A power-tunable algorithm to compute single-source shortest paths

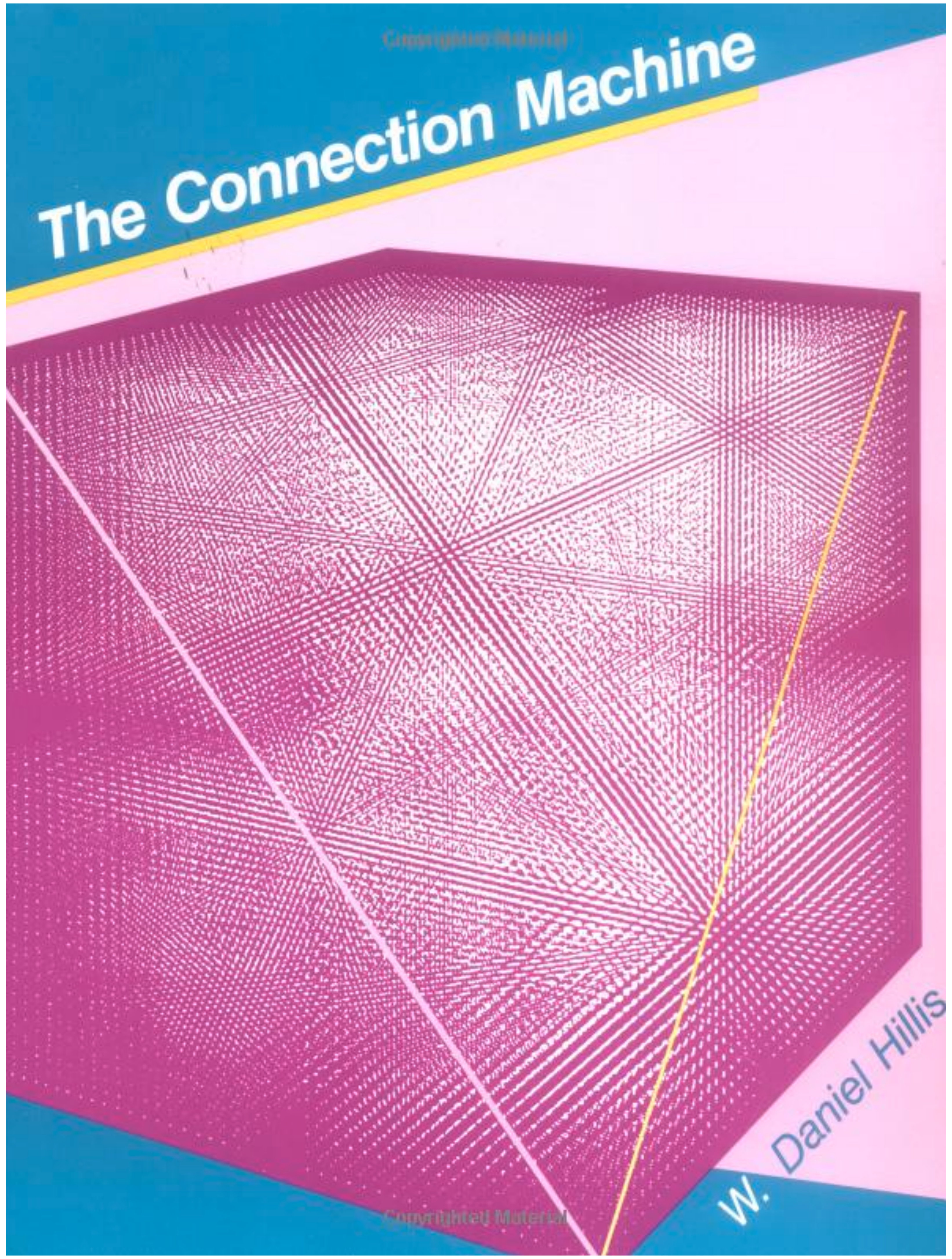
Sara Karamati · Jeffrey Young · **Richard (Rich) Vuduc**

November 17, 2017 — University of Colorado at Colorado Springs

Georgia Tech College of Computing
Computational Science and Engineering

hpcgarage

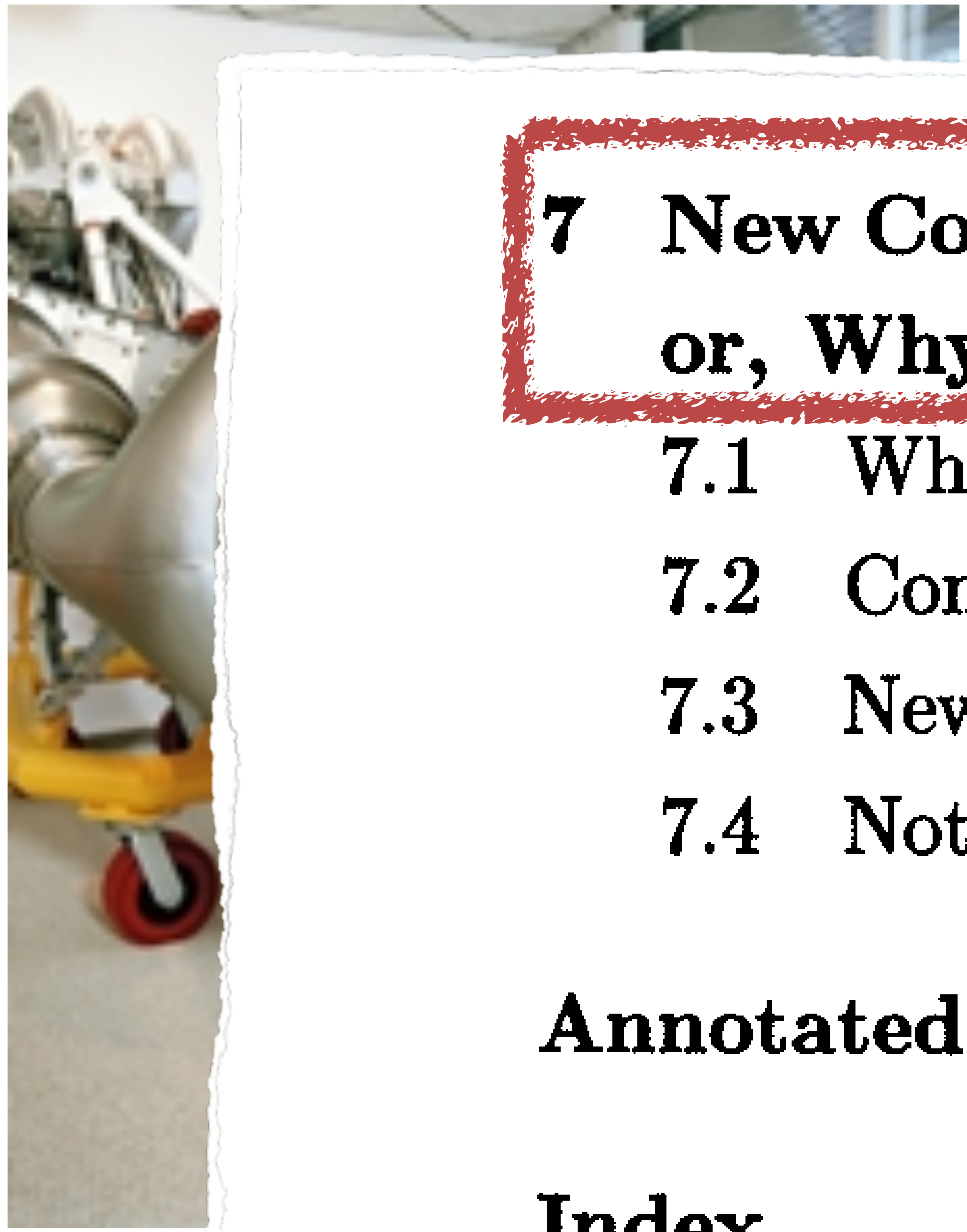
ACM Doctoral Dissertation Award Winner (1985)

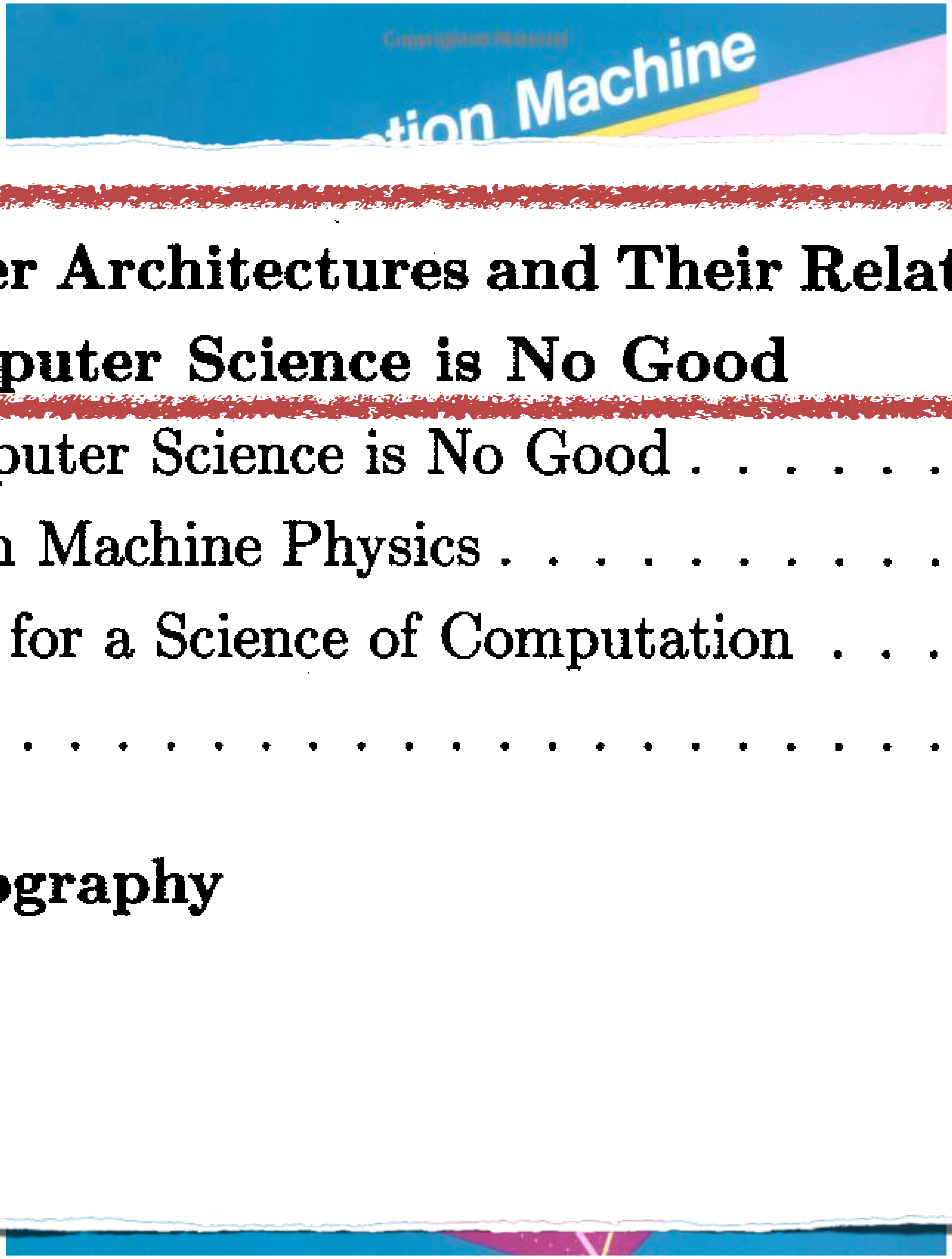


http://www.amazon.com/Connection-Machine-Artificial-Intelligence/dp/0262580977/ref=sr_1_1?ie=UTF8&qid=1319571691&sr=8-1
http://www.technologyreview.com/files/1158/1106_Q-A.tif.jpg

ACM Doctoral Dissertation Award Winner (1985)

hpcgarage.org/memsys16

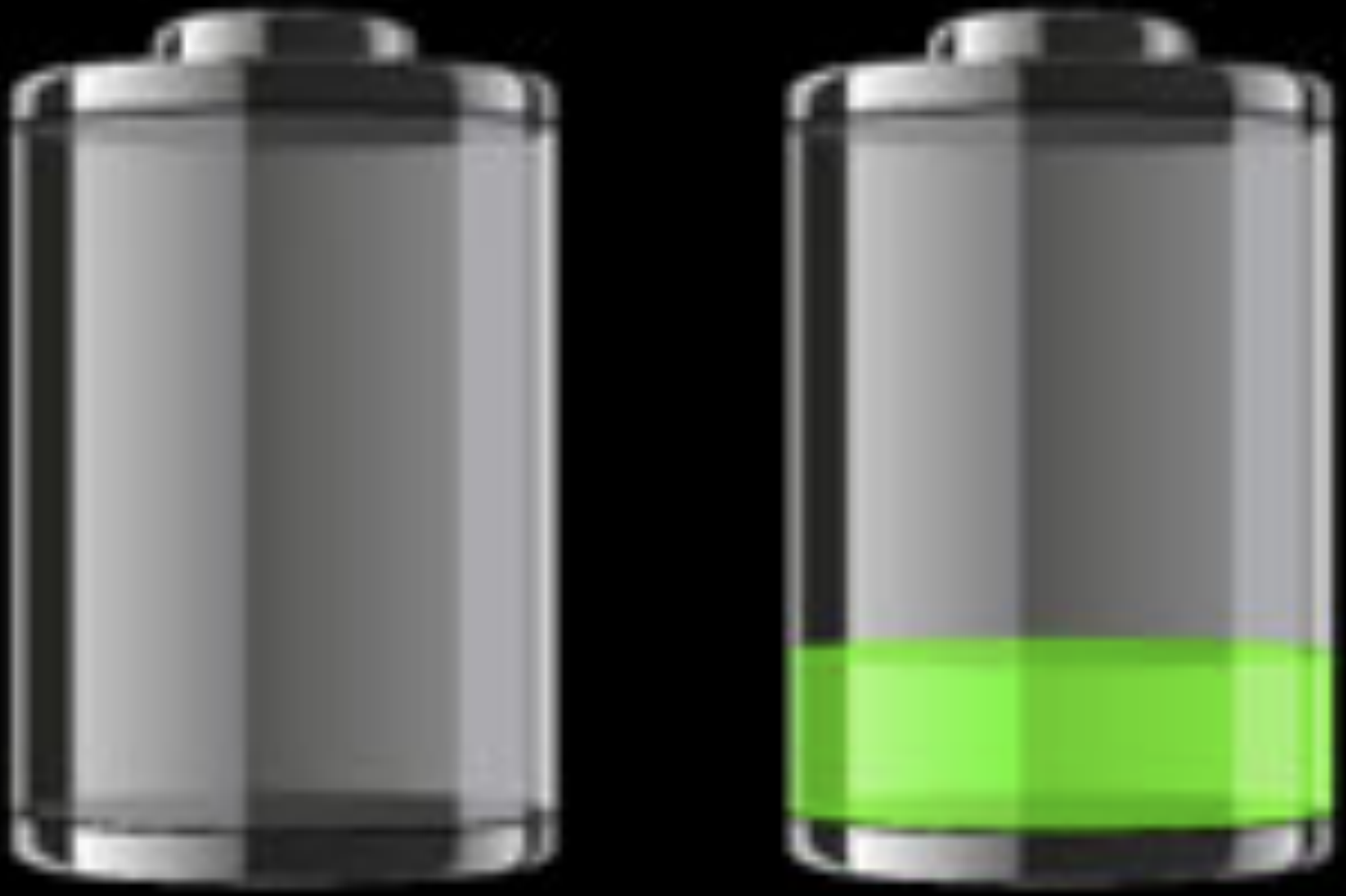


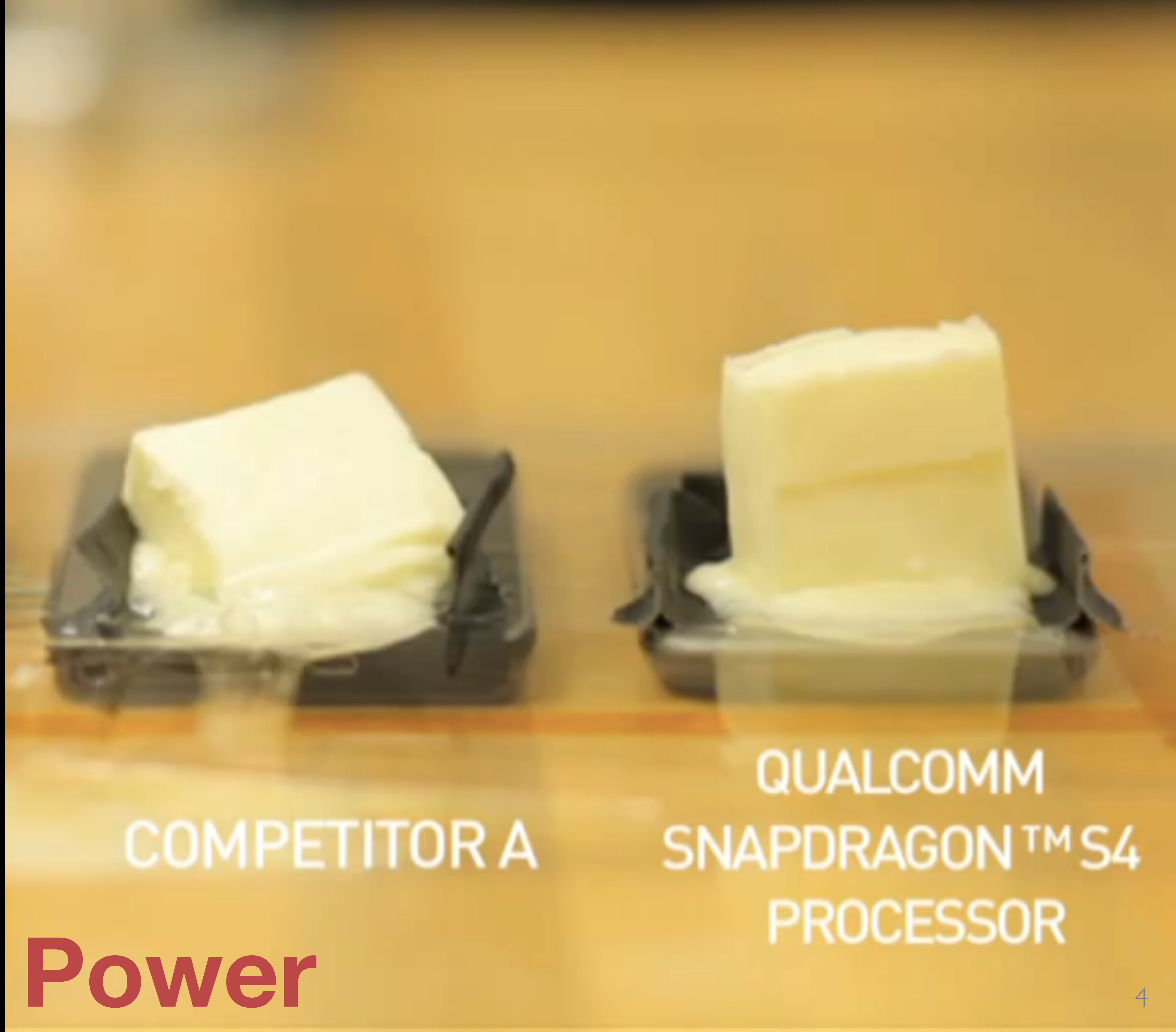


http://www.amazon.com/Connection-Machine-Artificial-Intelligence/dp/0262580977/ref=sr_1_1?ie=UTF8&qid=1319571691&sr=8-1
http://www.technologyreview.com/files/1158/1106_Q-A.tif.jpg

What **physical constraint** limits speed today?

What **physical constraint** limits speed today?





Energy **Power**

(An aside on the relationship between computational performance and power)

ImageNet Dataset

IMAGENET

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., ... & Fei-Fei, L. (2015). Imagenet large scale visual recognition challenge. *arXiv preprint arXiv:1409.0575*. [web]

(4 GPUs)
x (250 Watts / GPU)
x (1 week)
~ 0.6 billion Joules

(4 GPUs)
x (250 Watts / GPU)
x (1 week)
~ 0.6 billion Joules

(1 brain)
x (20 Watts / brain)
x (1 year)
~ 0.6 billion Joules

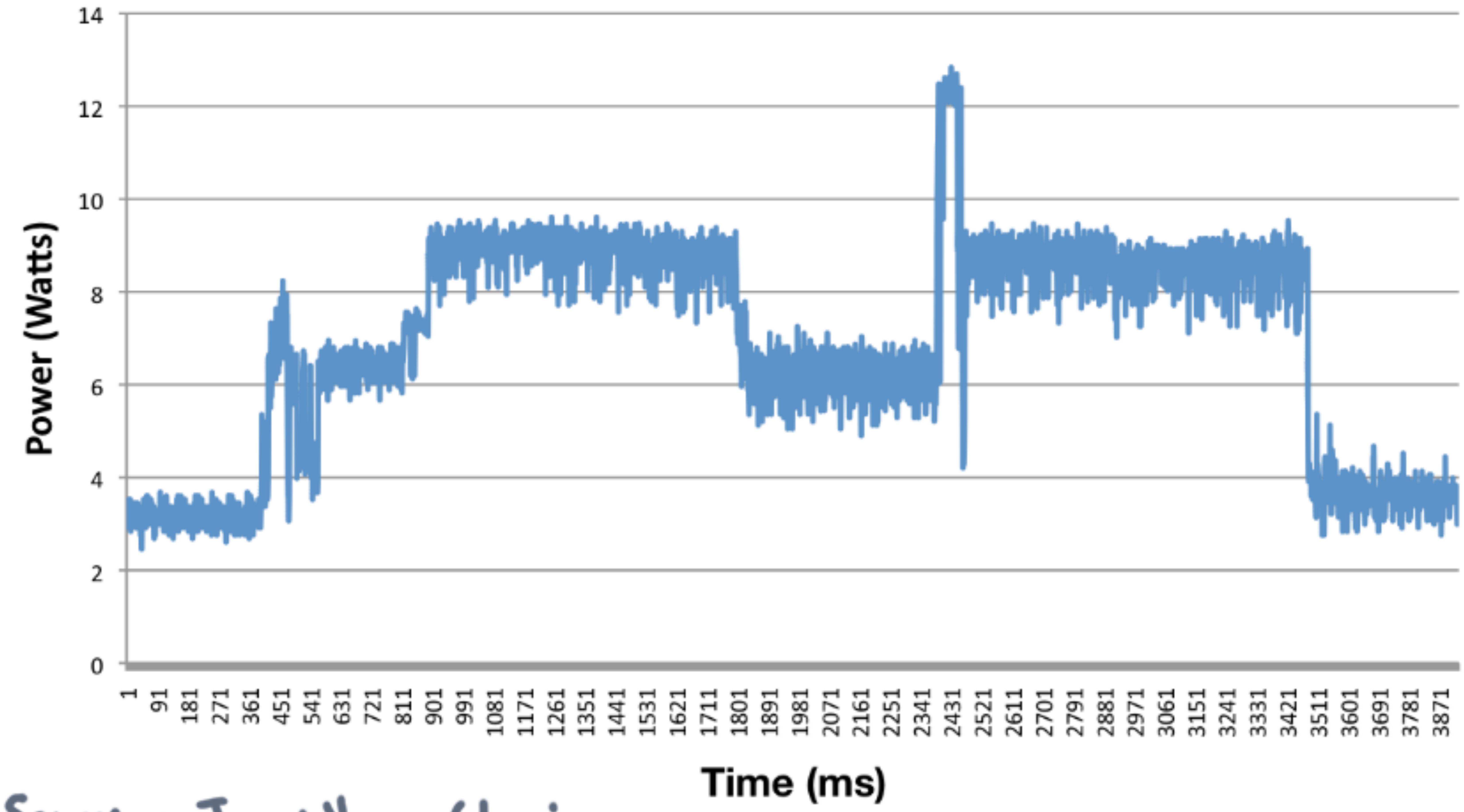
(4 GPUs)
x (250 Watts / GPU)
x (1 week)
~ 0.6 billion Joules

??

(1 brain)
x (20 Watts / brain)
x (1 year)
~ 0.6 billion Joules

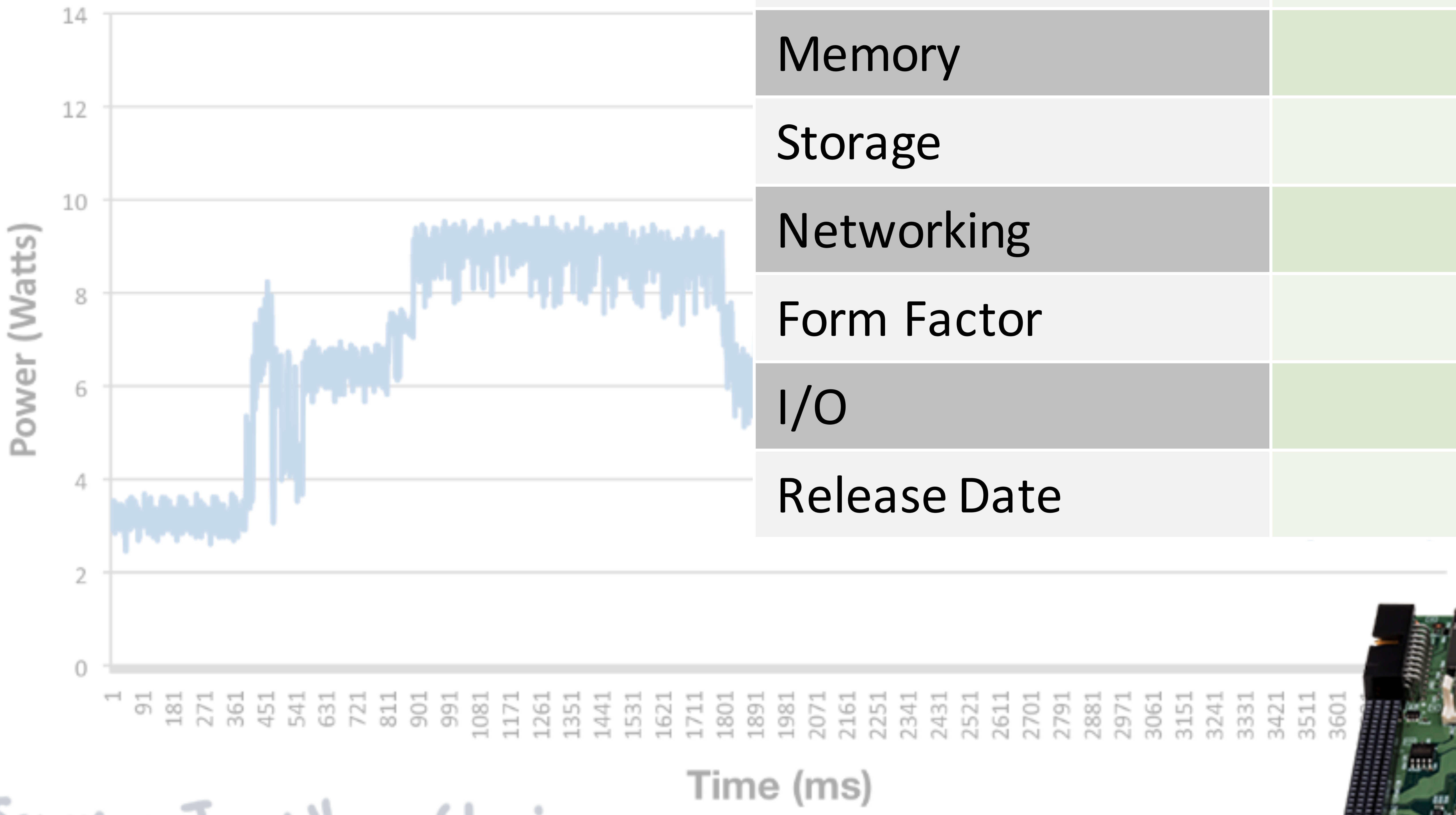
Power Limits

Power = $\frac{\text{Energy}}{\text{Time}}$



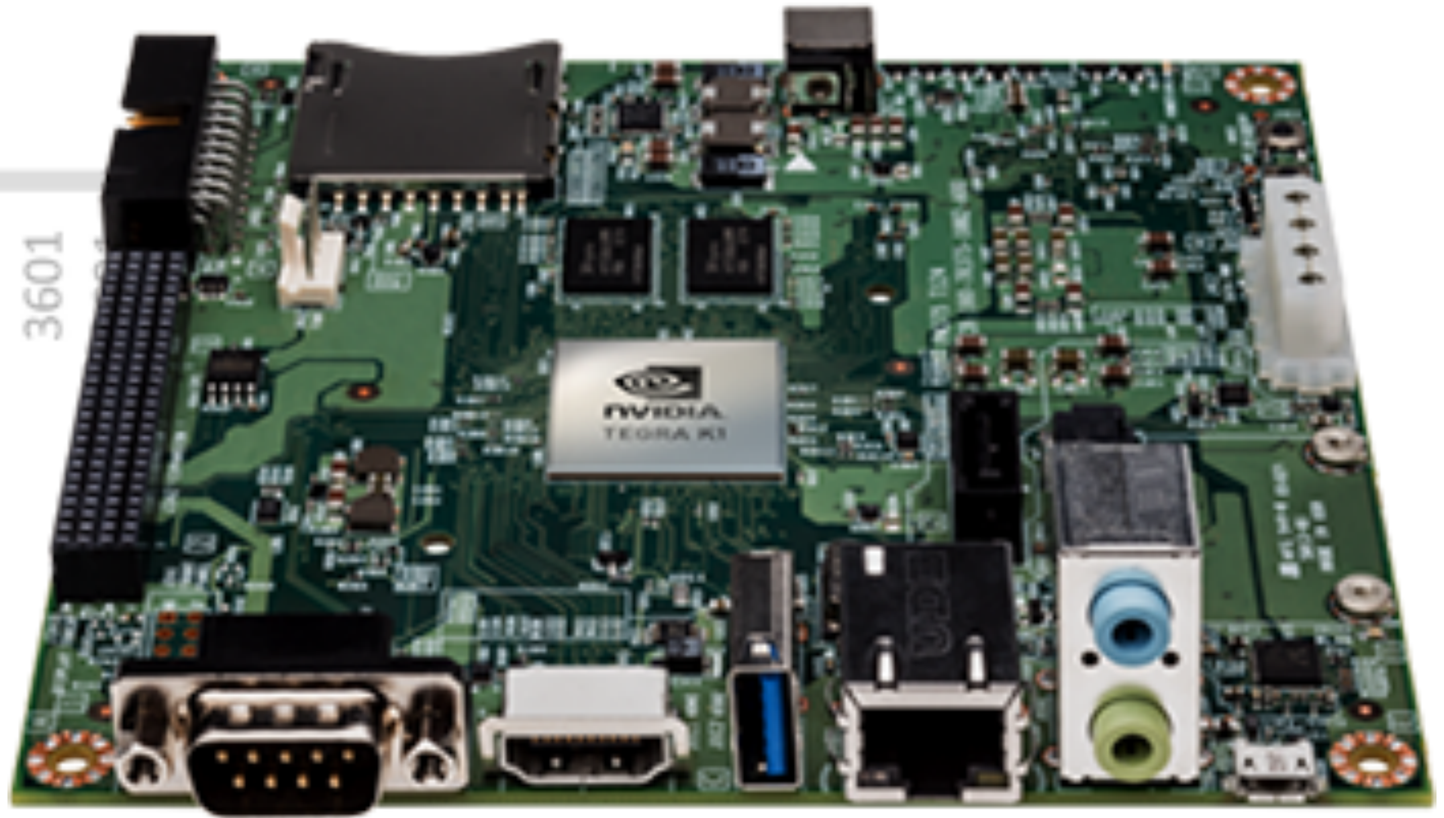
Source: Jee Whan Choi

Power Limits



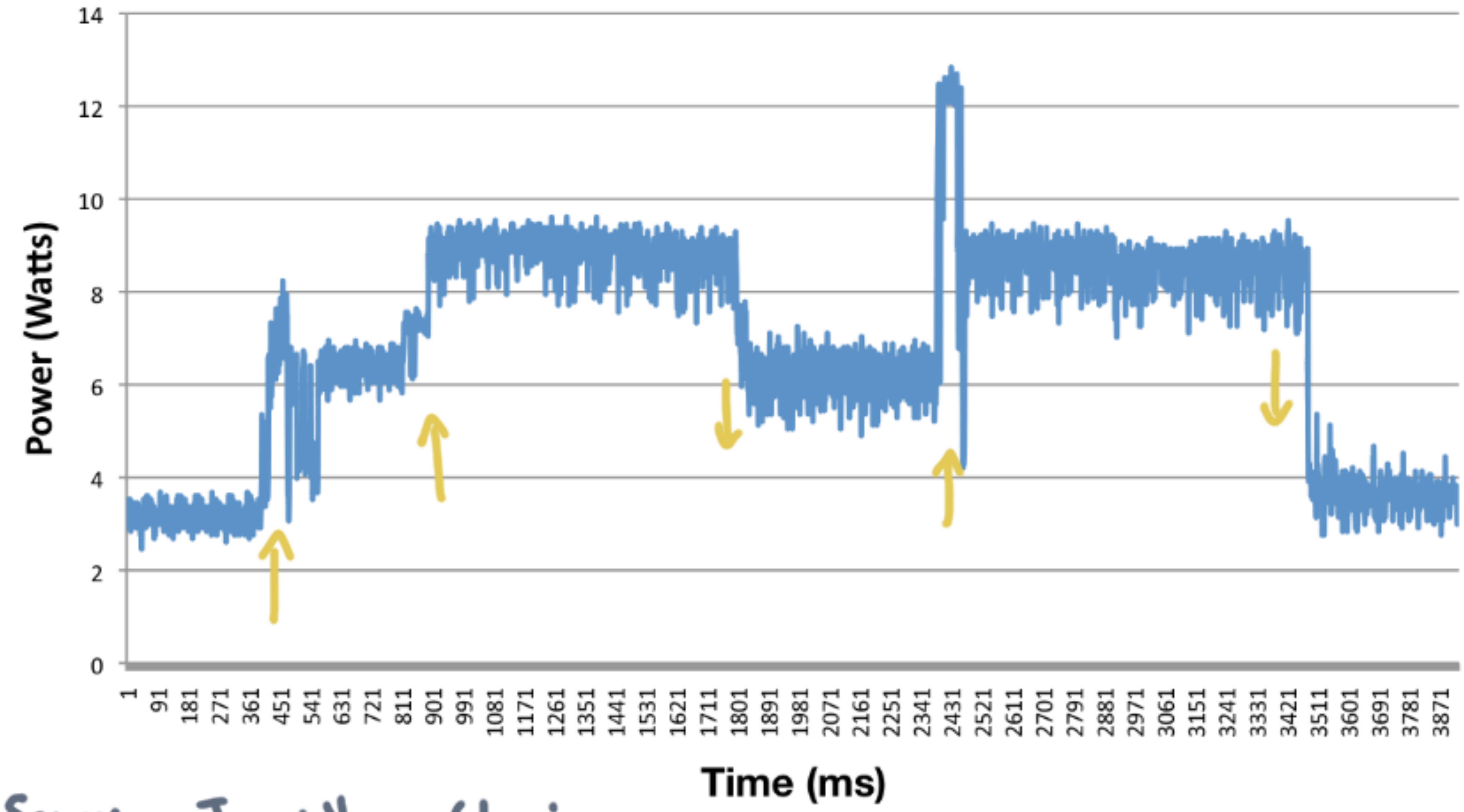
Source: Jee Whan Choi

	Jetson TK1
CPU	ARM A15 (*32-bit, 2.3 GHz, 4+1 cores*)
GPU	192 core Kepler, 326 GF/s (*peak*)
Memory	2 GB LPDDR3
Storage	16 GB eMMC
Networking	Ethernet
Form Factor	Dev board
I/O	USB, HDMI, Serial
Release Date	2014



Power Limits

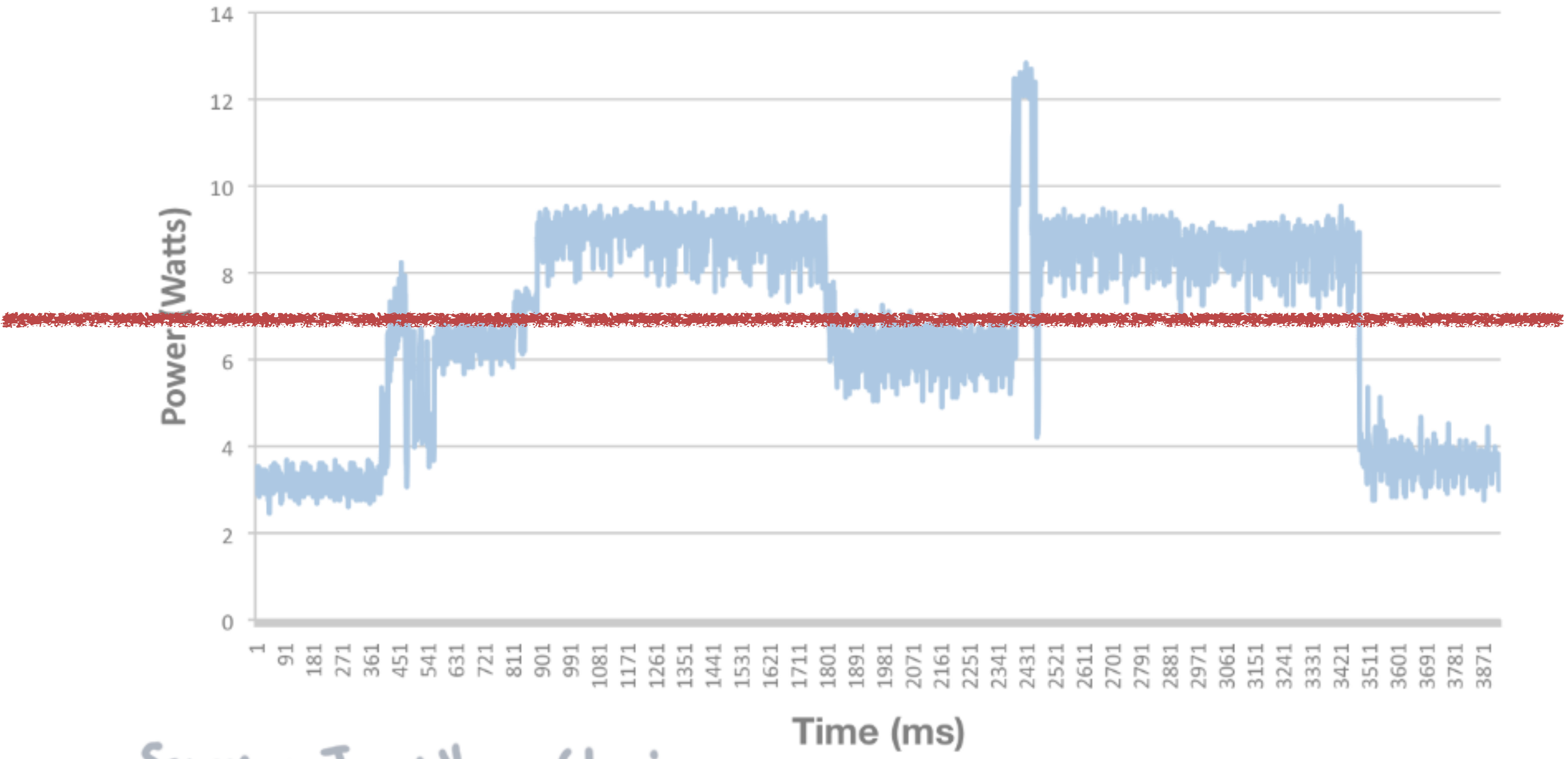
Power = $\frac{\text{Energy}}{\text{Time}}$



Source : Jee Whan Choi

Power Limits

Power ≡ Energy / Time



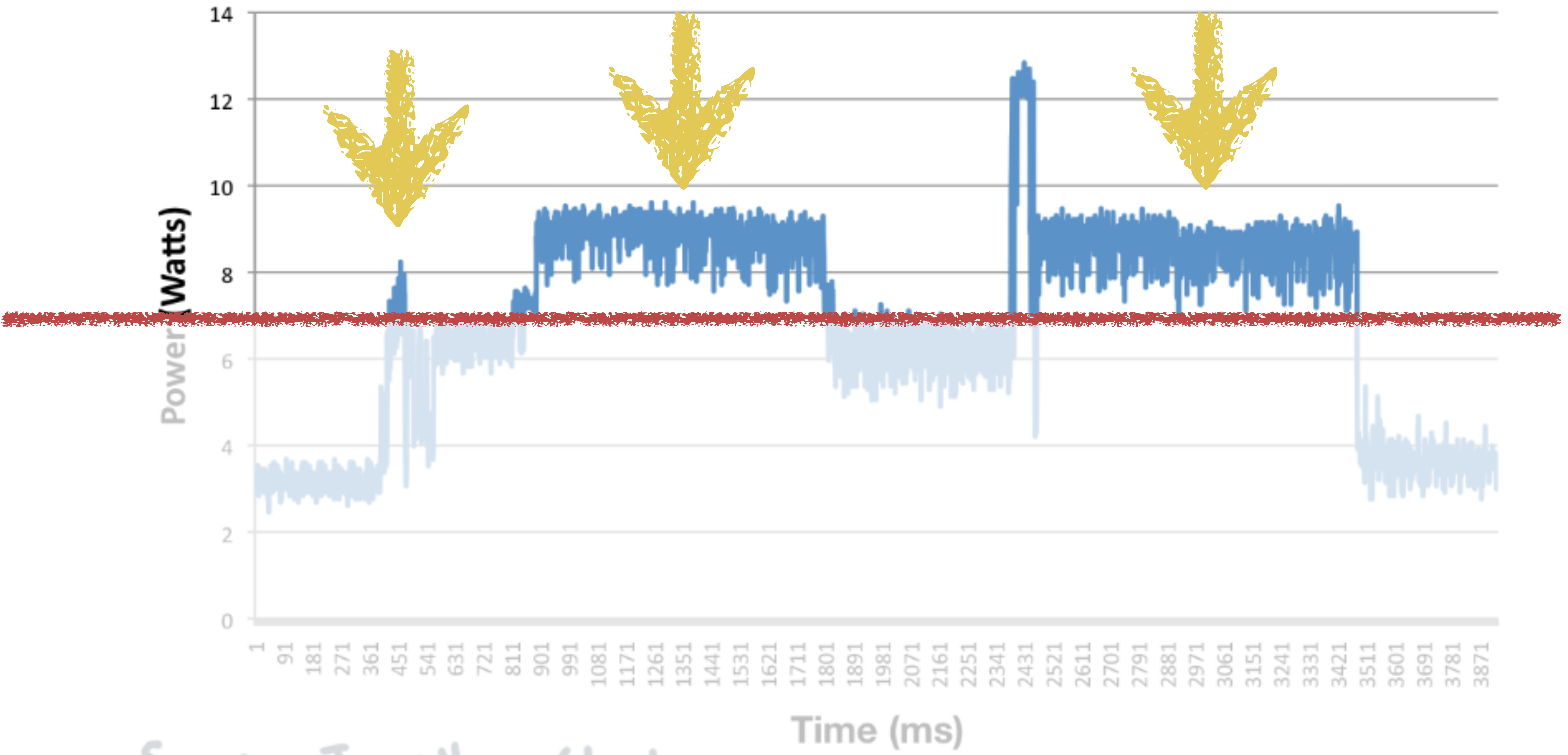
Limit

or "cap", from a user or the system

Source: Jee Whan Choi

Power Limits

Power = Energy / Time



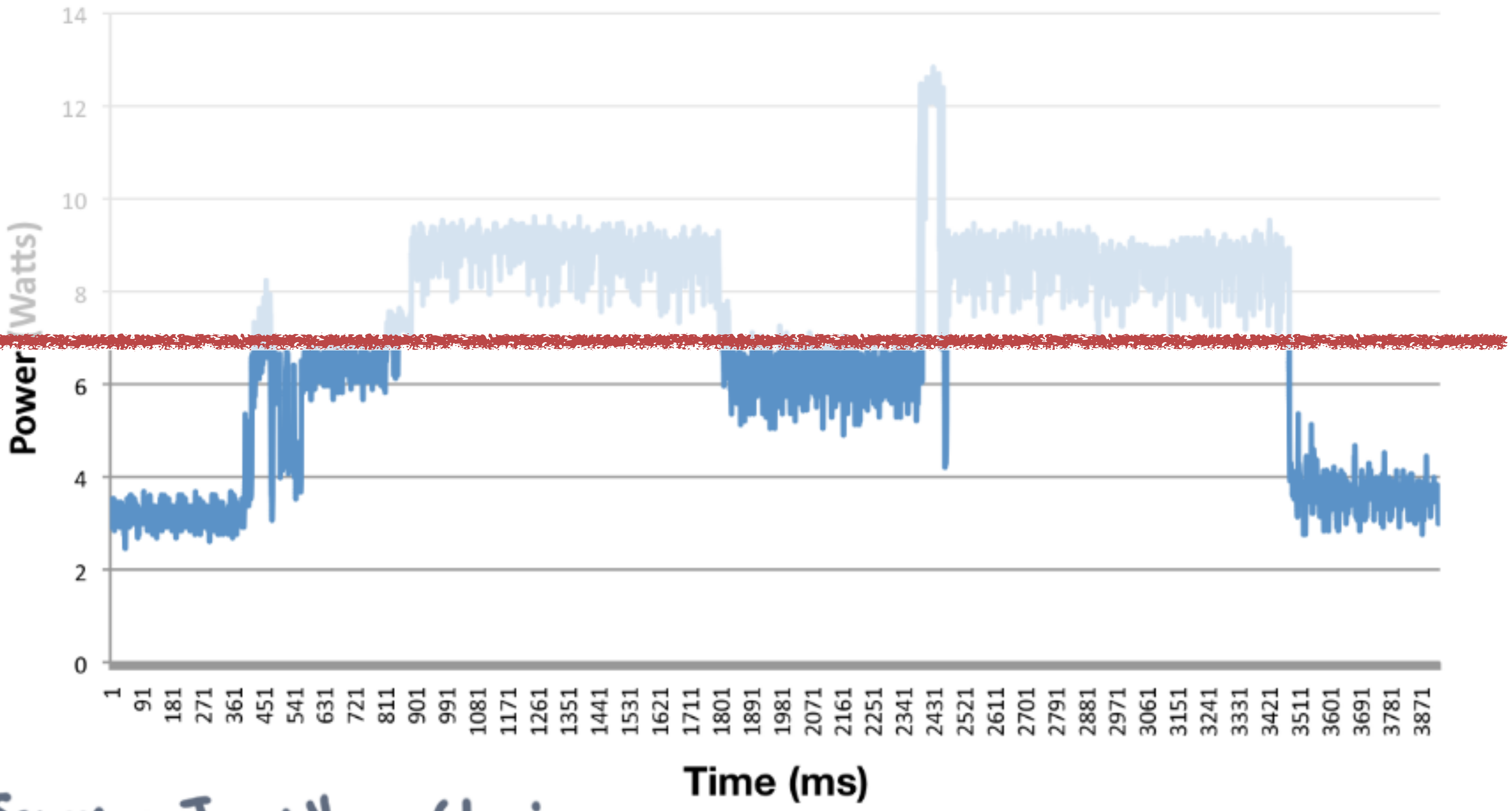
Limit

or "cap", from a user or the system

Source: Jee Whan Choi

Power Limits

Power ≡ Energy / Time



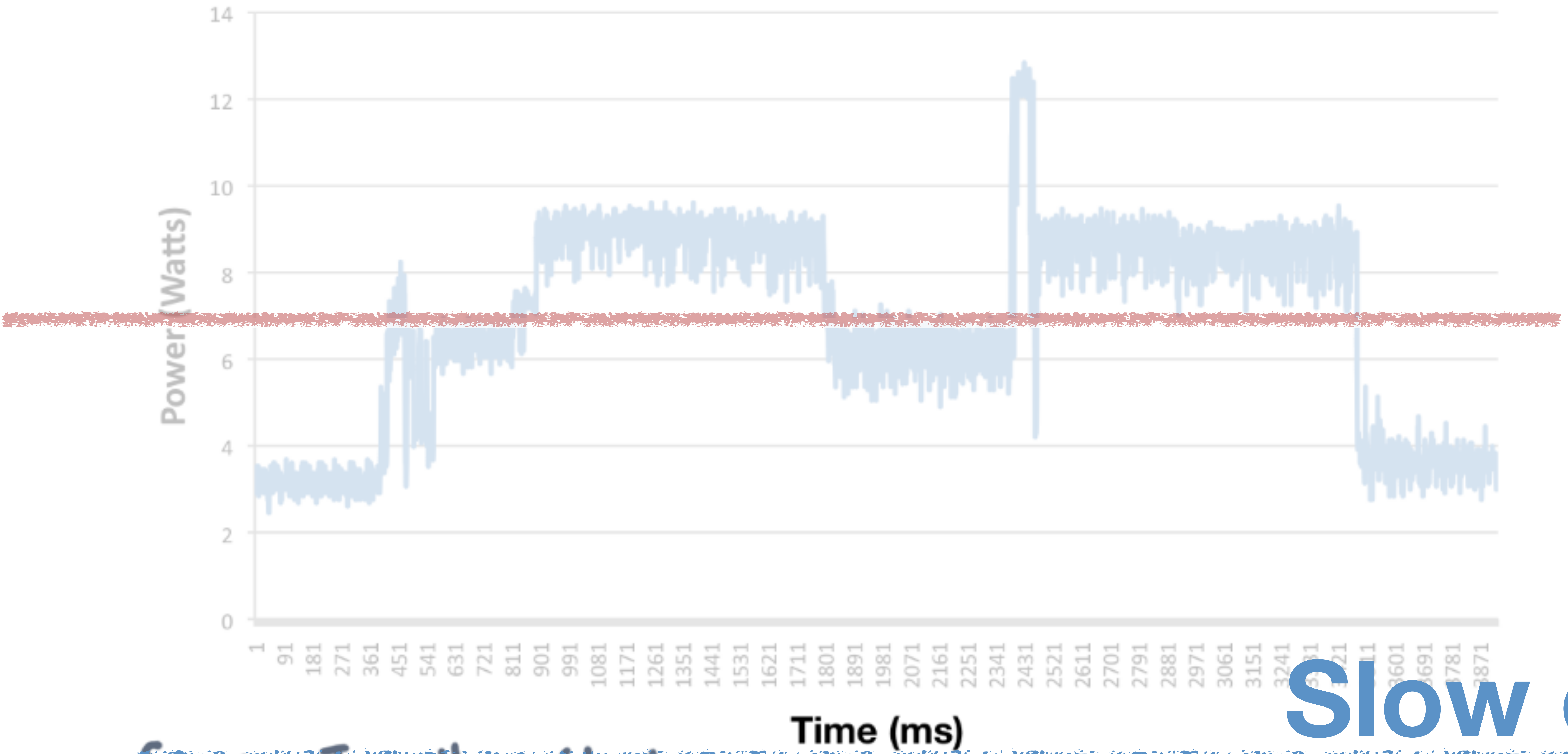
Limit

or "cap", from a user or the system

Source: Jee Whan Choi

Power Limits

$$\text{Power} \equiv \frac{\text{Energy}}{\text{Time}}$$



Limit

or “cap”, from a user or the system

Slow down?

Main question of this talk:

Can you **design an algorithm** in a way that you can **control its power**?

*We are interested in "algorithmic" methods that **complement** techniques available in hardware, like DVFS, and systems software or middleware.*

A first principle:

Relationships among **time**, **energy**, and **power**.

*J. Choi, D. Bedard, R. Fowler, R. Vuduc. "**A roofline model of energy**." In IPDPS'13.*
J. Choi, M. Dukhan, X. Liu, R. Vuduc. "Algorithmic time, energy, and power on candidate HPC building blocks." In IPDPS'14.

Time ~ ?

Energy ~ ?

Power = Energy / Time

Time ~ (# of operations) / (number of processors)

Energy ~ (# of operations)

Power = Energy / Time ~ (number of processors) = Speedup

Time ~ (# of operations) / (number of processors)

Energy ~ (# of operations)

Power = Energy / Time ~ (number of processors) = Speedup

Time ~ (# of operations) / (number of processors)

Energy ~ (# of operations)

Power = Energy / Time ~ (number of processors) = Speedup

Time ~ (# of operations) / (number of processors)

Energy ~ (# of operations)

Power = Energy / Time ~ (number of processors) = Speedup

Time ~ (# of operations) / (number of processors)

Energy ~ (# of operations)

Power = Energy / Time ~ (number of processors) = Speedup

Conclusion:

To save time & energy: *Must reduce work (# ops or cost/op)*

To save power: *Must slow down (e.g., use fewer cores)*

Time ~ ***(# of operations)*** */ (number of processors)*

Energy ~ ***(# of operations)***

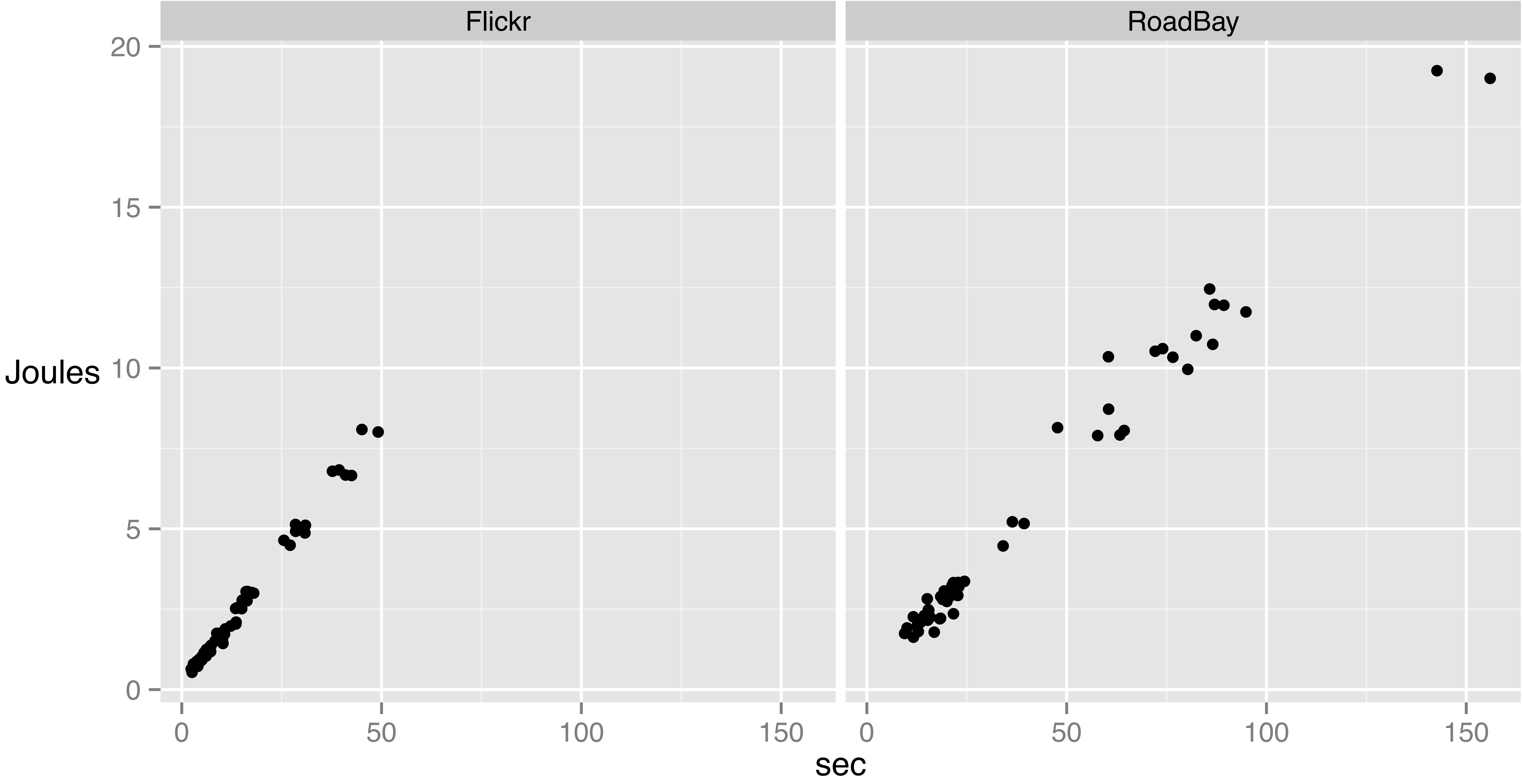
Power = Energy / Time ~ (number of processors) = Speedup

Conclusion:

To save time & energy: *Must* ***reduce work (# ops*** *or* ***cost/op)***

To save power: *Must slow down (e.g., use fewer cores)*

Execution energy is proportional to time (SSSP+GPU example)



Time ~ (# of operations) / (number of processors)

Energy ~ (# of operations)

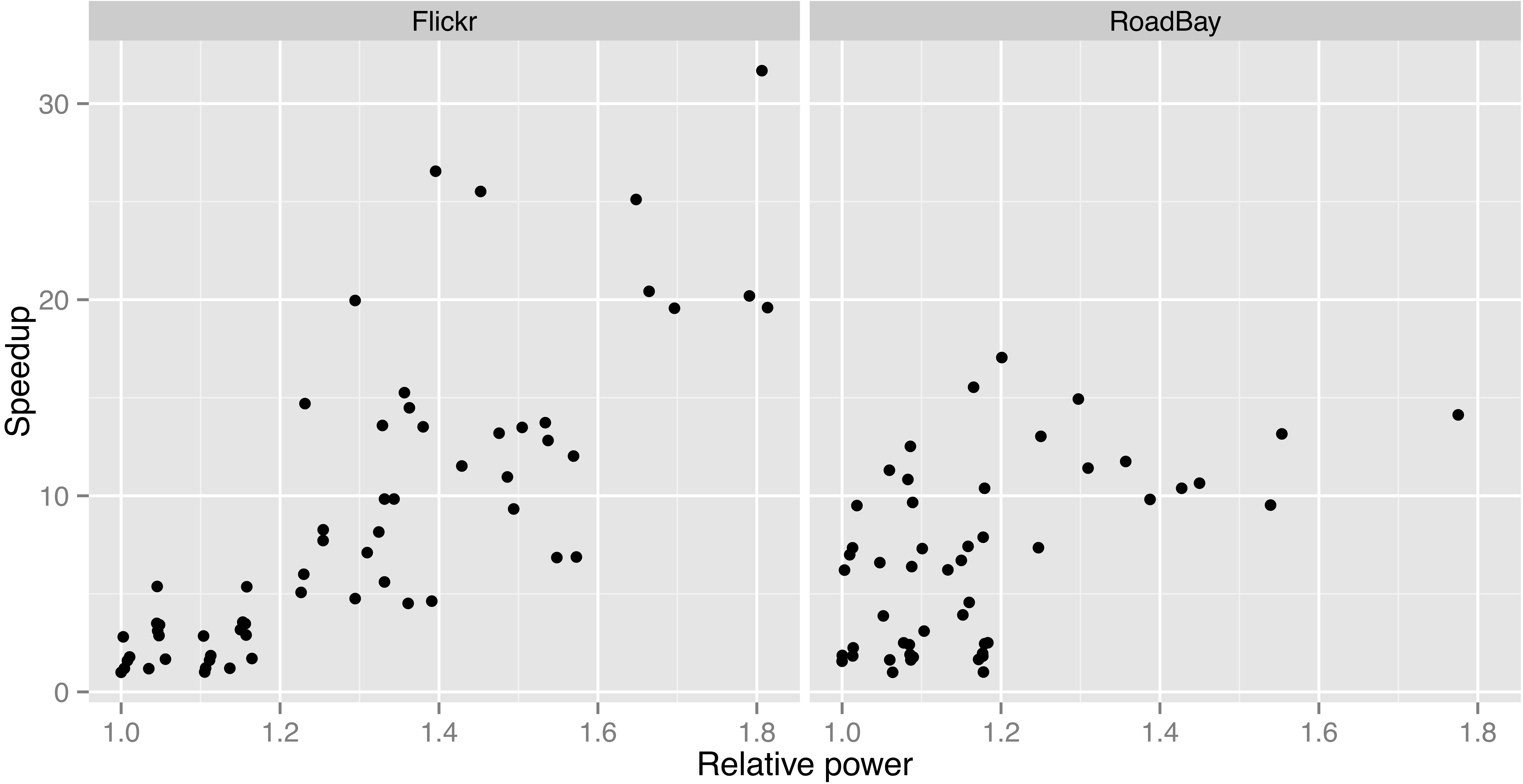
Power = Energy / Time ~ ***(number of processors) = Speedup***

Conclusion:

To save time & energy: *Must reduce work (# ops or cost/op)*

To save power: *Must* ***slow down*** *(e.g., use fewer cores)*

Speedup increases with additional power (SSSP+GPU example)



Main question of this talk:

Can you **design an algorithm** in a way that you can **control its power**?

*We are interested in "algorithmic" methods that **complement** techniques available in hardware, like DVFS, and systems software or middleware.*

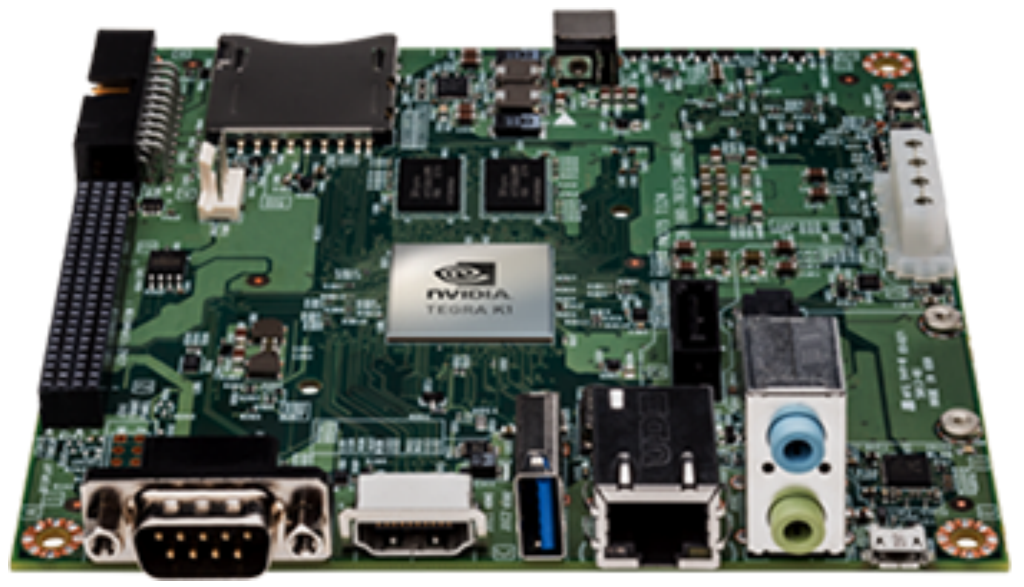
Yes!

Example: A **power-tunable** graph algorithm to compute single-source shortest paths **(SSSP)**.

Sara Karamati (Ph.D. student), Dr. Jeff Young (research faculty), R. Vuduc — new, ***unpublished*** *work*

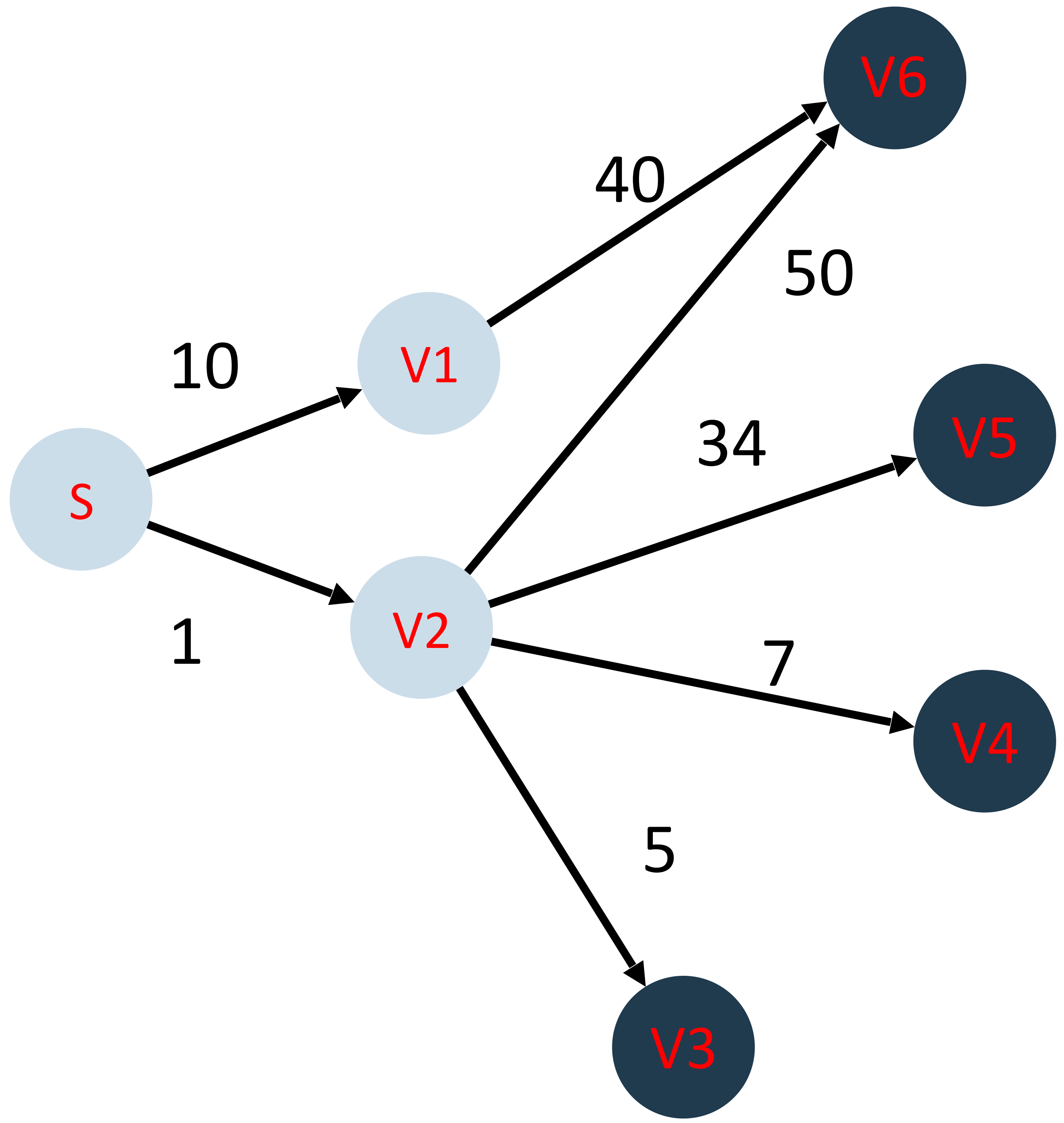
- Baseline: Fastest, work-efficient **“delta-stepping-like”** method*
    - **Tunable work-parallelism tradeoff**
- Tuned for a GPU and run on an **NVIDIA Jetson TK1**, which has tunable core frequencies (**10x**) and memory frequencies (**3x**)
- *No* preprocessing shortcuts, a la PHAST**

	Jetson TK1
CPU	ARM A15 (*32-bit, 2.3 GHz, 4+1 cores*)
GPU	192 core Kepler, 326 GF/s (*peak*)
Memory	2 GB LPDDR3
Storage	16 GB eMMC
Networking	Ethernet
Form Factor	Dev board
I/O	USB, HDMI, Serial
Release Date	2014

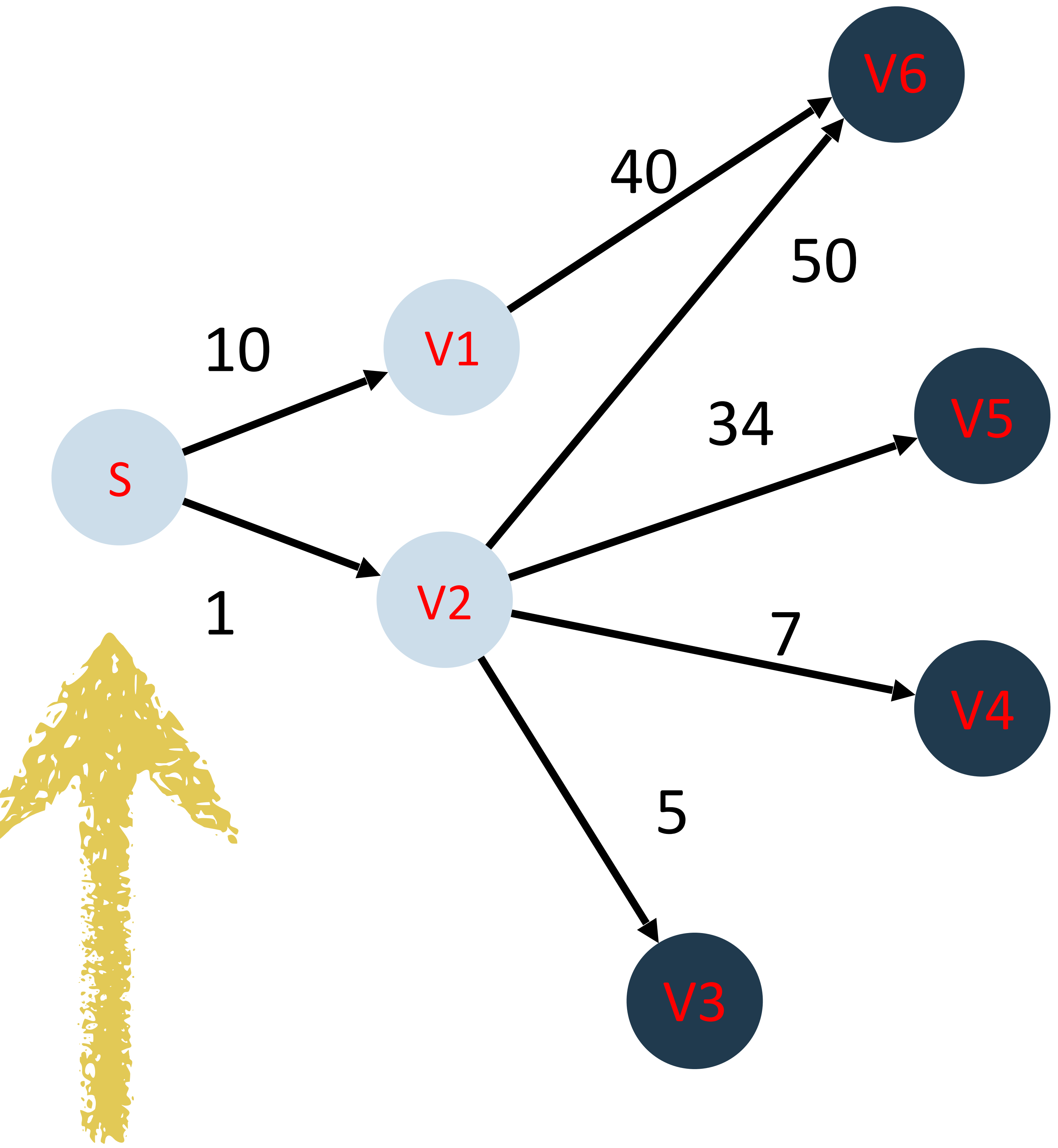


** Based on GunRock implementations of Davidson, Baxter, Garland, and Owens (IPDPS’14)*
*** Delling et al. “PHAST: Hardware-accelerated shortest path trees” (JPDC’10)*

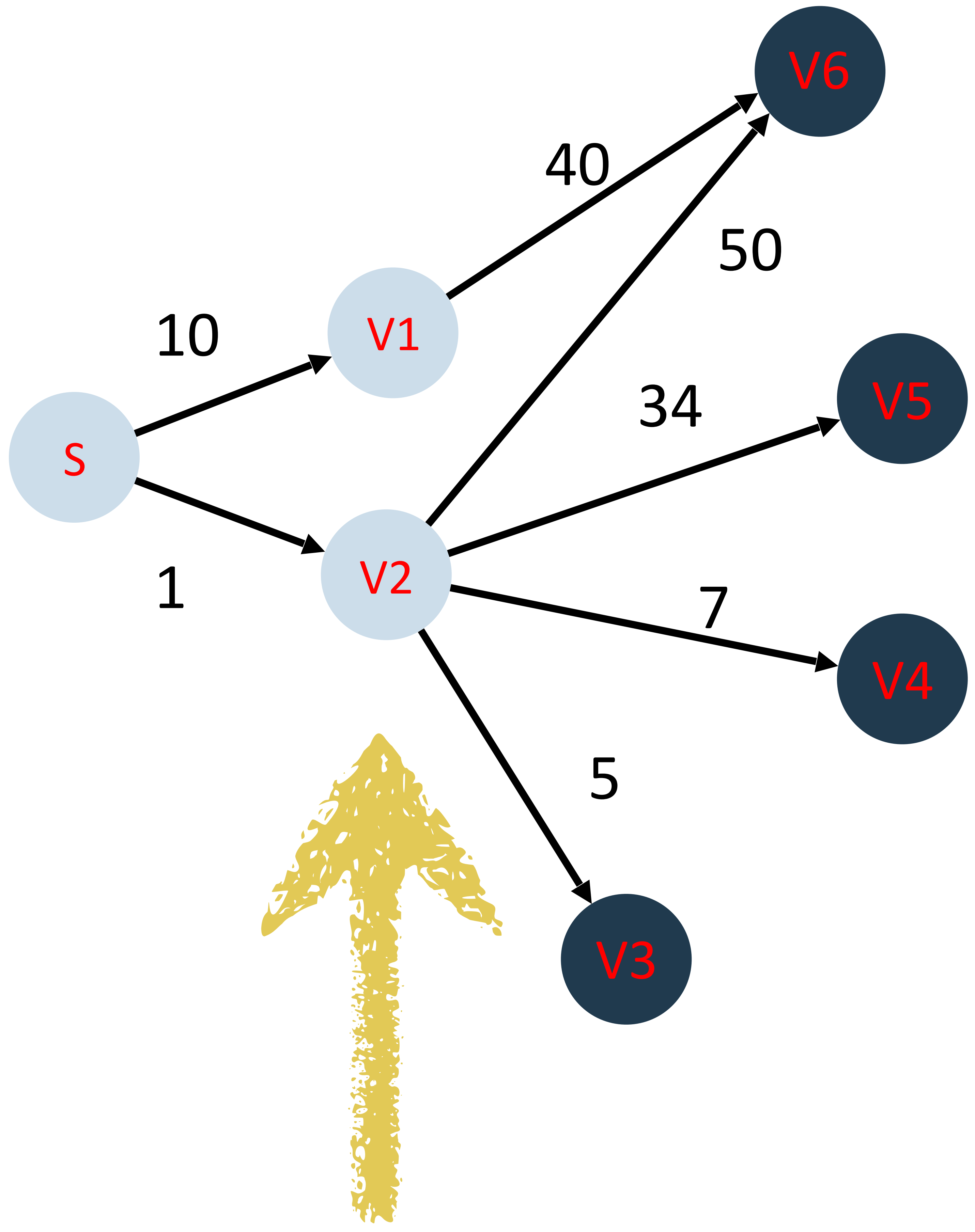
Baseline: Gunrock's "**Near+Far**" algorithm



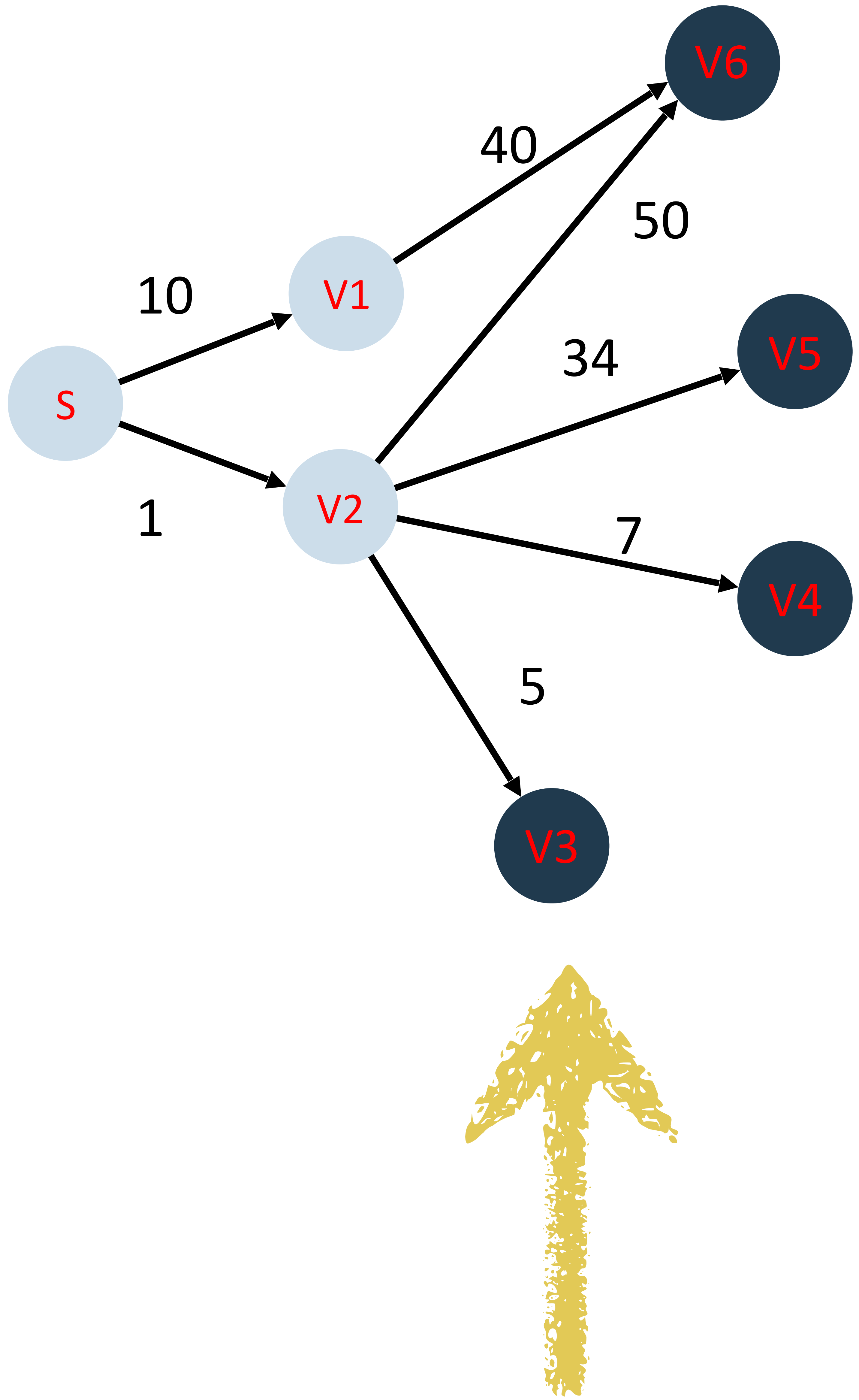
Baseline: Gunrock's "**Near+Far**" algorithm



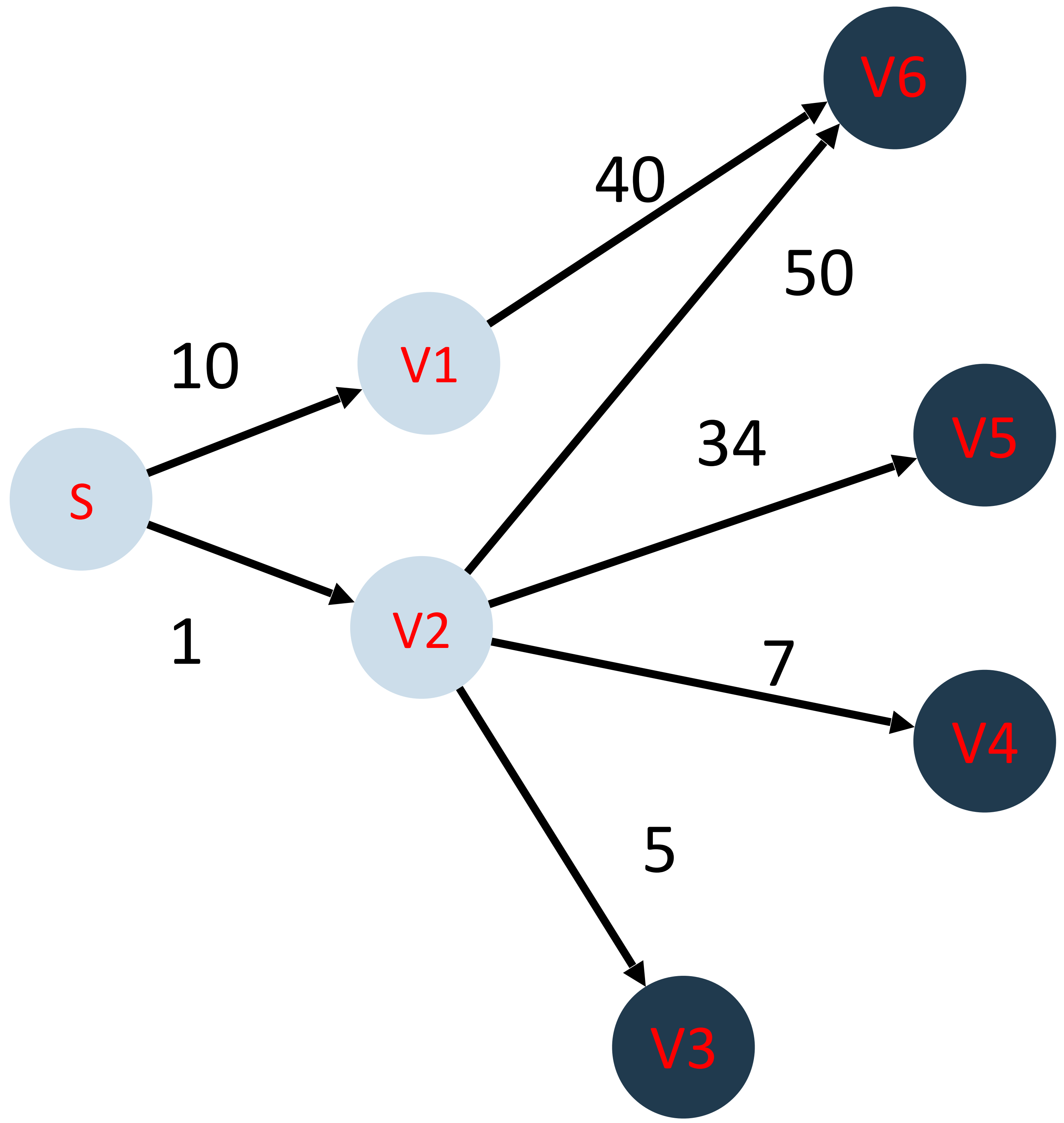
Baseline: Gunrock's "**Near+Far**" algorithm



Baseline: Gunrock's "**Near+Far**" algorithm



Baseline: Gunrock's "**Near+Far**" algorithm



Baseline: Gunrock's "**Near+Far**" algorithm

Baseline: Gunrock's "**Near+Far**" algorithm

Baseline: Gunrock's "**Near+Far**" algorithm

|Frontier| ~ parallelism

Baseline: Gunrock's "**Near+Far**" algorithm

Baseline: Gunrock's "**Near+Far**" algorithm

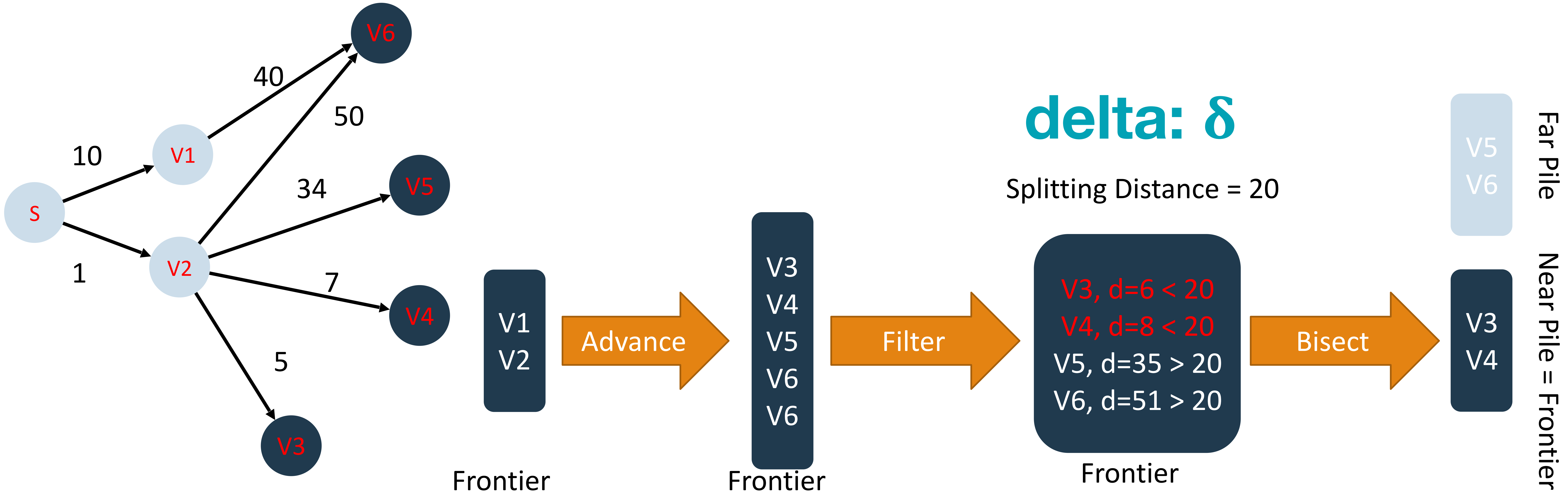
Baseline: Gunrock's "**Near+Far**" algorithm

Baseline: Gunrock's "**Near+Far**" algorithm

Baseline: Gunrock's "**Near+Far**" algorithm

Baseline: Gunrock's "**Near+Far**" algorithm

Baseline: Gunrock's "**Near+Far**" algorithm



Baseline: Gunrock's "**Near+Far**" algorithm

Power ~ Parallelism ~ Queue sizes

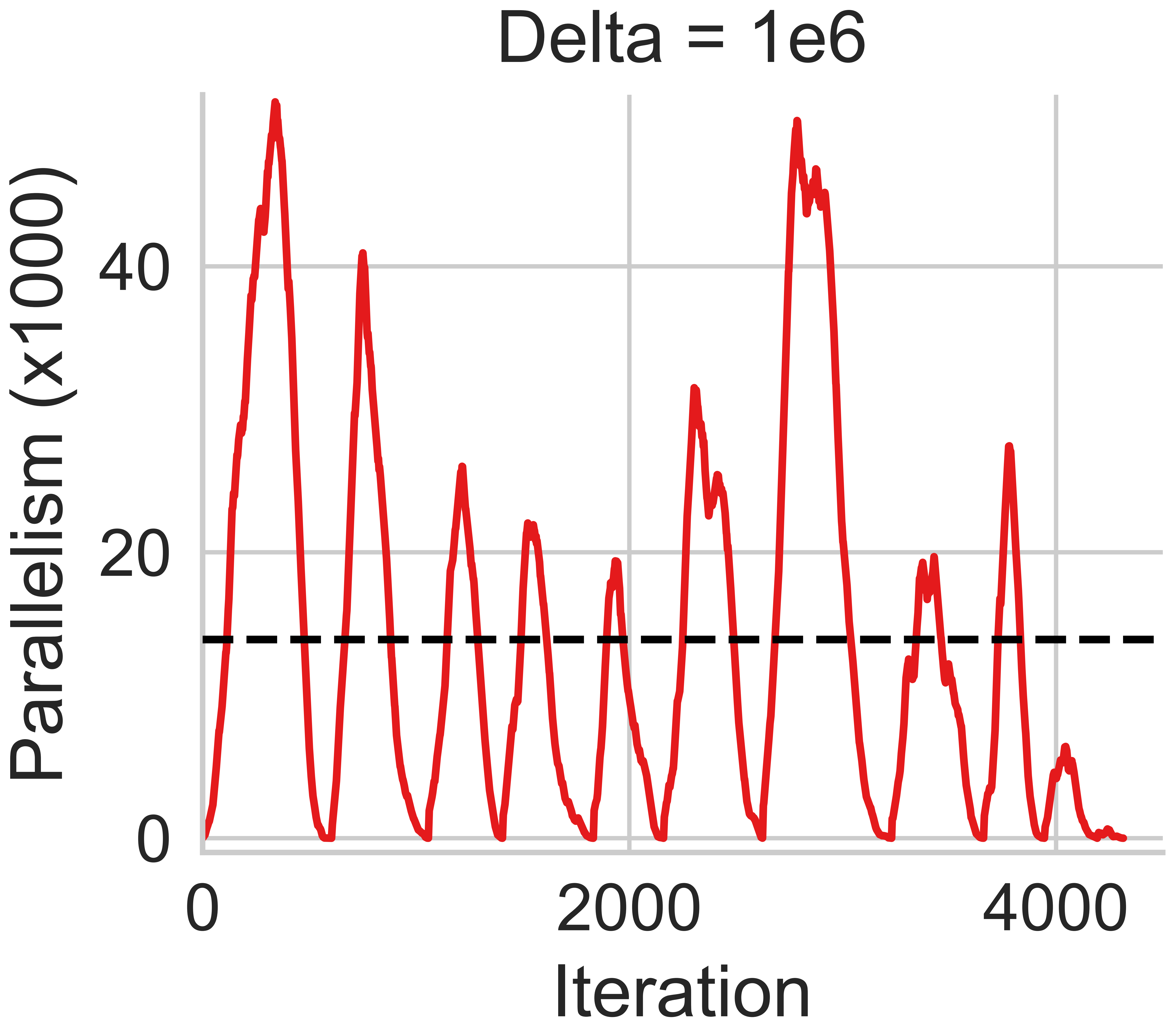
Baseline: Gunrock's "**Near+Far**" algorithm

hpcgarage.org/uccs

Power ~ Parallelism ~ Queue sizes

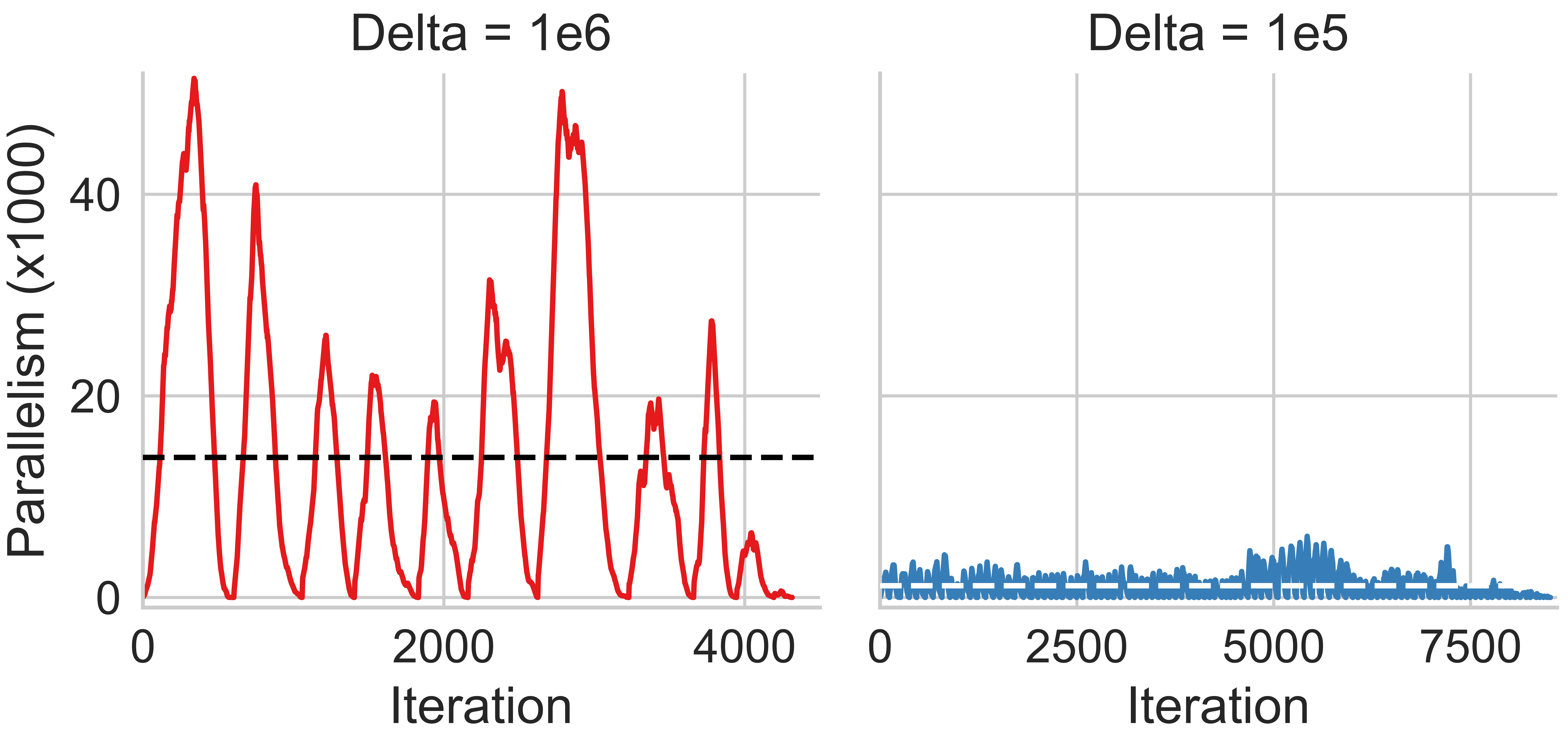
What is the effect of delta (δ)?

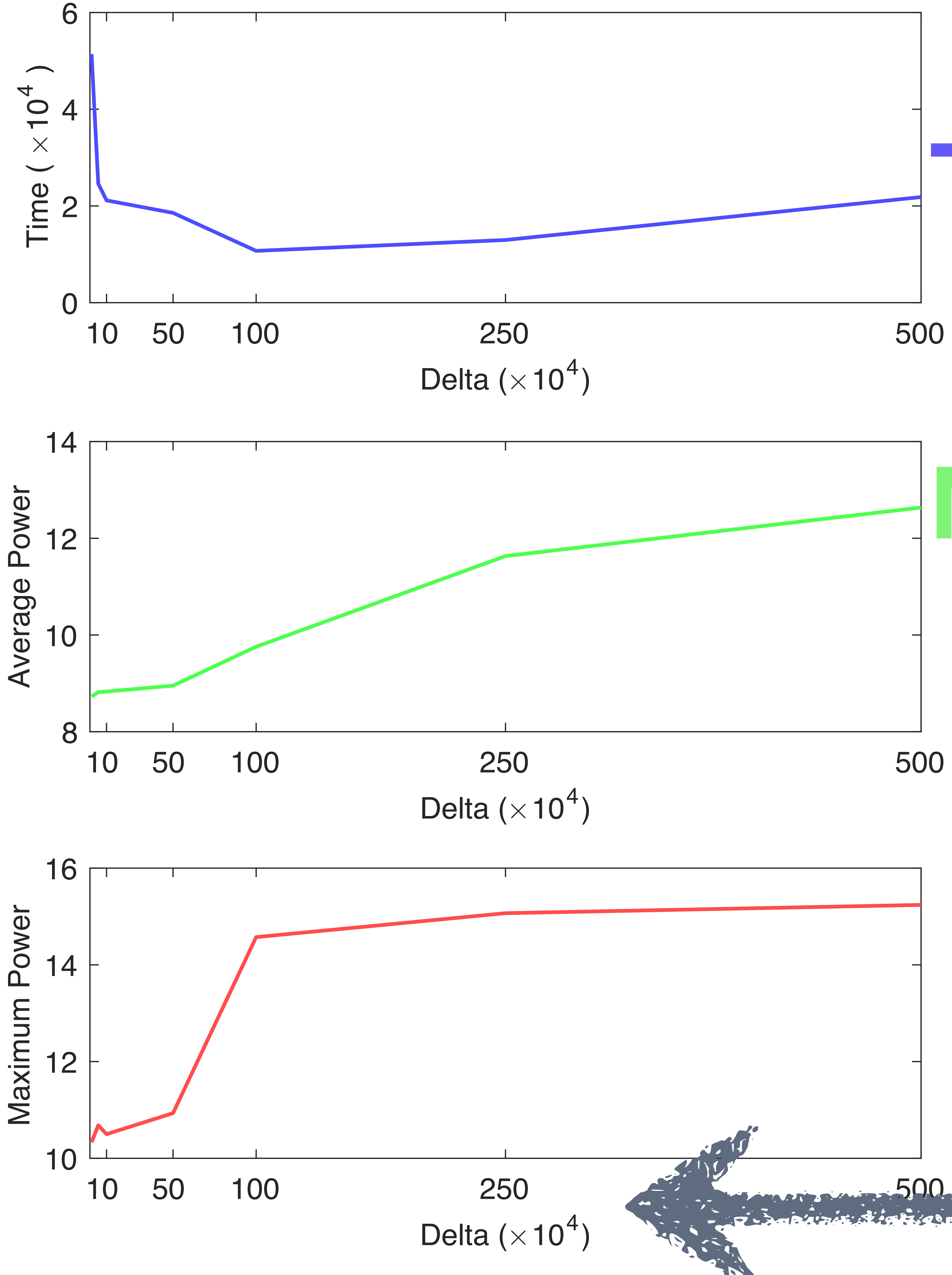
What is the effect of delta (δ)?



(road network)

What is the effect of **delta (δ)**?





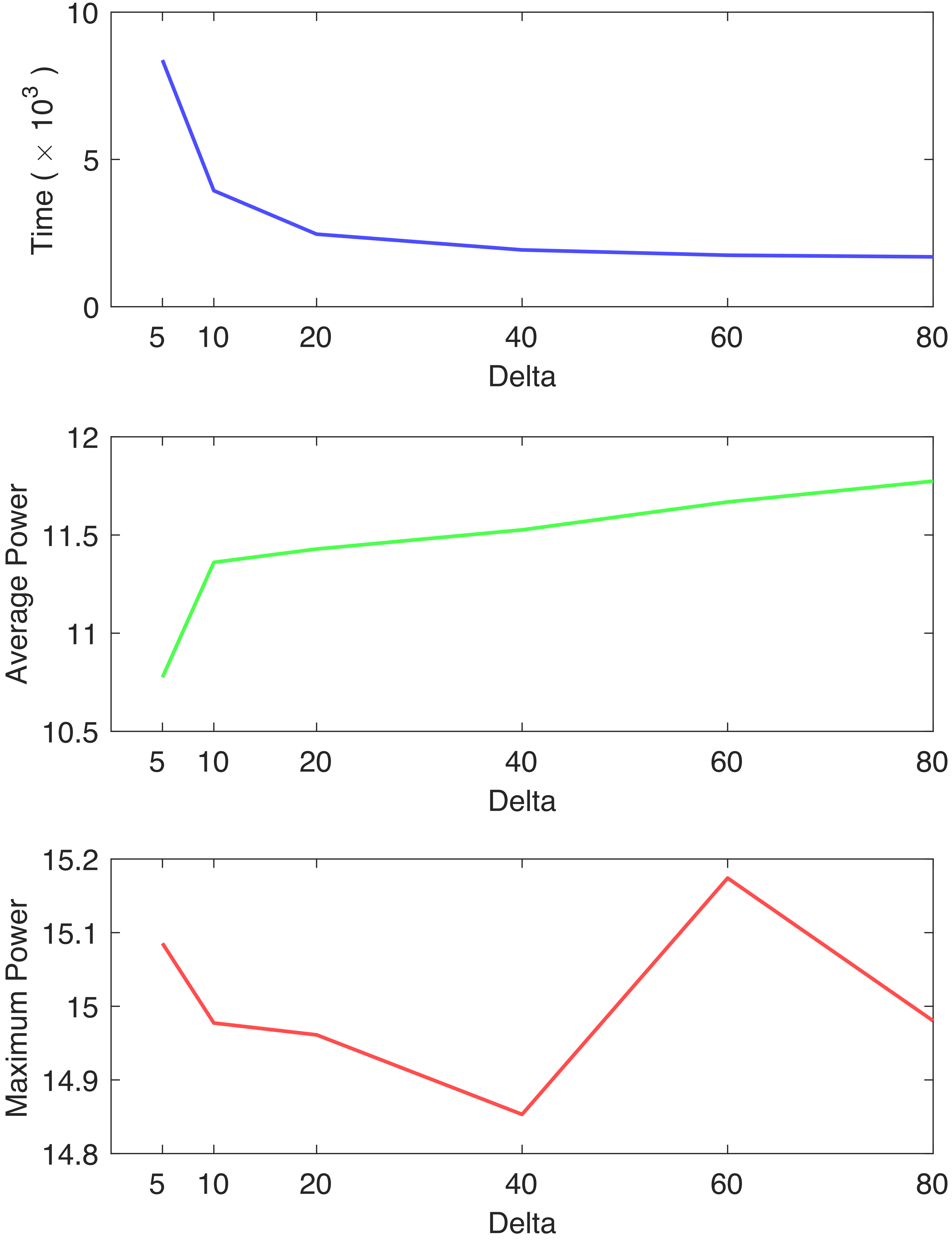
Time

(road network)

Mean power

Max power

delta: δ



Time

(scale-free)

Mean power

Max power

Observation:

Delta (δ) is a **tuning parameter** that can be used to control power-time tradeoffs.

But **how** *to choose it? It is input-dependent. And, in the literature, it is always treated as a fixed* a priori *parameter with little guidance on its ideal value.*

Sara's insight:

Treat δ as a parameter to be **learned** and **controlled**, **dynamically**.

Recall: Near+Far == stages.

Recall: Near+Far == stages.

Add a controller!

Simple models between stages…

$$X^2(k) \sim \textbf{(degree)} * X^1(k)$$

$$\hat{X}_k^{(2)} = d \cdot X_k^{(1)}$$

Estimator

$$\hat{X}_k^{(2)} = d \cdot X_k^{(1)}$$

Estimator

$$\hat{X}_k^{(2)} = d \cdot X_k^{(1)}$$

Parameter

Estimator

$$\hat{X}_k^{(2)} = d \cdot X_k^{(1)}$$

Parameter

Loss

$$\min_d \sum_k \left(X_k^{(2)} - \hat{X}_k^{(2)} \right)^2$$

Estimator

$$\hat{X}_k^{(2)} = d \cdot X_k^{(1)}$$

Parameter

Loss

$$\min_d \sum_k \left(X_k^{(2)} - \hat{X}_k^{(2)} \right)^2$$

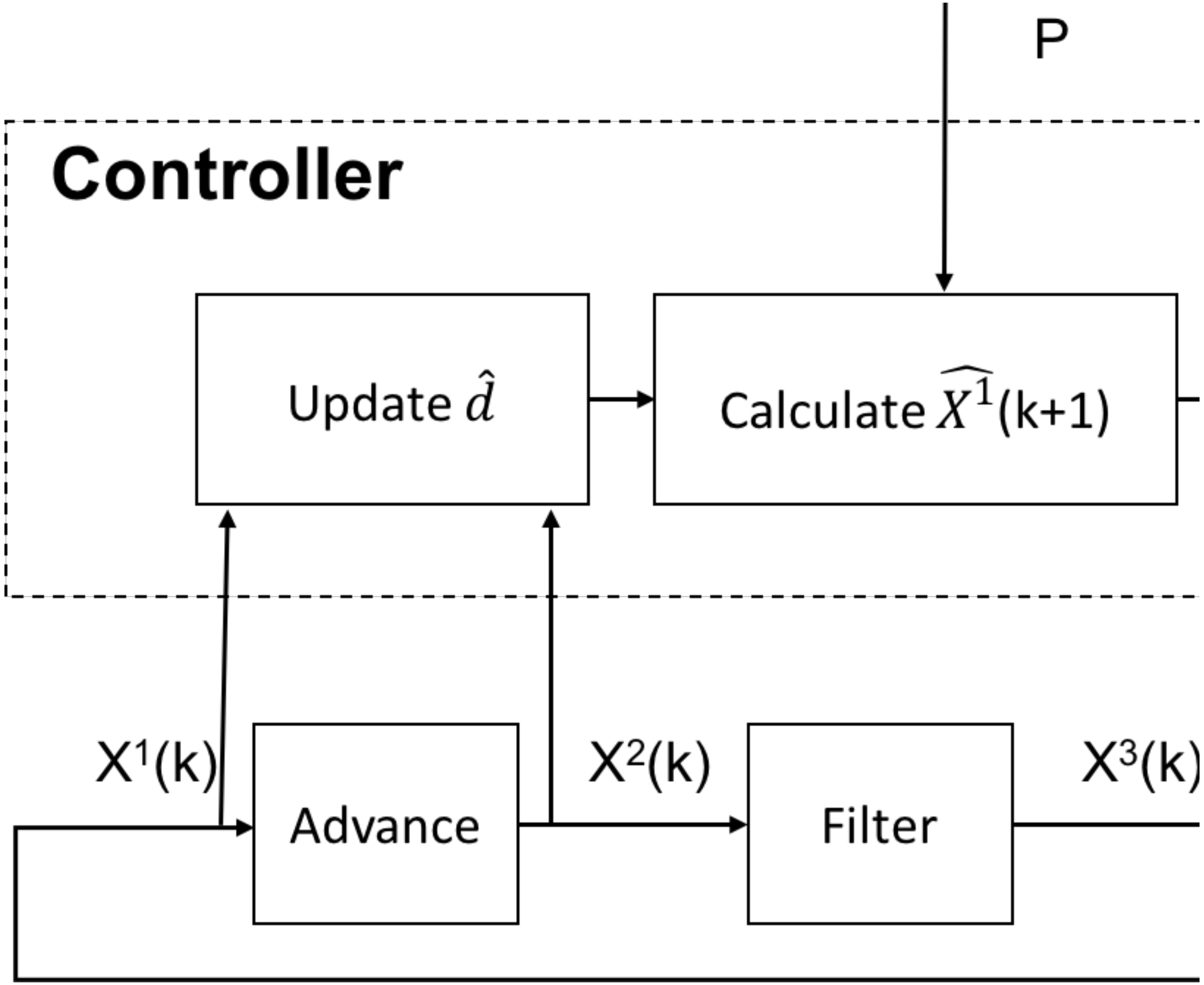
$$= \min_d \sum_k \left(X_k^{(2)} - d \cdot X_k^{(1)} \right)^2$$

Loss

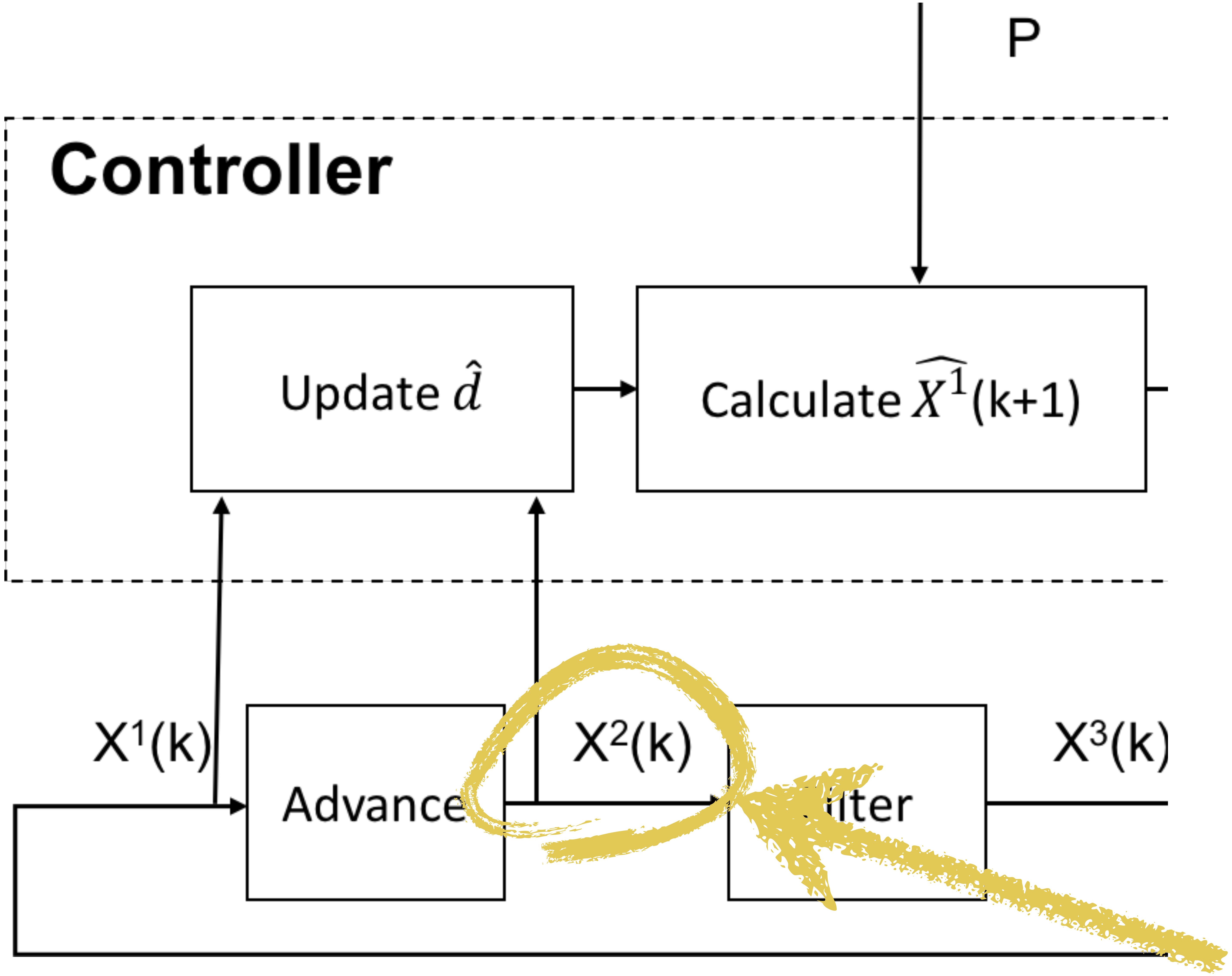
$$\min_{d} \sum_{k} \left(X_k^{(2)} - d \cdot X_k^{(1)} \right)^2 - \lambda \ln d$$

Regularize to stabilize the estimator during the early iterations and use online fitting method (e.g., stochastic gradient descent)

Constrain parallelism…

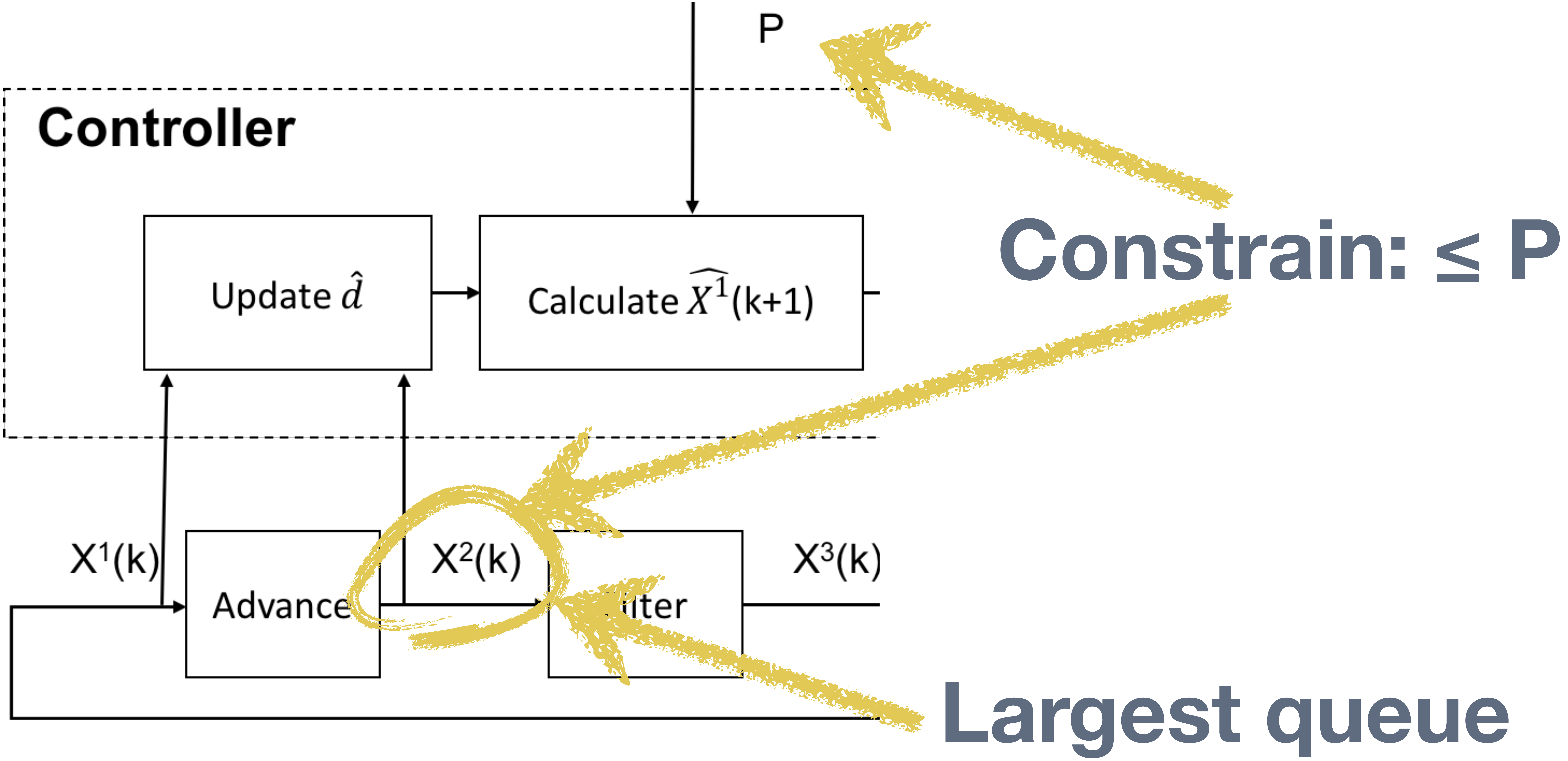


Constrain parallelism…

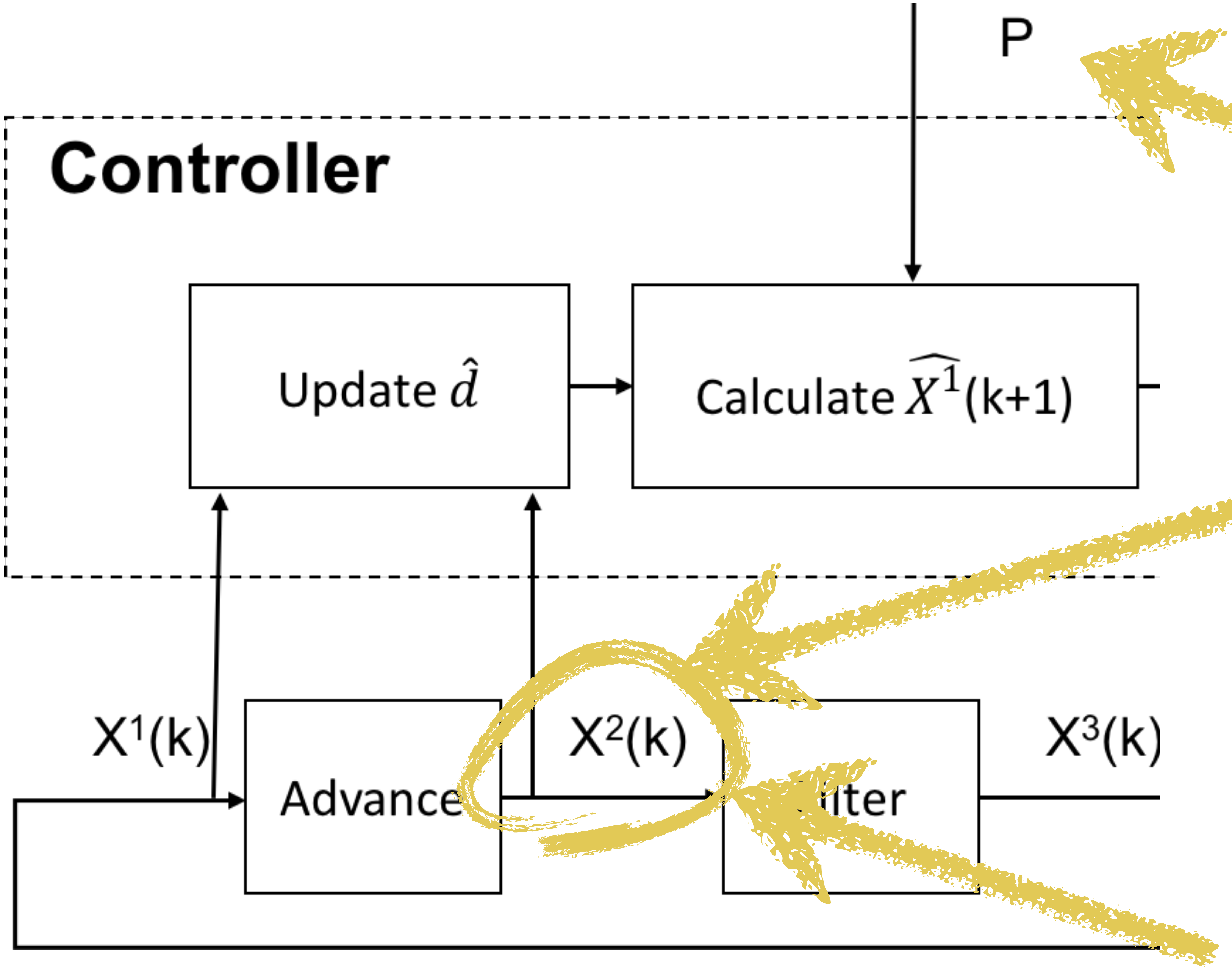


Largest queue

Constrain parallelism…



Constrain parallelism…



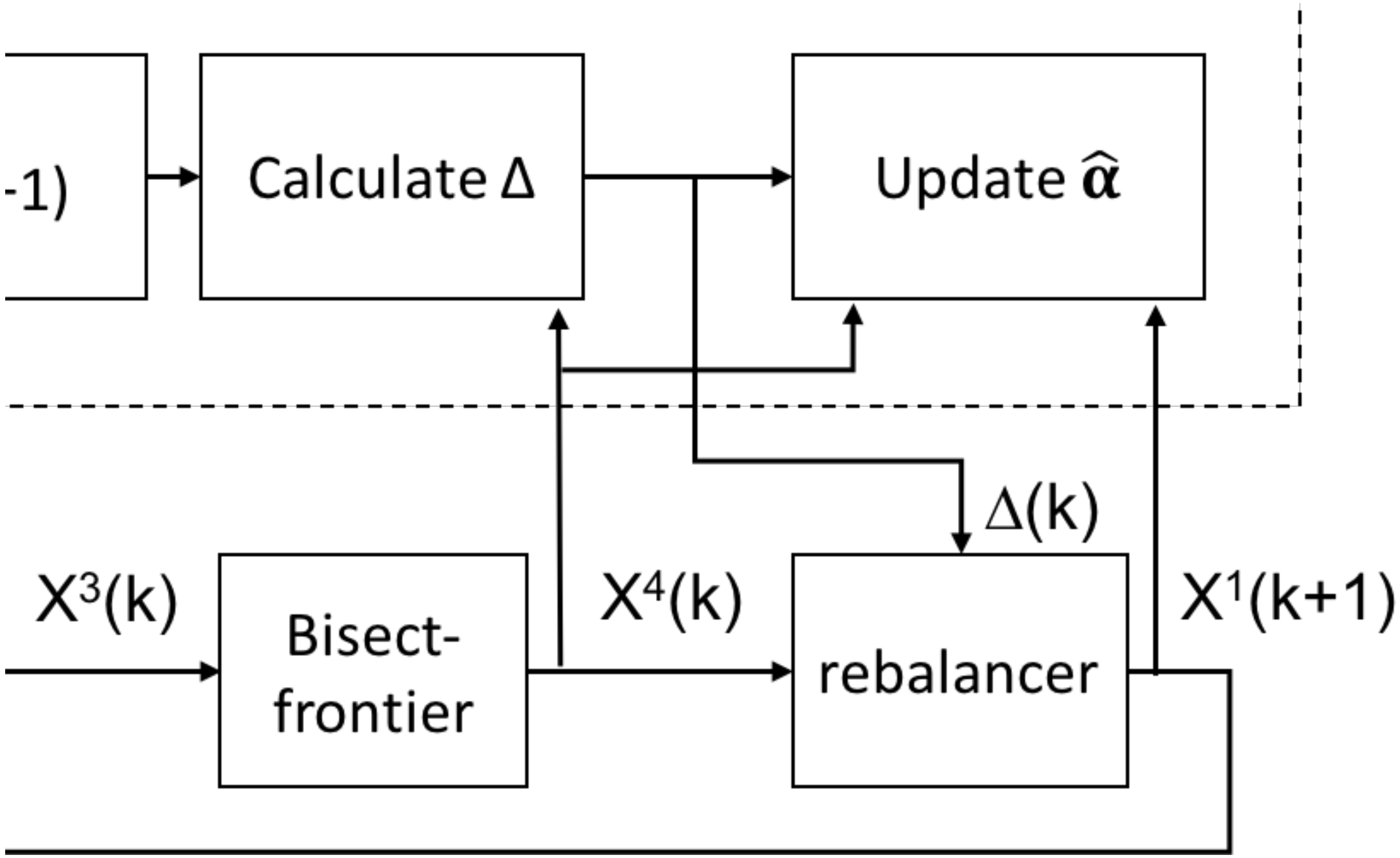
Constrain: ≤ P

$$\hat{X}^{(1)}_{k+1} \leq \frac{P}{d}$$

Largest queue

Estimate the effect of a *change* in **δ**

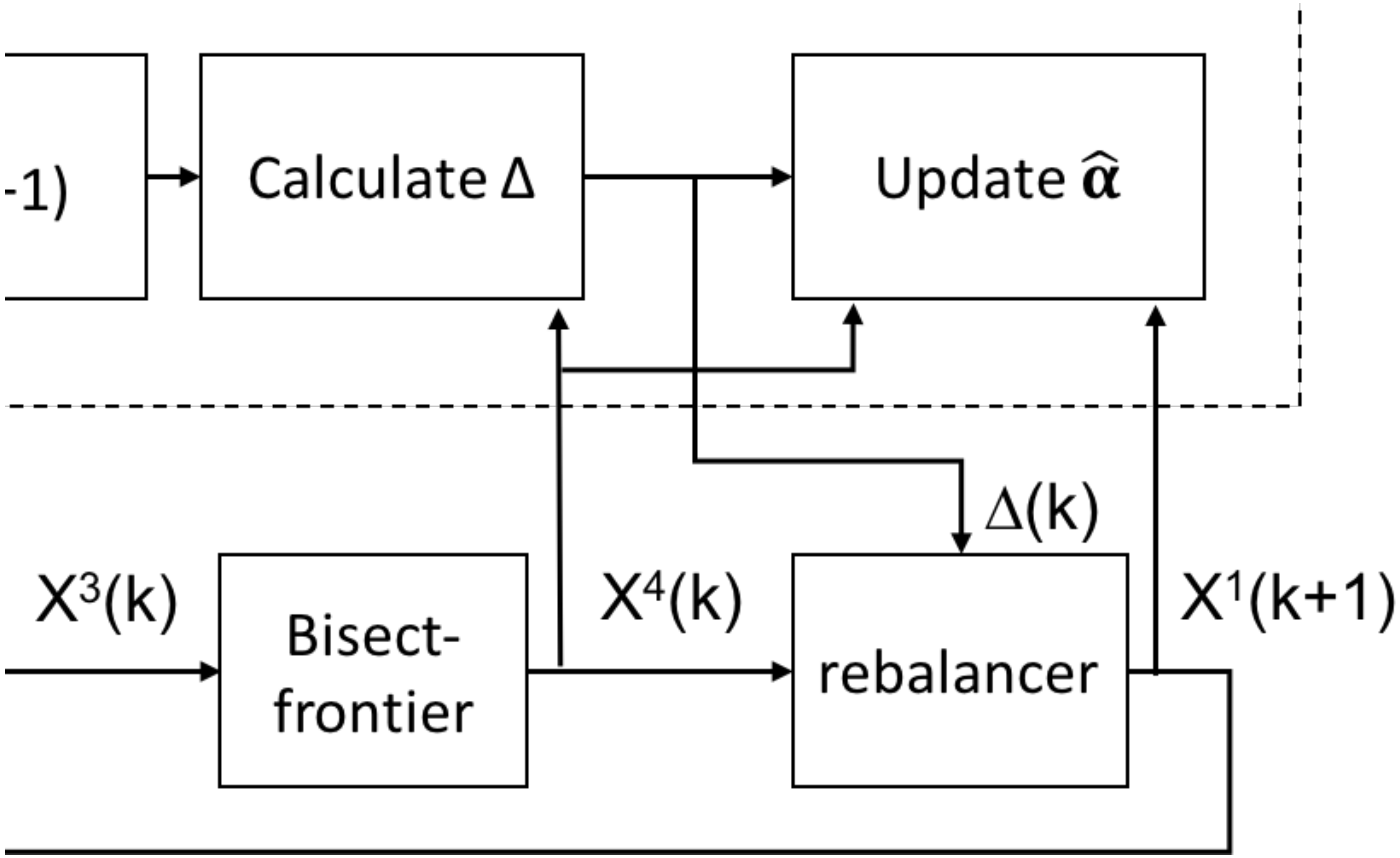
$$\hat{X}_{k+1}^{(1)} = X_k^{(4)} + \alpha \cdot \Delta\delta_k$$



Estimate the effect of a *change* in **δ**

$$\hat{X}^{(1)}_{k+1} = X^{(4)}_{k} + \alpha \cdot \Delta\delta_k$$

Estimator

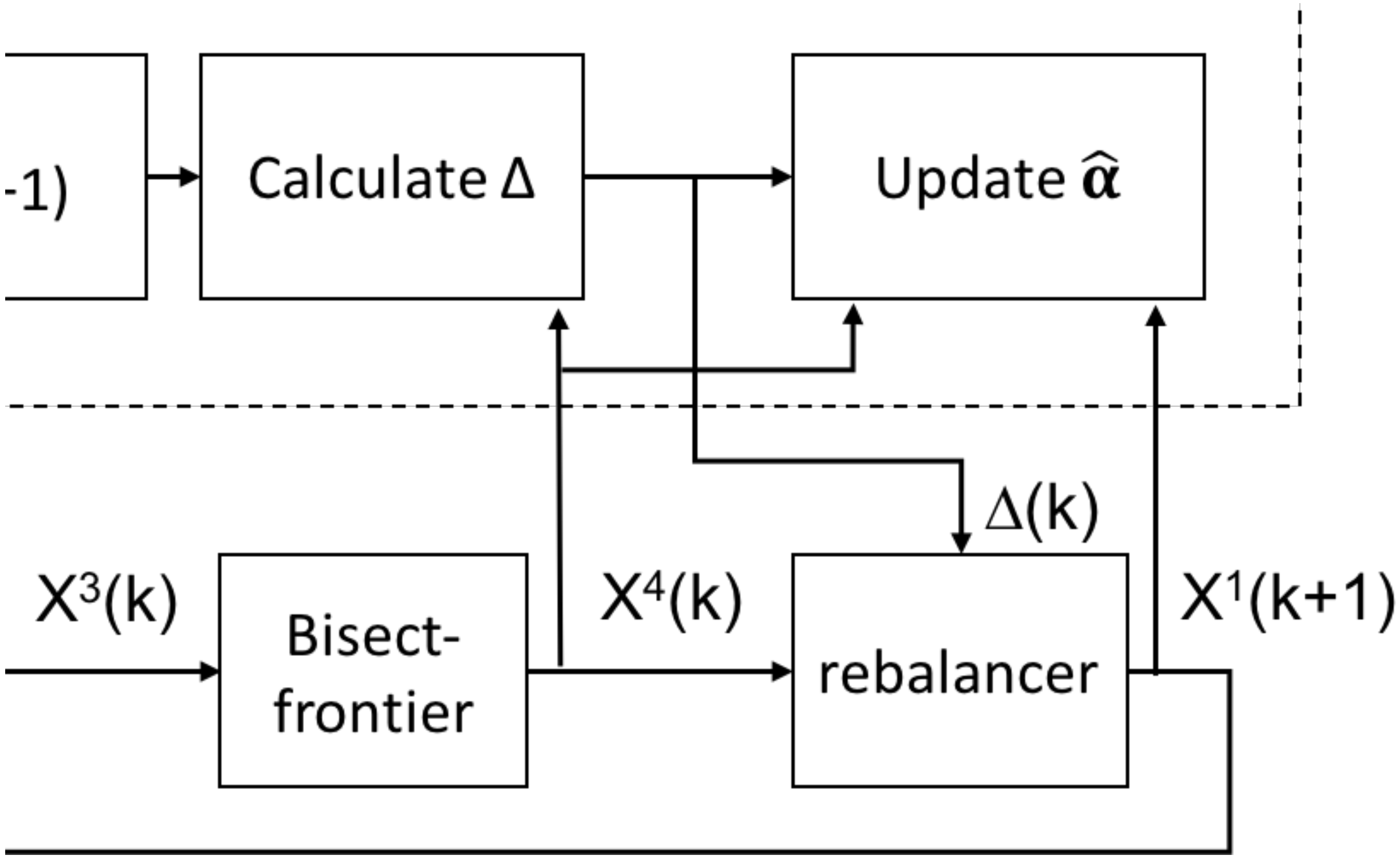


Estimate the effect of a *change* in **δ**

$$\hat{X}^{(1)}_{k+1} = X^{(4)}_{k} + \alpha \cdot \Delta\delta_k$$

Estimator

Parameter



Cool feature:

User sets ***P***, the max parallelism, not **δ**!

The controller chooses the hard-to-set **δ** *fully automatically, adjusting it as frequently as* **every iteration***.* Not discussed: *One can prove that the controller is bounded-input bounded-output (BIBO) stable, meaning frontier sizes will be "well-behaved."*

Next question:

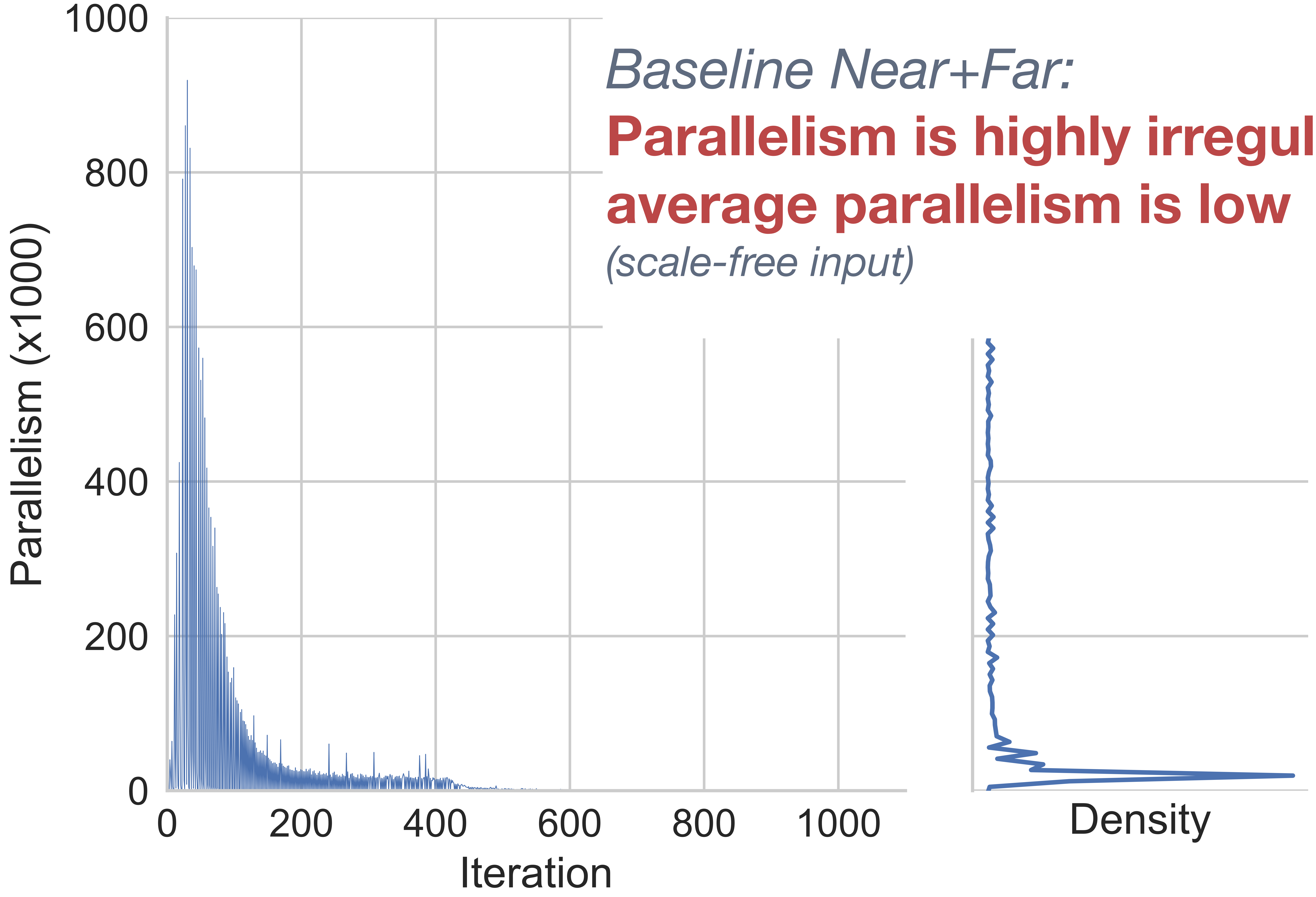
Does it work?

(Experimental results)

Baseline Near+Far:

Parallelism is highly irregular, average parallelism is low

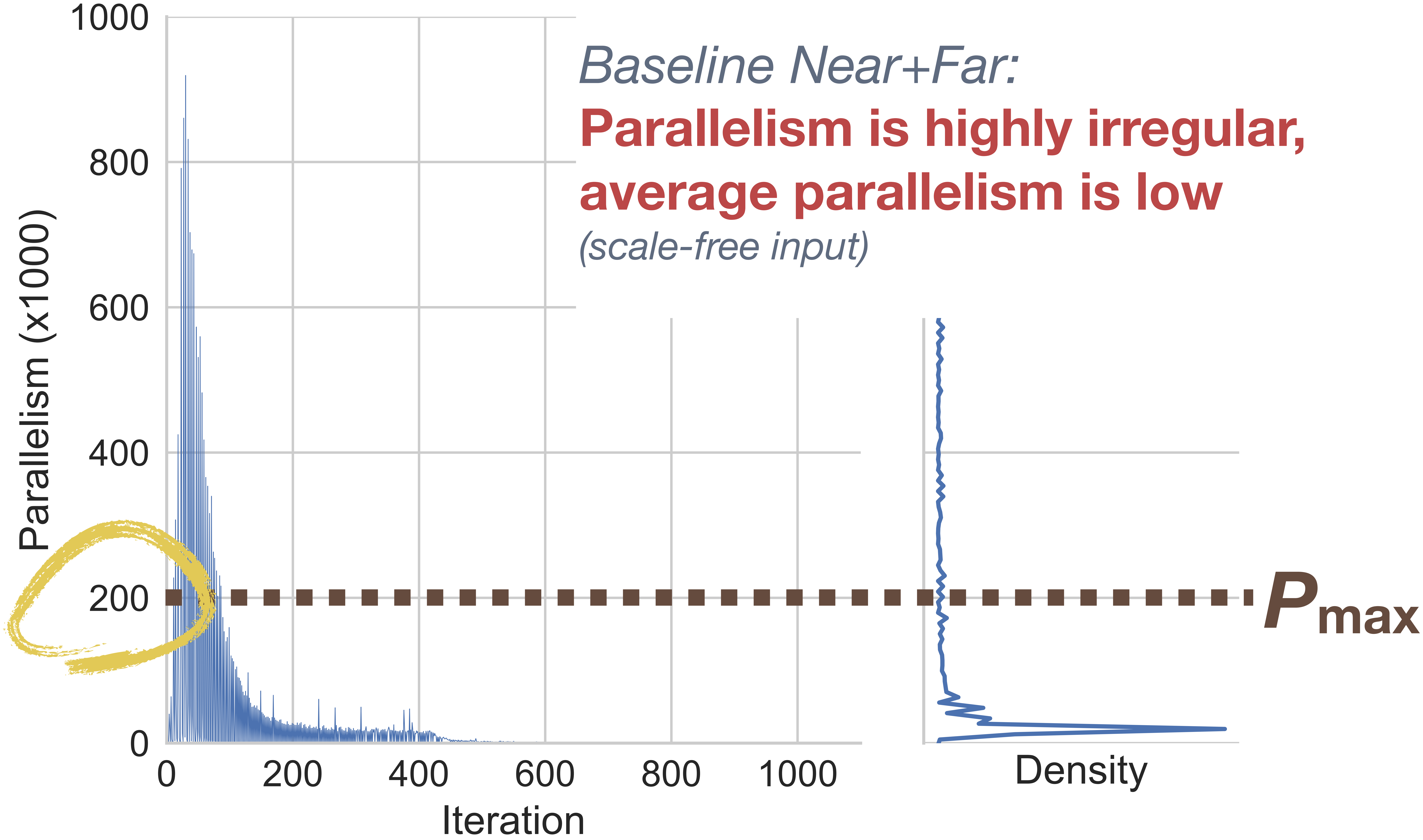
(scale-free input)

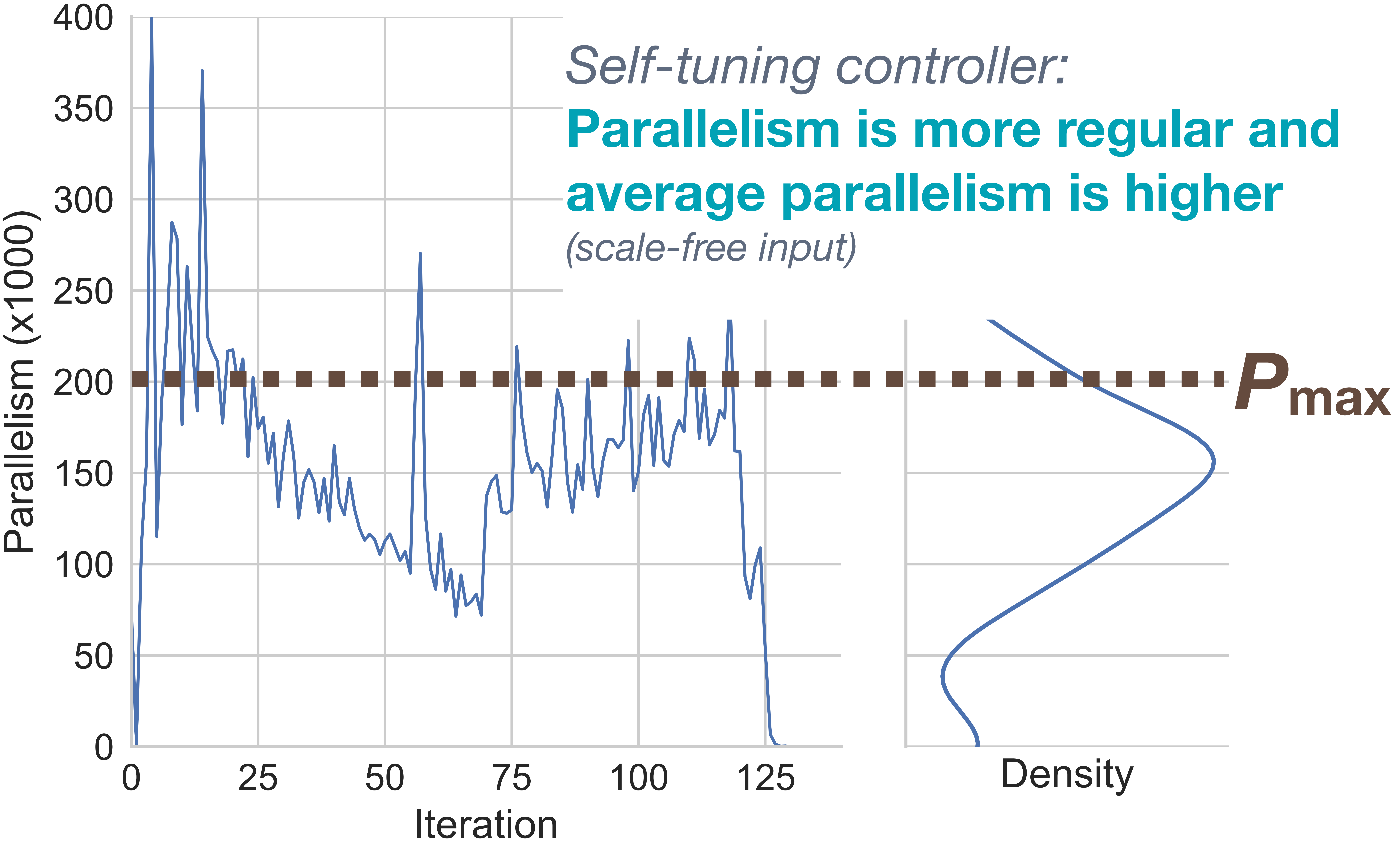


Baseline Near+Far:

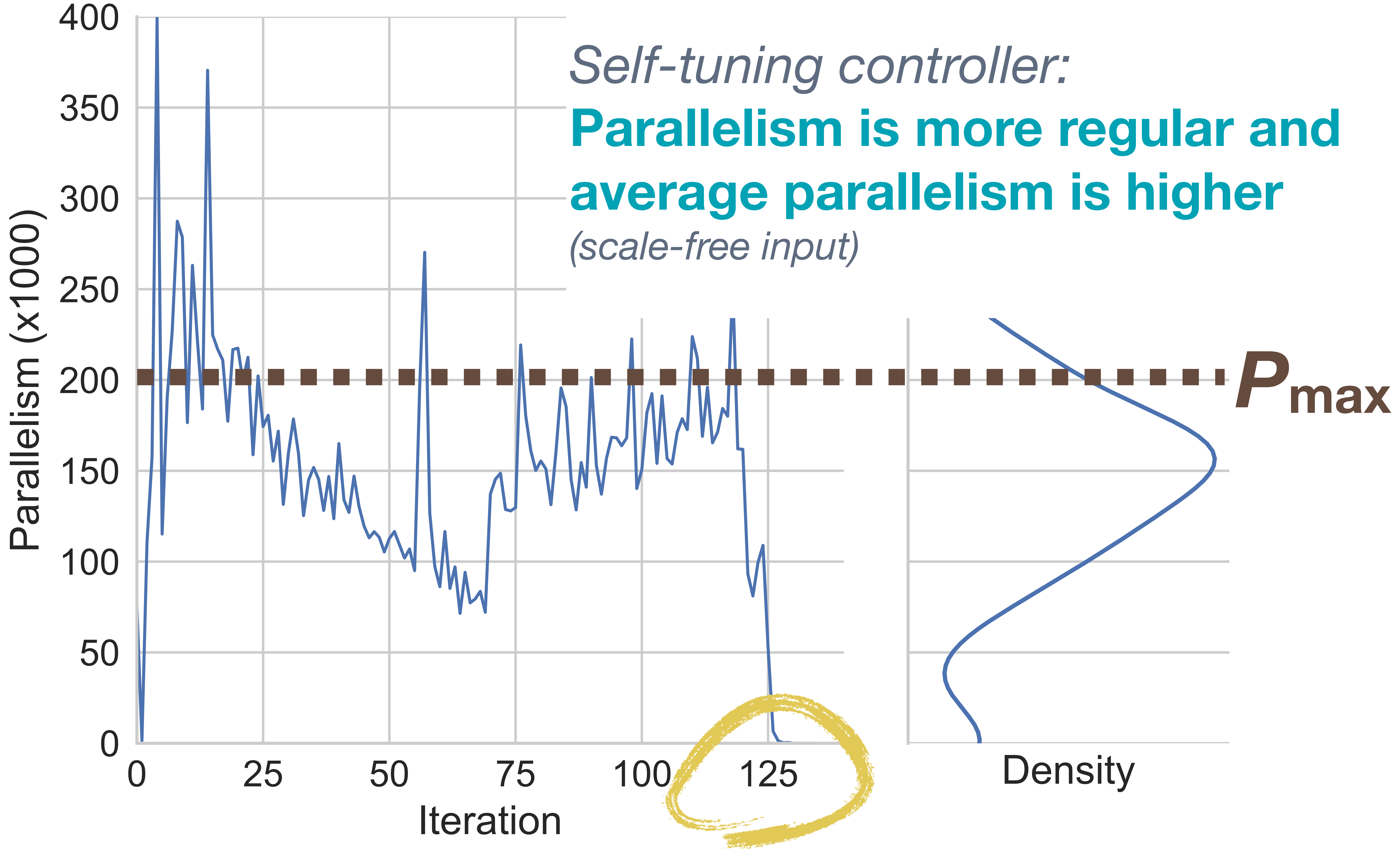
Parallelism is highly irregular, average parallelism is low

(scale-free input)





Self-tuning controller:
Parallelism is more regular and average parallelism is higher
(scale-free input)
Parallelism (x1000)
400
350
300
250
200
150
100
50
0
0
25
50
75
100
125
Iteration
Density
P_{max}

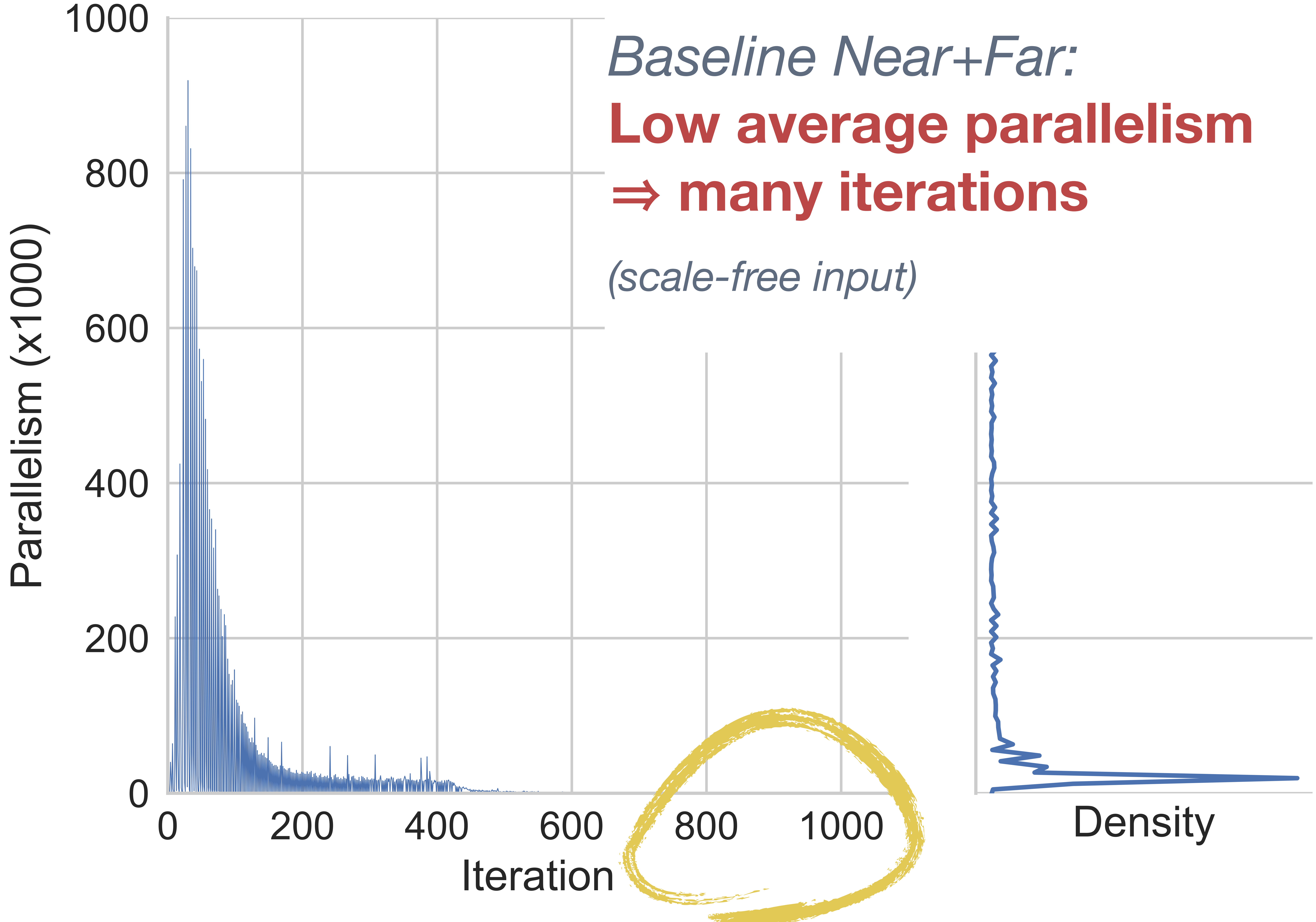


Self-tuning controller:
Parallelism is more regular and average parallelism is higher
(scale-free input)
Parallelism (x1000)
400
350
300
250
200
150
100
50
0
0
25
50
75
100
125
Iteration
Density
P_{max}

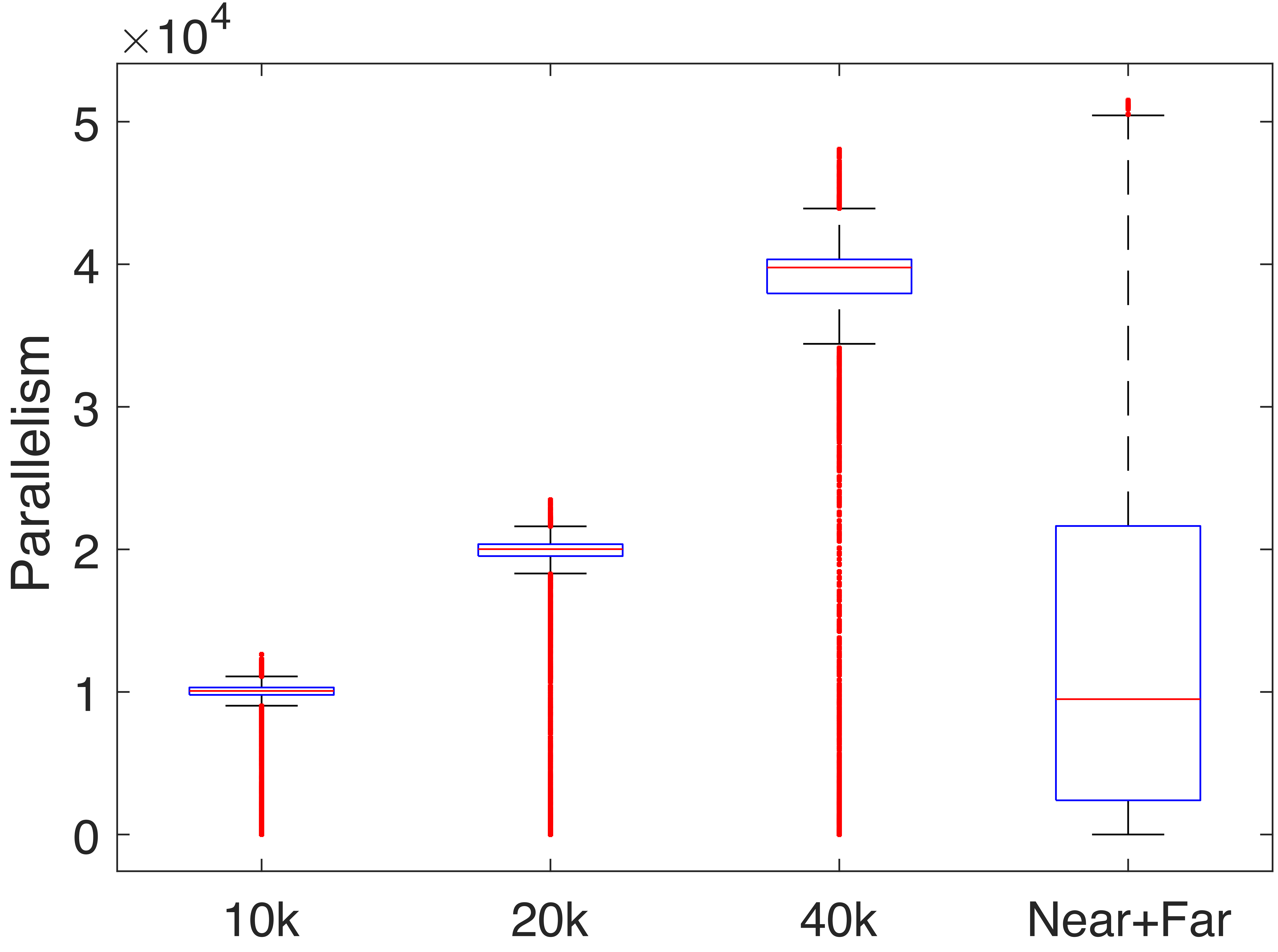
Baseline Near+Far:

Low average parallelism
⇒ many iterations

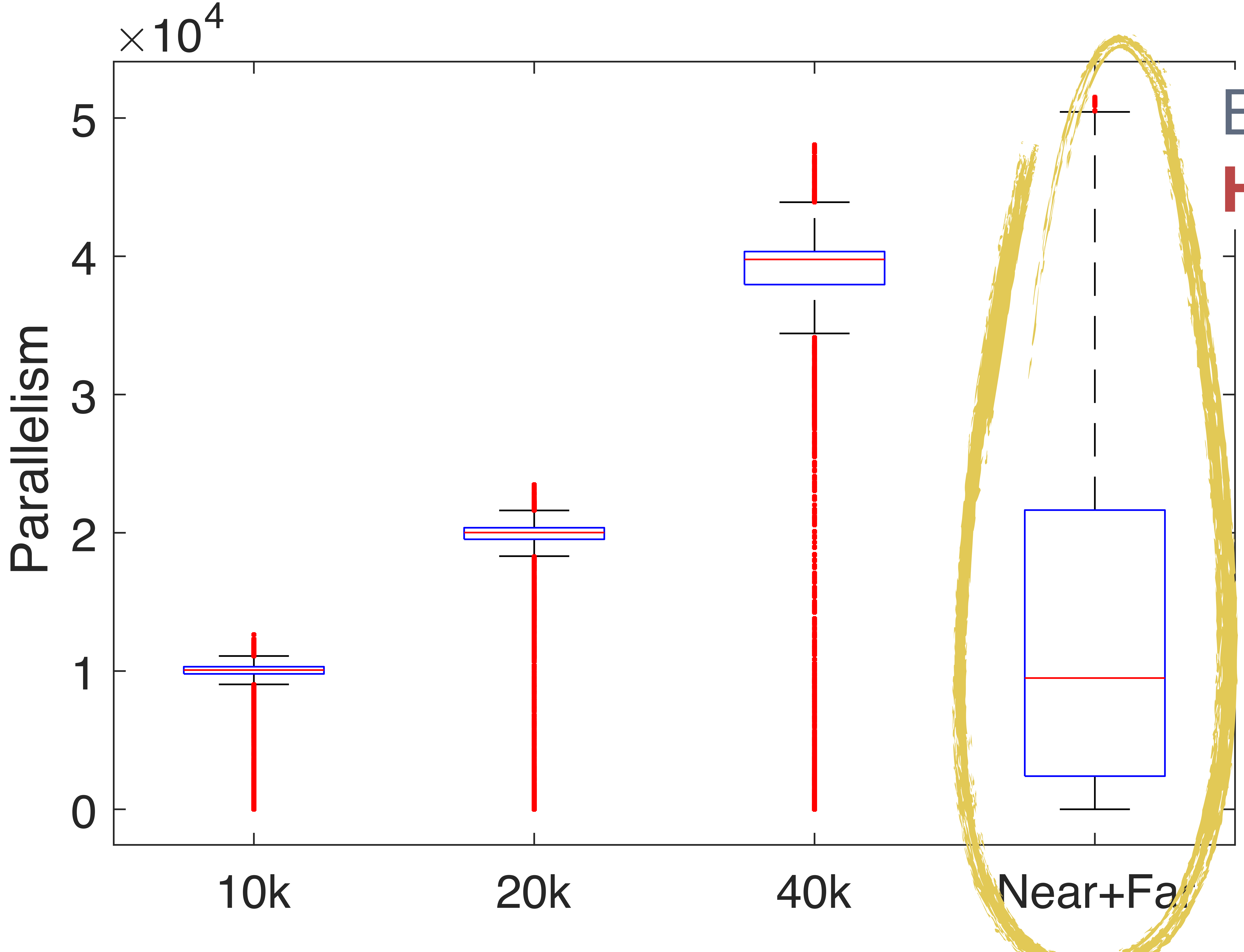
(scale-free input)



(road network)



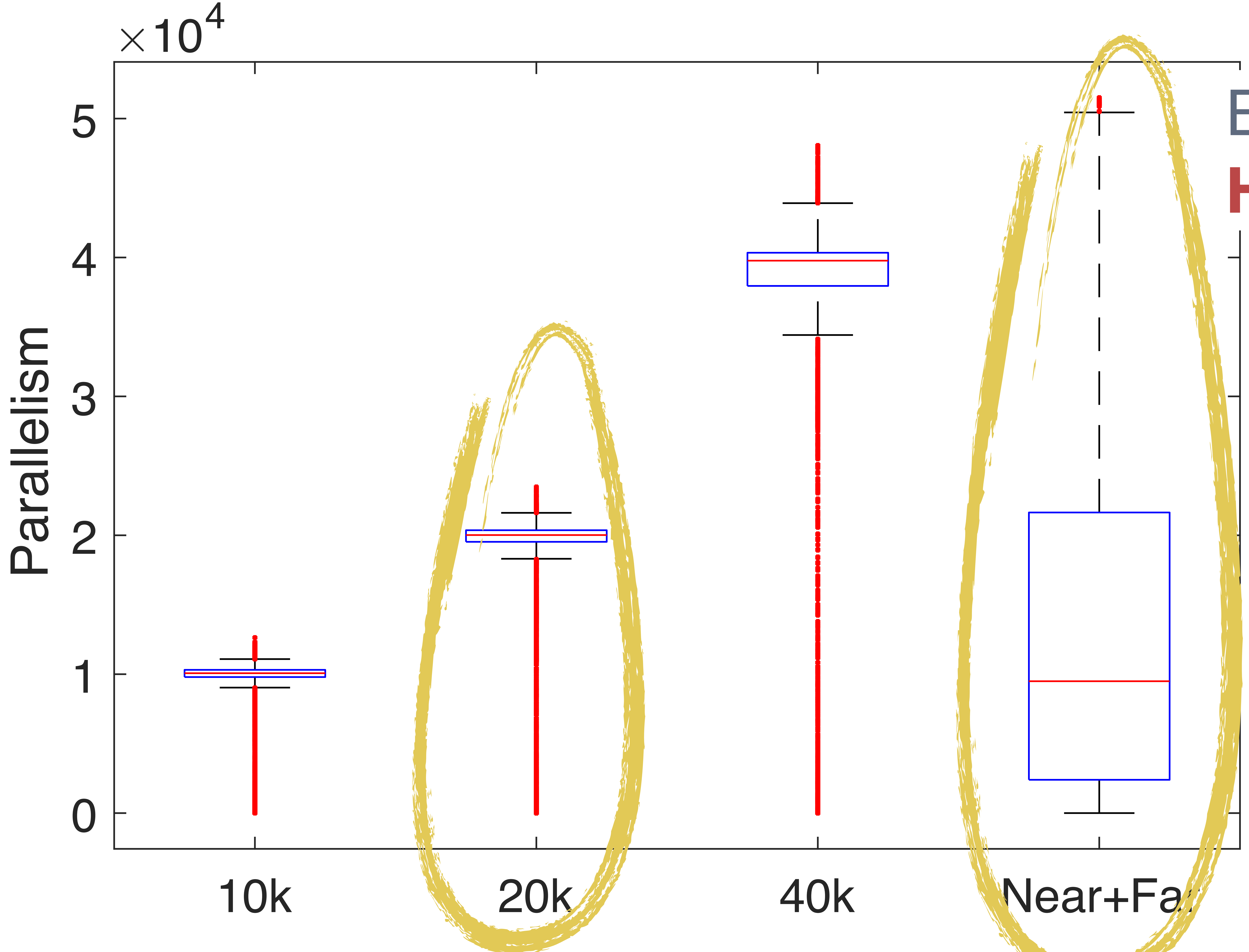
(road network)



Baseline:

High variance

(road network)



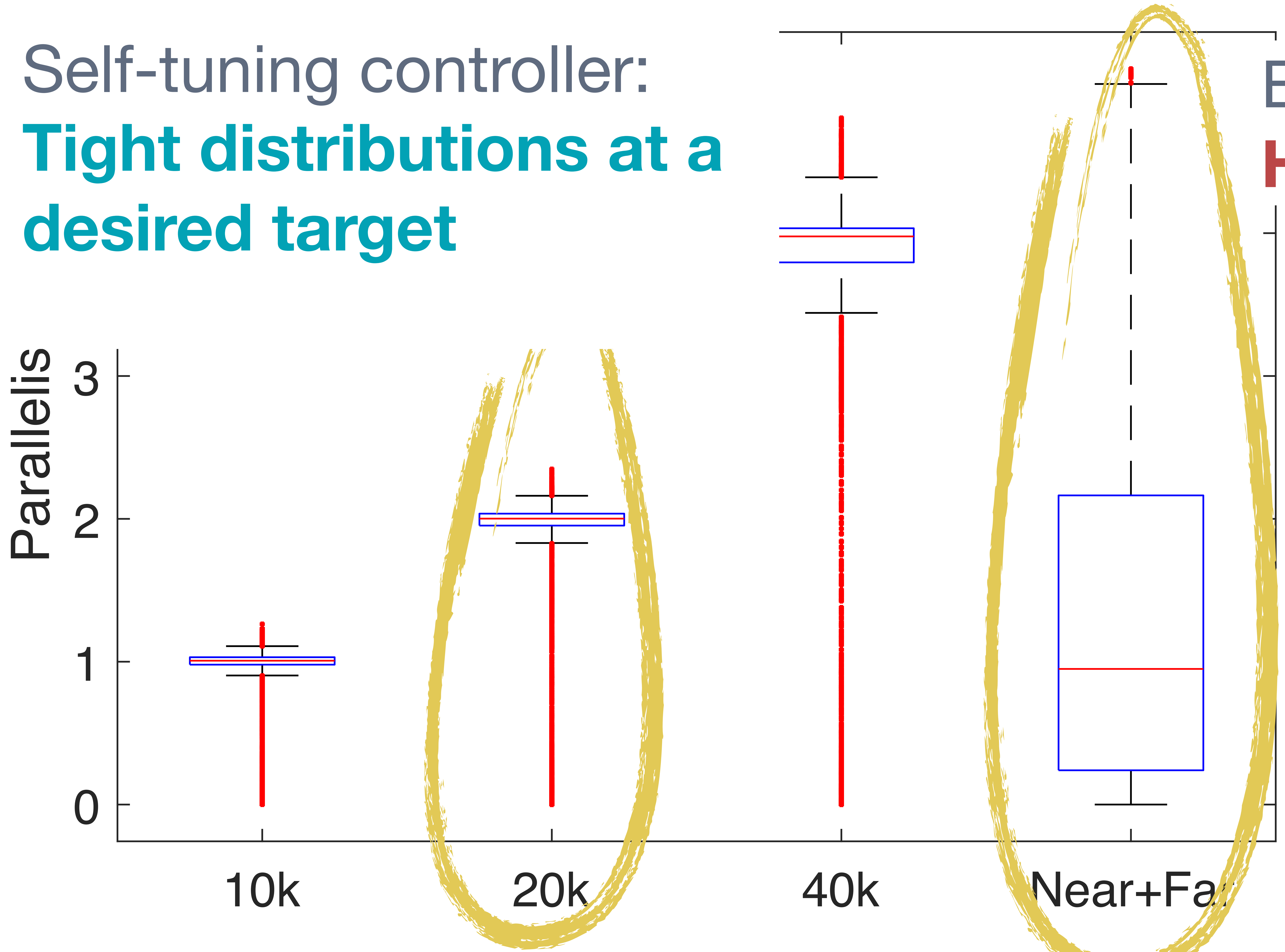
Baseline:

High variance

(road network)

Self-tuning controller:
Tight distributions at a desired target

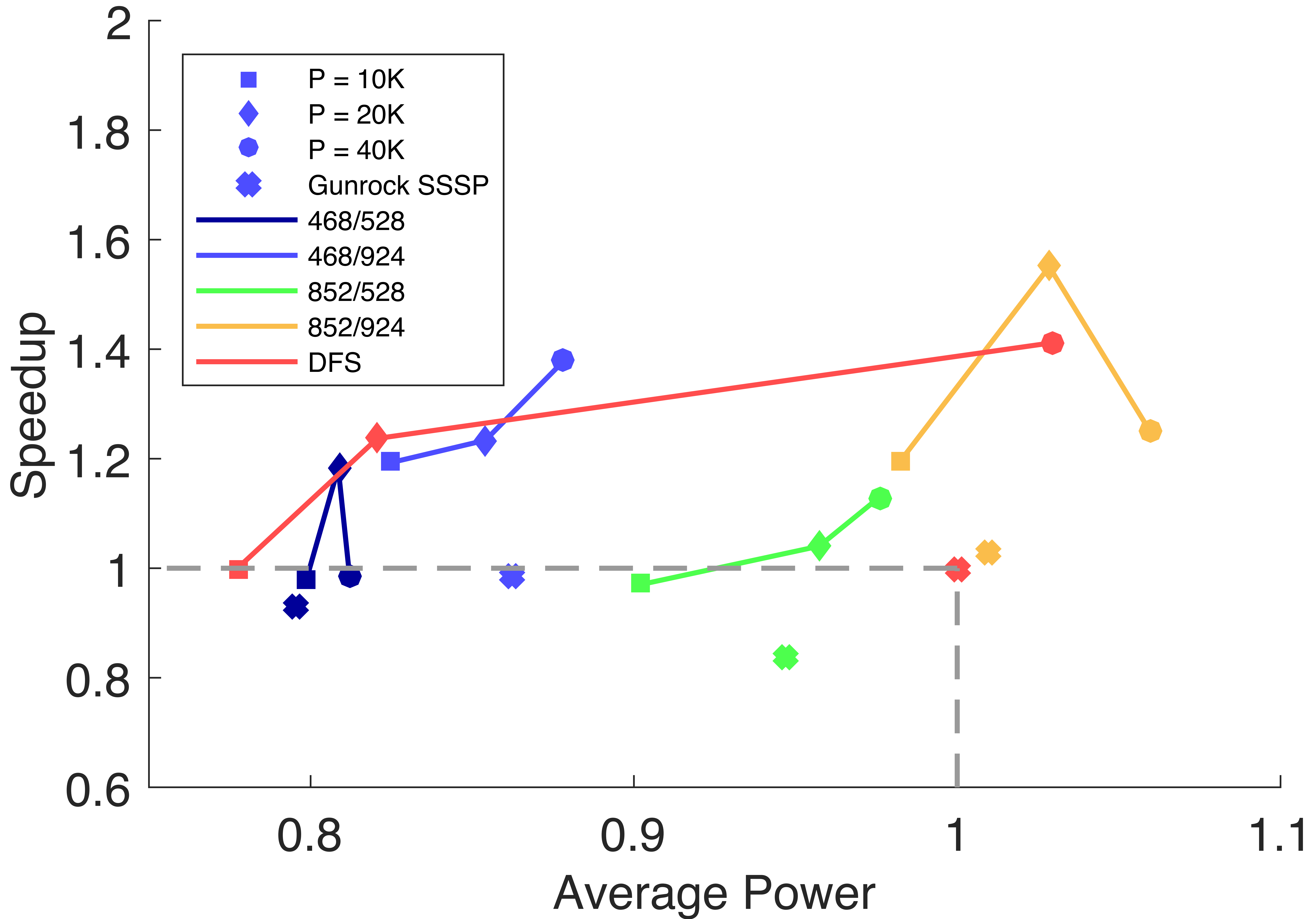
Baseline:
High variance



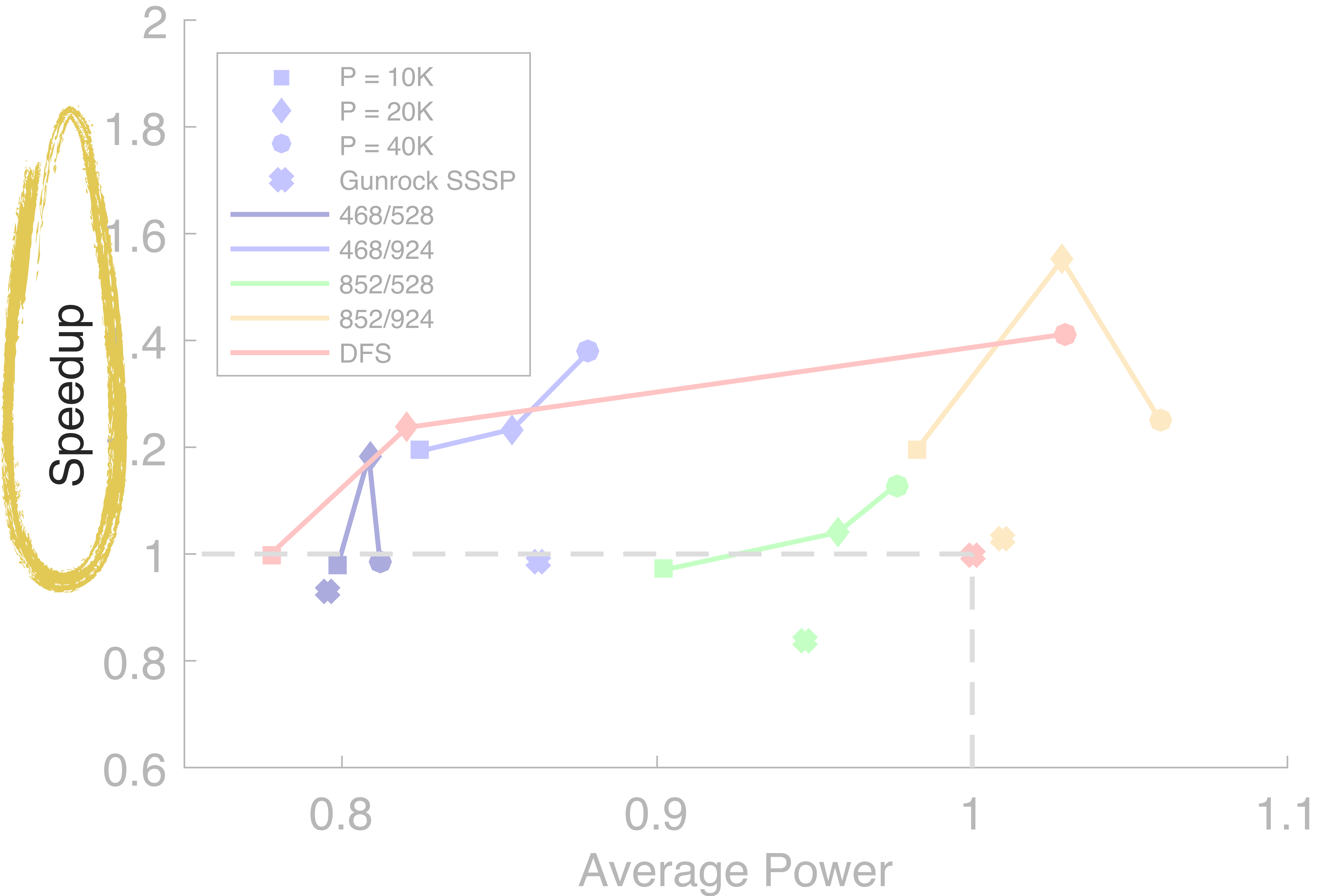
Next question:

Can it **save power** or **improve performance**?

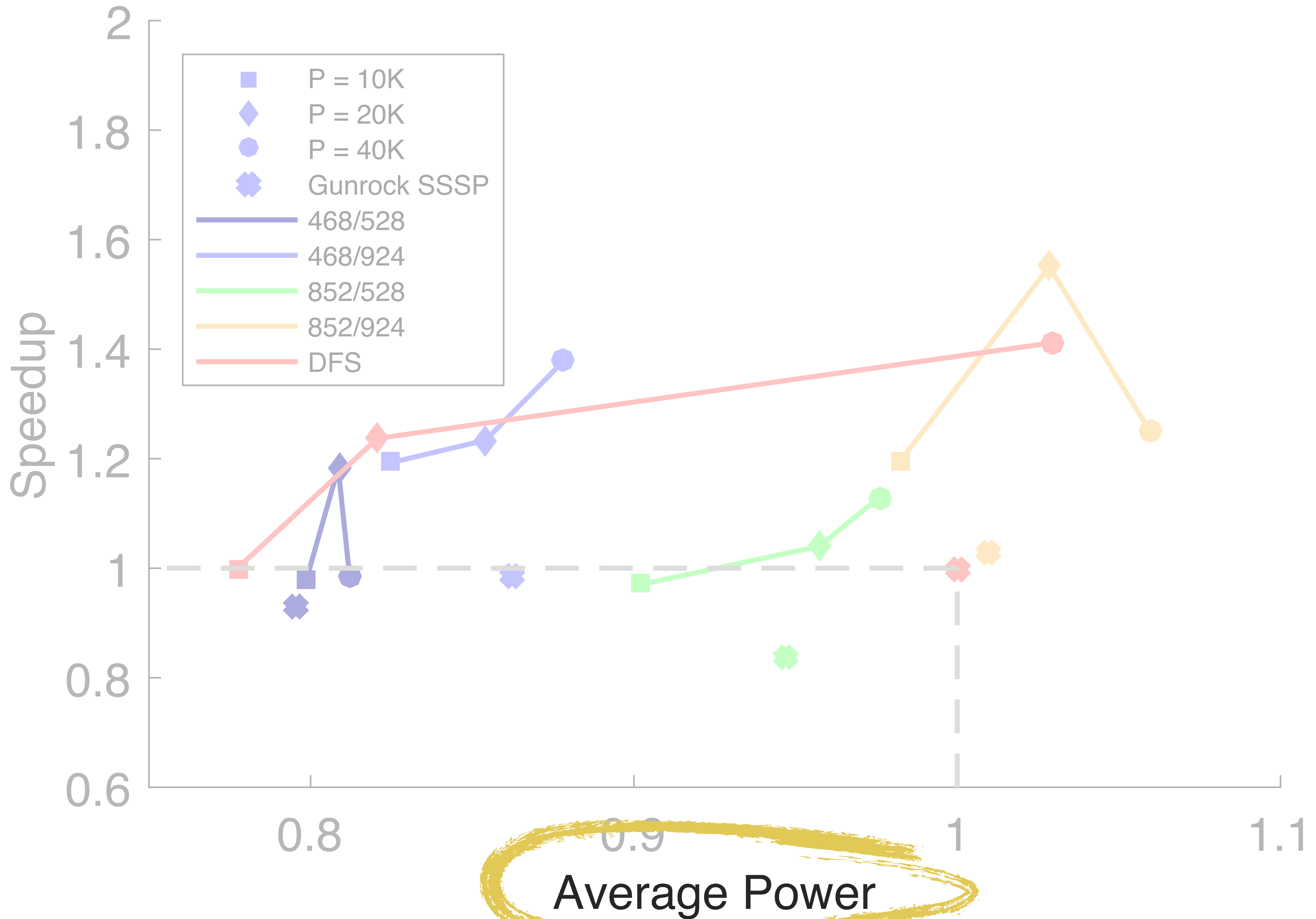
Compare against the baseline with a fixed δ and **hardware-based dynamic frequency scaling.**



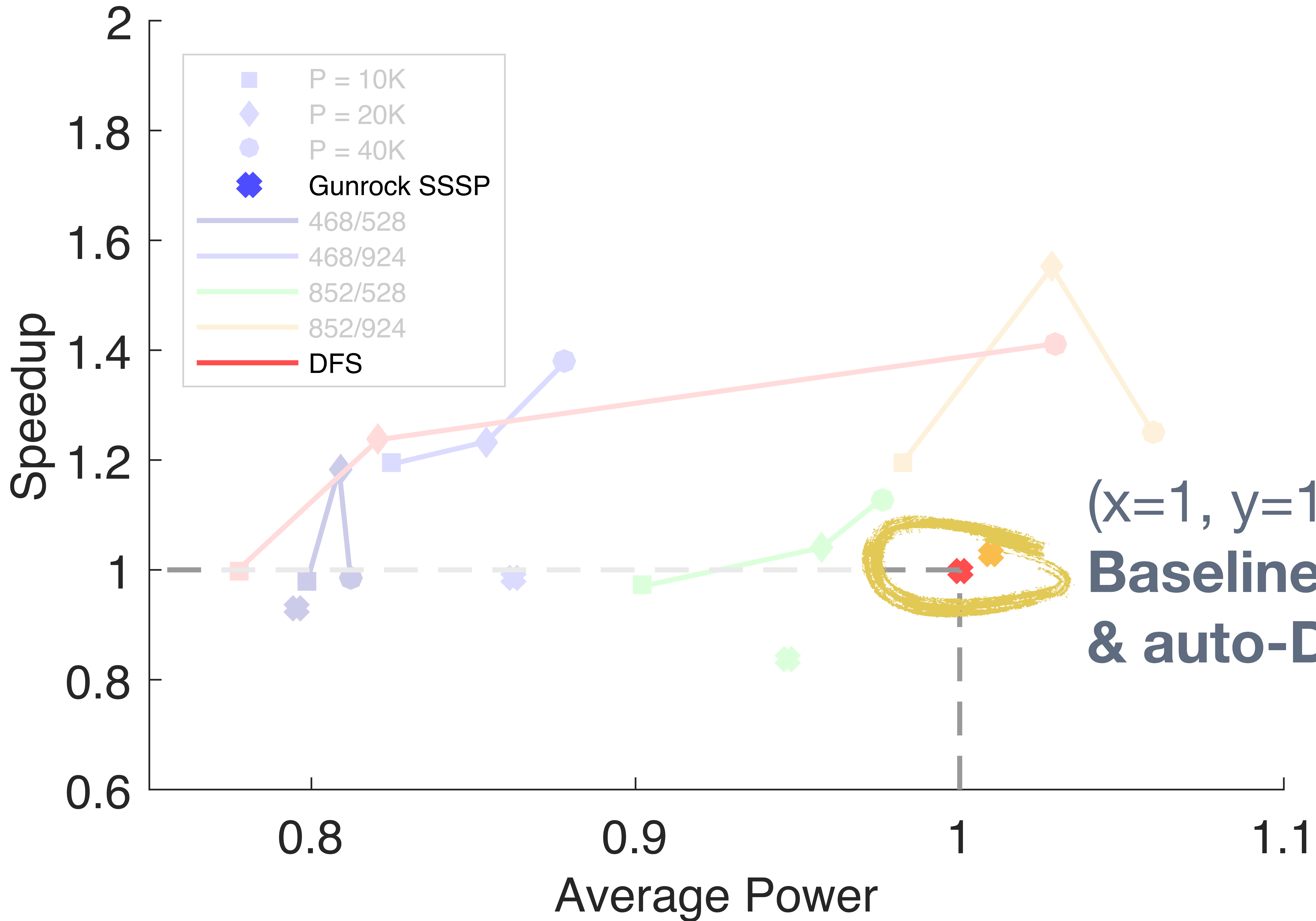
Speedup
Average Power
2
1.8
1.6
1.4
1.2
1
0.8
0.6
0.8
0.9
1
1.1
P = 10K
P = 20K
P = 40K
Gunrock SSSP
468/528
468/924
852/528
852/924
DFS



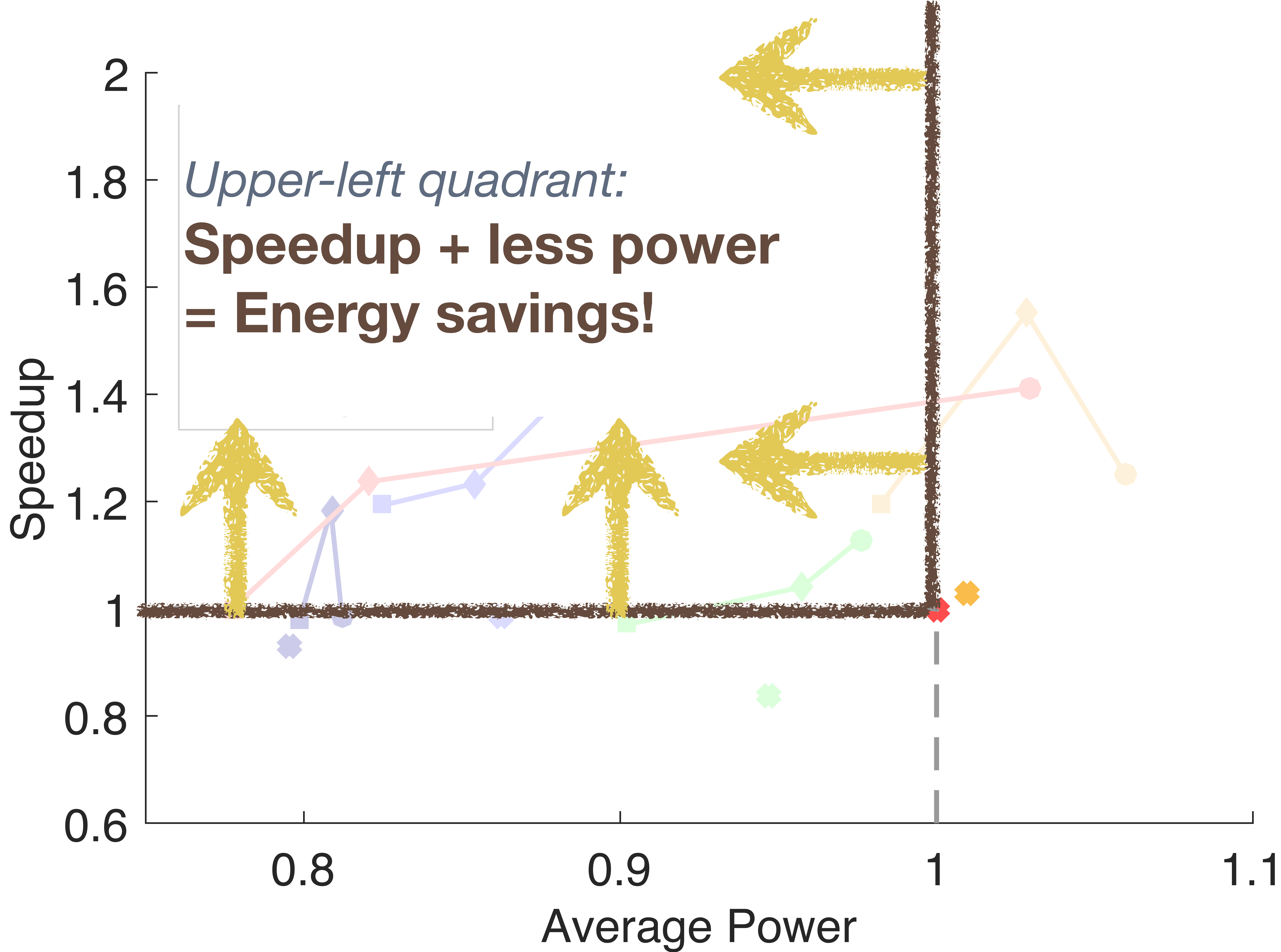
Speedup
Average Power
2
1.8
1.6
.4
.2
1
0.8
0.6
0.8
0.9
1
1.1
P = 10K
P = 20K
P = 40K
Gunrock SSSP
468/528
468/924
852/528
852/924
DFS

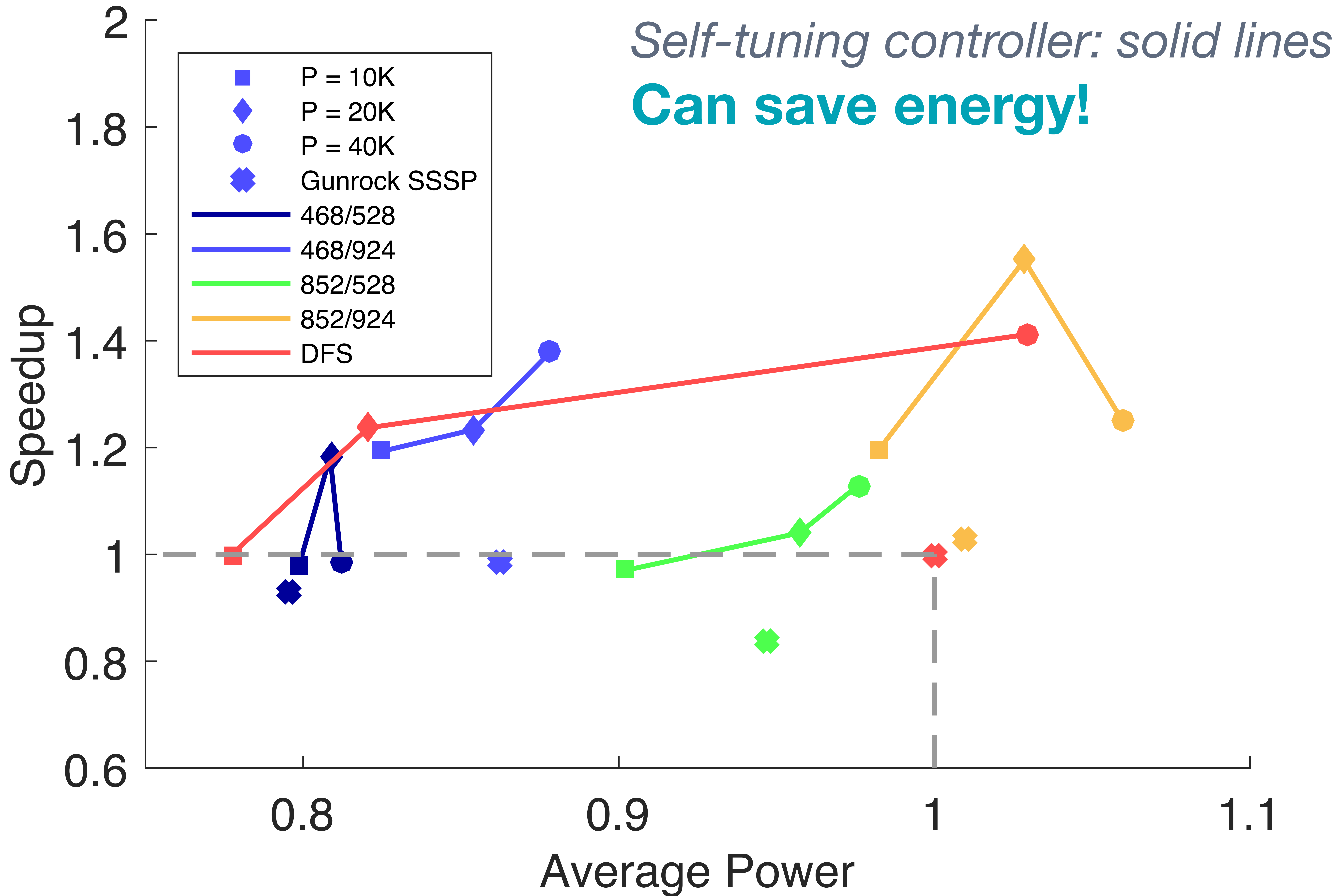


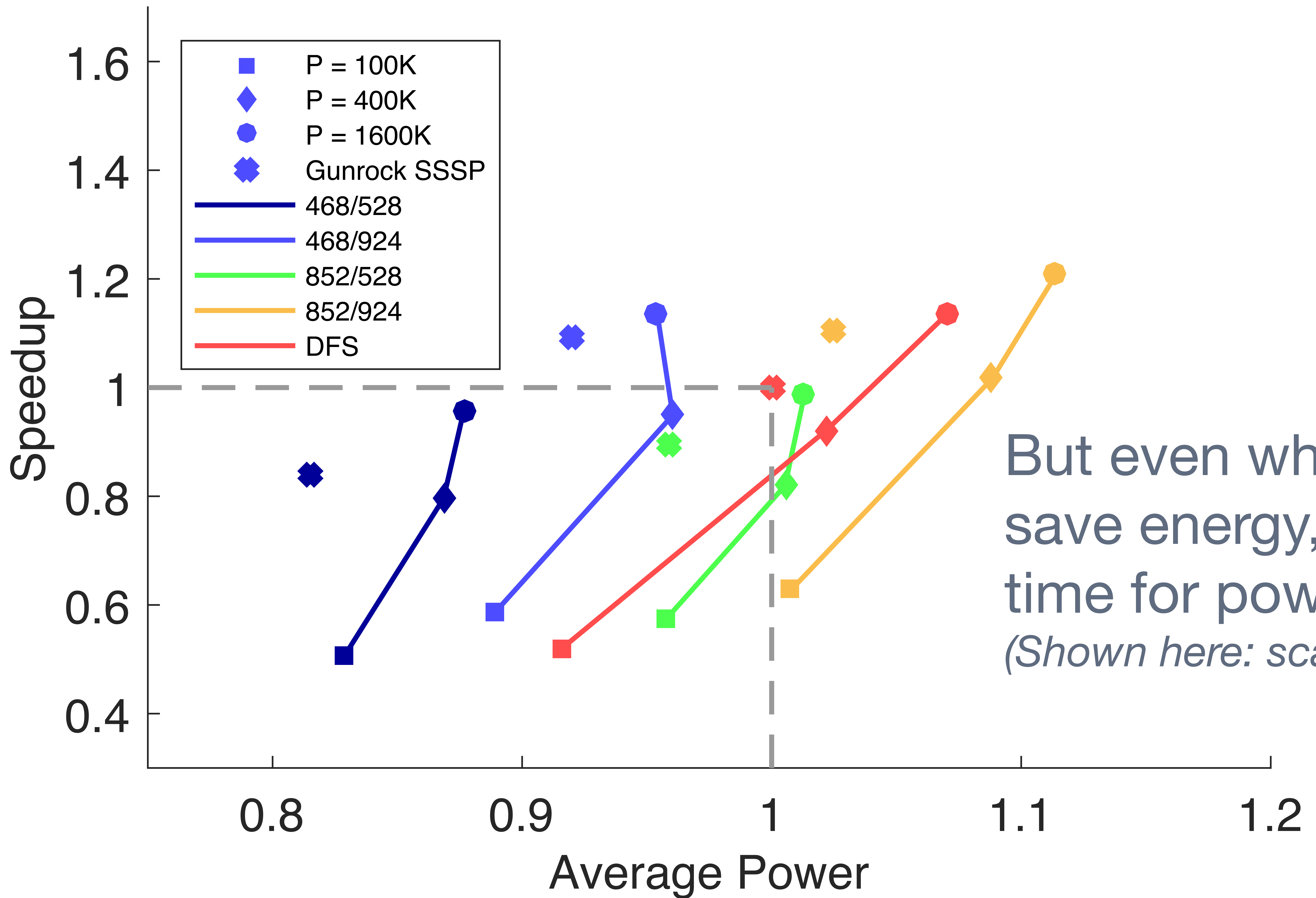
Speedup
2
1.8
1.6
1.4
1.2
1
0.8
0.6
P = 10K
P = 20K
P = 40K
Gunrock SSSP
468/528
468/924
852/528
852/924
DFS
0.8
0.9
1
1.1
Average Power



(x=1, y=1):
Baseline near+far & auto-DFS


Upper-left quadrant:
Speedup + less power
= Energy savings!
Speedup
2
1.8
1.6
1.4
1.2
1
0.8
0.6
0.8
0.9
1
1.1
Average Power


Self-tuning controller: solid lines
Can save energy!
P = 10K
P = 20K
P = 40K
Gunrock SSSP
468/528
468/924
852/528
852/924
DFS
Speedup
2
1.8
1.6
1.4
1.2
1
0.8
0.6
0.8
0.9
1
1.1
Average Power



But even when it can't save energy, it trades time for power gracefully.

(Shown here: scale-free network)

Limitation 1 (open-question):

Choosing *P* is **not the same** as asking for max power, which was our motivation.

There are limits on dynamic power measurement, which is needed to provide feedback to this scheme. But it would likely be easy to incorporate because of the model-based approach.

Limitation 2 (observational):

Power and energy **savings are not big**.

We observed ~ 40% speedups and ~ 15% reductions in maximum power consumption over hardware-only DFS. These savings cannot be bigger because the system baseline or "constant" power is high — it is upwards of 50% or more of maximum power, so the amount of dynamically controllable power is small.

Conclusions:

The “dynamic SSSP” algorithm improves on near+far, even when ignoring power. It’s easier to choose P than δ, making this SSSP easier to use.

The control-based scheme can be applied to any algorithm that is a “sequence of (filter) banks” (e.g., others in Gunrock), though the models may need specialization.

Energy savings are possible, especially when combined with hardware-only techniques like DFS.

hpcgarage.org/uccs

hpcgarage

Rich Vuduc

High-performance computing,
scalable parallel algorithms & software

Georgia Tech College of Computing
Computational Science and Engineering