❑ LECTURE 5

- PDEs discretization limitations (timestep)
- ODEs
- Physical basis for Moore's scaling & the reason the technical civilization took off like a rocket in the 2nd half of 20th century
- Break-down of Moore's law and its consequences for the world of programming
- Supercomputing today
- Programming and parallelism
- ODEs

Literature: **see links on our course home page, the references page, and the coding page available from the course page.**

PDEs

- PDEs use stencils
- Depending on the stencil, there may be limitations on discretization steps
- The implicit schemes often avoid them
- Explicit schemes have timestep and/or spatial discretization limits
- Courant-Friedrichs-Levy is an example for wave-type and hydrodynamics equations

Von Neumann stability analysis of PDE algorithms:
diffusion equation

PDEs Example u - spreading quantity $u(x,y,z,t)$ (in 3D)

$$\frac{\partial u}{\partial t} = D\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2}\right) = D\nabla^2 u = D\Delta u$$

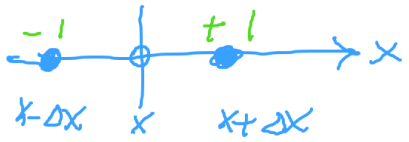
diff. coeff.: D, ν (alt. names)

$[D] = \frac{m^2}{s}$

Δ: Laplacian

Spatial differentiation

ex. $\frac{\partial u}{\partial x} \approx \frac{u(x+\Delta x) - u(x-\Delta x)}{2\Delta x}$

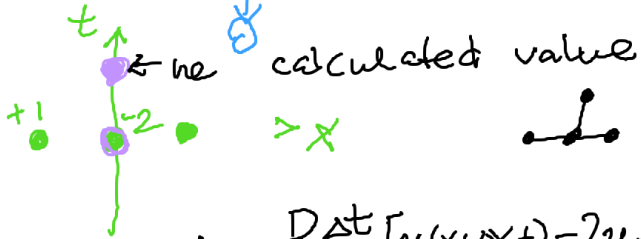


$$\frac{\partial^2 u}{\partial x^2} \approx \frac{1}{\Delta x}\left[\frac{u(x+\Delta x) - u(x)}{\Delta x} - \frac{u(x) - u(x-\Delta x)}{\Delta x}\right]$$

$$= \frac{1}{(\Delta x)^2}\left(u(x+\Delta x) - 2u(x) + u(x+\Delta x)\right)$$

stencils for heat eq.

(1-D)



$$\frac{\partial u}{\partial t} \approx \frac{u(t+\Delta t) - u(t)}{\Delta t}$$

$$u(x, \ldots, t+\Delta t) = \frac{D\Delta t}{(\Delta x)^2}\underbrace{\left[u(x+\Delta x, t) - 2u(x,t) + u(x-\Delta x, t)\right]}_{\text{old}} + \underbrace{u(x,t)}_{\text{old}}$$

$$\underbrace{u(x, t+\Delta t)}_{\text{new}} = u(x) + r\left(u(x+\Delta x) - 2u(x) + u(x-\Delta x)\right)$$

where $r := \frac{D\Delta t}{(\Delta x)^2}$

Von Neumann stability analysis of PDE algorithms:
diffusion equation (done below),
wave equation (exercise for you)

Von Neumann stability anal. for PDEs

$u = u^{num.} + \text{error } \epsilon$

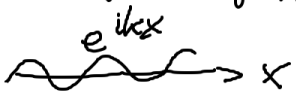
For linear eqs. ϵ: $\partial_t \epsilon = D \nabla^2 \epsilon$ (idea: Fourier)

discretize u_j^n ← timestep, ↓ spacial location

ϵ_j^n:

$$\epsilon_j^{n+1} = \epsilon_j^n + r\left(\epsilon_{j+1}^n - 2\epsilon_j^n + \epsilon_{j-1}^n\right)$$

$$\epsilon_j^n \sim E(t)\, e^{ikx}$$



$$e^{ikx} = \cos(kx) + i\sin(kx)$$

Plug that into PDE

$$E(t+\Delta t)\, e^{ikx} = E(t)\, e^{ikx} + r\left(E(t)\, e^{ik(x+\Delta x)} - 2E(t)\, e^{ikx} + E(t)\, e^{ik(x-\Delta x)}\right) \qquad \leq E(t)$$

Stability condition

$$|E(t+\Delta t)| \leq |E(t)| \iff \left|1 + r\left(e^{ik\Delta x} + e^{-ik\Delta x} - 2\right)\right| < 1$$

Trig: $\sin\frac{\theta}{2} = \frac{e^{i\theta/2} - e^{-i\theta/2}}{2i} \Rightarrow \sin^2\frac{\theta}{2} = \left(-\frac{1}{4}\right)\left(e^{i\theta} + e^{-i\theta} - 2\right)$

$\theta = k\Delta x$

Stab. crit:

$$\left|1 - 4r\, n \sin^2\frac{k\Delta x}{2}\right| < 1 \Rightarrow \boxed{r = \frac{D\Delta t}{(\Delta x)^2 n} < \frac{1}{2n}}$$

$n = \#\text{dim}$

$\sin^2\frac{k\Delta x}{2}$: $0 \ldots \frac{1}{2}$

$\Delta t < \frac{1}{2} \cdot \frac{(\Delta x)^2}{D}$ (1-d)

$\frac{\partial^2 u}{\partial t^2} = c^2 \nabla^2 u$

wave equation

Q: $\Delta t \leq$?

$\Delta t \leq$ □ $\frac{\Delta x}{c}$

Special theory of perturbations (numerical calculation)

Most popular numerical integration methods for **ODEs**

Euler method (1st order) & Runge-Kutta (2nd - 8th order)

Leonard Euler

Carle Runge

1856-1927

–

Martin Kutta

1867-1944

The Euler method

We want to approximate the solution of the differential equation

$$y'(t) = f(t, y(t)), \qquad y(t_0) = y_0, \qquad (1)$$

For instance, the Kepler problem which is a 2nd-order equation, can be turned into the 1st order equations by introducing double the number of equations and variables: e.g., instead of handling the second derivative of variable x, as in the Newton's equations of motion, one can integrate the first-order (=first derivative only) equations using variables x and $v_x = dx/dt$ (that latter definition becomes an additional equation to be integrated).

Starting with the differential equation (1), we replace the derivative *y'* by the finite difference approximation, which yields the following formula

which yields

$$y'(t) \approx \frac{y(t+h) - y(t)}{h}, \qquad (2)$$

$$y(t+h) \approx y(t) + hf(t, y(t)) \qquad (3).$$

This formula is usually applied in the following way.

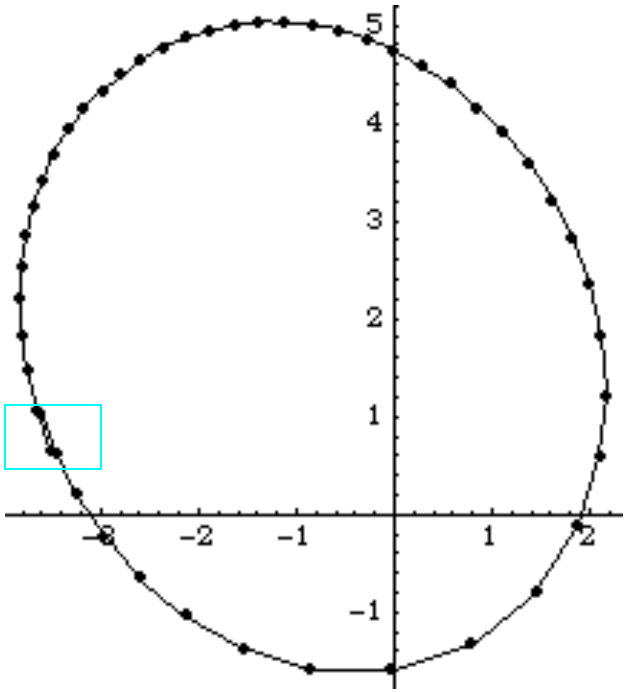
The Euler method (cont' d)

$$y(t+h) \approx y(t) + hf(t, y(t)) \qquad (3).$$

This formula is usually applied in the following way.
We choose a step size h, and we construct the sequence t_0, $t_1 = t_0 + h$, $t_2 = t_0 + 2h$, ... We denote by $\boldsymbol{y_n}$ a numerical estimate of the exact solution $y(t_n)$. Motivated by (3), we compute these estimates by the following recursive scheme

$$\boldsymbol{y_{n+1} = y_n + h\, f(t_n, y_n).}$$

This is the *Euler method* (1768), probably invented but not formalized earlier by Robert Hook.



It's a first (or second) order method, meaning that the total error is ~$h^{1\,(2)}$ It requires small time steps & has only moderate accuracy, but it's very simple!

The classical fourth-order Runge-Kutta method

One member of the family of Runge-Kutta methods is so commonly used, that it is often referred to as "RK4" or simply as "*the* Runge-Kutta method".

The RK4 method for the problem

$$y'(t) = f(t, y(t)), \qquad y(t_0) = y_0, \qquad (1)$$

is given by the following equation:

$$y_{n+1} = y_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

where

$$k_1 = f\left(t_n, y_n\right)$$

$$k_2 = f\left(t_n + \frac{h}{2}, y_n + \frac{h}{2}k_1\right)$$

$$k_3 = f\left(t_n + \frac{h}{2}, y_n + \frac{h}{2}k_2\right)$$

$$k_4 = f\left(t_n + h, y_n + hk_3\right)$$

Thus, the next value (y_{n+1}) is determined by the present value (y_n) plus the product of the size of the interval (h) and an estimated slope.

Runge-Kutta 4th order (cont' d)

Thus, the next value (y_{n+1}) is determined by the present value (y_n) plus the product of the size of the interval (h) and an estimated slope

$$\text{slope} = \frac{k_1 + 2k_2 + 2k_3 + k_4}{6}.$$

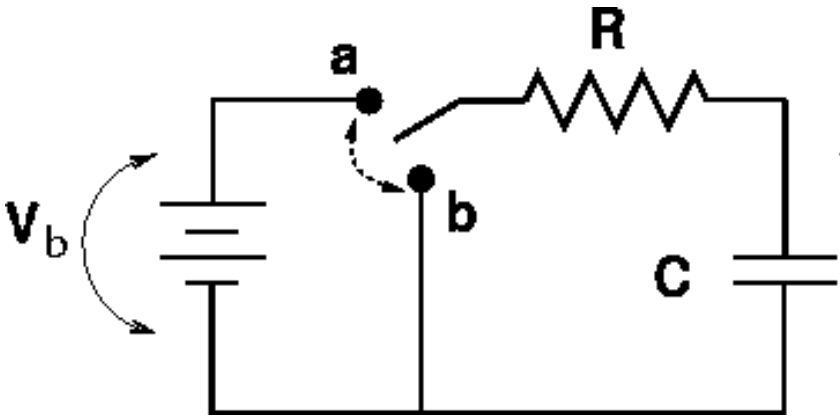
Thus, the next value (y_{n+1}) is determined by the present value (y_n) plus the product of the size of the interval (h) and an estimated slope. The slope is a weighted average of slopes:

- k_1 is the slope at the beginning of the interval;
- k_2 is the slope at the midpoint of the interval, using slope k_1 to determine the value of y at the point $t_n + h/2$ using Euler's method;
- k_3 is again the slope at the midpoint, but now using the slope k_2 to determine the y-value;
- k_4 is the slope at the end of the interval, with its y-value determined using k_3.

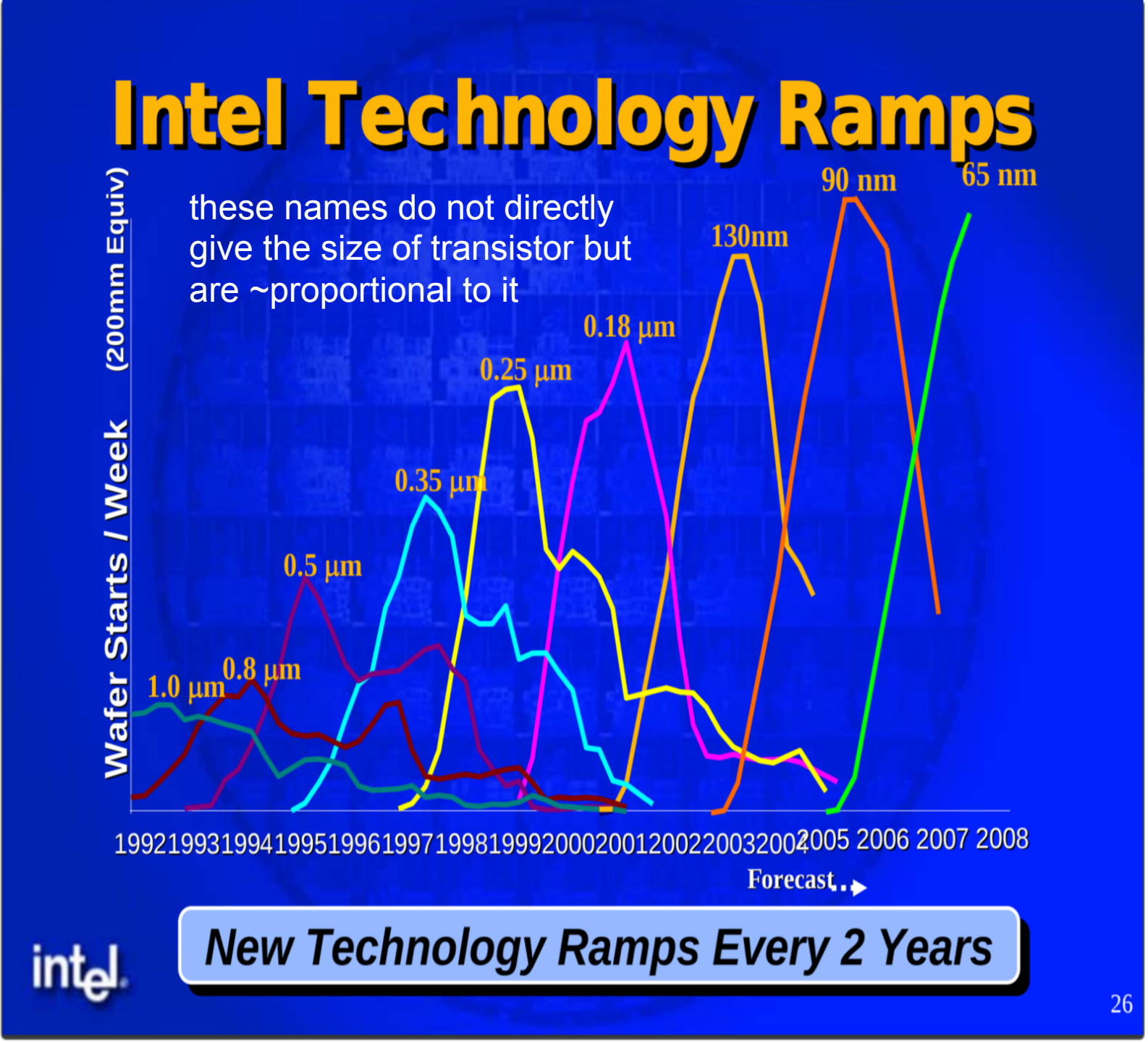
When the four slopes are averaged, more weight is given to the midpoint.

The RK4 method is a fourth-order method, meaning that the total error is ~h^4. It allows larger time steps & better accuracy than low-order methods.

Rise and present slow-down of the exponential growth of our TECHNICAL CIVILIZATION



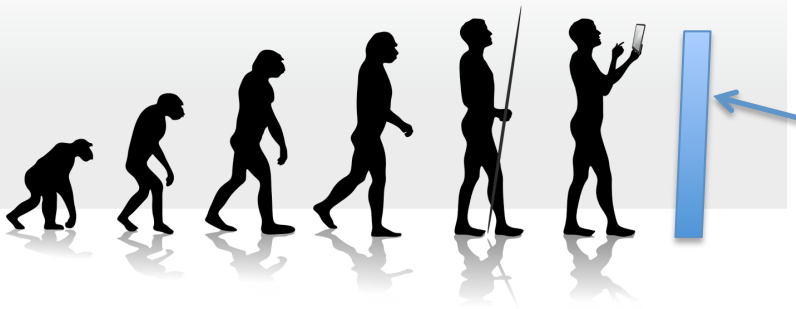
~10,000,000,000 transistors, acting like RC circuits, are etched in *every* CPU and GPU chip of a modern workstation (10^7 -10^8 /mm^2)



Name of technology cycle:

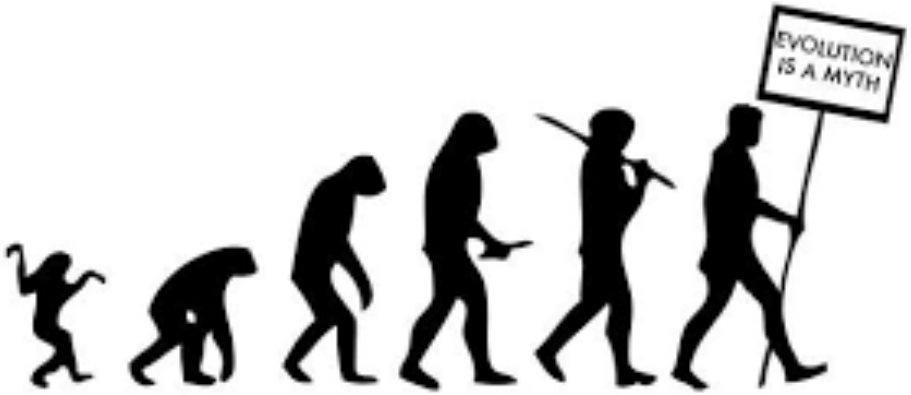
10 µm – 1971
6 µm – 1974
3 µm – 1977
1.5 µm – 1981
1 µm – 1984
800 nm – 1987
600 nm – 1990
350 nm – 1993
250 nm – 1996
180 nm – 1999
130 nm – 2001
90 nm – 2003
65 nm – 2005
45 nm – 2007
32 nm – 2009
22 nm – 2012
14 nm – 2014
10 nm – 2016
7 nm – 2019
5 nm – 2020+

Current density of transitors is of order 100 MT/mm^2

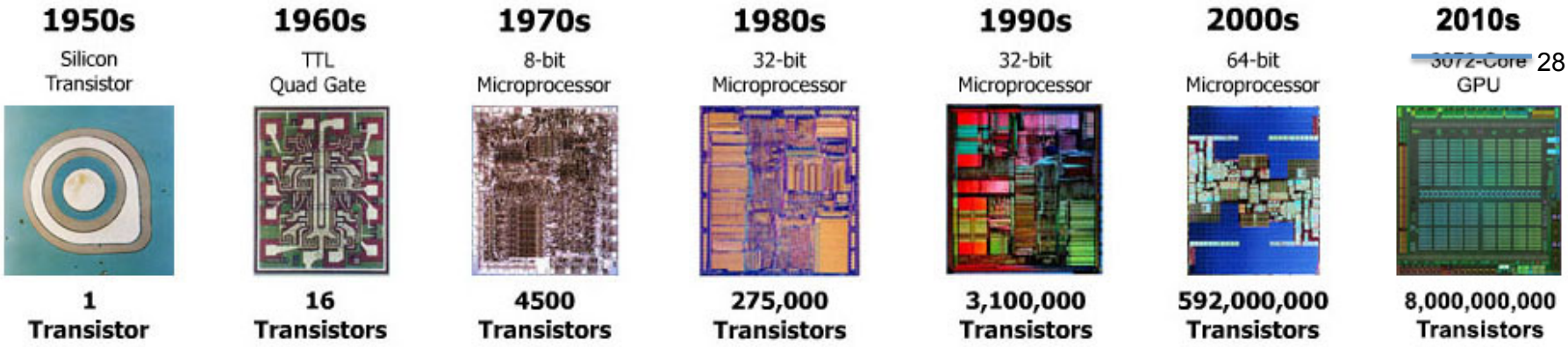


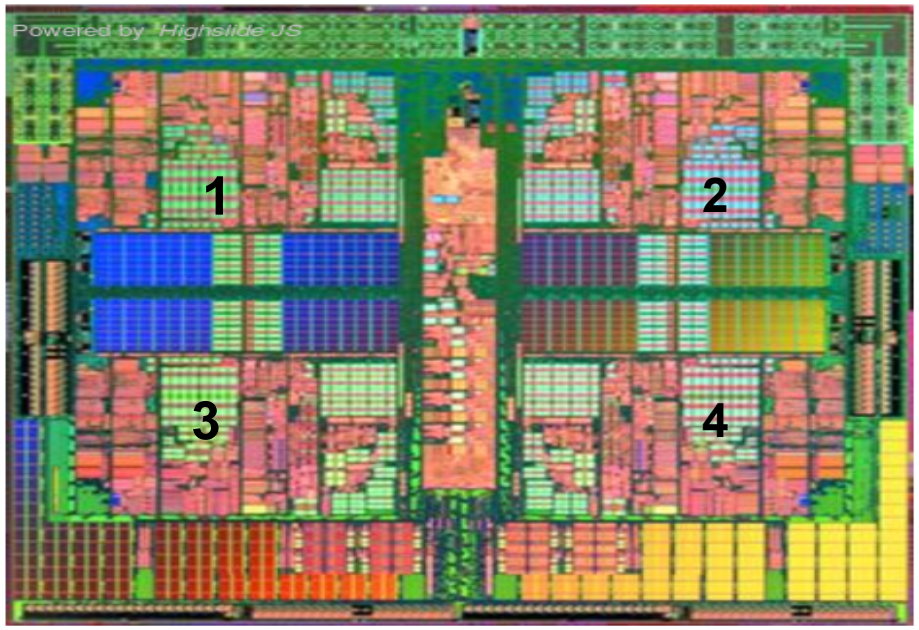
the wall in front of
homo smartphoniensis

RISE & CURRENT SLOWDOWN
of the exponential growth of our
TECHNICAL CIVILIZATION



1970s—2010s, the integrated circuit revolution
as a basis of **all our current technical civilization**: 40+ years of **Moore's law**

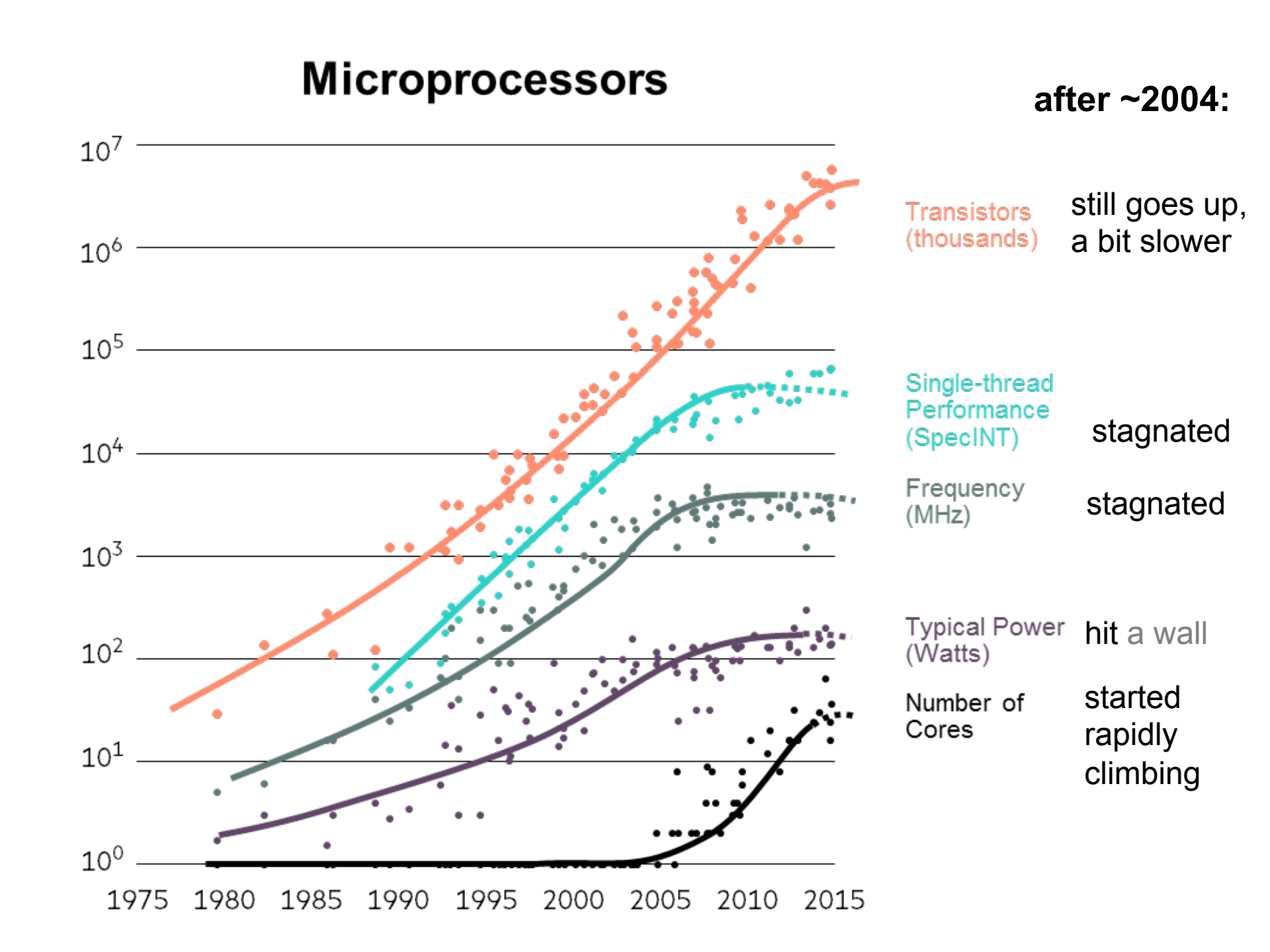




A highly integrated circuit (IC) has both arithmetic + logic units (ALU), fast cache memory hierachy, and communication circuits, all in a small package of a Central Processing Unit (CPU).

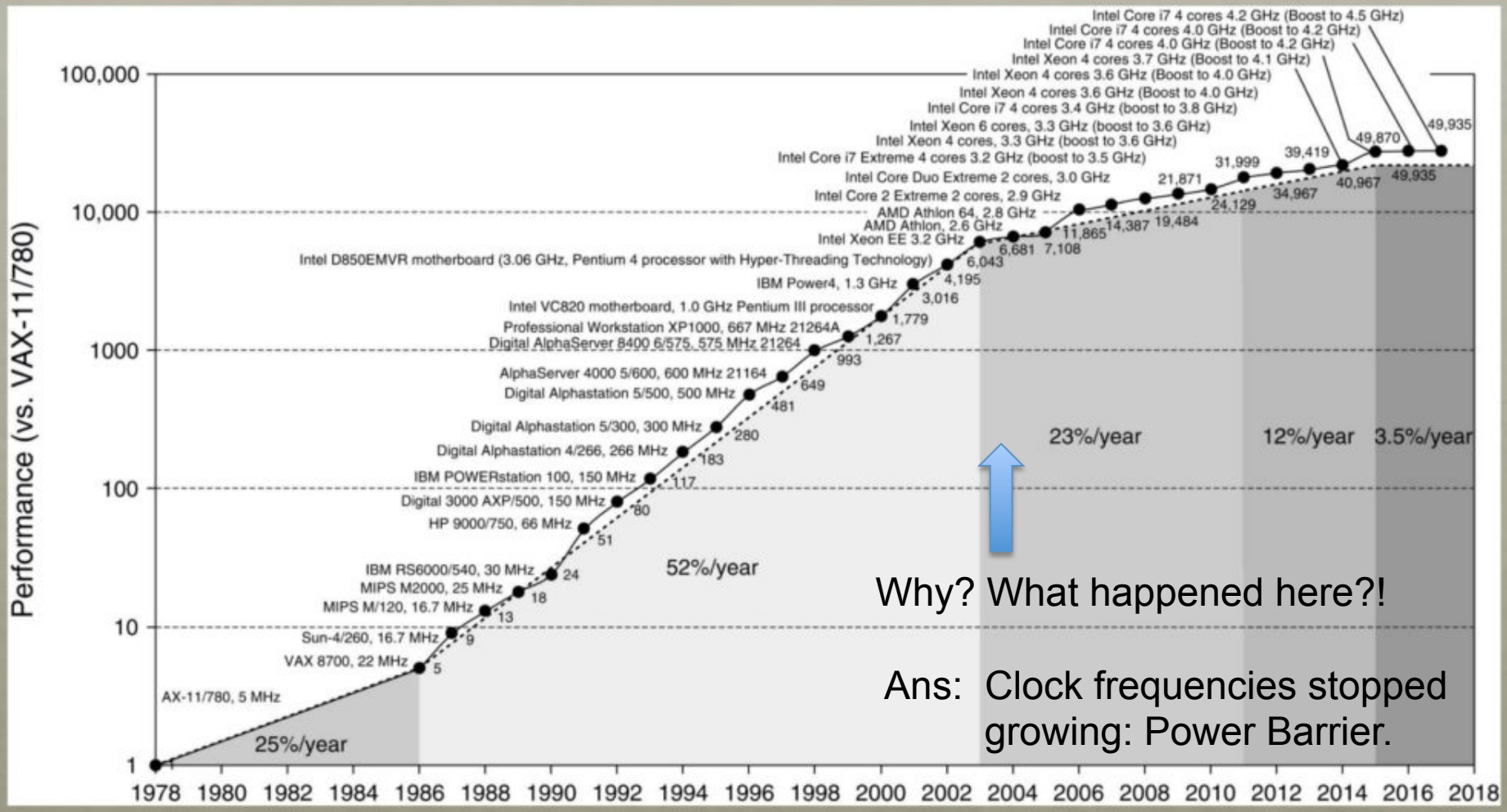
N = up to 10^{10} transistors

This processor looks like a 4-core CPU or old GPU (Central or Graphics Processing Unit).
It is not easy to execute one calculation simultaneously on many **cores (~ sub-CPUs)** of a CPU.
We call such processors **multicore**, and programs running on all cores **multithreaded**.


Microprocessors
after ~2004:
10^7
10^6
10^5
10^4
10^3
10^2
10^1
10^0
1975
1980
1985
1990
1995
2000
2005
2010
2015
Transistors (thousands)
still goes up, a bit slower
Single-thread Performance (SpecINT)
stagnated
Frequency (MHz)
stagnated
Typical Power (Watts)
hit a wall
Number of Cores
started rapidly climbing

CPU

UNIPROCESSOR PERFORMANCE (SINGLE CORE)

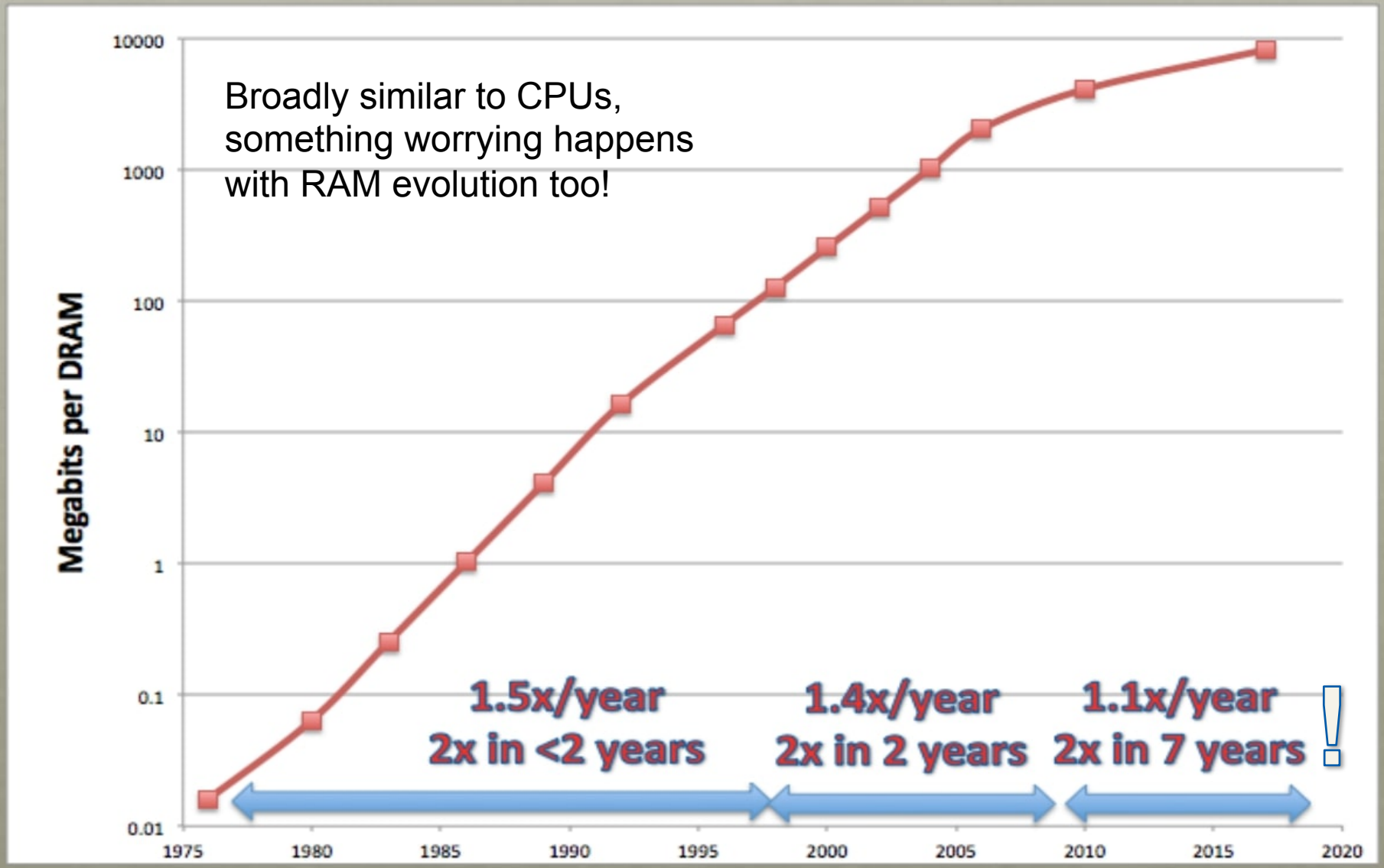


Why? What happened here?!

Ans: Clock frequencies stopped growing: Power Barrier.

Moore's Law in DRAMs (memory)

https://www.nextbigfuture.com/2019/02/the-end-of-moores-law-in-detail-and-starting-a-new-golden-age.html = a DARPA talk



Top 500 supercomputers in the world since ~2008 grow in speed slower than before