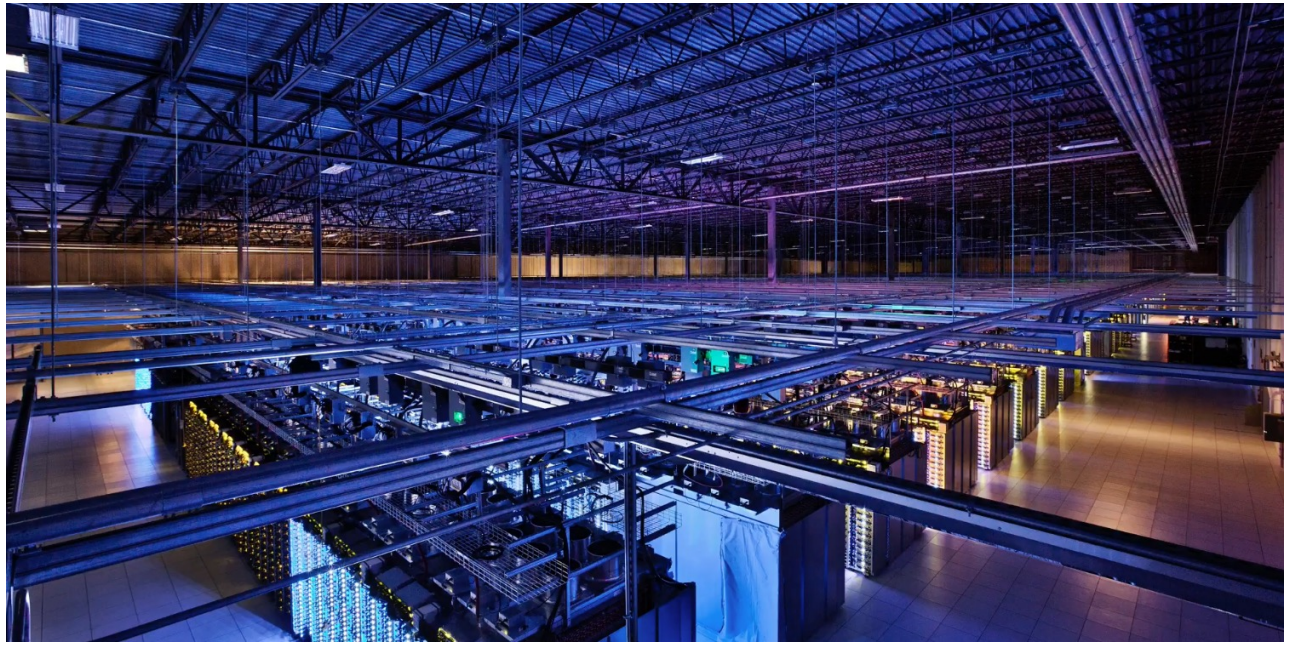
CS302

Scaling Trends



Inside Google's Datacenter

Spring 2025

Babak Falsafi, Arkaprava Basu

parsa.epfl.ch/course-info/cs302

Adapted from slides originally developed by Profs. Falsafi and Moshovos from EPFL/CMU and Toronto

Where are we?

M	T	W	T	F
17.Feb	18.Feb	19.Feb	20.Feb	21.Feb
24.Feb	25.Feb	26.Feb	27.Feb	28.Feb
03.Mar	04.Mar	05.Mar	06.Mar	07.Mar
10.Mar	11.Mar	12.Mar	13.Mar	14.Mar
17.Mar	18.Mar	19.Mar	20.Mar	21.Mar
24.Mar	25.Mar	26.Mar	27.Mar	28.Mar
31.Mar	01.Apr	02.Apr	03.Apr	04.Apr
07.Apr	08.Apr	09.Apr	10.Apr	11.Apr
14.Apr	15.Apr	16.Apr	17.Apr	18.Apr
21.Apr	22.Apr	23.Apr	24.Apr	25.Apr
28.Apr	29.Apr	30.Apr	01.May	02.May
05.May	06.May	07.May	08.May	09.May
12.May	13.May	14.May	15.May	16.May
19.May	20.May	21.May	22.May	23.May
	27.May	28.May	29.May	30.May

- Scaling Trends
  - Single processor performance
  - Power limitations
  - Future: Scaling up & out
- Thursday (29th May) is a holiday
- This Friday
  - **Final Exam**
  - Covers all course material excluding this lecture

Review: What We've Covered So Far

- Making use of hardware units → parallelism
- Example mechanisms:
  - Intel SSE/AVX → for SIMD programming
  - Language frameworks → OpenMP
- Three critical principles of parallel computing
  - Finding enough parallelism
  - Work division and balance
  - Communication and synchronization

Hardware Supporting Parallelism

- Cache coherence and memory models
  - Communicating data in shared memory
- Synchronization & TM
  - Synchronization and data access control
- Multithreading and GPUs
  - Specifically designed for parallel execution

In this Lecture

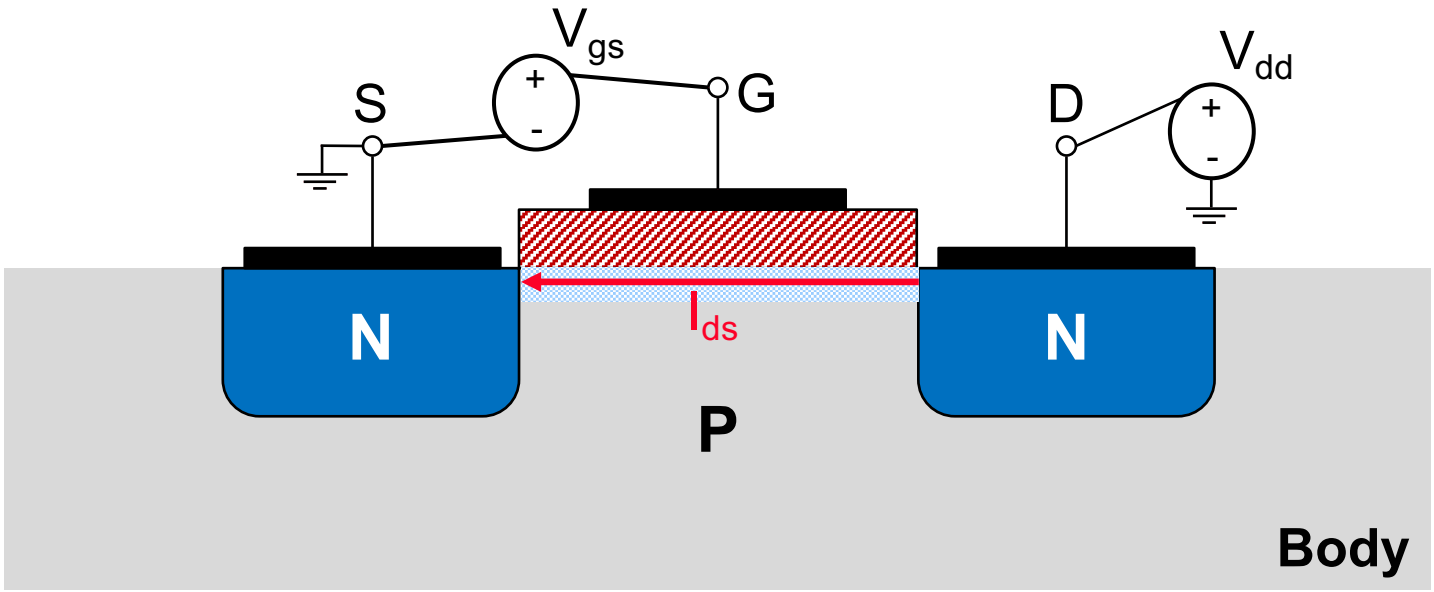
- Why parallel hardware exists in the first place…
  - Moore's Law
  - Dennard Scaling and its demise
  - Increasing compute without drawing too much power
- And, where the field is going in the future…
  - Programming/orchestrating many multiprocessors
  - Datacenters

Introduction: What is a Transistor?

- For our purposes, it works like a switch
  - Either **on** → current flowing
  - Or **off** → no current
- The **gate** controls whether it is on or off
  - Applying a voltage at the input, turns transistor on
  - Otherwise, it is off
- Simple, right?

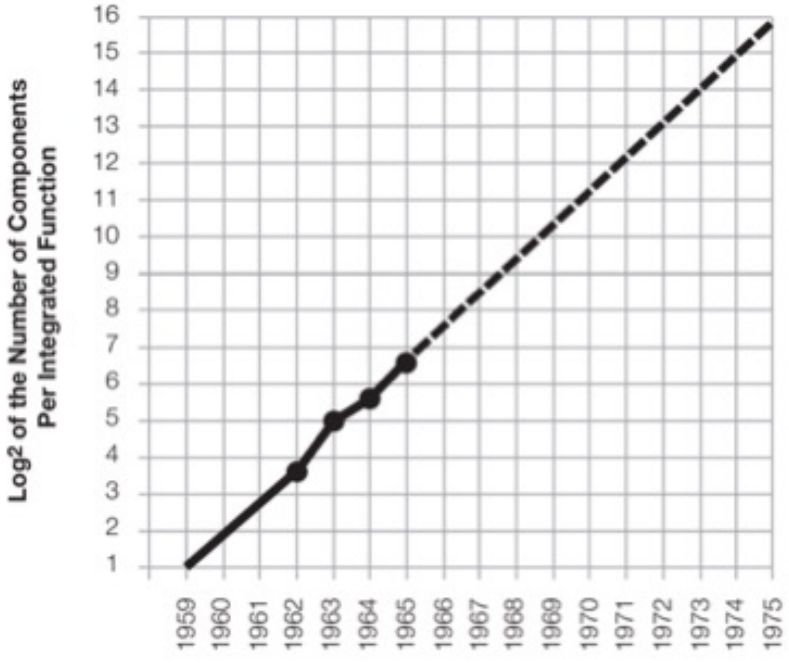
How Transistors Work

- How to get current to flow?
  - Think of it like needing a “wire” between D & S
  - Add a voltage on G, making an N-region
    - This only occurs above a “threshold” V_{th}
  - Connects D&S, voltage difference makes current flow

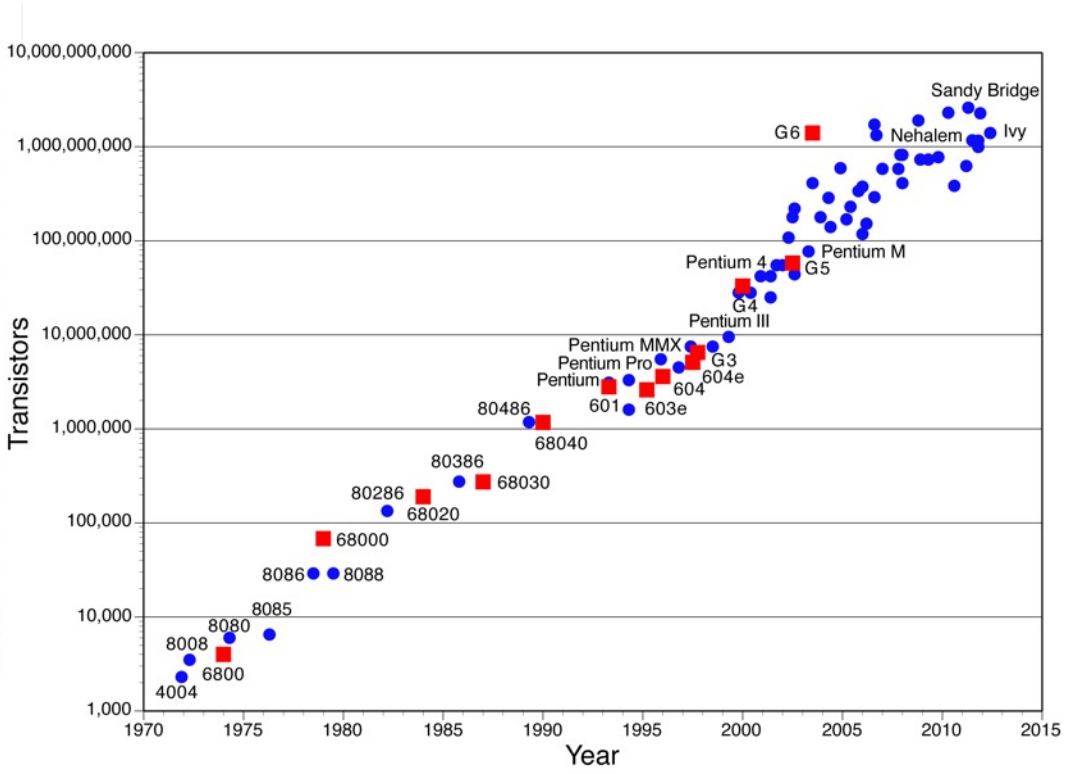


Moore's ~~Prediction~~ Law

- Started with very few data points, but predicted over 50 years of IC manufacturing
  - Number of transistors doubles every 2 years



Original Graph



… Over 50 years

Dennard Scaling Recipe

- Reduce channel length every 2 years by factor $\kappa \sim 0.7$
  - Increase frequency by κ, reduce capacitance and voltage by κ
  - $\mathrm{P} \simeq CV^2 f$
  - Power per transistor $\simeq (1/k)(1/k^2)\ k = 1/k^2$
- The same transistors take up $1/k^2$ area
  - Importantly, **power density stays constant**

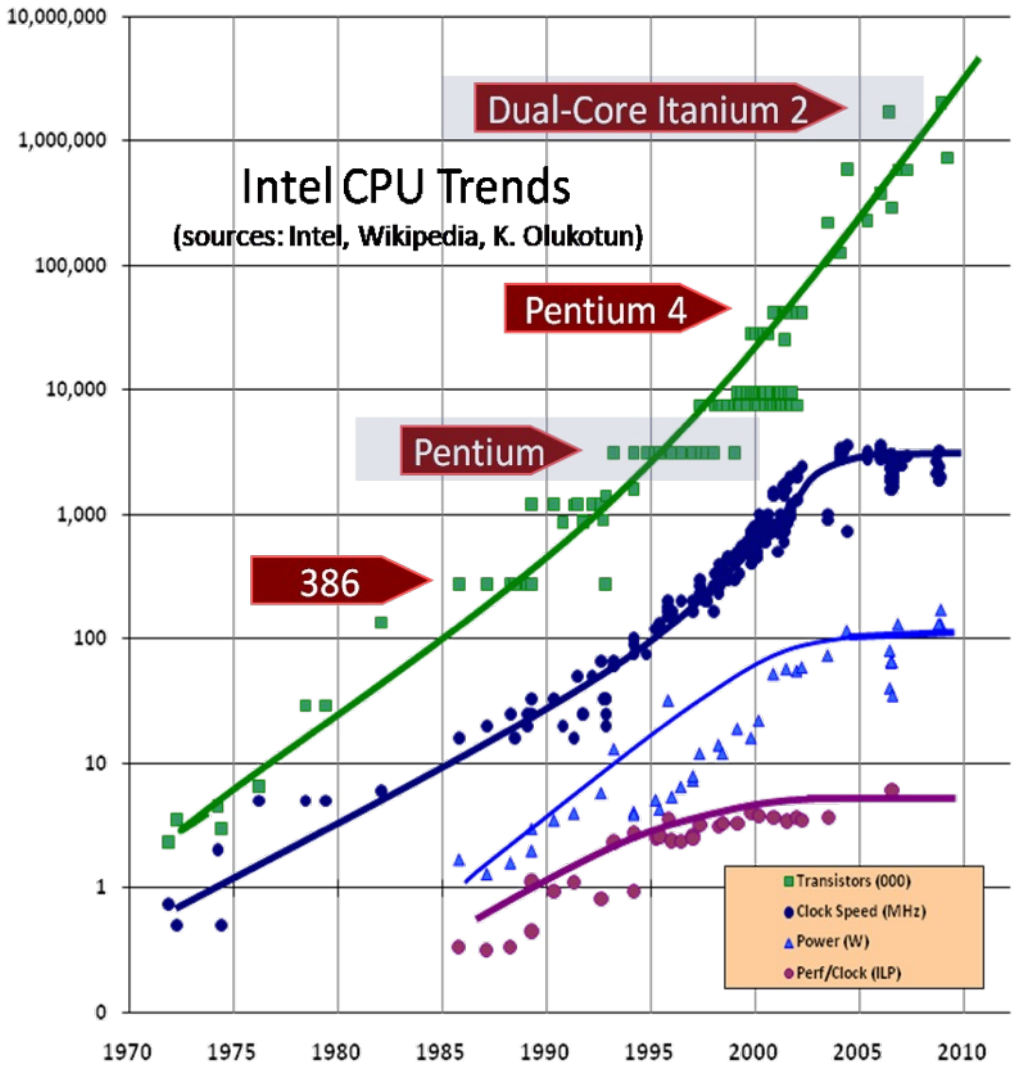
Dennard Scaling Implications

- Benefits of reducing voltage as we scale down
  - More transistors to build better processors
  - Faster switching frequency for better performance
  - … And, relatively constant power
- For approximately 40 years this trend continued
  - Drove clock frequencies from KHz to ~3GHz
  - Everyone's code became faster nearly for free

Oops! The 00s' Power Problem

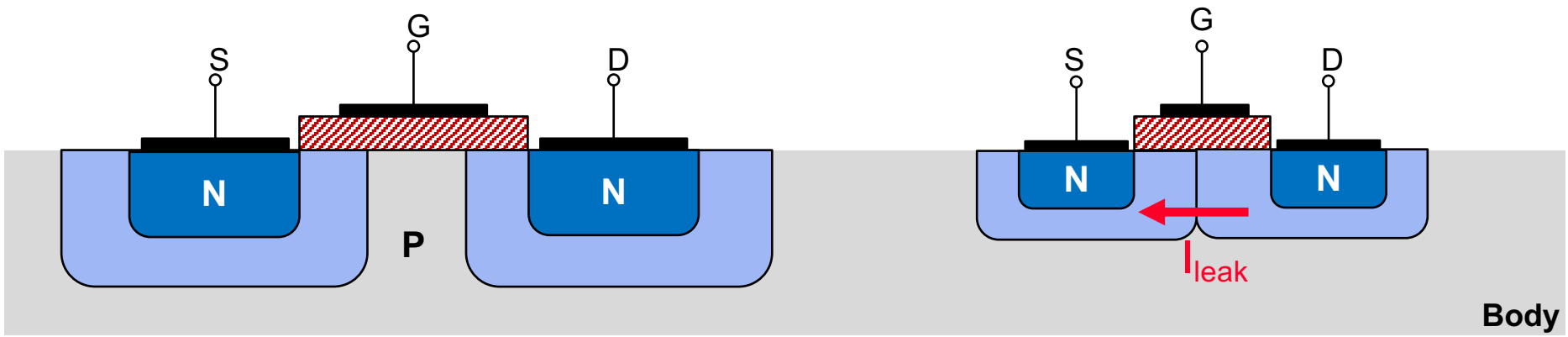
- In 2004, it all ended
  - Plateaus in:
    - Clock speed
    - Single chip power
    - ILP extracted from programs
  - But Moore's Law still continued on (green)

- Real problem is power
  - Limitations of CMOS
    - Sub-threshold current
    - Gate tunneling



Subthreshold Current: Breakdown of Dennard Scaling

- Recall:
  - Below V_{th} the transistor should be “off”
    - No channel for current to flow, supposedly
- As channel gets shorter, current starts to leak
  - Due to “depletion regions” of charge that form around the N areas
  - Exponential increase with temperature (runaway)

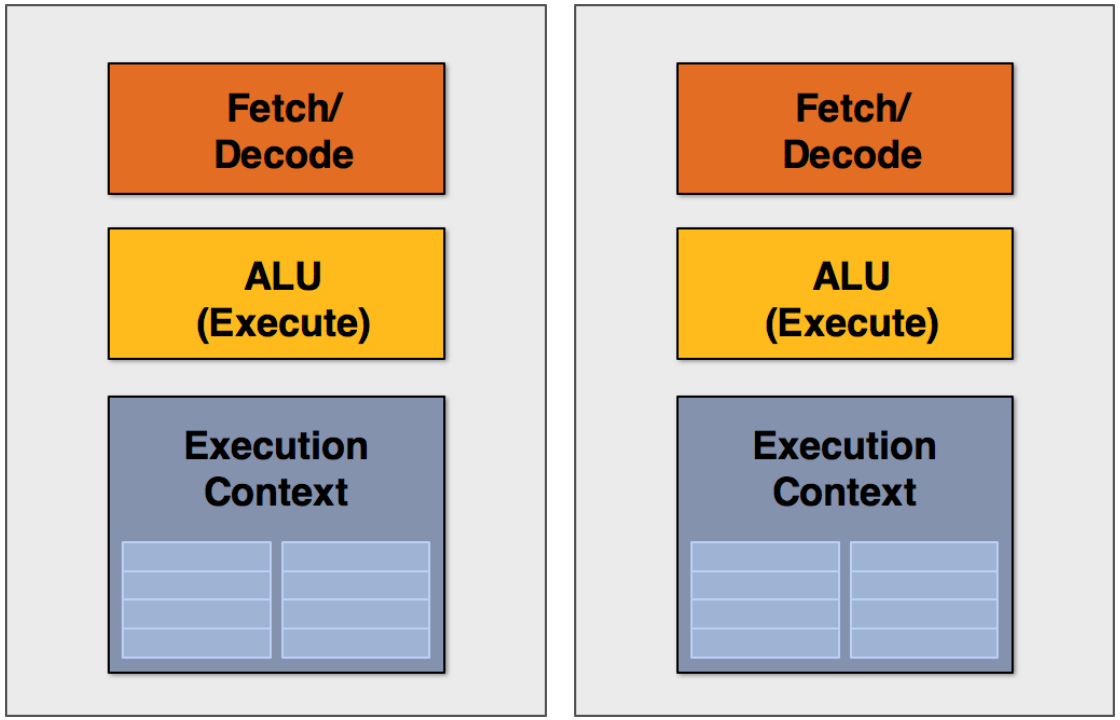


The Death of Dennard Scaling

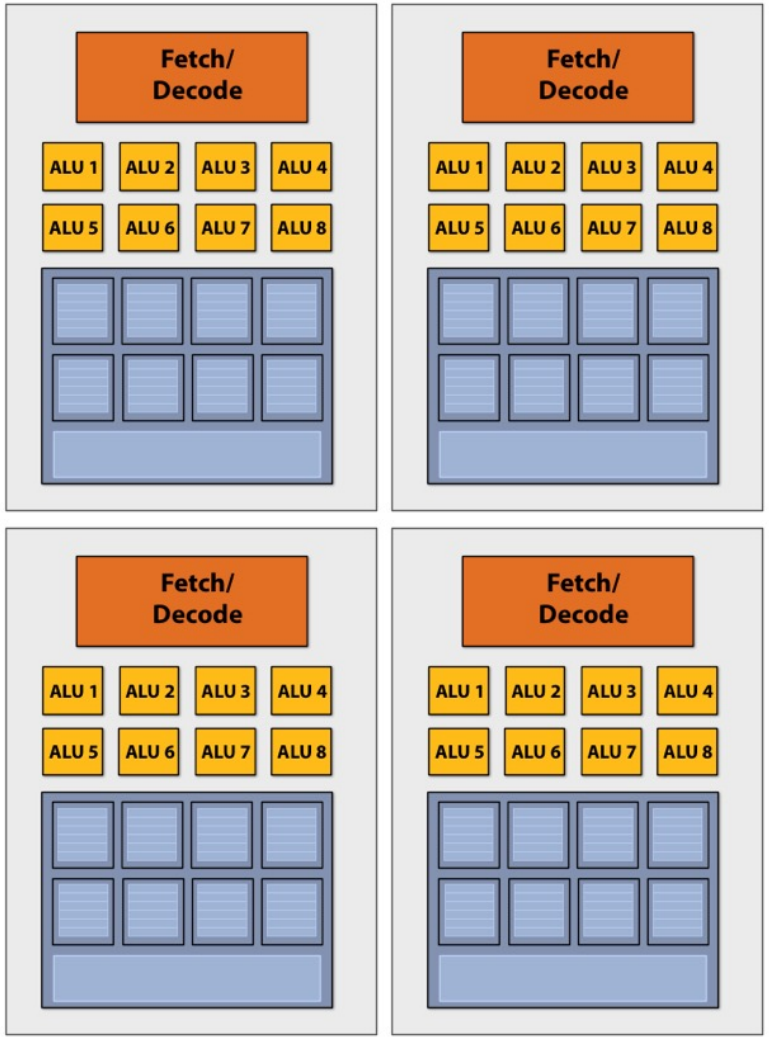
- Leaky transistors ended Dennard scaling
  - Reminder, power density no longer constant
- Intel ultra-perf. Tejas CPU targets (real!)
  - 7GHz+ clock frequency, 180W
  - 40-50 stage pipeline
- Actually released CPUs (Core)
  - 2.8GHz, ~80W
  - 10-12 stages

Common Choice: Parallelism

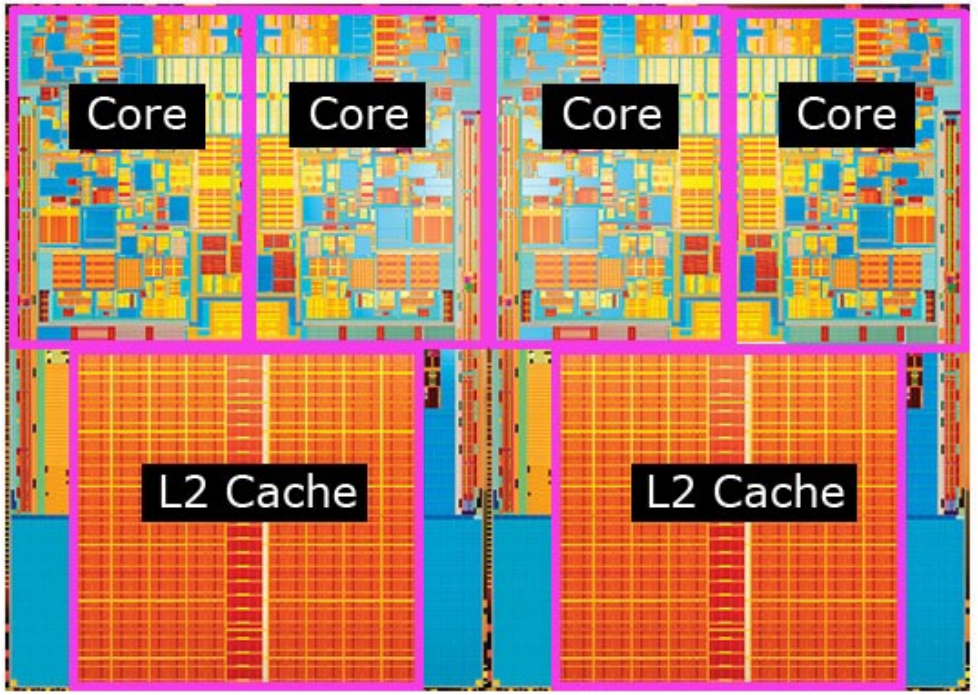
- Idea #1: Use increasing transistor count to add more cores
  - More transistors = larger cache, smarter OoO logic, smarter branch predictor, etc.
  - Simpler cores: each core is slower than original "fat" core (e.g., 25% slower)
  - But there are now two: 2 × 0.75 = 1.5 (potential for speedup!)



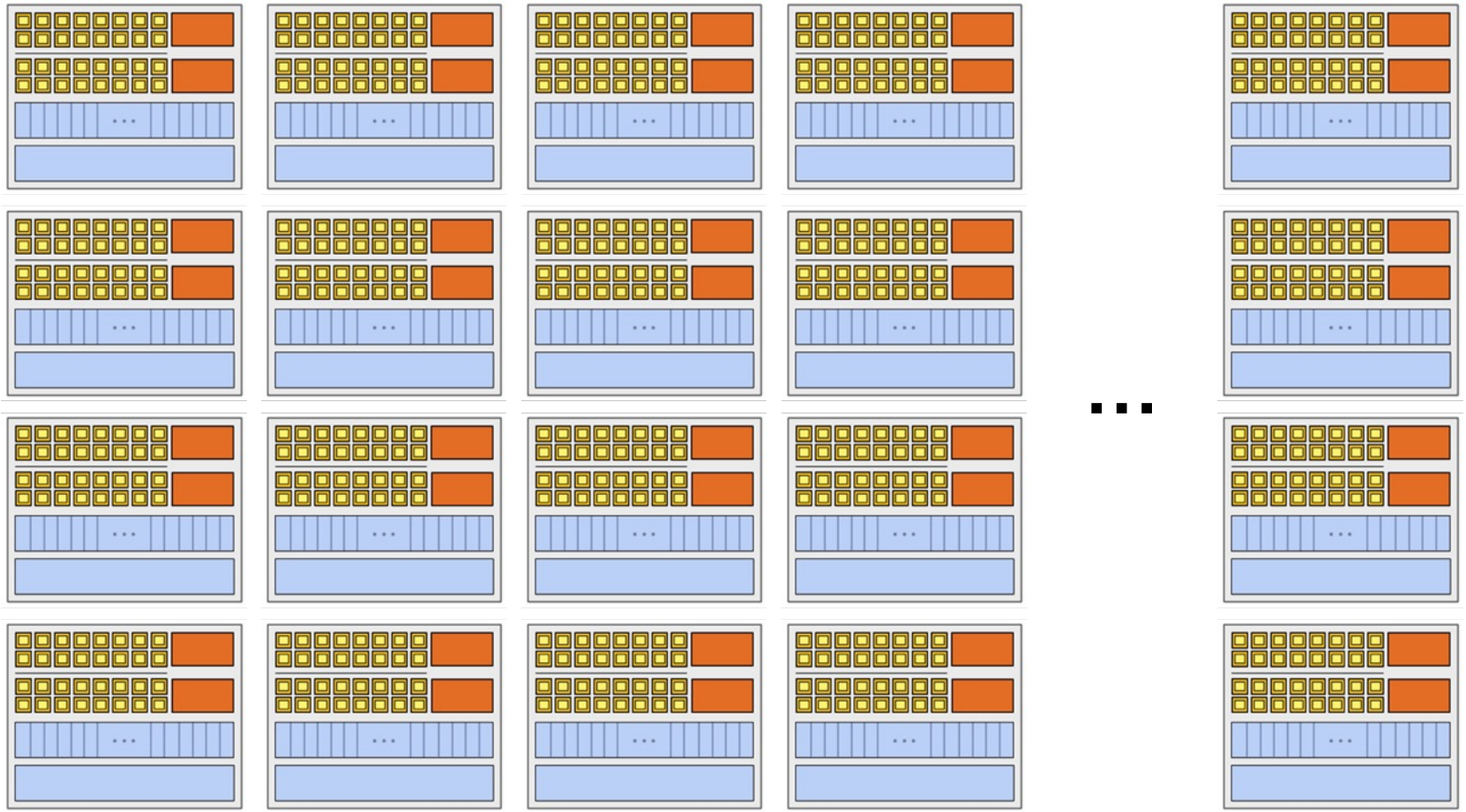
Examples We Saw: Multi-Core CPU



- Intel 4-core CPU (2006)
  - Instead of a single fast core
  - Higher throughput overall



Examples We Saw: NVIDIA Tesla V100

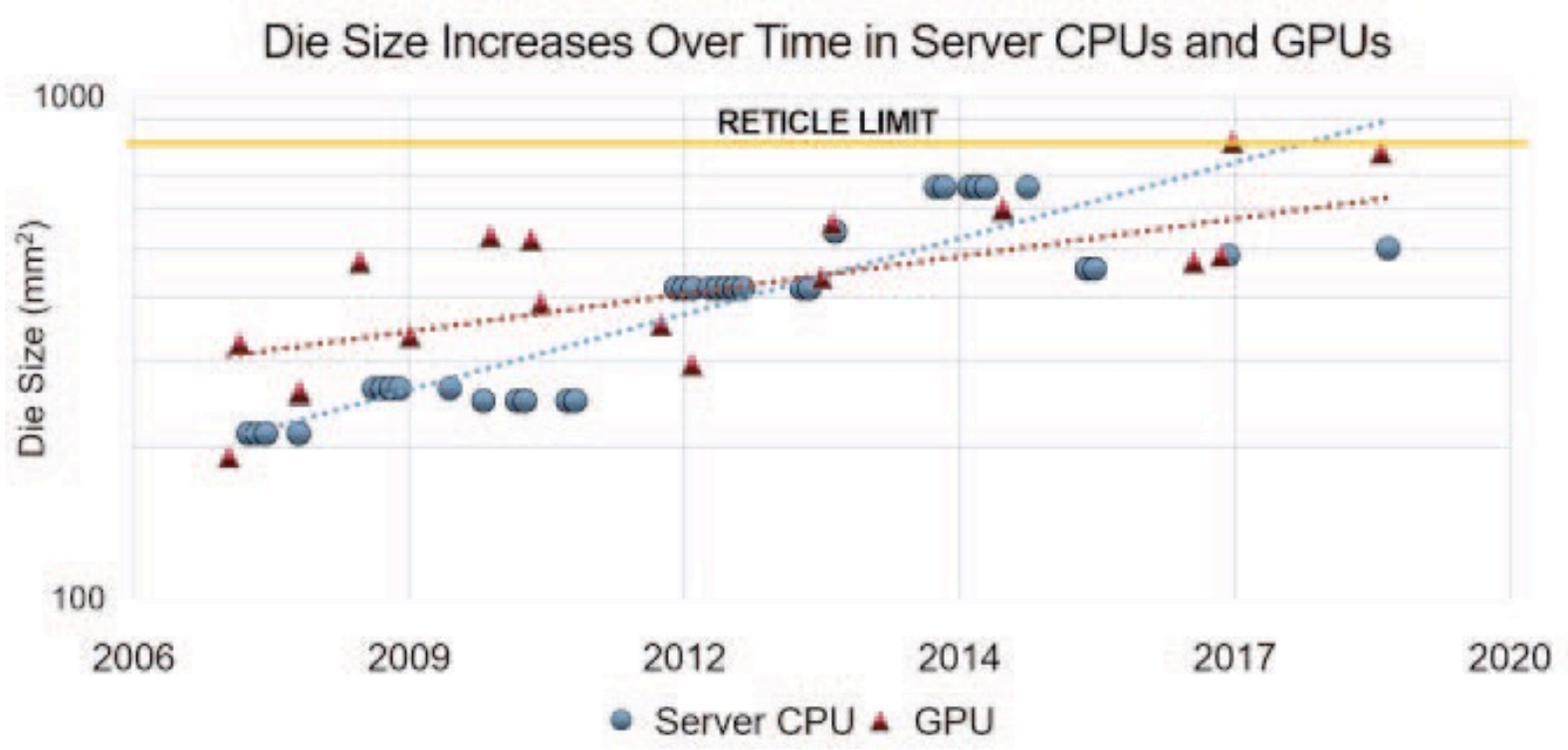


> 5000 CUDA cores (ALUs)!

Moore's Law Slowed Down: Not Cost-Effective Anymore

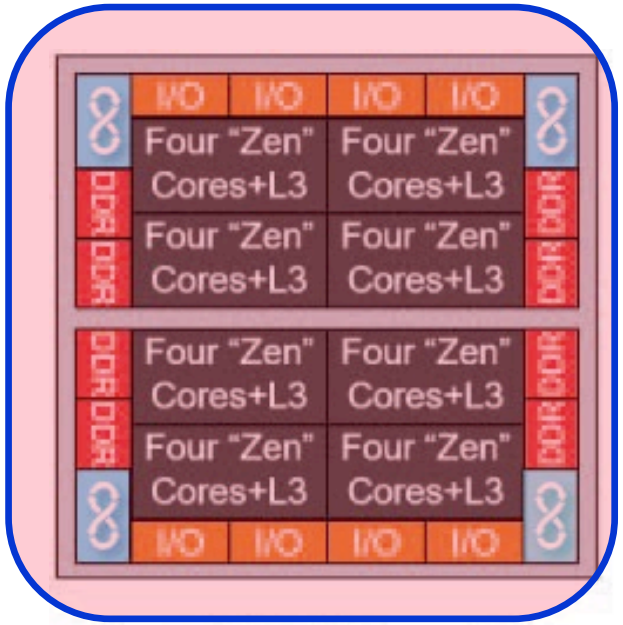
Picture from AMD

Chip sizes Reaching Lithographic Limit

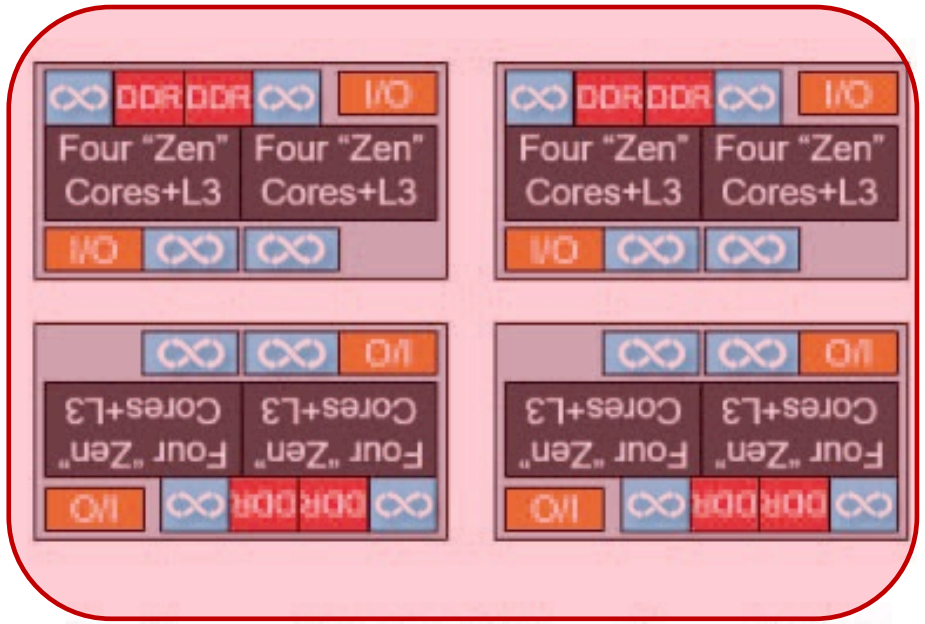


From Pioneering Chiplet Technology and Design for the AMD EPYC™ and Ryzen™ Processor Families : Industrial Product. ISCA 2021

Solution: Chiplets – Multi-chip Module (MCM) Processors



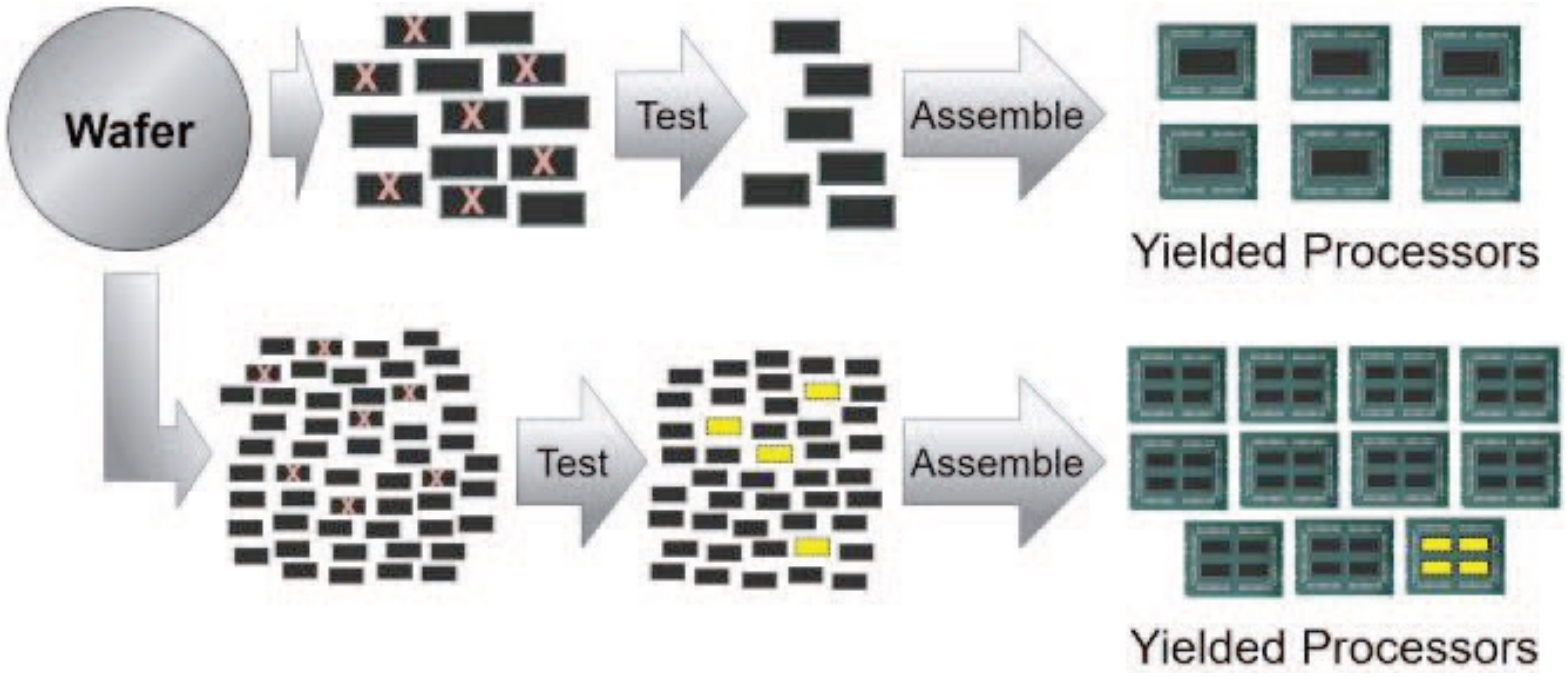
Monolithic 32-core Chip
777mm² total area
1.0x Cost



4 x 8-core Chiplet, 213mm² per chiplet
852mm² total area (+9.7%)
0.59x Cost

From Pioneering Chiplet Technology and Design for the AMD EPYC™ and Ryzen™ Processor Families : Industrial Product. ISCA 2021

Group of Small Chips (Chiplets) != Large Chip

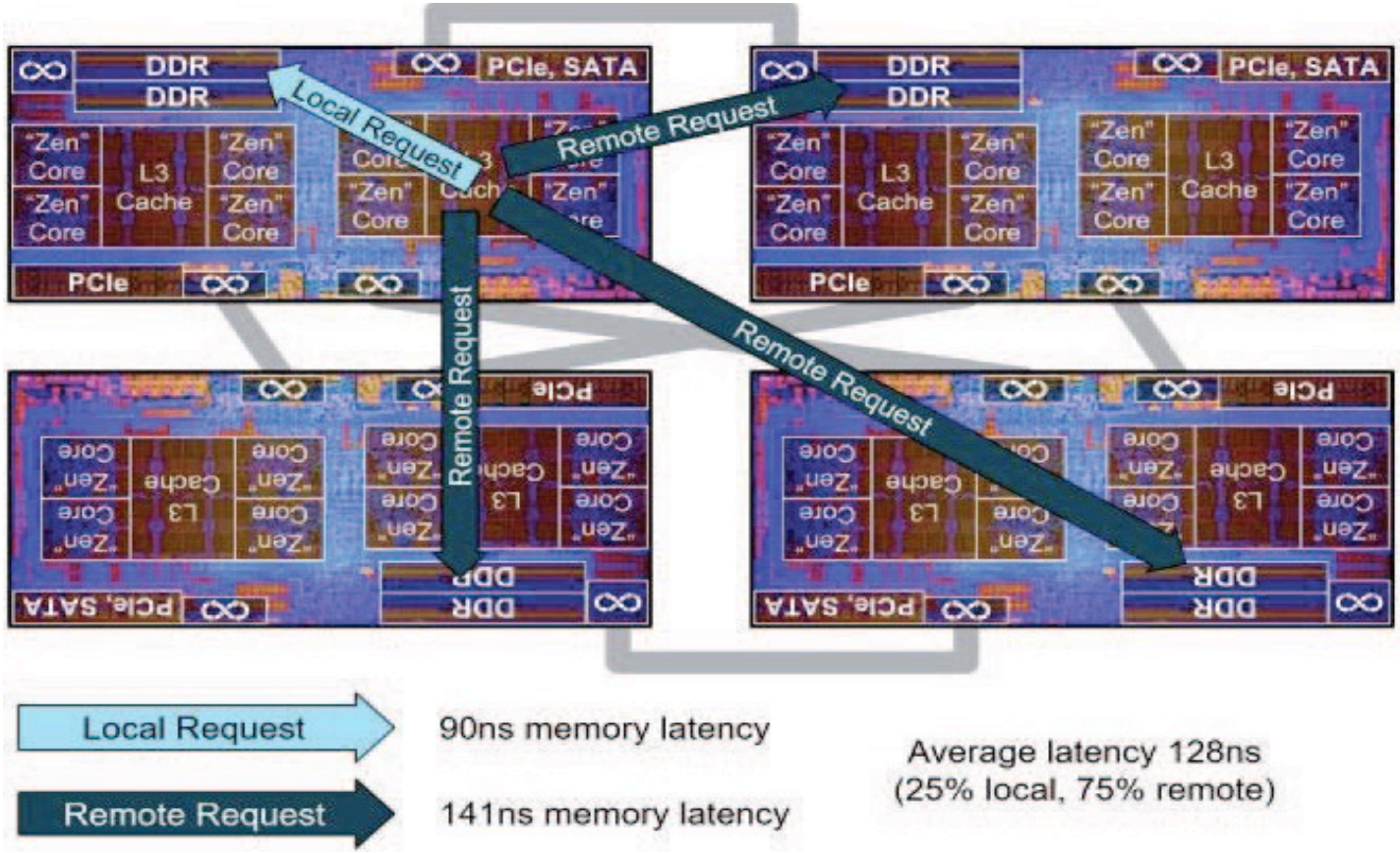


Chiplets allow processors to scale even with slowing Moore's law

More cores → More Chiplets

From Pioneering Chiplet Technology and Design for the AMD EPYC™ and Ryzen™ Processor Families : Industrial Product. ISCA 2021

Challenge of MCM Designs



- Logically a single processor to the software
- But, physically distributed resources → non-uniform latencies

ISA: Integration, Specialization, Approximation

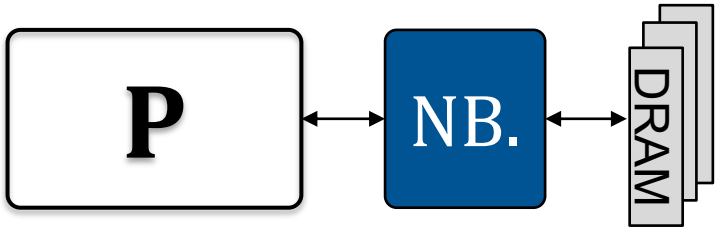
- Integration
  - Avoid going off-chip to reduce energy consumption
- Specialization
  - Trade off generality for efficiency
- Approximation
  - Trade off precision for efficiency

Integration: Bring more On-Chip

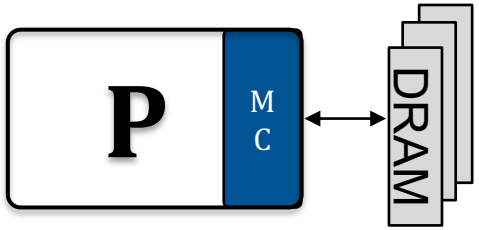
- "I/O" (refers to pins) is expensive
  - Need Digital → Analog converters to send data off-chip
    - e.g., To access memory, disk, PCI-Express
  - Every connection needs dedicated pins, limited supply
- With more transistors, bring external units on-chip
  - Reduces the number of I/Os in system
  - Frees up pins which have reached their physical limits

Example (Integration): Where did the Northbridge go?

- CPUs had a separate chip called a “Northbridge”
  - Between the CPU and memory devices



- Northbridge moved on-chip in ~2011
  - All memory controllers now closer to CPU
  - Lower latency to memory devices



Specialization: Optimized Computing Units

- Build execution units for different applications
  - Reformulate the hardware to reduce needed work
  - Can improve energy efficiency for a class of applications
- Stream / Vector processing is a current example
  - Exploit the fact that data appears in regular streams
  - No need for many registers or complex control flow

Example (Specialization): Google TPU

- Specialized ASIC for DNNs
  - Massive matrix multiplication unit
- Even more specialized than GPUs
  - Cheaper arithmetic (fixed-point, fp16)
  - Better usage of on chip storage
  - In Google servers today
- All of I, S and A

TPU's Computational Unit: Systolic Arrays

- Replace a pipeline structure with an array of processing elements (PEs) that can be programmed to perform a common operation
- Data is propagated between PEs → Lots of data reuse
  - Improves memory bandwidth and energy consumption
- Simple and regular design
- High concurrency
- Balanced computation and memory bandwidth

Systolic arrays

- H. T. Kung, “Why Systolic Architectures?,” IEEE Computer 1982.

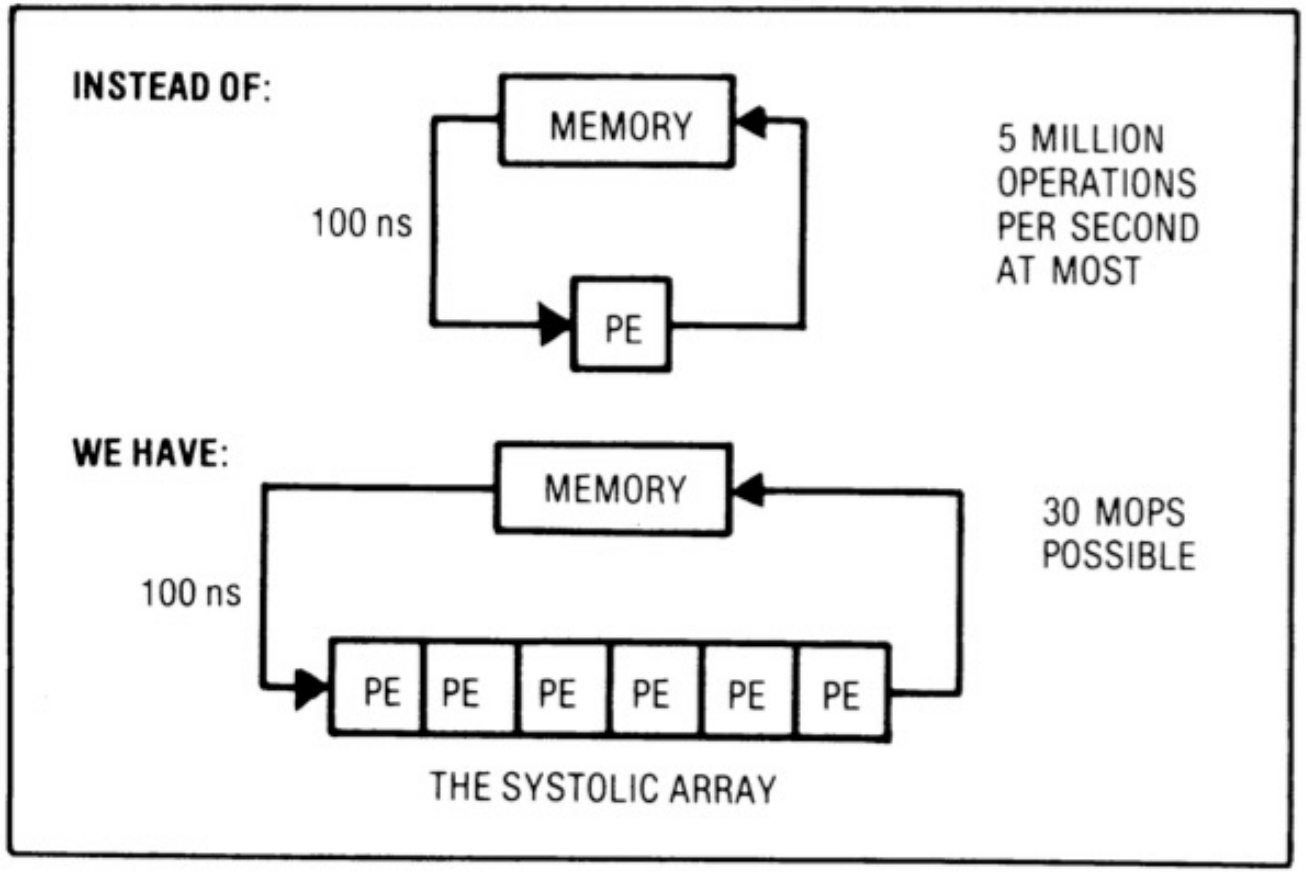


Figure 1. Basic principle of a systolic system.

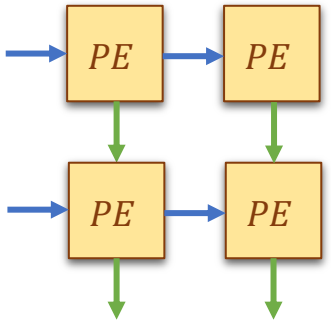
Memory: heart
PEs: cells

Memory pulses data through cells

Systolic arrays in DNNs

Core operation: matrix-matrix multiplication

- Spatially distributed processing elements (PE)



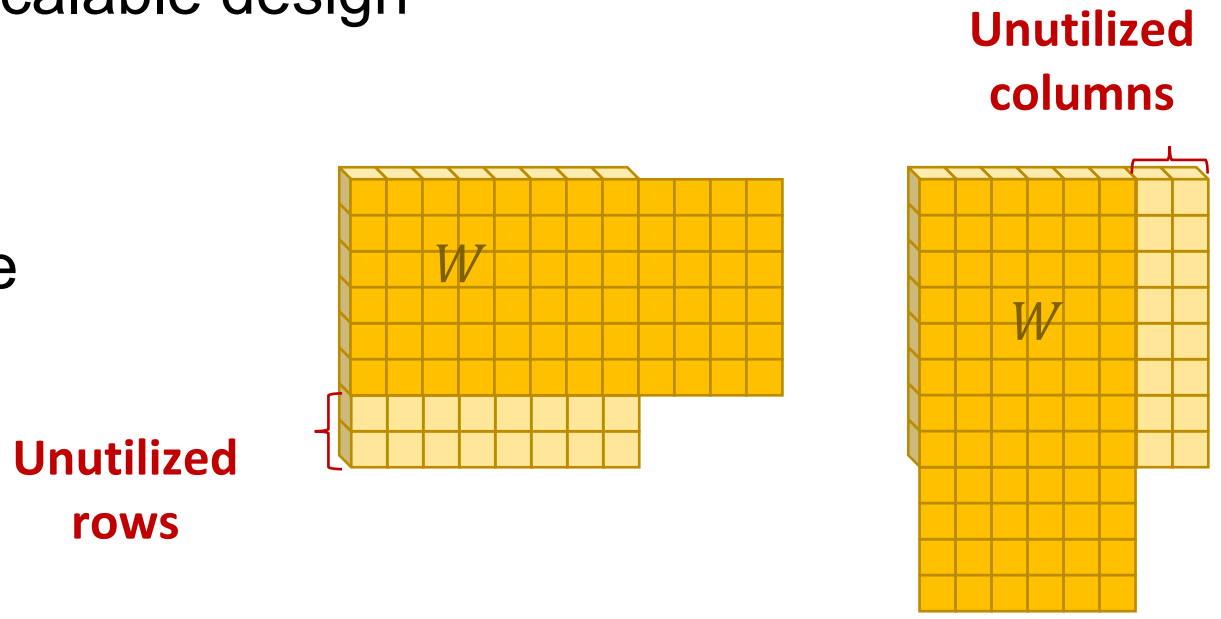
- Connect PEs in a mesh

- Use mesh links to communicate weights and activations (DNN parameters)
  - Activations are transferred by horizontal links
  - Weights are transferred by vertical links
  - PEs receive partial sums from vertical links

Pros & Cons of Systolic Arrays

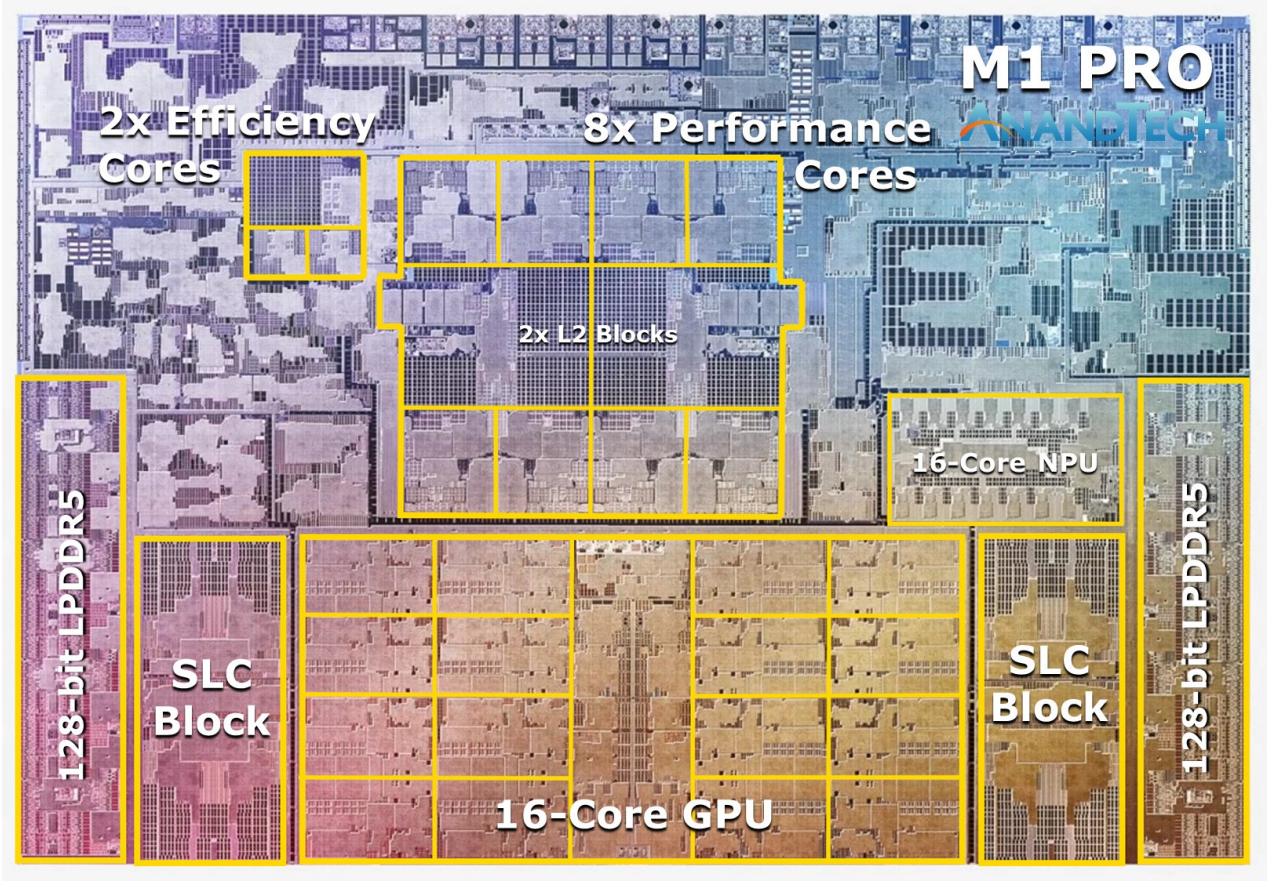
✓ Data movement: between adjacent PEs
- Energy consumption for data movement → 0
- Short connections → Scalable design

✗ Low utilization
- Matrix size ≠ Array size



Example (Specialization): Apple M1 Chip

- Apple 8-core CPU (2020)
  - Big-little architecture
    - Firestorm vs. Icestorm cores
  - ARM ISA
  - 5 nm fabrication process
  - Integrated GPU, neural engine
- All of I, S and A



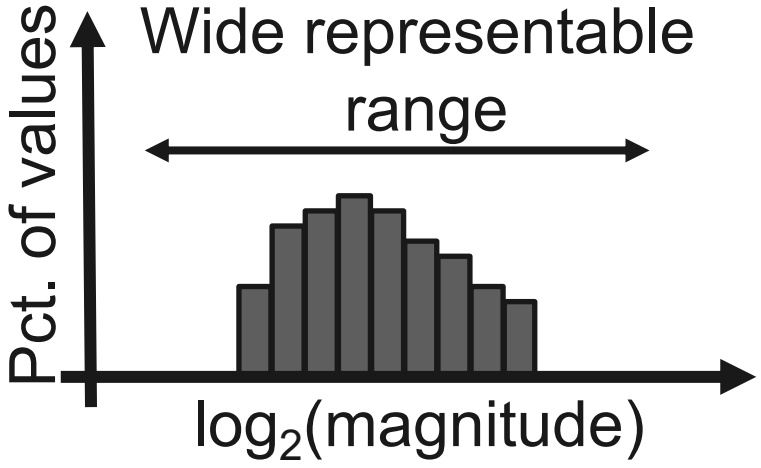
Approximation: Less Work, Similar Results

- Emerging applications are statistical
  - e.g., filtering data, compression, machine learning
- Savings possible by returning worse output
  - e.g., Doing 8-bit multiplication rather than 32-bit
- Tradeoff output quality for performance or energy
  - Ongoing research → when can users see difference?
- Example: ML Inference can tolerate lower precision than training

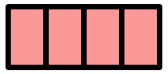
Example (Approximation): Numerical Encoding for DNNs

- High memory footprint, communication and computation requirements
  - Increase in the importance of the arithmetic density
- Floating-point format (FP)
  - Mantissa + exponent

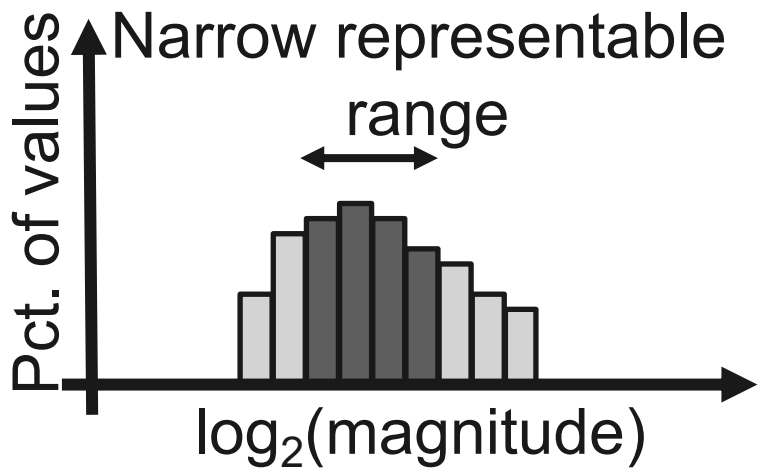
  - Wide representable range



- Fixed-point format
  - Mantissa

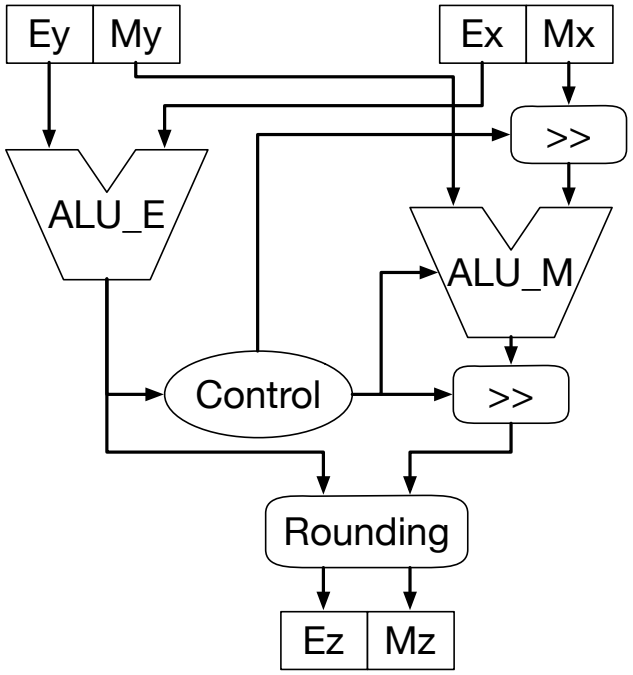


  - Narrow representable range

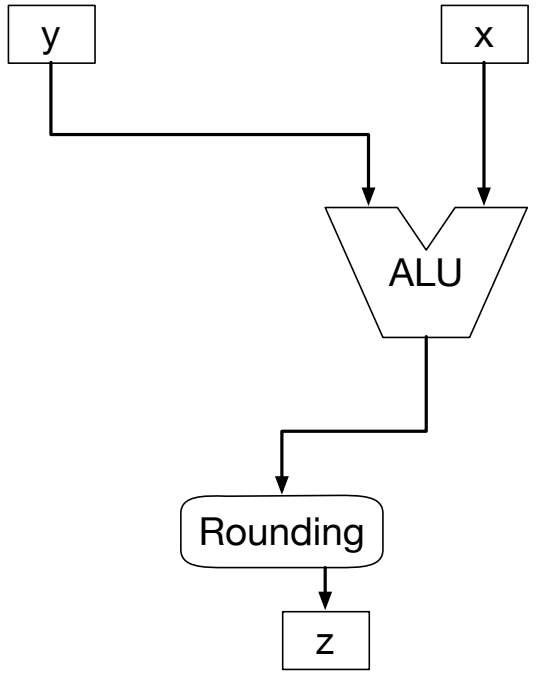


Example (Approximation): Numerical Encoding for DNNs

- Floating point (FP)
  - Complex exponent management
- Fixed point
  - No exponent management

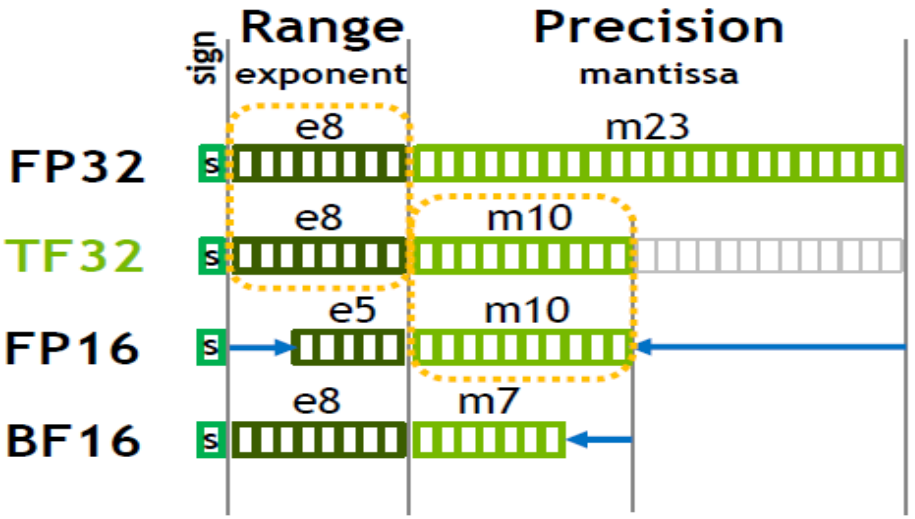


ALU Hardware



Approximation: Less Work, Similar Results

- Newer floating point format for ML workloads

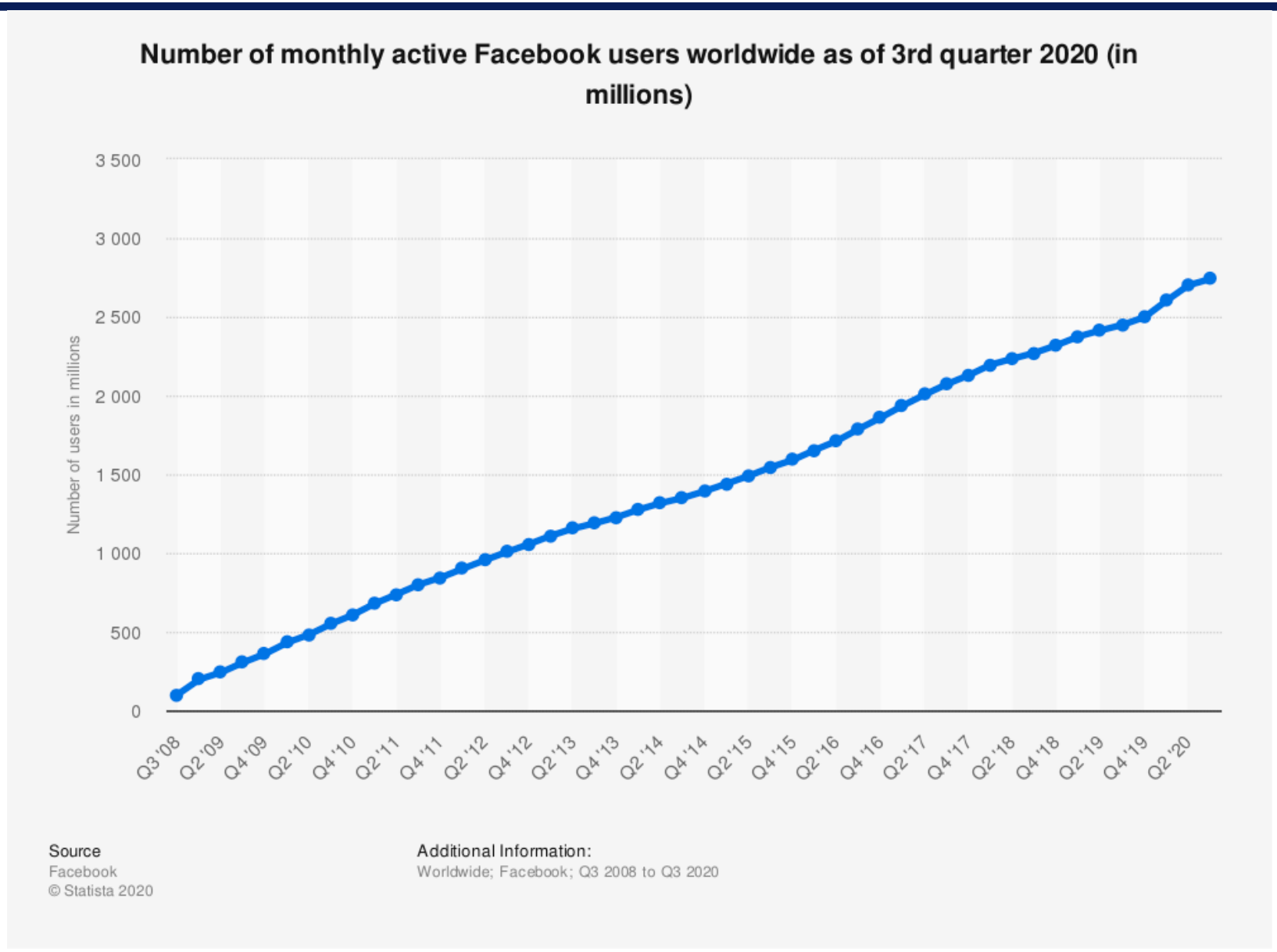


- NVIDIA's Tensor cores support many format
  - FP64, TF32, BF16, FP16, FP8, INT8

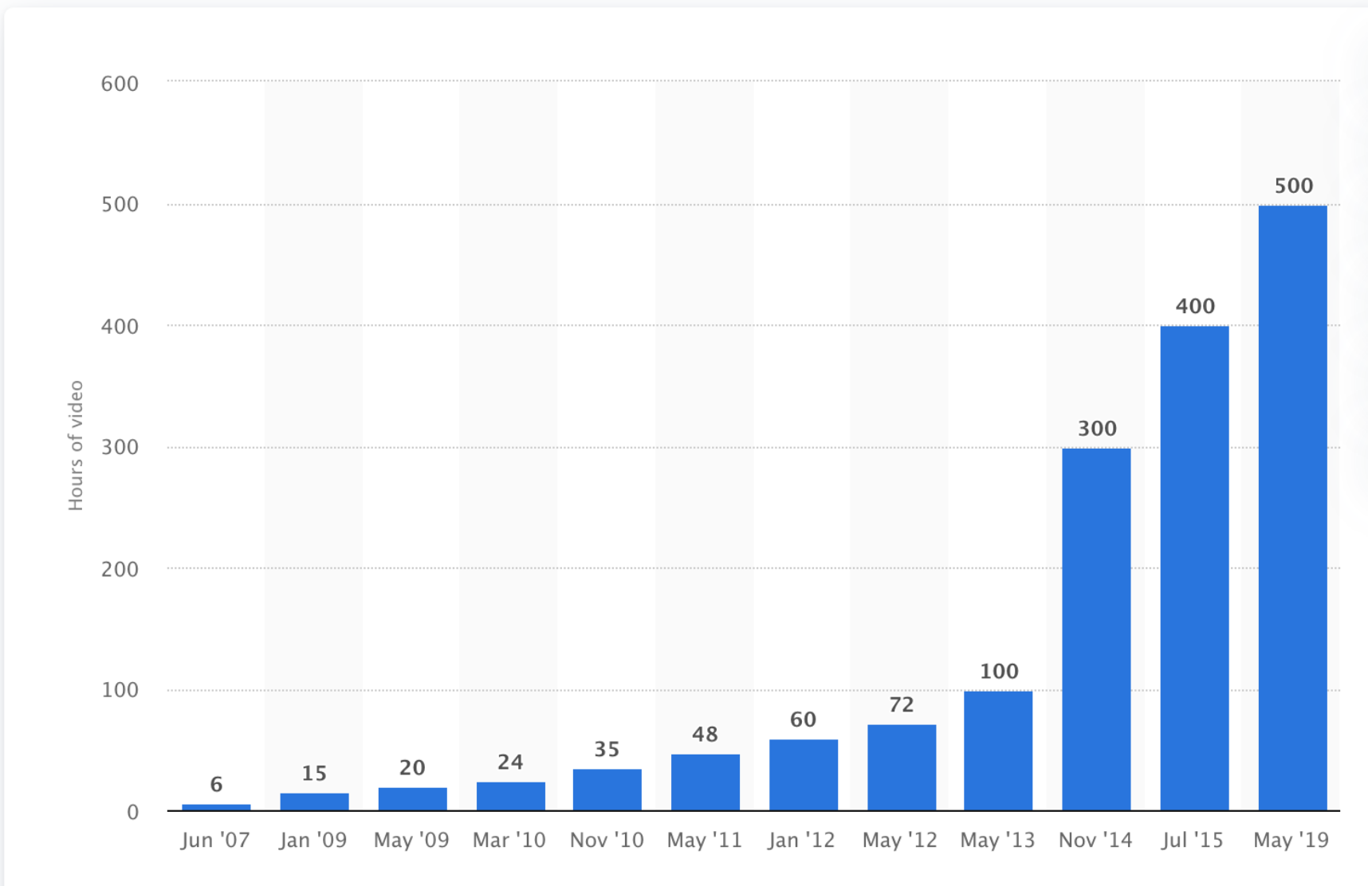
Rise of Big Data

- Amount of data to process didn't stop scaling!
  - All of us are generating data every day
    - e.g., your phone, your bus schedule, your shopping list
- Growth of data industry is still super-exponential
  - By end of 2020, we generate 40 ZB (trillion GB) of data
  - For perspective:
    - 1 ZB corresponds to **8 million years** of ultra-HD 8K video!
    - Theoretically, we could have videotaped 352 million years of Earth's history and put it in a data storage facility!!

Examples: Facebook w/ 2.7 billion active users



Examples: YouTube! 500 hrs of video uploaded/min



Simple Solution: Bigger Computers

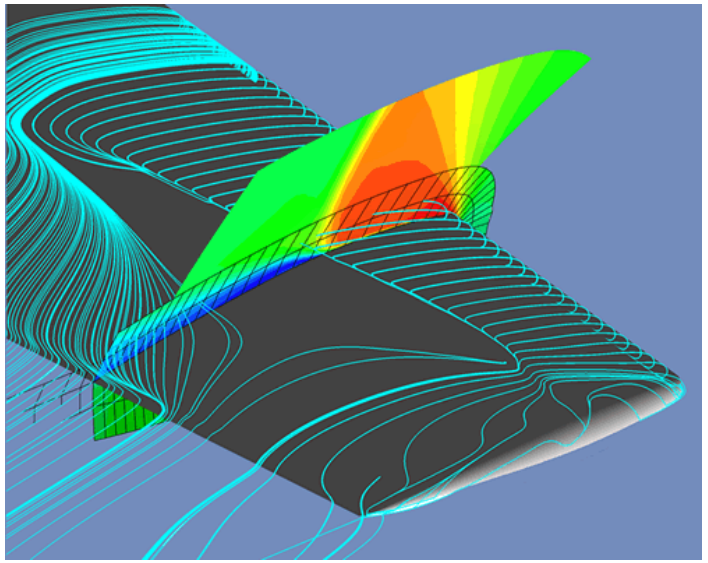
- Big problems need larger computers
- Buy more processors, connect them with cables and it should all work?
  - Fortunately it's not that easy (or we'd all have no job!)

Main Workload Classes

- These “big problems” are subdivided into:
  1. Compute intensive or “embarrassingly parallel”
     - Complex and intensive algorithms
  2. Memory intensive
     - The problem set size does not fit into a single node
  3. High throughput
     - Many unrelated problems executed concurrently
- High Performance Computing (HPC) generally looks like #1 and #2
- Modern Internet services look more like #3

Characteristics of HPC Applications

- Often embarrassingly parallel, recursive
  - e.g., simulating an aircraft wing
- Divide wing into a grid of small cells
  - Compute forces and air speeds on each cell
  - Simple to split over parallel processors
- Each cell can be divided into new grid

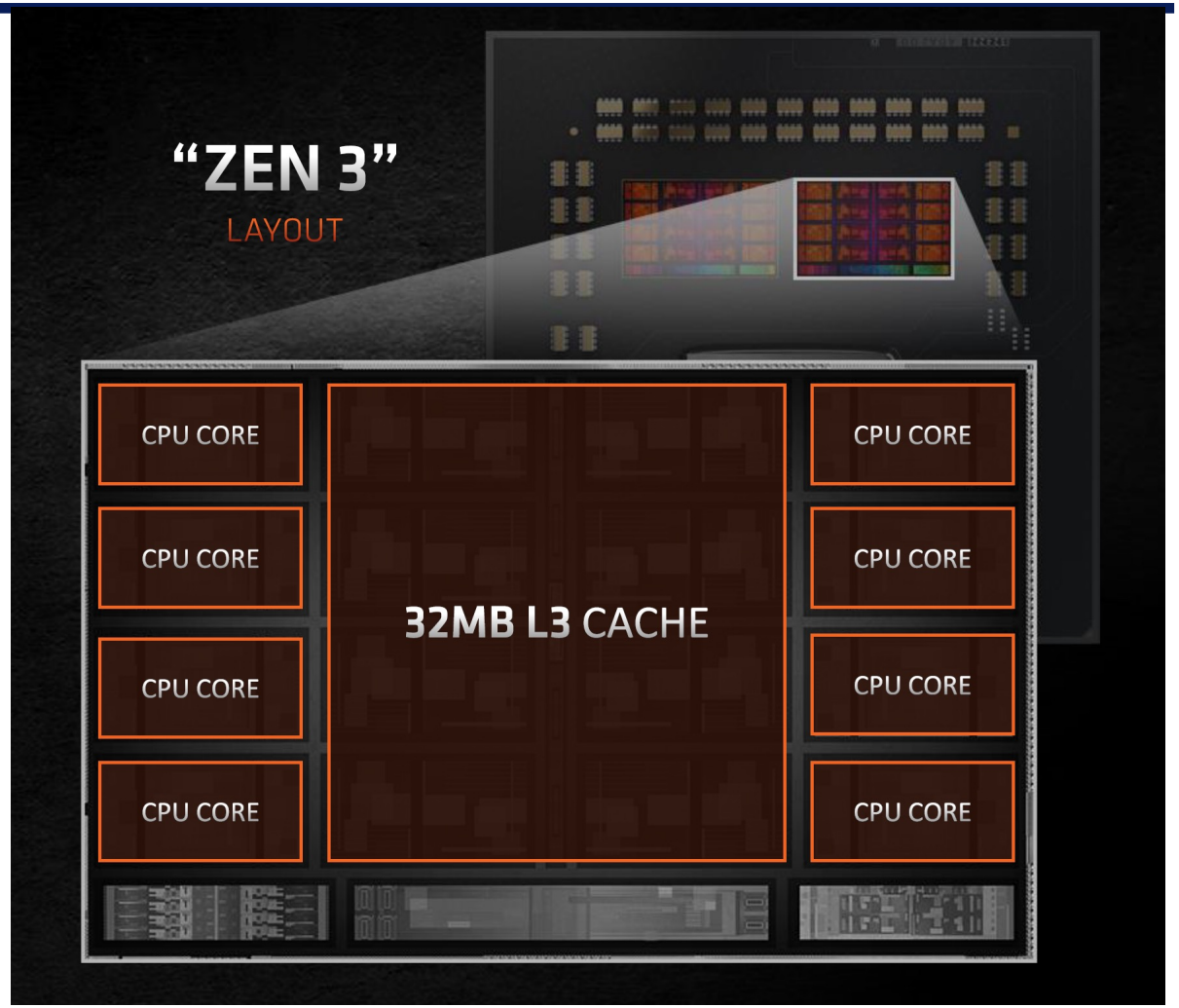


General Approach

- Distribute the single big problem over processors to reduce execution time
  - Requires “parallelization” → you have experience now!
  - A well parallelized program will use all the processors
- At the end of the parallel stage, all sub-processes communicate with each other, and restart
  - Similar to OMP fork/join!

Scale-Up Machines Today

- Collection of many cores
  - E.g., AMD Zen3 (2020)
  - 64 cores (8 cores per chiplet)
- Or Intel Sapphire Rapids (2023)
  - 56 cores (14 cores per chiplet)
  - Still cache-coherent!



Programming on Scale-Up System

- This course assumed a shared address space for threads
  - All of your OMP threads communicated through one of:
    - Memory locations with reads/writes
    - Locks & Barriers
  - Typical commercial scale-up machines for databases:
    - CC-NUMA hardware, programmed with C and beyond
    - Oracle/HPE/Huawei multi-socket x86/SPARC, TB memory

In Contrast… Modern Internet Service

- Recall:
  1. Compute Intensive or Embarrassingly Parallel
  2. Memory Intensive
  3. High Throughput

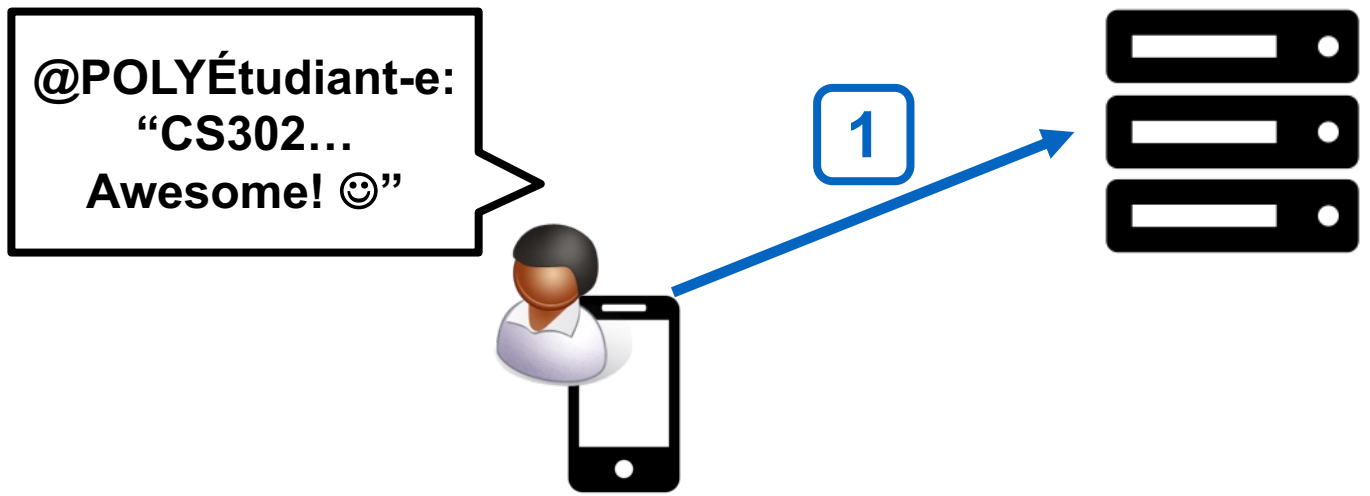
- Internet services (Facebook) look more like #3
  - All of your messages, are small tasks
  - … but there are a huge number of them!

Characteristics of Applications

- The whole Internet service appears like one app.
  - Cannot be run on a single machine due to user count
- Request- or user-level parallelism (concurrency)
  - Each task is small but they all run together
  - Therefore, most designs prioritize high throughput

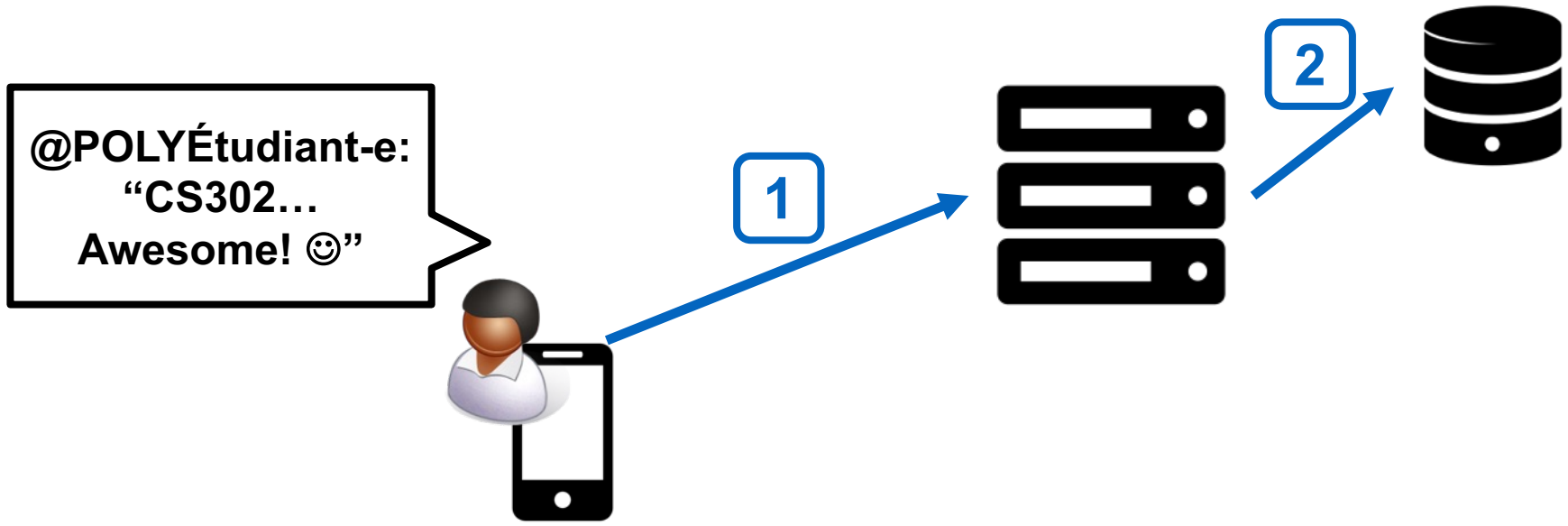
Example Internet Scale Request

- E.g., POLYTwitter post “CS302… Awesome! ☺”
  - Each user post needs to do all of these things:
    1. Connect to POLYTwitter servers



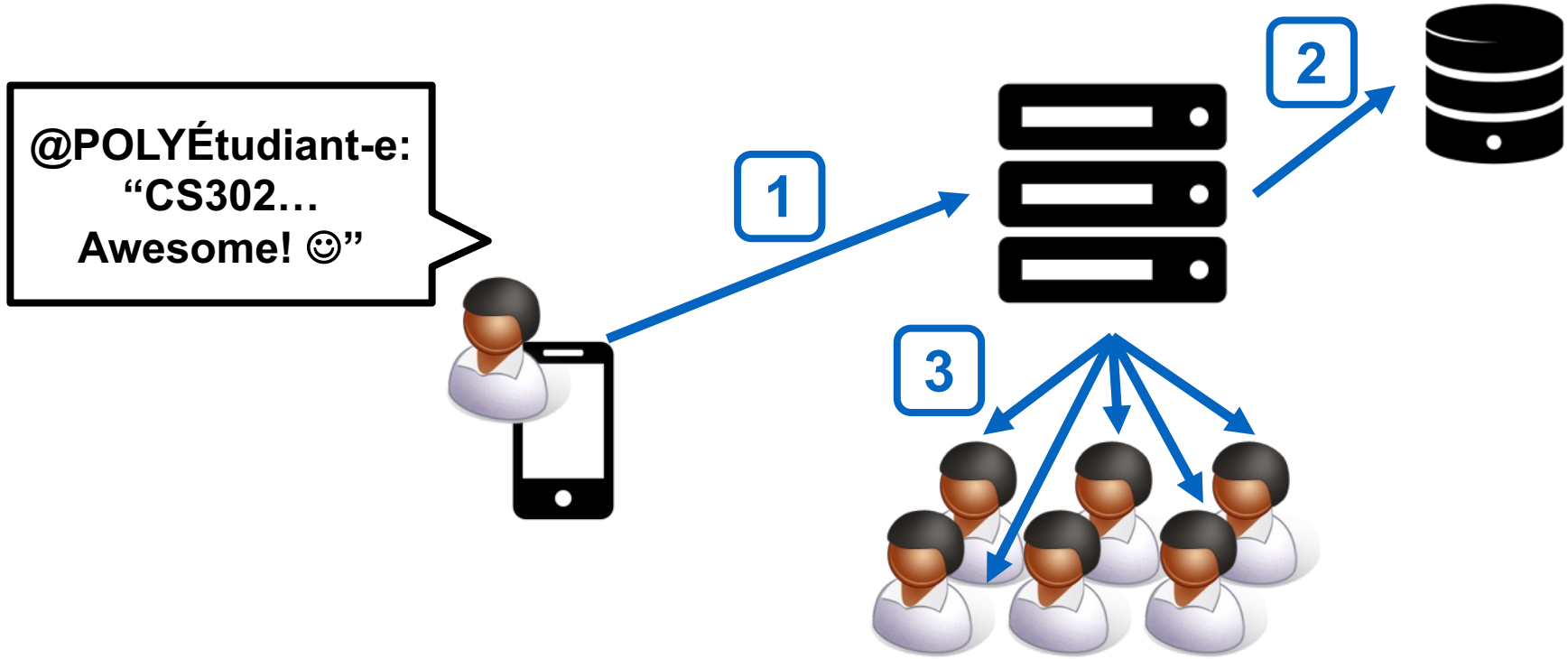
Example Internet Scale Request

- E.g., POLYTwitter post “CS302… Awesome! ☺”
  - Each user post needs to do all of these things:
    1. Connect to POLYTwitter servers
    2. Store the text of your post on disks (~140B)



Example Internet Scale Request

- E.g., POLYTwitter post “CS302… Awesome! ☺”
  - Each user post needs to do all of these things:
    1. Connect to POLYTwitter servers
    2. Store the text of your post on disks (~140B)
    3. Possibly notify your followers



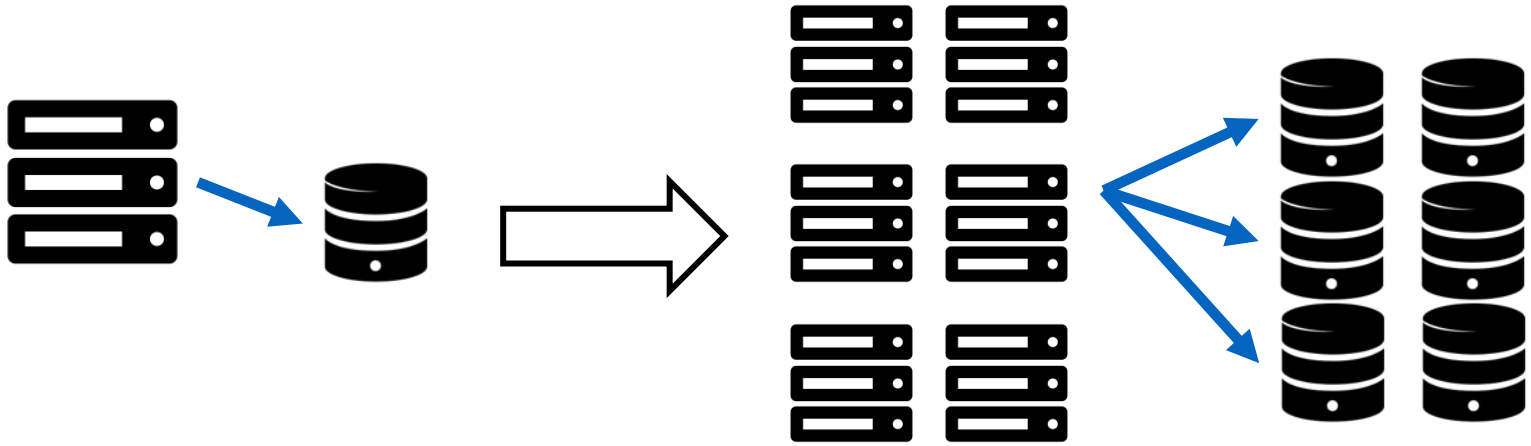
Scaling Up POLYTwitter?

- Now we have 50k users (say all of ETH joins too)
  - Clearly won't fit onto one server
- Does it make sense to scale up our service?
  - Buy super expensive HPE machines with 200 CPUs
    - Use shared memory (up to 48TB!)
- No… At some point we still hit the limit
  - 50k users at 10 tweets/day, we have 2000y of storage (in 48TB)
  - But with 200 CPUs we have a concurrency problem
    - If all of our users want to log in at the same time (e.g., when new holidays get announced)

Alternatively: “Scale-Out”

- A scale-out system adds multiple independent nodes that each can execute the desired service
  - e.g., More servers or disks for POLYTwitter

- Each node in the system can be added/removed
  - Overall service improves in aggregate



Main Differences from Scale-Up

- Scale-out systems better for throughput-oriented
  - Any incoming task can go to any of the nodes
- Applications need to be engineered for Scale-out
  - Non trivial task!
  - Scale-out nodes are smaller and often shared
    - Not immediately obvious on how to improve performance
  - Scaling up a node (more cores) → more threads can run
- Scale-out handles node failure much better
  - The service still works as long as a single node is online

Example Scale-out: Datacenters

- 1.6km x 200m
- 10^7 cores
- 10^{18} bytes

➔350 MW

Scale-out node:

- 2 sockets
- 256 GB DRAM
- FLASH
- NIC
- Basic HW/OS for desktop PC

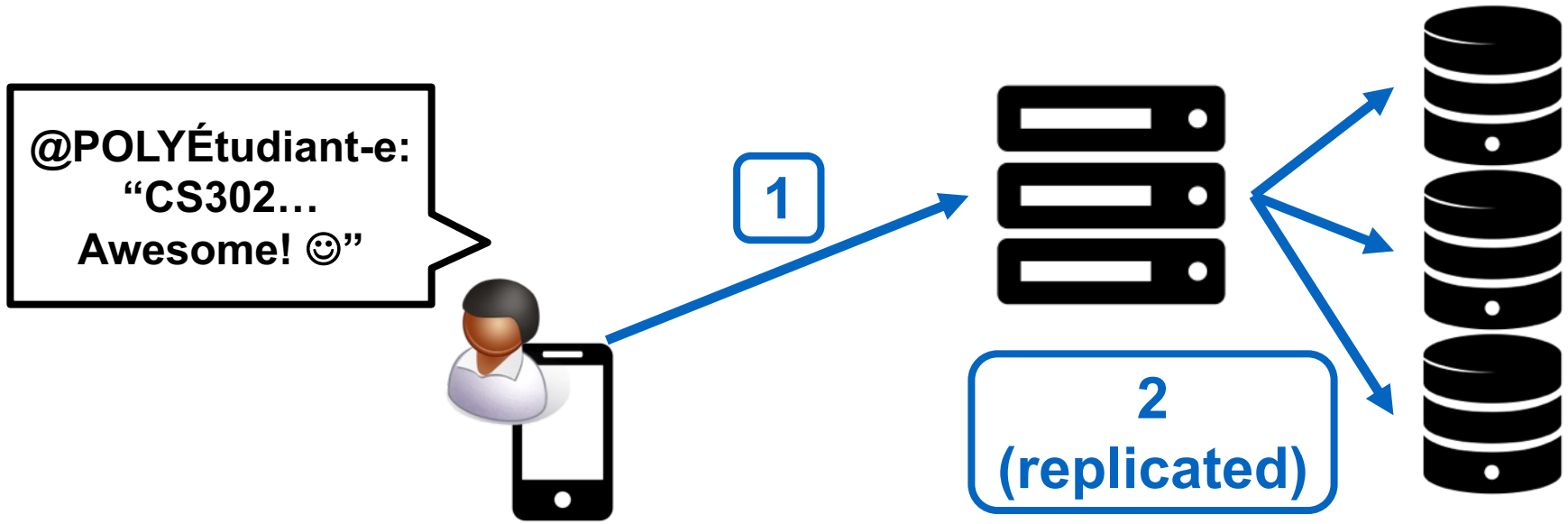
350MW

Failure Tolerance in Software

- Scale-out machines fail **extremely** often
  - Google: “If you have 1000 servers, one fails daily”
  - We know Google has upwards of 1M machines…
    - So that means 1k servers fail in a day
- Service needs to be always available
  - E.g., 99.99% uptime means one hour **per year**
  - Design the system specifically for redundancy

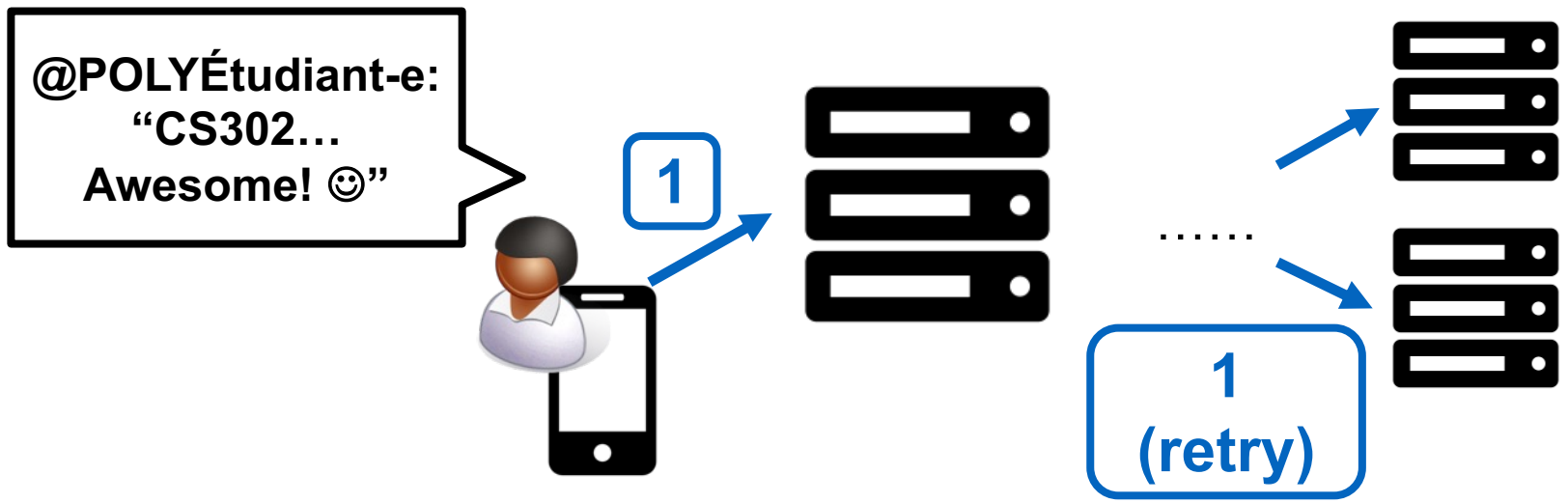
Example: Data Replication

- In POLYTwitter, if a disk fails, all tweets are gone
  - Instead of trying to build a super-reliable disk…
  - Replicate the data across many disks
- Real world design (e.g., Google File System)



Example: Request Duplication

- If a server fails, your tweet may not be recorded
  - Instead of trying to build a server that never crashes...
  - Assume long latency requests went to failed servers
    - Retry those requests on other servers
  - Also real world (e.g., Amazon Dynamo key-value store)



Programming on Scale-Out System

- Fundamentally, Scale-out systems are distributed
  - Requests can touch many nodes during lifetime
    - e.g., Frontend, web server, disk, and logging nodes
- Complex (fun) algorithms to orchestrate nodes
  - Example: how to know if a node failed, or if it is compromised and pretending to be offline?
- Programmers who know this field are valuable!

Convergence of Scale Up/Out

- Most scale-out systems use commodity parts
  - Highest cost efficiency with use of distributed software
- But, problem sizes in datacenters are growing
  - Starting to resemble “Memory Intensive” applications, #2 on our list that was more HPC-like
  - Want one node to access memory from other nodes
- Scale out the processors, scale up the memory?

Example: Use of Infiniband Network Gear

- High bandwidth IB networks normally for scale-up
  - Fine-grain and frequent communication
- IB making inroads into scale-out datacenters
  - Allows nodes to directly access remote memory (RDMA) as if it was their own
- Brings back fault tolerance challenges
  - If local node fails, both program and data both lost
  - But if remote node fails… Other programs' data lost

Summary

- Why parallel hardware exists
  - Moore's Law let designers scale up their chips
  - Eventually, Dennard scaling ended due to power
  - Alternative approaches: parallelism, ISA
- Multiple node systems
  - Scale-up → adding more functionality to single node
  - Scale-out → replicating nodes to add capability

REMINDER: Final Exam This Week!

- Material from **all lectures (but this one)** may be on exam
- The sample exam is released on Moodle