18-643 Lecture 5: "Performance" Metrics: Beyond Functional Correctness

James C. Hoe
Department of ECE
Carnegie Mellon University

Housekeeping

- Your goal today: review basic concepts and avoid common gotcha's

  Digested from three 18-447 lectures

- Notices
  - Handout #4: Lab 1, due noon, Mon, 9/26
  - Ultra96 ready for pick up
  - Recitation starts this week, Thu 5:00~6:30
  - Prof. Hoe Office Hours, Thusdays 11am~1pm
- Readings (see lecture schedule online)
  - 18-447 Spring 2022 Lectures 5, 12 and 23

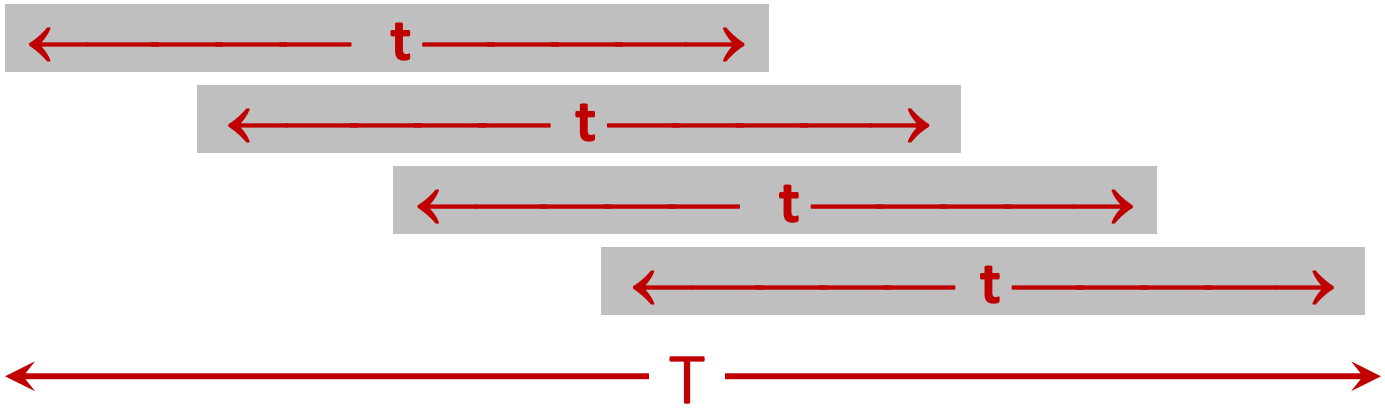
Performance (without “”)

Performance is about time

- To the first order, **performance $\propto$ 1 / time**
- Two very different kinds of performance!!
  - **latency** = time between start and finish of a task
  - **throughput** = number of tasks finished in a given unit of time (a rate measure)
- Either way, shorter the time, higher the performance, but not to be mixed up

Throughput ≠ 1/Latency

- If it takes T sec to do N tasks, throughput=N/T; latency$_1$=T/N?
- If it takes t sec to do 1 task, latency$_1$=t; throughput=1/t?
- When there is concurrency, throughput≠1/latency



- Optimizations can tradeoff one for the other

(think bus vs F1 race car)

Little's Law

- **L**=λ·**W**
  - **L**: number of customers
  - λ: arrival rate
  - **W**: wait time

In 643 language:

 # overlapped tasks

throughput

latency

- In steadystate, fix any two, the third is decided

- HW system examples
  - in-order instruction pipeline: ILP and RAW hazard distance determine instruction throughput
  - AXI DRAM read: latency and # outstanding requests determine achieved BW (until peak)

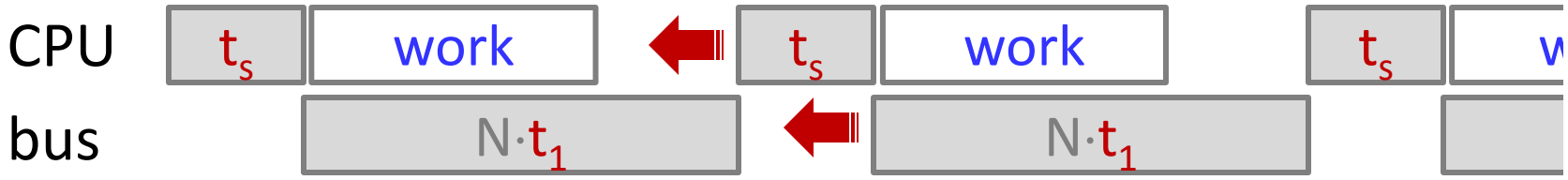
Overhead and Amortization

- Throughput becomes a function of N when there is a non-recurring start-up cost (aka overhead)
- E.g., using DMA to transfer on a bus
  - bus throughput_{raw} = 1 Byte / (10^{-9} sec) *steadystate*
  - 10^{-6} sec to setup a DMA
  - $\text{throughput}_{effective}$ to send 1B, 1KB, 1MB, 1GB?
- For start-up-time=t_s and throughput_{raw}=$1/t_1$
  - $\text{throughput}_{effective} = N / (t_s + N \cdot t_1)$
  - if $t_s >> N \cdot t_1$, $\text{throughput}_{effective} \approx N/t_s$
  - if $t_s << N \cdot t_1$, $\text{throughput}_{effective} \approx 1/t_1$

we say t_s is "amortized" in the latter case

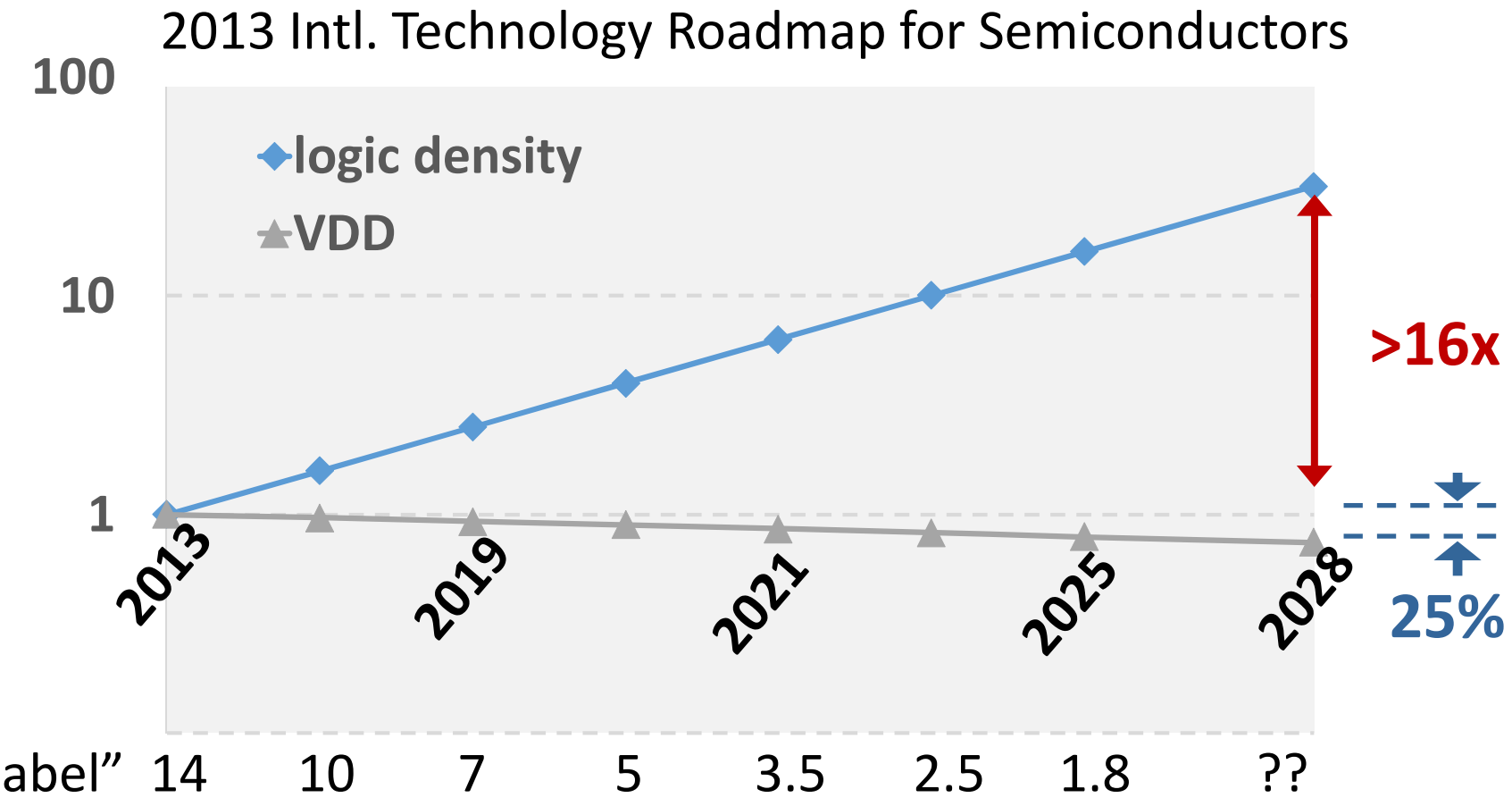
Latency Hiding

- What are you doing during the latency period?
- Latency = hands-on time + hands-off time
- In the DMA example
  - CPU is busy for the t_s to setup the DMA
  - CPU has to wait $N \cdot t_1$ for DMA to complete
  - CPU could be doing something else during $N \cdot t_1$ to "hide" that latency



“Performance” is more than time

Moore's Law without Dennard Scaling



Recall

Under fixed power ceiling, more ops/second only achievable if less Joules/op?

Power = Energy / time

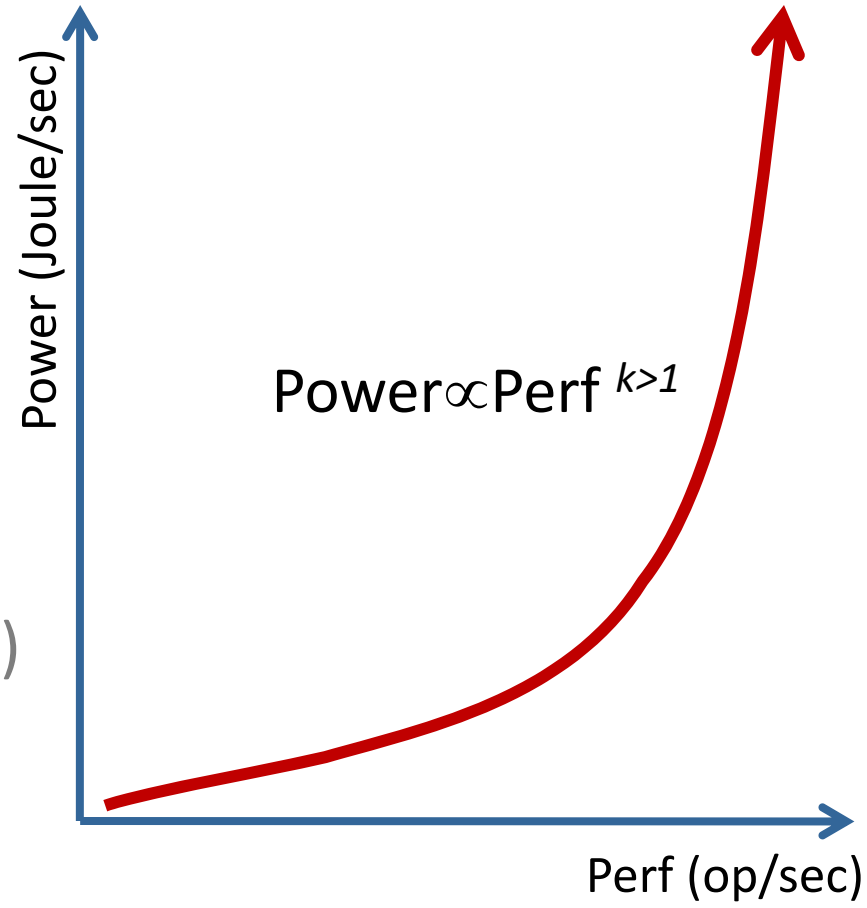
- Energy (Joule) dissipated as heat when "charge" move from VDD to GND
  - takes a certain amount of energy per operation, e.g., addition, reg read/write, (dis)charge a node
  - to the first order, energy $\propto$ work

    You care if on battery or pay the electric bill
- Power (Watt=Joule/s) is rate of energy dissipation
  - more op/sec $\Rightarrow$ more Joules/sec
  - to the first order, power $\propto$ performance

    Usually the problem is "thermal design power"

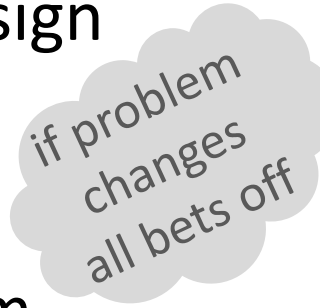
Power and Performance not Separable

- Easy to minimize power if don't care about performance
- Expect superlinear
  increase in power to
  increase performance
  - slower design is simpler
  - lower frequency needs
    lower voltage
- Corollary: Lower perf also use
  lower J/op (=slope from origin)
- Don't forget leakage power



Scale Makes a Difference

- Perf/Watt and J/op are normalized measures
  - hides the scale of problem and platform
  - recall, Watt $\propto$ perf^k for some k>1
- 10 GFLOPS/Watt at 1W is a very different design challenge than at 1KW or 1MW or 1GW
  - say 10 GFLOPS/Watt on a <GPGPU,problem>
  - now take 1000 GPUGPUs to the same problem
  - realized perf is < 1000x (less than perfect parallelism)
  - required power > 1000x (energy to move data & heat)
- Scaling down not always easier with real constraints



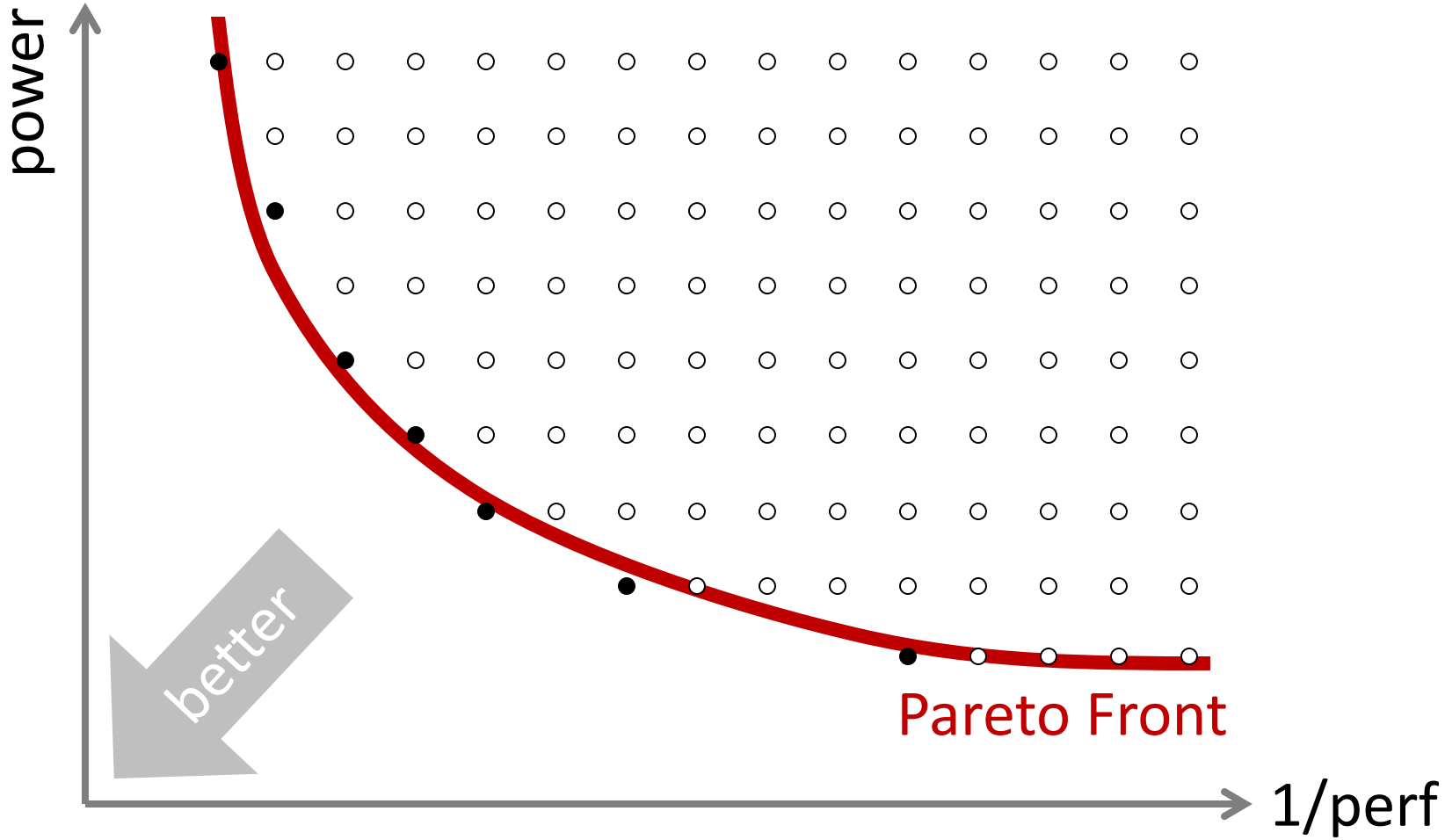
Pay attention to denominator of normalized metrics

Design Tradeoff

Multi-Dimensional Optimizations

- HW design has many optimization dimensions
  - throughput and latency
  - area, resource utilization
  - power and energy
  - complexity, risk, social factors . . .
- Cannot optimize individual metrics without considering **tradeoff** between them, e.g.,
  - reasonable to spend more power for performance
  - converse also true (lower perf. for less power)
  - but never more power for lower performance

Pareto Optimality (2D example)



All points on front are optimal (can't do better)

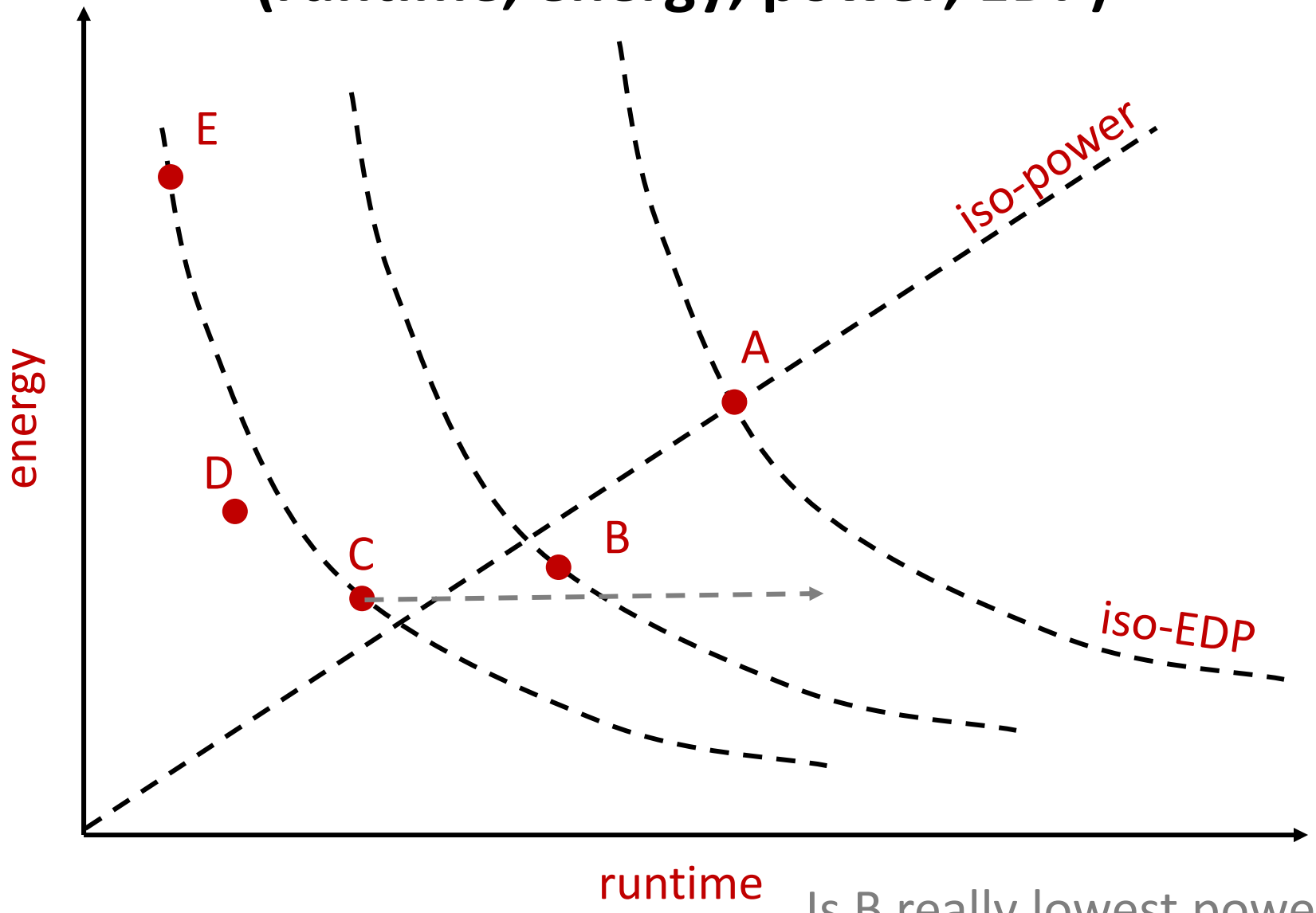
How to select between them?

Application-Defined Composite Metrics

- Define scalar function to reflect desiderata---incorporate dimensions and their relationships
- E.g., energy-delay-(cost) product
  - smaller the better
  - can't cheat by minimizing one ignoring others
  - what does it mean? why not energy^3×delay^2?
- Floors and ceilings
  - real-life designs more often about good enough than being optimal
  - e.g., meet a perf. floor under a power(cost)-ceiling (minimize design time, i.e., stop when you get there)

Which is Design Point is Best?

(runtime, energy, power, EDP)



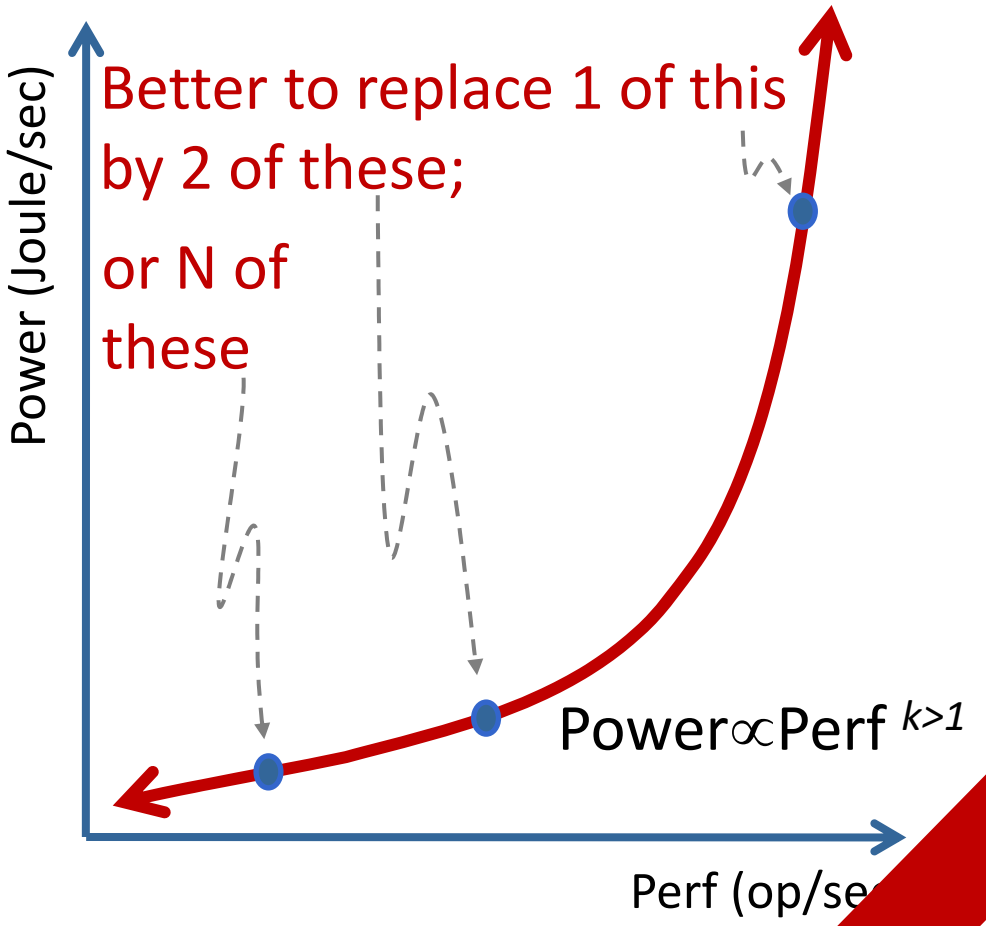
Is B really lowest power?

Parallelism, Speedup and Scalability

Parallelization and Efficiency

- For a given functionality, non-linear tradeoff between power and performance
  - slower design is simpler
  - lower frequency needs lower voltage

⇒For the same throughput, replacing 1 module by 2 half-as-fast reduces total power and energy



Good hardware designs derive performance from parallelism

Parallelism Defined

- T_1 (work measured in time):
  - time to do work with 1 PE
- T_∞ (critical path):
  - time to do work with infinite PEs
  - T_∞ bounded by dataflow dependence
- Average parallelism:

$$P_{avg} = T_1 / T_\infty$$

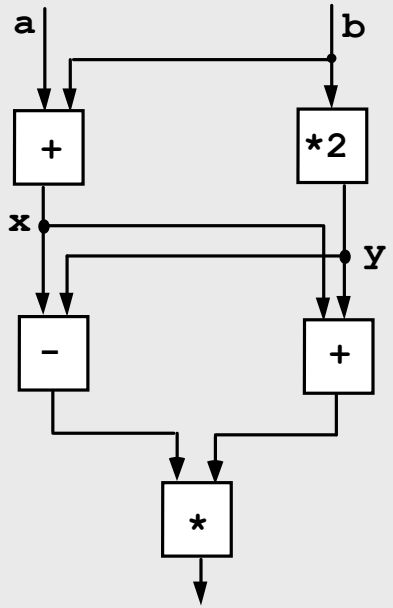
- For a system with p PEs

$$T_p \geq \max\{ T_1/p, T_\infty \}$$

- When $P_{avg} >> p$

$$T_p \approx T_1/p, \text{ aka “linear speedup”}$$

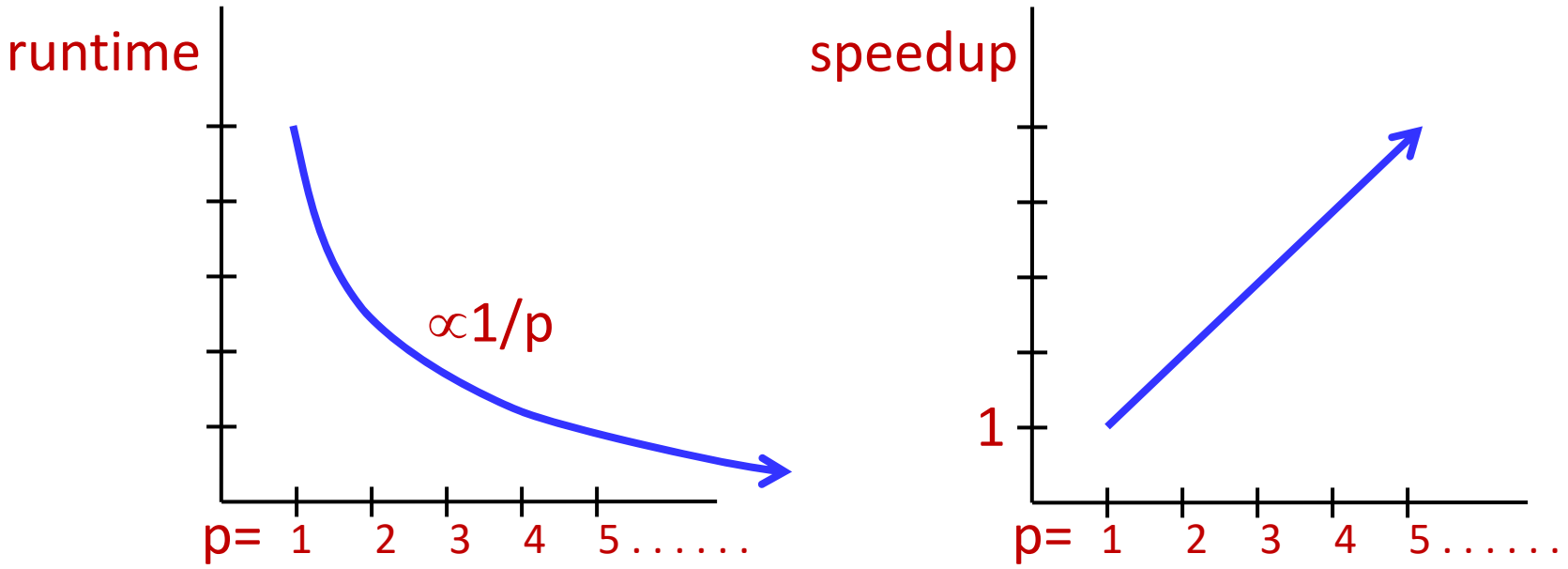
```
x = a + b;
y = b * 2
z =(x-y) * (x+y)
```



Linear Parallel Speedup

- Ideally, parallel speedup is linear with p

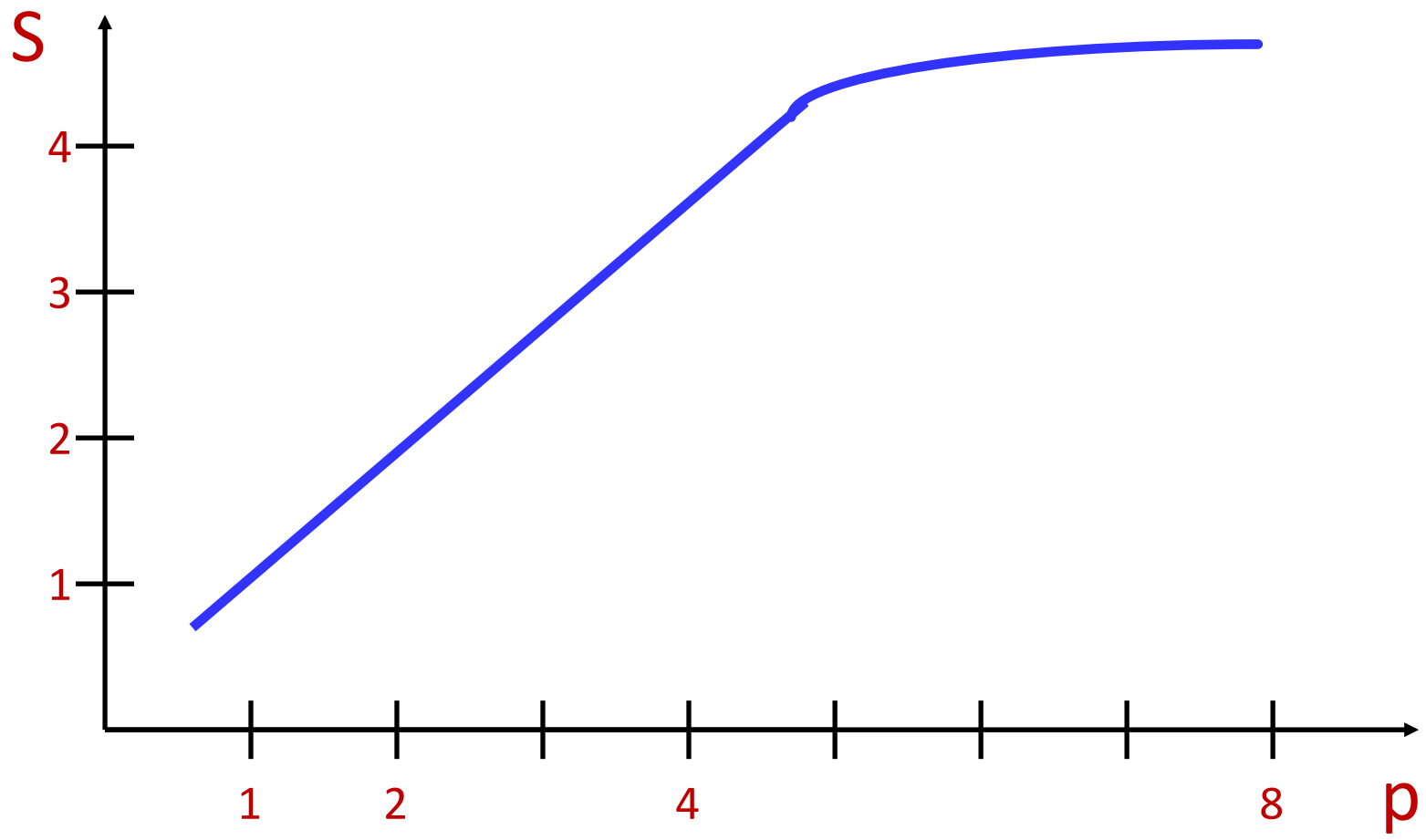
$$\text{speedup} = \frac{\text{runtime}_{\text{sequential}}}{\text{runtime}_{\text{parallel}}}$$



Strong vs. Weak Scaling

- **Strong Scaling** (assumed on last slide)
  - what is S_p as **p** increases for constant work, T_1
    run same workload faster on new larger system
  - harder to speedup as (1) **p** grows toward P_{avg} and (2) communication cost increases with **p**
- **Weak Scaling**
  - what is S_p as **p** increases for larger work, $T_1' = p \cdot T_1$
    run a larger workload faster on new larger system
  - $S_p = \text{time}_{\text{sequential}}(p \cdot T_1) / \text{time}_{\text{parallel}}(p \cdot T_1)$
- Which is easier depends on
  - how P_{avg} scales with work size T_1'
  - relative scaling of bottlenecks *(storage, BW, etc)*

Non-Ideal Speed Up



Parallelism Defined

- T_1 (work measured in time):
  - time to do work with 1 PE
- T_∞ (critical path):
  - time to do work with infinite PEs
  - T_∞ bounded by dataflow dependence
- Average parallelism:

$$P_{avg} = T_1 / T_\infty$$

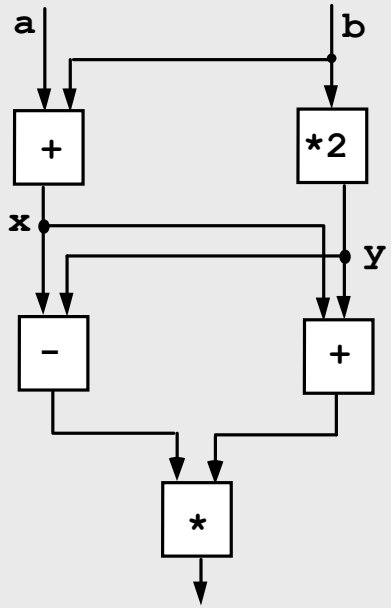
- For a system with **p** PEs

$$T_p \geq \max\{ T_1/p, T_\infty \}$$

- When $P_{avg} >> p$

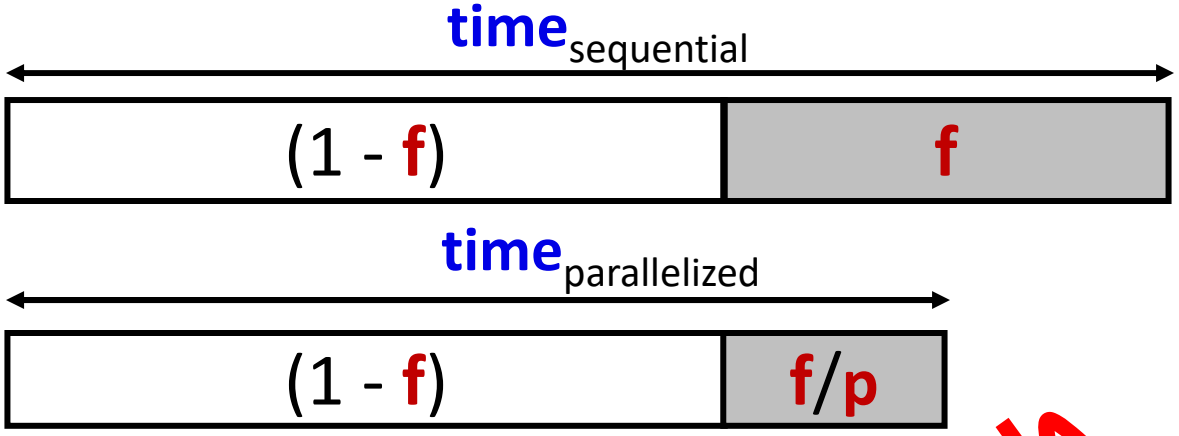
$$T_p \approx T_1/p, \text{ aka “linear speedup”}$$

```
x = a + b;
y = b * 2
z =(x-y) * (x+y)
```



Amdahl's Law

- If only a fraction **f** (by time) is parallelizable by **p**

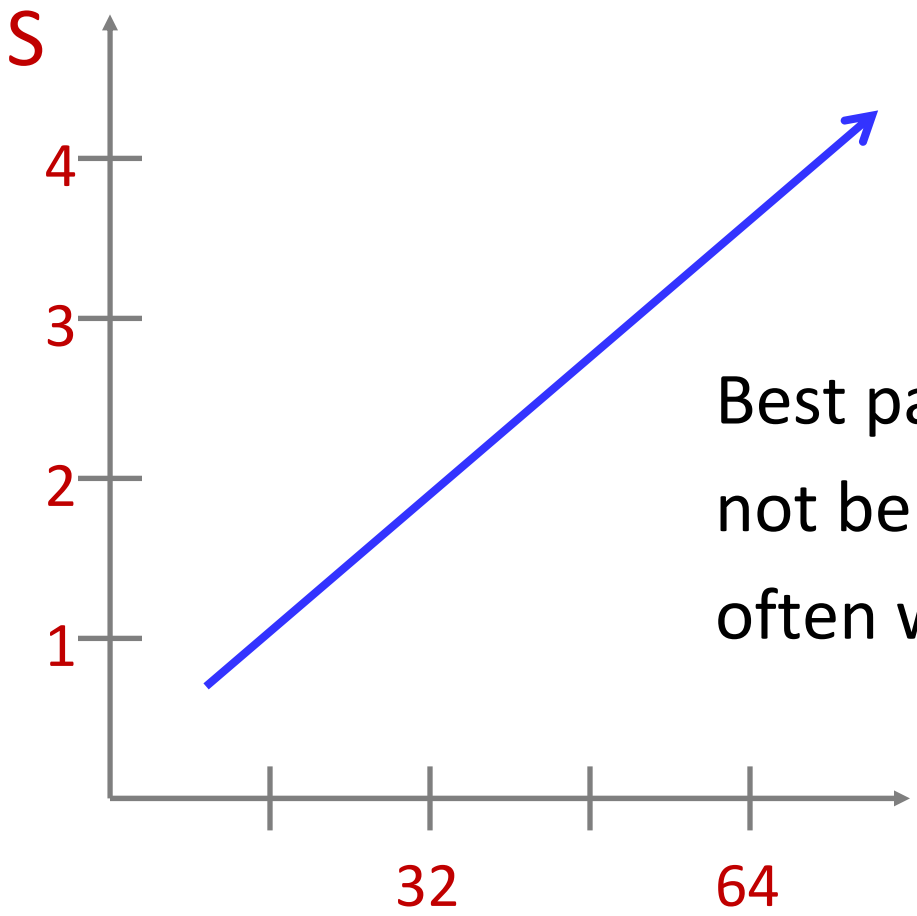


$$\text{time}_{\text{parallelized}} = \text{time}_{\text{sequential}} \cdot ((1-f) + f/p)$$

$$S_{\text{effective}} = 1 / ((1-f) + f/p)$$

  - if **f** is small, **p** doesn't matter
  - even when **f** is large, diminishing return on **p**; eventually "1-**f**" dominates

Non-Ideal Speed Up

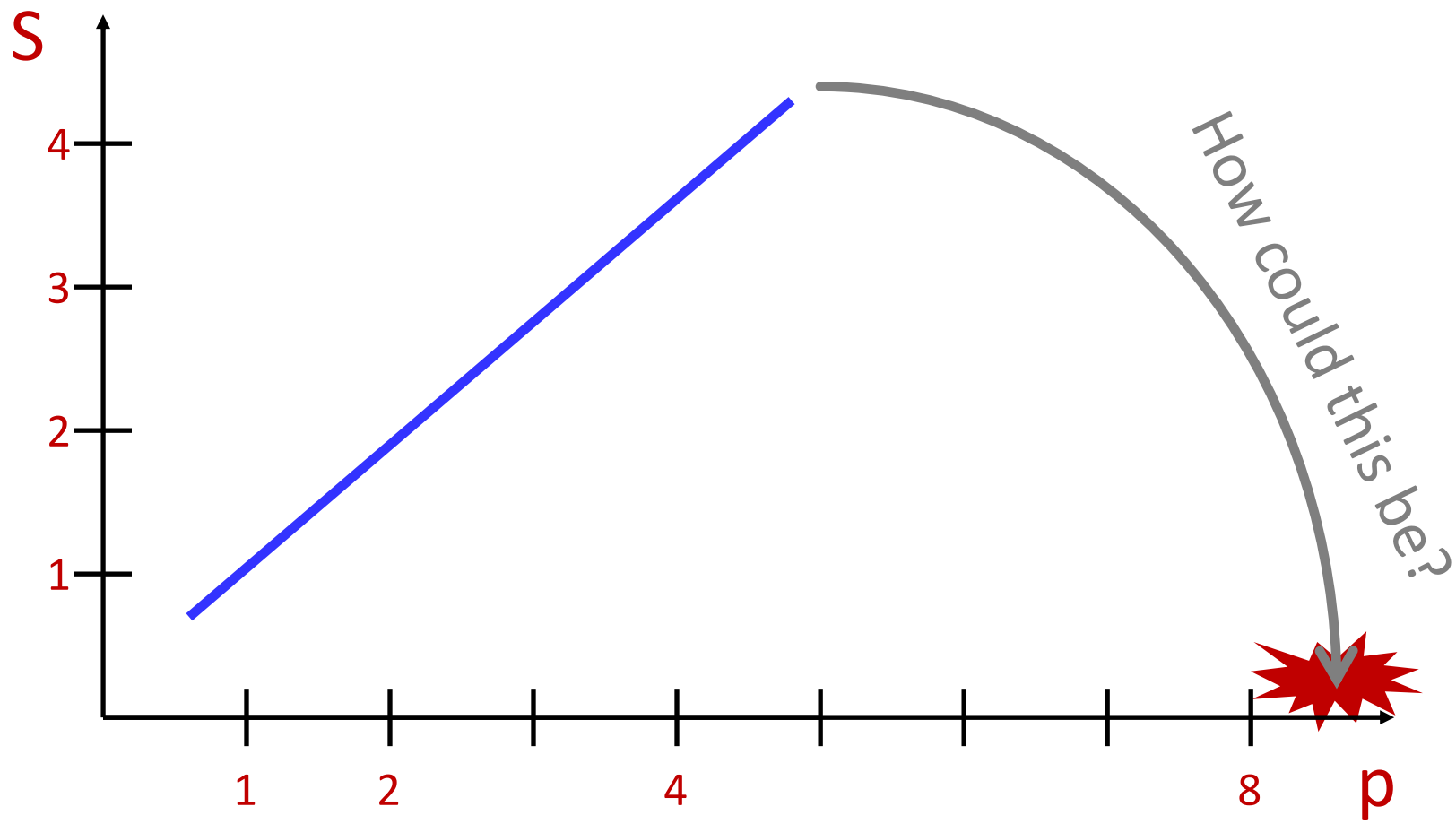


Best parallel algo may not be same as seq. algo, often worse at p=1

if $time_{parallel@p=1} = K \cdot time_{seq}$

then best-case speedup = p/K

Non-Ideal Speedup



Communication not free

- A processing element may spend extra time
  - in the act of sending or receiving data
  - waiting for data to be transferred from another PE
    - latency: data coming from far away
    - bandwidth: data coming thru finite channel
  - waiting for another PE to get to a particular point of the computation (a.k.a. synchronization)

How does communication cost grow with T_1?

How does communication cost grow with p?

LogP Communication Cost Model

- **Latency**: transit time between sender and receiver

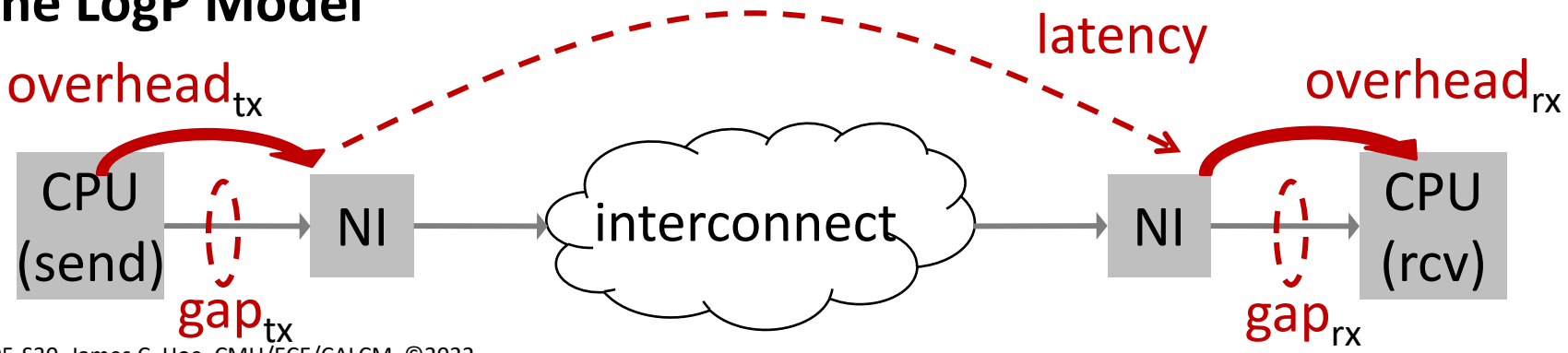
  as if lengthening critical path (T_∞)

- **overhead**: time used up to setup a send or a receive (cycles not doing computation)

  as if adding more work (T_1)

- **gap**: wait time in between successive send's or receive's due to limited transfer bandwidth

The LogP Model



Parting Thoughts

- Need to understand performance to get performance!
- Good HW/FPGA designs involve many dimensions (each one nuanced)
  - optimizations involve making tradeoff
  - over simplifying is dangerous and misleading
  - must understand application needs

Power and energy is first-class!!

- Real-life designs have non-technical requirements