# Performance and Supercomputers

*CPE380, Spring 2026*

**Hank Dietz**

http://aggregate.org/hankd/

University of Kentucky

# Performance is about Choices

| Airplane | Passengers | Range (mi) | Speed (mph) |
|---|---|---|---|
| Boeing 737-100 | 101 | 630 | 598 |
| Boeing 747 | 470 | 4150 | 610 |
| BAC/Sud Concorde | 132 | 4000 | 1350 |
| Douglas DC-8-50 | 146 | 8720 | 544 |

- Execution time for application
- Power / temperature / battery life
- Reliability / availability
- Cost for acceptable functionality
- Size

# Response Time vs. Throughput

- Often can trade one for the other:
  Response Time: Time to complete an operation
  Throughput: Jobs completed per unit time

- Performance(X) = 1/ExecutionTime(X)
- X is Performance(X)/Performance(Y) times faster than Y, also:
  ExecutionTime(Y)/ExecutionTime(X)

# For Whom The Clock Ticks

- Posix uses real, user, system time
  - Real "Wall Clock" time always ticks
  - User time while in your code
  - System time while in OS code for you
- Multiplied by #PEs in multiprocessors
- I/O time not reported under Posix
- There are really lots of timing components
  - Processor tick count register
  - OS scheduler in 1-10ms Jiffies

# Running What?

- Different program, different performance
- Application (all that really matters!)
- “Toy” program
- Benchmark: representative application
- Micro Benchmark: tests a certain feature
- Synthetic Benchmark: a program written solely to perform like a particular application, but doing nothing useful
- Benchmark Suite: multiple benchmarks

# Modeling Time: CPI and IPC

- CPI is clock Cycles / Instruction
- IPC is Instructions / Cycle; i.e., 1/CPI
- Program runtime is:

  (Instructions executed / Program) *
  CPI * (Clock Period)
- Really sum over all instruction types because different instructions have different CPI

# An Example

| Instruction Type | Execution Count | CPI | Clock Period |
|---|---|---|---|
| A | 20 | 10 | 10ns |
| B | 10 | 30 | 10ns |

- This program takes:
  ((20*10)+(10*30))*10ns = 5us
- What can be changed to make it 4us?

# What Effects What?

| | Instruction Count | CPI | Clock Rate |
|---|---|---|---|
| Program (Algorithm) | Yes! | Indirectly... | No! |
| Compiler | Yes! | Indirectly... | Power? |
| ISA | Yes! | Yes! | Indirectly... |
| Impl. Arch. | uOps? | Yes! | Yes! |
| VLSI | No! | Indirectly... | Yes! |

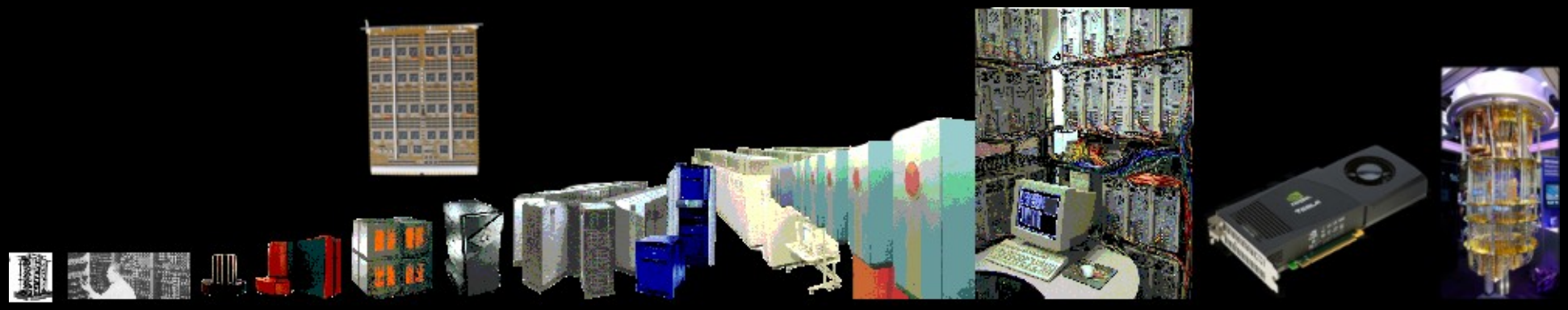
# Amdahl's Law

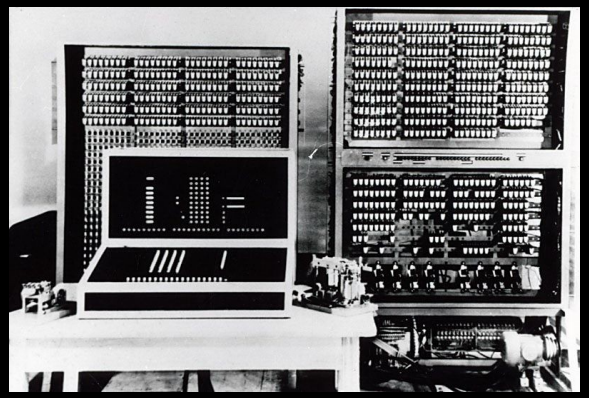
- If 1/N time is not affected by a change, the best possible speedup is only N
- Originally for sequential overhead in parallel code, but applies for any change

*Suppose a program spends 80% of its time doing multiplies... you can't get more than a 5X speedup by improving only multiplies!*

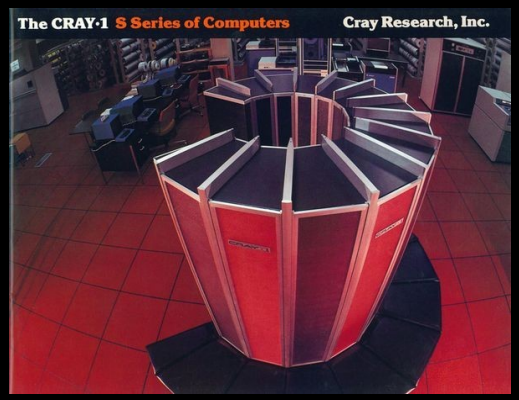
# What Is A Supercomputer?

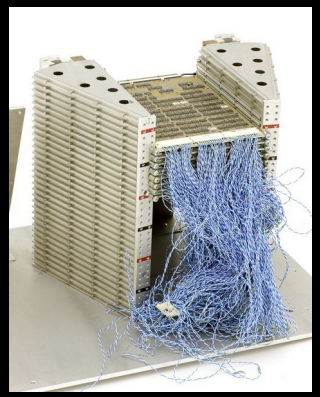
- Really two key characteristics:
  - Computer that solves big problems…
  - Performance can scale…
- The key is Parallel Processing…
  and modularity brings availability & reliability





- Before cheap transistors (– 1960s)
- E.g.: ENIAC
- Faster because machines are faster than humans
- Does one operation at a time because circuitry is too expensive to do more

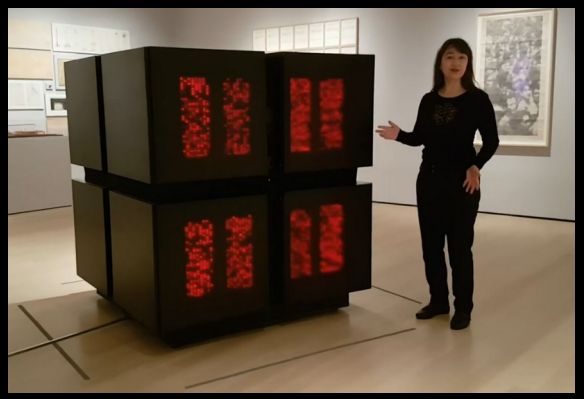


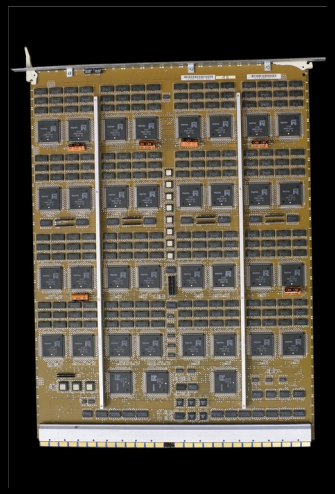


- Cheap transistors (1970s)
- E.g.: Cray 1
- Faster by applying the same operation to a vector of data at a time:

```
A[0..N] = B[0..N] + C[0..N]
```

- Can't change parallelism width without changing all register sizes and datapaths

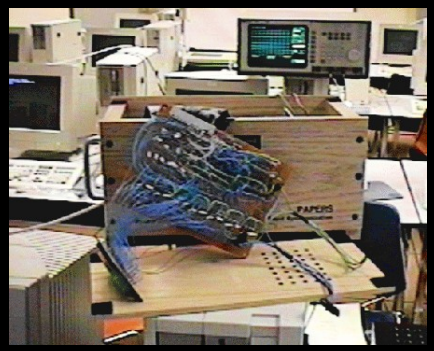
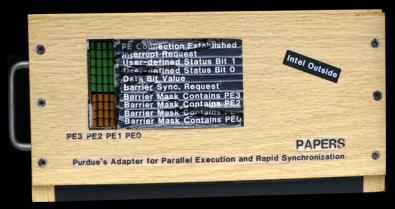


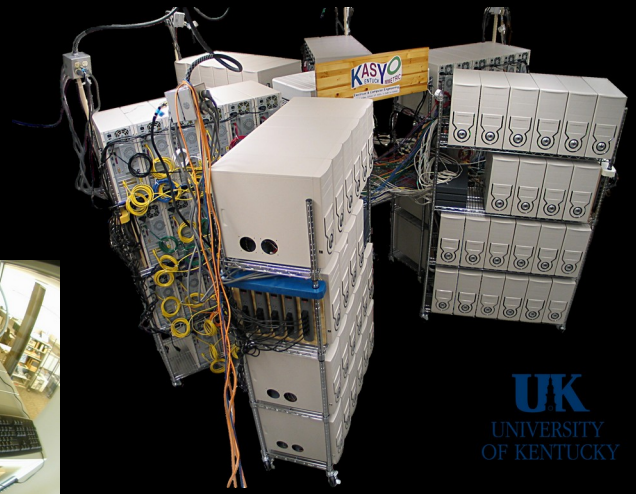


- Cheap chips with many transistors (1980s)
- E.g.: Thinking Machines Connection Machine
- Faster by applying the same operation to a vector of data at a time
- Modularly scalable SIMD (Single Instruction, Multiple Data) – can change parallelism simply by adding modules

- Cheap processors with custom memory
- E.g.: SGI Origin
- Faster by executing different code in parallel, using a custom shared memory to interact
- Modularly scalable MIMD (Multiple Instruction, Multiple Data) – can change parallelism simply by adding modules

- Cheap processors with custom network (1990s)
- E.g.: ASCI Red
- Faster by executing different code in parallel, using a custom messaging network to interact
- Modularly scalable MIMD (Multiple Instruction, Multiple Data) – can change parallelism simply by adding modules

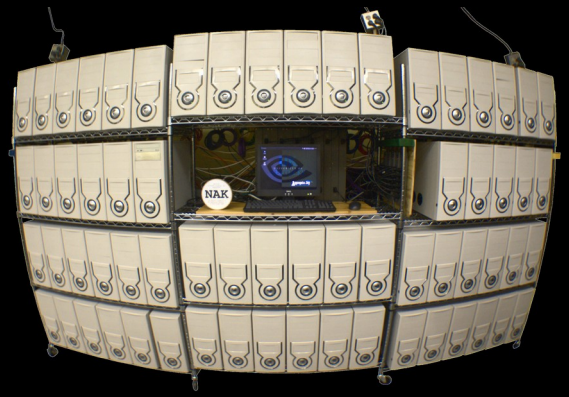






- Commodity parts and interfaces (1994 – )
- E.g.: Spareparticus, Beowulf, KLAT2, NAK
- Faster by executing different code in parallel, leverages commodity processors and network and also standard interfaces for custom parts
- Modularly scalable MIMD (Multiple Instruction, Multiple Data) – can change parallelism simply by adding modules

# Clusters And Bigger

- Mostly from interchangeable (PC) parts… and mostly running some form of Linux
- Cluster or Beowulf is a *parallel supercomputer* with tightly coupled, homogeneous, nodes
- Farm is homogeneous, colocated, machines with a common purpose (e.g., a render farm)
- Warehouse Scale Computer is a warehouse full of racked clusters used for *throughput*
- Grid is many internet-connected machines
- Cloud is *virtualized* grid/WSCs providing *services*




- GPUs (Graphics Processing Units) as cheap, fast, add-in cards in cluster nodes (2010 – )
- E.g.: NAK (and *most of the Top500* )
- The thing(s) on video cards... SIMDish, but:
  - Lots of little SIMDs (low fanout)
  - Multithreading to hide memory latency
  - Various restrictions to simplify HW
- NVIDIA CUDA and OpenCL...
- Intel's Xeon Phi is *sort-of* a GPU, but MIMDish
- GPU(s) will be on chip with SMP cores

# Types Of Hardware Parallelism

- Pipeline
- Superscalar, VLIW, EPIC
- SWAR (SIMD Within A Register)
- SMP (Symmetric MultiProcessor; multi-core)
- GPU (Graphics Processing Unit)
- Cluster
- Farm / Warehouse Scale Computer
- Grid / Cloud

Auto in processor; Manual; Manual across nodes

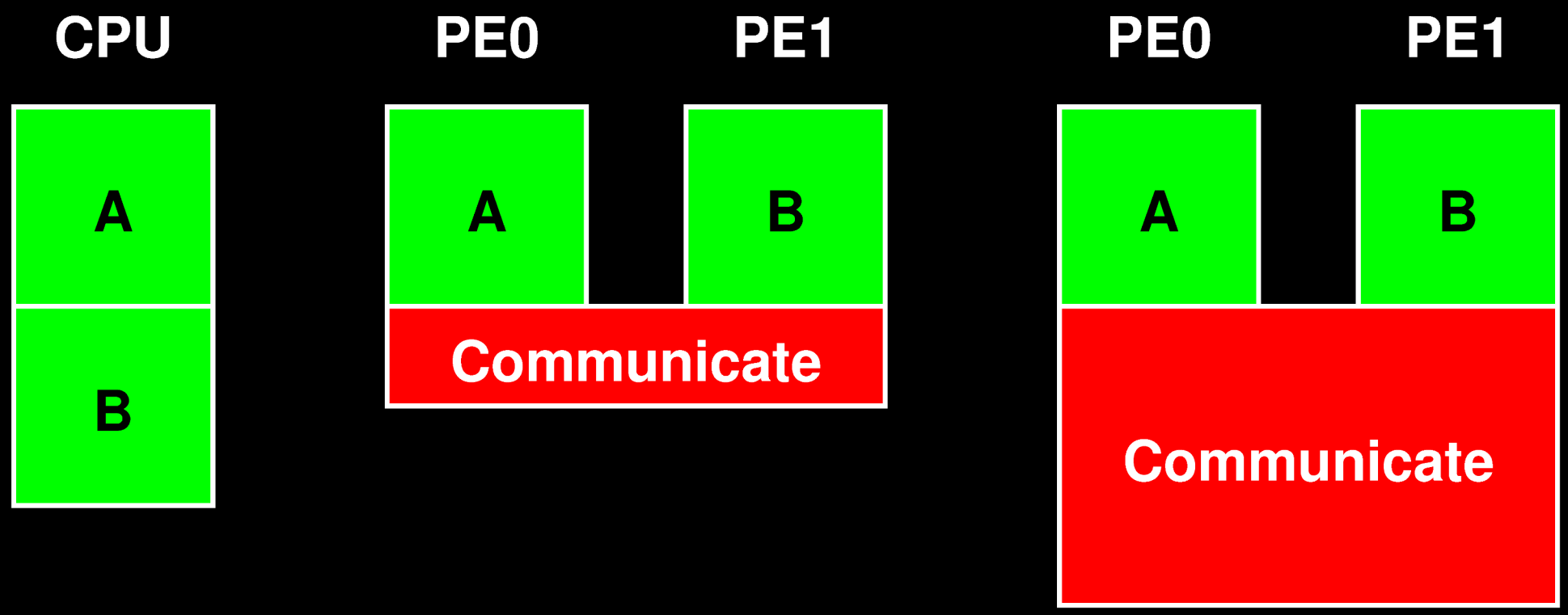
# Quantum?

- Quantum uses QuBits rather than Bits
- Superposition allows 0, 1, or *indeterminate*
- Entangled superposed QuBits can hold a PDF over all possible values simultaneously
  - E.g., 6 QuBits can hold {0, 1, 2, … 63}
  - Parallel processing without parallel hardware
- Reading a QuBit's value collapses superposition; you get only a single answer
- Could be the next big thing… or not

# Engineering an Interconnection Network

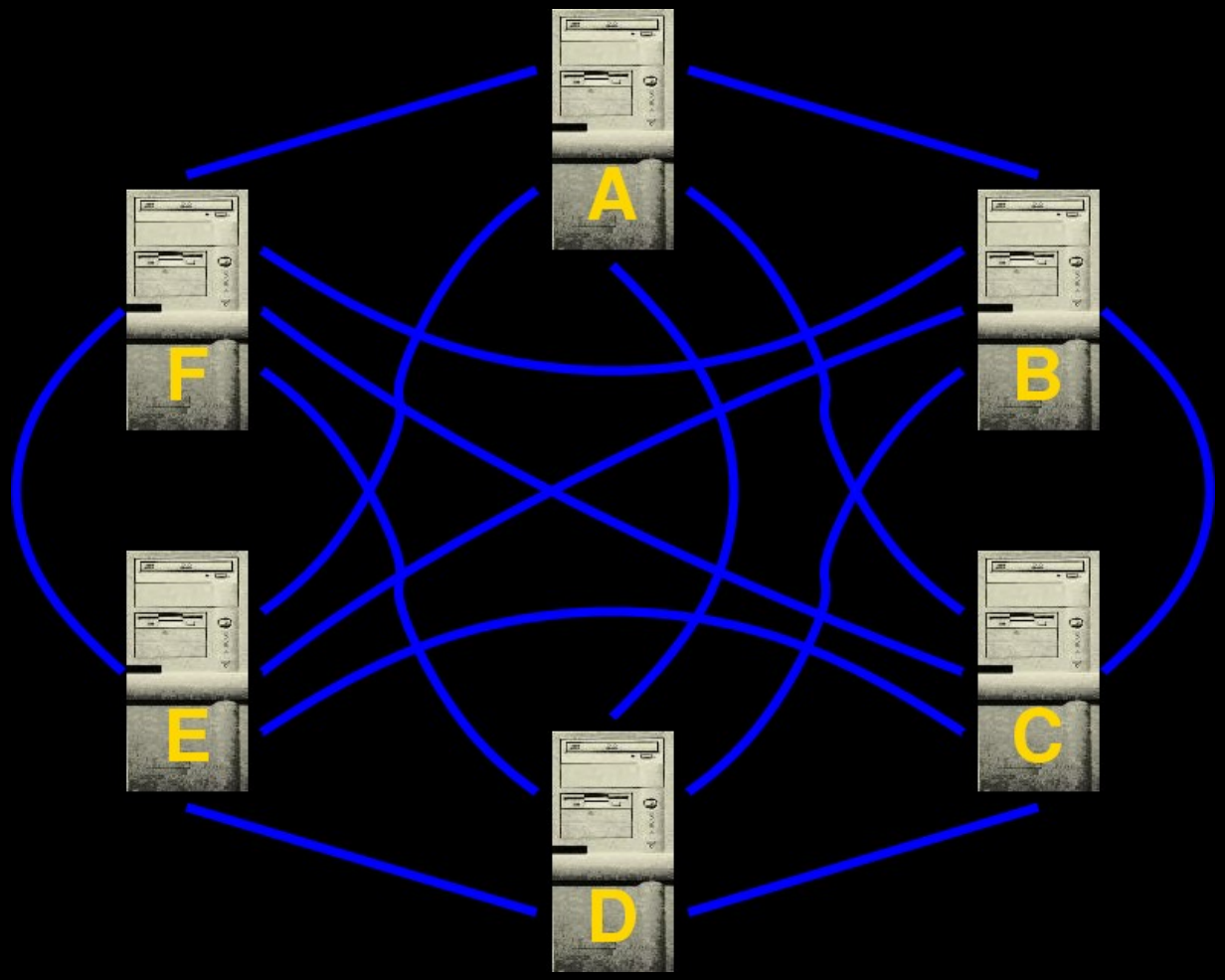
- Parallel supercomputer **nodes** interact
- **Bandwidth**
  - Bits transmitted per second
  - **Bisection Bandwidth** most important
- **Latency**
  - Time to send something here to there
  - Harder to improve than bandwidth
- We'll just consider topology here...

# Latency Determines Smallest Useful Parallel Grain Size



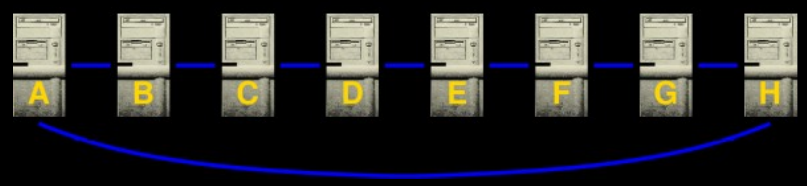
# No Network

# Direct Fully Connected

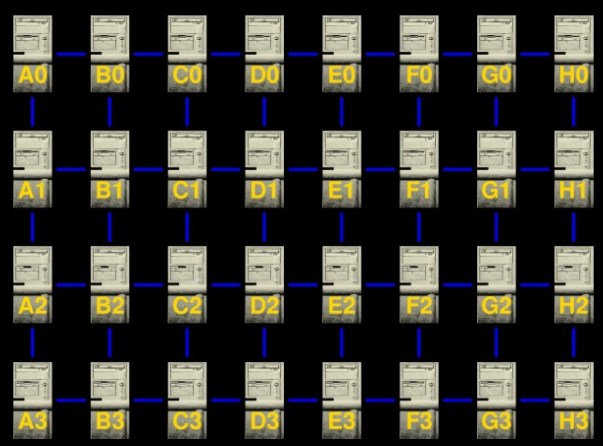


# Indirect Networks

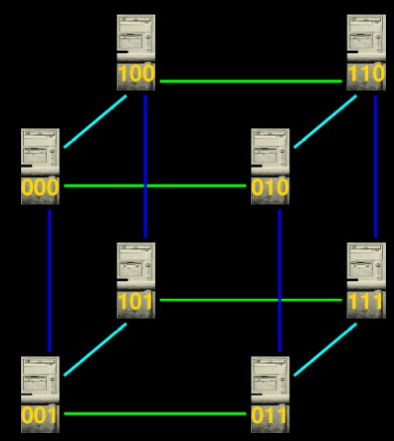
## Ring: Two Physical Layouts



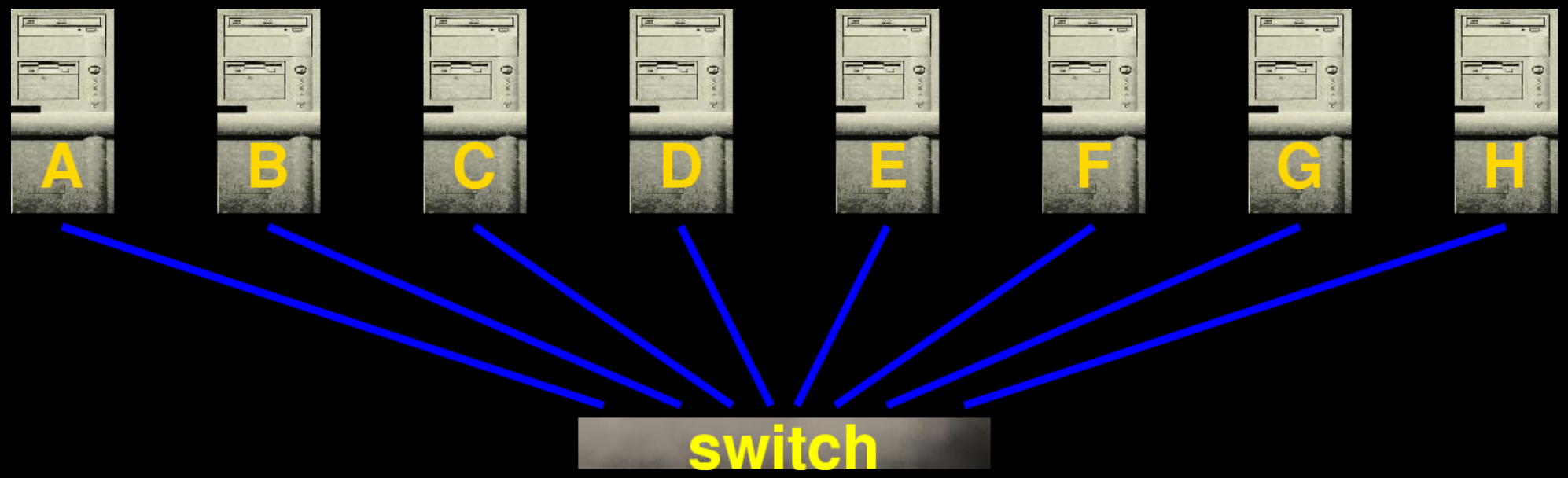
## Non-Toroidal 2D Mesh



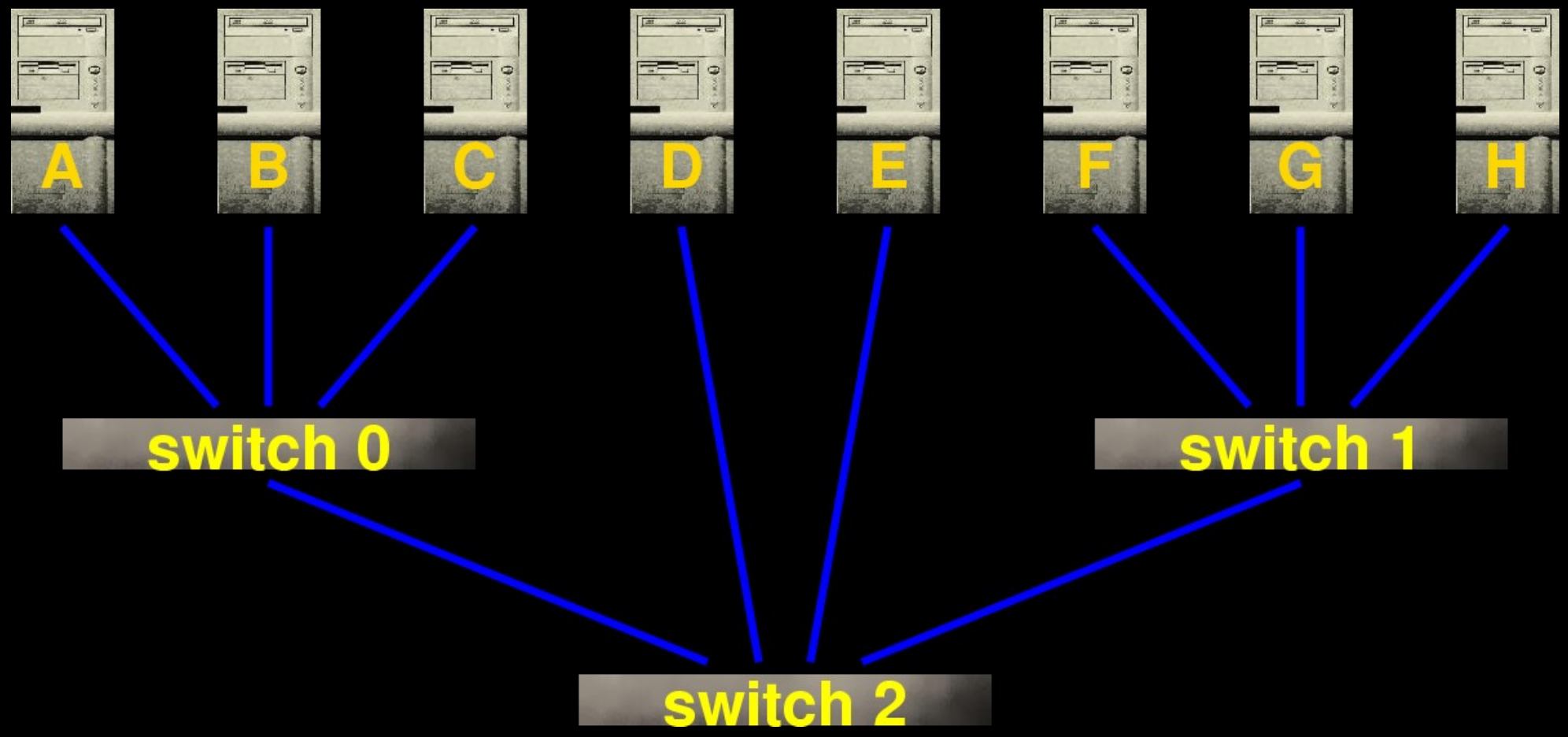

## 3-Cube (3D Toroidal Mesh)



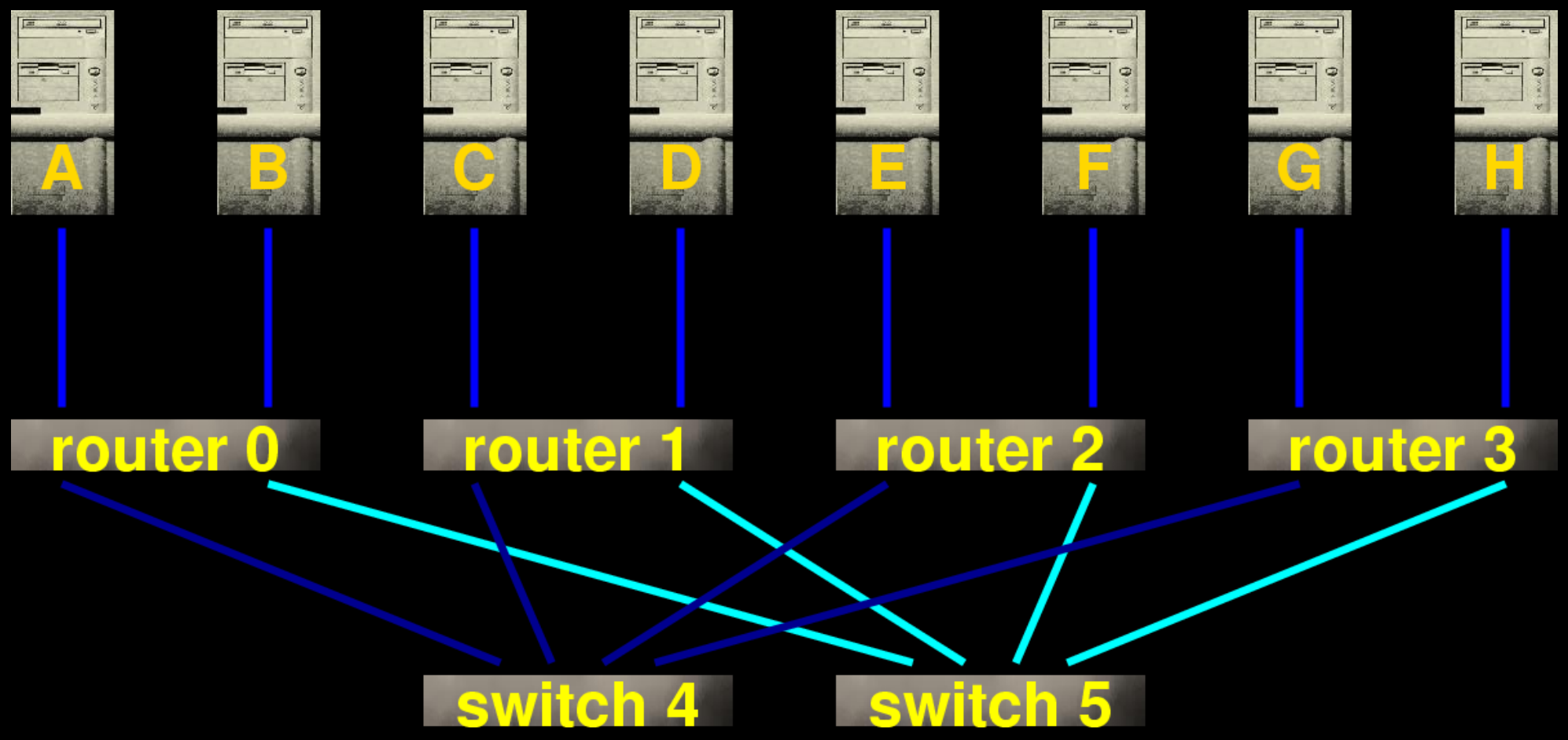
# Simple Switch (8-Port)



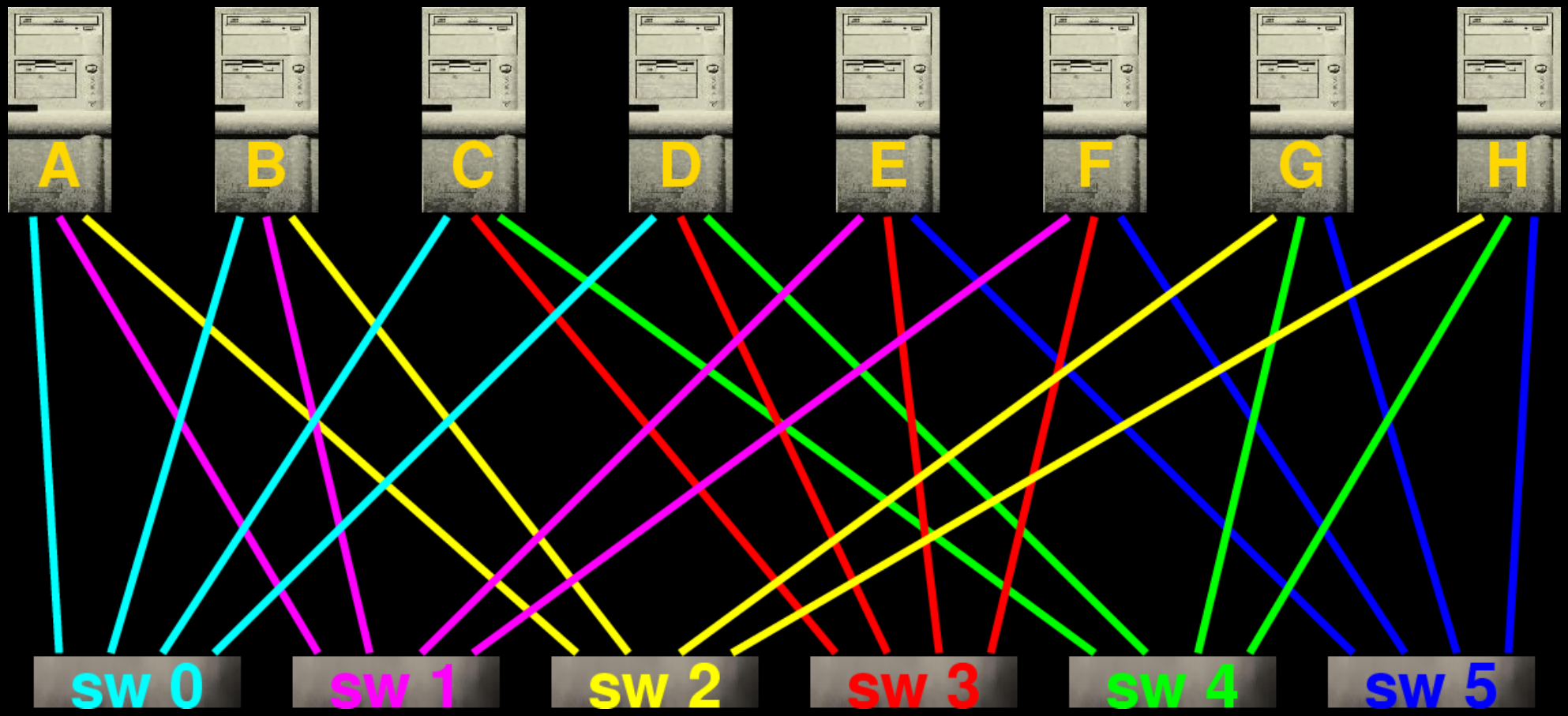
# A Maximum-Bisection Tree

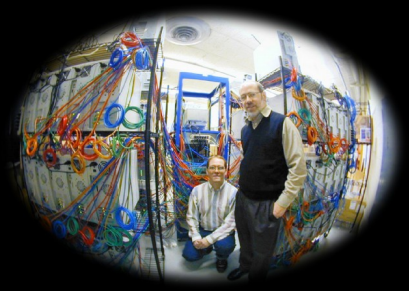


# Fat Tree


A
B
C
D
E
F
G
H
router 0
router 1
router 2
router 3
switch 4
switch 5

# Flat Neighborhood Network...
## from UK!





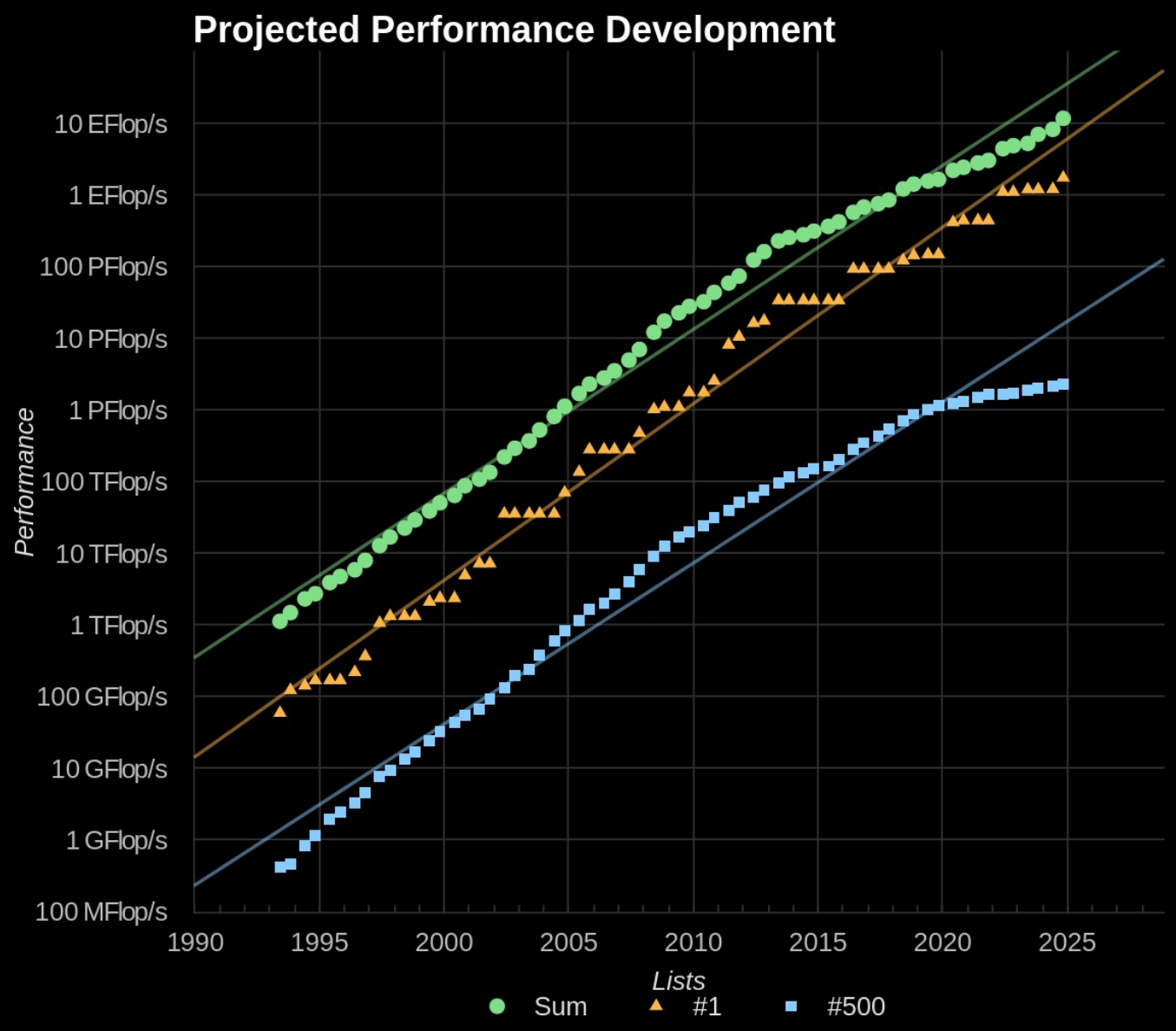
# A *Little* Progress

A GFLOPS is 1 Billion {+,*} per second

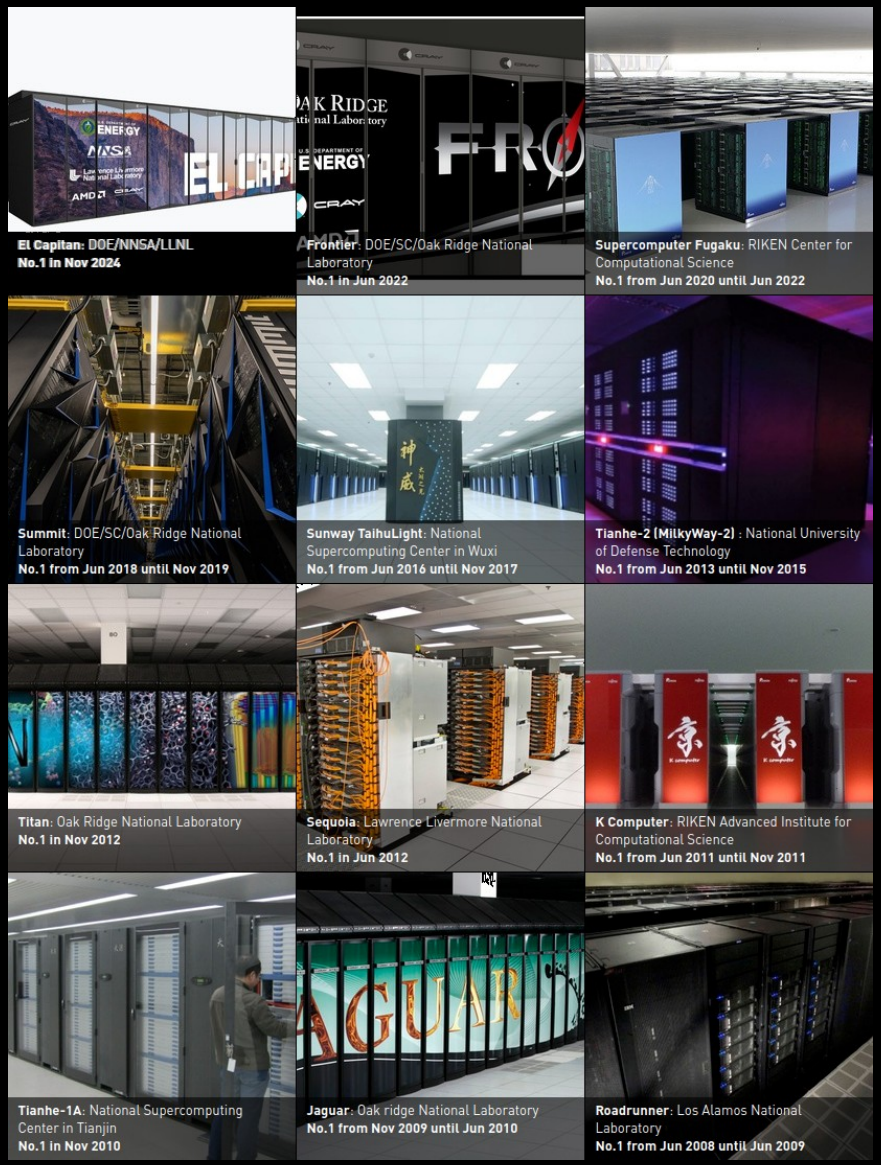
| | | |
|---|---|---|
| 1992 | MasPar MP1 | $1,000,000 / GFLOPS |
| 2000 | KLAT2 | $650 / GFLOPS |
| 2003 | KASY0 | $84 / GFLOPS |
| 2010 | NAK | $0.65 / GFLOPS |

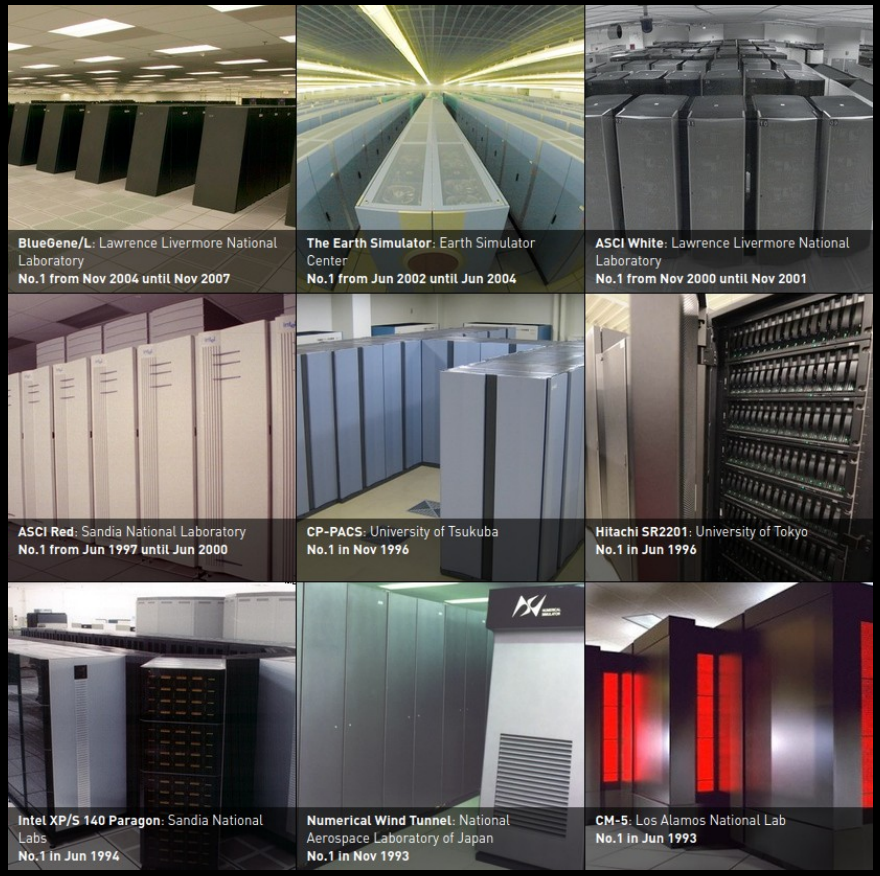
2022 GeForce RTX3090Ti peak is 40 TFLOPS @ $2K (not counting host)... $0.05 / GFLOPS

# A Lesson From `http://top500.org`



# #1 Machines, http://top500.org




**Now**: 11,039,616 cores, 1.742EFLOPS, 30MW

# The Future

- Everything is moving down...
  Your cell phone outruns the 1992 MasPar MP1; supercomputer today, in your cell phone soon
- More parallelism (and maybe **quantum** too?)
- More heterogeneous (helped by **dark silicon**)
- Everything contains a connected computer (e.g., IoT: Internet of Things)… a good thing?