Dynamic Speed Scaling Systems: Theory and Practice

Carey Williamson
Department of Computer Science
University of Calgary

UNIVERSITY OF CALGARY

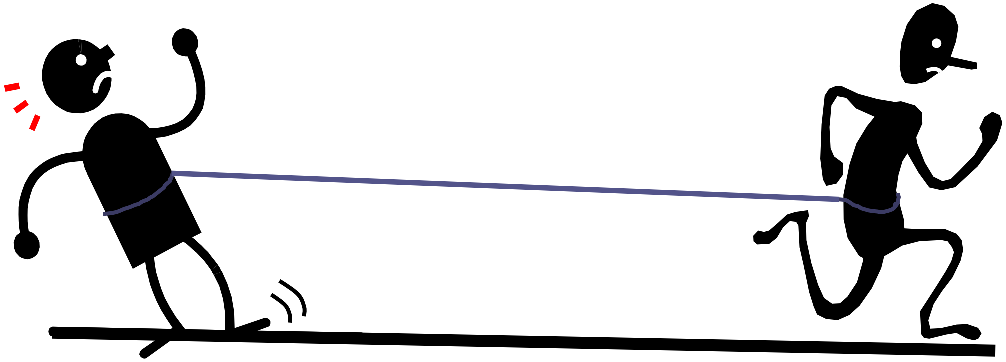
Motivation and Context

- The ICT ecosystem is responsible for 10% of the world's energy consumption [Mills 2013]
- Data centers account for roughly 2% of global energy consumption; growing at a rate of approximately 6% per annum
- The most energy-intensive component of any computer is its processor [Skrenes 2016]
  - 90% of energy usage when active (72W/80W)
  - 48% of energy usage when idle (3.1W/6.4W)
- Need for more energy-efficient computing

Speed Scaling: Inherent Tradeoffs

Dynamic Speed Scaling: adapt service rate to the current state of the system to balance energy consumption and performance.

Run slower: less ***energy***



Run faster: less ***delay***

- Minimize power consumption ***P***
  - Minimize energy cost $\boldsymbol{\varepsilon}$
  - Minimize heat, wear, etc.

- Minimize response time ***T***
  - Minimize delay
- Maximize job throughput

Main Messages (preview)

- There is a broad and diverse set of literature on speed scaling systems over the past 20+ years
- There is a dichotomy between theoretical work and systems work on speed scaling
- Modern processors provide surprisingly rich functionality for speed scaling that is not yet well exploited by systems software

Talk Outline

- Introduction and Motivation
- Background and Literature Review
- Review of Key Results and Insights
- Recent Results and Contributions
    - Decoupled Speed Scaling
    - Turbocharged Speed Scaling
    - Experimental Measurements
- Conclusions and Future Directions

Background: Theory and Systems

Theoretical Research	Systems Research
Goal: optimality	Goal: practicality
Domains: CPU, parallel systems	Domains: CPU, disk, network
Methods: proofs, complexity, competitive analysis, queueing theory, Markov chains, worst case, asymptotics, simulation	Methods: DVFS, power meter, measurement, benchmarking, simulation, power gating, over-clocking, simulation
Metrics: E[T], E[ε], combo, slowdown, competitive ratio	Metrics: response time, energy, heat, utilization
Power: $P = s^{\alpha} \quad (1 \leq \alpha \leq 3)$	Power: $P = a\, C_{eff}\, V^2 f$
Schedulers: PS, SRPT, FSP, YDS	Schedulers: FCFS, RR, FB
Speed scalers: job-count-based	Speed scalers: threshold-based
Venues: SIGMETRICS, PEVA, Performance, INFOCOM, OR	Venues: SIGMETRICS, SOSP, OSDI, ISCA, MASCOTS, TOCS

Literature #1: The Classics

- [Kelly 1979] Reversibility and Stochastic Networks, Wiley
- [Kleinrock 1975] Queueing Systems, Volume 1: Theory, Wiley
- [Schrage 1968] “A Proof of the Optimality of the SRPT Discipline”, Operations Research
- [Weiser et al. 1994] “Scheduling for Reduced CPU Energy”, OSDI (and Mobile Computing)
- ★ [Yao, Demers, Shenker 1995] “A Scheduling Model for Reduced CPU Energy”, FOCS

Literature #2: Scheduling

- [Bansal and Harchol-Balter 2001] “Analysis of SRPT Scheduling: Investigating Unfairness”, SIGMETRICS
- ★ [Friedman and Henderson 2003] “Fairness and Efficiency in Web Server Protocols”, SIGMETRICS
- [Harchol-Balter et al. 2002] “Asymptotic Convergence of Scheduling Policies with Respect to Slowdown”, IFIP Performance
- [Rai et al. 2003] “Analysis of LAS Scheduling for Job Size Distributions with High Variance”, SIGMETRICS
- [Wierman and Harchol-Balter 2003] “Classifying Scheduling Policies with Respect to Unfairness in an M/GI/1”, SIGMETRICS

Literature #3: Speed Scaling

- [Albers 2010] “Energy-Efficient Algorithms”, CACM
- [Albers et al. 2014] “Speed Scaling with Parallel Processors”, Algorithmica
- [Bansal et al. 2007] “Speed Scaling to Manage Energy and Temperature”, JACM
- [Bansal et al. 2009a] “Speed Scaling with an Arbitrary Power Function”, SIAM
- [Bansal et al. 2009b] “Speed Scaling for Weighted Flow Time”, SIAM
- ★ [Andrew, Lin, Wierman 2010] “Optimality, Fairness, and Robustness in Speed Scaling Designs”, SIGMETRICS
- [Elahi et al. 2012] “Decoupled Speed Scaling: Analysis and Evaluation”, QEST (PEVA 2014)
- [Elahi et al. 2014] “Turbo-charged Speed Scaling: Analysis and Evaluation”, MASCOTS
- [Wierman et al. 2009] “Power-Aware Speed Scaling in Processor Sharing Systems”, IEEE INFOCOM

Literature #4: Inexact Job Sizes

- [Dell’Amico et al. 2014] “Revisiting Size-based Scheduling with Estimated Job Sizes”, MASCOTS
- ★ [Dell’Amico et al. 2016] “PSBS: Practical Size-Based Scheduling”, IEEE Trans. on Computers
- [Lu et al. 2004] “Size-based Scheduling Policies with Inaccurate Scheduling Information”, MASCOTS
- [Rai et al. 2003] “Analysis of LAS Scheduling for Job Size Distributions with High Variance”, SIGMETRICS
- [Wierman et al. 2008] “Scheduling Despite Inexact Job Size Information”, SIGMETRICS

Literature #5: Systems

- [Hahnel et al. 2012] “Measuring Energy Consumption for Short Code Paths Using RAPL”, PER
- ★ [Meisner et al. 2009] “PowerNap: Eliminating Server Idle Power”, ASPLOS
- [Schroeder et al. 2006] “Web Servers Under Overload: How Scheduling Can Help”, TOIT
- [Skrenes and Williamson 2016] “Experimental Calibration and Validation of a Speed Scaling Simulator”, MASCOTS
- [Snowdon et al. 2009] “Koala: A Platform for OS-level Power Management”, EuroSys
- [Snowdon et al. 2007] “Accurate Online Prediction of Processor and Memory Energy Usage under Voltage Scaling”, Embedded Software
- [Spiliopoulos 2012] “Power-Sleuth: A Tool for Investigating Your Program’s Power Behaviour”, MASCOTS

Talk Outline

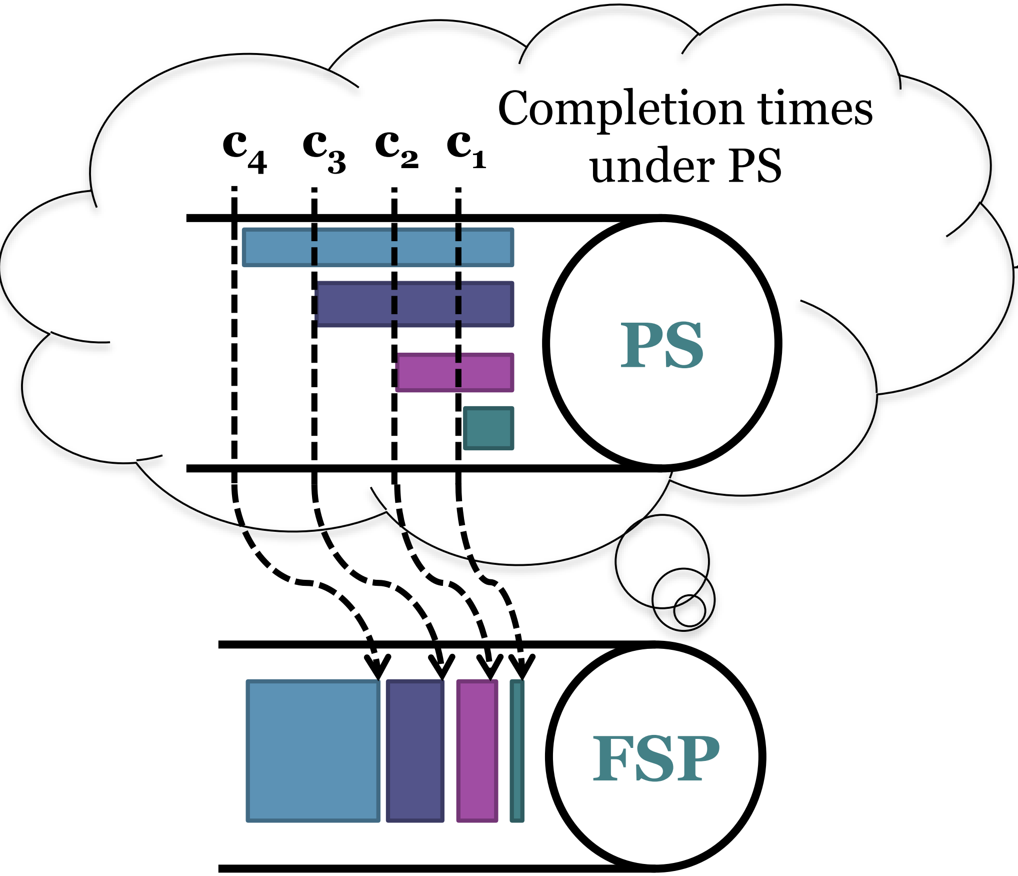
- Introduction and Motivation
- Background and Literature Review
- Review of Key Results and Insights
- Recent Results and Contributions
  - Decoupled Speed Scaling
  - Turbocharged Speed Scaling
  - Experimental Measurements
- Conclusions and Future Directions

Key Results: Single-Speed World

- PS is the gold standard for fairness
- Asymptotic convergence of slowdown for all work-conserving scheduling policies
- SRPT is “Sometimes Unfair”
- YDS is optimal for energy consumption
- FSP dominates PS for response time

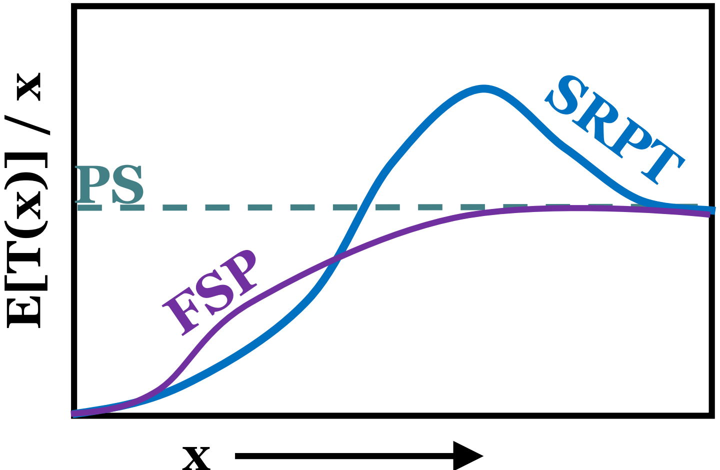
Fair Sojourn Protocol (single-speed world)

- **FSP:** Fair Sojourn Protocol
  [Friedman and Henderson, 2003]



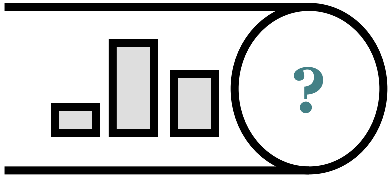
- Compute the completion time under PS
- Sort the jobs based on their virtual completion times
- Execute the job with the earliest PS completion time

Dominance over PS: No job finishes later under FSP than it does under PS. In fact, some (most!) jobs finish earlier under FSP than under PS.



Dynamic Speed Scaling: Decisions

Which job to serve?



At what speed?

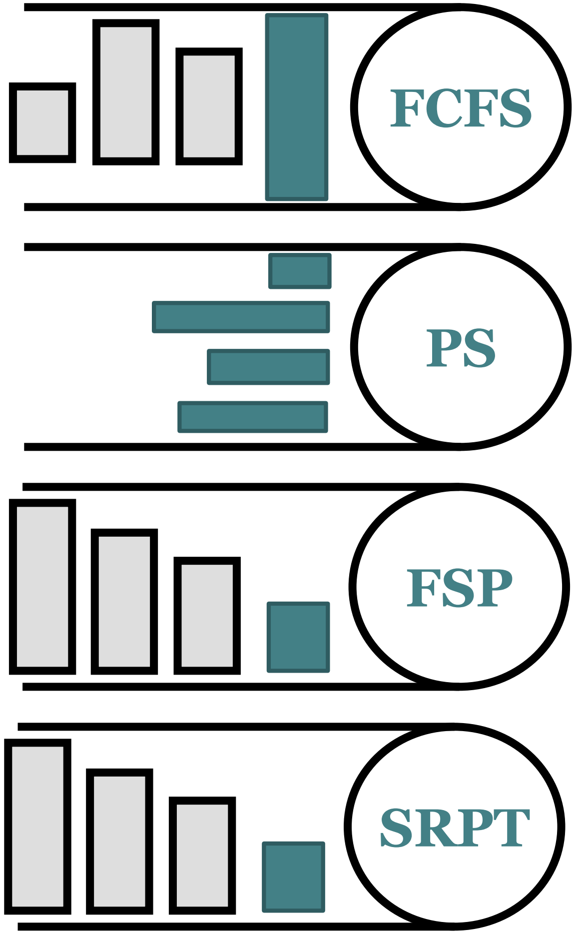
$P(s) = s^{\alpha}$

n: jobs in the system

Optimal policy is:
Shortest-Remaining Processing-Time (SRPT) with $s = P^{-1}(\beta n)$
[Andrew, Lin and Wierman, 2010]

Common heuristic for a variety of scheduling policies:
Job-count-based speed scaling (coupled speed scaling)
$\boldsymbol{s = f(n)}$, in particular $\boldsymbol{s = P^{-1}(\beta n)}$

Dynamic Speed Scaling: Fairness



Biased towards big jobs

Treats all jobs the same

Fair and near-optimal

Biased towards small jobs

Jobs that run when the queue is larger run faster
[Andrew, Lin and Wierman, 2010]

Key Results: Speed Scaling World

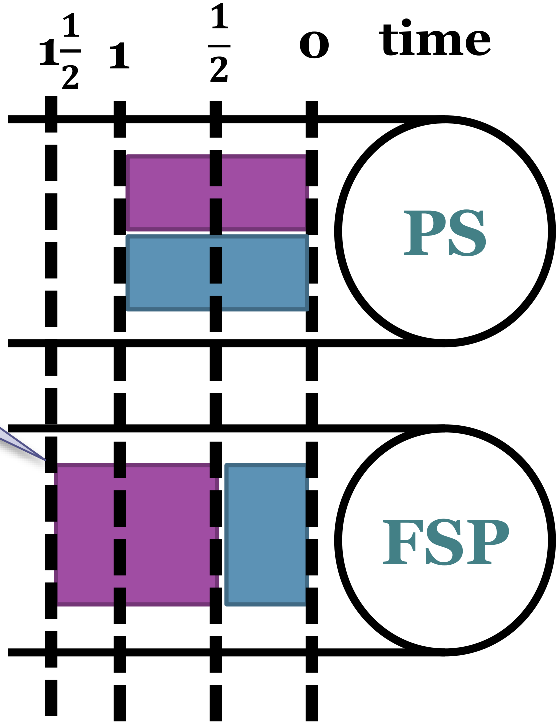
- Speed scaling exacerbates unfairness
- No policy can be optimal, robust, and fair
- Asymptotic convergence of slowdown property no longer holds
- FSP’s dominance of PS breaks under coupled speed scaling
- FSP’s dominance of PS is restored under decoupled speed scaling

Talk Outline

- Introduction and Motivation
- Background and Literature Review
- Review of Key Results and Insights
- Recent Results and Contributions
    - Decoupled Speed Scaling
    - Turbocharged Speed Scaling
    - Experimental Measurements
- Conclusions and Future Directions

FSP with dynamic speed scaling

- Simple example
- Two jobs arrive at time 0
- Both jobs are of size 1
- Speed: $s(n) = n$



Finishes later under FSP

Dominance breaks!

Research Questions

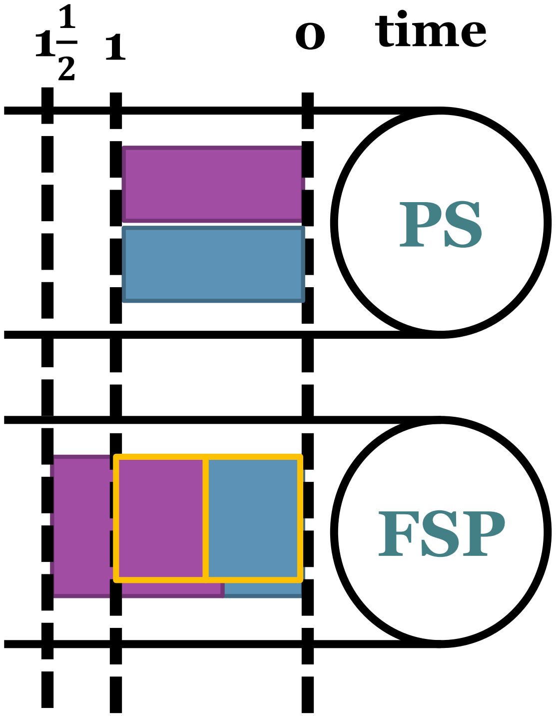
- How to restore dominance property of FSP under dynamic speed scaling?
  - Decoupled Speed Scaling [QEST 2012]
  - Turbocharged Speed Scaling [MASCOTS 2014]
- Which approach is better? By how much?

(Joint work with Maryam Elahi and co-supervisor Philipp Woelfel)

How to preserve dominance? (1)

- Decoupled speed scaling
- Run at the speed of PS
  - Preserves dominance
  - Speeds are not affected by scheduling decisions

- $s_{PS}(n) = n_{PS}$
- $s_{FSP}(.) = s_{ps}(n) = n_{ps}$



Decoupled Speed Scaling Idea

- Run virtual PS in background
- Drive FSP with same speeds that PS used

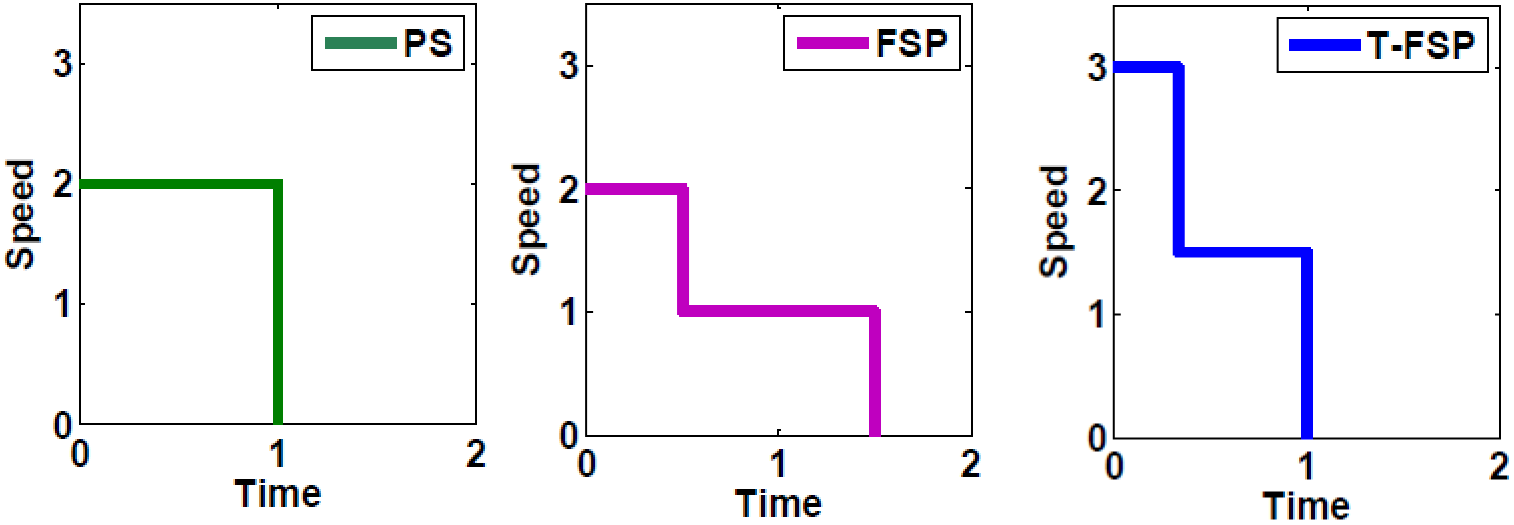
- FSP-PS uses FSP scheduling, but speed depends on occupancy of the virtualized PS system, and not that of FSP itself

Decoupled Speed Scaling [QEST 2012]

- Advantages:
  - Exactly same speeds as PS
  - Exactly same power consumption as PS
  - Much better mean response time than PS
  - Dominance property (preserves fairness)
- Disadvantages:
  - “Unnatural”
  - Difficult to implement
  - Need to compute external speed schedule

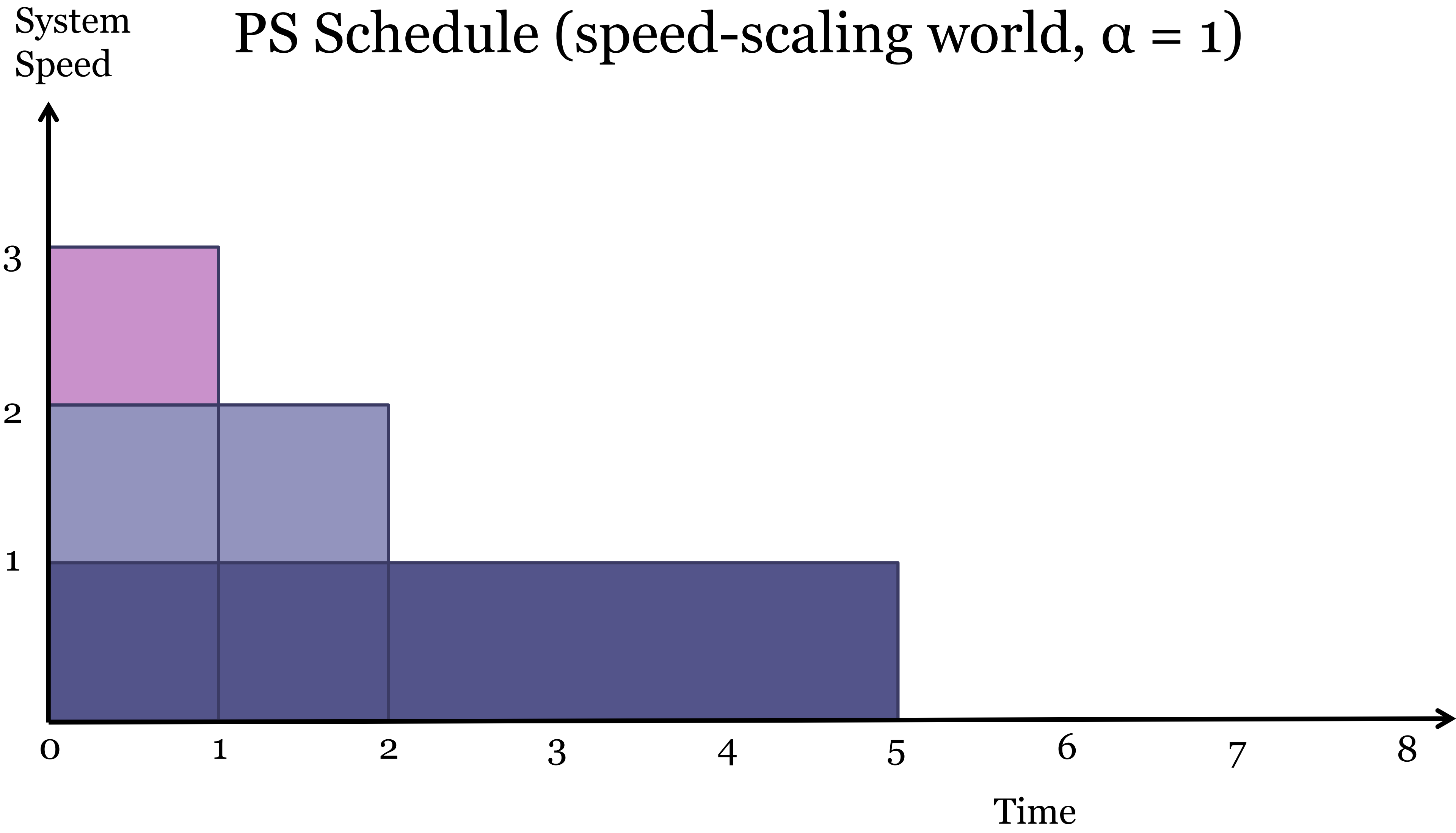
How to preserve dominance? (2)

- Turbocharging [MASCOTS 2014]
- Scale up the speed to finish in time
- Need to compute scaling factor



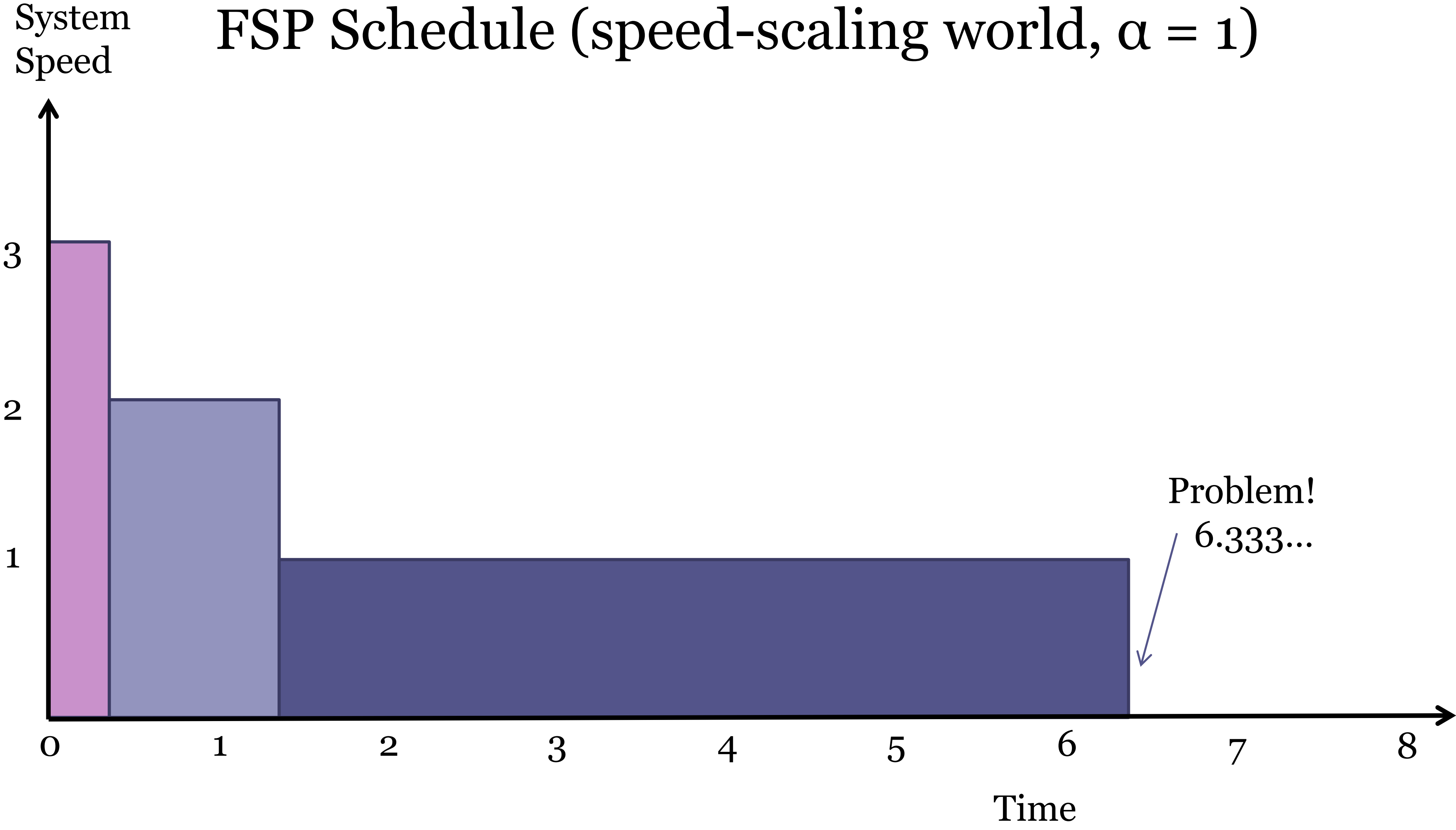
Example: 3 jobs of sizes {1, 2, 5} at time 0

PS Schedule (speed-scaling world, $\alpha = 1$)



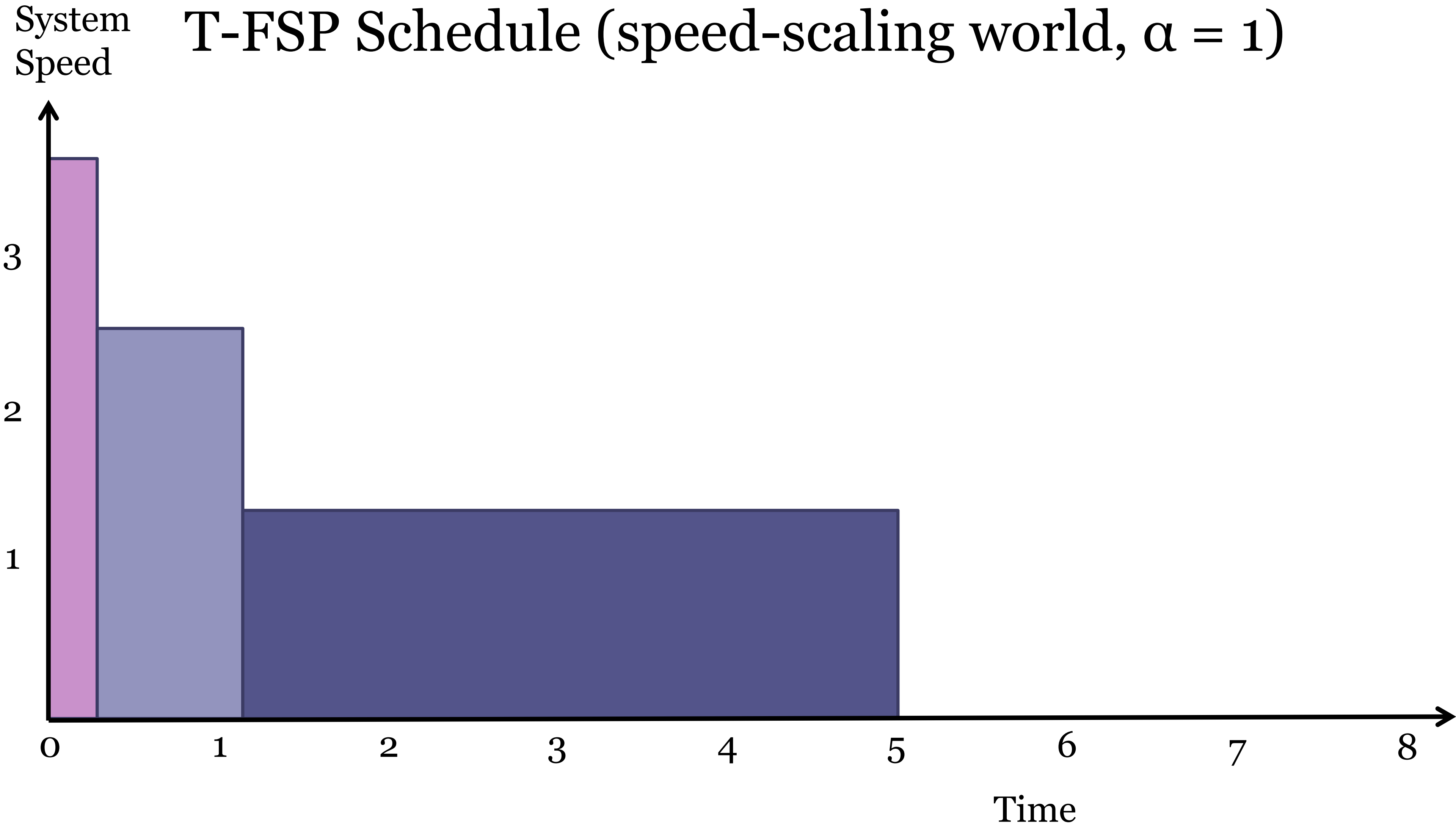
Example: 3 jobs of sizes {1, 2, 5} at time 0

FSP Schedule (speed-scaling world, α = 1)



Example: 3 jobs of sizes {1, 2, 5} at time 0

T-FSP Schedule (speed-scaling world, α = 1)



Single Batch Case

- Consider n jobs with sizes $w_1 \leq w_2 \leq \cdots \leq w_n$
- Compute the largest completion time under PS

$$\max_k X_k{}^{PS} = \sum_{i=1}^{n} \frac{(w_i - w_{i-1})(n-i+1)}{(n-i+1)^{1/\alpha}} = X_n{}^{PS}$$

- Compute the largest completion time under FSP

$$\max_k X_k{}^{FSP} = \sum_{i=1}^{n} \frac{w_i}{(n-i+1)^{1/\alpha}} = X_n{}^{FSP}$$

- Scale up by $b_n = X_n{}^{FSP} / X_n{}^{PS}$

Key Analytical Results

- General case:

$$b_n = 1 + \frac{\sum_{i=1}^{n} (n-i)\, w_i \left(\frac{1}{f(n-i)} - \frac{1}{f(n-i+1)} \right)}{\sum_{i=1}^{n} w_i \left(\frac{n-i+1}{f(n-i+1)} - \frac{n-i}{f(n-i)} \right)}$$

- Special case: $\alpha = 1$

$$b_n = 1 + \frac{1}{w_n} \sum_{i=1}^{n-1} \frac{w_i}{n-i+1}$$

Insights and Observations

- Turbocharging rate can never be less than 1
- For a batch of n jobs, the turbocharging rate:
  - Tends to increase with n
  - Depends directly on sizes w_i of the first n-1 jobs
  - Depends **inversely** on the size w_n of the last job
- Additional observations:
  - Only relative job sizes matter (not absolute sizes)
  - Worst case is homogeneous job sizes
  - Rate bounded by Harmonic numbers ($\alpha = 1$)
  - Larger α value makes turbocharging rate lower
  - Greater variability in job sizes is beneficial

Is turbocharging enough? *NO!*

- $b_k = X_k^{FSP} / X_k^{PS}$

- The necessary condition for dominance is: $b_k \leq b_n;\ \forall k$
  - Does it hold?

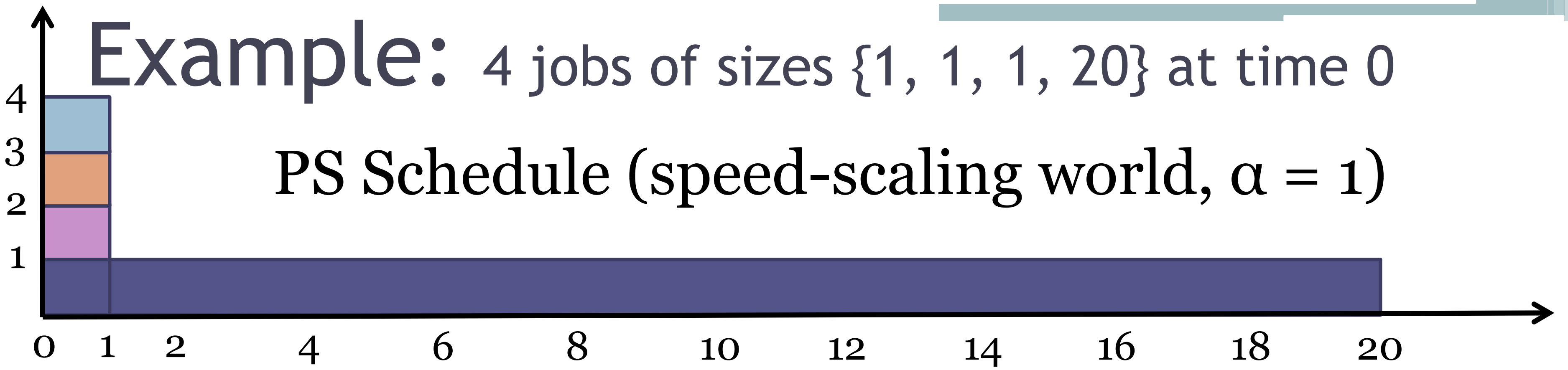
- Assume $w_1, \ldots, w_{n-1} = 1$ and $1 < w_n$

$$b_n = 1 + \frac{H_{n,1/\alpha} - n^{\frac{\alpha-1}{\alpha}}}{w_n + n^{\frac{\alpha-1}{\alpha}} - 1} \quad , b_{n-1} = n^{\frac{1-\alpha}{\alpha}} \left(H_{n,1/\alpha} - 1\right)$$

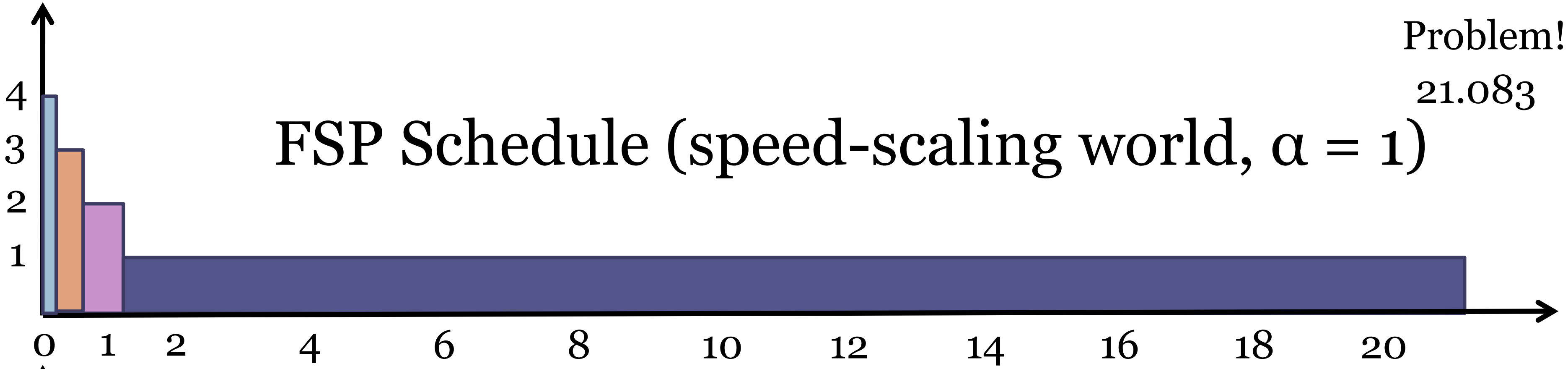
- We can show that $b_{n-1} > b_n$ for all n and w_n where,

$$H_{n,1/\alpha} \geq 1 + n^{\frac{\alpha-1}{\alpha}} \qquad , \qquad w_n > \frac{\left(n^{\frac{\alpha-1}{\alpha}} - 1\right)\left(H_{n,1/\alpha} - 1\right)}{H_{n,1/\alpha} - 1 - n^{\frac{\alpha-1}{\alpha}}}$$

Example: 4 jobs of sizes {1, 1, 1, 20} at time 0

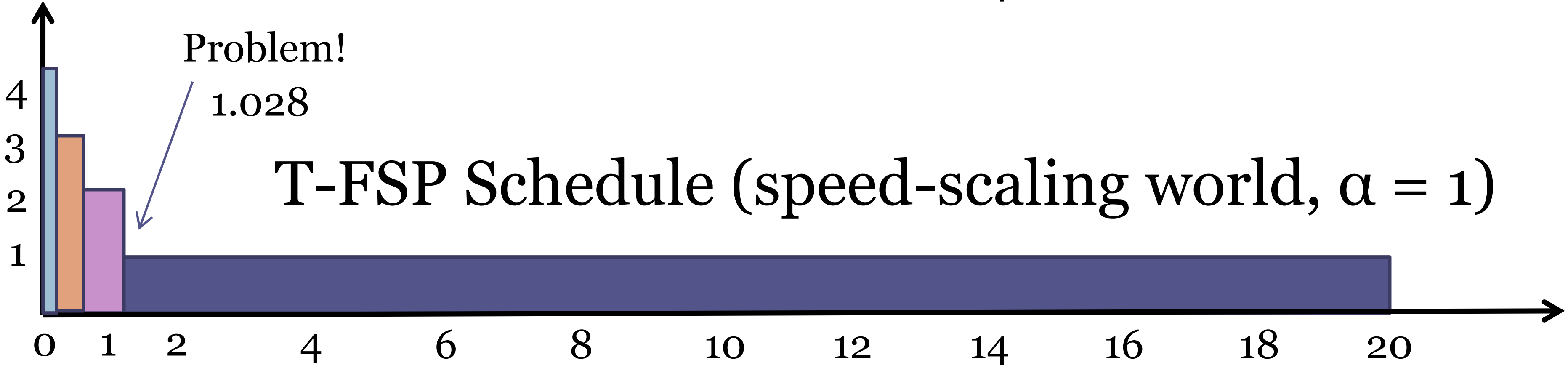


PS Schedule (speed-scaling world, α = 1)



FSP Schedule (speed-scaling world, α = 1)

Problem!
21.083



T-FSP Schedule (speed-scaling world, α = 1)

Problem!
1.028

Critical Job

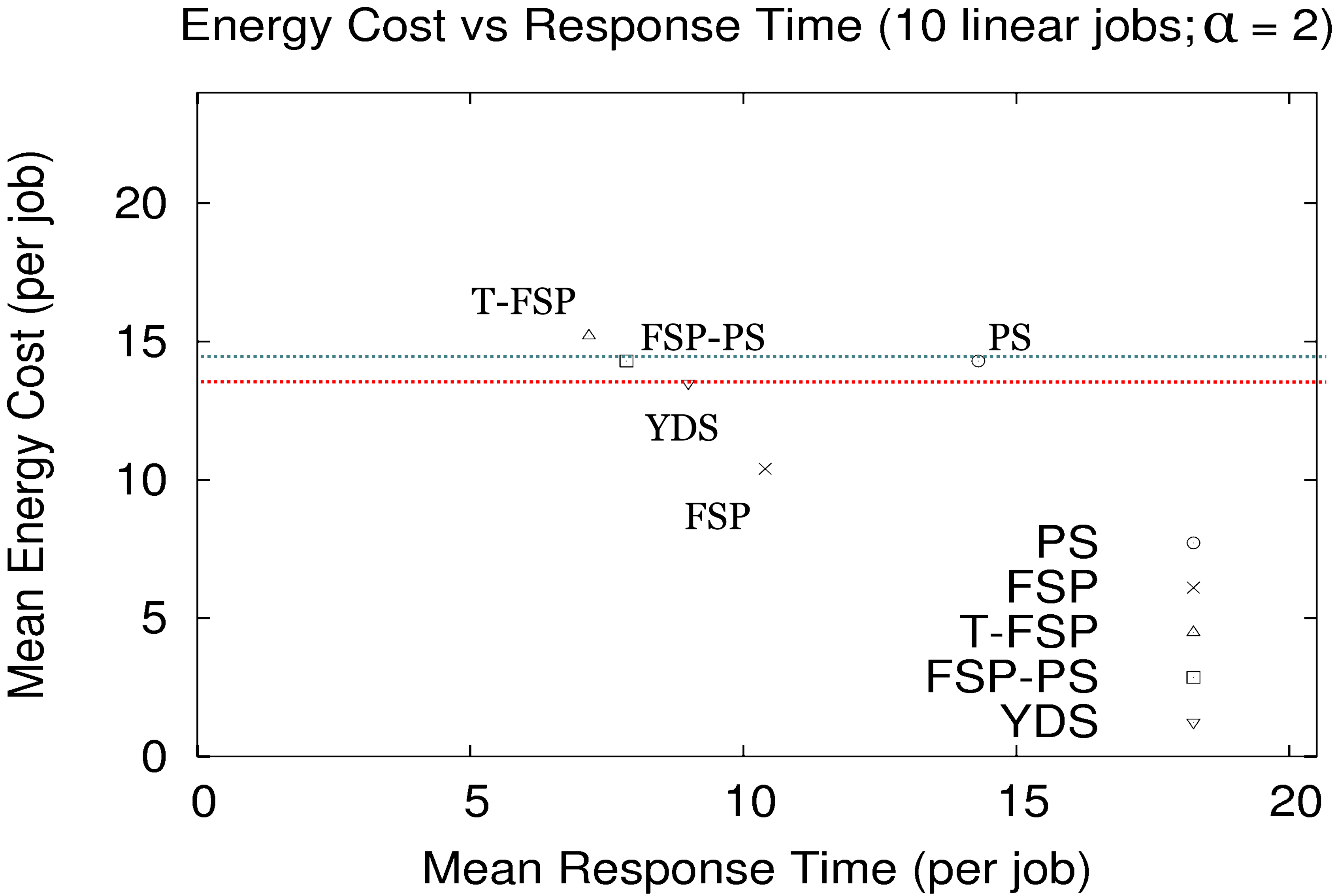
- Need to identify the **critical job** within the batch (i.e., needs highest turbocharging rate)
- Start service of the batch at this rate until the critical job is completed (exactly on time)
- Service rate for the rest of the batch can then be reduced, based on the remaining jobs and their PS completion deadlines (virtual batch)
- We call this **envelope-based** turbocharging

Simulation Evaluation

- Effect of scheduling policy (PS, FSP)
- Effect of speed scaling policy
- Effect of α $(1 \leq \alpha \leq 3)$
- Effect of job size variability
  - Simple batch workloads (n=10, CoV={L,M,H})
  - Dynamic online arrivals (n=100…1000)

- Metrics: response time and energy cost
  - Comparison to decoupled speed scaling
  - Comparison to YDS approach

Simulation Results: Example



Summary (so far)

- Naïve turbocharging of FSP won't work
- Envelope-based turbocharging can work
- Promising approach, and perhaps more practical than decoupled speed scaling
- Energy costs are slightly higher though

- Cognate work: experimental evaluation and comparison of speed scaling policies

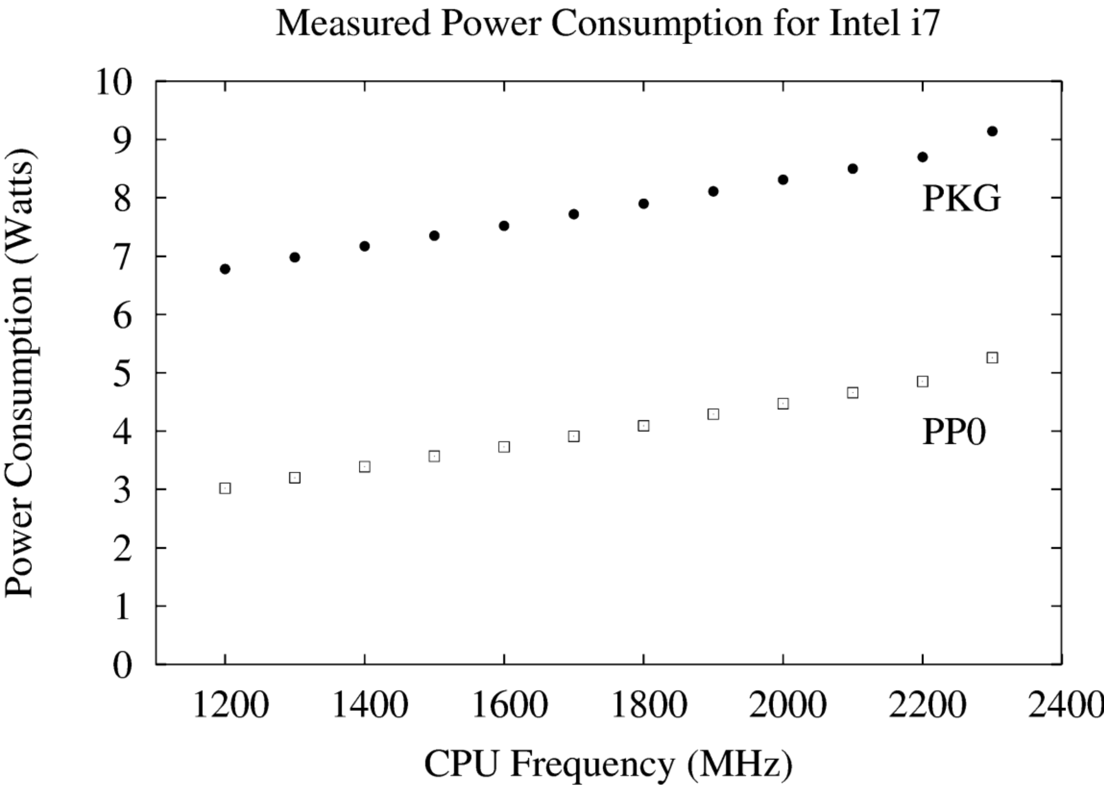
Experimental Results [Skrenes 2016]

- Micro-benchmarking experiments by MSc student Arsham Skrenes
- Fine-grain energy measurements using RAPL MSRs (PP0, PKG)
- Platform: 2.3 GHz quad-core Intel i7-3615 QM Ivy Bridge processor
- 12 discrete speeds ranging from 1200 MHz to 2300 MHz
- CPU-bound compute-intensive workload (primality testing)
- Reported results are the mean from 10 replications (error < 2%)

- Default governors (Ubuntu Linux 14.04 LTS):
  - performance: use max frequency available
  - powersave: use min frequency available
  - ondemand: dynamic using up/down thresholds
  - conservative: like ondemand, but gradual increase
  - userspace: user-defined control

Frequency (MHz)	PP0 (W)	PKG (W)
2301 (3300)	11.5	15.3
2300	5.4	9.2
2200	5.0	8.9
2100	4.8	8.6
2000	4.6	8.4
1900	4.5	8.3
1800	4.3	8.0
1700	4.1	7.9
1600	3.9	7.6
1500	3.7	7.5
1400	3.5	7.3
1300	3.3	7.1
1200	3.1	6.9

Quite unpredictable and uncontrollable!



Highly linear throughout most of range!

Plus multiple sleep and idle modes (not shown here)

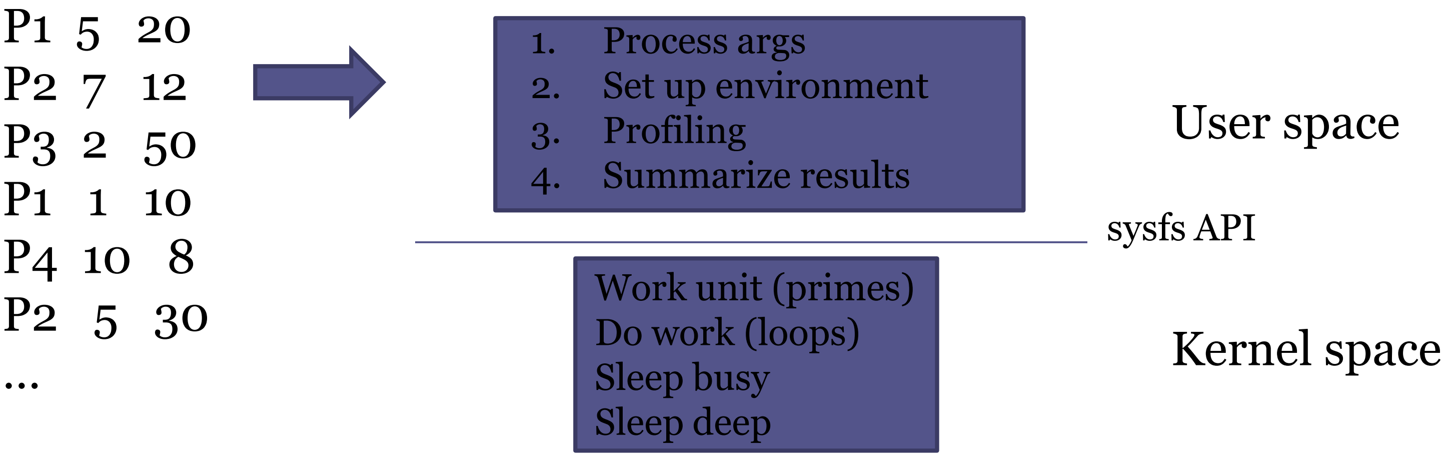
Frequency (MHz)	PP0 (W)	PKG (W)	Context Switch (us)
2301 (3300)	11.5	15.3	1.140
2300	5.4	9.2	1.634
2200	5.0	8.9	1.708
2100	4.8	8.6	1.808
2000	4.6	8.4	1.898
1900	4.5	8.3	1.999
1800	4.3	8.0	2.118
1700	4.1	7.9	2.213
1600	3.9	7.6	2.369
1500	3.7	7.5	2.526
1400	3.5	7.3	2.709
1300	3.3	7.1	2.886
1200	3.1	6.9	3.167

Frequency (MHz)	PP0 (W)	PKG (W)	Context Switch (us)	Speed Switch (us)
2301 (3300)	11.5	15.3	1.140	0.76
2300	5.4	9.2	1.634	1.09
2200	5.0	8.9	1.708	1.14
2100	4.8	8.6	1.808	1.20
2000	4.6	8.4	1.898	1.26
1900	4.5	8.3	1.999	1.32
1800	4.3	8.0	2.118	1.38
1700	4.1	7.9	2.213	1.47
1600	3.9	7.6	2.369	1.56
1500	3.7	7.5	2.526	1.67
1400	3.5	7.3	2.709	1.81
1300	3.3	7.1	2.886	1.93
1200	3.1	6.9	3.167	2.09

Frequency (MHz)	PP0 (W)	PKG (W)	Context Switch (us)	Speed Switch (us)	Mode Switch (ns)
2301 (3300)	11.5	15.3	1.140	0.76	44.8
2300	5.4	9.2	1.634	1.09	64.2
2200	5.0	8.9	1.708	1.14	67.0
2100	4.8	8.6	1.808	1.20	70.2
2000	4.6	8.4	1.898	1.26	73.7
1900	4.5	8.3	1.999	1.32	78.3
1800	4.3	8.0	2.118	1.38	81.9
1700	4.1	7.9	2.213	1.47	86.7
1600	3.9	7.6	2.369	1.56	92.1
1500	3.7	7.5	2.526	1.67	98.6
1400	3.5	7.3	2.709	1.81	105.3
1300	3.3	7.1	2.886	1.93	113.4
1200	3.1	6.9	3.167	2.09	123.1

Profilo Design [Skrenes 2016]

- Flexible framework for the experimental evaluation of arbitrary scheduling and speed scaling policies (emulation)
- Hybrid user-mode and kernel-mode implementation
- User space: CSV file input of workload to be performed
- Kernel space: carefully controlled API for job execution, timing, and energy measurement using RAPL MSRs



P1	5	20
P2	7	12
P3	2	50
P1	1	10
P4	10	8
P2	5	30
…		

1. Process args
2. Set up environment
3. Profiling
4. Summarize results

User space

sysfs API

Work unit (primes)
Do work (loops)
Sleep busy
Sleep deep

Kernel space

Speed Scaling Results [Skrenes 2016]

Profilo results for batch of n = 12 jobs on Ubuntu Linux system
Platform: 2.3 GHz quad-core Intel i7-3615 QM Ivy Bridge processor
12 discrete speeds ranging from 1200 MHz to 2300 MHz

Policy
PS
FSP-PS
YDS

Speed Scaling Results [Skrenes 2016]

Profilo results for batch of n = 12 jobs on Ubuntu Linux system
Platform: 2.3 GHz quad-core Intel i7-3615 QM Ivy Bridge processor
12 discrete speeds ranging from 1200 MHz to 2300 MHz

Homogeneous Job Sizes

Policy	Time (s)	E[T] (s)	PP0 (J)	PKG (J)
PS	14.57	14.55	76.80	131.50
FSP-PS	14.57	7.89	76.77	131.60
YDS	14.55	7.88	76.49	130.93

Speed Scaling Results [Skrenes 2016]

Profilo results for batch of n = 12 jobs on Ubuntu Linux system
Platform: 2.3 GHz quad-core Intel i7-3615 QM Ivy Bridge processor
12 discrete speeds ranging from 1200 MHz to 2300 MHz

	Homogeneous Job Sizes				Linear Job Sizes			
Policy	**Time (s)**	**E[T] (s)**	**PP0 (J)**	**PKG (J)**	**Time (s)**	**E[T] (s)**	**PP0 (J)**	**PKG (J)**
PS	14.57	14.55	76.80	131.50	46.23	30.16	199.99	372.98
FSP-PS	14.57	7.89	76.77	131.60	46.21	16.33	199.41	372.36
YDS	14.55	7.88	76.49	130.93	45.80	17.81	198.83	369.88

Speed Scaling Results [Skrenes 2016]

Profilo results for batch of n = 12 jobs on Ubuntu Linux system
Platform: 2.3 GHz quad-core Intel i7-3615 QM Ivy Bridge processor
12 discrete speeds ranging from 1200 MHz to 2300 MHz

	Homogeneous Job Sizes				Linear Job Sizes				Multiplicative Job Sizes			
Policy	**Time (s)**	**E[T] (s)**	**PP0 (J)**	**PKG (J)**	**Time (s)**	**E[T] (s)**	**PP0 (J)**	**PKG (J)**	**Time (s)**	**E[T] (s)**	**PP0 (J)**	**PKG (J)**
PS	14.57	14.55	76.80	131.50	46.23	30.16	199.99	372.98	166.15	38.10	562.47	1184.36
FSP-PS	14.57	7.89	76.77	131.60	46.21	16.33	199.41	372.36	166.08	25.43	560.35	1180.83
YDS	14.55	7.88	76.49	130.93	45.80	17.81	198.83	369.88	163.12	27.15	560.94	1170.05

Speed Scaling Results [Skrenes 2016]

Profilo results for batch of n = 12 jobs on Ubuntu Linux system
Platform: 2.3 GHz quad-core Intel i7-3615 QM Ivy Bridge processor
12 discrete speeds ranging from 1200 MHz to 2300 MHz

	Homogeneous Job Sizes				Linear Job Sizes				Multiplicative Job Sizes			
Policy	**Time (s)**	**E[T] (s)**	**PP0 (J)**	**PKG (J)**	**Time (s)**	**E[T] (s)**	**PP0 (J)**	**PKG (J)**	**Time (s)**	**E[T] (s)**	**PP0 (J)**	**PKG (J)**
PS	14.57	14.55	76.80	131.50	46.23	30.16	199.99	372.98	166.15	38.10	562.47	1184.36
FSP-PS	14.57	7.89	76.77	131.60	46.21	16.33	199.41	372.36	166.08	25.43	560.35	1180.83
YDS	14.55	7.88	76.49	130.93	45.80	17.81	198.83	369.88	163.12	27.15	560.94	1170.05

Observation 1: Decoupled speed scaling (FSP-PS) provides a significant response time advantage over PS, for the "same" energy costs

Speed Scaling Results [Skrenes 2016]

Profilo results for batch of n = 12 jobs on Ubuntu Linux system
Platform: 2.3 GHz quad-core Intel i7-3615 QM Ivy Bridge processor
12 discrete speeds ranging from 1200 MHz to 2300 MHz

	Homogeneous Job Sizes				Linear Job Sizes				Multiplicative Job Sizes			
Policy	**Time (s)**	**E[T] (s)**	**PP0 (J)**	**PKG (J)**	**Time (s)**	**E[T] (s)**	**PP0 (J)**	**PKG (J)**	**Time (s)**	**E[T] (s)**	**PP0 (J)**	**PKG (J)**
PS	14.57	14.55	76.80	131.50	46.23	30.16	199.99	372.98	166.15	38.10	562.47	1184.36
FSP-PS	14.57	7.89	76.77	131.60	46.21	16.33	199.41	372.36	166.08	25.43	560.35	1180.83
YDS	14.55	7.88	76.49	130.93	45.80	17.81	198.83	369.88	163.12	27.15	560.94	1170.05

Observation 2: The response time advantage of FSP-PS decreases as job size variability increases

Speed Scaling Results [Skrenes 2016]

Profilo results for batch of n = 12 jobs on Ubuntu Linux system
Platform: 2.3 GHz quad-core Intel i7-3615 QM Ivy Bridge processor
12 discrete speeds ranging from 1200 MHz to 2300 MHz

	Homogeneous Job Sizes				Linear Job Sizes				Multiplicative Job Sizes			
Policy	**Time (s)**	**E[T] (s)**	**PPo (J)**	**PKG (J)**	**Time (s)**	**E[T] (s)**	**PPo (J)**	**PKG (J)**	**Time (s)**	**E[T] (s)**	**PPo (J)**	**PKG (J)**
PS	14.57	14.55	76.80	131.50	46.23	30.16	199.99	372.98	166.15	38.10	562.47	1184.36
FSP-PS	14.57	7.89	76.77	131.60	46.21	16.33	199.41	372.36	166.08	25.43	560.35	1180.83
YDS	14.55	7.88	76.49	130.93	45.80	17.81	198.83	369.88	163.12	27.15	560.94	1170.05

Observation 3: FSP-PS has a slight energy advantage over PS because of fewer context switches between jobs

Speed Scaling Results [Skrenes 2016]

Profilo results for batch of n = 12 jobs on Ubuntu Linux system
Platform: 2.3 GHz quad-core Intel i7-3615 QM Ivy Bridge processor
12 discrete speeds ranging from 1200 MHz to 2300 MHz

	Homogeneous Job Sizes				Linear Job Sizes				Multiplicative Job Sizes			
Policy	**Time (s)**	**E[T] (s)**	**PP0 (J)**	**PKG (J)**	**Time (s)**	**E[T] (s)**	**PP0 (J)**	**PKG (J)**	**Time (s)**	**E[T] (s)**	**PP0 (J)**	**PKG (J)**
PS	14.57	14.55	76.80	131.50	46.23	30.16	199.99	372.98	166.15	38.10	562.47	1184.36
FSP-PS	14.57	7.89	76.77	131.60	46.21	16.33	199.41	372.36	166.08	25.43	560.35	1180.83
YDS	14.55	7.88	76.49	130.93	45.80	17.81	198.83	369.88	163.12	27.15	560.94	1170.05

Observation 4: YDS has the lowest energy consumption among these policies
(even better than expected due to discretization effect, and no speed changes)

Talk Outline

- Introduction and Motivation
- Background and Literature Review
- Review of Key Results and Insights
- Recent Results and Contributions
  - Decoupled Speed Scaling
  - Turbocharged Speed Scaling
  - Experimental Measurements
- Conclusions and Future Directions

Summary and Conclusions

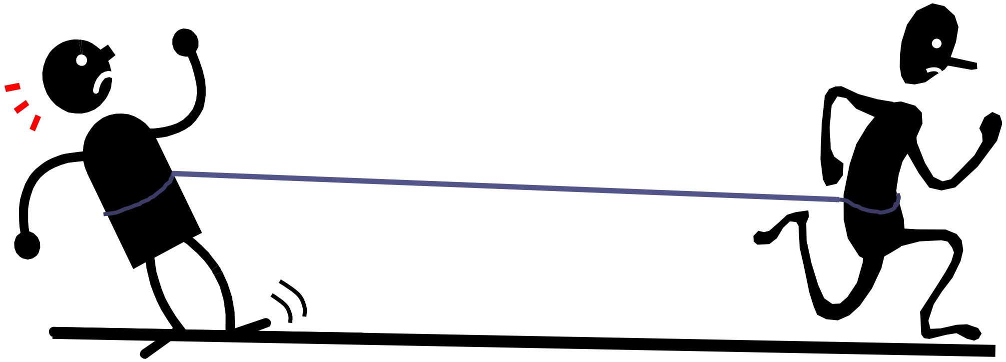
- There is a broad and diverse set of literature on speed scaling systems over the past 20+ years
- There is a dichotomy between theoretical work and systems work on speed scaling
- Modern processors provide surprisingly rich functionality for speed scaling that is not yet well exploited by systems software

Future Directions

- Cost function for speed scaling optimization
- Redefining the benchmark for fairness
- Autoscaling effects and overload regimes
- Extending PSBS to speed scaling scenario
- Practical schedulers and speed scalers for modern operating systems that better exploit the available hardware features
- Speed scaling policies on multi-core systems

The End

- Thank you!
- Questions?



Virtual Batches

- The idea of **virtual batches** can also be used to handle dynamic job arrivals
- At point of new arrival, the new job competes with old jobs that are either done, partially done, or not yet started
- Re-order jobs based on remaining sizes, and re-compute turbocharging rates