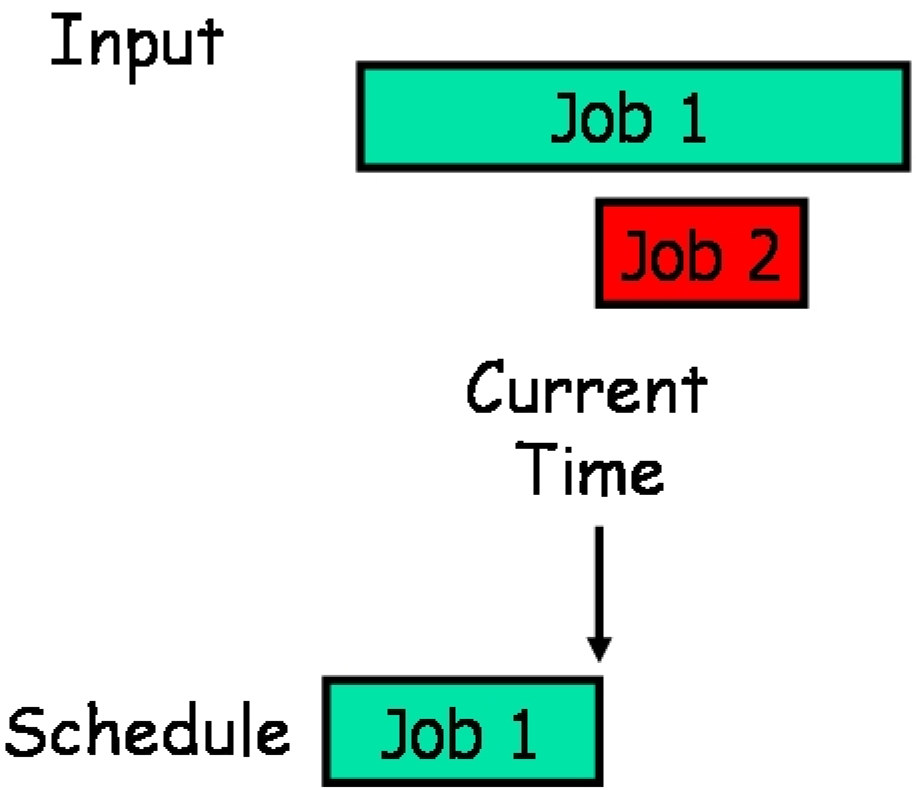
Outline

- Introduction
  - Importance of power management for energy and temperature
  - Speed scaling power management technique
  - Modeling energy and temperature
  - Brief review of scheduling
  - Brief history of the literature

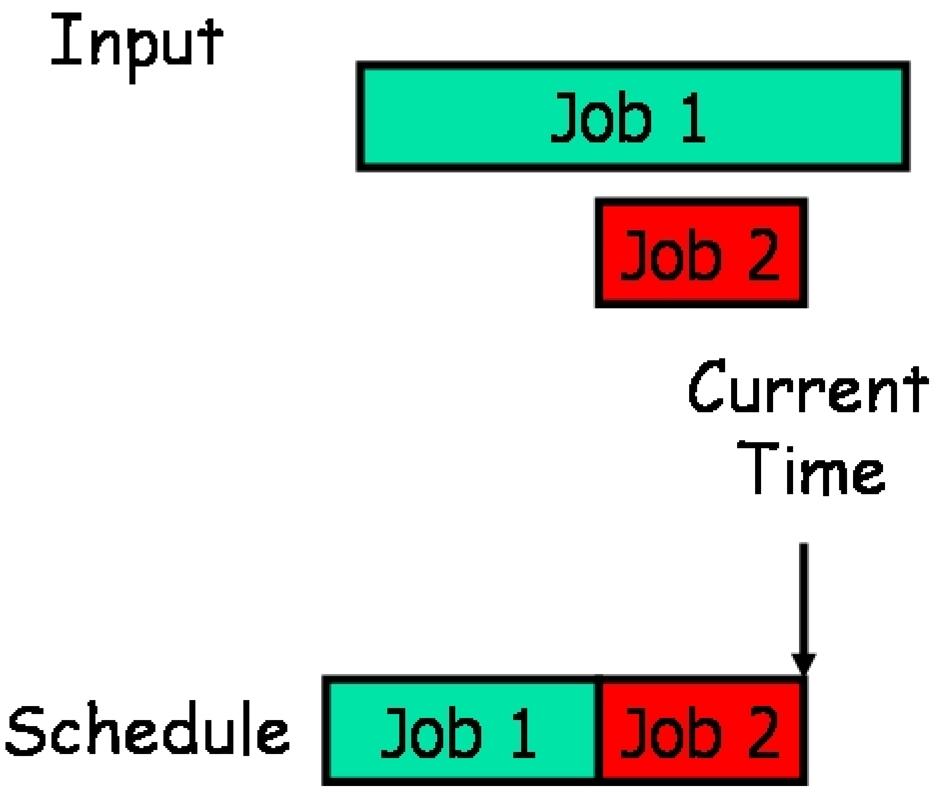
Online Scheduling Without Speed Scaling



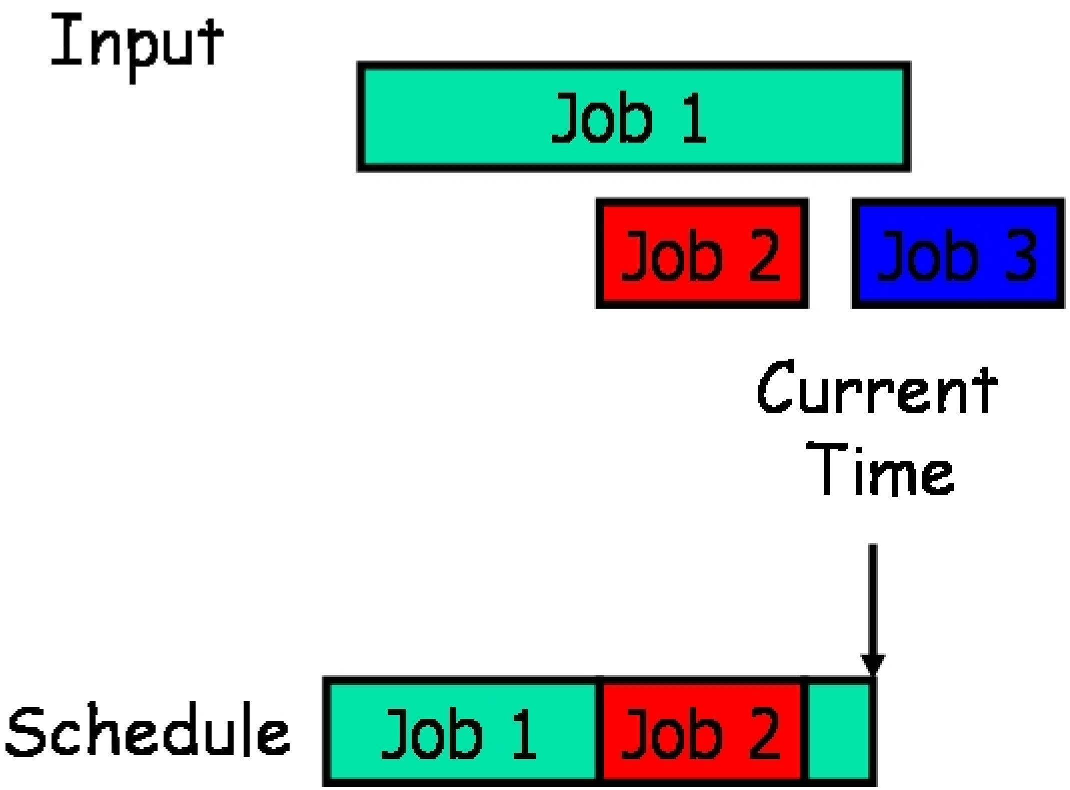
Online Scheduling Without Speed Scaling



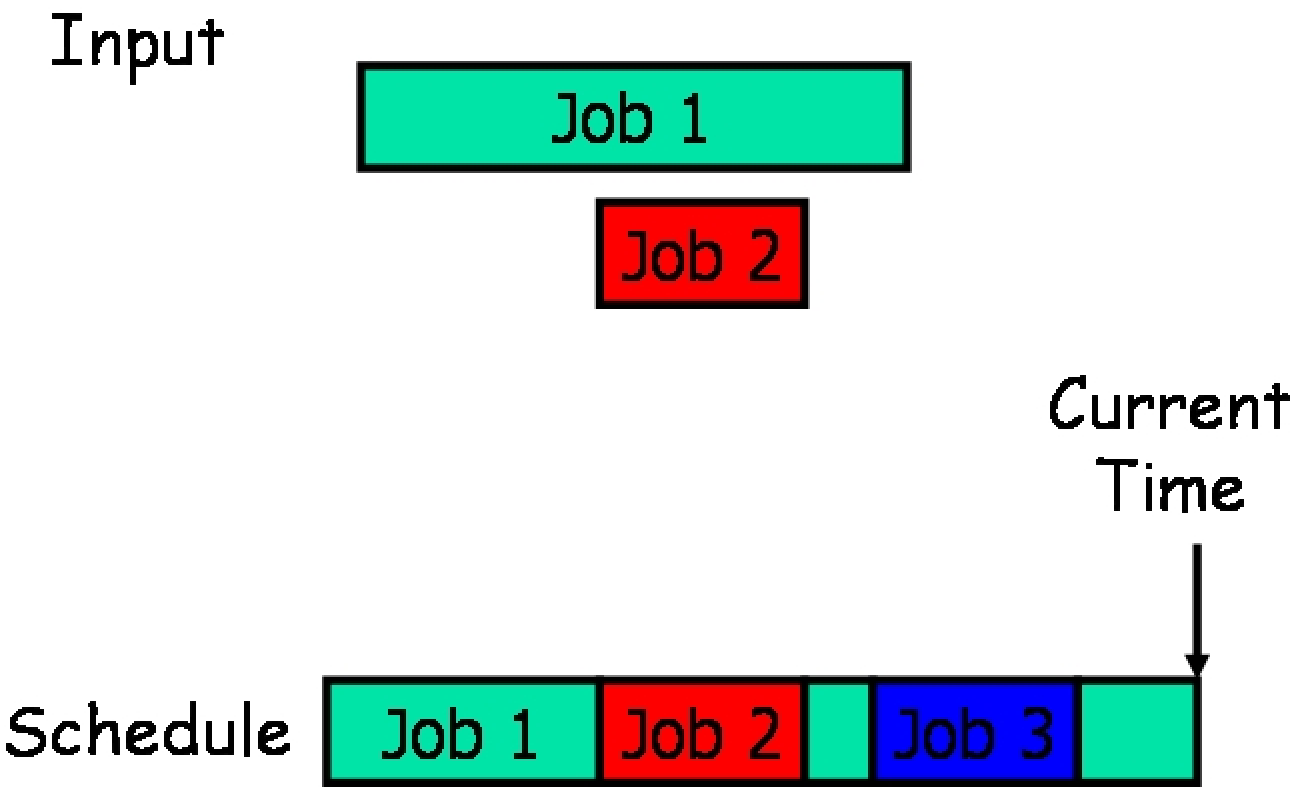
Online Scheduling Without Speed Scaling



Online Scheduling Without Speed Scaling



Online Scheduling Without Speed Scaling



Standard Scheduling Problem Without Speed Scaling

- Find a job selection policy A that optimizes some Quality of Service (QoS) measure of the schedule
- The two QoS measures that we care about here are:
  - Deadline feasibility = each job i finishes by a specified deadline d_i
    - Optimal job selection policy: Earliest Deadline First (EDF)
  - Total (Average) flow time = Sum of flow times of jobs
    - Flow time F_i of a job i is completion time $C_i - r_i$
    - Most common QoS measure in systems literature
    - Optimal job selection policy: Shortest Remaining Processing Time (SRPT)

Outline

- Introduction
  - Importance of power management for energy and temperature
  - Speed scaling power management technique
  - Modeling energy and temperature
  - Brief review of scheduling
  - Brief history of the literature

History

- [YDS95] Frances Yao, Alan Demers, and Scott Shenker, A Scheduling Model for Reduced CPU Energy, FOCS 95. First theoretical paper on energy management.
  - QoS = deadline feasibility
- 2004 - : 10's of papers on speed scaling of jobs with deadlines. Concentrate on the following papers which introduced temperature management.
  - [BKP04] Nikhil Bansal, Tracy Kimbrel, and Kirk Pruhs, Dynamic Speed Scaling to Manage Energy and Temperature, FOCS 2004
  - [BP04] Nikhil Bansal, and Kirk Pruhs, Speed Scaling to Manage Temperature, STACS 2005
- 2004 - : 3 papers on speed scaling for flow time problems
  - [PUW04] Kirk Pruhs, Patchrawat Uthaisombut, and Gerhard Woeginger, Getting the Best Response for Your Erg, SWAT 2004
  - [AF06] Susanne Albers and Hiroshi Fujiwara, Energy-efficient algorithms for flow time minimization , STACS 06.
  - [BPS07] Nikhil Bansal, Kirk Pruhs and Cliff Stein, Speed Scaling for Weighted Flow, SODA 2007

PPT to PDF 1.4

Outline

- Introduction
- Algorithmic results
  - Offline optimal speed scaling algorithms
    - Deadline feasibility and energy
    - Deadline feasibility and temperature
    - Flow time and energy
  - Online speed scaling algorithms
    - Flow time and energy
    - Deadline feasibility and energy
    - Deadline feasibility and temperature

Outline: Offline optimal speed scaling algorithms

- Deadline feasibility and energy
  - Simple greedy algorithm
  - Proof of correctness comes from KKT conditions of convex programming formulation
- Deadline feasibility and temperature
  - Show that Ellipsoid algorithm can be applied to convex programming formulation
- Flow time and energy
  - Restrict to unit work jobs so that we can have a convex programming formulation
  - Show how to trace the optimal flow time schedule as a function of the available energy

Deadline Feasibility and Energy

- Input: A collection of tasks, where task i has
  - Release time r_i when it arrives in the system
  - Deadline d_i when it must finish by
  - Work requirement w_i
- The processor must perform w_i units of work on each task i after time r_i and before time d_i (Preemption is allowed)
- For each time, the scheduler must specify both
  - Job Selection Policy: which job to run
    - wlog, may assume EDF
  - Speed Setting Policy: set speed the processor should run at
- Objective: Minimize the total energy subject to deadline feasibility

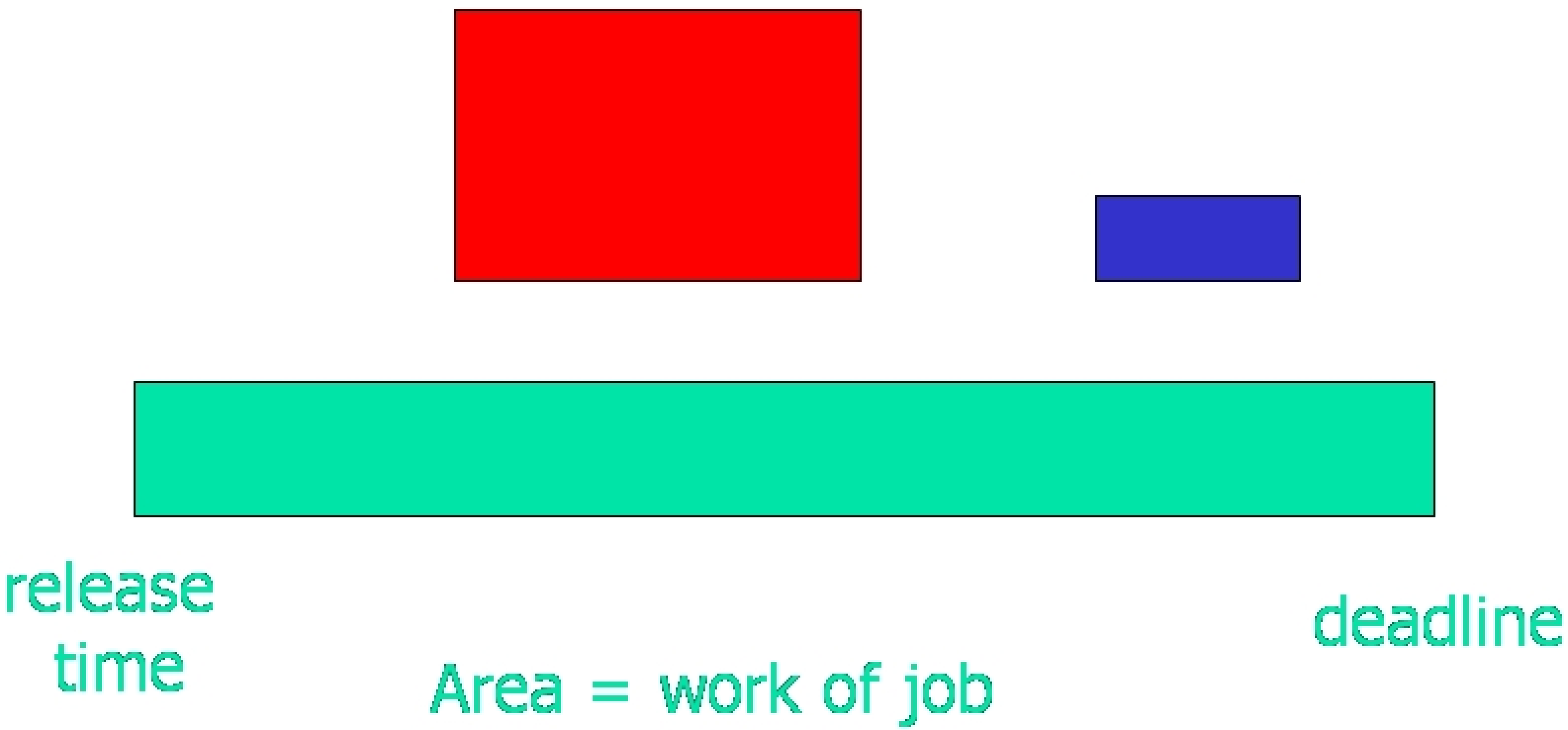
Offline YDS Algorithm [YDS 95]

- Repeat
  - Find the interval I with maximum intensity
    - Intensity of time interval I = $\Sigma\, w_i$ / |I|
      - Where the sum is over tasks i with [r_i, d_i] in I
  - During I
    - speed = the intensity of I
    - earliest deadline first scheduling policy
  - Remove I, and the jobs completed in I

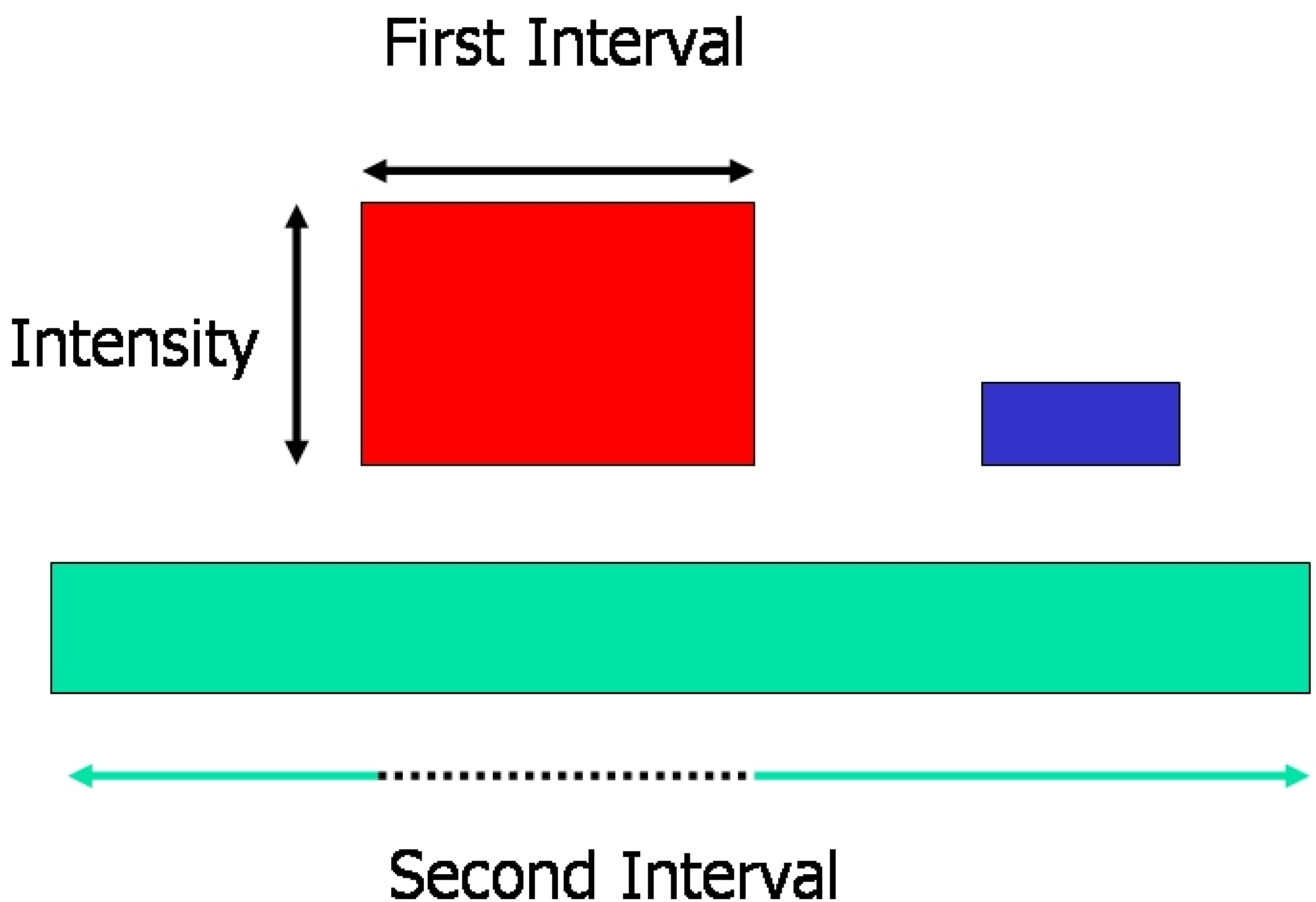
PPT to PDF 1.4

YDS Example(1)

- Input



YDS Example(2)

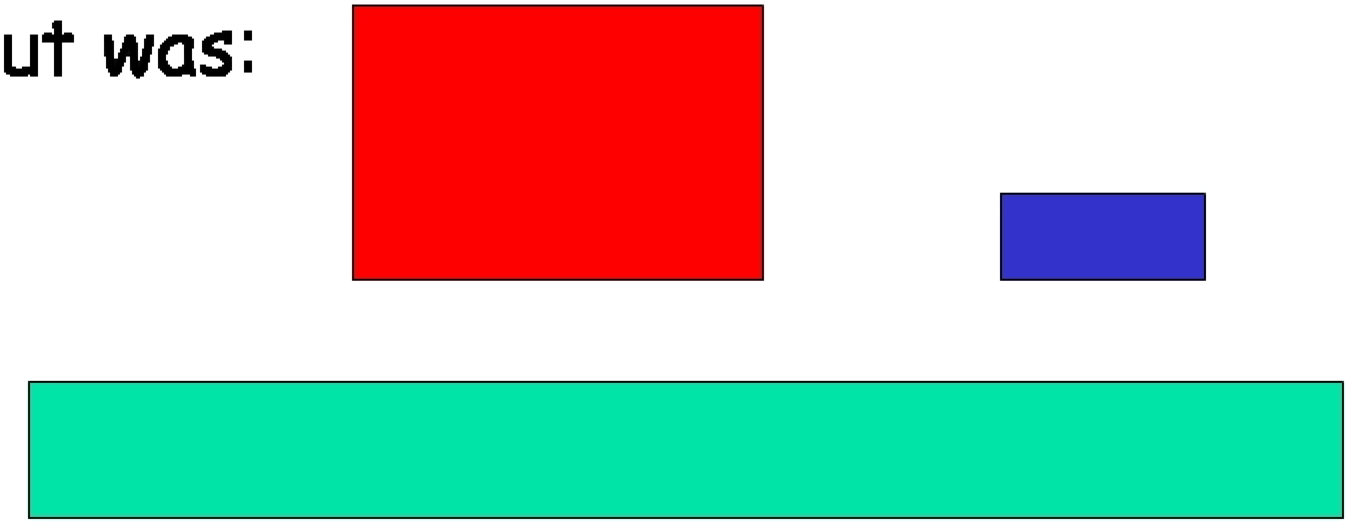


$$\text{Intensity} = \frac{\text{green work} + \text{blue work}}{\text{Length of solid green line}}$$

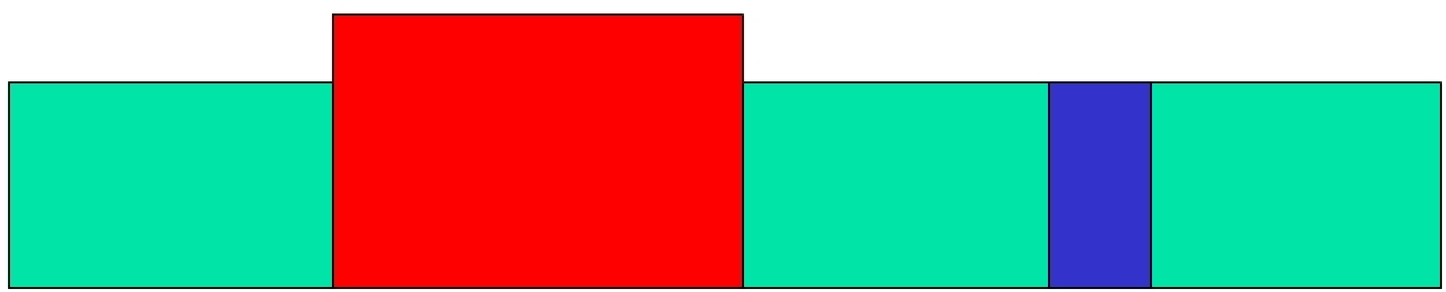
PPT to PDF 1.4

YDS Example(3)

- Recall input was:



- Final YDS Schedule
  - Height = processor speed



YDS Theorems

- Easy Theorem: The YDS schedule is optimal for maximum power ($b=\infty$).
  - Proof: Every schedule must have maximum power equal to the power of the first interval that YDS considers.
- Theorem (YDS95) : The YDS schedule is optimal for energy ($b=0$).
  - *Our Proof (on next slides): A cute consequence of KKT optimality*
- Theorem (BP05): The YDS schedule is cooling oblivious. That is, YDS at worst 20-competitive with respect to temperature for all cooling parameters b
  - Proof idea: If YDS uses a lot of energy over an interval of length 1/b then every schedule uses almost that much energy over some interval of length 1/b

Correctness Proof of YDS Algorithm for Energy [BP05]

Interval Indexed Convex Program

$$
\begin{aligned}
\min \quad & E \\
w_j \;\leq\; & \sum_{i \in J^{-1}(j)} w_{i,j} \qquad j = 1, \ldots, n \\
\sum_{i=1}^{m} \left(\frac{\sum_{j \in J(i)} w_{i,j}}{t_{i+1} - t_i} \right)^p (t_{i+1} - t_i) \;\leq\; & E \\
w_{i,j} \;\geq\; & 0 \qquad i = 1, \ldots, m \qquad j \in J(i)
\end{aligned}
$$

- $W_{i,j}$ = work on job j in interval i
- Interval i = $[t_i, t_{i+1}]$ = maximal time period with no release times or deadlines
- J(i) = jobs that can run in interval i

KKT Optimality Conditions(2)

Consider a strictly-feasible convex differentiable program

$$\begin{aligned} &\min f_0(x) \\ &\quad f_i(x) \le 0 \qquad i = 1, \dots, n \end{aligned}$$

A sufficient condition for a solution x to be optimal is the existence of Lagrange multipliers λ_i such that

$$\begin{aligned} f_i(x) &\le 0 \qquad i = 1, \dots, n \\ \lambda_i &\ge 0 \qquad i = 1, \dots, n \\ \lambda_i f_i(x) &= 0 \\ \nabla f_0(x) + \sum_{i=1}^{n} \lambda_i \nabla f_i(x) &= 0 \end{aligned}$$

Introducing Lagrange Multipliers

Interval Indexed Convex Program

$$\min \quad E$$

$$\alpha_j \qquad w_j \le \sum_{i \in J^{-1}(j)} w_{i,j} \qquad j = 1, \ldots, n$$

$$\beta \qquad \sum_{i=1}^{m} \left(\frac{\sum_{j \in J(i)} w_{i,j}}{t_{i+1} - t_i} \right)^p (t_{i+1} - t_i) \le E$$

$$\gamma_{i,j} \qquad w_{i,j} \ge 0 \qquad i = 1, \ldots, m \qquad j \in J(i)$$

KKT Optimality Conditions (3)

- The $w_{i,j}$ component of the gradient equation is

$$-\alpha_j + \beta p \left(\frac{\sum_{k \in J(i)}^{n} w_{i,k}}{t_{i+1} - t_i} \right)^{p-1} - \gamma_{i,j} = 0$$

- If $w_{i,j} > 0$ (that is, job j is run in interval i) then $\gamma_{i,j} = 0$ by complementary slackness. Hence,

$$\alpha_j = p \left(\frac{\sum_{k \in J(i)}^{n} w_{i,k}}{t_{i+1} - t_i} \right)^{p-1}$$

- Therefore, a task j is run at the same speed s_j in every interval in which it is run.
- Note $a_j = p\ s_j^{(p-1)}$

PPT to PDF 1.4

KKT Optimality Conditions (4)

- If $w_{i,j} = 0$ then

$p\, s_j^{(p-1)}$

$$\gamma_{i,j} = p \left(\frac{\sum_{k \in J(i)}^{n} w_{i,k}}{t_{i+1} - t_i} \right)^{p-1} - \alpha_j$$

- This has a solution with $\gamma_{i,j} >= 0$ if the speed that the processor is run during interval i is $>= s_j$
- Since YDS satisfies these conditions, it is optimal

Outline: Offline optimal speed scaling algorithms

- Deadline feasibility and energy
  - Simple greedy algorithm
  - Proof of correctness comes from KKT conditions of convex programming formulation
- Deadline feasibility and temperature
  - Show that Ellipsoid algorithm can be applied to convex programming formulation
- Flow time and energy
  - Restrict to unit work jobs so that we can have a convex programming formulation
  - Show how to trace the optimal flow time schedule as a function of the available energy

Offline Speed Scaling to Minimize Temperature

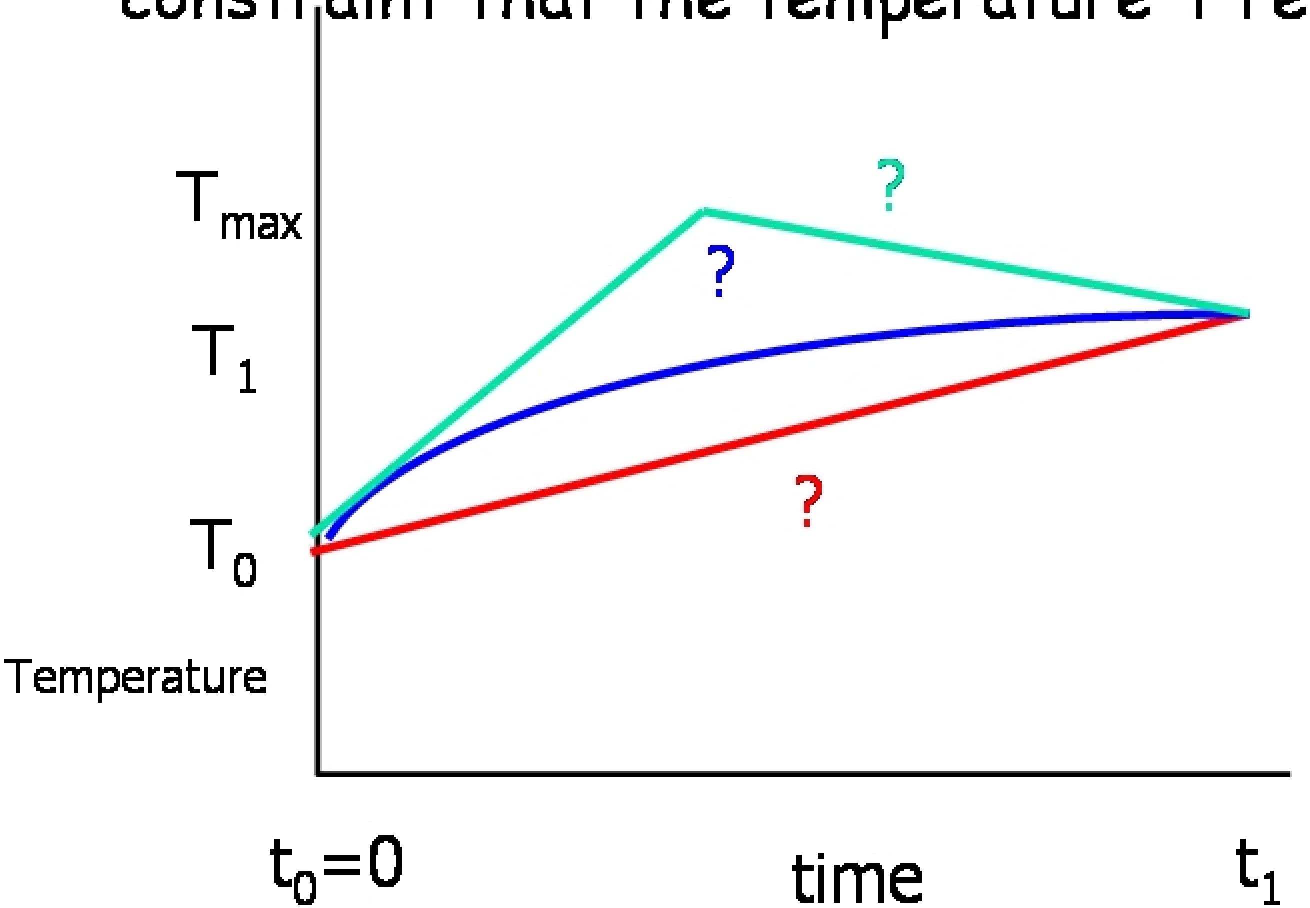
- Convex program formulation of determining whether temperature T_{max} is feasible

$$
\begin{aligned}
w_j &\le \sum_{i:j \in J(i)} w_{i,j} \qquad 1 \le j \le n \\
\sum_{j \in J(i)} w_{i,j} &\le MaxW(t_i, t_{i+1}, T_i, T_{i+1}) \qquad 1 \le i \le m-1 \\
0 &\le T_i \qquad 1 \le i \le m \\
0 &\le w_{i,j} \qquad 1 \le i \le m, 1 \le j \le n
\end{aligned}
$$

- $MaxW(t_i, t_{i+1}, T_i, T_{i+1})$ = maximum work that can be accomplished during time interval $[t_i, t_{i+1}]$, starting at temperature T_i, ending at temperature T_{i+1}, maintaining the invariant that $T \le T_{max}$
- To apply Ellipsoid algorithm we need to be able to find subgradient of $MaxW(t_i, t_{i+1}, T_i, T_{i+1})$ constraints

MaxW Subproblem

- You start at time t_0 with temperature T_0 and want to end at time t_1 with temperature T_1. What is the maximum work you can accomplish subject to the constraint that the temperature T remains $\leq T_{max}$?



Solution without Boundary Constraint $T \le T_{max}$ (1)

- You want to find the T that maximizes maximum work
  - $W = \int s \, dt$
    - By cube root rule $P = s^3$
    - So $W = \int P^{1/3}$
    - Recall temperature equation $dT/dt = P - b\,T = s^3 - b\,T$
  - $W = \int ((dT/dt + bT)/a)^{1/3} \, dt$
- By fundamental theorem of calculus of variations, T satisfies
  - $F_T = d\,F_{T'}/dt$
    - Functional $F(T, T') = ((dT/dt + bT)/a)^{1/3}$
    - $T' = dT/dt$
    - F_T = the partial of F with respect to T
    - $F_{T'}$ = the partial of F with respect to T'

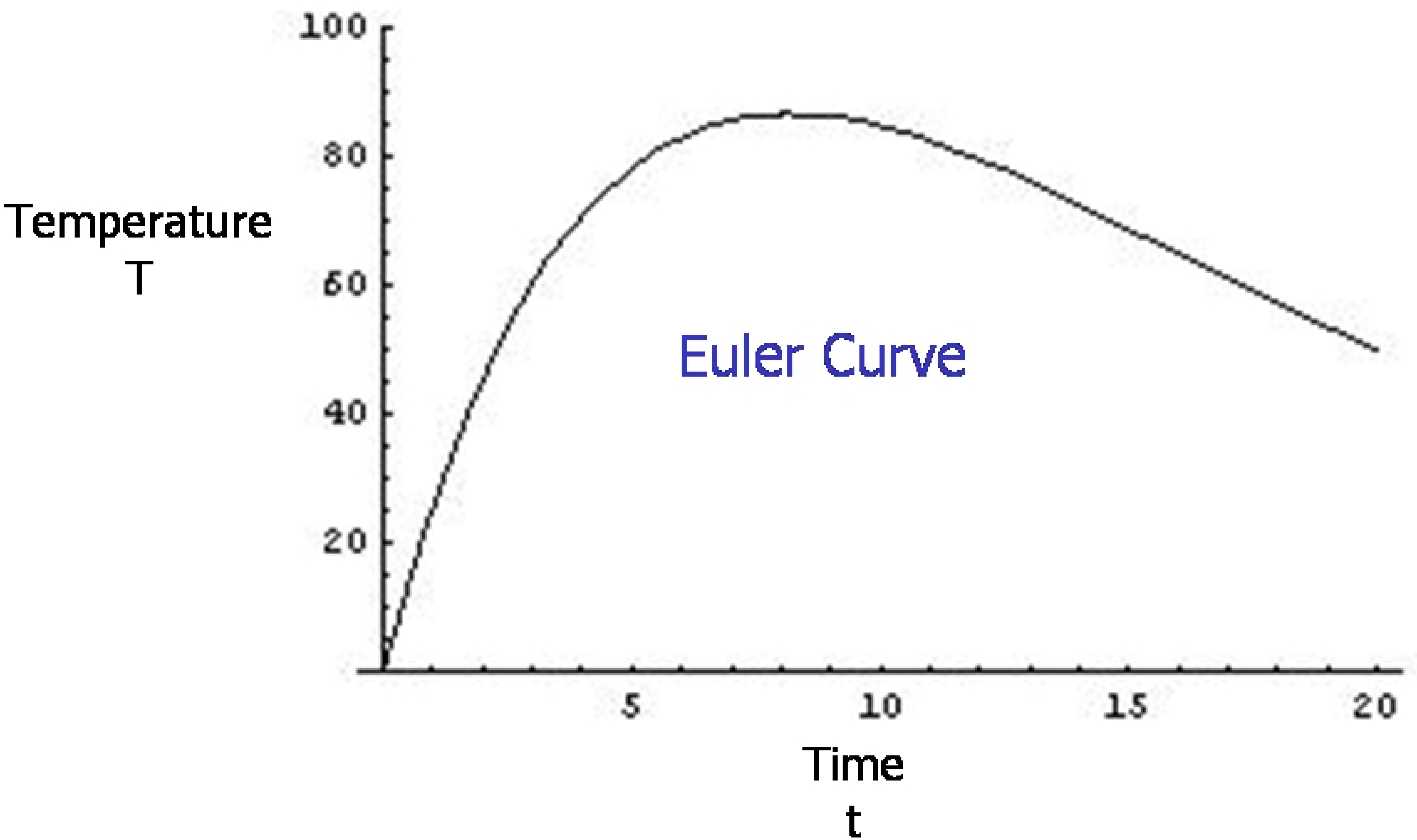
Solution without Boundary Constraint $T \leq T_{max}$ (2)

- Evaluating $F_T = d\, F_{T'} / dt$ and solving for T we get
- $T = c \exp(-bt) + d \exp(-3bt/2)$
  - Where $d = (T_0 \exp(-bt_1) - T_1)/(\exp(-bt_1) - \exp(-3bt_1/2))$
  - and $c = T_0 - d$ and are constants determined from the boundary conditions
- Plugging T back into the integral $\int ((dT/dt + bT)/a)^{1/3}\, dt$ we get
  - $\text{Max Work} = (-4d/ab^2)^{1/3} (1-\exp(-bt_1/2))$

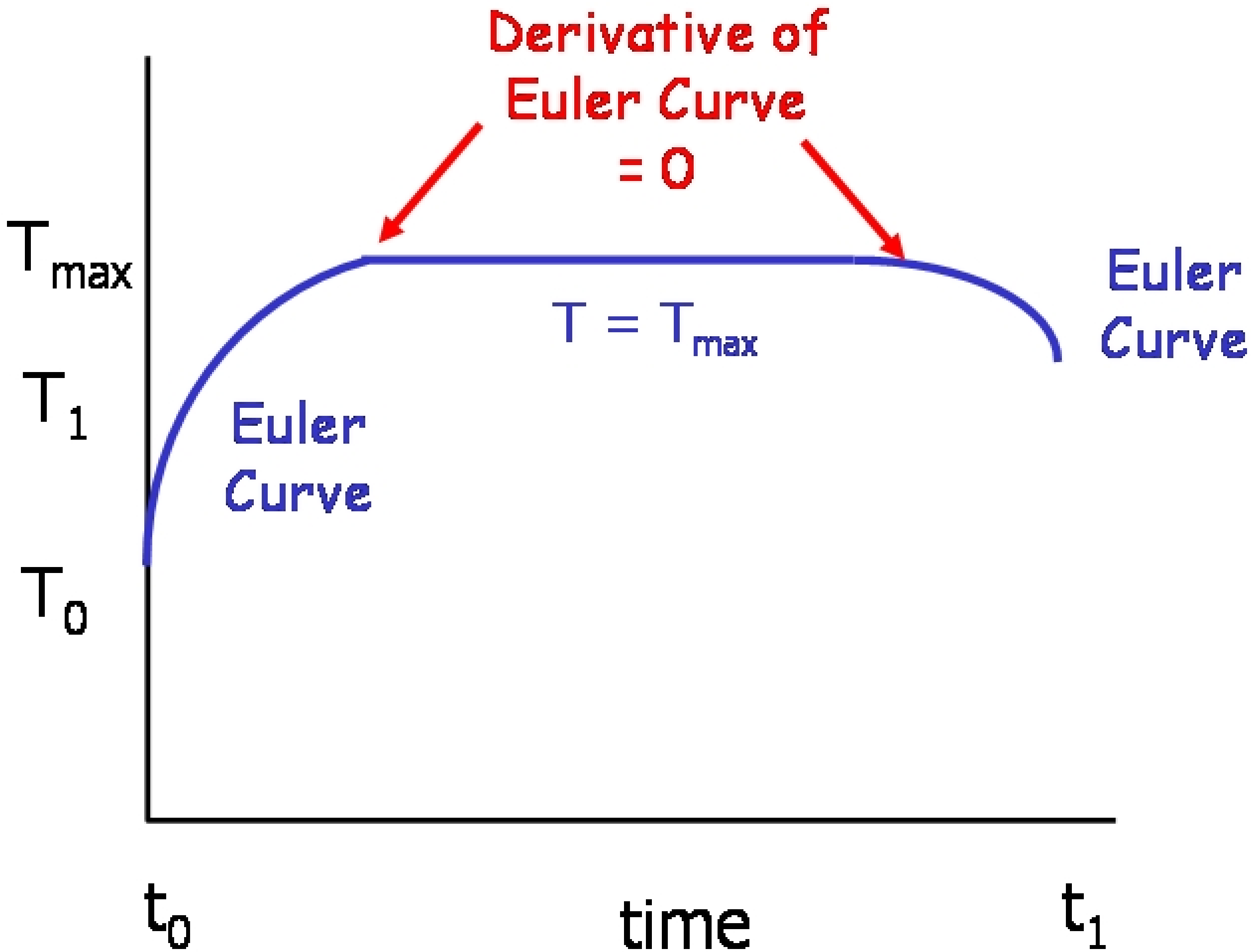
PPT to PDF 1.4

Solution without Boundary Constraint $T \leq T_{max}$ (3)

- $b = .1$, $T_0 = 0$, $t_1 = 20$, and $T_1 = 50$



Solution with Boundary Constraint $T \leq T_{max}$



Outline: Offline optimal speed scaling algorithms

- Deadline feasibility and energy
  - Simple greedy algorithm
  - Proof of correctness comes from KKT conditions of convex programming formulation
- Deadline feasibility and temperature
  - Show that Ellipsoid algorithm can be applied to convex programming formulation
- Flow time and energy
  - Restrict to unit work jobs so that we can have a convex programming formulation
  - Show how to trace the optimal flow time schedule as a function of the available energy

QoS measure = Total Flow Time

- Flow time f_i of a job i is completion time $C_i - r_i$
- Minimize total/average flow time subject to the constraint that at most E energy is used
- We make the simplifying assumptions that all jobs have the same (unit) amount of work
  - In this case the optimal job selection policy is First Come First Served.
  - We thus focus on speed setting policy.
- wlog assume, $r_1 \le r_2 \le \ldots \le r_n$

Convex Programming Formulation

$$\min \sum_{i=1}^{n} C_i$$

$$\sum_{i=1}^{n} \frac{1}{x_i^{\alpha-1}} \leq E$$

$$C_{i-1} + x_i \leq C_i$$

$$r_i + x_i \leq C_i$$

- x_i = execution time of task i

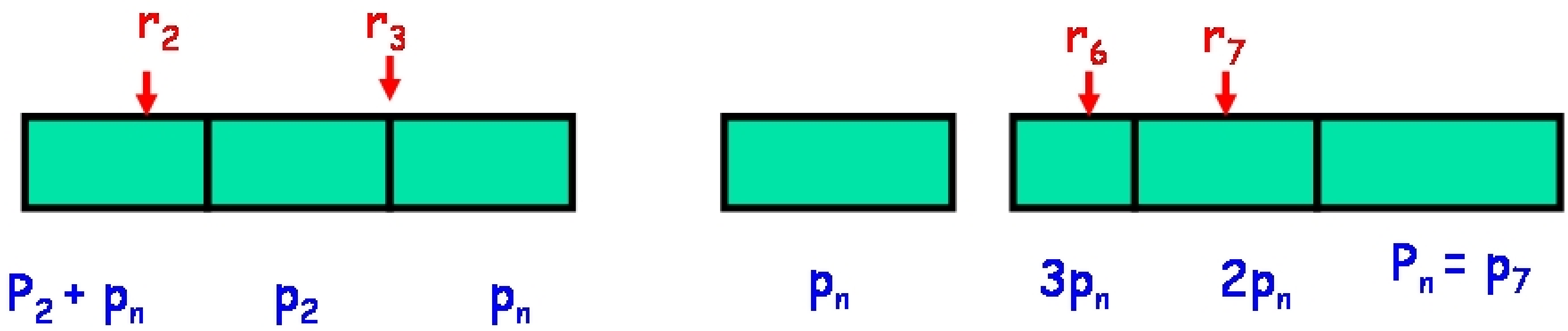
KKT Optimality Conditions

- Total energy of E is used
- $C_i < r_{i+1}$ implies $\rho_i = \rho_n$
  - ρ_i = power of task i
- $C_i > r_{i+1}$ implies $\rho_i = \rho_{i+1} + \rho_n$
- $C_i = r_{i+1}$ implies $\rho_n \le \rho_i \le \rho_{i+1} + \rho_n$

PPT to PDF 1.4

KKT Optimality Conditions

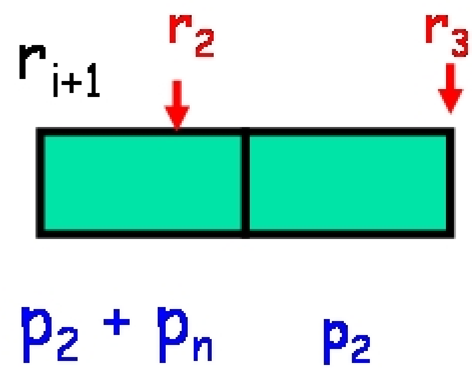
- Total energy of E is used
- $C_i < r_{i+1}$ implies $\rho_i = \rho_n$
- $C_i > r_{i+1}$ implies $\rho_i = \rho_{i+1} + \rho_n$
- $C_i = r_{i+1}$ implies $\rho_n \leq \rho_i \leq \rho_{i+1} + \rho_n$
- Example:



PPT to PDF 1.4

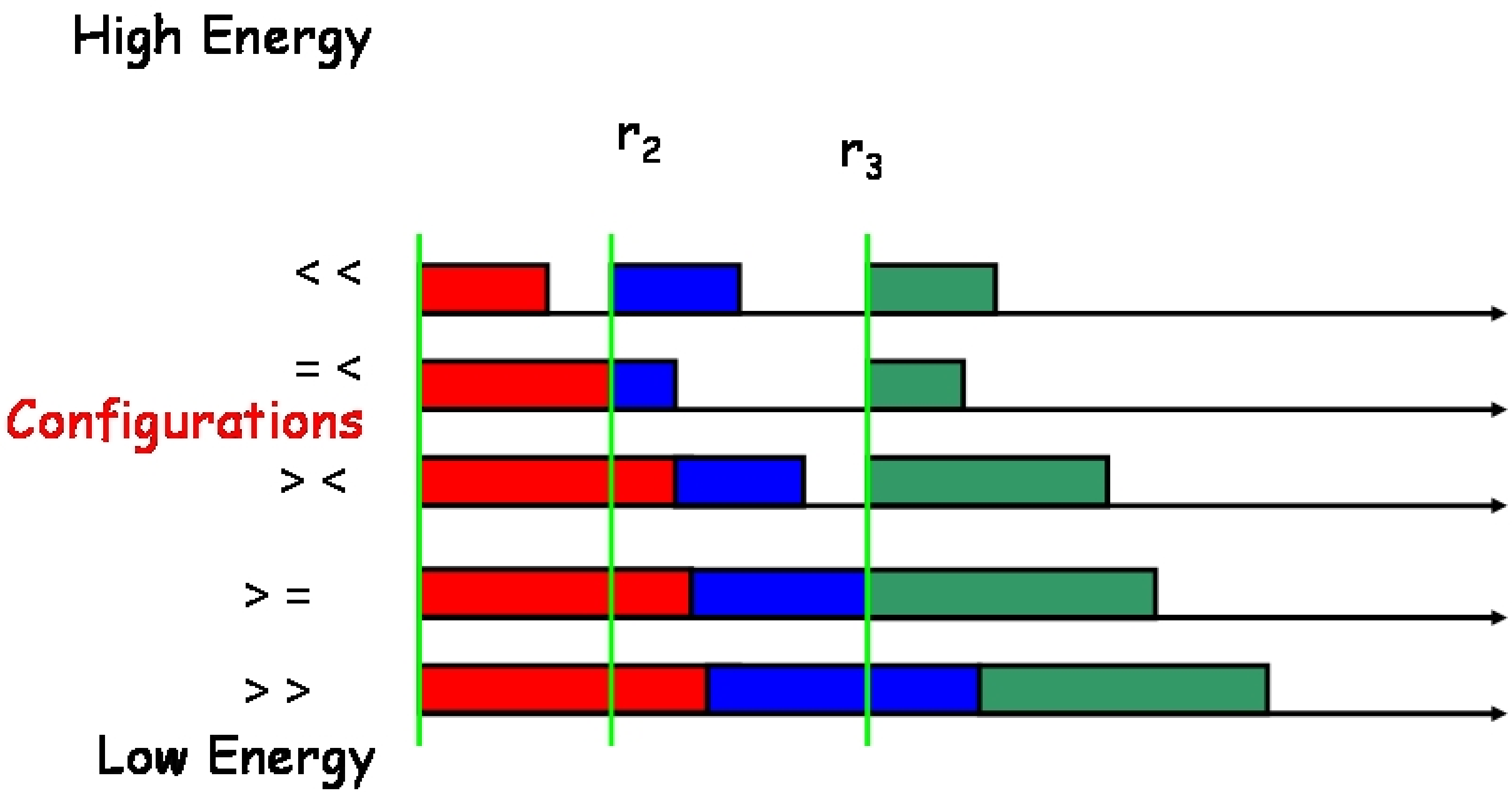
KKT Optimality Conditions

- Algorithmic Difficulties:
  - This doesn't tell us the value of ρ_n
    - Solution: Binary search
  - Don't know the value of p_i when $C_i = r_{i+1}$
    - Solution: Can calculate since you know interval when job runs

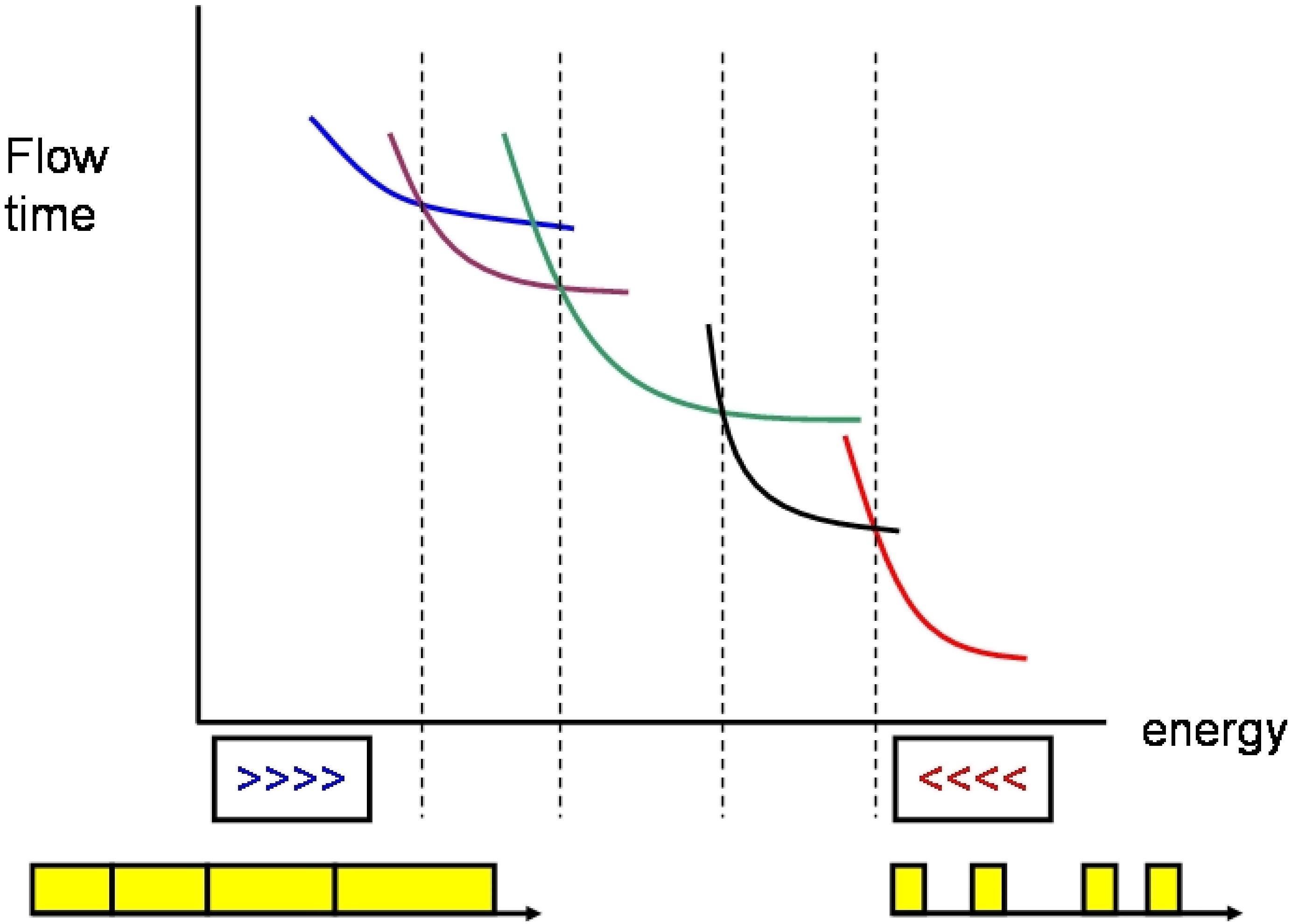


  - Don't know if $C_i < r_{i+1}$, $C_i = r_{i+1}$, or $C_i > r_{i+1}$
    - Easy for high energy E, $C_i < r_{i+1}$
    - Solution: Trace out optimal schedules as E decreases

Configurations

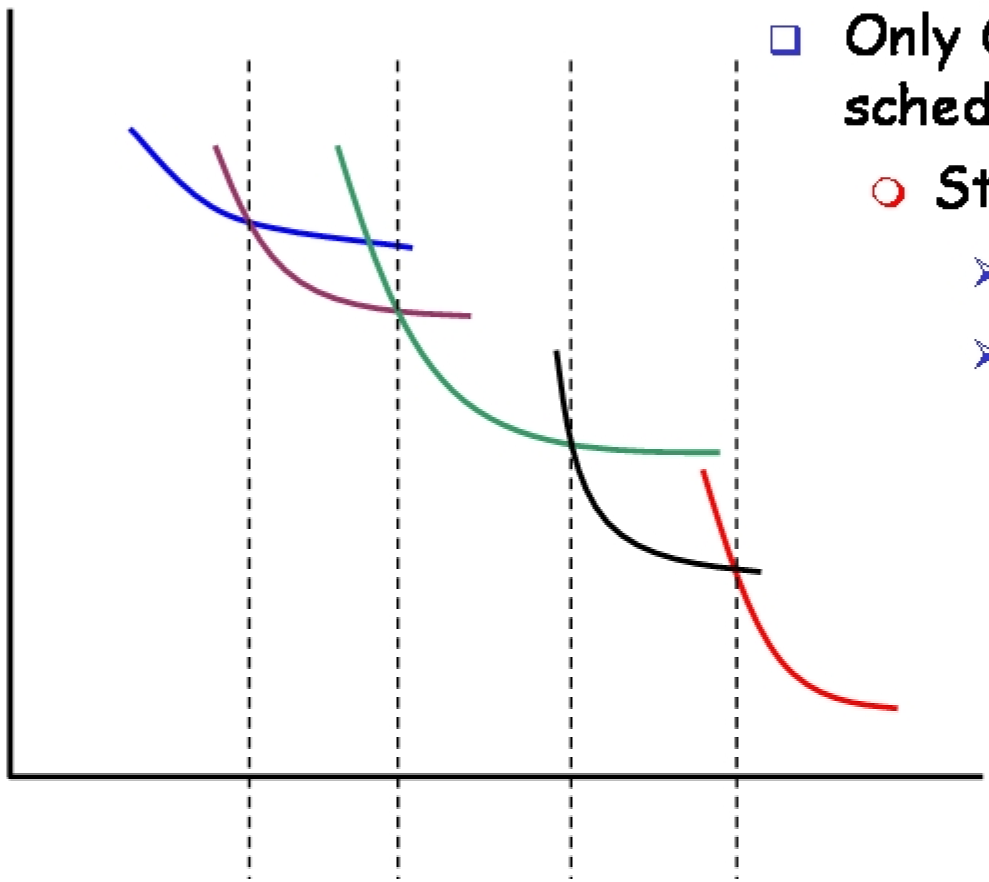


One Curve For Each Configuration



$\tilde{O}(n^2)$ Time Algorithm

- Decrease ρ_n (or equivalently energy), keeping track of the schedule until the energy used is ≤ E
  - Saving Grace: The schedule is a continuous function of ρ_n

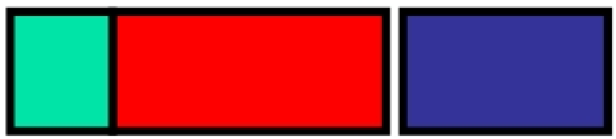


- Only O(n) structural changes in schedule
  - Structural changes are either
    - a C_i becoming = to r_{i+1},
    - or a C_i becoming > than r_{i+1}

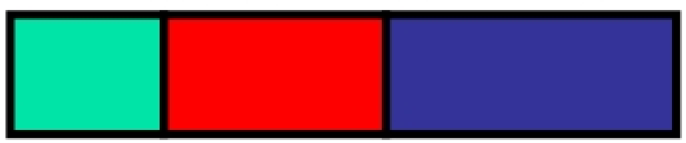
PPT to PDF 1.4

Intuition

- Intuitively as you lose energy, jobs should run slower, but this intuition is false
- Example:
  - Higher energy: $p_1=2p_3$ and $p_2 = p_3$

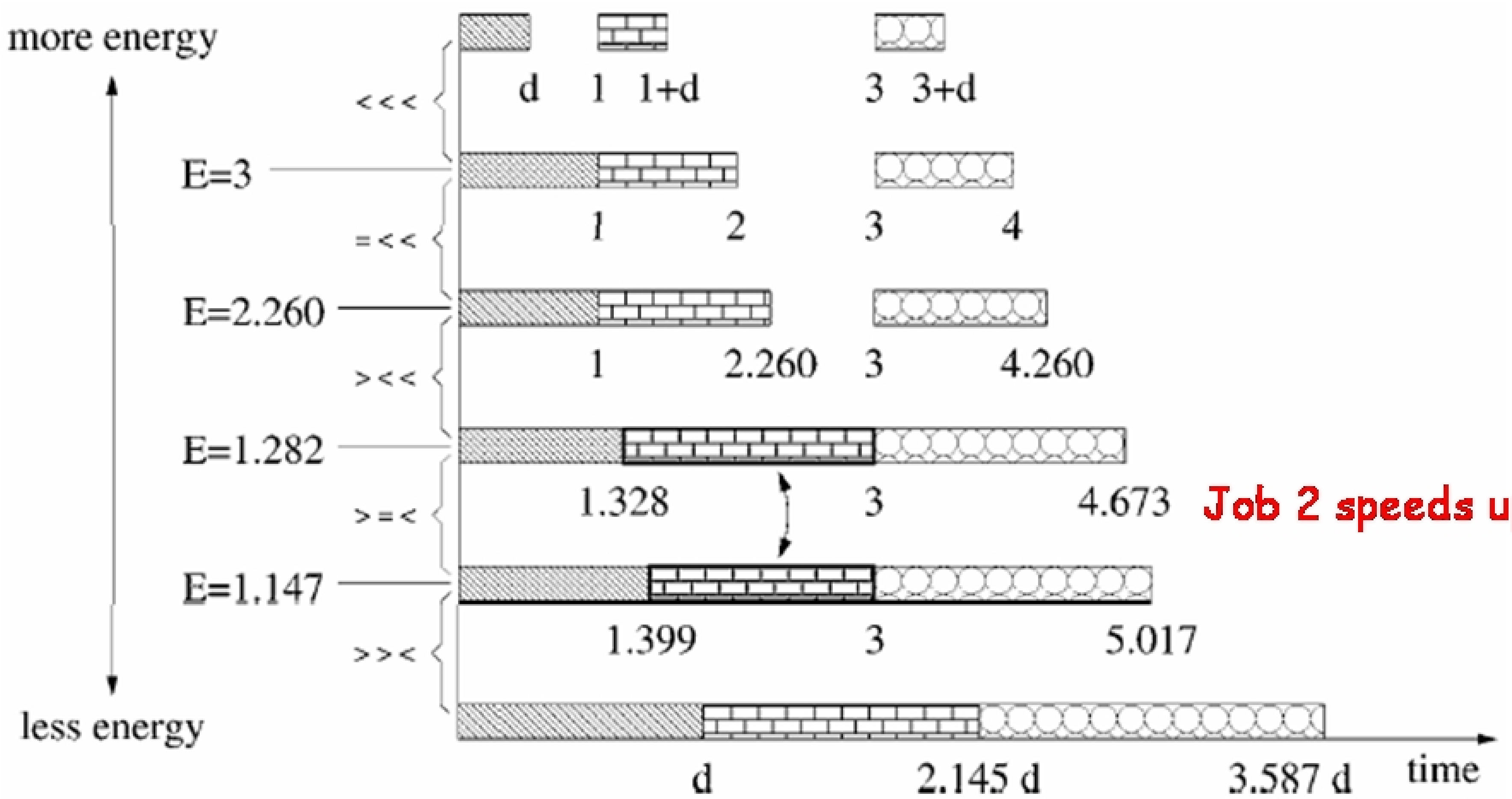


  - Lower energy: $p_1 = 3p_3$ and $p_2 = 2\ p_3$



  - p_1/p_2 decreases and job 2 speeds up as we lose energy

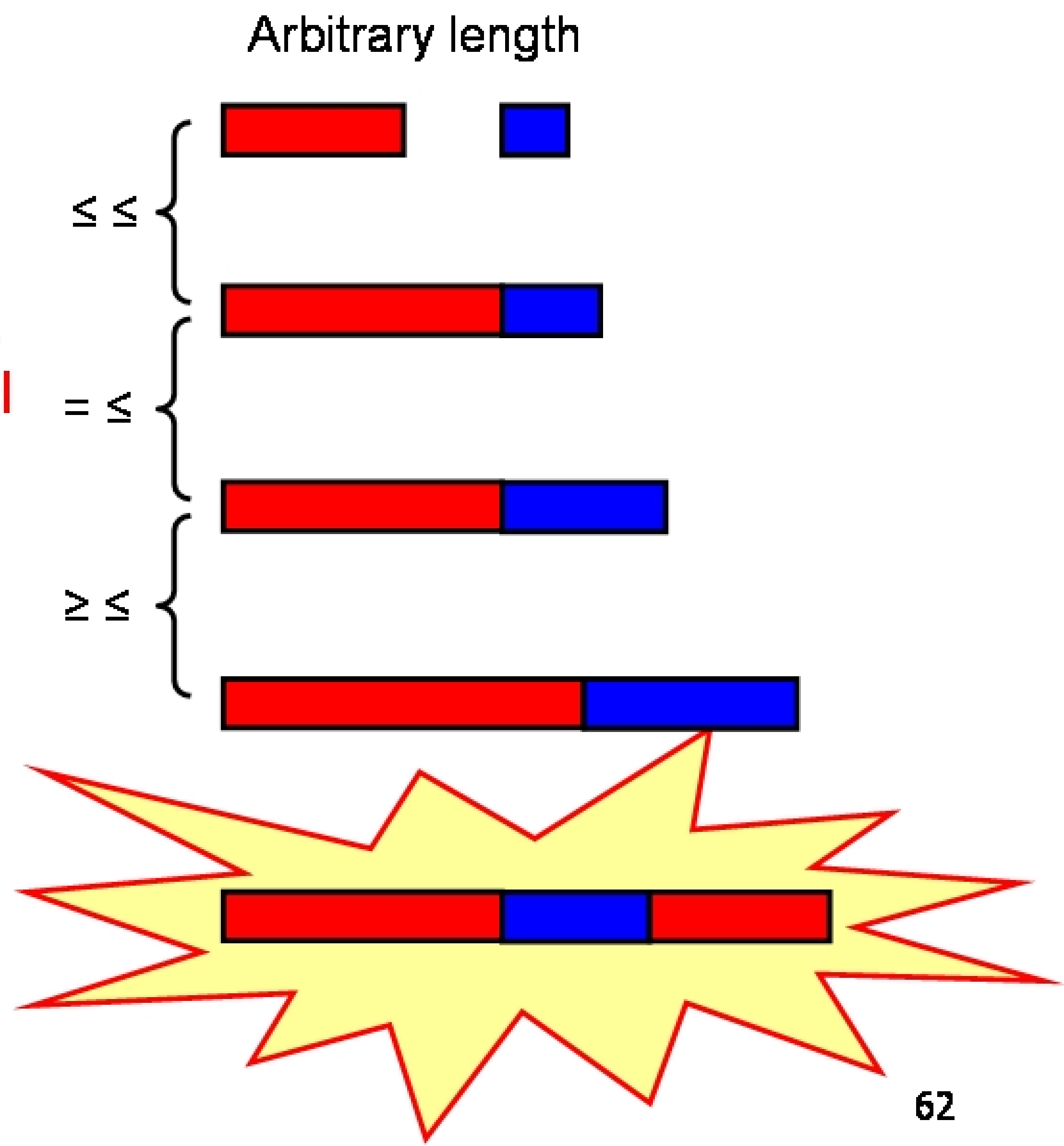
A Concrete Example



Job 2 speeds up

PPT to PDF 1.4

What Goes Wrong With Arbitrary Work Jobs



Open Question: What is the complexity of finding optimal flow time schedules when jobs have arbitrary work?

Optimal scheudule is not a continuous function of energy E

Outline

- Introduction
- Algorithmic results
  - Offline optimal speed scaling algorithms
  - Online speed scaling algorithms
    - Flow time and energy
    - Deadline feasibility and energy
    - Deadline feasibility and temperature

Outline: Online speed scaling algorithms

- Online speed scaling algorithms
  - Review
    - Competitiveness
    - Local competitiveness
    - Resource augmentation
    - Amortized local competitiveness
  - Flow time and energy
  - Deadline feasibility and energy
  - Deadline feasibility and temperature

Competitive Analysis

- Competitive ratio of algorithm A =

$$\max_I A(I)/Opt(I)$$

  - A(I) is the total flow time on input I using algorithm A
  - Opt(I) is the total time for the optimal schedule
- An algorithm with a competitive ratio of 2 means that it guarantees flow time at most 2 times optimal on all inputs

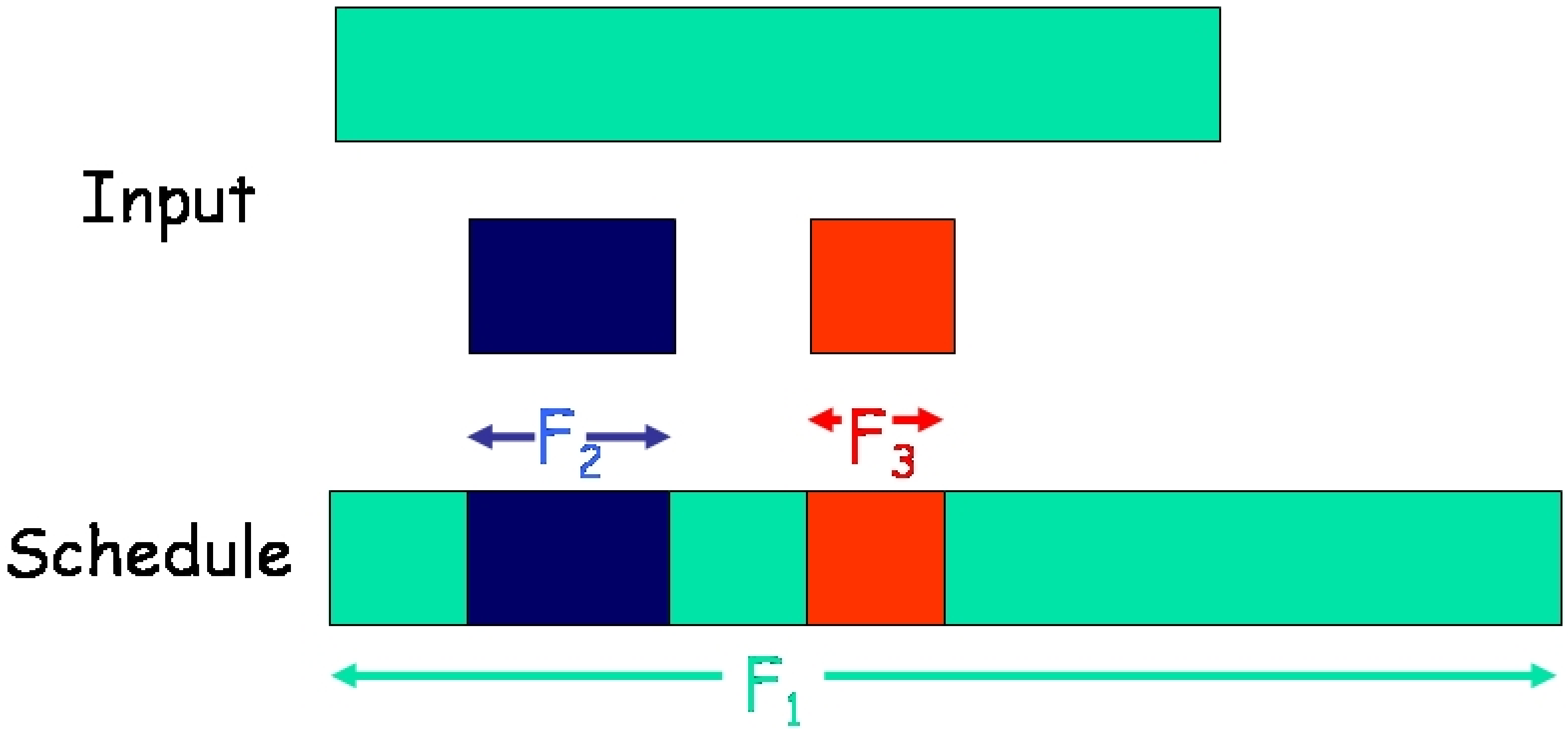
Standard Local Competitiveness Analysis to Prove Competitiveness

- Standard local competitiveness analysis technique:
  - Show that at all times, the increase in the objective function G for the candidate algorithm A is competitive with the increase in the objective function for an arbitrary algorithm Opt

$$\frac{dG_A(t)}{dt} \leq \gamma \frac{dG_{Opt}(t)}{dt}$$

  - γ is competitive ratio

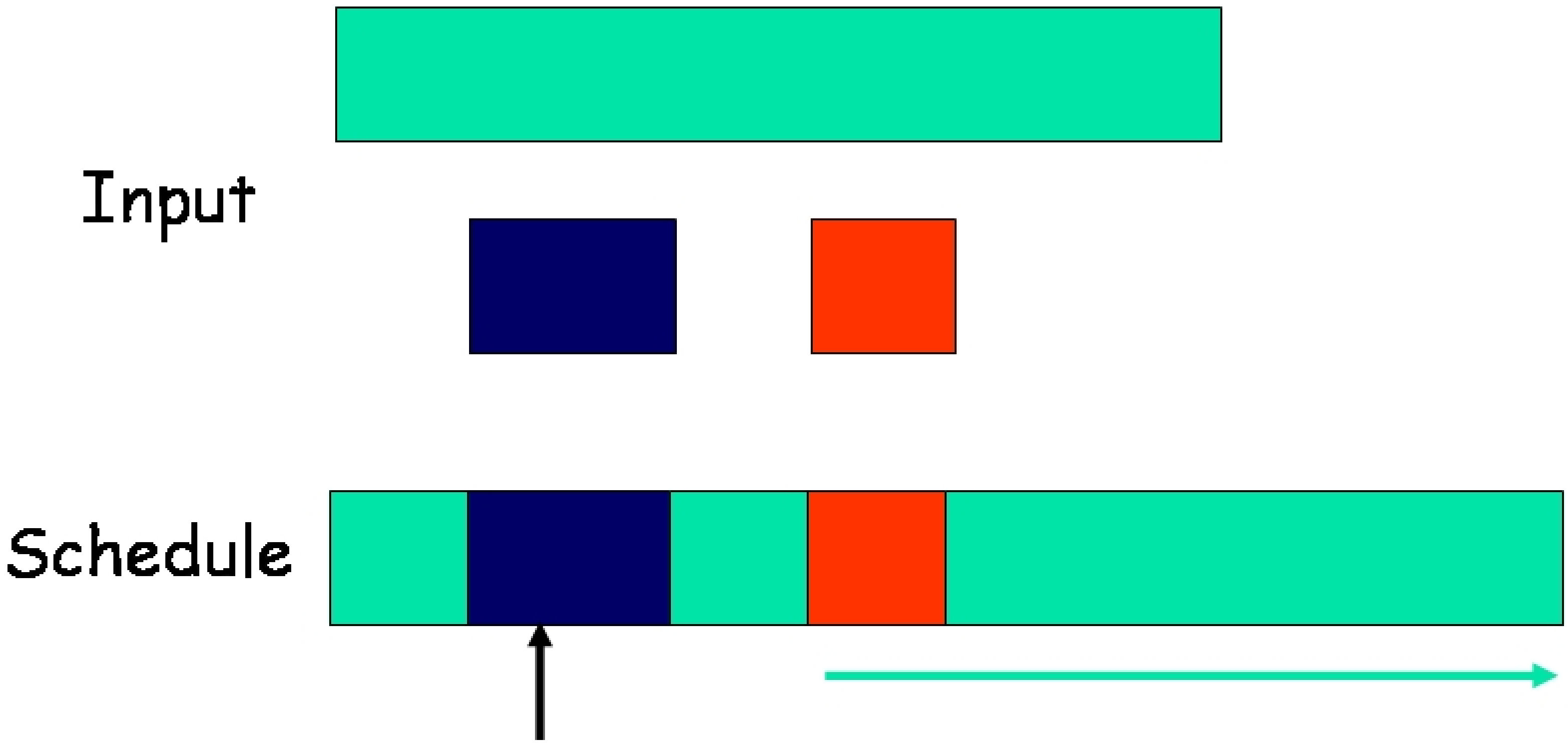
Total Flow Time Objective



- $F_1 + F_2 + F_3 = \int_t$ number of unfinished jobs at time t dt
- Increase in total flow objective = number of alive jobs

PPT to PDF 1.4

Total Fractional Flow Time Objective



$n_t = 2/3 + .9$

- n_t = fractional unfinished jobs at time t
    - A job that is 1/3 finished counts 2/3 toward nt
- Fractional flow = $\int_t n_t \, dt$
- Increase in fractional flow objective = n_t

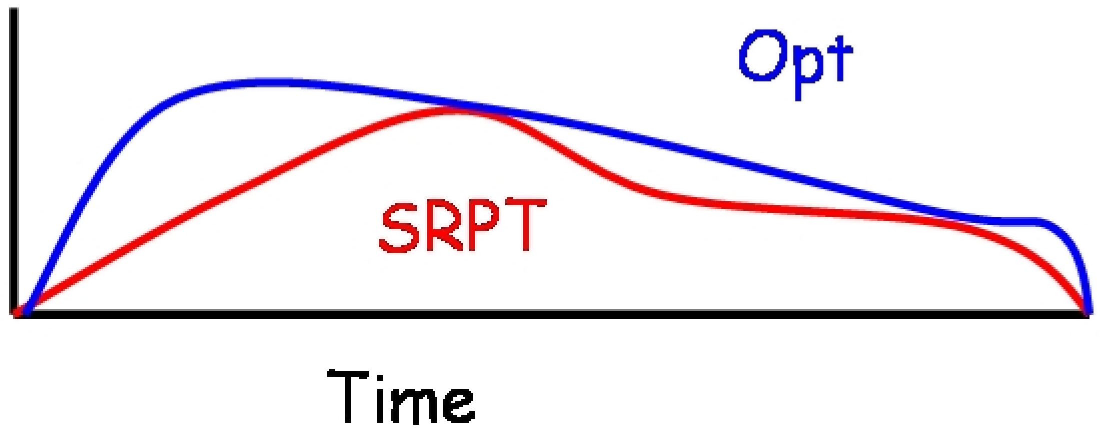
Questions

- What is the optimal algorithm for total flow time?
  - Shortest Remaining Processing Time (SRPT) = run the job that has the least work remaining unfinished
- What is the optimal algorithm for total fractional flow time?
  - Shortest Job First (SJF) = run he job that initially had the least work

Example Local Competitiveness Argument(1)

- Theorem [Folklore]: Shortest Remaining Processing Time (SRPT) is optimal for total flow time for fixed speed processor
- Proof:

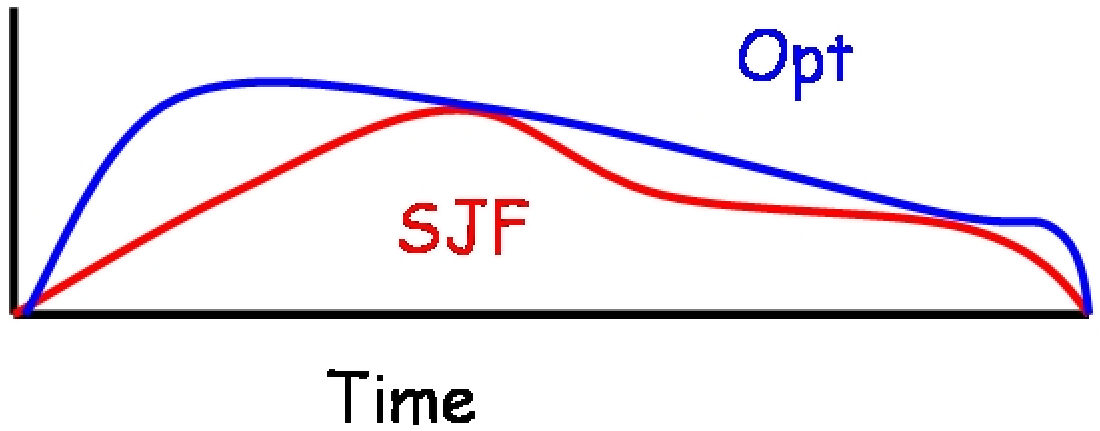
dG(t)/dt = Number of unfinished jobs



Example Local Competitiveness Argument(1)

- Theorem [Folklore]: Shortest Job First (SJF) is optimal for total fractional flow time for fixed speed processor
- Proof:

dG(t)/dt = Number of unfinished jobs



Nonclairvoyant Schedulers

- One can not in general implement SRPT in an operating system setting since one doesn't know processing time of a job
- A nonclairvoyant job selection policy doesn't know the work (processing time) of a job when it arrives

- Example nonclairvoyant job selection policy
  - Shortest Elapsed Time First (SETF)
  - Run the job that has been run the least so far
  - Favors newly arriving jobs until that have been run as much as old jobs

Resource Augmentation Analysis [KP 95]

- Compare the limited (e.g. online) algorithm with more resources (e.g. a faster processor or more processors) to the optimal algorithm with less resources
- Online algorithm A is s-speed c-competitive if

$$\max_I A_s(I)/Opt_1(I) < c$$

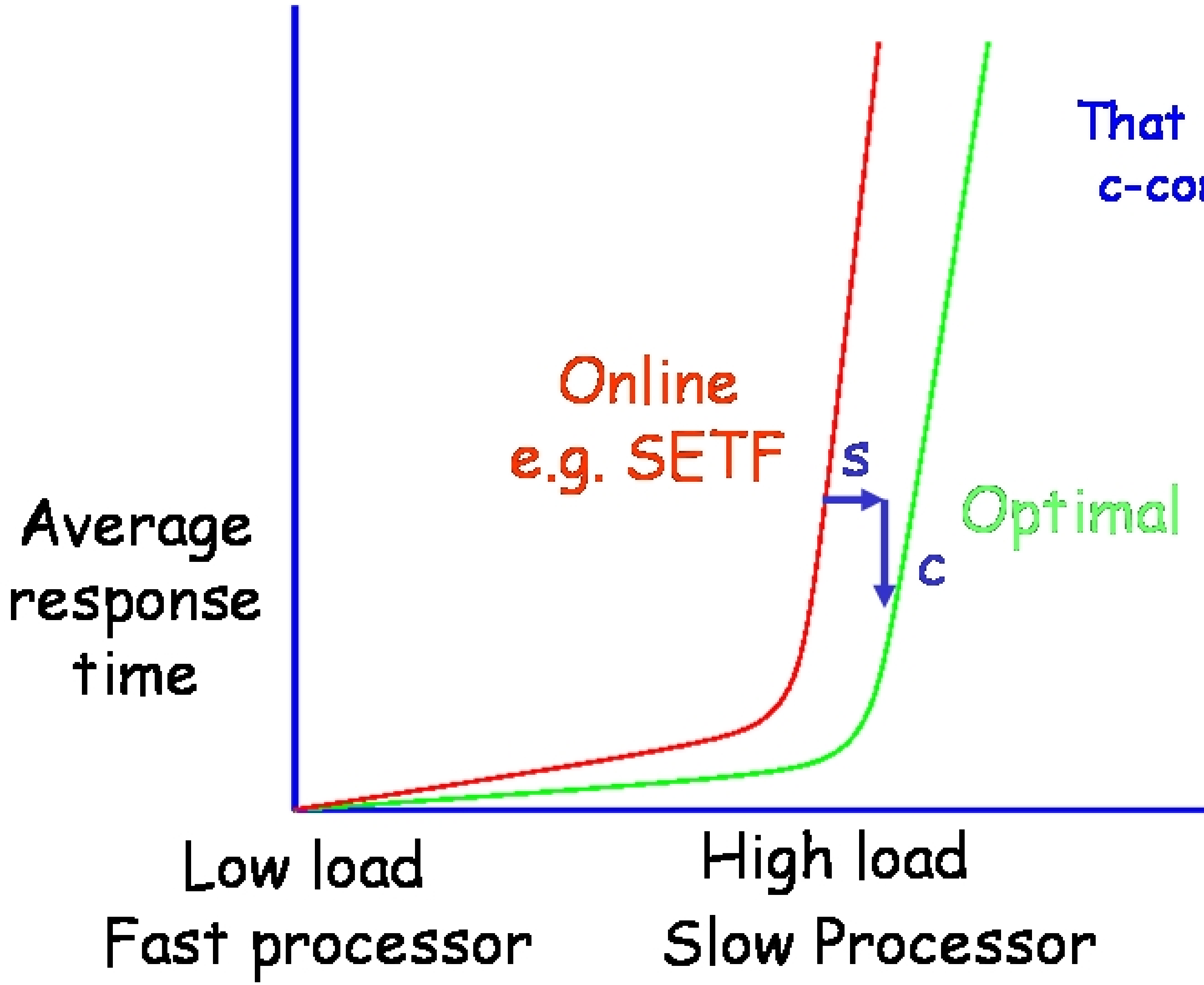
  - Subscript denotes processor speed

- Example: A 2-speed 3-competitive algorithm equipped with a speed 2 processor guarantees an average response time at most 3 times the optimal average response time for a 1 speed processor

PPT to PDF 1.4

Classic Server QoS Curves

That SETF is s-speed c-competitive means



PPT to PDF 1.4

Old Chinese Saying:

- Two blind shoemakers are better than one politician

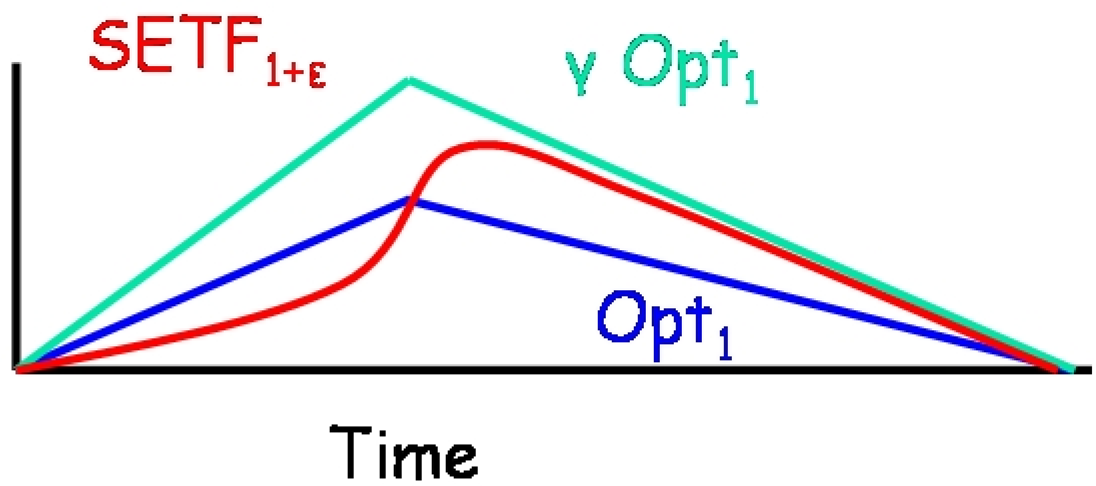
三个臭皮匠

抵上诸葛亮

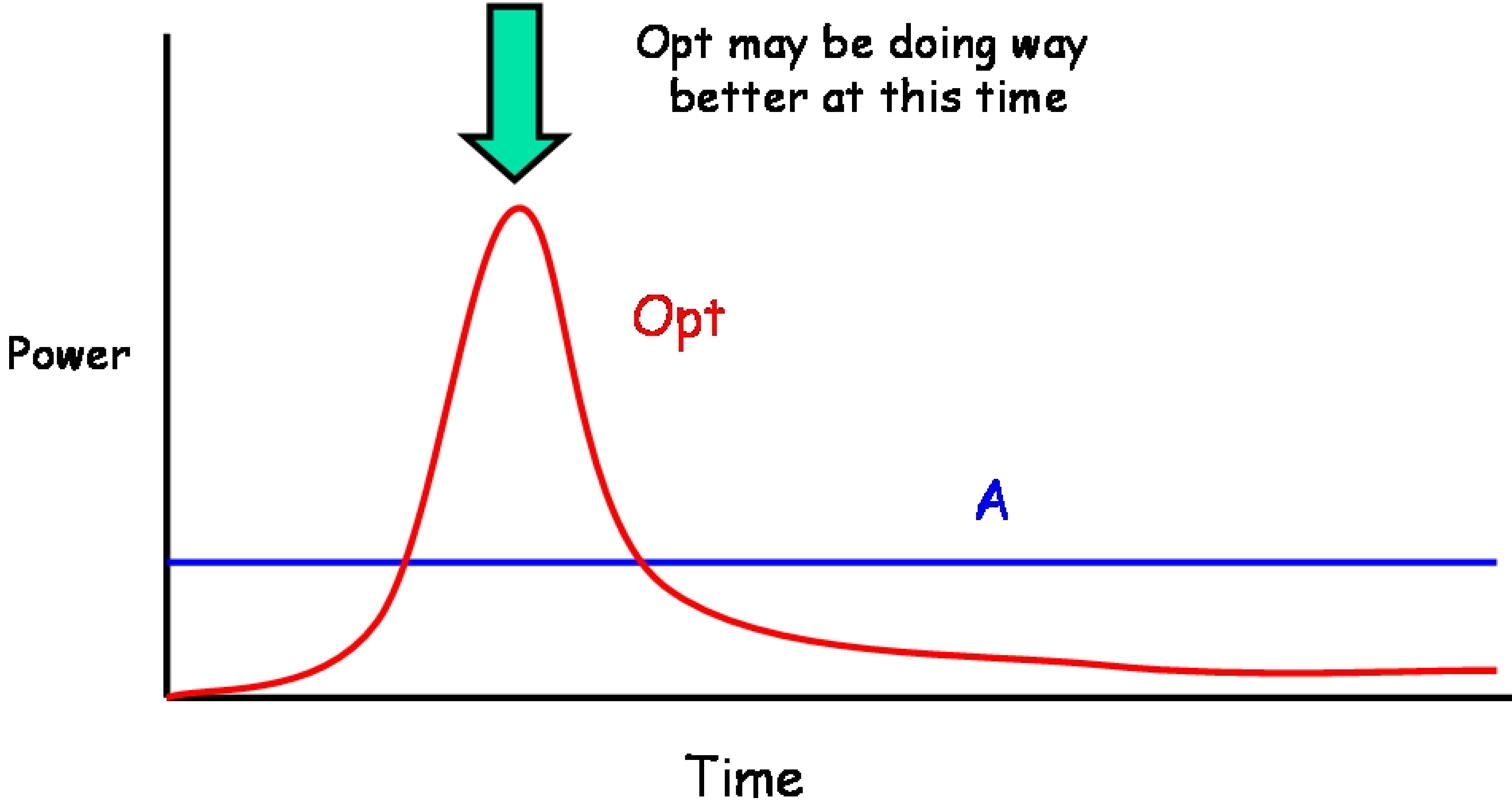
Example Local Competitiveness Argument(2)

- Theorem [KP95]: Shortest Elapsed Time First (SETF) is $(1+\varepsilon)$-speed $O(1 + 1/\varepsilon)$-competitive for total flow time
- Proof: Let γ = competitive ratio

dG(t)/dt = Number of unfinished jobs



Why Local Competitiveness won't work with Speed Scaling



Algorithm A is Amortized Locally γ-Competitive for Objective G with Potential Function Φ

- Boundary Condition

$$\Phi(0) = 0 \text{ and } \Phi(+\infty) \geq 0$$

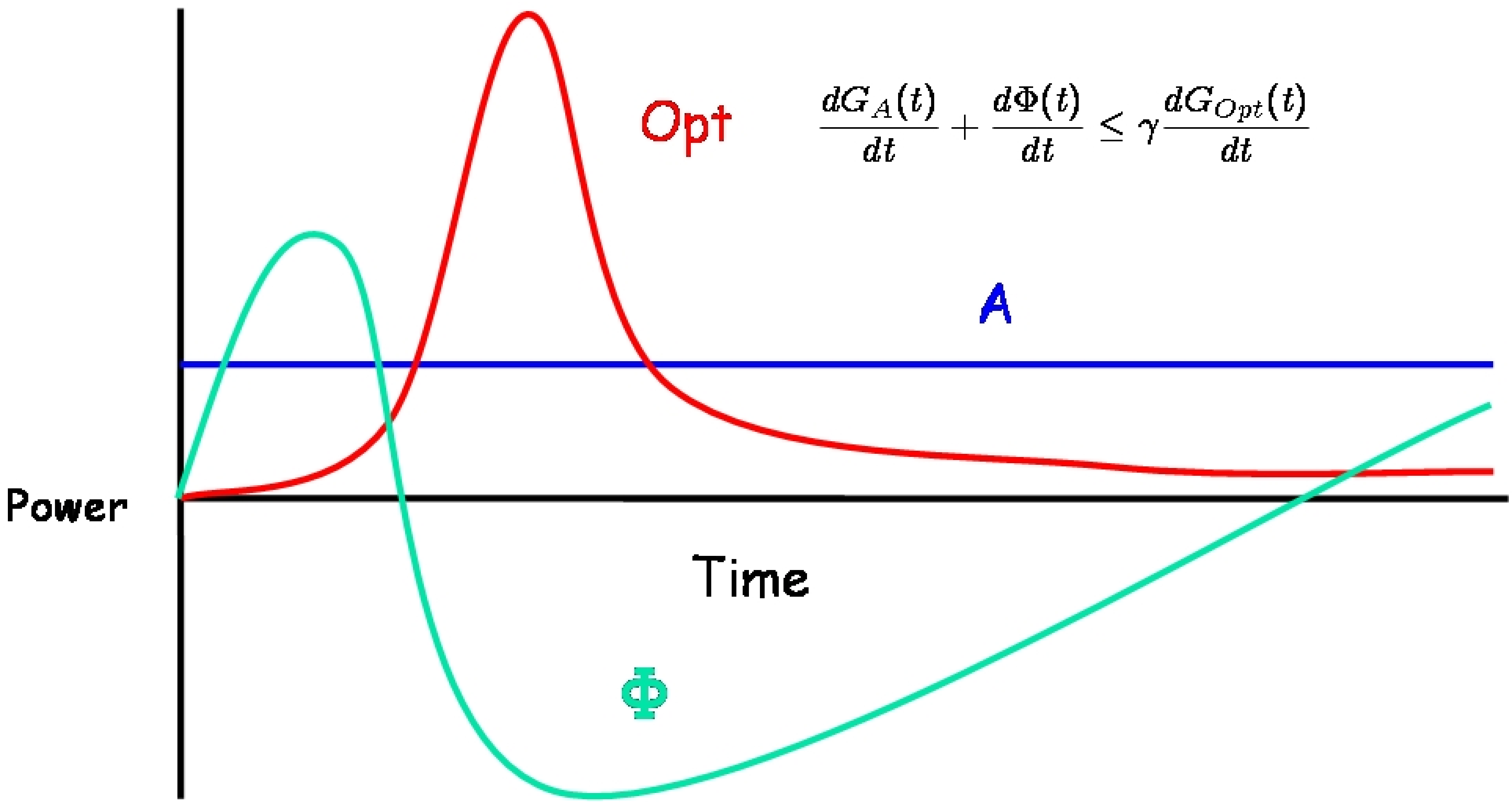
Intuition:
Φ = an energy bank for the online algorithm

- Running Condition

$$\frac{dG_A(t)}{dt} + \boxed{\frac{d\Phi(t)}{dt}} \leq \gamma \frac{dG_{Opt}(t)}{dt}$$

Removing potential change returns us to local competitiveness condition

Local Competitiveness and Speed Scaling



Outline: Online speed scaling algorithms

- Online speed scaling algorithms
  - Review
    - competitiveness
    - local competitiveness
    - resource augmentation
    - Amortized local competitiveness
  - Flow time and energy
  - Deadline feasibility and energy
  - Deadline feasibility and temperature

PPT to PDF 1.4

First Observation About Speed Scaling for Flow Problems

- Bounded Energy Problem
  - Minimize total flow time
  - Subject to the constraint that the energy consumed is bounded by E, the energy in the battery
- Theorem: There is no O(1)-competitive online algorithm for the bounded energy problem
  - Proof Idea: How much energy do you give the first job that arrives?
  - If it is not an Ω(E) then you are not O(1)-competitive

Energy/Flow Trade-Off Problem Definition [AF06]

- Job i has release date r_i and work y_i
- Optimize total flow + ρ * energy used
- Natural interpretation: User specifies an energy amount ρ that he is willing to spend to get a unit improvement in response
    - e.g. If the user is willing to spend 1 ergs of energy for a 3 microsecond improvement in response, then ρ=3.
- wlog, ρ=1.

Offline Bounded Energy Problem

- Recall that the KKT optimality conditions imply that in a normal schedule, power of job i p_i is proportional to the number of jobs delayed by job i
  - Normal = no job completes at exactly the time that another job is released
- [AF06] Propose online algorithm naturally suggested by this corollary
  - Online lower bound to delayed jobs:
    - Number of alive jobs ≤ number of jobs that the selected jobs delays
  - Online speed scaling algorithm:
    - p_i = number of alive jobs

Energy/Flow Trade-Off Results

- [AF06] Show natural online algorithm is about 400-competitive for unit-work jobs when the cube-root rule holds (α =3)
  - Reasoned about optimal schedule
- [BPS07] show this algorithm is 4-competitive for all α for unit-work jobs
- [BPS07] show a natural generalization of this algorithm for arbitrary weight and arbitrary work jobs is about 20-competitive when the cube-root rule holds

Running Condition for Flow Plus Energy Objective

- If objective G is flow plus energy then

$$\boxed{\frac{dG(t)}{dt} = n(t) + p(t) = n(t) + s(t)^{\alpha}}$$

  - s(t) = speed at time t
  - p(t) = power at time t
  - n(t) = number of jobs alive at time t
- And thus the running condition

$$\frac{dG_A(t)}{dt} + \frac{d\Phi(t)}{dt} \leq \gamma \frac{dG_{Opt}(t)}{dt}$$

- becomes

$$\boxed{n_A(t) + s_A(t)^{\alpha} - \gamma\left(n_{Opt}(t) + s_{Opt}(t)^{\alpha}\right) + \frac{d\Phi(t)}{dt} \leq 0}$$

Running Condition for Flow Plus Energy Objective

$$n_A(t) + s_A(t)^\alpha - \gamma(n_{Opt}(t) + s_{Opt}(t)^\alpha) + \frac{d\Phi(t)}{dt} \leq 0$$

- Als suggests the speed scaling algorithm

$$\boxed{s_A(t)^\alpha = n_A(t)}$$

- With this speed scaling algorithm, the running condition reduces to

$$\boxed{2n_A(t) - \gamma(n_{Opt}(t) + s_{Opt}(t)^\alpha) + \frac{d\Phi(t)}{dt} \leq 0}$$

[BPS07] Unit Work Flow

- Theorem: For unit-weight unit-work jobs, the natural speed scaling algorithm is 2-competitive with respect to fractional flow plus energy
  - Proof: Amortized local competitiveness argument with potential function

[BPS07] Unit Work Flow

- A drop of intuition:
  - Standard potential function
    - e.g Edmonds 1999 analysis of Round Robin
  - Φ = future online cost – γ (future adversary costs)
    - assuming no more jobs arrive
  - Standard potential function generally works if the worst case future for the online algorithm is if no more jobs arrive

[BPS07] Unit Work Flow

- Standard potential function here would be something like

$$\Phi(t) = n_A(t)^{(2\alpha+1)/\alpha} - n_{Opt}(t)^{(2\alpha+1)/\alpha}$$

- Doesn't work when $n_A \gg n_{Opt}$ and one new job arrives.
- In speed scaling problems, the empty future is apparently never the worst case future for the online algorithm.
- So we end up using:

$$\Phi(t) = \frac{2\alpha^2}{(2\alpha+1)} \left(\max(0, n_A(t) - n_{Opt}(t))\right)^{(2\alpha+1)/\alpha}$$

- Note that this new potential function decreases faster when $n_A \gg n_{Opt}$ and a new job arrives

[BPS07] Unit Work Flow

- Recall

$$\Phi(t) = \frac{2\alpha^2}{(2\alpha+1)} (\max(0, n_A(t) - n_{Opt}(t)))^{(2\alpha+1)/\alpha}$$

- Trivially Φ is initially zero, and never negative
- When a job completes, Φ remains unchanged since we are considering fractional weight
- When a job arrives, Φ remains unchanged since both n_A and n_{Opt} increase by 1.
- So we are left to consider times when no jobs arrive or are completed

[BPS07] Unit Work Flow

- Recall

$$\Phi(t) = \frac{2\alpha^2}{(2\alpha+1)} \left(\max(0, n_A(t) - n_{Opt}(t))\right)^{(2\alpha+1)/\alpha}$$

- If $n_A < n_{Opt}$ then $\Phi = 0$ and $d\Phi/dt = 0$, and by setting $\gamma=2$ we have that

$$2n_A(t) - \gamma(n_{Opt}(t) + s_{Opt}(t)^{\alpha}) + \frac{d\Phi(t)}{dt} \leq 0$$

[BPS07] Unit Work Flow

- Recall $$\Phi(t) = \frac{2\alpha^2}{(2\alpha+1)} (\max(0, n_A(t) - n_{Opt}(t)))^{(2\alpha+1)/\alpha}$$

- If $n_A > n_{Opt}$ then
$$\frac{d\Phi(t)}{dt} = 2\alpha(n_a - n_o)^{\beta} \frac{d(n_a - n_o)}{dt} = -2\alpha(n_a - n_o)^{\beta}(s_a - s_o)$$
$$= -2\alpha(n_a - n_o)^{\beta}(n_a^{1/\alpha} - s_o)$$

- By Young inequality: $\mu \frac{a^p}{p} + \left(\frac{1}{\mu}\right)^{q/p} \frac{b^q}{q} \geq ab$

- We get $$\frac{d\Phi(t)}{dt} \leq 2(n_a - n_o) + 2s_o$$ and by setting γ=2, we get

$$2n_A(t) - \gamma(n_{Opt}(t) + s_{Opt}(t)^{\alpha}) + \frac{d\Phi(t)}{dt} \leq 0$$

[BPS07] Unit Work Flow

- Corollary: For unit-weight unit-work jobs, the natural speed scaling algorithm is 4-competitive with respect to flow plus energy

Weighted Flow Plus Energy Objective

- Problem definition: each job has a weight and the QoS objective is the weighted sum of flow times
- Job selection algorithm= Highest Density First (HDF)
  - Density = weight/work
- The natural speed scaling algorithm is now

$$s_A(t)^\alpha = w_A(t)$$

  - where $w_A(t)$ is fractional weight of unfinished jobs
- and the running condition is then

$$2w_A(t) - \gamma(w_{Opt}(t) + s_{Opt}(t)^\alpha) + \frac{d\Phi(t)}{dt} \leq 0$$

- Everything is essentially the same, except that "weights" replace "number of jobs" and that the potential function will have to be different

[BPS07] Weighted Flow

- Theorem: For arbitrary work and weight jobs, the natural speed scaling algorithm is (2α-2)-competitive with respect to fractional weighted flow plus energy
  - Proof: Amortized local competitiveness argument with more complicated potential function.

[BPS07] Weighted Flow

- Intuition: The following potential function works for unit work jobs

$$\Phi(t) = w_A^{(\alpha-1)/\alpha}(w_A - \frac{2\alpha}{\alpha-1} w_{Opt})$$

- Intuitively this potential is the future online cost minus the future adversary cost assuming that the adversary has to work at least the speed that online is now working

Proof Continued

- More intuition: Notice that this same potential function also works for unit density jobs

$$\Phi(t) = w_A^{(\alpha-1)/\alpha}\left(w_A - \frac{2\alpha}{\alpha-1} w_{Opt}\right)$$

- Now if all jobs have inverse density h, some calculation shows that the potential function should be multiplied by h

$$\Phi(t) = h \cdot w_A^{(\alpha-1)/\alpha}\left(w_A - \frac{2\alpha}{\alpha-1} w_{Opt}\right)$$

Proof Continued

- More intuition: Now if you have jobs with different densities, the weight of the lower density jobs should add to the weight in the potential for higher density jobs. The potential for inverse density h jobs is then

$$\Phi(t) = h \cdot w_A(h)^{(\alpha-1)/\alpha}\left(w_A(h) - \frac{2\alpha}{\alpha-1} w_{Opt}(h)\right)$$

  - w(h) is fractional weight of alive jobs with inverse density at least h
- Summing up the potential for all the possible densities gives us our potential

$$\Phi(t) = \sum_h h \cdot w_A(h)^{(\alpha-1)/\alpha}\left(w_A(h) - \frac{2\alpha}{\alpha-1} w_{Opt}(h)\right)$$

Proof Continued

- Or equivalently

$$\Phi(t) = \eta \int_{h=0}^{\infty} \left(w_A(h)^{(\alpha-1)/\alpha} (w_A(h) - \frac{2\alpha}{\alpha - 1} w_{Opt}(h)) \right) dh$$

- To finish we need to verify the running condition

$$2w_A(t) - \gamma(w_{Opt}(t) + s_{Opt}(t)^{\alpha}) + \frac{d\Phi(t)}{dt} \leq 0$$

- Corollary: For arbitrary work and weight jobs, there is a speed scaling algorithm that is 20-competitive with respect to weighted flow plus energy when the cube-root rule holds
  - Proof: Uses resource augmentation analysis of HDF from [BLMP01]

Outline: Online speed scaling algorithms

- Online speed scaling algorithms
  - Review
    - competitiveness
    - local competitiveness
    - resource augmentation
    - Amortized local competitiveness
  - Flow time and energy
  - Deadline feasibility and energy
  - Deadline feasibility and temperature

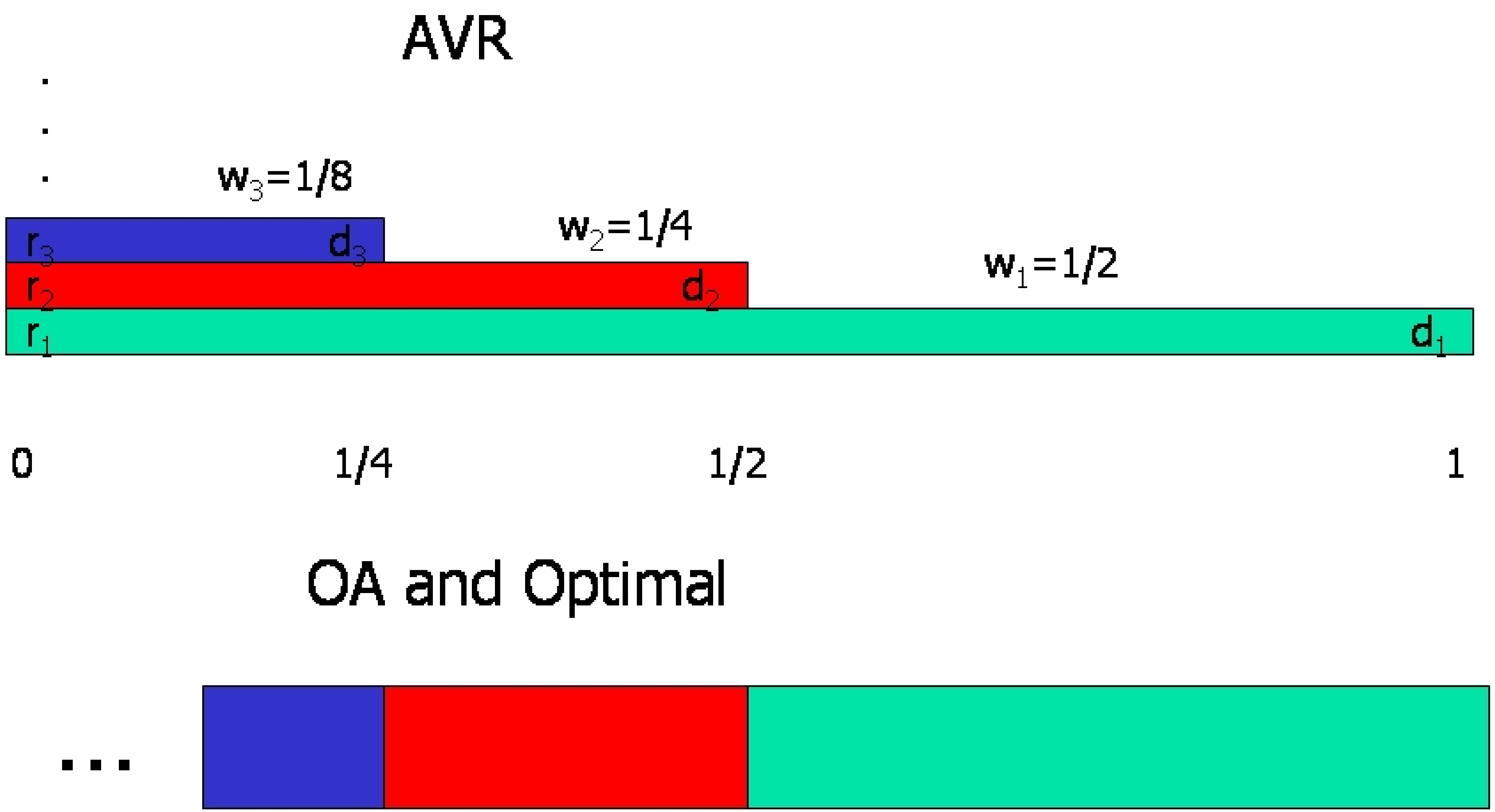
Natural Online Algorithms Given in YDS95

- **Average Rate (AVR):** Run each job I at rate

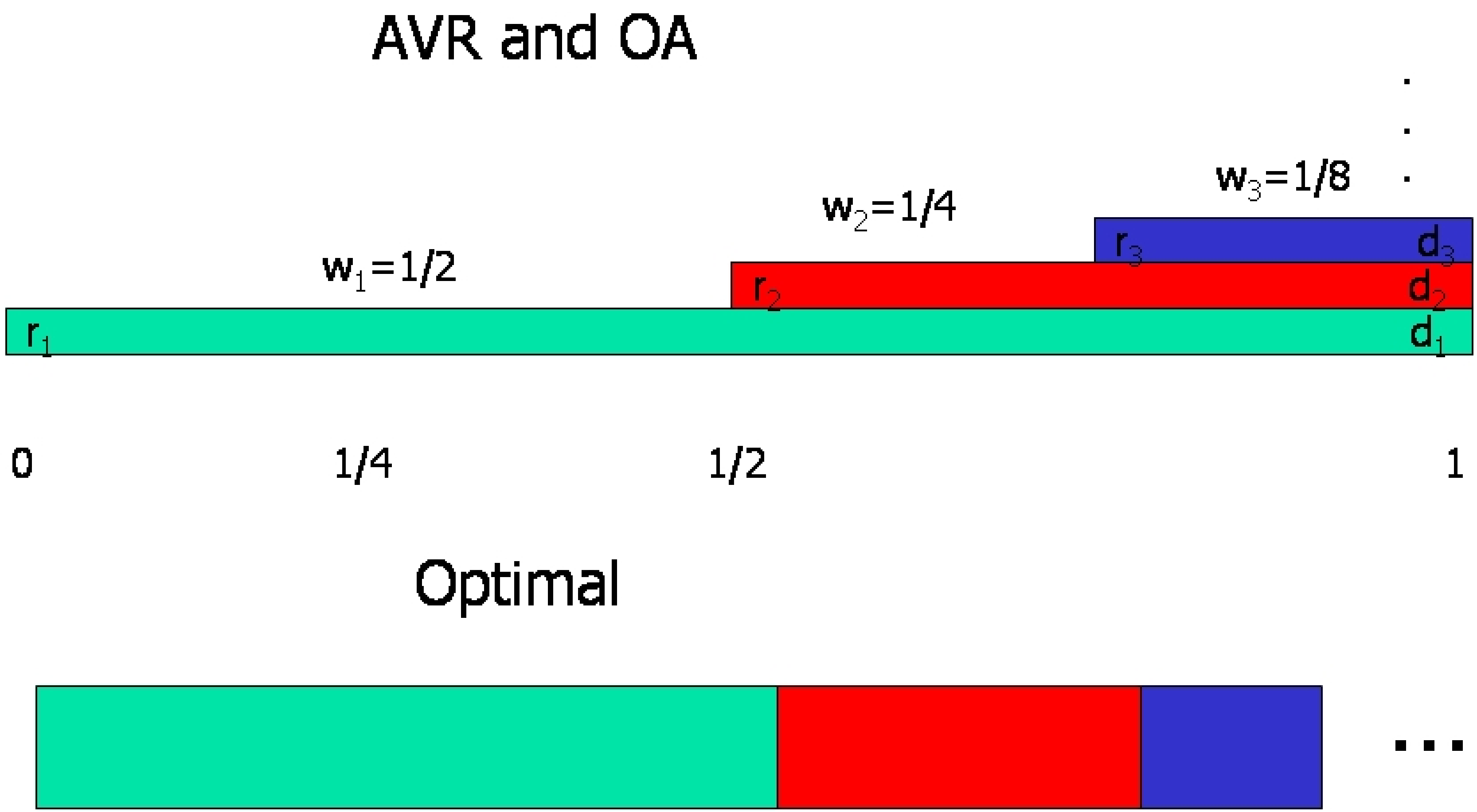
$$\frac{w_i}{(d_i - r_i)}$$

- **Optimal Available (OA):** After each arrival, recompute the YDS schedule assuming no more arrivals.
  - Essentially all jobs are treated as having equal release times

First Example Instance



Second Example Instance



Results for Online Scheduling to Mange Energy

- YDS95
  - p^p lower bound on competitive ratio for AVR
    - Easy to see this lower bound also holds for OA
  - $2^{p-1}p^p$ upper bound on competitive ratio for AVR
    - Complicated spectral analysis
- BKP04
  - Tight p^p bound on competitive ratio of OA
  - New online algorithm BKP with competitive ratio at most 8 e^p, for p at least 2.
  - BKP is e-competitive with respect to the objective function $\max_{\text{times } t}$ speed at time t
    - No deterministic algorithm can have a better competitive ratio.

Upper Bound on Competitive Ratio for OA (1)

- Introduce potential function Φ
  - If all deadlines are equal then

$$\Phi = s_{OA}^{p-1} (p\, w_{OA} - p^2\, w_{adv})$$

  - w_{OA} be work left for OA
  - w_{adv} be work left for the adversary
  - s_{OA} be speed OA is working
  - s_{adv} be speed that the adversary is working

Upper Bound on Competitive Ratio for OA (2)

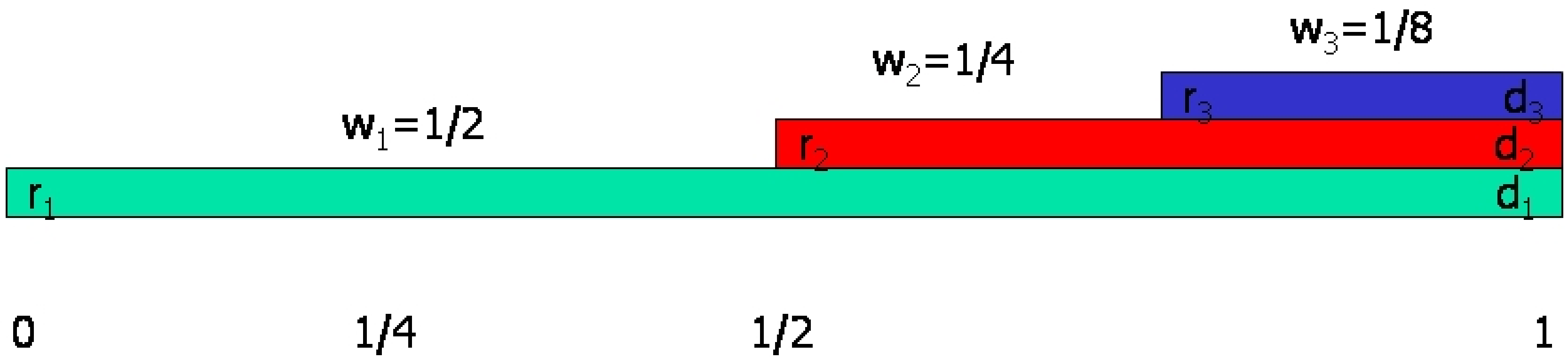
- Recall $\Phi = s_{OA}^{p-1} (p\, w_{OA} - p^2\, w_{adv})$
- We then need to show
  - $$\frac{dG_A(t)}{dt} + \frac{d\Phi(t)}{dt} \leq \gamma \frac{dG_{Opt}(t)}{dt}$$
  - Here G = energy
  - dG/dt = power
  - Need to consider 2 cases:
    - When OA runs a job and
    - when a new job arrives

Outline: Online speed scaling algorithms

- Online speed scaling algorithms
  - Review
    - competitiveness
    - local competitiveness
    - resource augmentation
    - Amortized local competitiveness
  - Flow time and energy
  - Deadline feasibility and energy
  - Deadline feasibility and temperature

Online Speed Scaling to Minimize Temperature

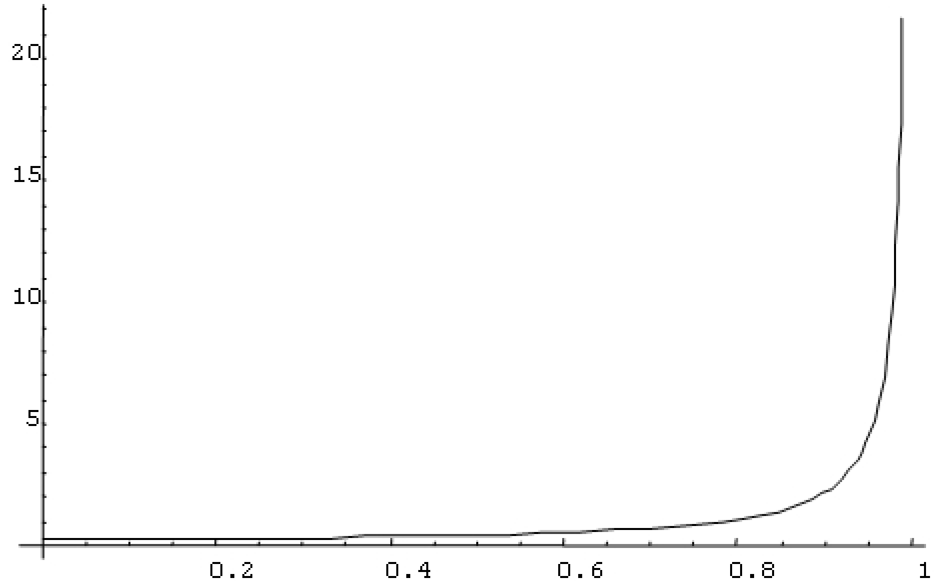
- It is clear that neither the online algorithms proposed by YDS, that is, OA and AVR, are not O(1)-competitive with respect to temperature



e Lower Bound on Competitiveness for Maximum Speed(1)

- The adversary releases work at the speed $s_{adv}(t) = -1 / ((1-t)\ln \varepsilon)$ that the adversary works
- The deadline for all work = 1

s_{adv}



t

e Lower Bound on Competitiveness for Maximum Speed

- If the adversary stops releasing work at some time t > (e-1)/e, then by some calculation the first YDS interval will be [et – (e-1), 1] and thus the optimal maximum speed is

$$\frac{\int_{et-(e-1)}^{t} s_{adv}(t)dt}{1-(et-(e-1))}$$

- A c-competitive online algorithm can work no faster than c times this amount
- Then c needs to be sufficiently large so that online finishes all work by time 1. By calculation, c has to be at least e.

BKP Algorithm

- Algorithm Description:

  Speed k(t) at time t =

  e * maximum over all $t_2 > t$ of
  $\Sigma\, w_i / (t_2 - t_1)$

  - Sum is over jobs i with $t_1 = et - (e-1)t_2 < r_i < t$ and $d_i < t_2$

$t_1 = et - (e-1)t_2$ r_i d_i t d_i t_2

current time

BKP Analysis

- Theorem (BKP04) BKP completes all jobs by their deadlines
- Theorem (BKP04): BKP is cooling oblivious, that is, O(1)-competitive with respect to temperature for all
  - Proof: If YDS does y(t) work at time t, then we modify the instance so that y(t) work arrives at time t with deadline t+1
  - This transformation doesn't effect YDS and won't decrease speed/temperature for BKP
  - Show that $\int_t^{t+1/b} k(t)\,dt$ (an upper bound for the energy used by BKP during a interval of length 1/b) is O(1) times the energy that YDS uses during that interval
    - Hilbert's Theorem, Hardy and Littlewood inequalities
- Corollary: BKP is O(1)-competitive with respect to total energy and maximum power
  - Proof: BKP is cooling oblivious

Summary of Results for Deadline Scheduling

Recall dT/dt = a P - b T	Equals $Max_t \int_t^{t+x} P\,dt$	Offline	Online
Energy b=0	$x=\infty$	Optimal YDS algorithm YDS 1995 Cute correctness proof BP2005	O(1)-competitive algorithms OA AVR : YDS 1995 BKP : BKP 2004
Temperature $0 < b < \infty$	$x= \Theta(1/b)$	Ellipsoid Exact BKP 2004 YDS is O(1)-approximation BP2005	BKP is O(1)-competitive BP2005
Maximum Power $b=\infty$	x=infinitesimal	Optimal YDS algorithm YDS 1995	BKP is strongly e^P-competitive BKP 2004

Exercise

- Assume that we want to minimize the total flow time plus 4 times the energy, for unit work jobs. Assume that the power is the cube of the speed. By applying the KKT optimality conditions, explain how to recognize an optimal schedule. Recall the KKT optimality conditions are

$$\min f_0(x)$$
$$f_i(x) \le 0 \qquad i = 1, \ldots, n$$

$$\begin{aligned} f_i(x) &\le 0 \qquad i = 1, \ldots, n \\ \lambda_i &\ge 0 \qquad i = 1, \ldots, n \\ \lambda_i f_i(x) &= 0 \\ \nabla f_0(x) + \sum_{i=1}^{n} \lambda_i \nabla f_i(x) &= 0 \end{aligned}$$

- We consider the online algorithm A for minimizing fractional flow time plus energy for unit work jobs. Assume that power = the square of speed. Recall that speed s_A for A is then $(n_A)^{1/2}$, where n_A is the fractional number of unfinished jobs (fractional means that a job that is 1/3 finished only adds 2/3 to n_A). The fractional flow for A is the integral over time of n_A. We wish to show that A is O(1)-competitive using a different potential function, namely $\Phi = \sigma(n_A^{3/2} - 4n_A^{1/2}n_{Opt})$, where σ is some constant. Here n_{Opt} is the fractional number of jobs remaining in the optimal solution.
  - First show that the equation $2n_A + \gamma(n_{Opt} + s_{Opt}^2) + d\Phi/dt \le 0$ holds at times when no jobs arrive, for some constant γ
    - Hints: First evaluate $d\Phi/dt$. Recall $dn_A/dt = - s_A$. Using Young's inequality we know that $(n_A)^{1/2} s_{Opt} \le n_A/2 + s_{Opt}^2/2$. This is not too hard.
  - Then show the potential function Φ does not increase when a new job arrives, that is when n_A and n_{Opt} both increase by 1.