ADFOCS

Kirk Pruhs

UNIVERSITY OF PITTSBURGH VERITAS VIRTUS 1787

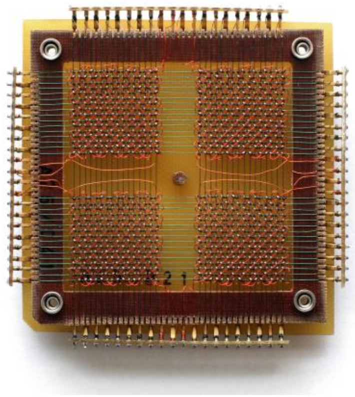
Green Computing Algorithmics

Talk 1: Introduction and Speed Scaling

ADFOCS 2015

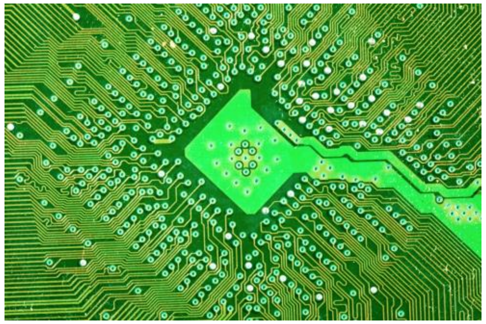
Dominant Computational Resource

Memory



Time

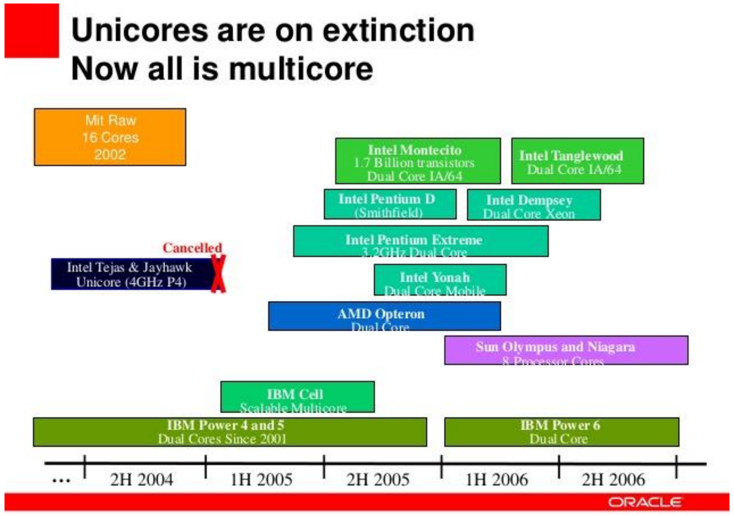
Energy



1940 1950 1960 1970 1980 1990 2000 2010 2020 2030

Example: Computer Processors

- 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.



Example: Data Centers

- "What matters most to the computer designers at Google is not speed, but power, low power, because data centers can consume as much electricity as a city."--- Eric Schmidt, Google CEO in 2002

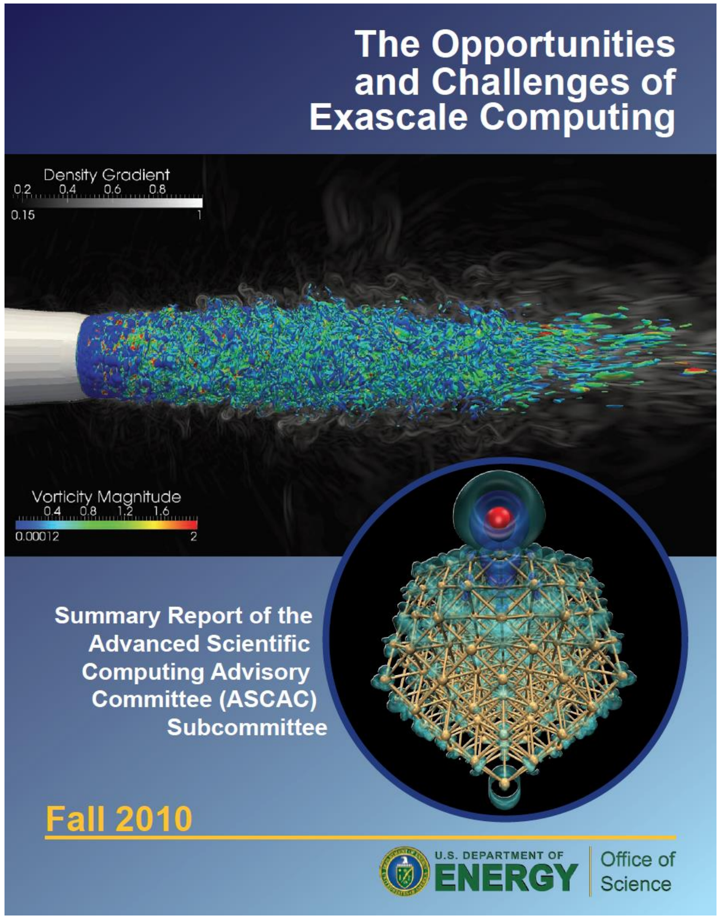
Example: High Performance Computing

- Finding: Making the transition to exascale poses numerous unavoidable scientific and technological challenges. Three challenges to be resolved are:

1. Reducing power requirements. Based on current technology, scaling today's systems to an exaflop level would consume more than a gigawatt of power, roughly the output of Hoover Dam. Reducing the power requirement by a factor of at least 100 is a challenge for future hardware and software technologies.

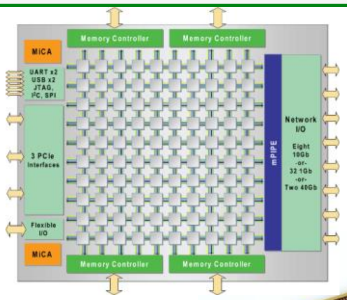
2. Coping with runtime errors

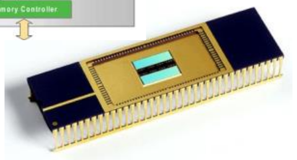
3. Dealing with massive parallelism



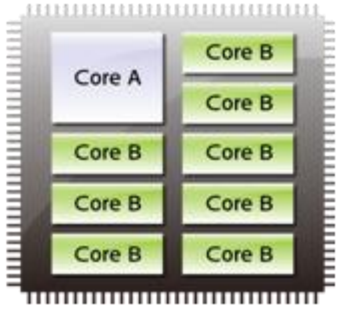
The Opportunities and Challenges of Exascale Computing
Density Gradient
Vorticity Magnitude
Summary Report of the Advanced Scientific Computing Advisory Committee (ASCAC) Subcommittee
Fall 2010
U.S. DEPARTMENT OF ENERGY
Office of Science

We are in a Green Computing Revolution





- During the last decade, information technology is being redesigned with energy efficiency as a first order resource
  - Technological reasons
  - General societal trend toward most sustainable technologies

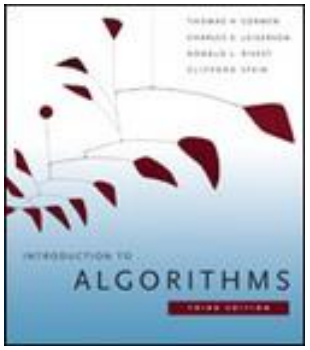


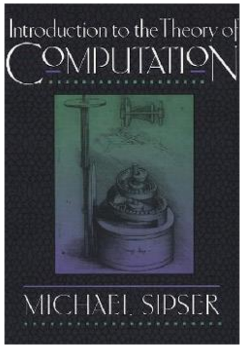
Most Common Vision of Green Computing Algorithmics

It would be great to have a big Oh theory for energy

Participant at generic
National Science Foundation
visioning workshop

Translation

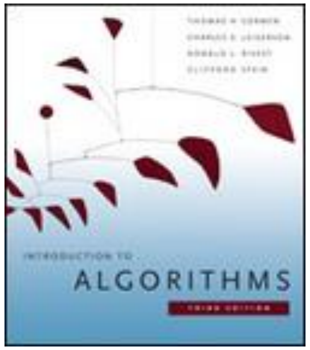


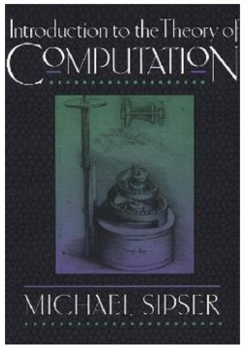


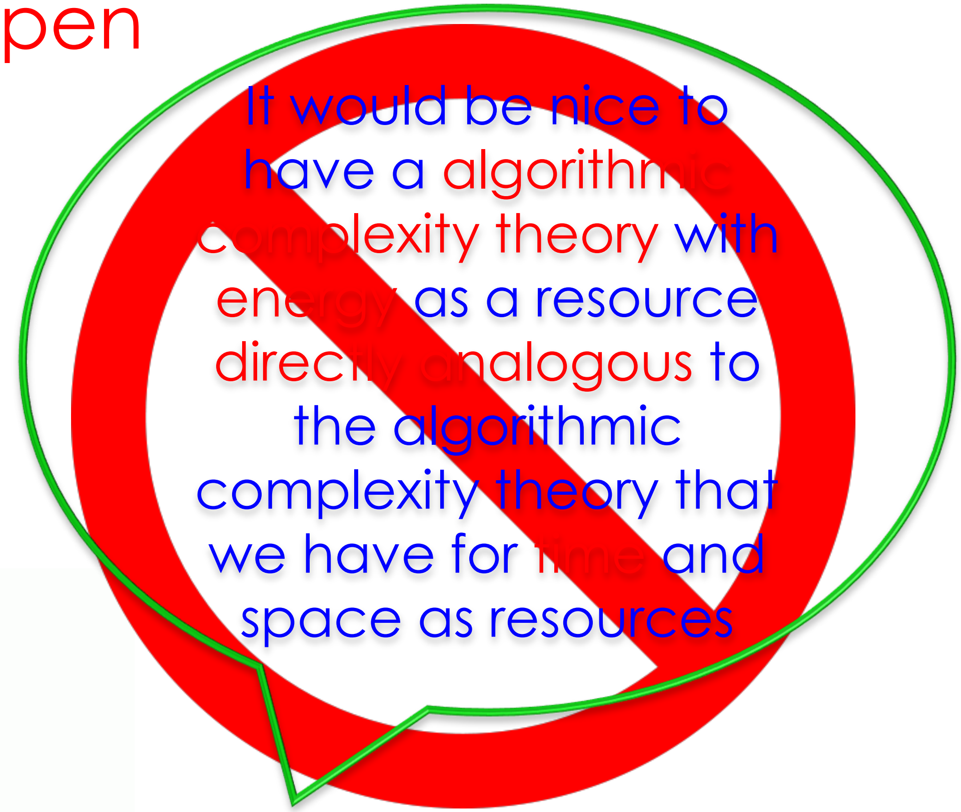
It would be nice to have a algorithmic complexity theory with energy as a resource directly analogous to the algorithmic complexity theory that we have for time and space as resources

Participant at generic National Science Foundation visioning workshop

Not going to Happen



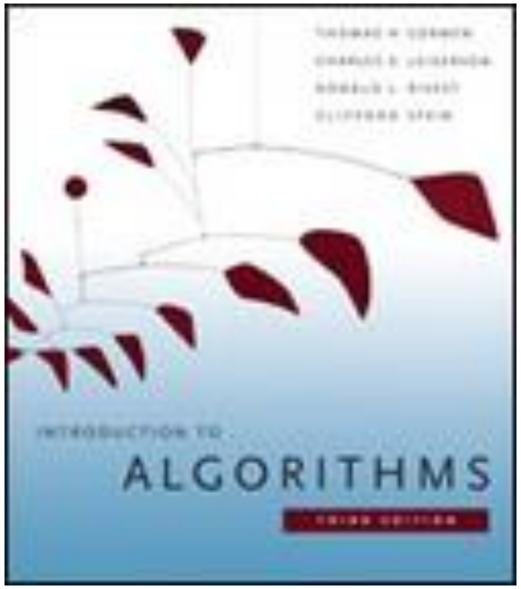


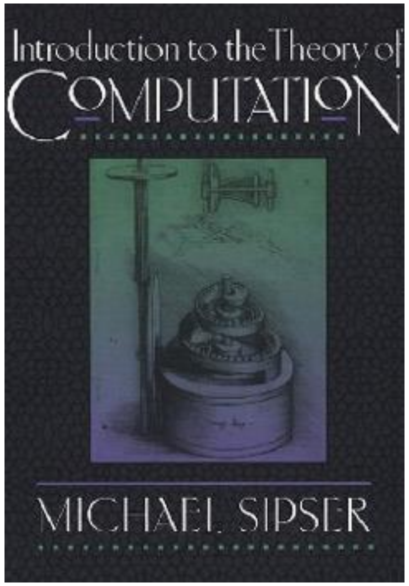


It would be nice to have a algorithmic complexity theory with energy as a resource directly analogous to the algorithmic complexity theory that we have for time and space as resources

Participant at generic National Science Foundation visioning workshop

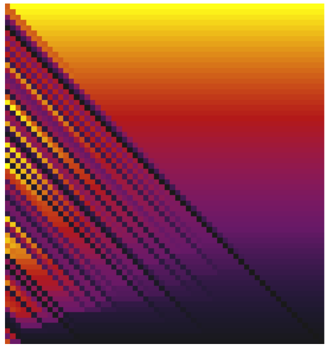
- What is something you learned about time (or space) as a computational resource from standard texts?

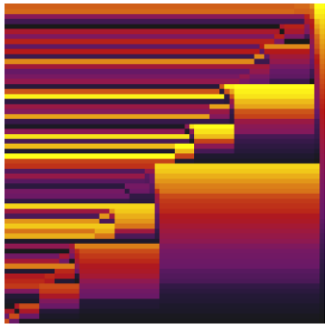




One Problem: How do you assign energies to algorithms?

- BubbleSort
  - Time = n^2
  - Space = n
  - Energy = ?
- MergeSort
  - Time = n log n
  - Space = n
  - Energy = ?

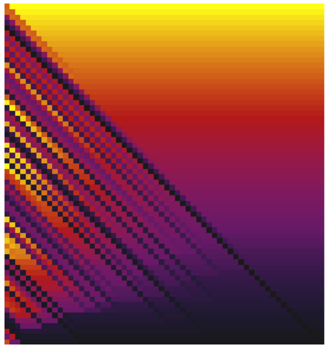


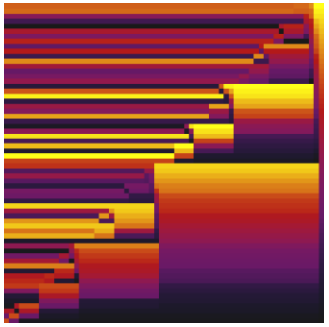


One Problem: How do you assign energies to algorithms?

- BubbleSort
  - Time = n^2
  - Space = n
  - Energy = ?

- MergeSort
  - Time = n log n
  - Space = n
  - Energy = ?

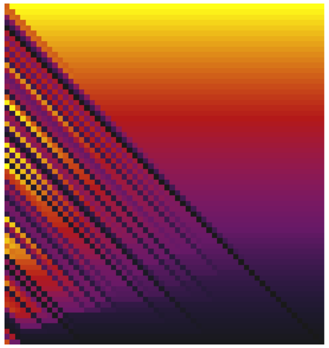


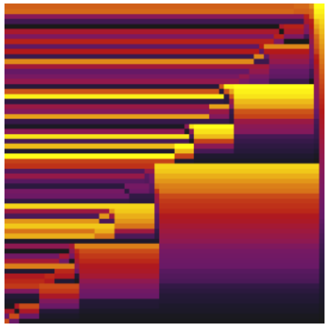


Option 1: Assume all operations take constant energy.
Problem?

One Problem: How do you assign energies to algorithms?

- BubbleSort
  - Time = n^2
  - Space = n
  - Energy = ?
- MergeSort
  - Time = n log n
  - Space = n
  - Energy = ?



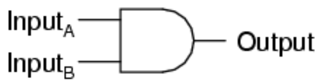


Option 2: Assume some operations require more energy than others. Which operations require energy?

What Operations Require Energy?

- Second Law of Thermodynamics: Entropy can't can't decrease
- Landauer's principle: Irreversible operations, like AND, require energy because information is lost
- To erase a bit costs k T ln 2 units of energy
  - k = Boltzmann's constant
  - T = temperature in Kelvin
  - Erasing 1 bit at room temperature costs at least a millijoule

2-input AND gate



A	B	Output
0	0	0
0	1	0
1	0	0
1	1	1

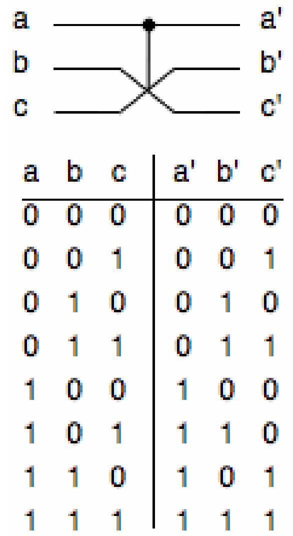
What Operations Require Energy

- Second Law of Thermodynamics: Entropy can't can't decrease
- Landauer's principle: Irreversible operations, like AND, require energy

- However, all computation can be made reversible
- Thus thus no minimum energy for any computation

- How would you make computation reversible on a Turing Machine or RAM model of computation?

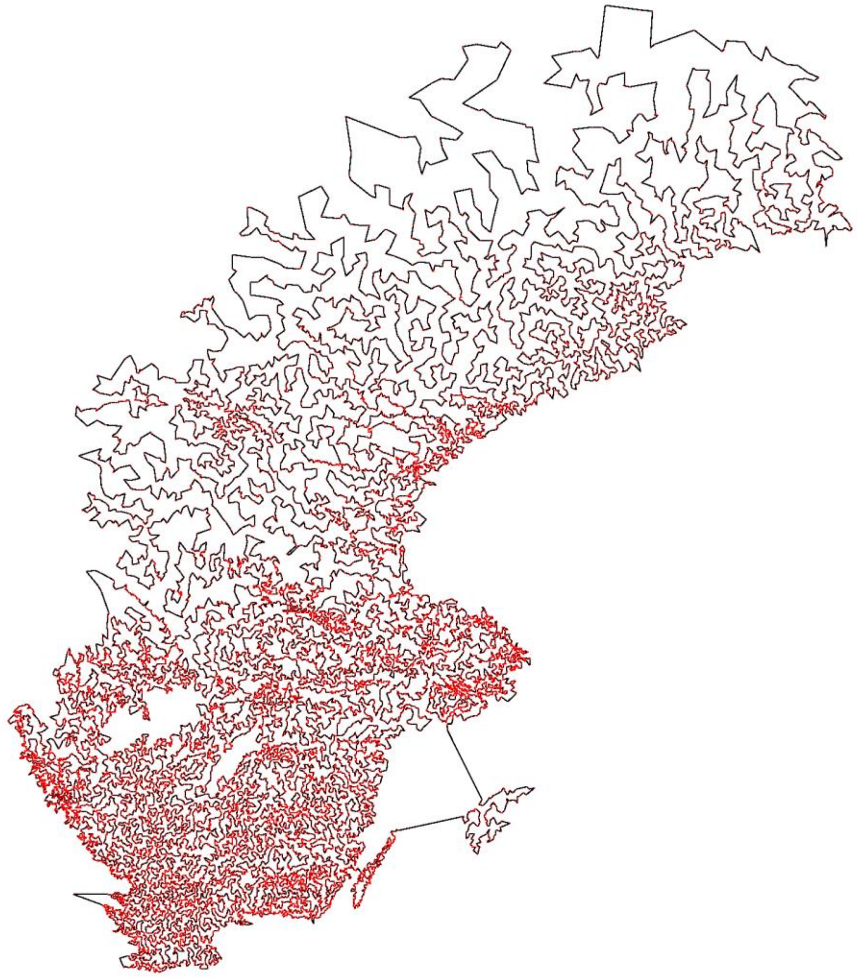
Fredkin Gate



a	b	c	a'	b'	c'
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	1	0
0	1	1	0	1	1
1	0	0	1	0	0
1	0	1	1	1	0
1	1	0	1	0	1
1	1	1	1	1	1

TSP is easy

- Arbitrary large instances of TSP can be solved with arbitrarily small energy

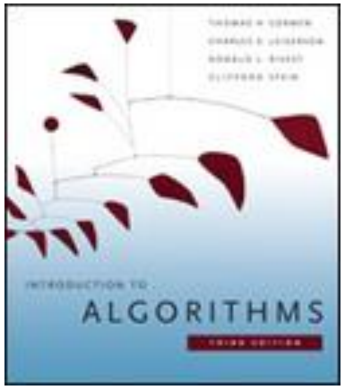


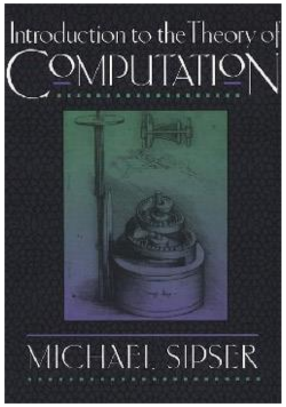
So …

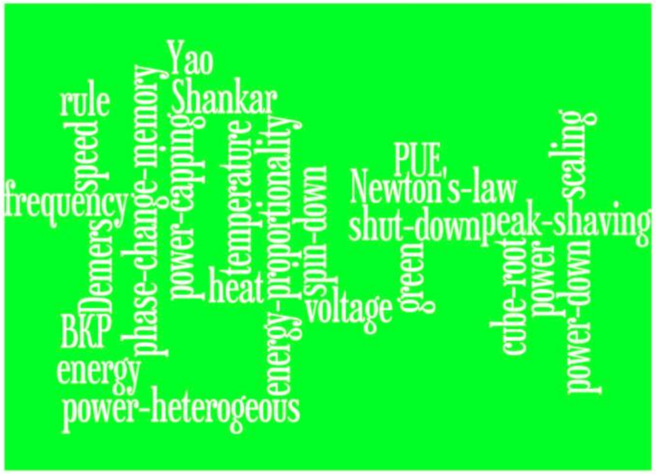
- We need different models to study energy as a computational resource than we use for time and space

Sales Pitch for Green Computing Algorithms

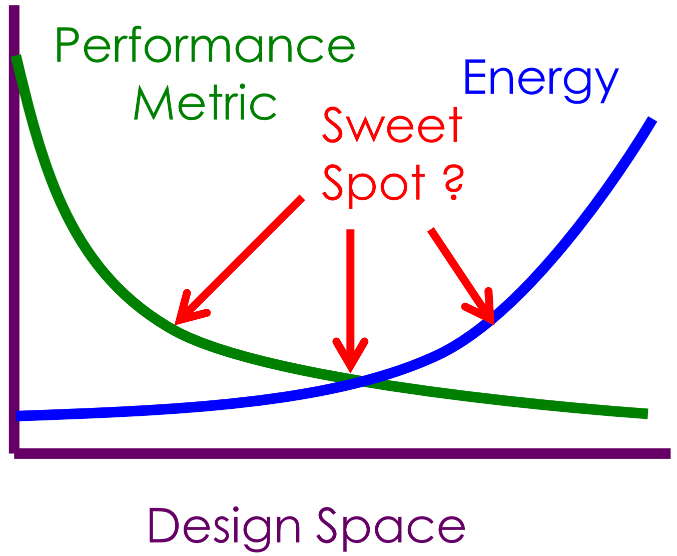
- Build a theory of energy as a computational resource that allows software engineers to reason abstractly about power, energy and temperature as effectively as they can currently abstractly reason about time and space







Current State Of Theory of Energy as a Computational Resource: Energy vs. Performance Tradeoffs



Ubiquitous figure at Green Computing Conferences

Road Map

- Wildly optimistic goal for the rest of this talk
  - Brief digression on algorithmic research
  - Managing speed scalable processors
    - Offline algorithm design and analysis using KKT conditions
    - Online algorithm design and analysis using potential functions
    - Online algorithm design and analysis using dual fitting of Lagrangian dual
- Talk 2: Routing in a network of routers that are speed scalable and that may be shutdown
- Talk 3: Near-threshold computing and energy efficient circuit design

Merriam-Webster Definition of Research

- investigation or experimentation aimed at the discovery and interpretation of facts, revision of accepted theories or laws in the light of new facts, or practical application of such new or revised theories or laws

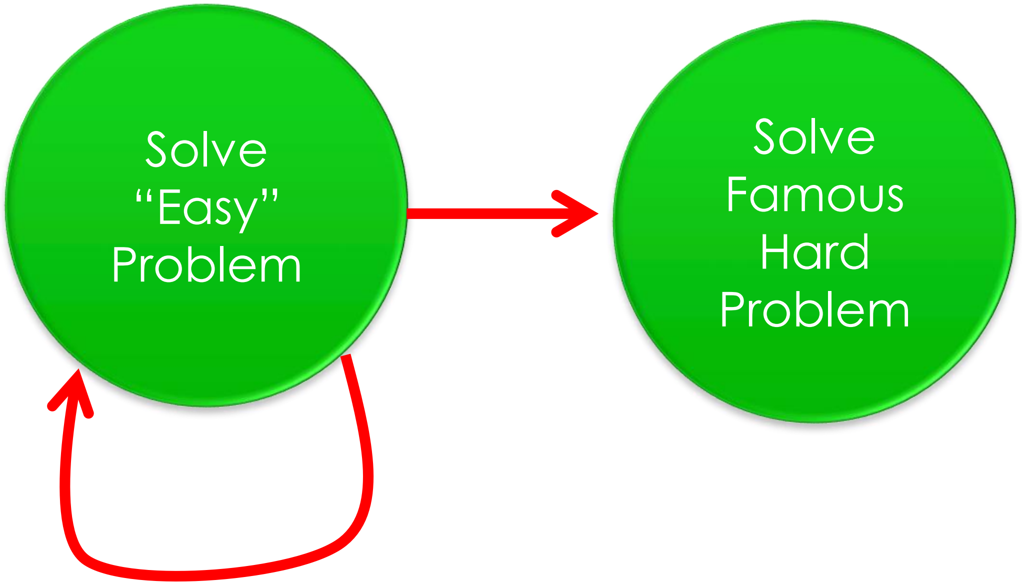
One Type of Algorithmic Research

- Solve a well-known hard/important open problem that has stumped many
  - Analogy: Climbing an unclimbable mountain
  - Examples:
    - O(1)-approximation algorithm for ATSP
    - 4/3 approximation for TSP
  - Advantages: Fame is virtually assured if successful
  - Disadvantages: Failure common

Another Type of Algorithmic Research

- Applying known techniques to "easy" problems where its not surprising these techniques work
  - Analogy: Climbing an foothill that others have bypassed
  - Advantages:
    - Often good way to get into research for students
    - In some educational systems that emphasize quantity over quality, this is a good strategy to achieve promotion
  - Disadvantages: Won't get you invited as a speaker at ADFOCS

Nondeterministic Automata for Progress of Most Algorithms PhD Students



Type of Algorithmic Research that Green Computing Has Fallen Into

- Problem Discovery/Mining. Starting new line of research.
  - Analogy: Discovering new lands
  - Advantages: Can pay off big
  - Disadvantages:
    - Often difficult to find algorithmically interesting problems.
    - Often difficult to gain (rapid) acceptance within established community.
  - Requires additional skills
    beyond problem solving.

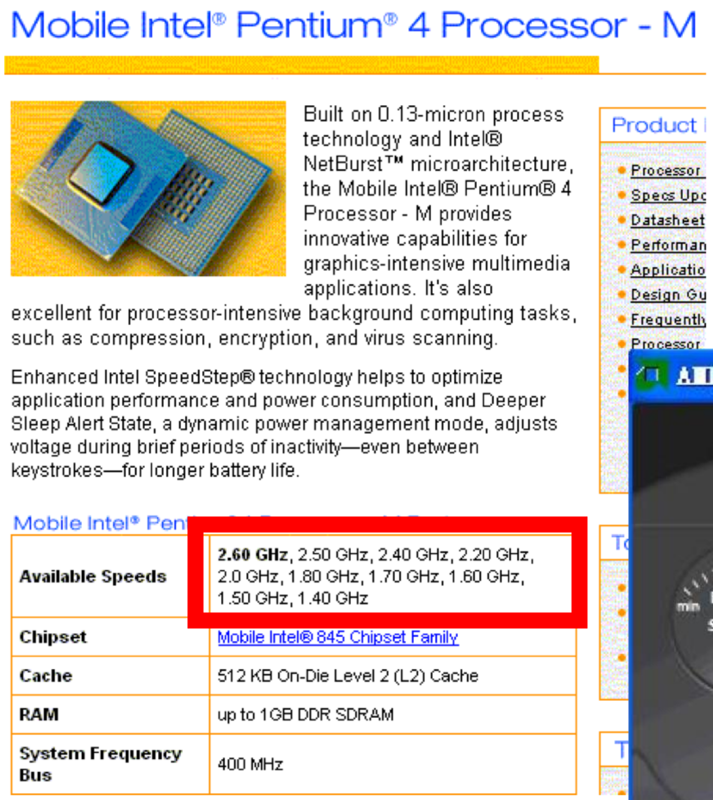
You have to know how to search
Mine for problems. Requires a
Different mentality.

Road Map

- Managing speed scalable processors
  - Introduction to power heterogeneity and speedy scaling
  - Philosophical discussion about modeling
  - Unfortunately long introduction formal model
  - Warmup problems
  - Offline algorithm design and analysis using KKT conditions
  - Online algorithm design and analysis using potential functions
  - Online algorithm design and analysis using dual fitting of Lagrangian dual

Speed Scaling: Power Heterogeneity

Clock	Voltage	Idle	Peak
1.6 GHz	0.925V	59 Watts	78 Watts
1.7 GHz	0.950V	67 Watts	81 Watts
1.8 GHz	0.975V	68 Watts	84 Watts
1.9 GHz	1.000V	68 Watts	86 Watts
2.2 GHz	1.035V	70 Watts	89 Watts
2.3 GHz	1.050V	72 Watts	92 Watts
2.4 GHz	1.075V	74 Watts	94 Watts
2.6 GHz	1.100V	76 Watts	99 Watts
2.7 GHz	1.125V	78 Watts	101 Watts
2.885 GHz	1.175V	83 Watts	110 Watts
2.940 GHz	1.200V	84 Watts	114 Watts
3.060 GHz	1.250V	92 Watts	120 Watts



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.

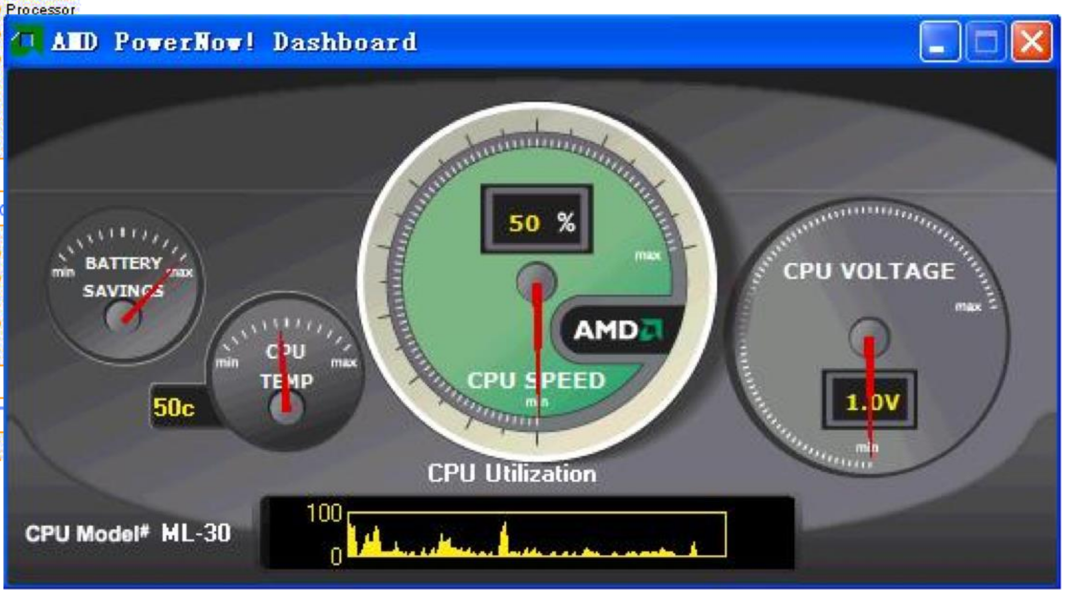
Enhanced Intel SpeedStep® technology helps to optimize application performance and power consumption, and Deeper Sleep Alert State, a dynamic power management mode, adjusts voltage during brief periods of inactivity—even between keystrokes—for longer battery life.

Product I

- Processor
- Specs Upd
- Datasheet
- Performan
- Applicatio
- Design Gu
- Frequently
- Processor

Mobile Intel® Pen

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



Power Heterogeneity

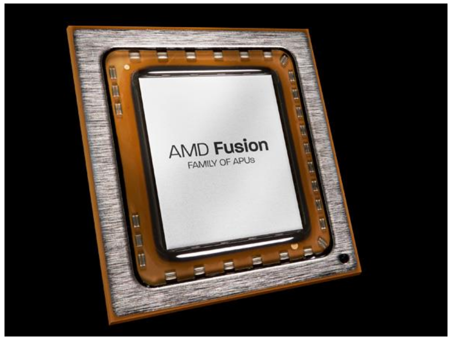
- Physics fact: Faster devices (cars) are less energy efficient than slower devices (cars)
  - Energy efficiency = speed/power
- One reason to have cars of different speed:
  - Take the Ferrari if there is an important event you need to get to quickly, and take the Prius if the event is less important or time-critical.

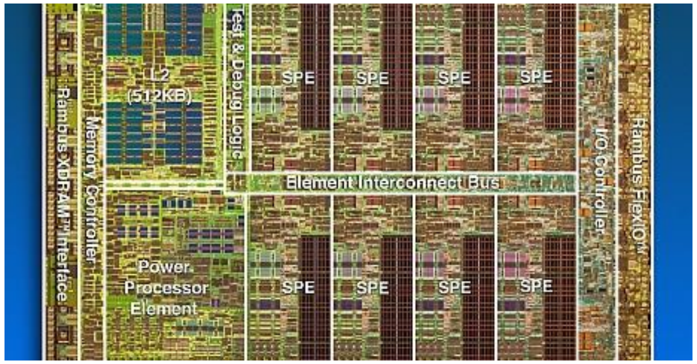
Power Heterogeneity in Processors

- Physics fact: Faster processing is generally less energy efficient than slower processing

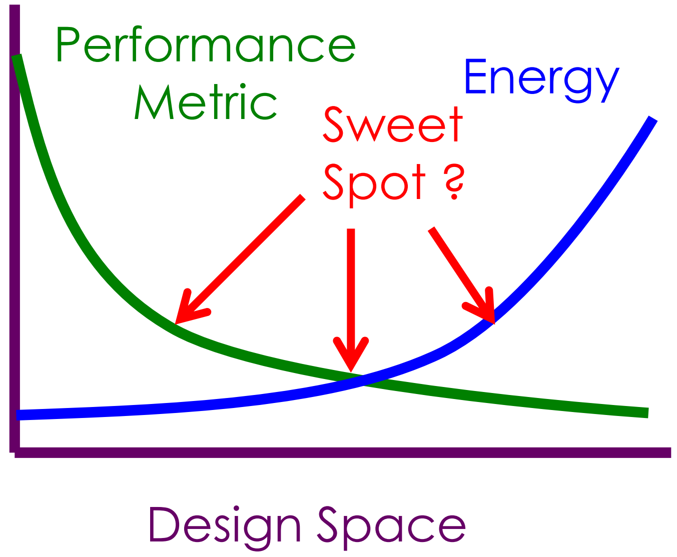
- Power heterogeneous computing technologies
  - Speed scalable processors
  - Heterogenous multiprocessors

- One reason to have processing of different speeds:
  - Use high power processing for important tasks and energy efficient processing for unimportant tasks





- Desired Theorem: Algorithm A gives a near optimal tradeoff between energy usage and performance (how long jobs have to wait, weighted by importance of the jobs).
- What do we have to do first?

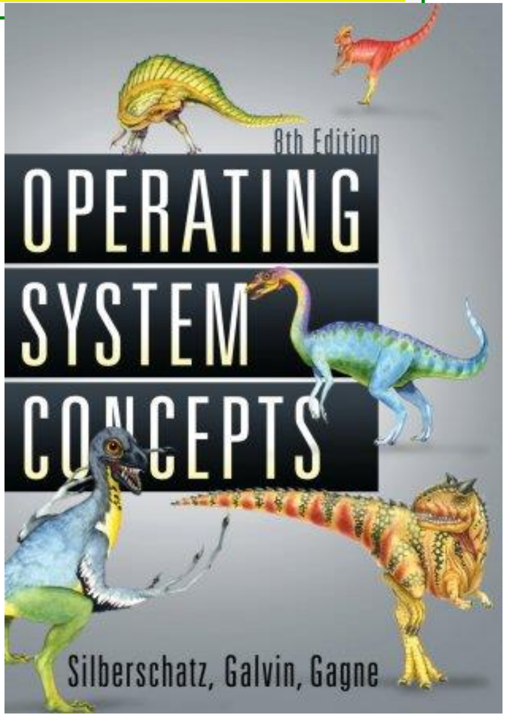


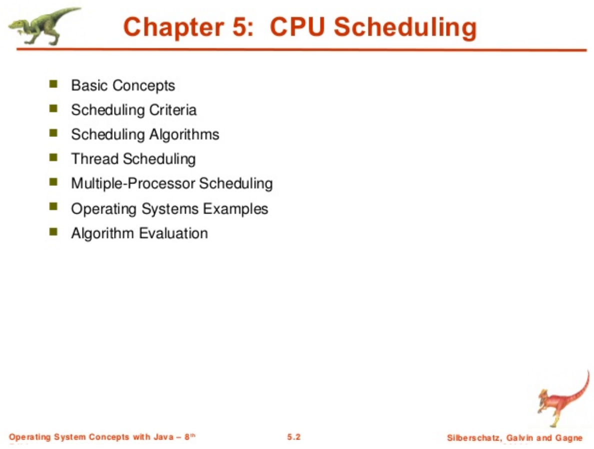
Road Map

- Managing speed scalable processors
  - Introduction to power heterogeneity and speedy scaling
  - Philosophical discussion about modeling
  - Unfortunately long introduction formal model
  - Warmup problems
  - Offline algorithm design and analysis using KKT conditions
  - Online algorithm design and analysis using potential functions
  - Online algorithm design and analysis using dual fitting of Lagrangian dual

We Need To Model

- Processor environment
  - Allowable Speeds
  - Associated Powers
- Job environment
  - Largely inherited
- Performance vs. energy objective
  - Partially inherited




Chapter 5: CPU Scheduling

- Basic Concepts
- Scheduling Criteria
- Scheduling Algorithms
- Thread Scheduling
- Multiple-Processor Scheduling
- Operating Systems Examples
- Algorithm Evaluation

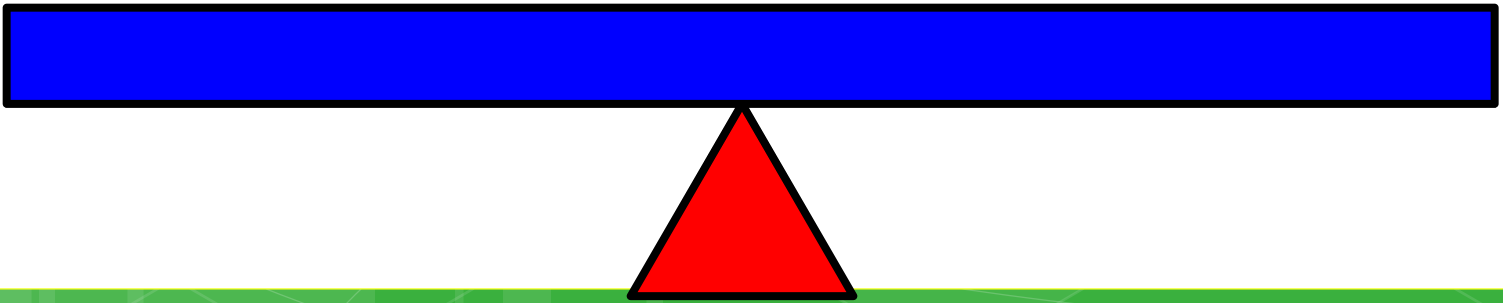
Operating System Concepts with Java – 8th 5.2 Silberschatz, Galvin and Gagne

Algorithmists' View of Science/Theory

- Science research tries to model a complex system by something simple, accurate, amenable to math and predictive. Muthu Muthukrishnan's blog

•Accuracy
•Realism
•Predictive

•Simplicity
•Amenable to math



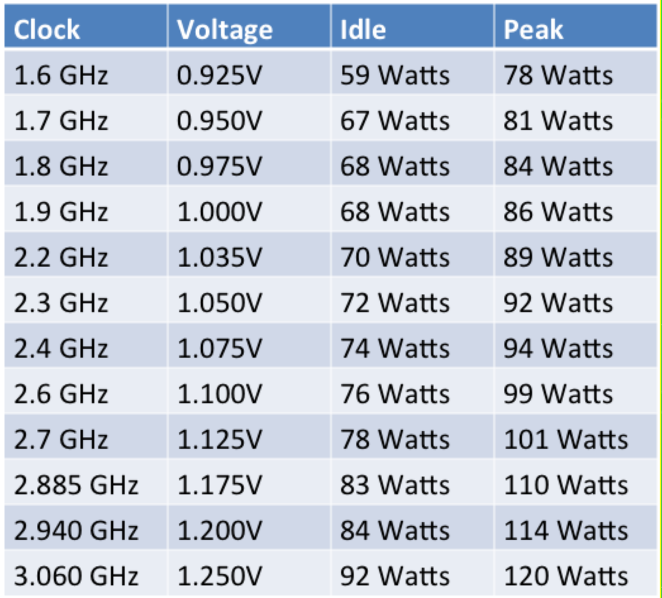
Which is the Best Theoretical Algorithmic Model for Allowable Speeds?

A. 1.6, 1.7, 1.8, 1.9, 2.2, 2.3, 2.4, 2.6, 2.7, 2.885, 2.94, 3.06

B. Arbitrary discrete speeds $s_1, \ldots, s_k$

C. Reals in a range $[0, s_{max}]$ for an arbitrary constant s_{max}

D. Any nonnegative reals



Clock	Voltage	Idle	Peak
1.6 GHz	0.925V	59 Watts	78 Watts
1.7 GHz	0.950V	67 Watts	81 Watts
1.8 GHz	0.975V	68 Watts	84 Watts
1.9 GHz	1.000V	68 Watts	86 Watts
2.2 GHz	1.035V	70 Watts	89 Watts
2.3 GHz	1.050V	72 Watts	92 Watts
2.4 GHz	1.075V	74 Watts	94 Watts
2.6 GHz	1.100V	76 Watts	99 Watts
2.7 GHz	1.125V	78 Watts	101 Watts
2.885 GHz	1.175V	83 Watts	110 Watts
2.940 GHz	1.200V	84 Watts	114 Watts
3.060 GHz	1.250V	92 Watts	120 Watts

Road Map

- Managing speed scalable processors
  - Introduction to power heterogeneity and speedy scaling
  - Philosophical discussion about modeling
  - Unfortunately long introduction formal model
  - Offline algorithm design and analysis using KKT conditions
  - Online algorithm design and analysis using potential functions
  - Online algorithm design and analysis using dual fitting of Lagrangian dual

Formal Model (0)

- Setting: Speed scalable processor
  - Allowable speeds $[0, \infty)$
  - Power function P(s) specifying power as a function of speed s

Formal Model (0)

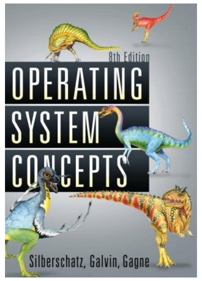
- Setting: Speed scalable processor
  - Allowable speeds [0, ∞)
  - Power function P(s) specifying power as a function of speed s

- Standard Architectural Model
  - Power = dynamic power + static power
    - Static power =constant
      - now comparable to dynamic power.
      - Ignore static power for now
    - Dynamic power ≈ speed^a, $a \approx 3$
    - So for now, think $P(s)=s^3$

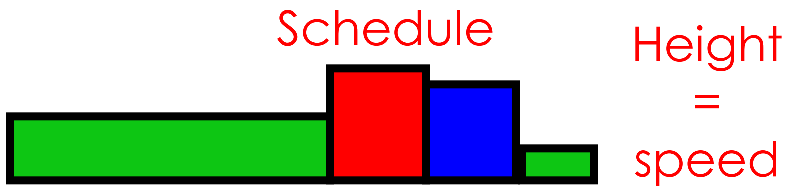
- Of course Energy is power integrated over time

Formal Model (1)

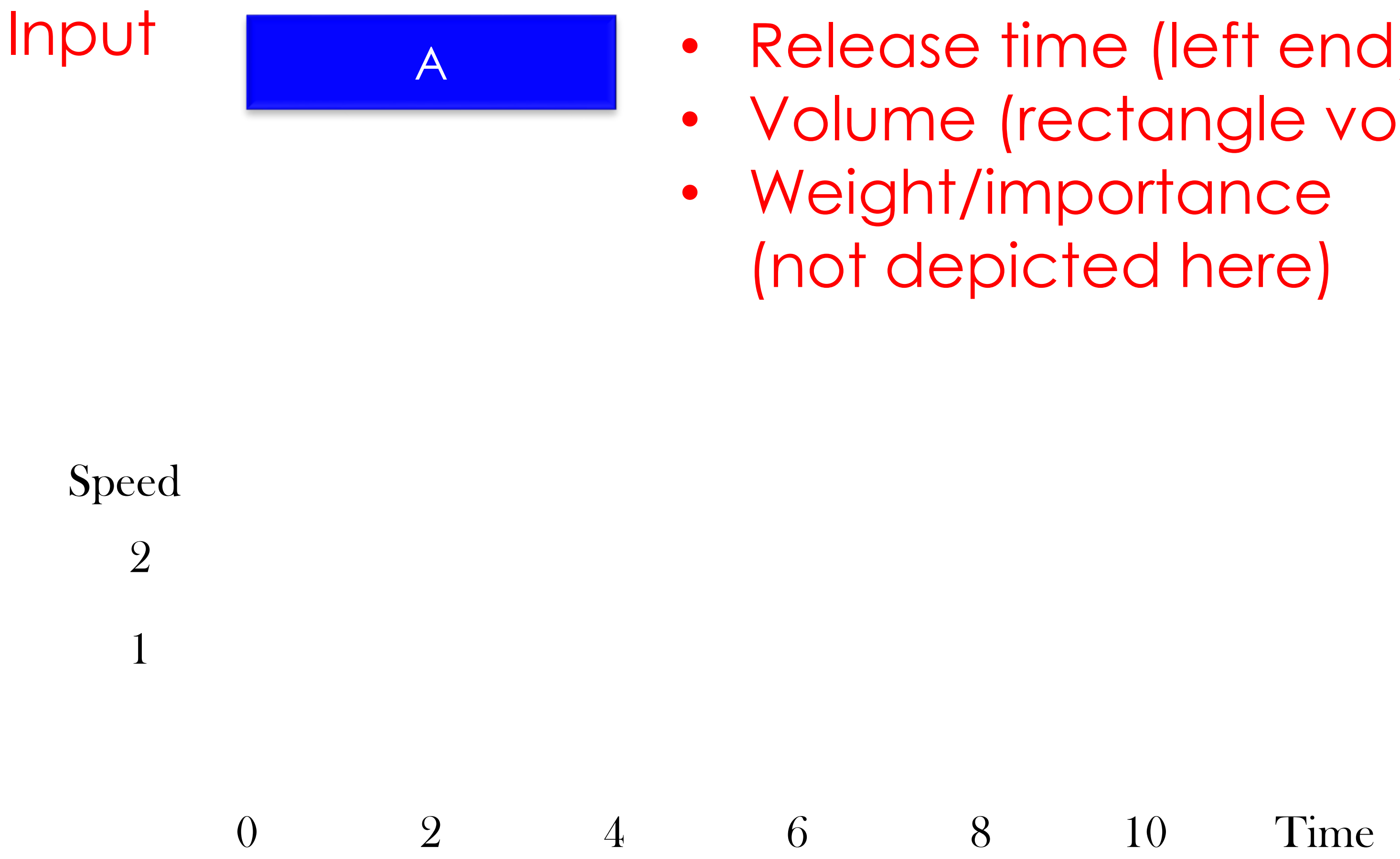
- Setting: Speed scalable processor
  - Allowable speeds [0, ∞)
  - Power function P(s) specifying power as a function of speed s
- Input: Jobs, each j having an
  - Arrival/release time r_j
  - Size/volume/work p_j
  - Importance/weight w_j



- Output: A schedule that specifies for each time
  - The job that is run
  - The speed the processor is run



Online Scheduling

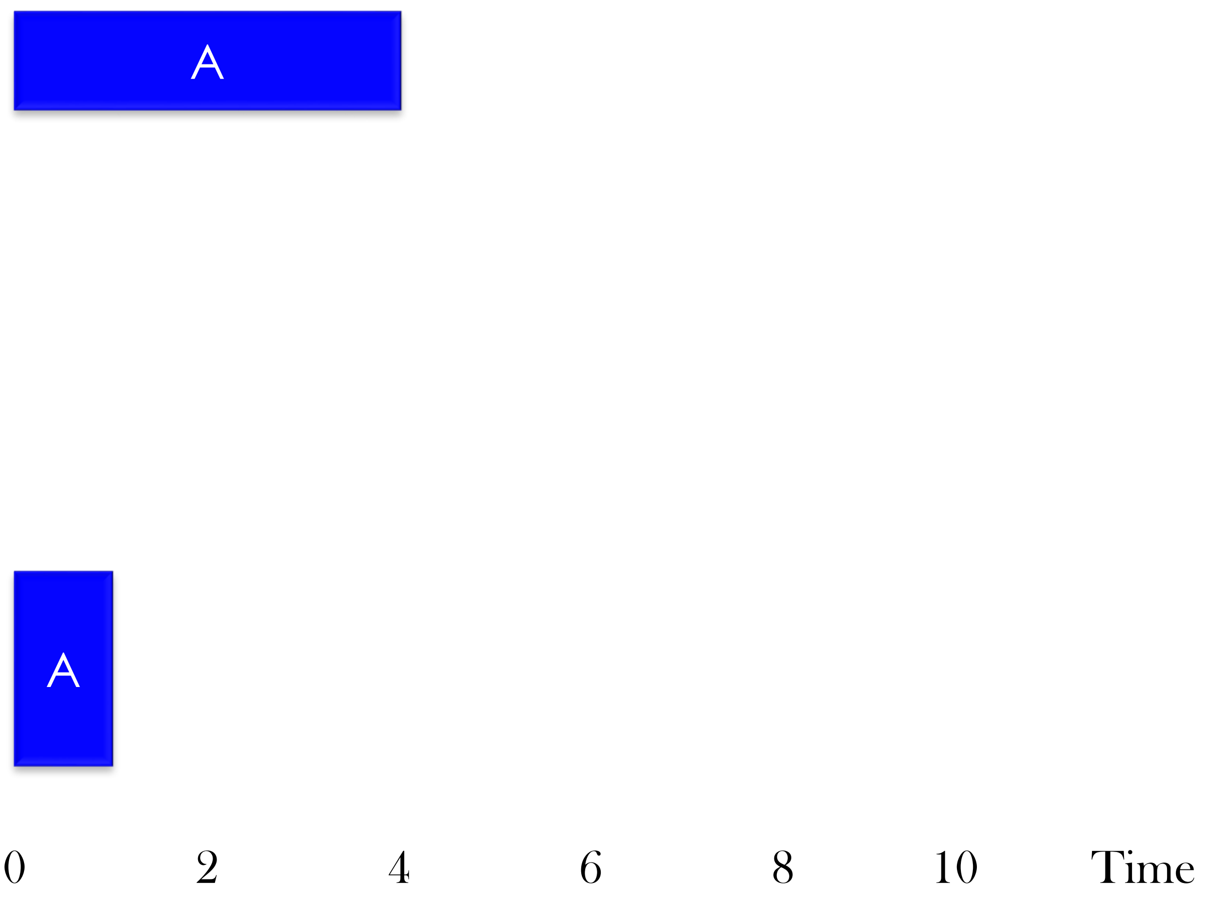


Jobs have

- Release time (left end)
- Volume (rectangle volume)
- Weight/importance (not depicted here)

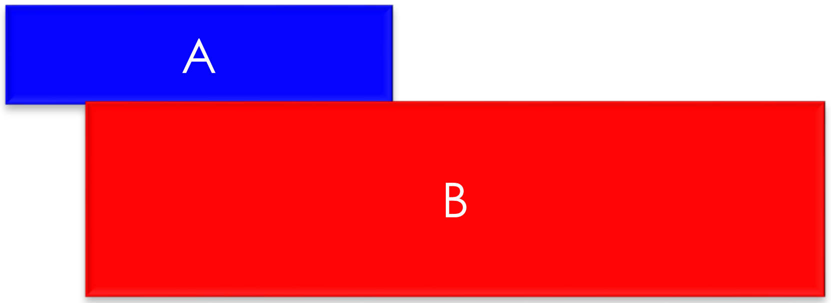
Online Scheduling

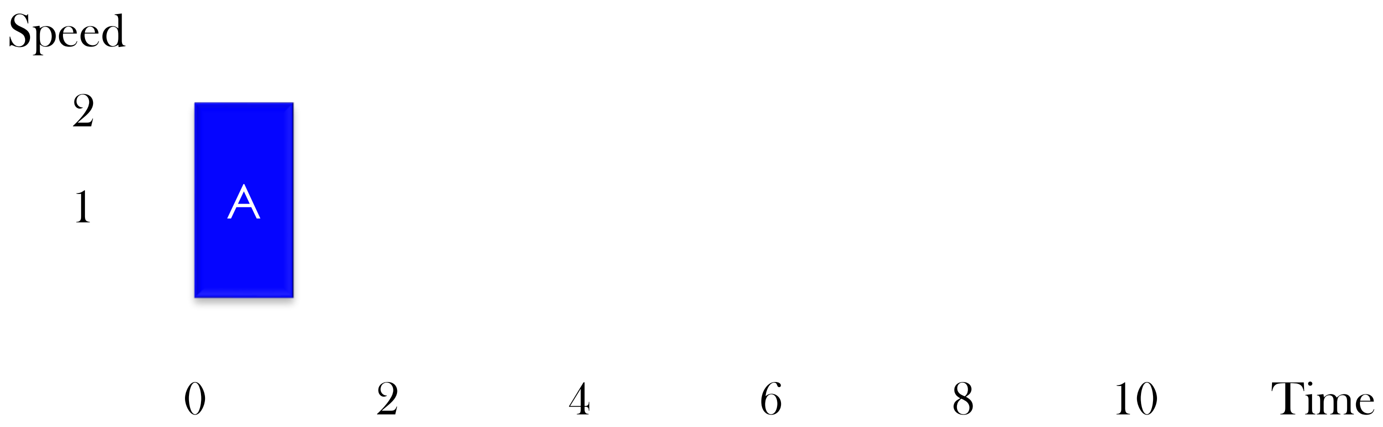
Input



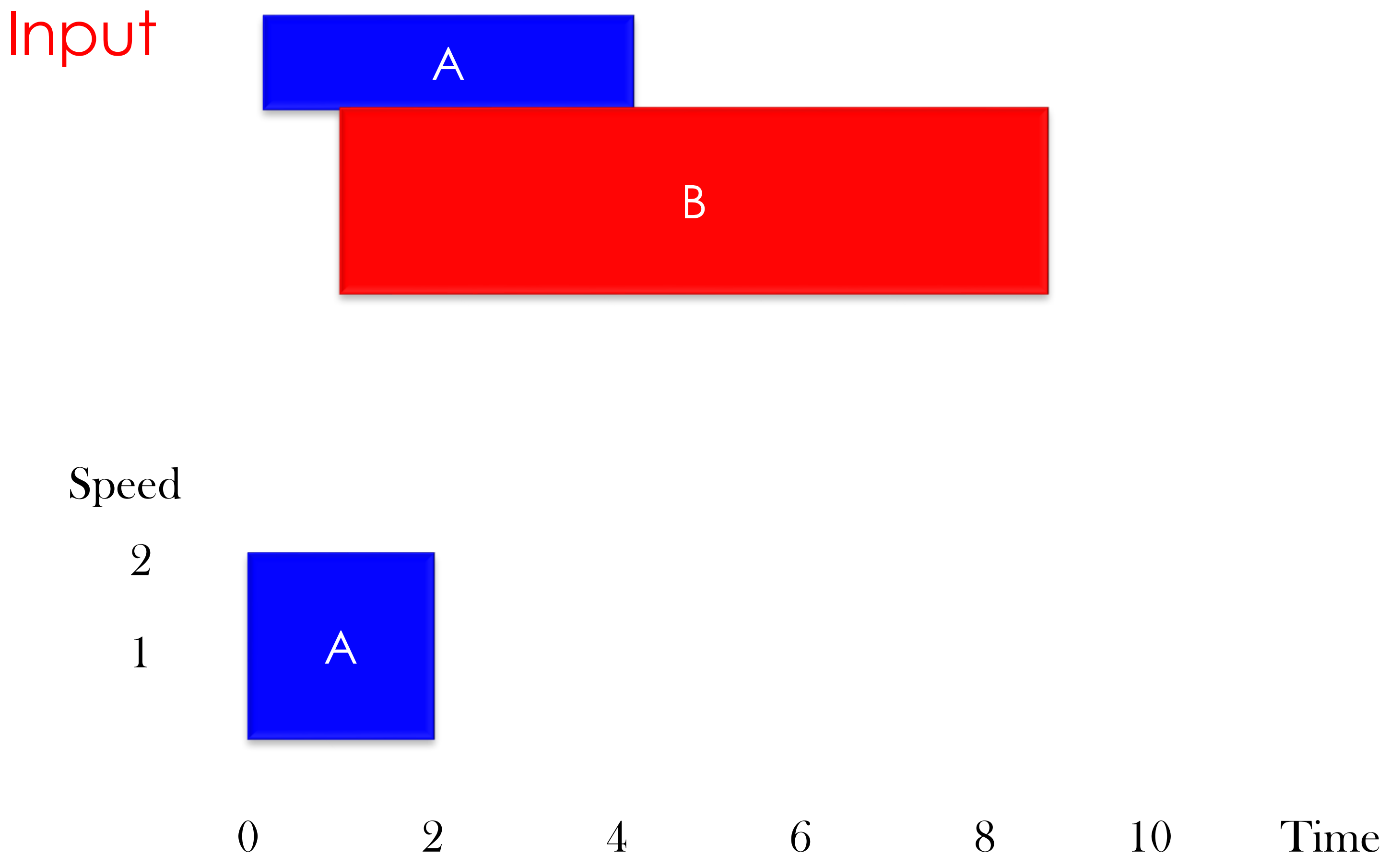
Online Scheduling

Input

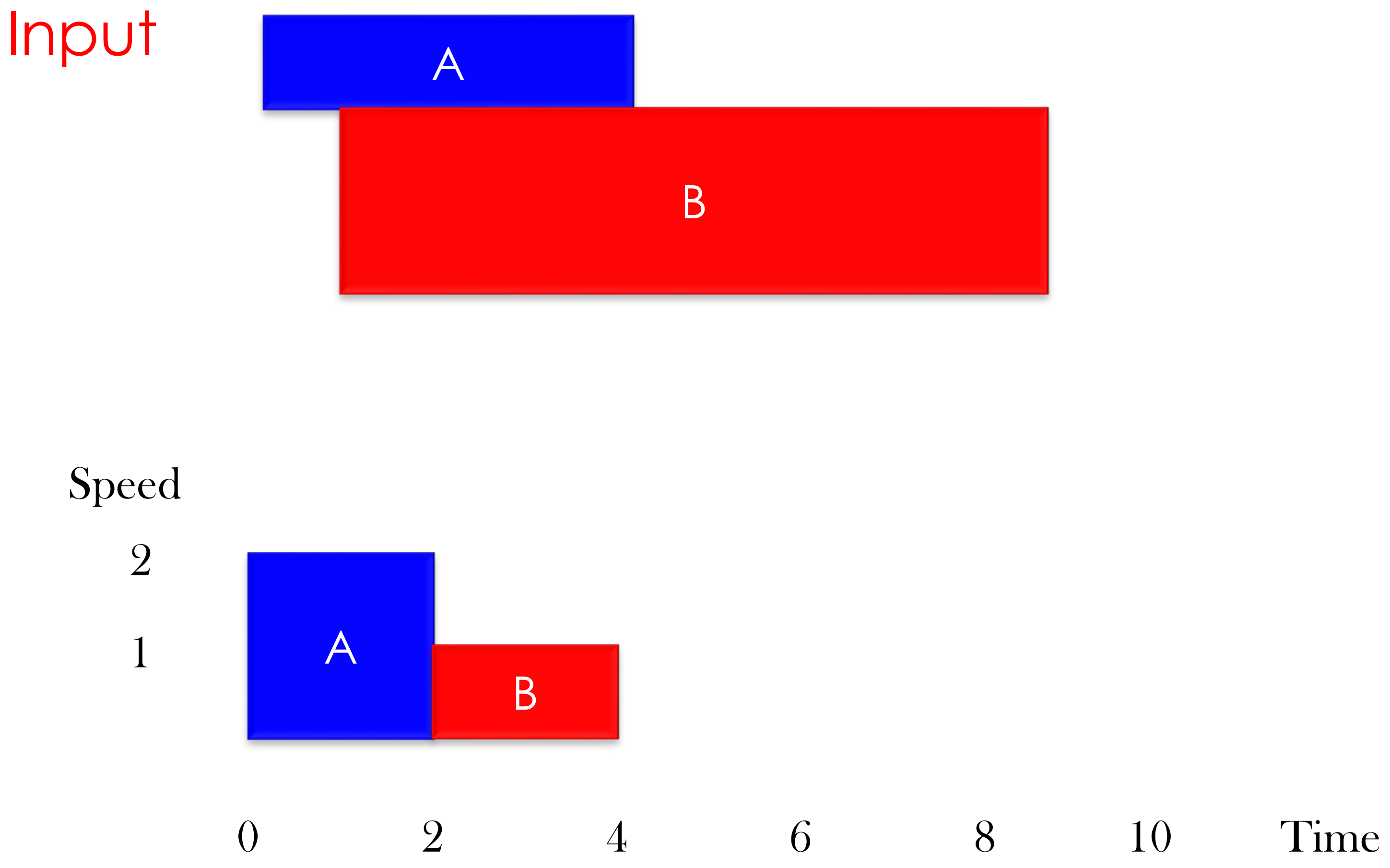




Online Scheduling

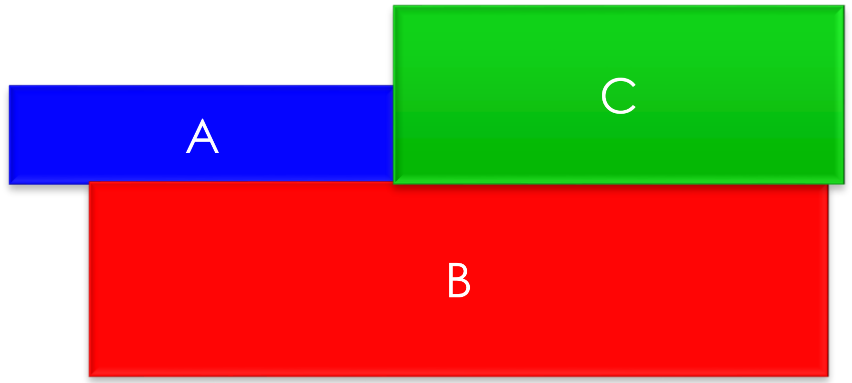


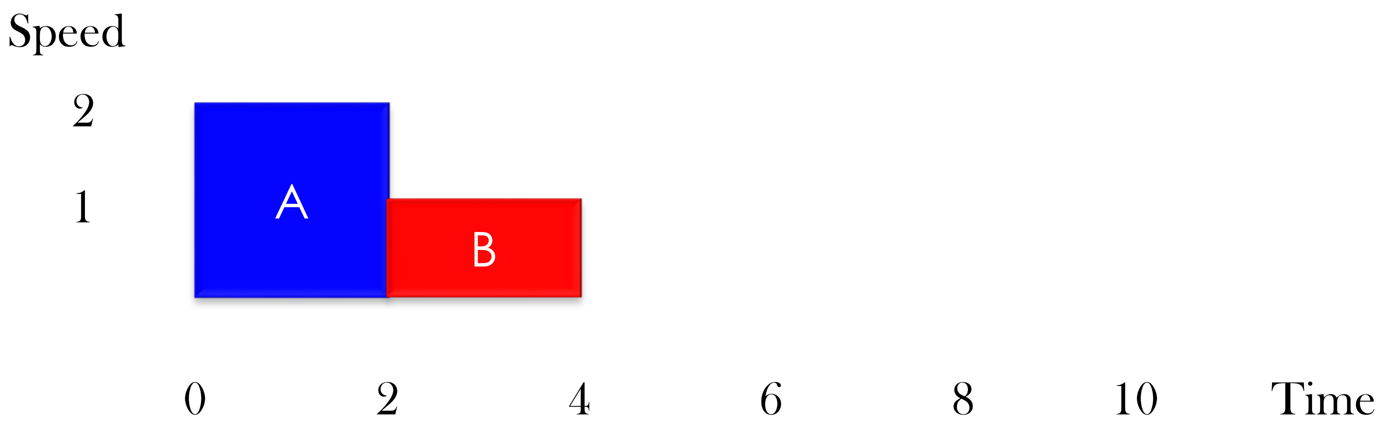
Online Scheduling



Online Scheduling

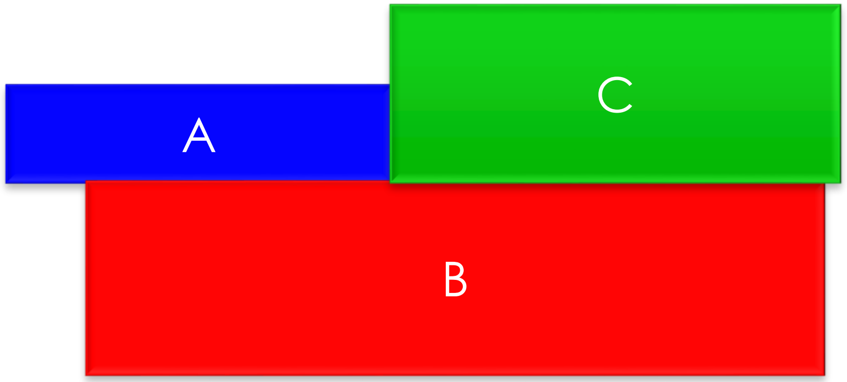
Input

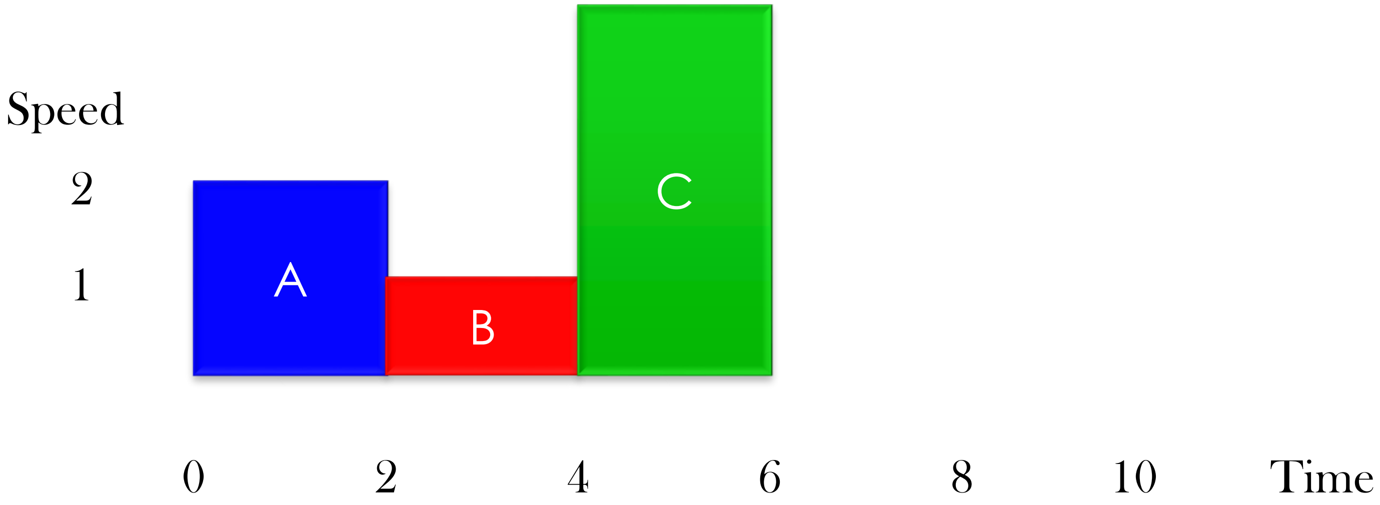




Online Scheduling

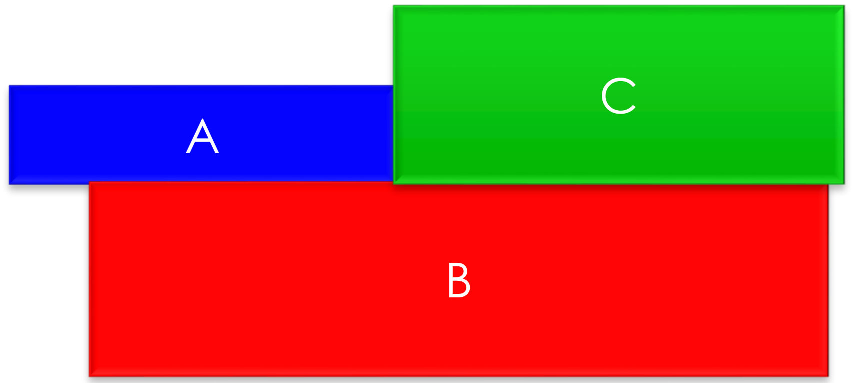
Input

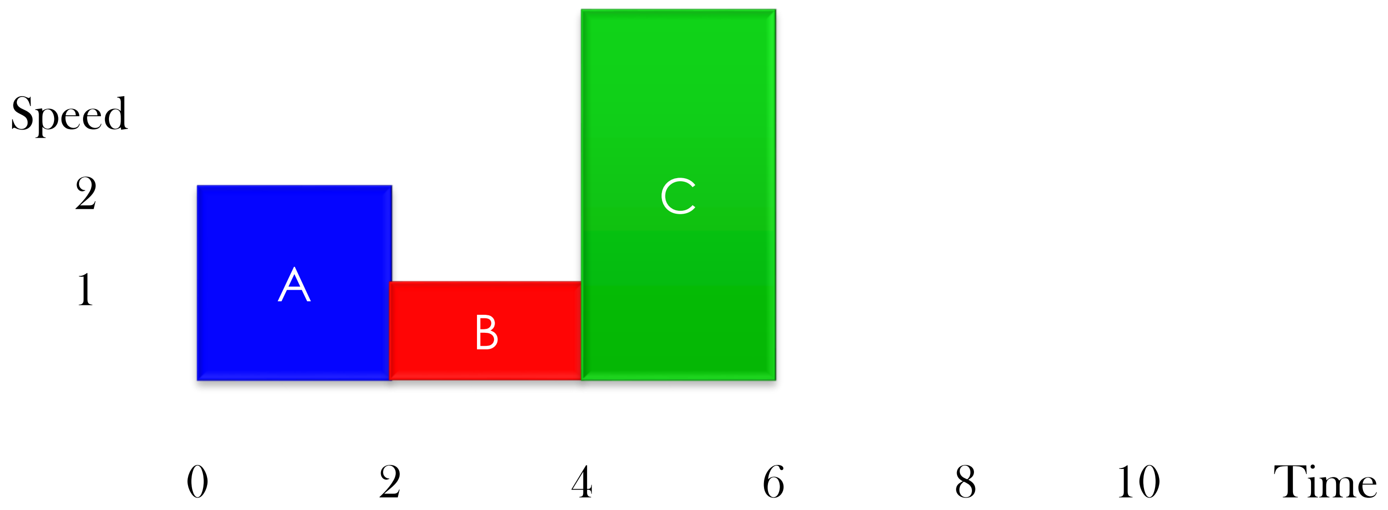




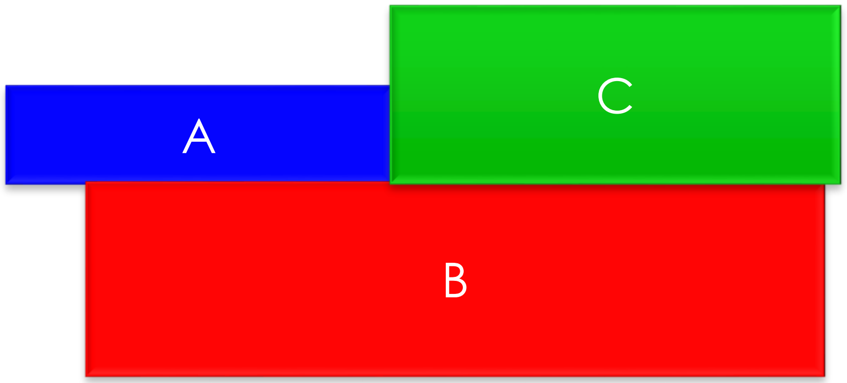
Online Scheduling

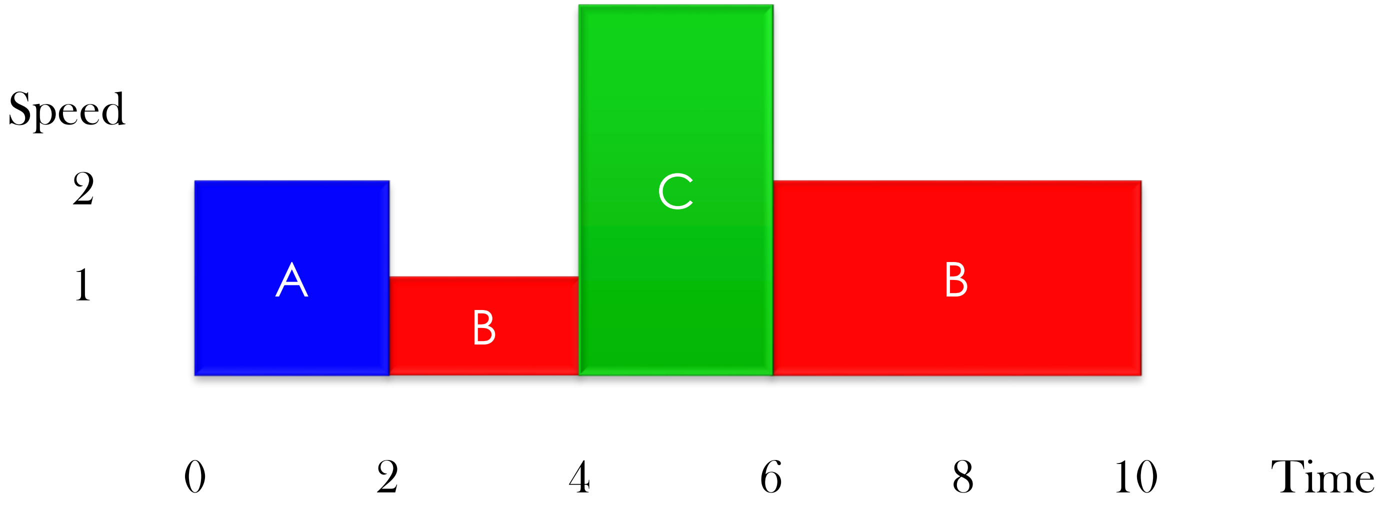
Input





Online Scheduling

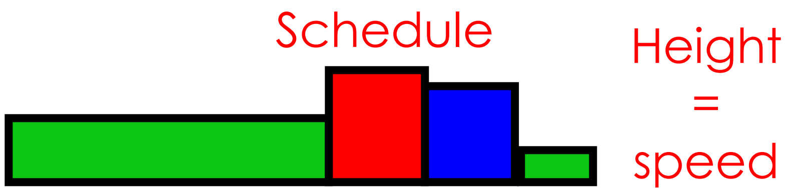


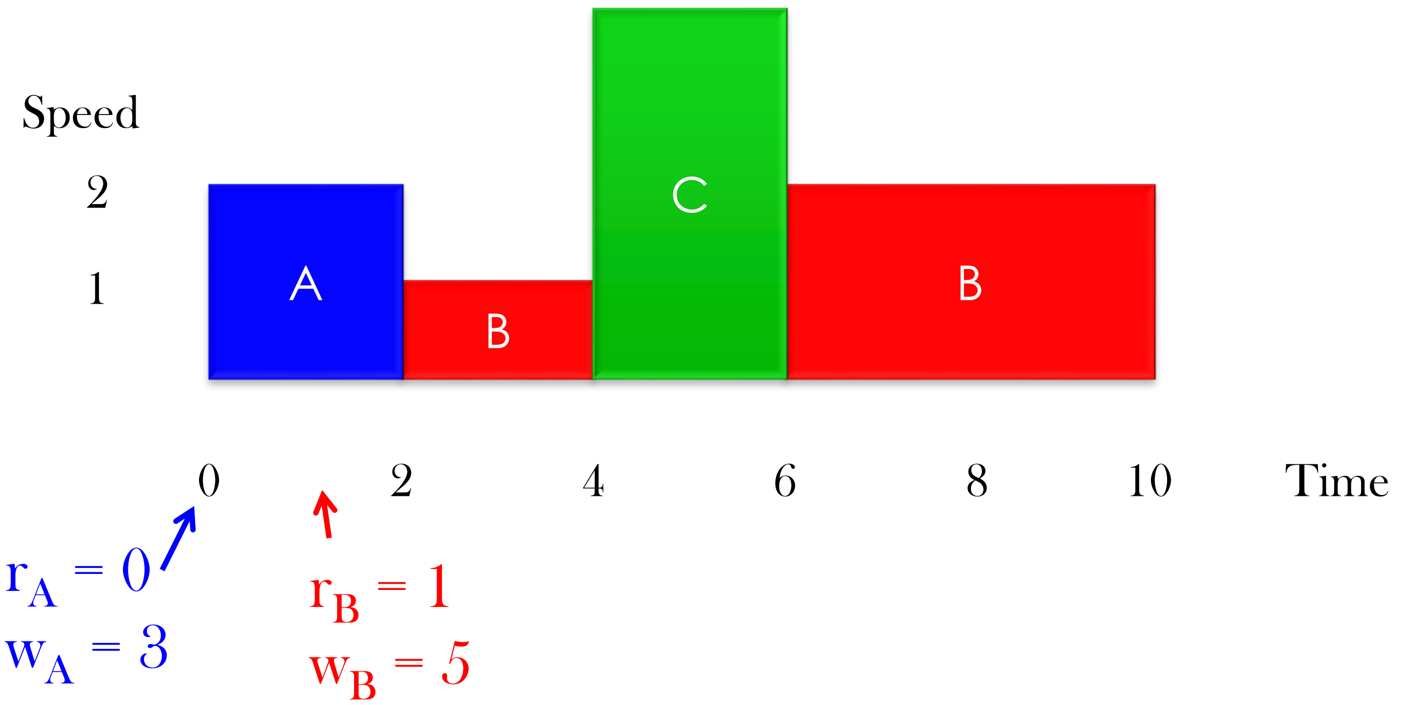


Formal Model (2)

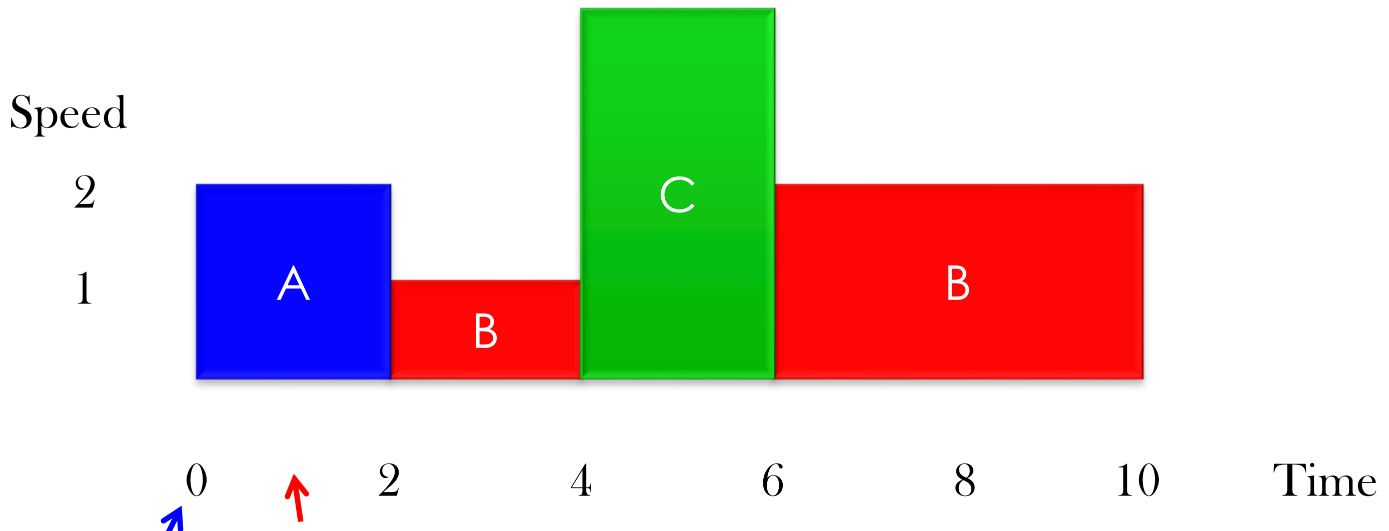
- Performance Objectives
  - Primarily: Weighted Fractional Waiting Time: Total (weighed by importance) time instructions have to wait to be executed

  - Secondarily: Weighted Waiting Time: Total (weighted by importance) time job have to wait to be finished





- Size/work/volume of job B = ?
- Waiting time for job B = ?
- Weighted waiting time for job B = ?
- Fractional weighted waiting time for job A = ?
- Fractional weighted waiting time for job B = ?



$r_A = 0$ $r_B = 1$
$w_A = 3$ $w_B = 5$

- Size/work/volume of job B = 2*1 + 4*2 = 10
- Waiting time for job B = 10 – 1 = 9
- Weighted waiting time for job B = 9* 5 = 45
- Fractional weighted waiting time for job A = 3*1
- Fractional weighted waiting time for job B =
  5((2/10)(3-1) + (8/ 10) (8-1))

Digression: For Which Objective Can the Optimal Schedule Be Easily Computed if The Speed is Constant?

- Performance Objective
  - Weighted Fractional Waiting Time: Total (weighed by importance) time <u>instructions</u> have to wait to be executed
  - Weighted Waiting Time: Total (weighted by importance) time job have to wait to be finished

Digression: For Which Objective Can the Optimal Schedule Be Easily Computed if The Speed is Fixed?

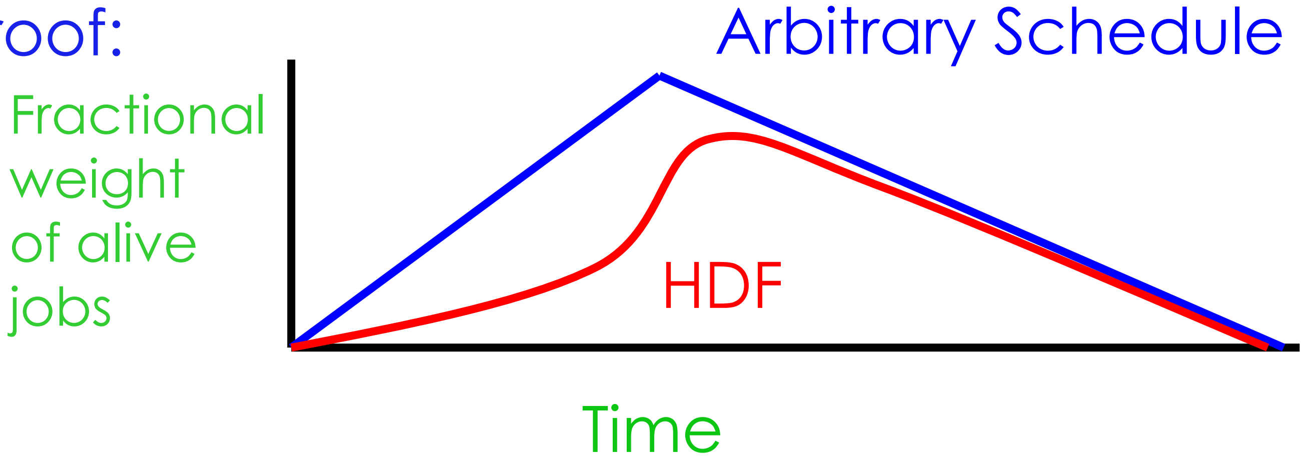
- Performance Objective
  - Weighted Fractional Waiting Time: : Total (weighed by importance) time instructions have to wait to be executed
    - Online algorithm Highest Density First (HDF) is optimal.
    - HDF always runs the job with the highest density
    - Density = weight/work
  - Weighted Waiting Time: Total (weighted by importance) time job have to wait to be finished
    - NP-hard by reduction from partition

Local Competitiveness

- Theorem: HDF is optimal for fractional weighted waiting time
- Proof: ?

Local Competitiveness

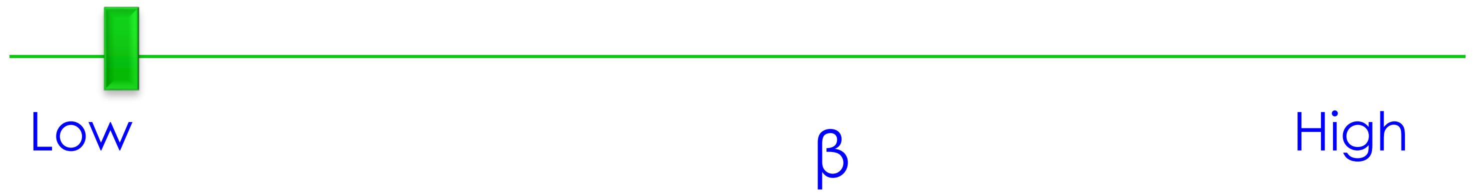
- Theorem: HDF is optimal for fractional weighted waiting time
- Proof:



Fractional weighted waiting time=
$\int_t$ Fractional weight of alive jobs at time t dt

Formal Model (3)

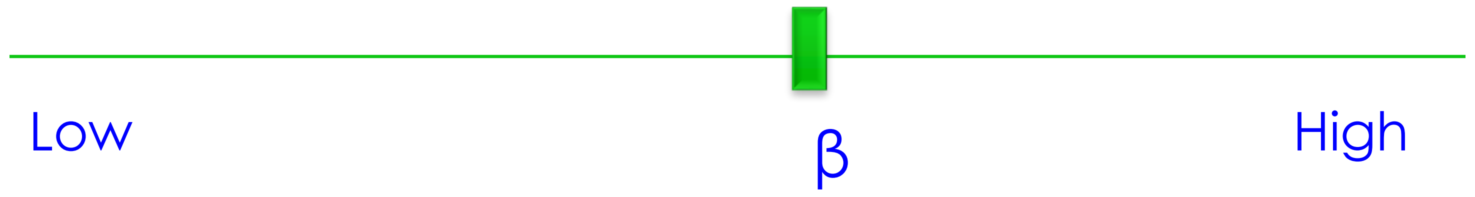
- Slider specifying β, relative importance of energy to waiting time



- βspecifies how much improvement in weighted waiting time is sufficient to justify the expenditure of one more unit of energy

Formal Model (3)

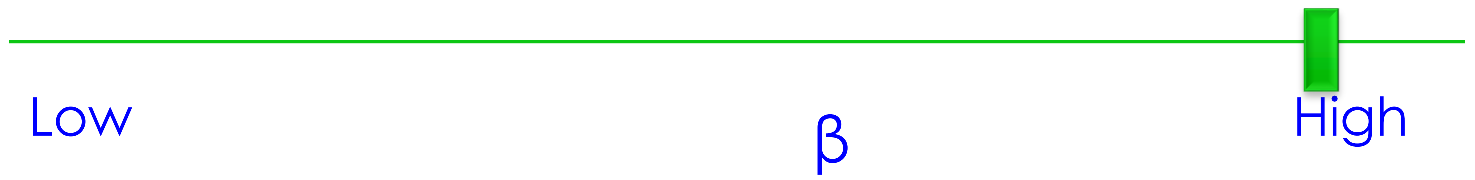
- Slider specifying β, relative importance of energy to waiting time



- βspecifies how much improvement in weighted waiting time is sufficient to justify the expenditure of one more unit of energy

Formal Model (3)

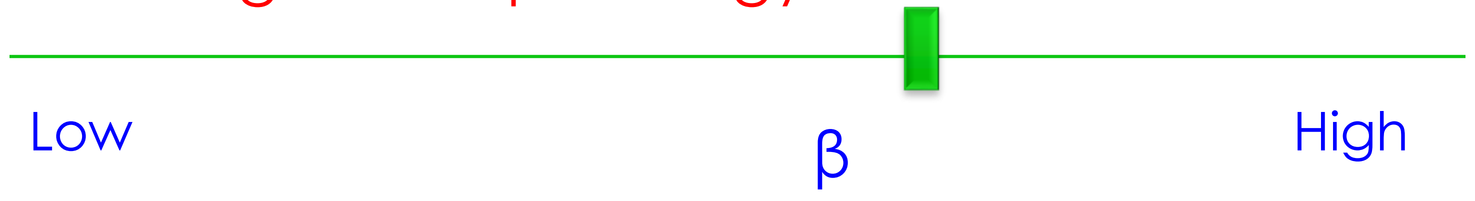
- Slider specifying β, relative importance of energy to waiting time

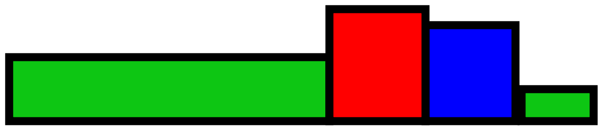


- βspecifies how much improvement in performance is sufficient to justify the expenditure of one more unit of energy

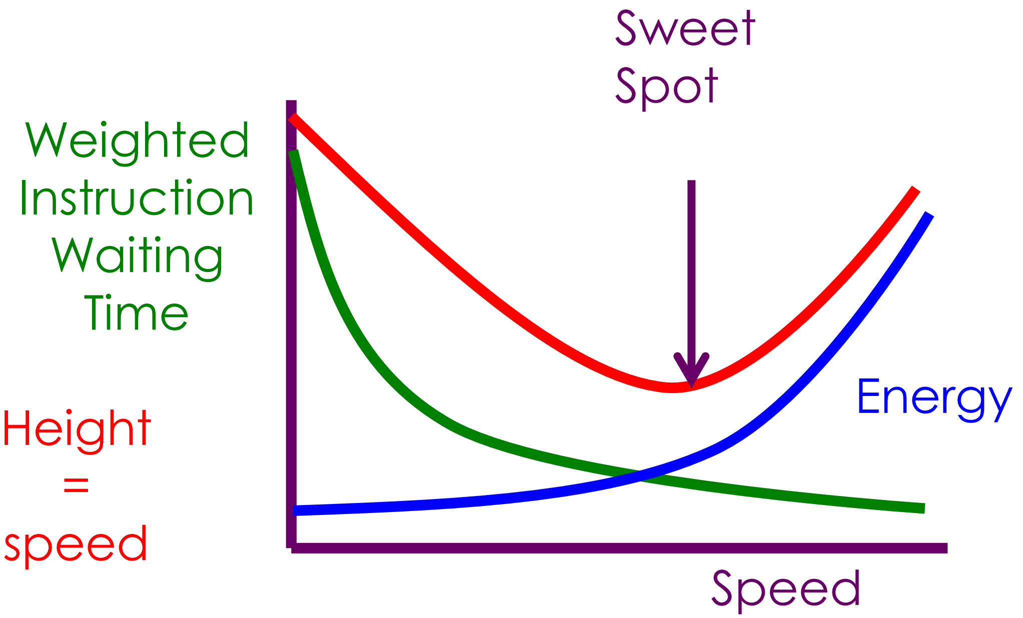
Formal Model (4)

- Final objective: minimize weighted fractional waiting time + β* energy





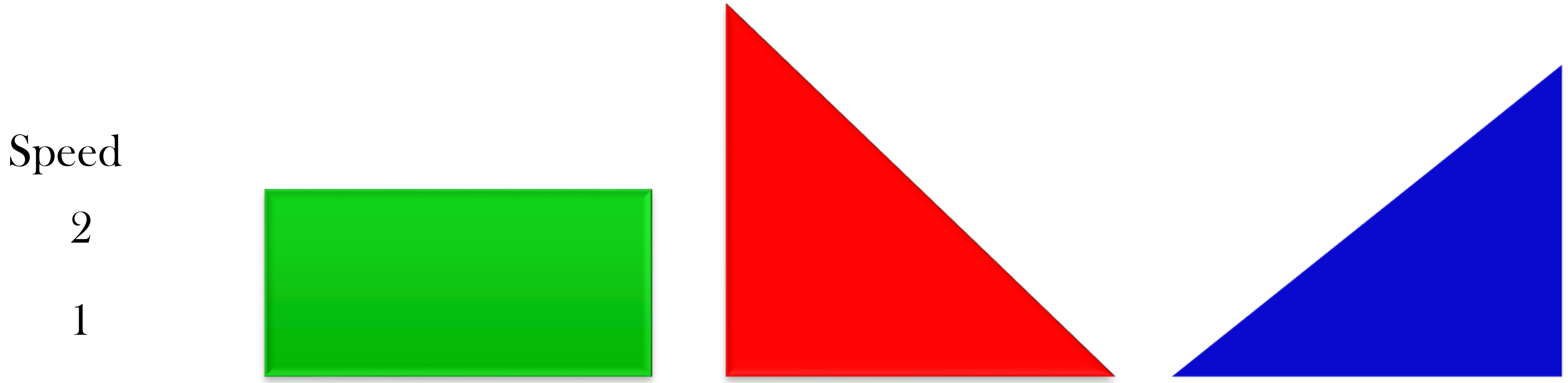
Height = speed



Road Map

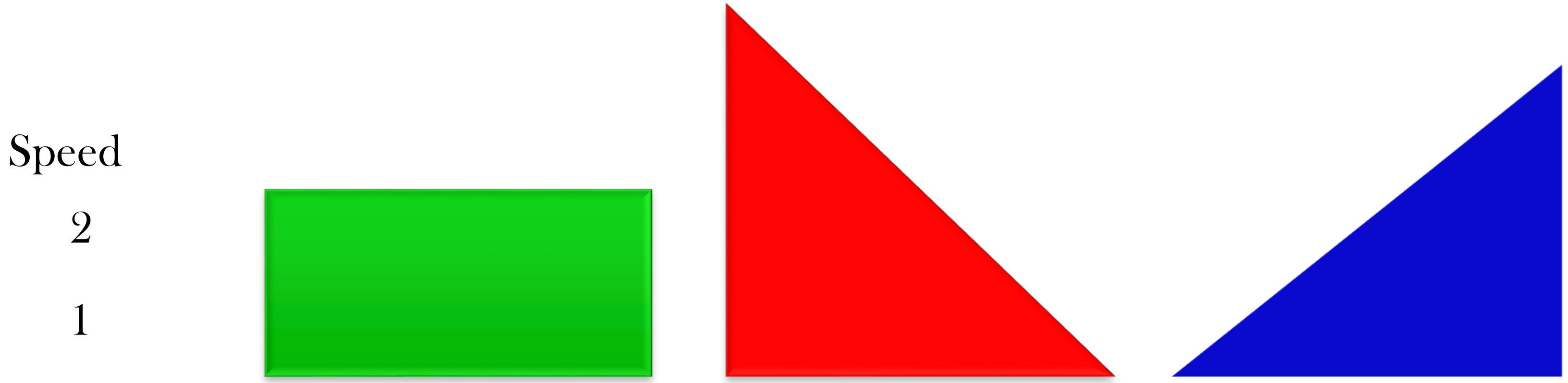
- Managing speed scalable processors
  - Introduction to power heterogeneity and speedy scaling
  - Philosophical discussion about modeling
  - Unfortunately long introduction formal model
  - Offline algorithm design and analysis using KKT conditions
  - Online algorithm design and analysis using potential functions
  - Online algorithm design and analysis using dual fitting of Lagrangian dual

Consider Following 3 Schedules for 1 Job



- For weighted waiting time + energy:
  - Is red or blue schedule better?
  - Which schedules have the “right” shape?
- For weighted fractional waiting time + energy
  - Is red or blue schedule better?
  - Which schedules have the “right” shape?

Consider Following 3 Schedules for 1 Job



- For weighted waiting time + energy:
  - Is red or blue schedule better? same
  - Which schedules have the “right” shape? green
- For weighted fractional waiting time + energy
  - Is red or blue schedule better? red
  - Which schedules have the “right” shape? red

What is the Optimal Schedule for one with the Objective of Weighted Job Waiting Time plus Energy?

- Inputs:
  - w = weight importance of job
  - p = work/volume of job
  - Assume power = speed cubed
- Output:
  - Speed s

- Objective value = ?

What is the Optimal Schedule for one with the Objective of Weighted Job Waiting Time plus Energy?

- Inputs:
  - w = weight importance of job
  - p = work/volume of job
  - Assume power = speed cubed
- Output:
  - Speed s
- Objective value = w p/s + (p/s) s^3
- Optimal Solution: s ≈

What is the Optimal Schedule for one with the Objective of Weighted Job Waiting Time plus Energy?

- Inputs:
  - w = weight importance of job
  - p = work/volume of job
  - Assume power = speed cubed
- Output:
  - Speed s

- Objective: $w\ p/s + (p/s)\ s^3$

- Optimal Solution: $s \approx (w)^{1/3}$
  - Or $s = (w)^{1/3}/(2)^{1/3}$

What is the Optimal Schedule for one with the Objective of Weighted Job Waiting Time plus Energy?

- Inputs:
  - w = weight importance of job
  - p = work/volume of job
  - Assume power = speed cubed
- Output:
  - Speed s

- Objective: $w\, p/s + (p/s)\, s^3$

- Solution: $s \approx (w)^{1/3}$

- Optimal objective value $\approx p\, w^{2/3}$
  - $\approx p\, w^{(1-1/a)}$ for general a
  - $\approx w^{(2-1/a)}$ in unit density case that p=w

What is the Optimal Schedule for one with the Objective of Fractional Weighted Waiting Time plus Energy?

- Inputs:
  - w = weight importance of job
  - p = work/volume of job
  - Assume power = speed cubed
- Output:
  - Speed s(t) at time t
- Objective value = ?
  - Subject to ?

What is the Optimal Schedule for one with the Objective of Fractional Weighted Waiting Time plus Energy?

- Inputs:
  - w = weight importance of job
  - p = work/volume of job
  - Assume power = speed cubed
- Output:
  - Speed s(t) at time t
- Objective: Σ_t (w t s(t)/p + s(t)^3)
  - Subject to: Σ_t s(t) dt = p
- How do you solve this sort of optimization problem?

Method of Lagrange Mulitpliers

- Min f(x, y, z) subject to g(x, y, z) =0
- A necessary condition for optimality is that there there exist Lagrange multiplier λ such that:
  - d f(x, y, z)/dx + λ d g(x, y, z)/dx =0,
  - d f(x, y, z)/dy + λ d g(x, y, z)/dy =0, and
  - d f(x, y, z)/dz + λ d g(x, y, z)/dz =0

Method of Lagrange Mulitpliers

- Min f(x, y, z) subject to g(x, y, z) =0
- A necessary condition for optimality is that there there exist Lagrange multiplier λ such that:
  - d f(x, y, z)/dx = λ d g(x, y, z)/dx,
  - d f(x, y, z)/dy = λ d g(x, y, z)/dy, and
  - d f(x, y, z)/dz = λ d g(x, y, z)/dz

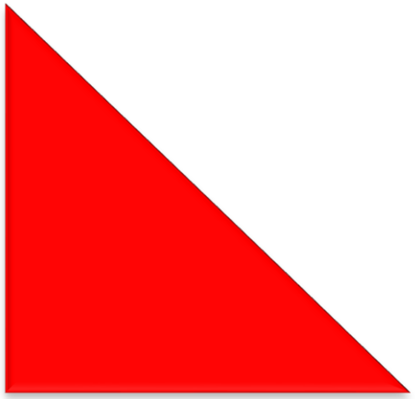
What is the Optimal Schedule for one with the Objective of Fractional Weighted Waiting Time plus Energy?

- Objective: Σ_t (w t s(t)/p + s(t)^3)
  - Subject to: Σ_t s(t) dt = p

- Solution via method of Lagrange multipliers:

  w t/p + 3s(t)^2 = λ
  - So, Hypopower 3s(t)^2 = (λ- w t/p)

Hypopower = Deriviative of Power wrt speed



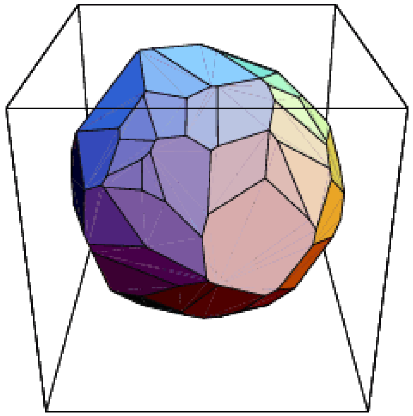
Road Map

- Managing speed scalable processors
  - Introduction to power heterogeneity and speedy scaling
  - Philosophical discussion about modeling
  - Unfortunately long introduction formal model
  - Offline algorithm design and analysis using KKT conditions
  - Online algorithm design and analysis using potential functions
  - Online algorithm design and analysis using dual fitting of Lagrangian dual

Convex Programming Formulation

- Recall problem: Find schedule that minimizes weighted fractional waiting time plus energy
  - P is power function, think P(s) ≈ s^3
  - w_j = importance, p_j =size, and r_j = arrival time

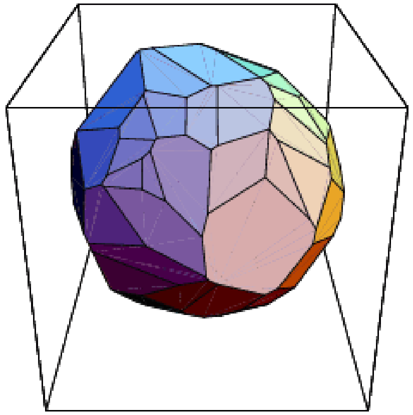
- Convex Optimization:
  - Min convex function f(x)
  - Subject to x in some convex region



- Always key question: What should the variables be?

Convex Programming Formulation

- Recall problem: Find schedule that minimizes weighted fractional waiting time plus energy
  - P is power function, think $P(s) \approx s^3$
  - w_j = importance, p_j =size, and r_j = arrival time
- Convex Optimization:
  - Min convex function f(x)
  - Subject to x in some convex region



- Variable x_{jt} = number of instructions of job j done at time t
- What are the constraints?

Convex Programming Formulation

- Min $\Sigma_j (w_j/p_j)\Sigma_j (t - r_j)x_{jt} + \beta\Sigma_t P(\Sigma_j x_{jt})$
- Subject to
  - $\Sigma_t x_{jt} \geq p_j$ [each job is finished]
  - $x_{jt} \geq 0$
- Variable x_{jt} = number of instructions of job j done at time t
- Variable s_t = speed at time t
- P is power function, think $P(s) \approx s^3$
- w_j = importance, p_j =size, and r_j = arrival time

Convex Programming Formulation

- Min $\Sigma_j (w_j/p_j)\Sigma_j (t - r_j)x_{jt} + \beta\Sigma_t P(s_t)$
- Subject to
  - $\Sigma_t x_{jt} \geq p_j$ [each job is finished]
  - $\Sigma_j x_{jt} = s_t$ [speed = total work]
  - $x_{jt} \geq 0$

- Variable x_{jt} = number of instructions of job j done at time t
- Variable s_t = speed at time t
- P is power function, think $P(s) \approx s^3$
- w_j = importance, p_j =size, and r_j = arrival time

KKT Optimality Conditions: Generalizes Lagrangian Multipliers and Complementary Slackness Optimality Conditions for Linear Programs

Consider a strictly-feasible convex differentiable program

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

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

$$\begin{aligned} f_i(x) &\leq 0 && i = 1, \ldots, n \\ \lambda_i &\geq 0 && i = 1, \ldots, n \\ \lambda_i f_i(x) &= 0 && \text{Complementary slackness} \\ \nabla f_0(x) + \sum_{i=1}^{n} \lambda_i \nabla f_i(x) &= 0 && \text{Key equation: from method of Lagrange multipliers} \end{aligned}$$

Applying KKT

- Min $\Sigma_j (w_j/p_j)\Sigma_j (t - r_j)x_{jt} + \beta\Sigma_t P(s_t)$
- Subject to
  - $p_j - \Sigma_t x_{jt} \leq 0$ Dual variable λ_j
  - $- x_{jt} \leq 0$ Dual variable δ_{jt}

$$
\begin{aligned}
f_i(x) &\leq 0 \qquad i = 1, \ldots, n \\
\lambda_i &\geq 0 \qquad i = 1, \ldots, n \\
\lambda_i f_i(x) &= 0 \qquad \text{Complementary slackness} \\
\nabla f_0(x) + \sum_{i=1}^{n} \lambda_i \nabla f_i(x) &= 0 \qquad \text{Key equation}
\end{aligned}
$$

- Key equation: $(w_j/p_j)(t-r_j) + \beta P'(s_t) - \lambda_j - \delta_{jt} = 0$

Applying KKT

- Min $\Sigma_j (w_j/p_j)\Sigma_j (t - r_j)x_{jt} + \beta\Sigma_t P(s_t)$
- Subject to
  - $p_j - \Sigma_t x_{jt} \leq 0$ Dual variable λ_j
  - $- x_{jt} \leq 0$ Dual variable δ_{jt}

$$\begin{array}{rcll} f_i(x) & \leq & 0 & i = 1, \ldots, n \\ \lambda_i & \geq & 0 & i = 1, \ldots, n \\ \lambda_i f_i(x) & = & 0 & \text{Complementary slackness} \\ \nabla f_0(x) + \sum_{i=1}^{n} \lambda_i \nabla f_i(x) & = & 0 & \text{Key equation} \end{array}$$

- Key equation: $(w_j/p_j)(t-r_j) + \beta P'(s_t) - \lambda_j - \delta_{jt} = 0$
- If job j is run at time t then δ_{jt} =?

Applying KKT

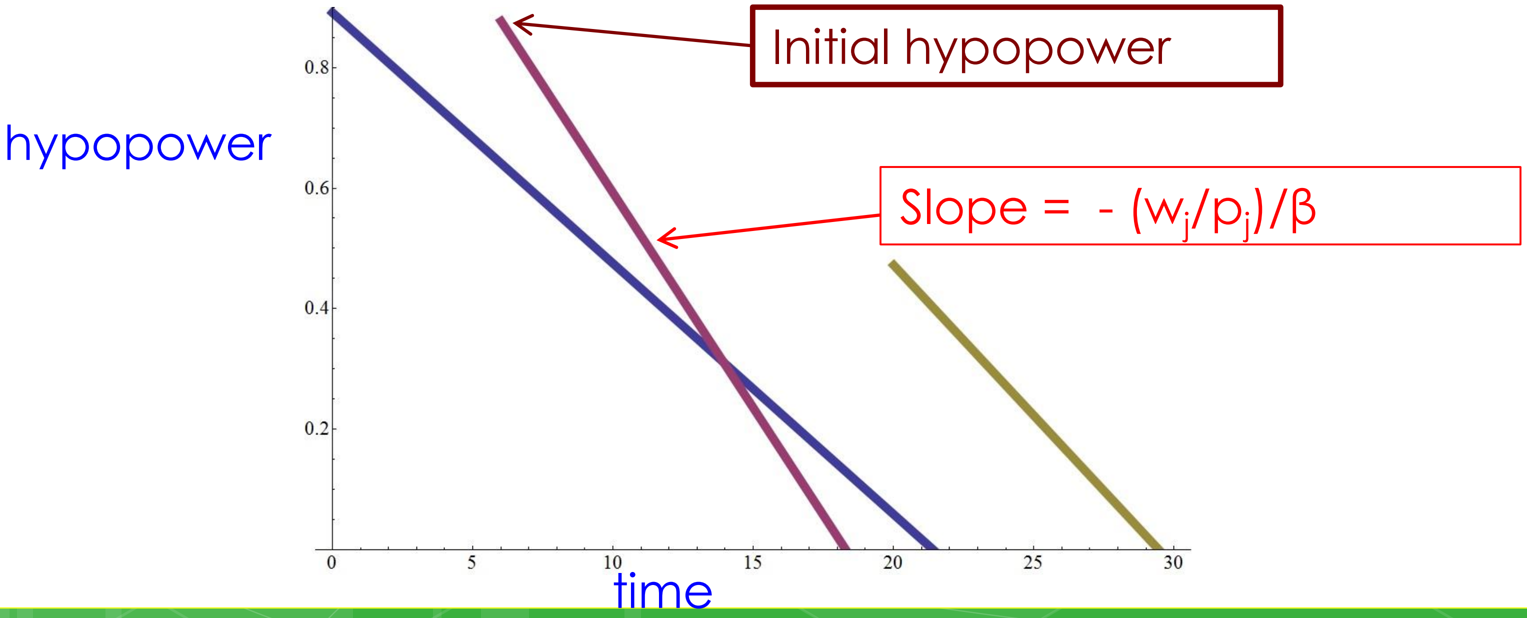
- Min $\Sigma_j (w_j/p_j)\Sigma_j (t - r_j)x_{jt} + \beta\Sigma_t P(s_t)$
- Subject to
  - $p_j - \Sigma_t x_{jt} \leq 0$ Dual variable λ_j
  - $- x_{jt} \leq 0$ Dual variable δ_{jt}

$$\begin{aligned} f_i(x) &\leq 0 \quad & i = 1, \ldots, n \\ \lambda_i &\geq 0 \quad & i = 1, \ldots, n \\ \lambda_i f_i(x) &= 0 \quad & \text{Complementary slackness} \\ \nabla f_0(x) + \sum_{i=1}^{n} \lambda_i \nabla f_i(x) &= 0 \quad & \text{Key equation} \end{aligned}$$

- Key equation: $(w_j/p_j)(t-r_j) + \beta P'(s_t) - \lambda_j - \delta_{jt} = 0$
- By complementary slackness, $\delta_{jt}=0$ if job j is run at t
- By some algebra, at all times t when j is run: $P'(s_t) = \lambda_j - (w_j/\beta p_j)(t-r_j)$

Characterization of Optimal Schedule From KKT Conditions (1)

- By some algebra, at all times t when j is run: $P'(s_t) = \lambda_j - (w_j/\beta p_j)(t - r_j)$



Applying KKT

- Min $\Sigma_j\ (w_j/p_j)\Sigma_j\ (t - r_j)x_{jt} + \beta\Sigma_t\ P(s_t)$
- Subject to
  - $p_j - \Sigma_t\ x_{jt} \leq 0$ Dual variable λ_j
  - $- x_{jt} \leq 0$ Dual variable δ_{jt}

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

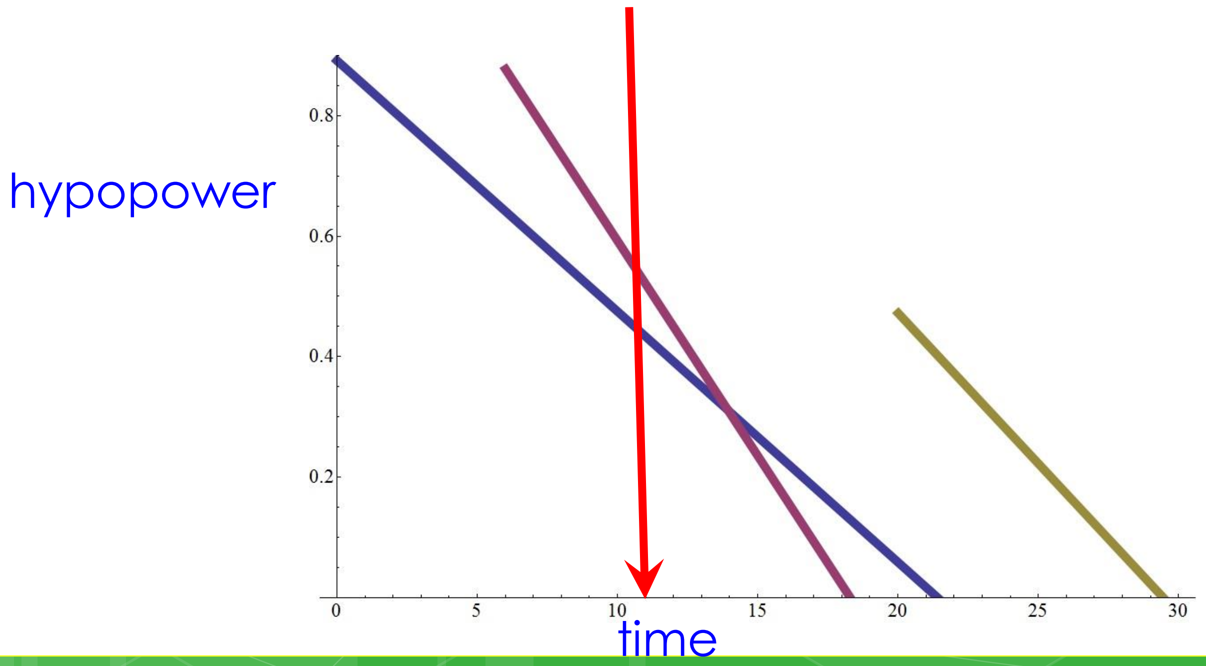
Complementary slackness

Key equation

- Key equation: $(w_j/p_j)(t-r_j) + \beta\ P'(s_t) - \lambda_j - \delta_{jt} = 0$
- At all times t when j is run: $P'(s_t) = \lambda_j - (w_j/\beta p_j)(t-r_j)$
- By nonnegativity of the δ dual variables, if j is not run at time t, then $P'(s_t) > \lambda_j - (w_j/\beta p_j)(t-r_j)$

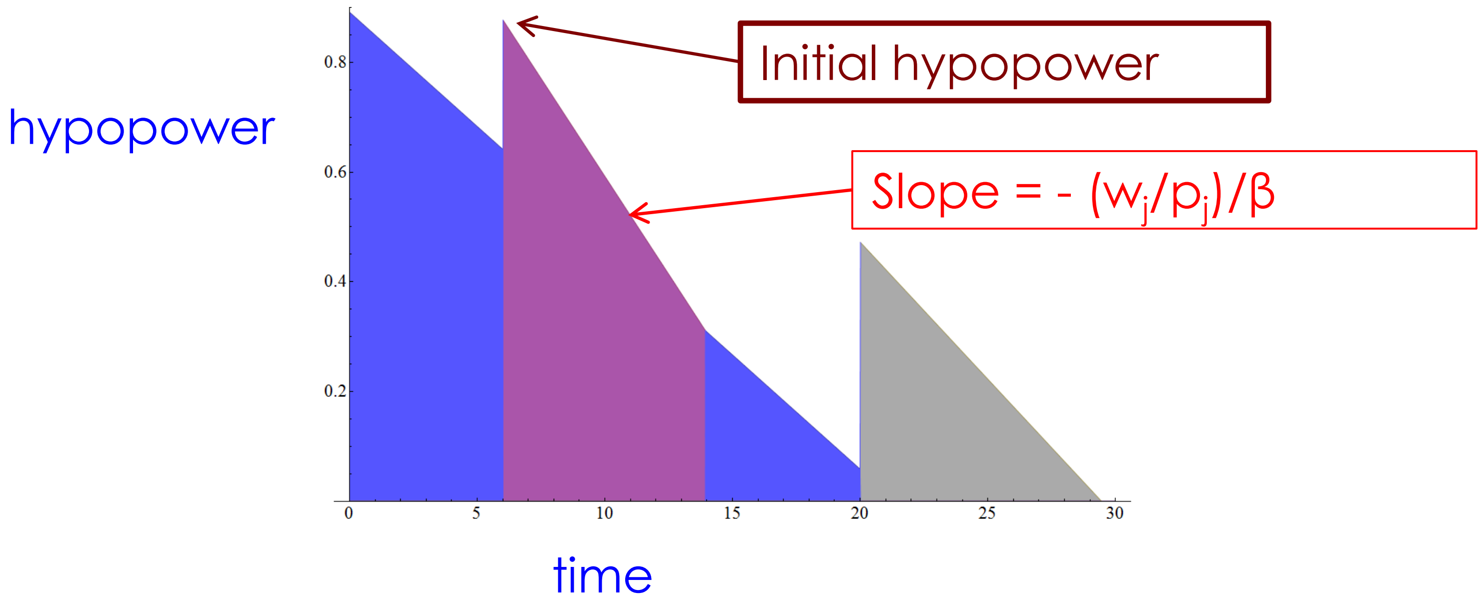
Characterization of Optimal Schedule From KKT Conditions (1)

- If j is run at time t, then it is run at the hypopower on its hypopower liner
- If a job j is not run at a time t, then hypopower of processor at time t lies above j's hypopower line.
- What job is run at this time in optimal schedule?



Characterization of Optimal Schedule From KKT Conditions (2)

- Schedule is upper envelope of hypopower functions and
- Each job j is processed the right amount (feasibility)



Polytime Algorithm for Discrete Speeds[ABCKNPS2014]: Continuous evolution as Job Sizes Increase

Open Question

- What is the complexity (in P or NP-equivalent or something else) of computing the optimal (unweighted integer) total waiting time plus energy schedule?
  - Shortest Remaining Processing Time is optimal for total waiting time for a fixed speed processor. So hardness must come from speed setting.
  - NP-hard to minimize waiting time subject to an energy constraint.
  - I don't know how to solve this in poly time even when processor has only 2 speeds.

Road Map

- Managing speed scalable processors
  - Introduction to power heterogeneity and speedy scaling
  - Philosophical discussion about modeling
  - Unfortunately long introduction formal model
  - Offline algorithm design and analysis using KKT conditions
  - Online algorithm design and analysis using potential functions
  - Online algorithm design and analysis using dual fitting of Lagrangian dual

Natural Online Algorithm

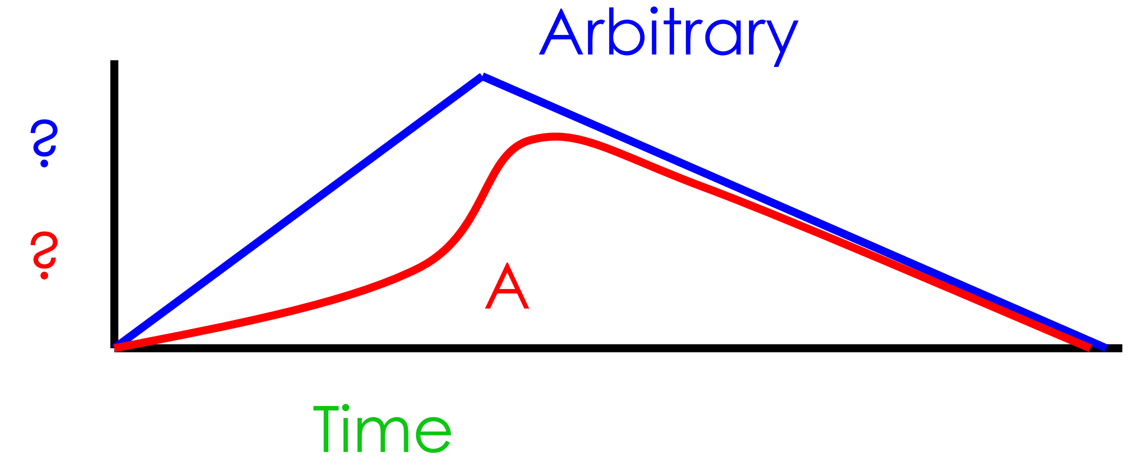
- Job selection policy to determine which job to run
  - Natural candidate policy is ?
- Speed selection policy to determine the speed to run at
  - Natural candidate policy is ?

Natural Online Algorithm A

- Job selection policy to determine which job to run
  - Natural candidate policy is HDF, running the job with highest weight/size ratio
- Speed selection policy to determine the speed to run at
  - Natural candidate policy is power = unfinished fractional weight
    - Recall, this is approximately optimal if no more jobs arrive

Local Competitiveness Proof

- Theorem: Natural Algorithm A is 2-competitive for fractional weighted waiting time plus energy
- Proof:

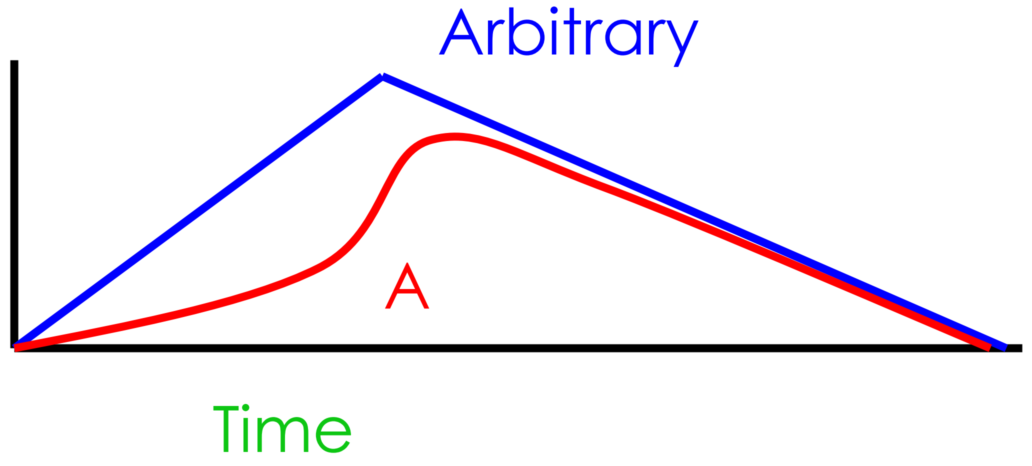


Fractional weighted waiting time + energy =
$\int_t$? dt

Local Competitiveness Proof

- Theorem: Natural Algorithm A is 2-competitive for fractional weighted waiting time plus energy
- Proof:

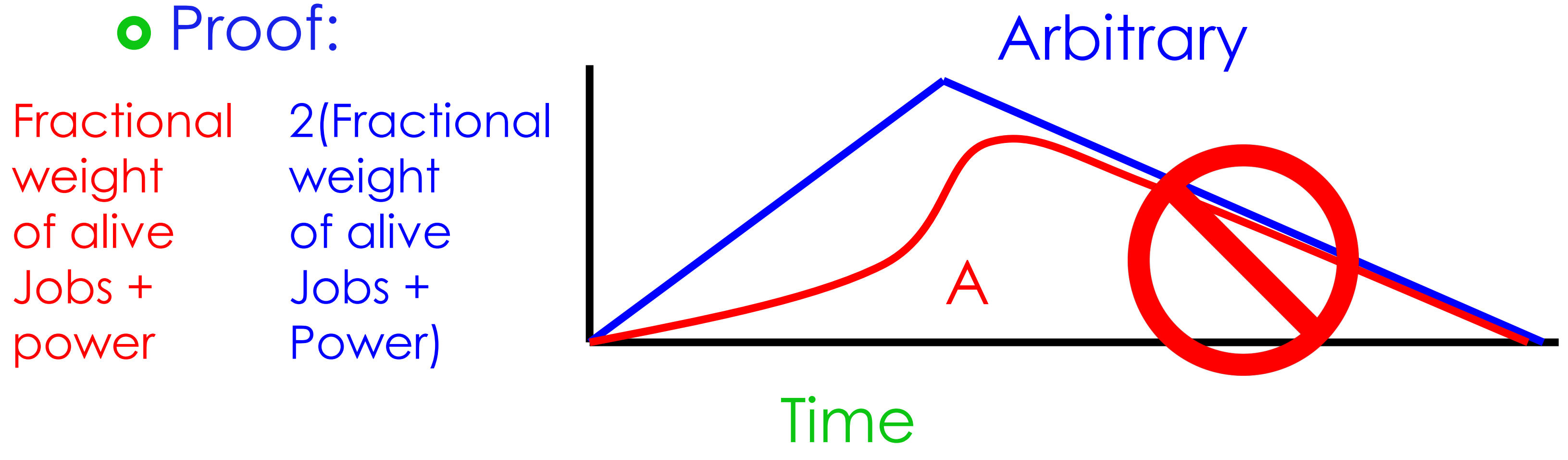
Fractional weight of alive Jobs + power	2(Fractional weight of alive Jobs + Power)



Fractional weighted waiting time + energy =
$\int_t$ (Fractional weight of alive jobs at time t + power)dt

Why Won't Local Competitiveness work?

- Theorem: Natural Algorithm A is 2-competitive for fractional weighted waiting time plus energy
- Proof:

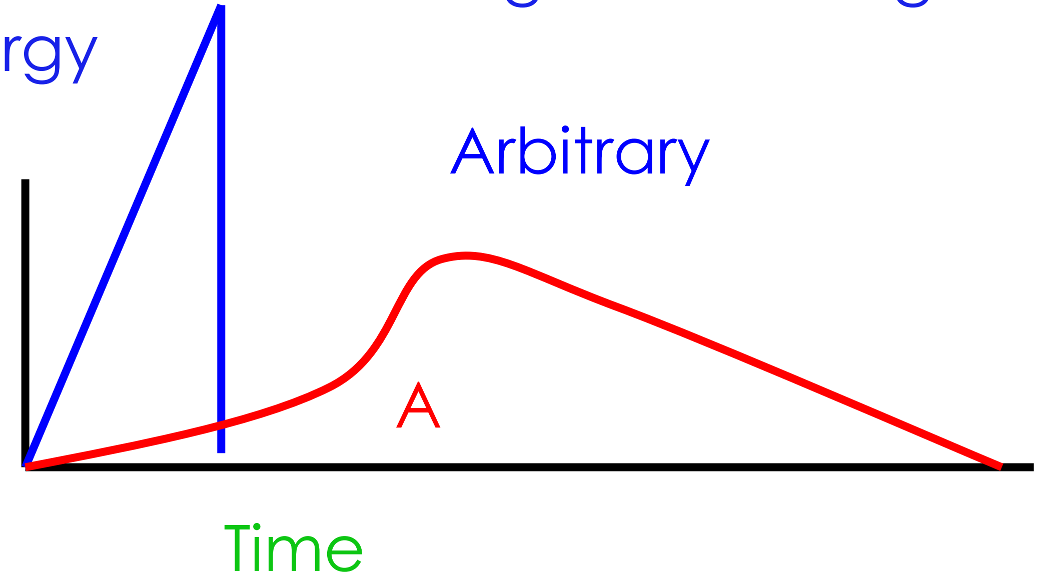

Fractional weight of alive Jobs + power
2(Fractional weight of alive Jobs + Power)
Arbitrary
A
Time

Why Won't Local Competitiveness work?

- Theorem: Natural Algorithm A is 2-competitive for fractional weighted waiting time plus energy
- Proof:

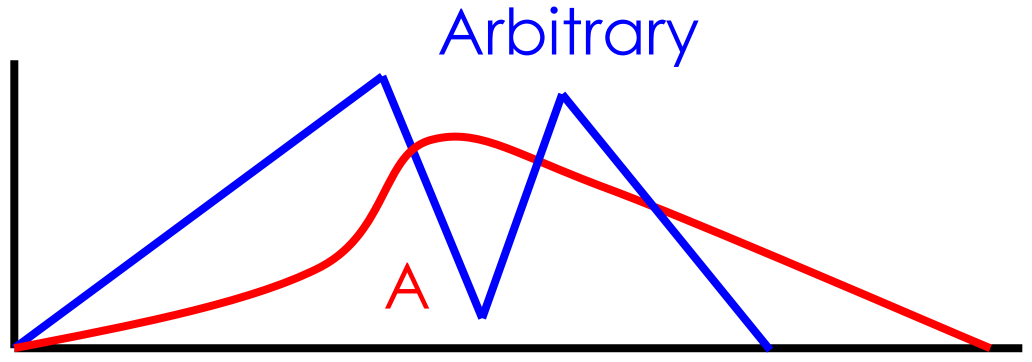
Fractional weight of alive Jobs + power

2(Fractional weight of alive Jobs + Power)



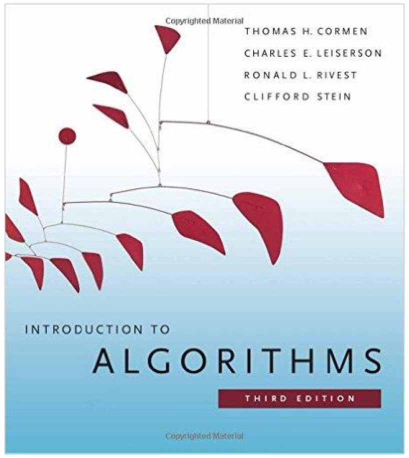
Why Won't Local Competitiveness work?

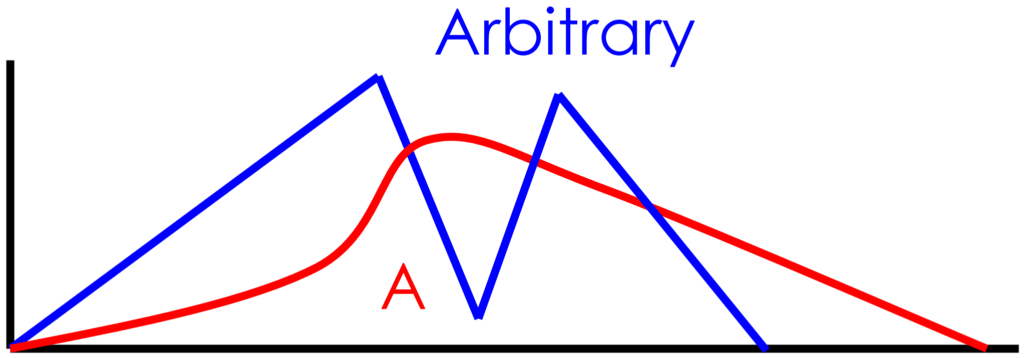
- Theorem: Natural Algorithm A is 2-competitive for fractional weighted waiting time plus energy
- Proof:



Why Won't Local Competitiveness work?

- Theorem: Natural Algorithm A is 2-competitive for fractional weighted waiting time plus energy
- Proof:

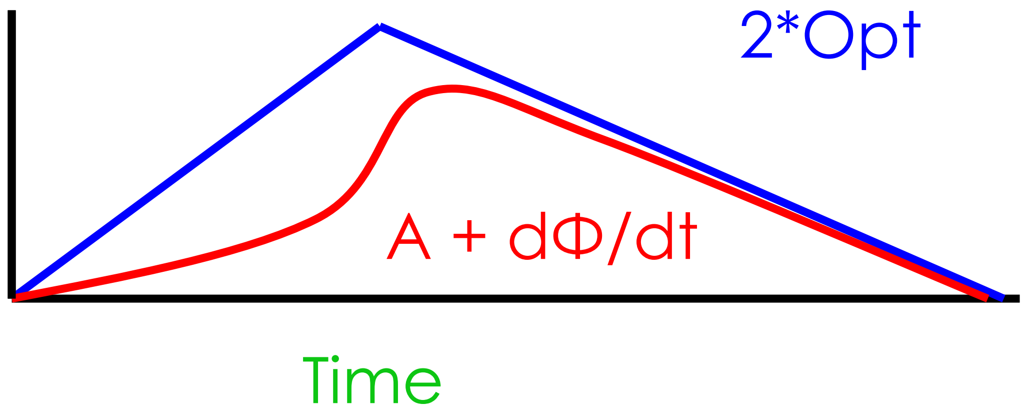




Do you recall seeing this issue come up in CLRS text?
If so, what chapter? And how did CLRS resolve the issue?

Amortized Local Competitiveness

- Theorem: A is 2-competitive for the objective of fractional weighted waiting time plus energy
- Proof: Amortized Local Competitiveness
- 1.Potential function Φ initially 0,
- 2. Φ finally positive, and
- 3. For all t, $A(t) + d\Phi/dt \le 2*Opt(t)$



The Subtle Magic of Potential Functions

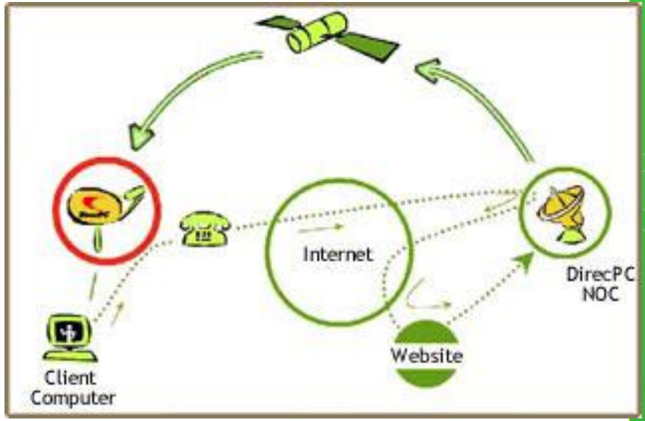
- Most natural to define Φ(t) in terms of Φ(t-1) and ΔΦ

- Potential function method: Define ΔΦ in terms of Φ(t-1) and Φ(t)

Potential Functions in Speed Scaling

- There is a standard potential function [BPS07]
  - Most potential functions used in the speed scaling literature are variations thereof [IMP2011]

- The use of this standard potential function has now become a standard technique for analyzing scheduling problems that are unrelated to energy
  - e.g. Broadcast scheduling



Now Standard Potential Function [BPS07]

- Semi-formal definition: Φ = Future online cost where the size of each job is how far the online algorithm is behind on that job
- For our running example, future cost = $\Sigma x_j W_j^{(1-1/\alpha)}$ where
  - x_j is size of jth most dense job
  - W_j is weight of jobs less dense than j
- Then standard potential function $\Phi = \Sigma y_j V_j^{(1-1/\alpha)}$
  - Where y_j is how far the online algorithm is behind on job j
  - V_j is the amount of weight in work that is less dense than the density of job j and that online is behind on

Unit Density Jobs with a= 2.

- Recall that we calculated cost for A with n unit work unit weight jobs to be $n^{2-1/a}$, which equals $n^{3/2}$ when a=2.

- Standard potential function
  $\Phi= (8/3)\ (\max(0, n_a - n_o))^{3/2}$
  - n_a and n_o are the fractional number of unfinished jobs for algorithm and optimal
  - From here we will assume the hard case that $n_a > n_o$

Unit Density Jobs with a= 2.

- Potential function $\Phi = (8/3)\ (n_a - n_o)^{3/2}$
  - n_a and n_o are the fractional number of unfinished jobs for algorithm and optimal
- Key Equation:

$$A(t) + d\Phi/dt \leq 2^*Opt(t)$$

  - s_a is speed of algorithm and s_o is speed of optima.
  - What is cost A(t) is incurring at time t?
  - What is cost Opt incurring at time t?

Unit Density Jobs with a= 2.

- Potential function $\Phi = (8/3)\ (n_a - n_o)^{3/2}$
  - n_a and n_o are the fractional unfinished jobs

- Key Equation: $s_a^2 + n_a + d\Phi/dt \leq 2*(s_o^2 + n_o)$
  - s_a is speed of algorithm and s_o is speed of optima.

- Simple things to check:
  - Φ initially 0
  - Φ finally nonnegative
  - Key equation holds when job arrives
  - Key equation holds when A finishes a job
  - Key equation holds when Opt finishes a job
- Nontrivial thing to check is it holds due A and Opt running jobs

Unit Density Jobs with a= 2

- Standard potential function $\Phi = (8/3)\ (n_a - n_o)^{3/2}$
  - n_a and n_o are the fractional number of unfinished jobs for algorithm and optimal. s_a is speed of algorithm and s_o is speed of optima.

- Key Equation: $s_a^2 + n_a + d\Phi/dt \leq 2^*(s_o^2 + n_o)$

- How are s_a^2 and n_a related?
- What is $d\Phi/dt$?

Unit Density Jobs with a= 2

- Standard potential function $\Phi = (8/3)\ (n_a - n_o)^{3/2}$
  - n_a and n_o are the fractional number of unfinished jobs for algorithm and optimal. s_a is speed of algorithm and s_o is speed of optima.

- Key Equation: $s_a^2 + n_a + d\Phi/dt \le 2^*(s_o^2 + n_o)$

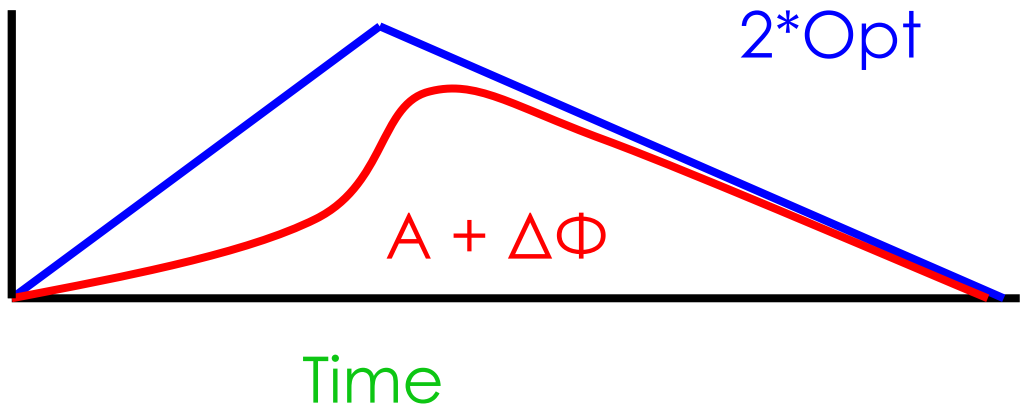
- $s_a^2 = n_a$ by the definition of algorithm A
- $d\Phi/dt = 4\ (n_a - n_o)^{1/2}\ (-s_a + s_o)$

Unit Density Jobs with $\alpha = 2$

- Need to verify $2n_a + 4(n_a - n_o)^{1/2} (s_o - n_a^{1/2}) \le 2^*(s_o^2 + n_o)$
- By simple algebra
  - $2n_a - 4(n_a - n_o) + 4(n_a - n_o)^{1/2} s_o \le 2^*(s_o^2 + n_o)$
  - $-2n_a + 4(n_a - n_o)^{1/2} s_o \le 2^*s_o^2 - 2 n_o$
  - $(n_a - n_o)^{1/2} s_o \le ((n_a - n_o) + s_o^2)/2$
    - Since $ab \le a^2/2 + b^2/2$
- Plugging in: $-2n_a + 2((n_a - n_o) + 2s_o^2 \le 2^*s_o^2 - 2 n_o$

Amortized Local Competitiveness

- Theorem: Natural algorithm is 2-competitive for the objective of fractional weighted waiting time plus energy
- Proof: Amortized Local Competitiveness
- 1. Φ initially 0,
- 2. Φ finally positive, and
- 3. For all t, A(t) + dΦ/dt ≤ 2*Opt(t)



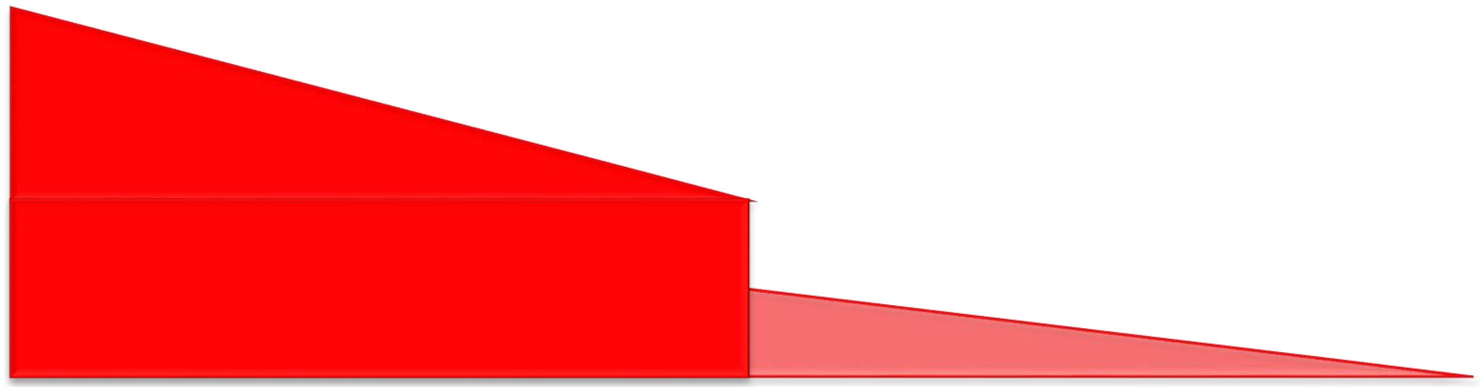
Amortized Local Competitiveness

- Theorem: Natural algorithm is 2-competitive for the objective of fractional weighted waiting time plus energy
- Theorem: There is a 16-competitive algorithm for the objective of (integer) weighted waiting time plus energy when $P(s)=s^3$
- Proof: Hint, think of one job. How can you get waiting time ≤ fractional waiting time while increasing flow by at most a factor of 8?

Amortized Local Competitiveness

- Theorem: Natural algorithm is 2-competitive for the objective of fractional weighted waiting time plus energy

- Theorem: There is a 16-competitive algorithm for the objective of (integer) weighted waiting time plus energy when $P(s)=s^3$
- Proof: Run same job as fractional algorithm at twice the speed

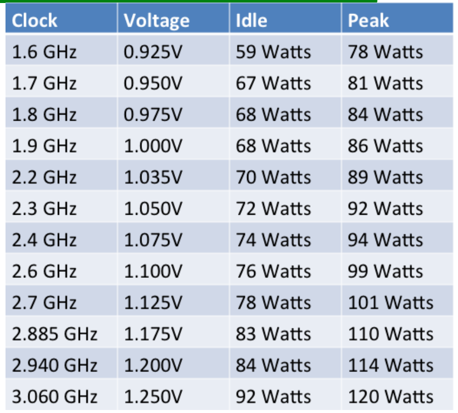


Amortized Local Competitiveness

- Theorem: A is 2-competitive for the objective of fractional weighted waiting time plus energy

- Theorem[BCP09]: Natural algorithm is a 2-competitive algorithm for the objective of (integer) unweighted waiting time plus energy for any power function
- Proof: Non-standard potential function

Which is the Best Theoretical Algorithmic Model for Allowable Speeds?



Clock	Voltage	Idle	Peak
1.6 GHz	0.925V	59 Watts	78 Watts
1.7 GHz	0.950V	67 Watts	81 Watts
1.8 GHz	0.975V	68 Watts	84 Watts
1.9 GHz	1.000V	68 Watts	86 Watts
2.2 GHz	1.035V	70 Watts	89 Watts
2.3 GHz	1.050V	72 Watts	92 Watts
2.4 GHz	1.075V	74 Watts	94 Watts
2.6 GHz	1.100V	76 Watts	99 Watts
2.7 GHz	1.125V	78 Watts	101 Watts
2.885 GHz	1.175V	83 Watts	110 Watts
2.940 GHz	1.200V	84 Watts	114 Watts
3.060 GHz	1.250V	92 Watts	120 Watts

A. 1.6, 1.7, 1.8, 1.9, 2.2, 2.3, 2.4, …, 3.06
- Too specific

B. Arbitrary discrete speeds $s_1, \ldots, s_k$
- Optimal online algorithm: Do what algorithm would do for continuous speeds, and then round to nearest discrete speed

C. Reals in a range $[0, s_{max}]$ for an arbitrary constant s_{max}
- Optimal online algorithm: Run at the minimum of what the continuous speed algorithm would run at and s_{max}

D. Any nonnegative reals
- In hindsight, best theoretical/algorithmic model

Road Map

- Managing speed scalable processors
  - Introduction to power heterogeneity and speedy scaling
  - Philosophical discussion about model
  - Unfortunately long introduction formal model
  - Offline algorithm design and analysis using KKT conditions
  - Online algorithm design and analysis using potential functions
  - Online algorithm design and analysis using dual fitting of Lagrangian dual

Online Algorithm Design and Analysis Technique

- Design
  - Express the problem as a mathematical program
  - View the online problem as constraints of this primal program arriving online
  - Consider the online greedy algorithm that raises the primal variables to satisfy the newly arriving constraint so as to minimize the increase in the primal objective
- Analysis (Dual fitting)

Recall Convex Programming Formulation

- Min $\Sigma_j \Sigma_t c_{j,t} x_{jt} + \beta\Sigma_t (\Sigma_j \; x_{j,t})^a$
- Subject to
  - $\Sigma_t \; x_{jt} \geq 1$ [each job is finished]
- Variable x_{jt} = number of instructions of job j done at time t
- $c_{j,t}$ = cost for finishing work at time t
  - Note that we abstract out the specific scheduling objective
- Job arrival equivalent to new constraint

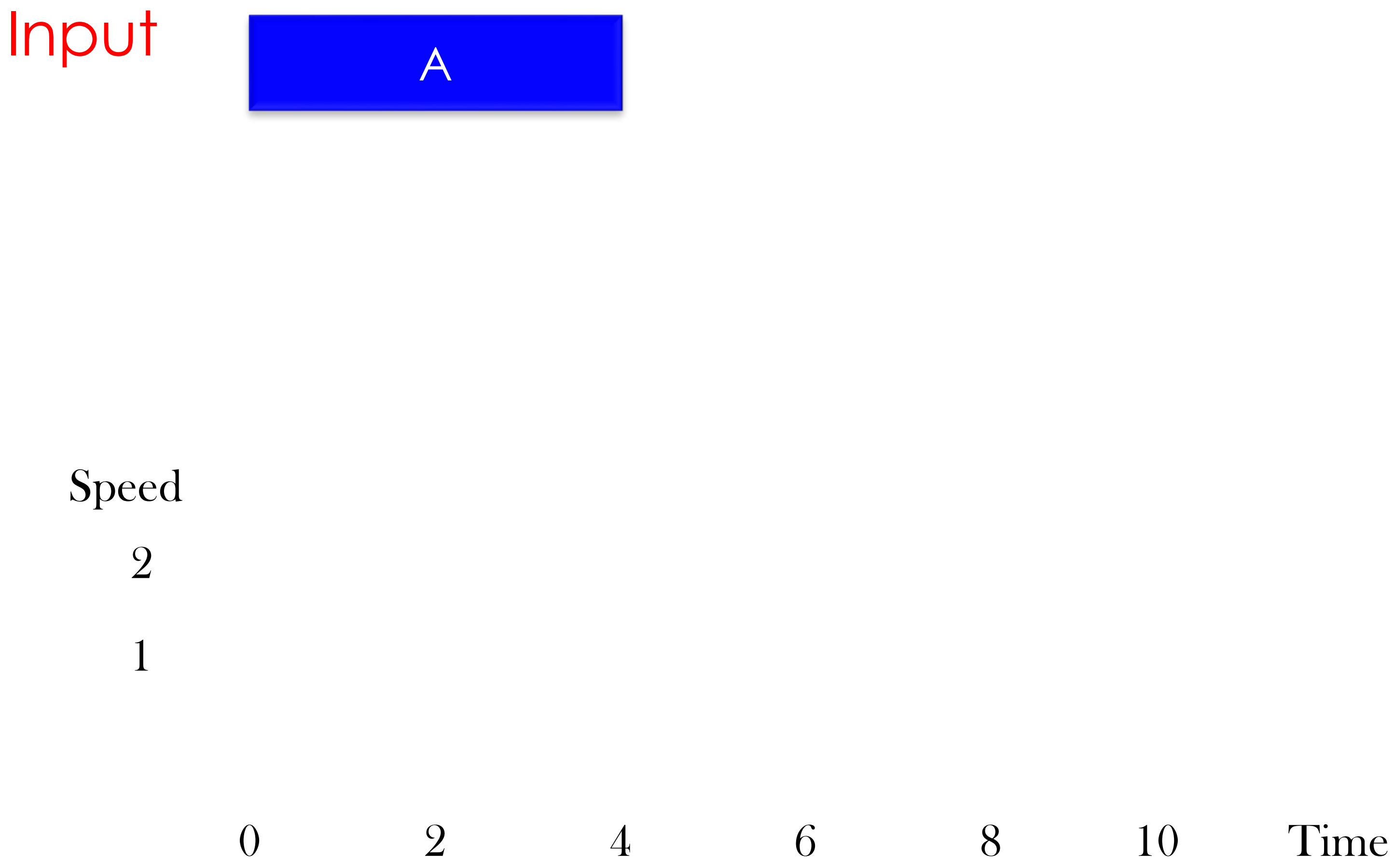
Greedy Algorithm

- Min $\Sigma_t (\Sigma_j \; x_{j,t})^a + \Sigma_t \Sigma_j \; c_{j,t} x_{j,t}$
- Subject to $\Sigma_t x_{j,t} \geq 1$

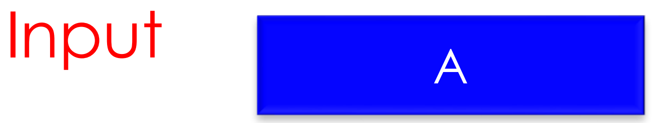
- Online Greedy Algorithm: When a new job j arrives raise the primal variables so that the increase in the kludged primal objective $\delta\Sigma_t (\Sigma_j x_{j,t})^a + \Sigma_t \Sigma_j \; c_{j,t} x_{j,t}$ is minimal
  - Kludge: $\delta < 1$ is a discount factor for energy costs

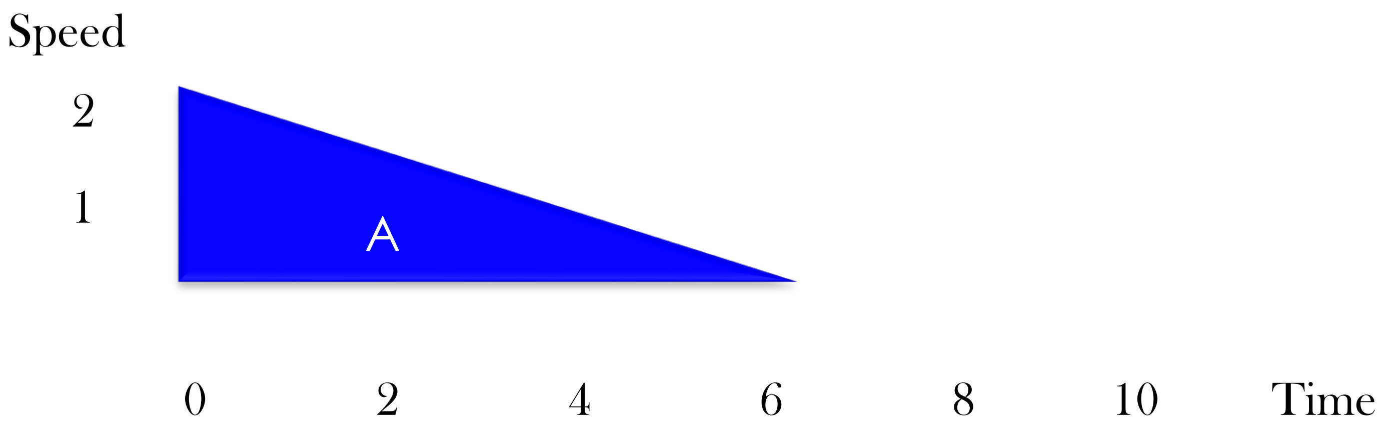
  - Let y_{jt} be final setting of variables for online

Greedy Algorithm



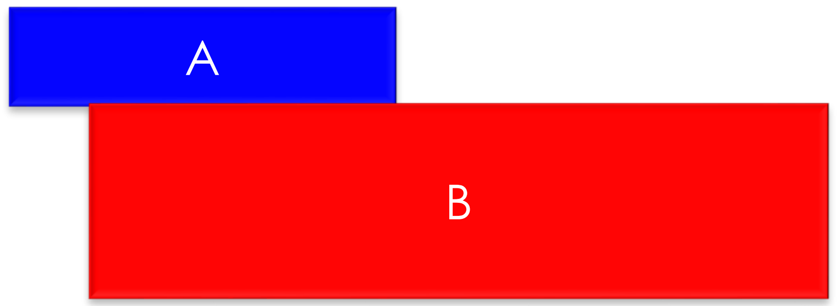
Online Scheduling

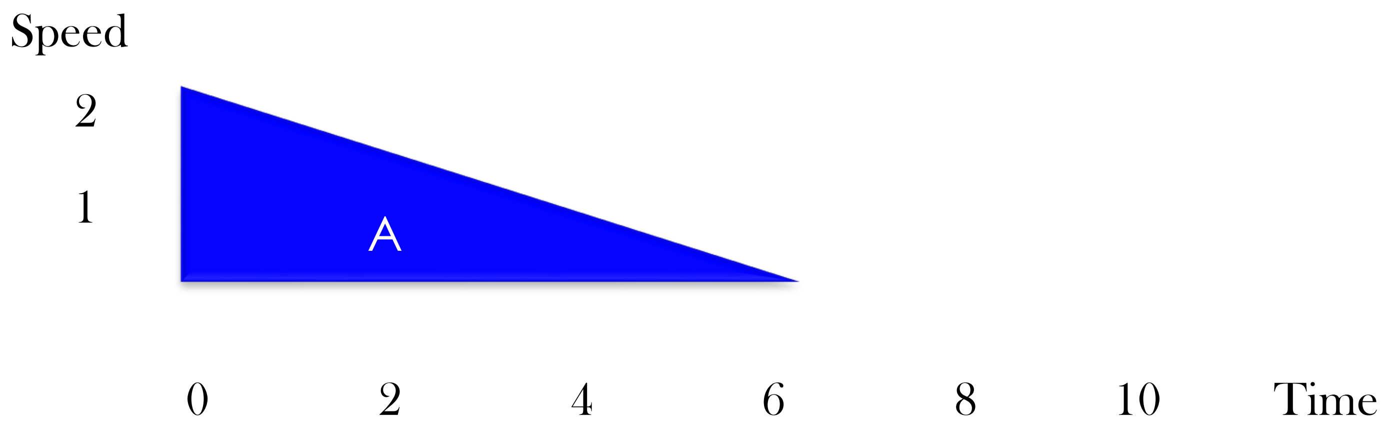




Online Scheduling

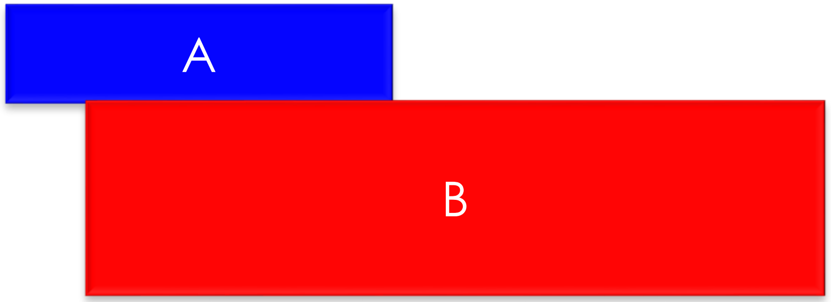
Input

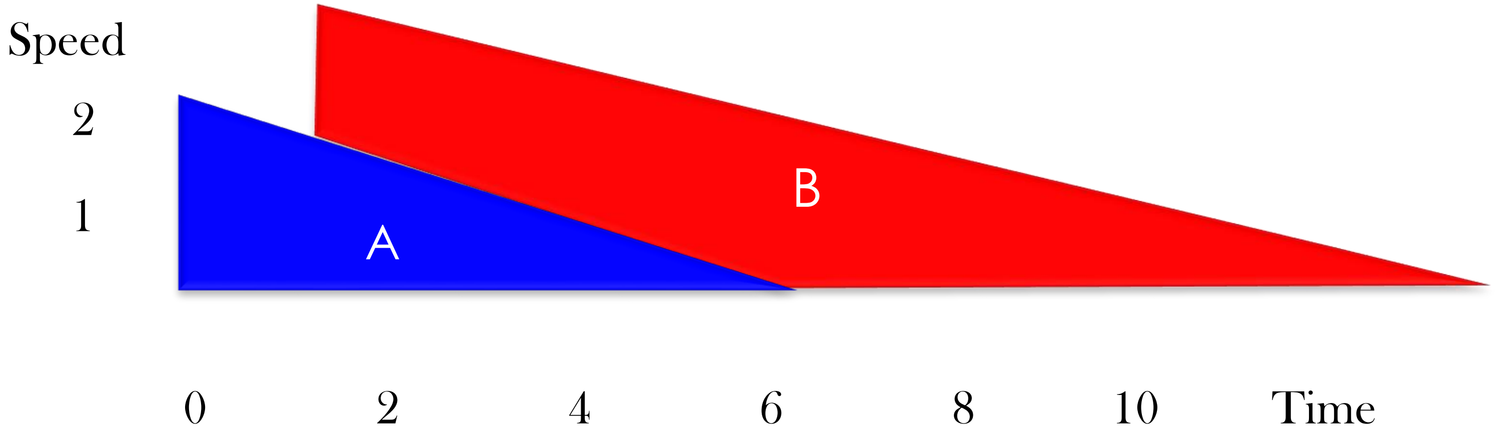




Online Scheduling

Input

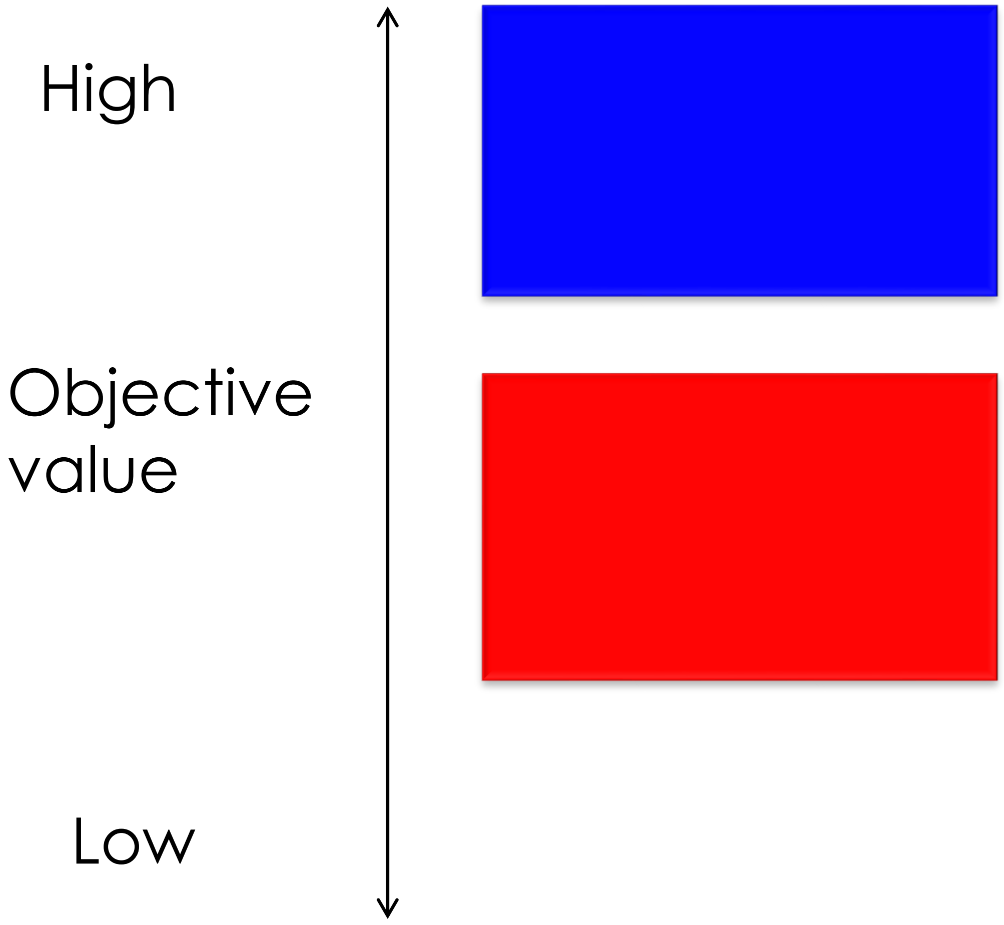




Online Algorithm Design and Analysis Technique

- Design
  - Express the problem as a mathematical program
  - View the online problem as constraints of this primal program arriving online
  - Consider the online greedy algorithm that raises the primal variables to satisfy the newly arriving constraint so as to minimize the increase in the primal objective
- Analysis (Dual fitting)

Dual Fitting Ideal



Feasible solutions to (primal) mathematical programming formulation of minimization problem

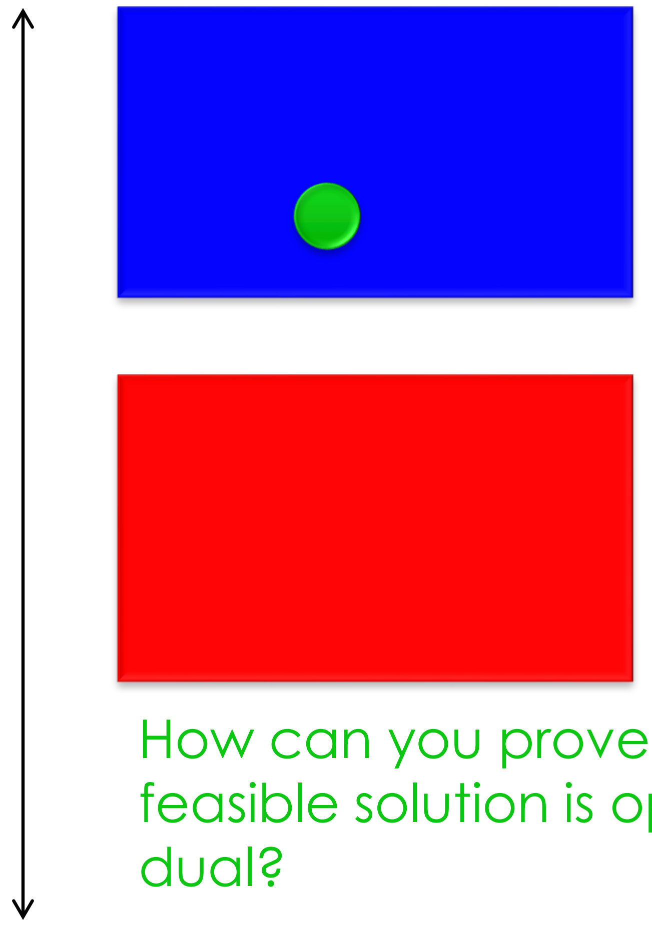
Feasible solutions to dual maximization problem

Dual Fitting Ideal

High

Objective value

Low

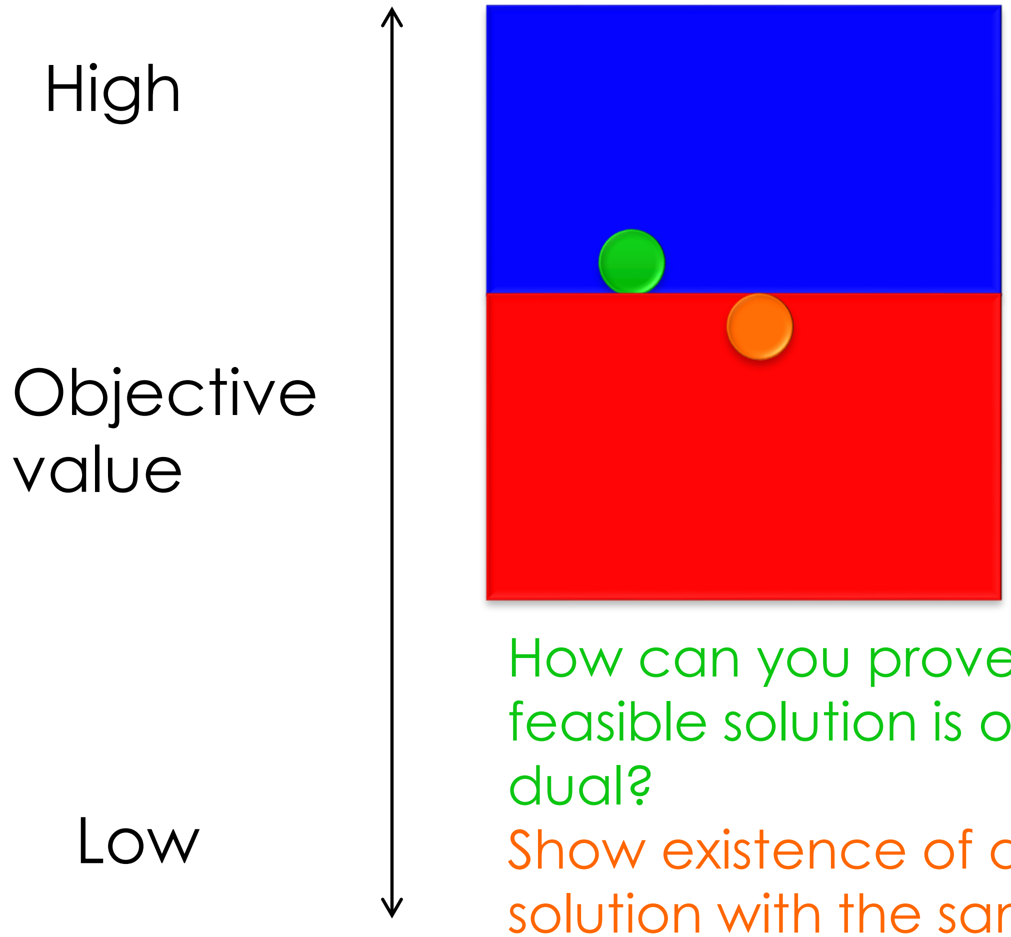


Feasible solutions to (primal) mathematical programming formulation of minimization problem

Feasible solutions to dual maximization problem

How can you prove a particular primal feasible solution is optimal using the dual?

Dual Fitting Ideal



Feasible solutions to (primal) mathematical programming formulation of minimization problem

Feasible solutions to dual maximization problem

How can you prove a particular primal feasible solution is optimal using the dual?

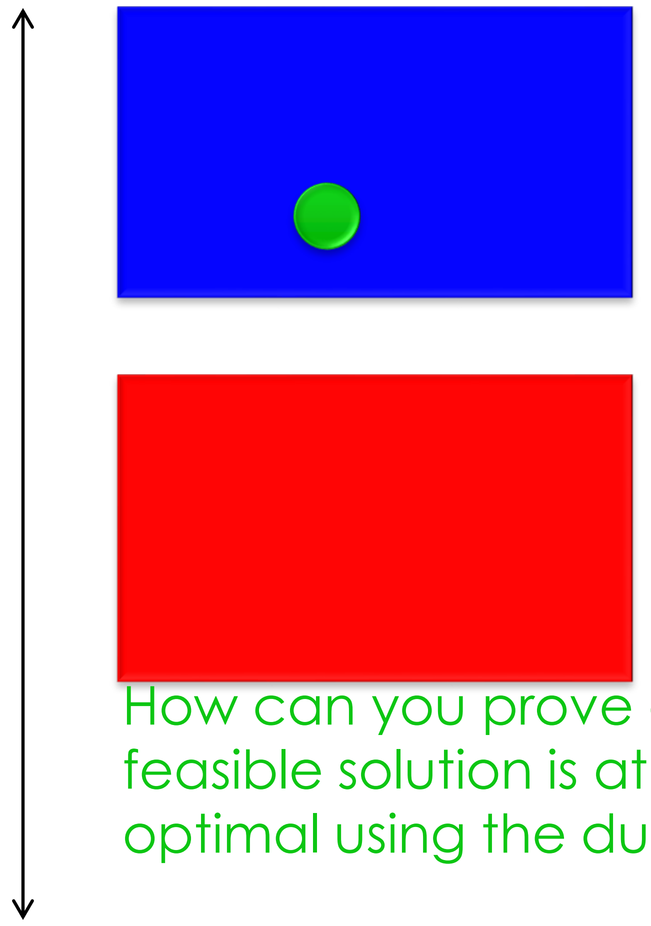
Show existence of dual feasible solution with the same objective value

Dual Fitting Ideal

High

Objective value

Low

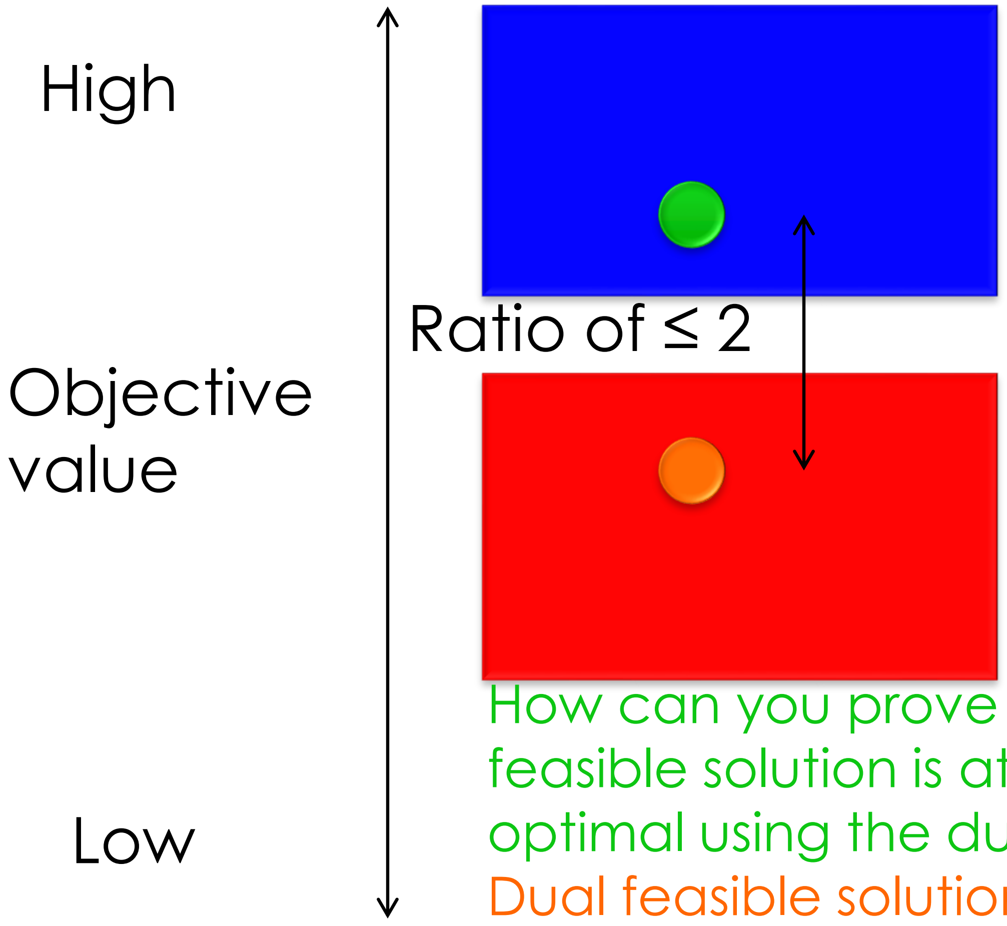


Feasible solutions to (primal) mathematical programming formulation of minimization problem

Feasible solutions to dual maximization problem

How can you prove a particular primal feasible solution is at most twice optimal using the dual?

Dual Fitting Ideal



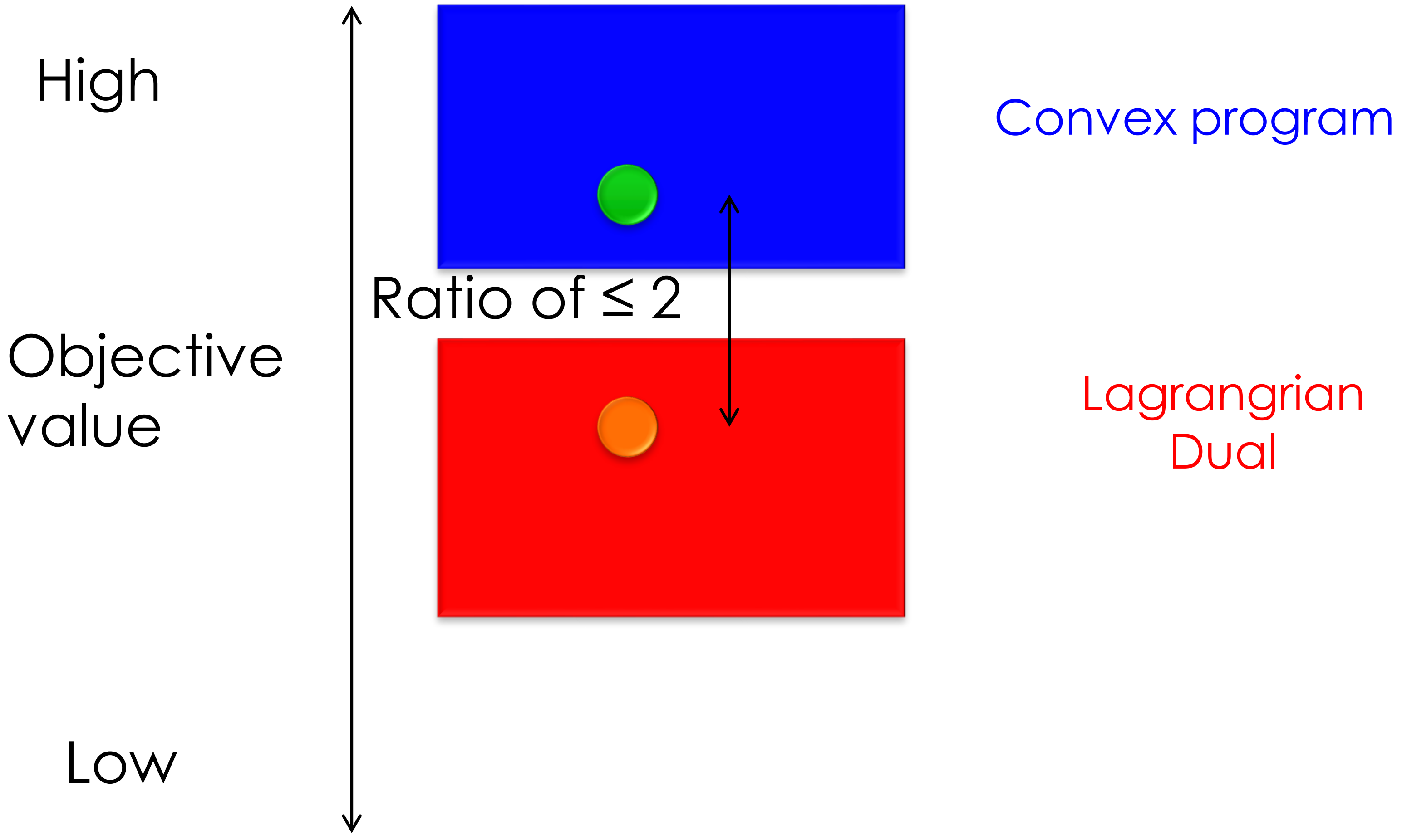
Feasible solutions to (primal) mathematical programming formulation of minimization problem

Feasible solutions to dual maximization problem

How can you prove a particular primal feasible solution is at most twice optimal using the dual?

Dual feasible solution with objective value within a factor of 2

For Our Speed Scaling Problem



Lagrangian Relaxation

- Primal: $\mathrm{Min}_x\ f(x)$ subject to $h(x) \leq 0$
- Lagrangian dual: $g(\lambda) = \min_x [f(x) + \lambda h(x)]$
- If $\lambda \geq 0$ then $g(\lambda)$ is lower bound to primal

Lagrangian Dual $g(\lambda)$

- Primal
  - $\text{Min}_x \sum_t (\sum_j x_{j,t})^a + \sum_t \sum_j c_{j,t} x_{j,t}$
  - Subject to $\sum_t x_{j,t} \geq 1$

- Lagrangian dual:
  - $g(\lambda) = \text{Min}_x (\sum_t (\sum_j x_{j,t})^a + \sum_t \sum_j c_{j,t} x_{j,t} + \sum_j \lambda_j (1 - \sum_t x_{j,t}))$

Dual Fitting Strategy

- Theorem: Online algorithm A is a^{a} competitive
- Proof Strategy:
- Opt ≥ g(λ)
- ≥ something involving only dual variablesλ
  - Requires determining optimal solution z of g(λ) in terms of λ
- ≥ something involving online solution y but not λ
  - Requires dual variables λbe defined in terms of y
- ≥ A/a^{a}

Dual Fitting Strategy

- Theorem: Online algorithm A is a^{a} competitive
  - Proof Strategy:
  - Opt ≥ g(λ)
  - ≥ something involving only dual variablesλ
    - Requires determining optimal solution z of g(λ) in terms of λ
  - ≥ something involving online solution y but not λ
    - Requires dual variables λbe defined in terms of y
  - ≥ A/a^{a}

- Harder for Lagrangian dual of convex program than standard dual of linear program because you have 3 types of variables floating around:
  - Setting of primal variables by online algorithm y
  - Dual variablesλ
  - Setting of primal variables in the dual z

To Do List

- Online solution y defined (done)
- Define dual variables λ in terms of y
- Determine optimal solution z to g(λ) in terms of λ

- Calculate g(λ) in terms of y's

Dual Variable Setting

- λ_j = the rate at which "kludged primal objective" $\delta\Sigma_t\ (\Sigma_j\ x_{j,t})^\alpha + \Sigma_t\ \Sigma_j\ c_{j,t}\, x_{j,t}$ is increasing for the last bit of work for job j

To Do List

- Online solution y defined
- Define dual variables λ in terms of y
- Determine optimal solution z to g(λ) in terms of λ
  - Key insight: g(λ) can be solved by simple greedy algorithm
- Calculate g(λ) in terms of y's

Interpretation of Lagrangian Dual $g(\lambda)$

- $g(\lambda) = \text{Min}_x \left(\Sigma_t \left(\Sigma_j \; x_{j,t}\right)^{\alpha} + \Sigma_t \Sigma_j \; c_{j,t} x_{j,t} + \Sigma_j \lambda_j \left(1 - \Sigma_t x_{j,t}\right)\right)$

- No constraint on how much work is scheduled for each job

- Objective:
  - Primal costs as before
  - There is a cost λ_j for each job j
  - One gains a payment λ_j of for each unit of job j that is scheduled

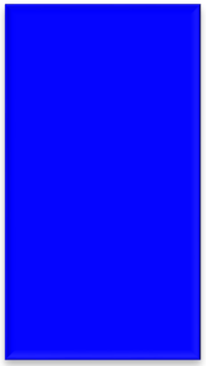
Interpretation of Lagrangian Dual g(λ)

- Objective:
  - Primal costs as before
  - There is a cost λ_j for each job j
  - One gains a payment λ_j of for each unit of job j that is scheduled

- Why doesn't optimal dual schedule way more stuff than online algorithm?
- Why doesn't optimal dual schedule way less stuff than online algorithm?

Consider 1 Job and 1 Time where c_{jt}=0

- Online algorithm schedules L units of work from job j at time t, i.e. $y_{jt} = L$
- Thus λ ≈ rate of increase of energy cost at the end $\approx \alpha L^{\alpha-1}$
- Thus the optimal solution $z_{jt} \approx L$ since this is where the energy costs start to exceed the payment rate λ



Interpretation of Lagrangian Dual g(λ)

- Objective:
  - Primal costs as before
  - There is a cost λ_j for each job j
  - One gains a payment λ_j of for each unit of job j that is scheduled

- Optimal greedy algorithm for this dual problem? Consider a particular time t.

Interpretation of Lagrangian Dual $g(\lambda)$

- Objective:
  - Primal costs as before
  - There is a cost λ_j for each job j
  - One gains a payment λ_j of for each unit of job j that is scheduled

- Greedy algorithm for dual problem: For each time, schedule work from the job that is most profitable for that time until costs start to exceed payments

To Do List

- Online solution y defined (done)
- Define dual variables λ in terms of y
- Determine optimal solution z to g(λ) in terms of λ

- Calculate g(λ) in terms of y’s (exercise)

Review

- Power heterogeneity, speed scaling
- Relationship of energy, power, speed
- KKT conditions
- Amortized local competitiveness, and the standard potential function
- Lagrangian relaxation
- Dual fitting

Covered Papers

- Daniel Cole, Dimitrios Letsios, Michael Nugent, Kirk Pruhs: Optimal energy trade-off schedules. IGCC 2012: 1-10
- Antonios Antoniadis, Neal Barcelo, Mario E. Consuegra, Peter Kling, Michael Nugent, Kirk Pruhs, Michele Scquizzato: Efficient Computation of Optimal Energy and Fractional Weighted Flow Trade-off Schedules. STACS 2014: 63-74
- Nikhil Bansal, Kirk Pruhs, Clifford Stein: Speed Scaling for Weighted Flow Time. SIAM J. Comput. 39(4): 1294-1308 (2009)
- Nikhil Bansal, Ho-Leung Chan, Kirk Pruhs: Speed Scaling with an Arbitrary Power Function. ACM Transactions on Algorithms 9(2): 18 (2013)
- Anupam Gupta, Ravishankar Krishnaswamy, Kirk Pruhs: Online Primal-Dual for Non-linear Optimization with Applications to Speed Scaling. Workshop on Approximation and Online Algorithms, 2012: 173-186

Thanks to all my collaborators!
Thanks for listening!