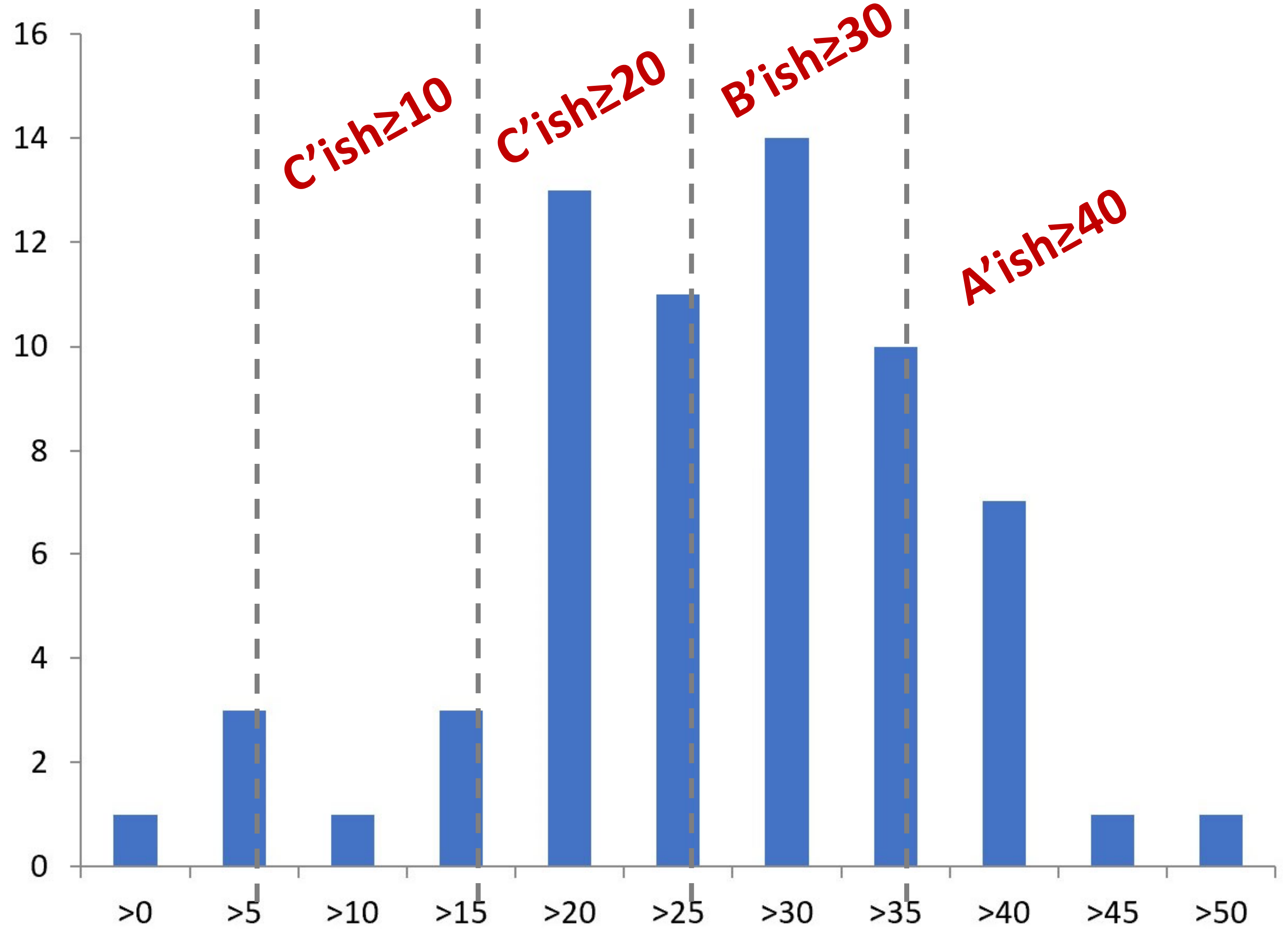
# 18-447 Lecture 12: Energy and Power

James C. Hoe

Department of ECE

Carnegie Mellon University

# 2021 Midterm 1 Histogram



## Housekeeping

- Your goal today
  - a working understanding of energy and power
  - appreciate their significance in comp arch today
- Notices
  - Lab 2, due Friday 3/19 noon (not extended)
  - HW 3, due Monday 3/22 noon
- Readings
  - Design challenges of technology scaling, Borkar, 1999.
  - Synthesis Lectures (advanced optional):
    - Comp Arch Techniques for Power-Efficiency, 2008
    - Power-Efficient Comp Arch: Recent Advances, 2014

## First some intuitions

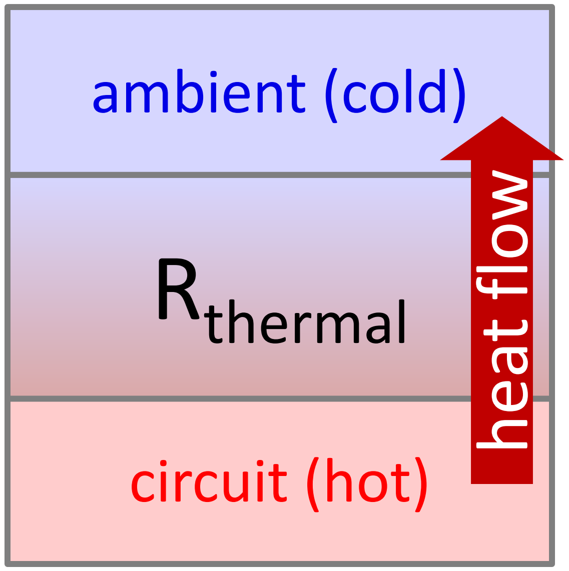
# Energy and Power

- CMOS logic transitions involve charging and discharging of parasitic capacitances
- Energy (Joule) dissipated as resistive heat when "charges" flow from VDD to GND
  - take a certain amount of energy per operation (e.g., addition, reg read/write, (dis)charge a node)
  - to the first order, energy $\propto$ amount of compute
- Power (Watt=Joule/s) is rate of energy dissipation
  - more op/sec then more Joules/sec
  - to the first order, power $\propto$ performance

Power concerns usually more about heat removal

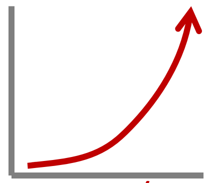
# Heat and Thermal Resistance

- Resistive heat in the circuit must be removed in steadystate (www.youtube.com/watch?v=BSGcnRanYMM)
- Can summarize everything between circuit and ambient by characteristic $R_{thermal}$=K/W
  - convey power W in heat across temperature difference K=$T_{circuit}-T_{ambient}$
- To dissipate more power in circuit
  1. let $T_{circuit}$ get hotter (to a point)
  2. turn-up AC $\Rightarrow$ lower $T_{ambient}$
  3. better cooling $\Rightarrow$ lower R
- Economics/market driven choices



## Work and Perf. from Joules and Watt

- Fastest without energy/power awareness won't be fastest once constrained
  - power bounds performance directly
  - energy bounds work directly; want for lower J/op bounds performance indirectly



recall that power$\propto$perf$^{(\alpha>1)}$

- Consider in context
  - mobile device: limited energy source, hard-to-cool form factor
  - desktop: cooler size and noise
  - data-center: electric bill, cooling capacity and cost

Ultimately driven by size/weight/$$$

## Some (first-order) nitty-gritty

## Work and Runtime

- ***Work***
  - scalar quantity for "amount of work" associated with a task
  - e.g., number of instructions to compute a SHA256 hash

- $T$ = ***Work*** / $k_{perf}$
  - runtime to perform a task
  - $k_{perf}$ is a scalar constant for the rate in which work is performed, e.g., "instructions per second"

# Energy and Power

- $E_{switch} = k_{switch} \cdot Work$
  - "switching" energy associated with task
  - $k_{switch}$ is a scalar constant for "energy per unit work"
- $E_{static} = k_{static} \cdot T = k_{static} \cdot Work \, / \, k_{perf}$
  - "leakage" energy just to keep the chip powered on
  - $k_{static}$ is the so called "leakage power"

*Faster execution means lower leakage energy???*

- $E_{total} = E_{switch} + E_{static} = k_{switch} \cdot Work + k_{static} \cdot Work / k_{perf}$
- $P_{total} = E_{total} / T = k_{switch} \cdot k_{perf} + k_{static}$

Static power can be 50% in high-perf processors

## In Short

- $T = Work \,/\, k_{perf}$

  less work finishes faster

- $E = E_{switch} + E_{static} = (k_{switch} + k_{static}/k_{perf})\cdot Work$

  less work use less energy

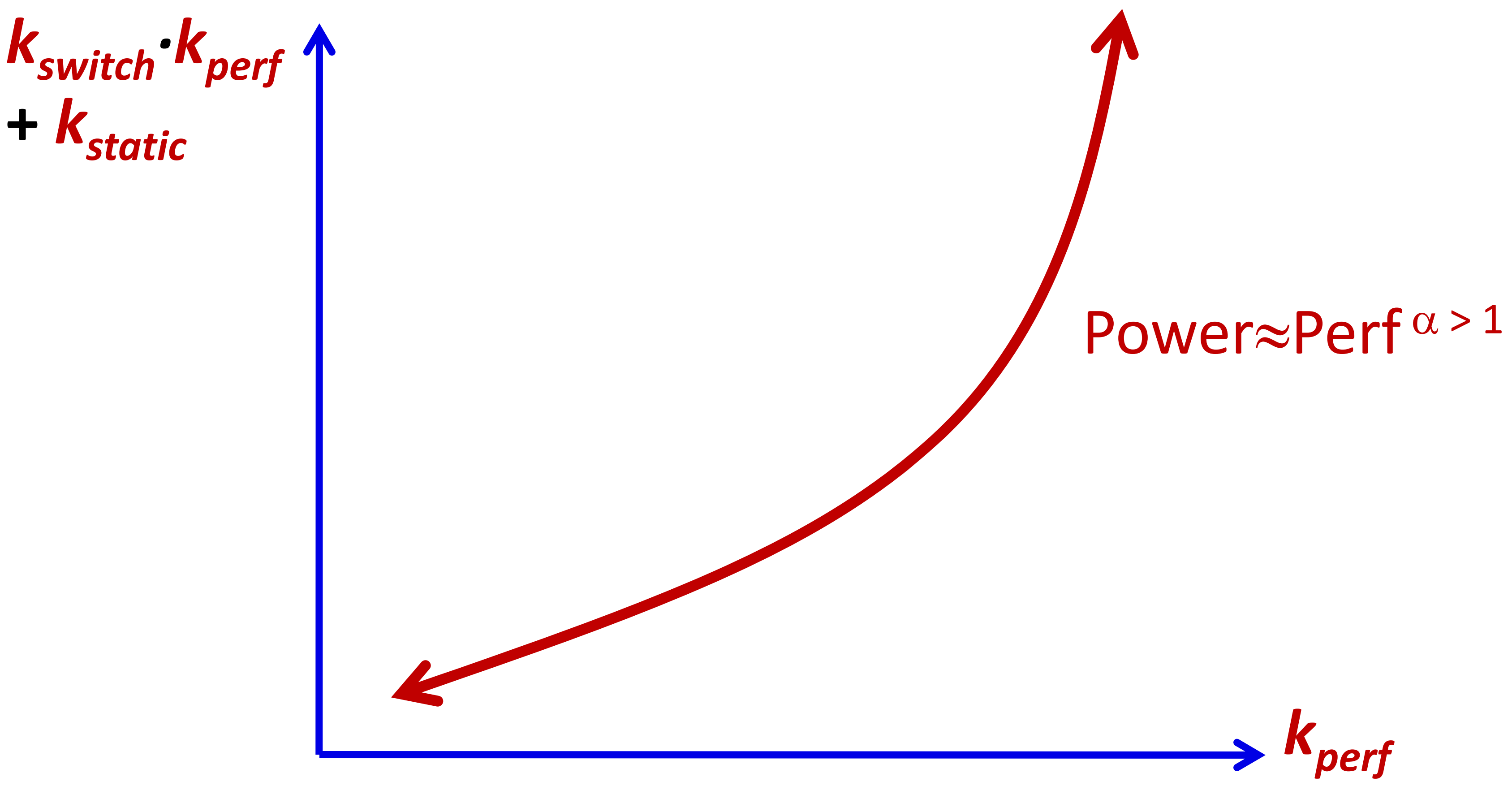
- $P = P_{switch} + P_{static} = k_{switch}\cdot k_{perf} + k_{static}$

  power independent of amount of work

- Reality check
  - *Work* not a simple scalar, inst mix, dependencies ...
  - *k*'s are neither scalar nor constant

$k_{perf}$: inst/sec
$k_{switch}$: J/inst
$k_{static}$: J/sec

# $k_{switch}$, $k_{static}$, $k_{perf}$ not independent



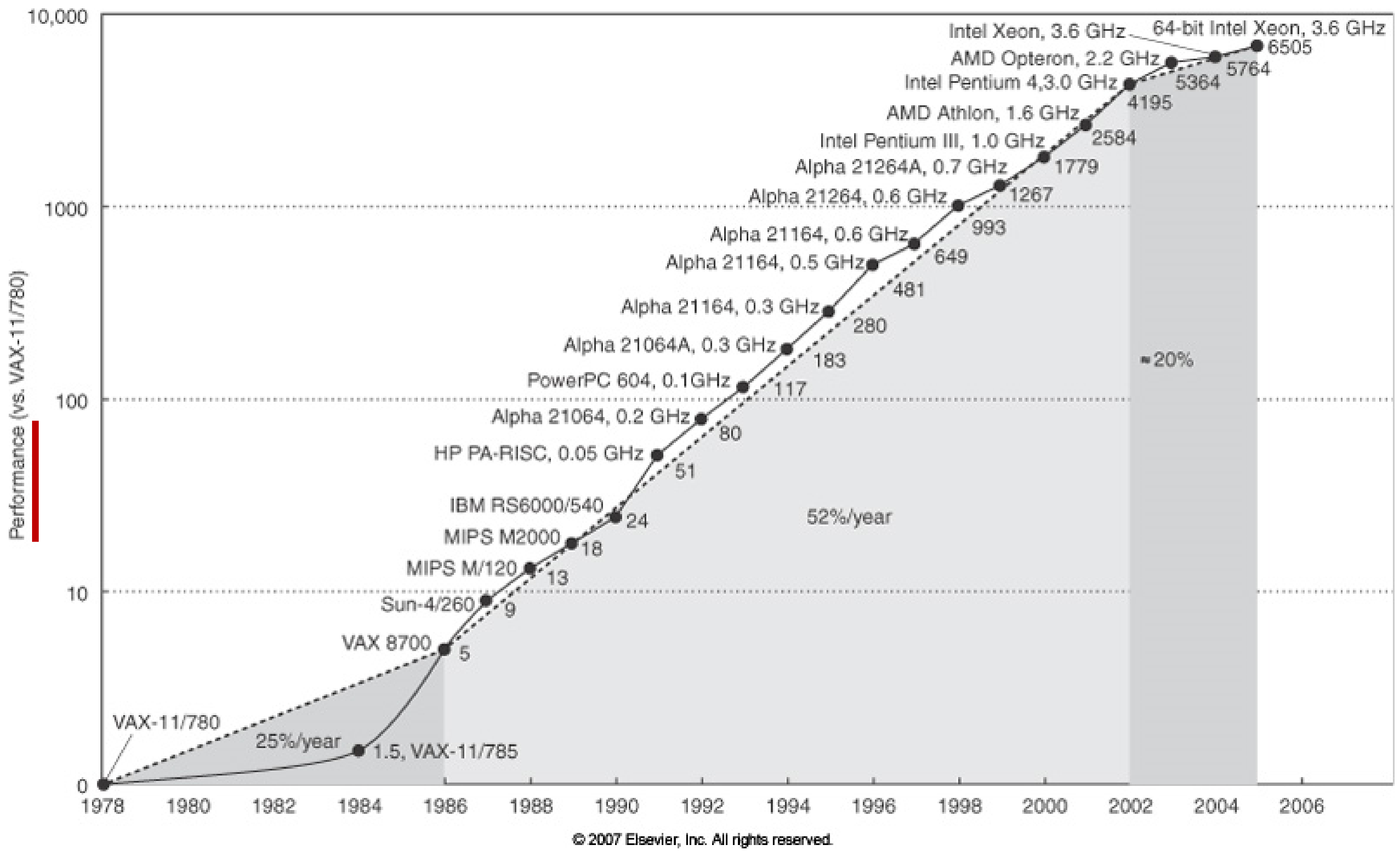
To increase $k_{perf}$: increase die area increases $k_{switch}$ and $k_{static}$
faster transistors increases $k_{static}$

## Why so important now?

## Technology Scaling for Dummies

- Planned scaling occurs in discrete “nodes” where each is ~0.7x of the previous in linear dimension
- Take the same design, reducing linear dimensions by 0.7x (aka “gate shrink”) leads to ****ideally****
  - die area = 0.5x
  - delay = 0.7x; frequency=1.43x
  - capacitance = 0.7x
  - Vdd = 0.7x (constant field) or 1x (constant voltage)
  - power = 0.5x (const. field) or 1x (const. voltage)
- Take the same area, then
  - transistor count = 2x, transistor speed=1.43x
  - power = 1x (const field) or 2x (const voltage)

# The Other Moore's Law



# Moore's Law → Performance

- According to scaling theory

  @constant complexity ("gate-shrink"):

  1x transistors at 1.43x frequency

  $\Rightarrow$ 1.43x performance at 0.5x power

  if perf $\propto$ freq

  @max complexity ("reticle limited"):

  2x transistors at 1.43x frequency

  $\Rightarrow$ 2.8x performance at constant power

  if perf $\propto$ t-cnt $\times$ freq

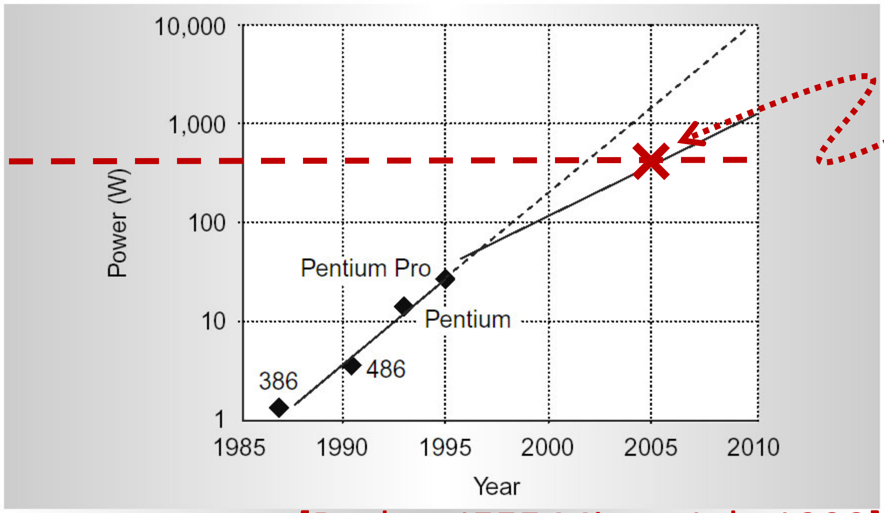
- Historical (until 2005'ish), for high-perf CPUs
  - expected – ~2x transistors
  - higher – ~2x frequency (note: faster than scaling predicts)
  - all together, ~2x performance at ~2x power

lower higher *Why so far off?*

# Performance (In)efficiency

- To hit "expected" performance target
  - push frequency harder by deepening pipelines
  - used the 2x transistors to build more complicated microarchitectures so fast/deep pipelines don't stall (i.e., caches, BP, superscalar, out-of-order)
- The consequence of performance inefficiency is

limit of economical cooling [ITRS]

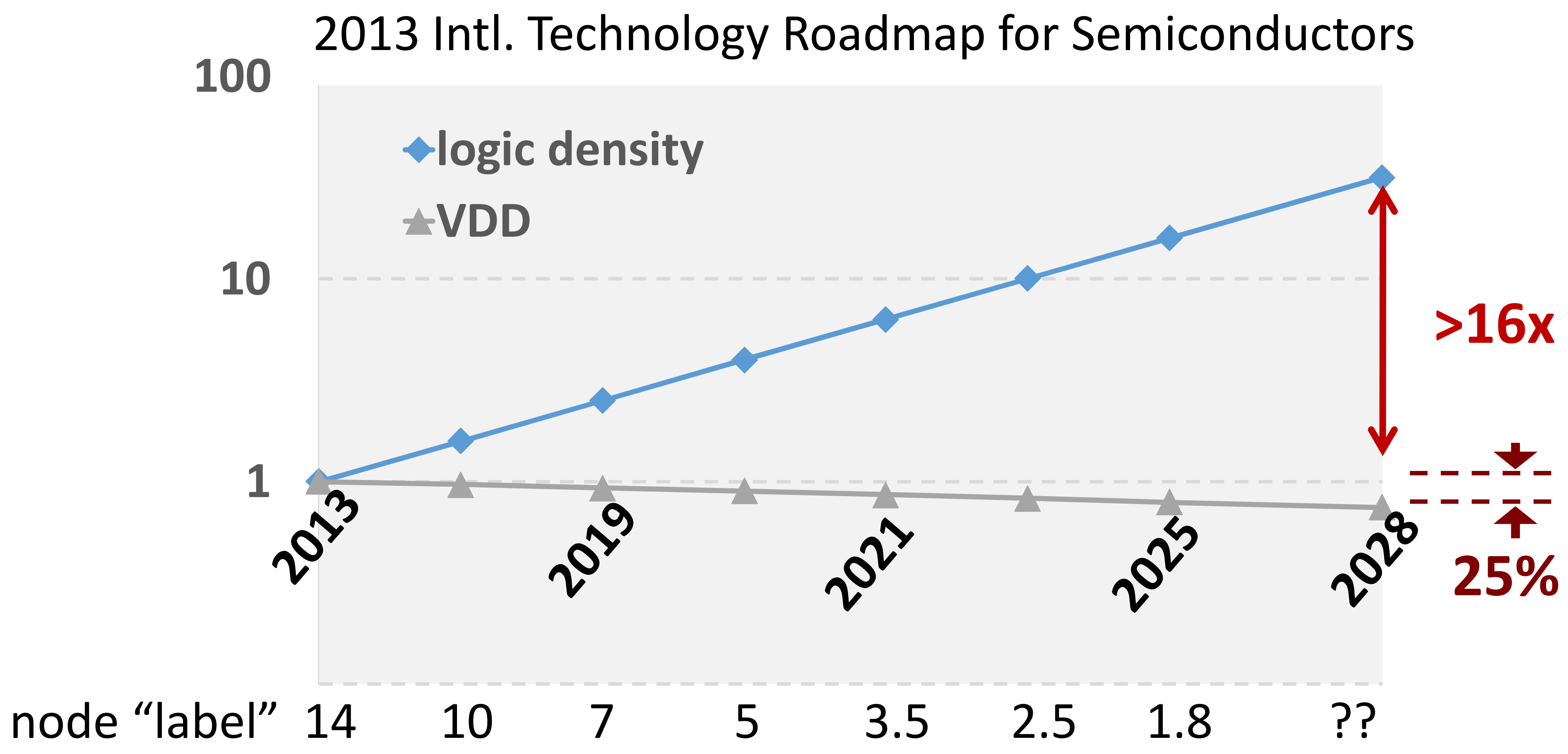


2005, Intel P4 Tehas 150W

Figure 8. Power dissipation projections.

[Borkar, IEEE Micro, July 1999]

# Moore's Law without Dennard Scaling



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

## Frequency and Voltage Scaling: run <u>slower</u> at lower energy-per-op

## Frequency and Voltage Scaling

- Switching energy per transition is

  $\frac{1}{2}CV^2$ (modeling parasitic capacitance)

- Switching power at $f$ transitions-per-sec is

  $\frac{1}{2}CV^2f$

- To reduce power, slow down the clock
- If clock is slower ($f'$), reduce supply voltage ($V'$) too since transistors don't need to be as fast
  - reduced switching energy, $\frac{1}{2}CV^2 \rightarrow \frac{1}{2}CV'^2$
  - lower $V'$ also reduced leakage current/power

## Frequency Scaling (by itself)

- If $Work\,/\,k_{perf} < T_{bound}$, we can derate performance by frequency scaling by a factor $\boldsymbol{s_{freq}}$

  $$(Work/k_{perf})/T_{bound} < \boldsymbol{s_{freq}} < 1$$

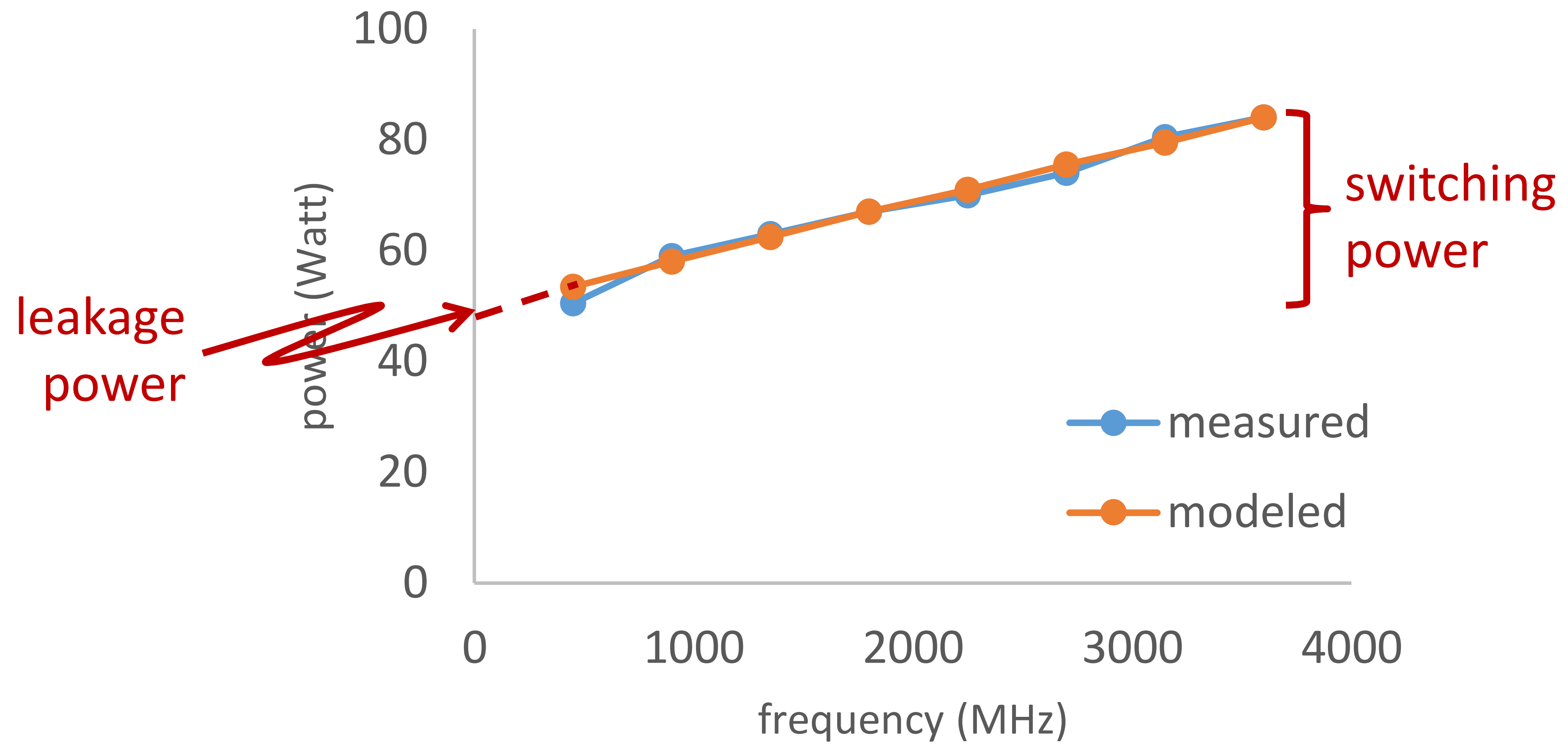
  *s.t.* $\quad k_{perf}' = k_{perf}\,\boldsymbol{s_{freq}}$

- $T' = Work\,/\,(k_{perf}\,\boldsymbol{s_{freq}})$
  - $1/\boldsymbol{s_{freq}}$ longer runtime
- $E' = (k_{switch} + k_{static}\,/\,(k_{perf}\,\boldsymbol{s_{freq}})\,)\cdot Work$
  - higher (leakage) energy due to longer runtime
- $P' = k_{switch}\cdot k_{perf}\,\boldsymbol{s_{freq}} + k_{static}$
  - lower (switching) power due to longer runtime

Not such a good idea

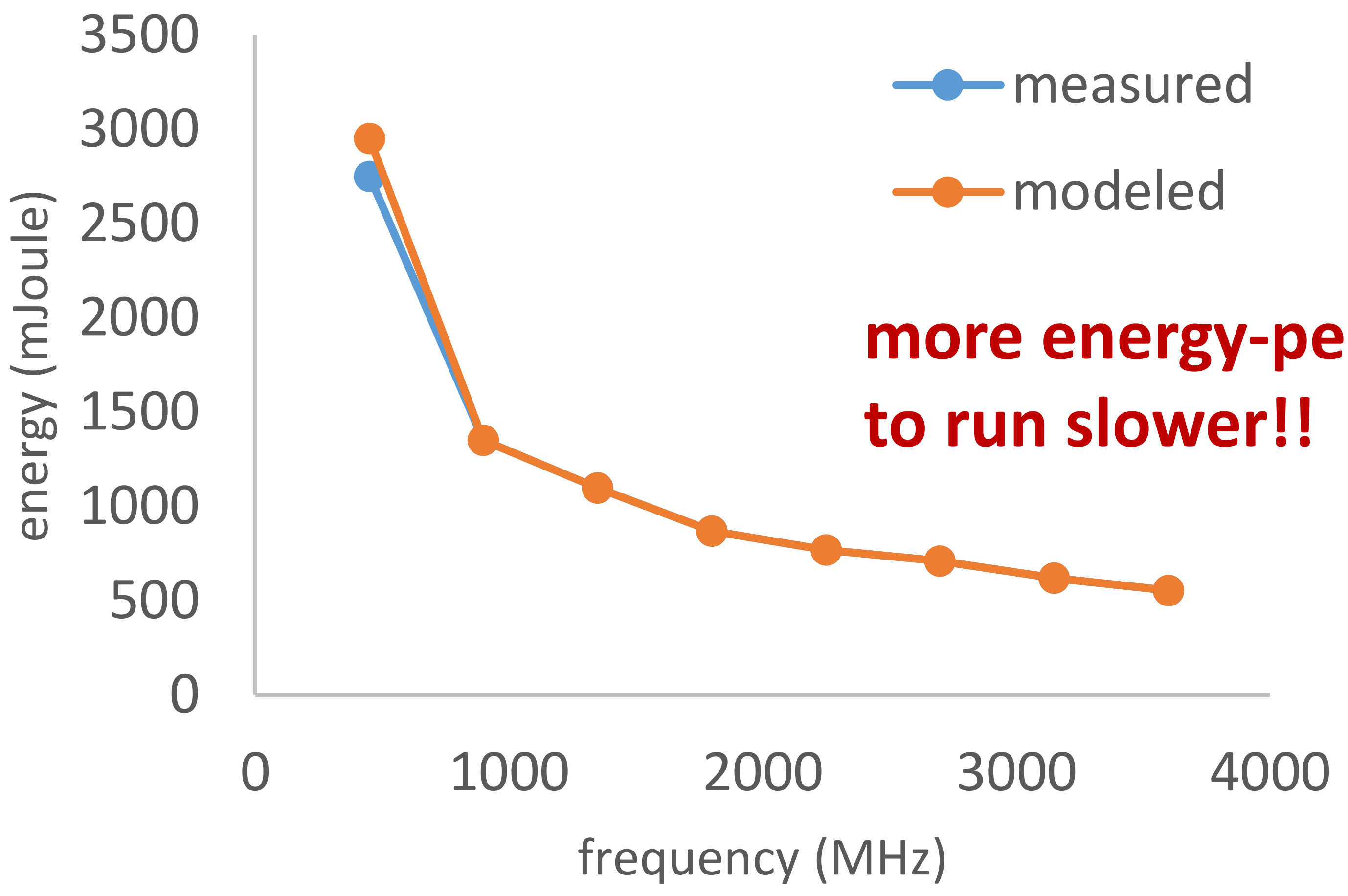
## Intel P4 660 Frequency Scaling: FFT$_{64K}$

circa 2005, 90nm



***k***$_{perf}$=145 FFT64K/sec; ***k***$_{switching}$=0.24 J/FFT64K; ***k***$_{static}$=49.4J/sec

# Intel P4 660 Frequency Scaling: $\text{FFT}_{64K}$



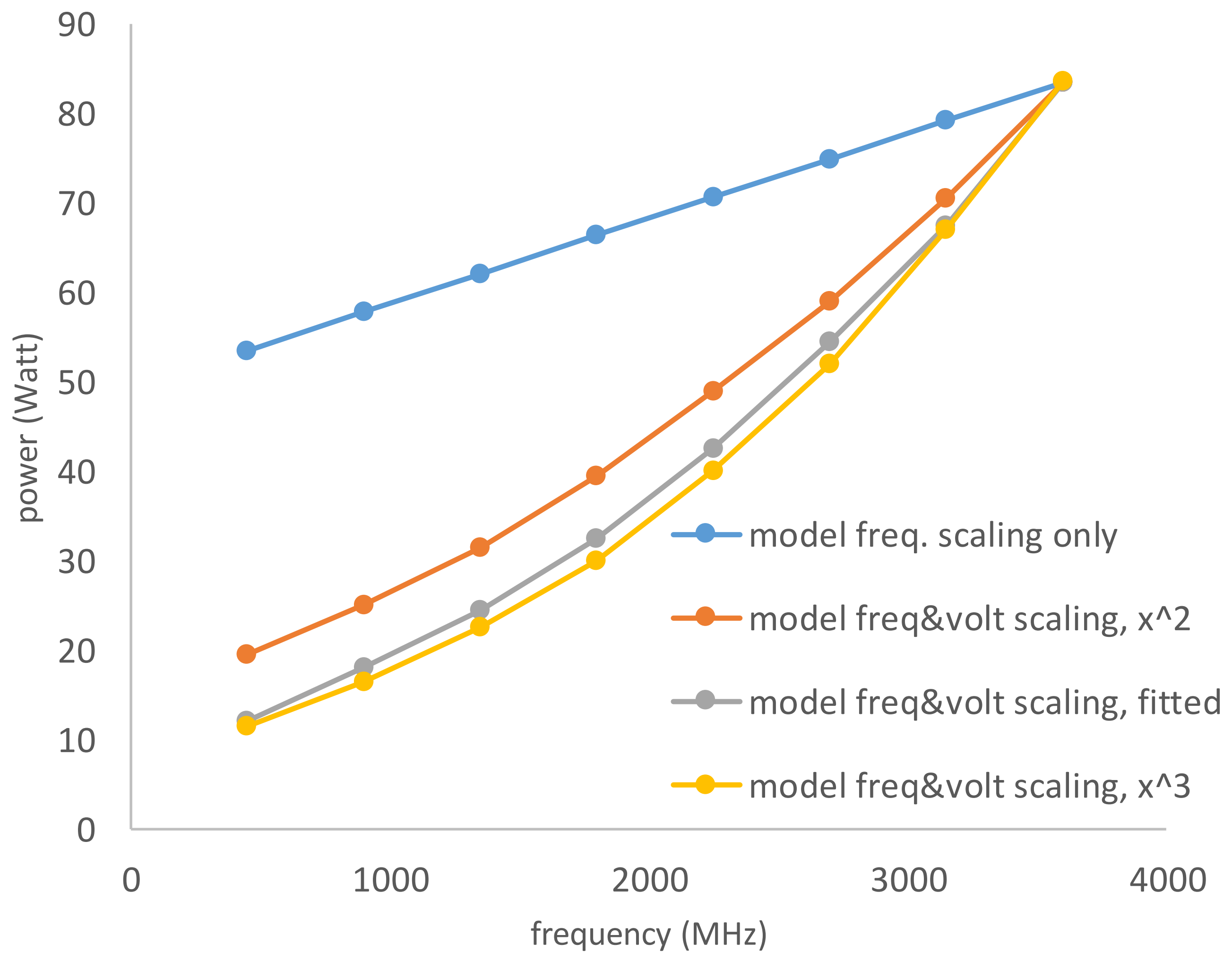
**more energy-per-fft to run slower!!**

$k_{perf}$=145 FFT64K/sec; $k_{switching}$=0.24 J/FFT64K; $k_{static}$=49.4J/sec

# Frequency + Voltage Scaling

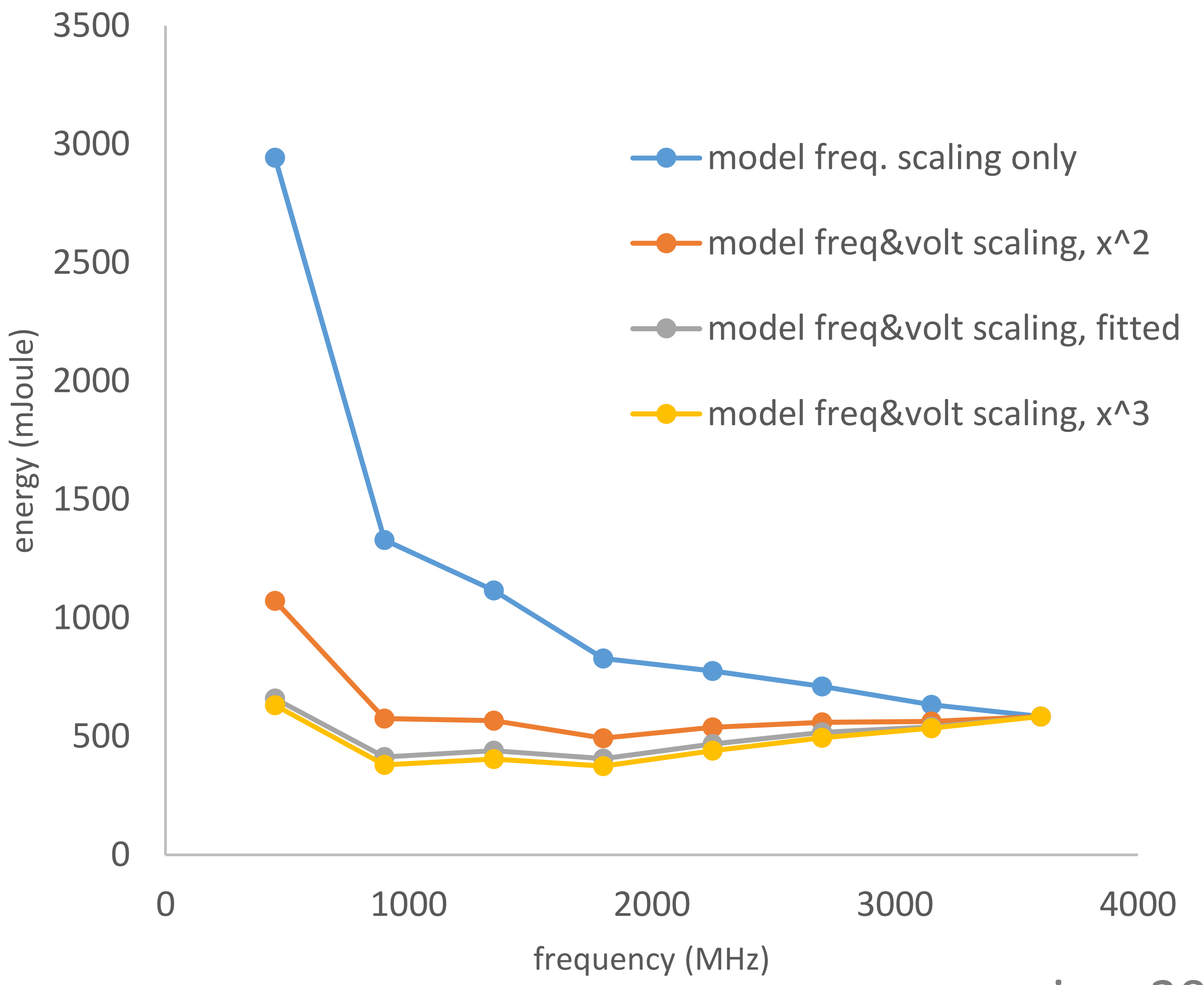
- Frequency scaling by $s_{freq}$ allows supply voltage to be scaled by a corresponding factor $s_{voltage}$
- $E \propto V^2$ thus $k_{switch}'' = k_{switch} \cdot s_{voltage}^{2}$
- $k_{static}'' = k_{static} \cdot s_{voltage}^{2\sim3}$ $\Leftarrow$ very gross approximation of something complicated
- $T'' = Work / (k_{perf} \cdot s_{freq})$
  - $1/s_{freq}$ longer runtime
- $E'' = (k_{switch} \cdot s_{voltage}^{2} + k_{static} \cdot s_{voltage}^{3} / k_{perf} \cdot s_{freq}) \cdot Work$
- $P'' = k_{switch} \cdot s_{voltage}^{2}\, k_{perf} \cdot s_{freq} + k_{static} \cdot s_{voltage}^{3}$
  - superlinear reduction in power and energy to performance degradation

# Intel P4 660 F+V Scaling: $\text{FFT}_{64K}$



circa 2005, 90nm

# Intel P4 660 F+V Scaling: $FFT_{64K}$

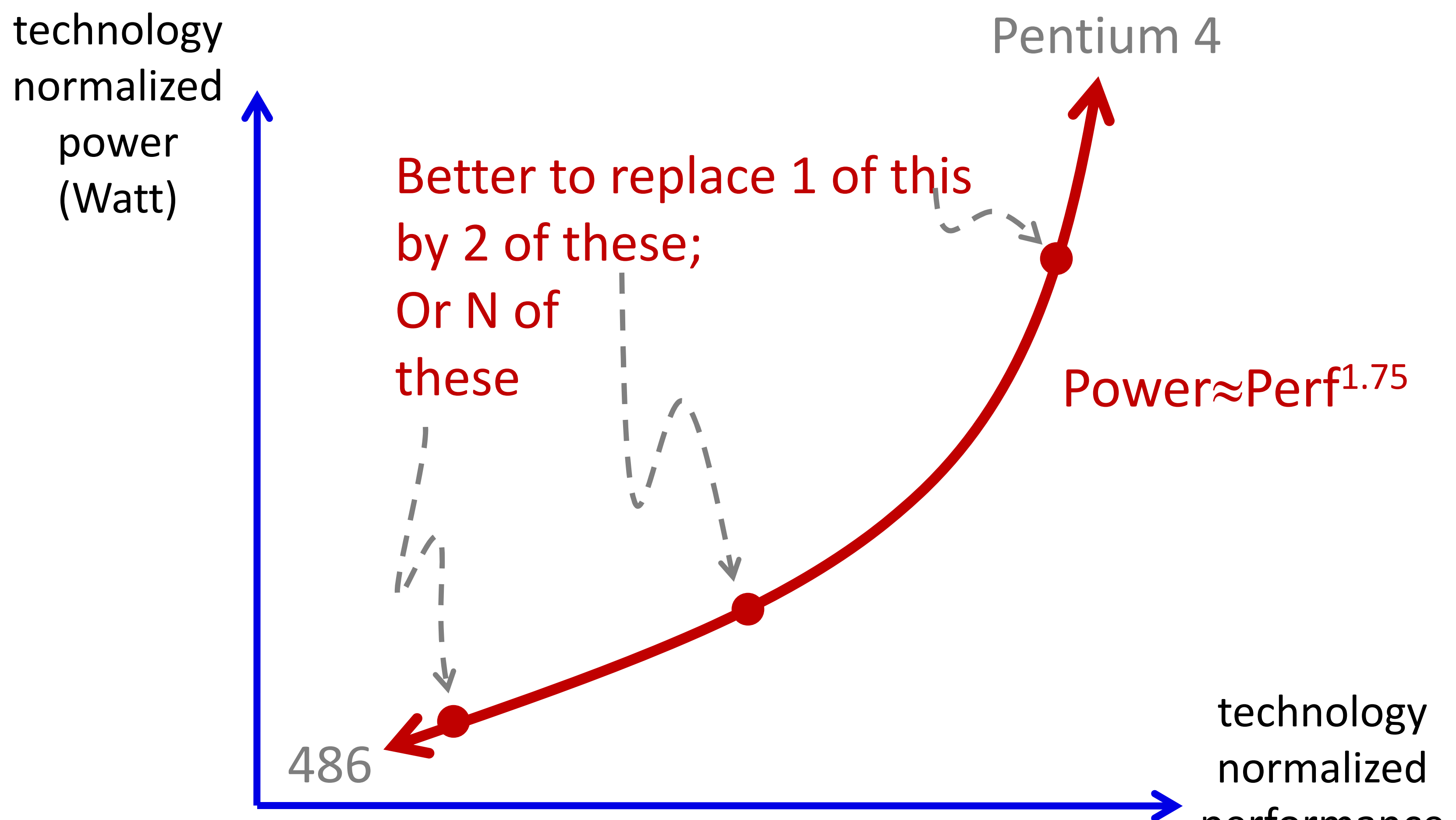


circa 2005, 90nm

## Parallelization:

**run <u>faster</u> at lower energy-per-op**
**by**
**running <u>slower</u> at lower energy-per-op**

# Cost of Performance in Power



[Energy per Instruction Trends in Intel® Microprocessors, Grochowski et al., 2006]

## Parallelization

- Ideal parallelization over ***N*** CPUs (to go fast)
  - $T = Work / (k_{perf} \cdot \boldsymbol{N})$
  - $E = (k_{switch} + k_{static} / k_{perf}) \cdot Work$

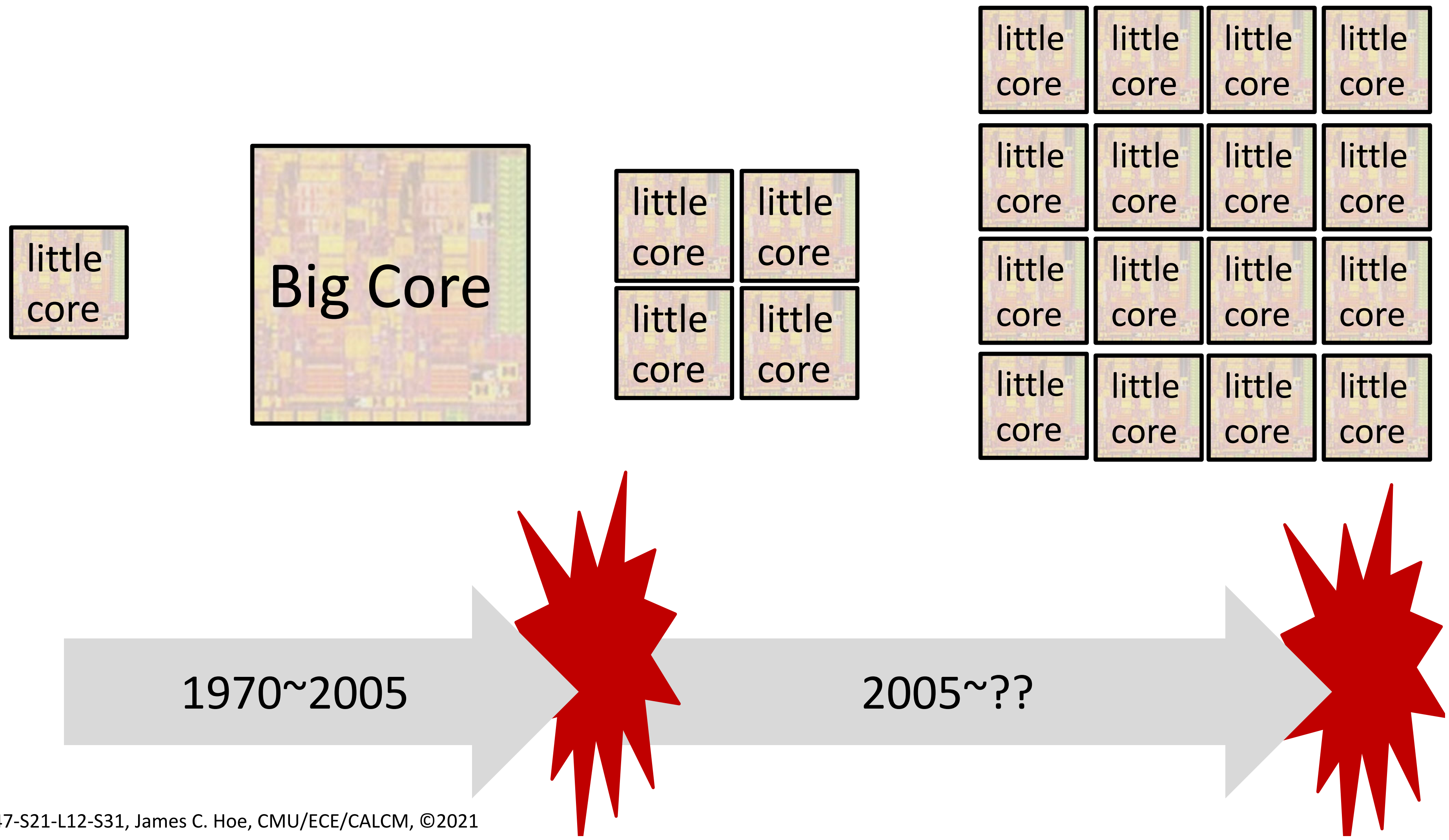
    ***N***-times static power, but ***N***-times faster runtime
  - $P = \boldsymbol{N}\ (k_{switch} \cdot k_{perf} + k_{static})$
- Alternatively, forfeit speedup for power and energy reduction by $s_{freq}=1/\boldsymbol{N}$ (assume $s_{voltage} \approx s_{freq}$ below)
  - $T = Work / k_{perf}$
  - $E'' = (k_{switch} / \boldsymbol{N}^2 + k_{static} / (k_{perf} \boldsymbol{N})) \cdot Work$
  - $P'' = k_{switch} \cdot k_{perf} / \boldsymbol{N}^2 + k_{static} / \boldsymbol{N}$
- Also works with using ***N*** slower-simpler CPUs

## So what is the problem?

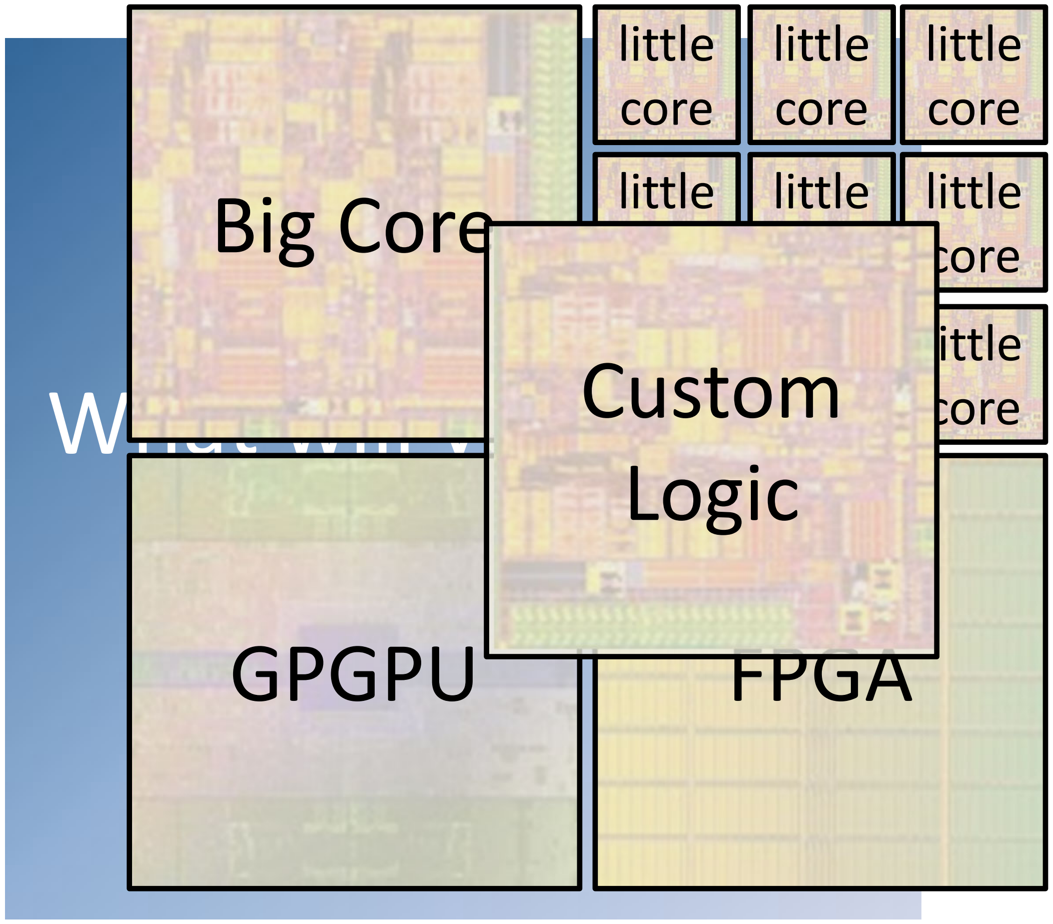
- "Easy" to pack more cores on a die to stay on Moore's law for "aggregate" or "throughput" performance
- How to use them?
  - life is good if your ***N*** units of work is ***N*** independent programs $\Rightarrow$ just run them
  - what if your ***N*** units of work is ***N*** operations of the same program? $\Rightarrow$ rewrite as parallel program
  - what if your ***N*** units of work is ***N*** sequentially dependent operations of the same program? $\Rightarrow$ ??

How many cores can you use up meaningfully?

# Moore's Law Scaling with Cores



# Remember: it is all about Perf/Watt and Ops/Joules



We talk about HW specialization in a later lecture