CS 1541 Introduction

Instructor Introduction

Course Introduction

Wonsun Ahn
Department of Computer Science
School of Computing and Information

Instructor Introduction

My Technical Background

- Wonsun Ahn
  - First name is pronounced *one-sun* (if you can manage)
  - Or you can just call me Dr. Ahn (rhymes with *naan*)
- PhD in CPU Design and Compilers
  - University of Illinois at Urbana Champaign
- Industry Experience
  - Bluebird Corporation (70-person startup company)
    - ➢Manufactures industrial hand-held devices
    - ➢Me: Built software stack based on Windows Embedded
  - IBM Research (thousands of people)
    - ➢Does next-gen stuff like carbon nanotubes, quantum computers
    - ➢Me: Designed supercomputers for ease of parallel programming

My World View

- Everything is connected
  - Pandemic: If my neighbors catch the virus, so will I
  - Environment: If my neighbors pollute, I will feel the effects
  - Economy: Think of how the subprime mortgage crisis spread
- Zero-sum thinking (old way of thinking)
  - “If you get a larger slice of the pie, I get a smaller slice.”
  - Therefore, if you lose, I win (and vice versa)
- Zero-sum thinking no longer works
  - If you catch the virus, do I become safer from the virus?
- Collaboration is replacing competition

Collaboration is Replacing Competition

- Is happening in all spheres of life
- Collaboration is also happening in the IT industry
  - The *open source* movement
  - Increasing importance of the software/hardware *ecosystem*
  - Increasing importance of the developer *community*
- Collaboration is also important for learning
  - During my undergrad years, what do I remember best?
  - Stuff that I explained to my classmates
  - Stuff that my classmates taught me

Supporting Collaborative Learning

- I *do not* grade on a curve
  - You will not be competing against your classmates
  - You are graded on your own work on an absolute scale

- You are a member of a Team
  - You are already a member of the class on Microsoft Teams
  - I encourage you to be on Teams at most times (I will too)
    - ➢ You can install app on both laptop and cell phone
  - If you have a question, you can ask in the Team "Posts" tab
    - ➢ Either your classmate or your instructor will answer
  - You can chat with any individual on the Team
    - ➢ "Manage Team" item in the "…" Team context menu

Supporting Collaborative Learning

- You are a member of a Study Group
  1. Right now, form a group of 4~6 members with neighbors
  2. Create a Teams chat group comprised of these members
  3. Invite your instructor to chat group (ID: wahn@pitt.edu)
  4. Send at least one message in chat group so I can see it
  - Can discuss TopHat questions/answers before submitting
  - Goal: develop a basis of knowledge for homeworks/exams

- You are a member of a Project Group
  - Find one partner with whom you want to do projects
  - On GradeScope, submit to “Partnership Contract”
    (Please do one submission per group by adding partner)

Course Introduction

Structure of the Course

- (45% of grade) Two midterms
- (20% of grade) Two projects
  - Implementing a CPU simulator using C programming language
- (20% of grade) Four homeworks
- (15% of grade) Participation
  - Attendance, TopHat lecture questions, Teams participation
- Class resources:
  - Canvas: announcements, Zoom meetings, recorded lectures
  - GitHub: syllabus, lectures, homeworks, projects
  - Tophat: online lecture questions
  - GradeScope: homework / projects submission, grading and feedback
  - Microsoft Teams: all out-of-class communication

Textbook (You Probably Have it)

- "Computer Organization and Design - The Hardware/Software Interface" by David Patterson and John Hennessy Fifth Edition - Morgan & Kaufmann.

For More Details

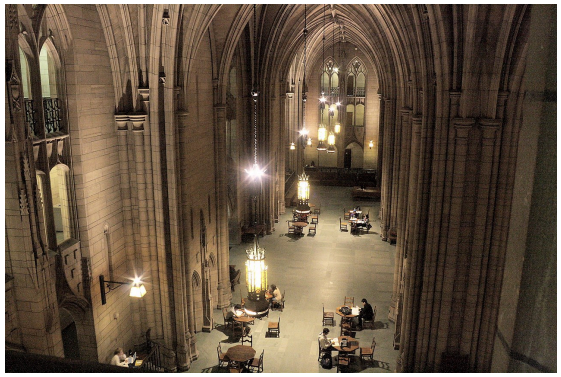
- Please refer to the syllabus page: https://github.com/wonsunahn/CS1541_Spring2023/blob/main/syllabus.md

- Please follow the course schedule: https://github.com/wonsunahn/CS1541_Spring2023/blob/main/schedule.md

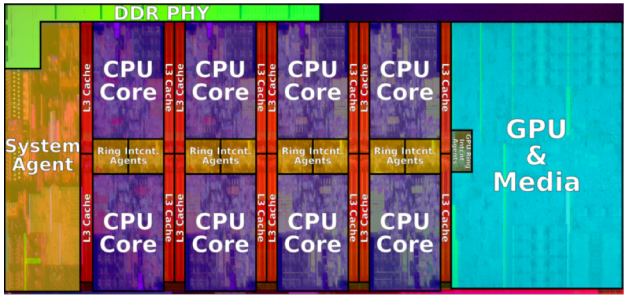
Computer Architecture. What is it?

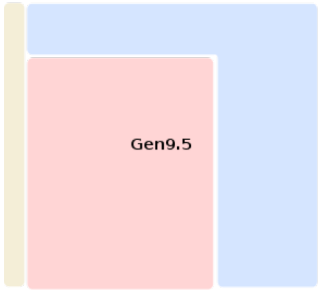
Computer Architecture: the architecture of a CPU

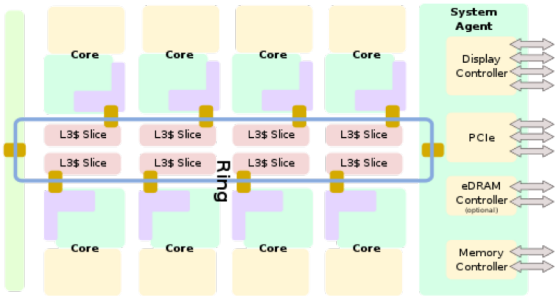
- Every building has an architecture:

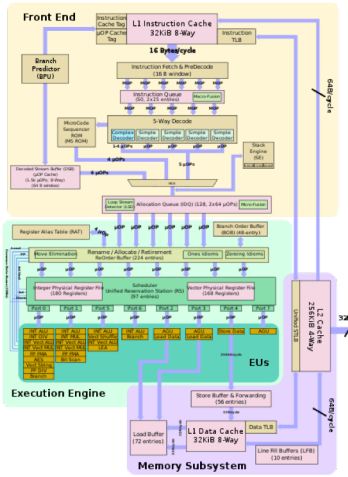




- Every CPU has an architecture as well:





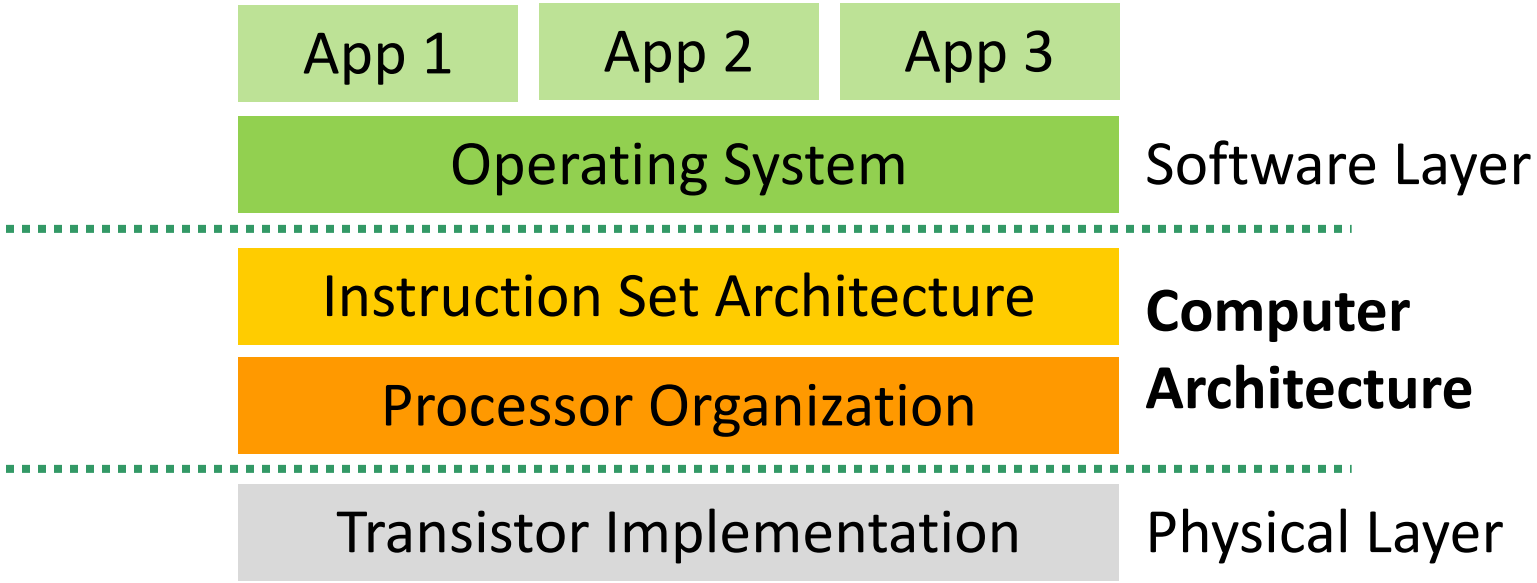




ISA: the architecture of an instruction set

- At a high-level: how a computer is built
  - Computer here meaning the processor (CPU)
- You probably heard of a similar term before: ISA
  - ISA (Instruction Set Architecture)
- Review: what is defined by an ISA?
  - *Set of instructions* usable by the computer
  - *Set of registers* available in the computer
  - Other functional attributes
- What is *not* defined by an ISA?
  - *Speed* of computer
  - *Energy efficiency* of computer
  - *Reliability* of computer
  - Other performance attributes

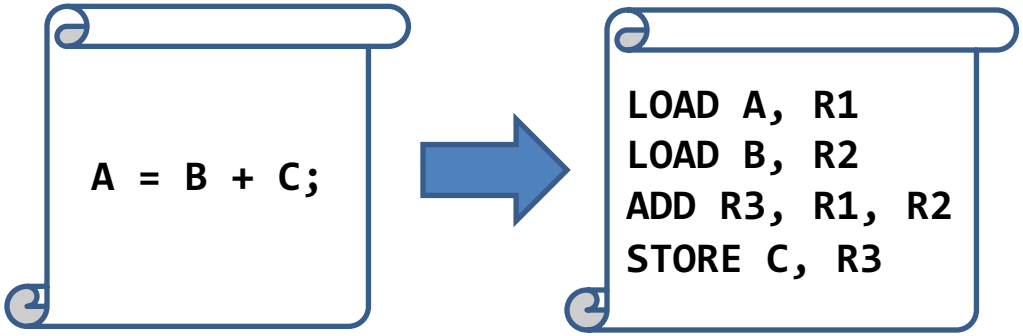
Computer Architecture = ISA+Processor Organization



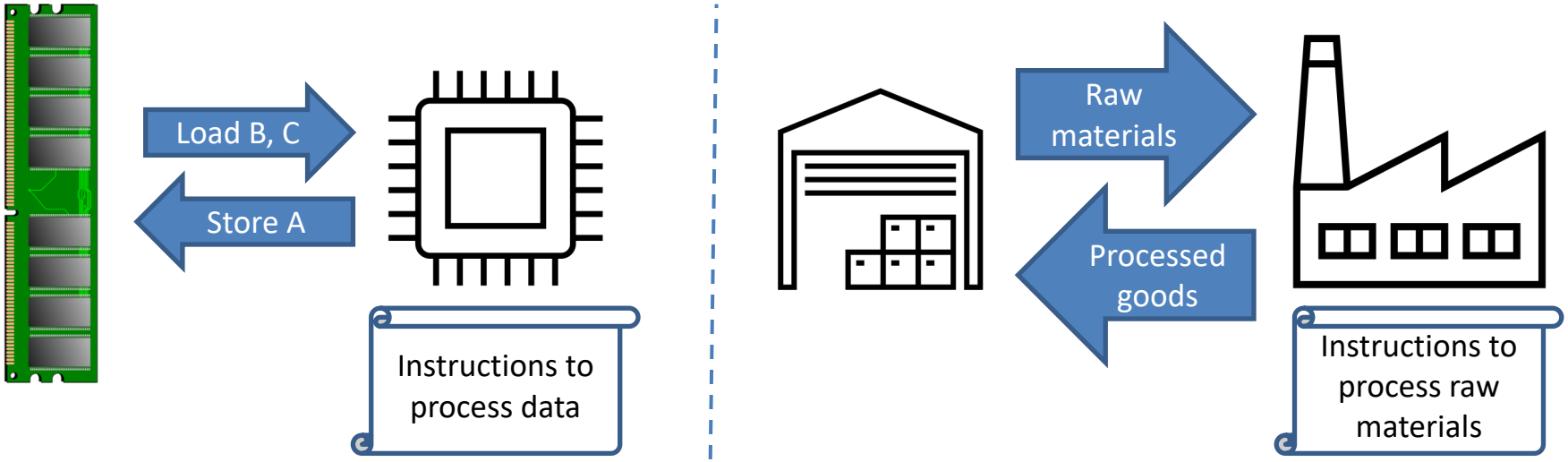
- Computer Architecture = ISA + Processor Organization
  - Processor organization is also called *Microarchitecture*
- Given an ISA, performance is determined by:
  - Processor organization (internal design of the processor)
  - Transistor implementation (semiconductor technology)

A CPU is like a factory

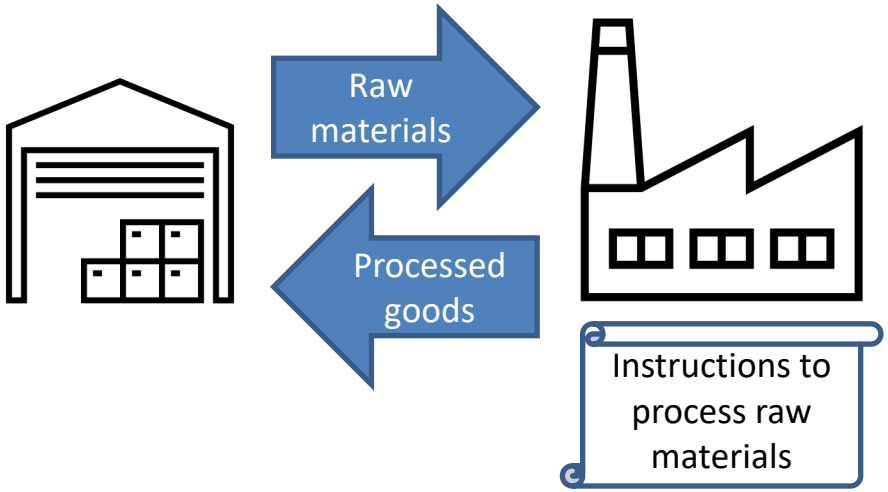
- A program written in Python, Java, or C is converted to instructions



- A CPU processes data according to instructions, just like a factory



Factory efficiency depends on factory design

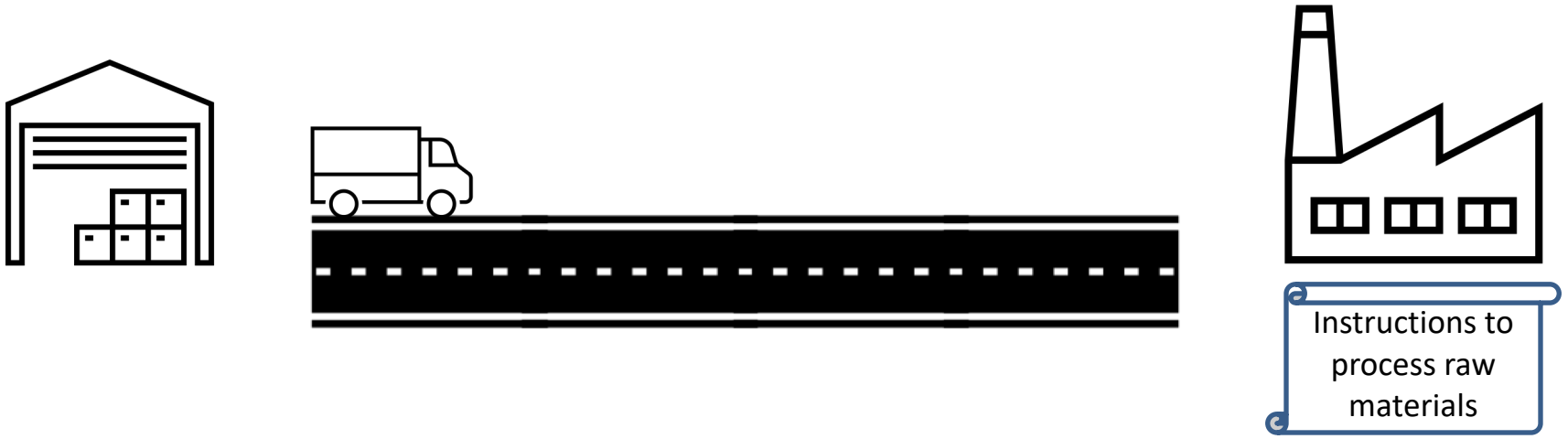


1. How efficiently goods are moved between warehouse and factory

2. How efficiently workers process goods in factory

How to design a shirt factory
Part 1: Logistics

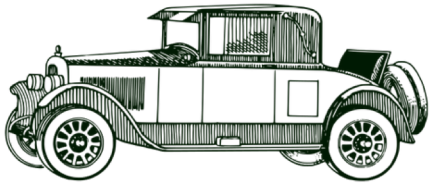
You cannot succeed without good logistics



- If you can't move goods in a timely manner your output will suffer
- Factory efficiency doesn't matter if logistics is the bottleneck
  - If deliveries are slow (high memory latency)
  - Or do not arrive in sufficient quantities (low memory bandwidth)

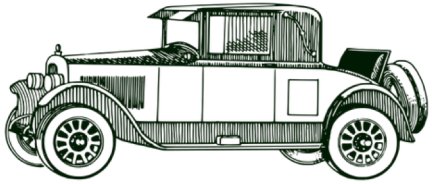
Memory Latency vs. Memory Bandwidth

- Memory bottleneck comes from two sources:
  - **Memory latency**: hours to handle a single delivery

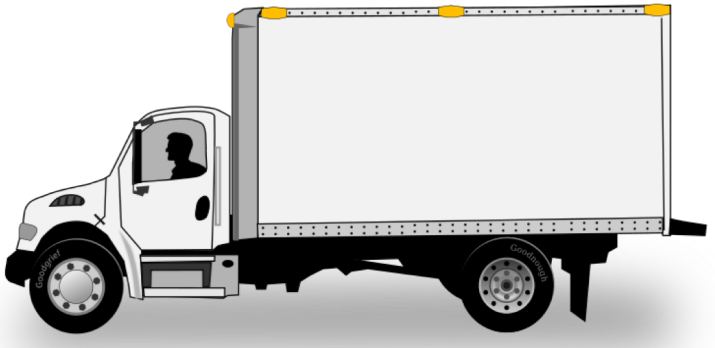


vs.

  - **Memory bandwidth**: maximum volume handled per hour



vs.



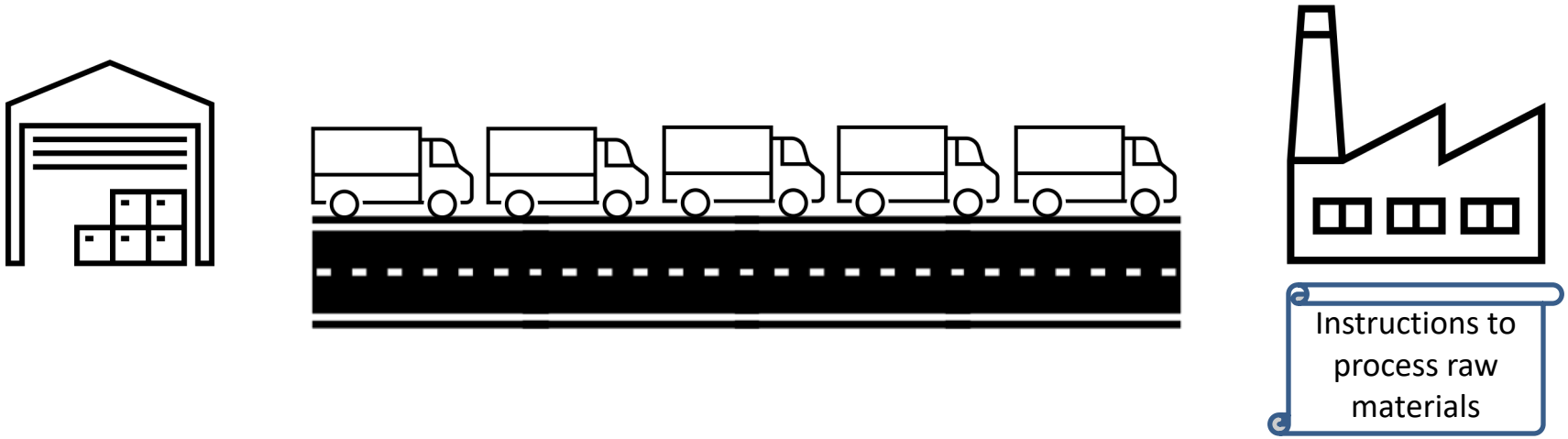
Memory Latency vs. Memory Bandwidth

- Race cars are rarely used for logistics. Trucks are. Why?

1. Can place orders ahead of time to arrive just in time (*prefetching*)
2. Or the factory can produce something else while waiting (*scheduling*)

- There are ways to circumvent latency so it does not impact performance.
  - Except when neither prefetching or scheduling works

Memory bandwidth puts a cap on performance



- If the road is fully utilized but still can't keep up with demand…
  - Factory needs 5 trucks / minute but road handles 1 truck / minute
- No amount of pre-ordering or scheduling can alleviate this problem

→ Unlike latency, bandwidth puts a hard limit on performance!

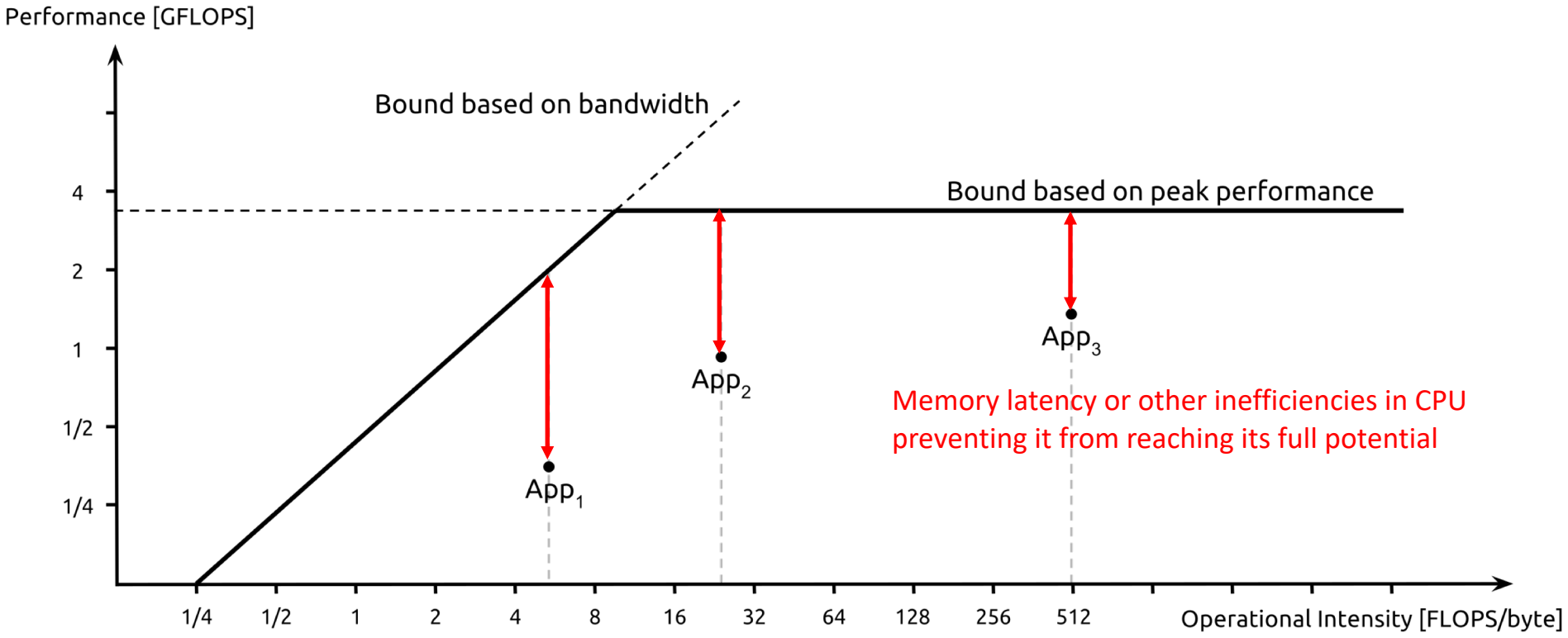
Memory bandwidth puts a cap on performance

- When you get a traffic jam, performance is dictated by bandwidth
  - How quickly you pull data in, rather than how fast you process it
  - **Operational intensity** = work (FLOP) per memory access (byte)
  - Performance = work / second
    = work / byte * byte / second
    = operational intensity * memory bandwidth

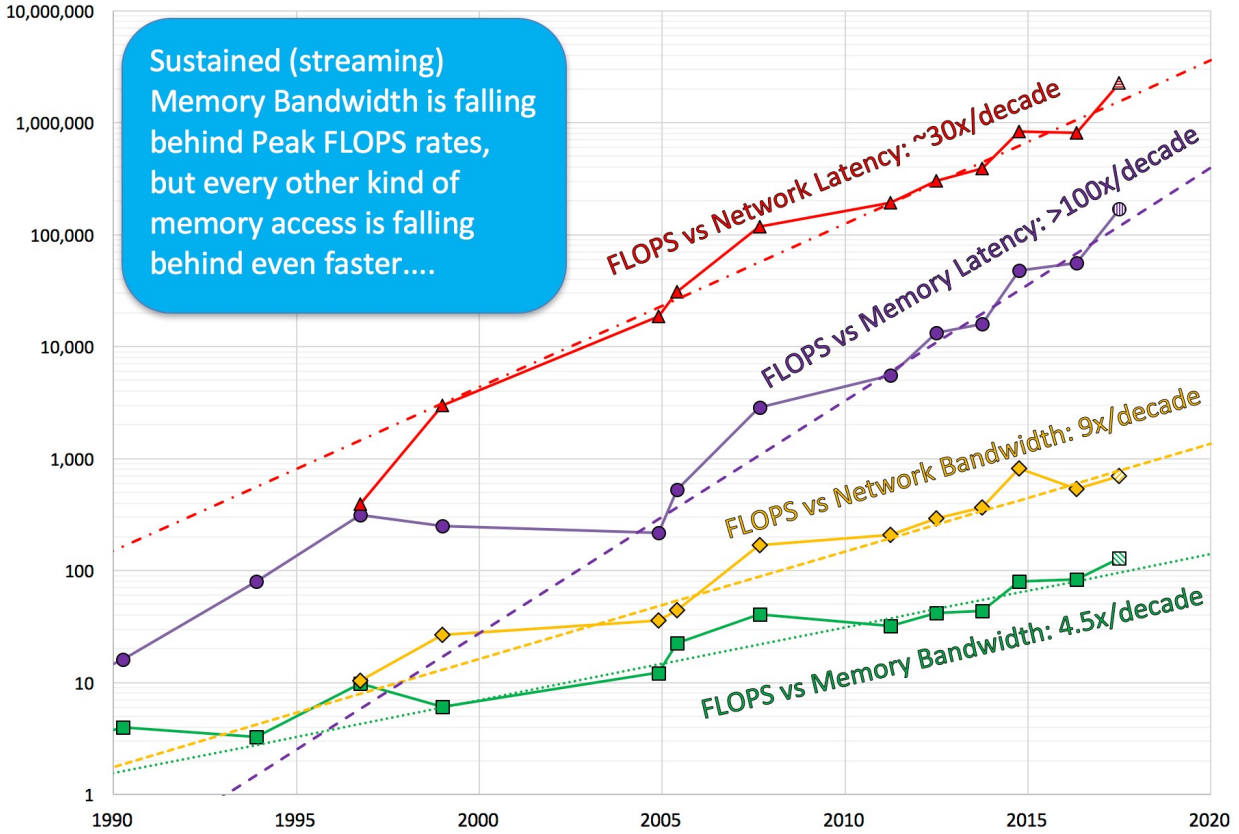
→ Linear relationship between performance and operational intensity

The Roofline Model: A bird's eye view of performance

- **Roofline**: theoretical performance achievable given an intensity
  - Formed by memory bandwidth bounds + peak CPU performance



Memory Wall: widening gap between memory and CPU

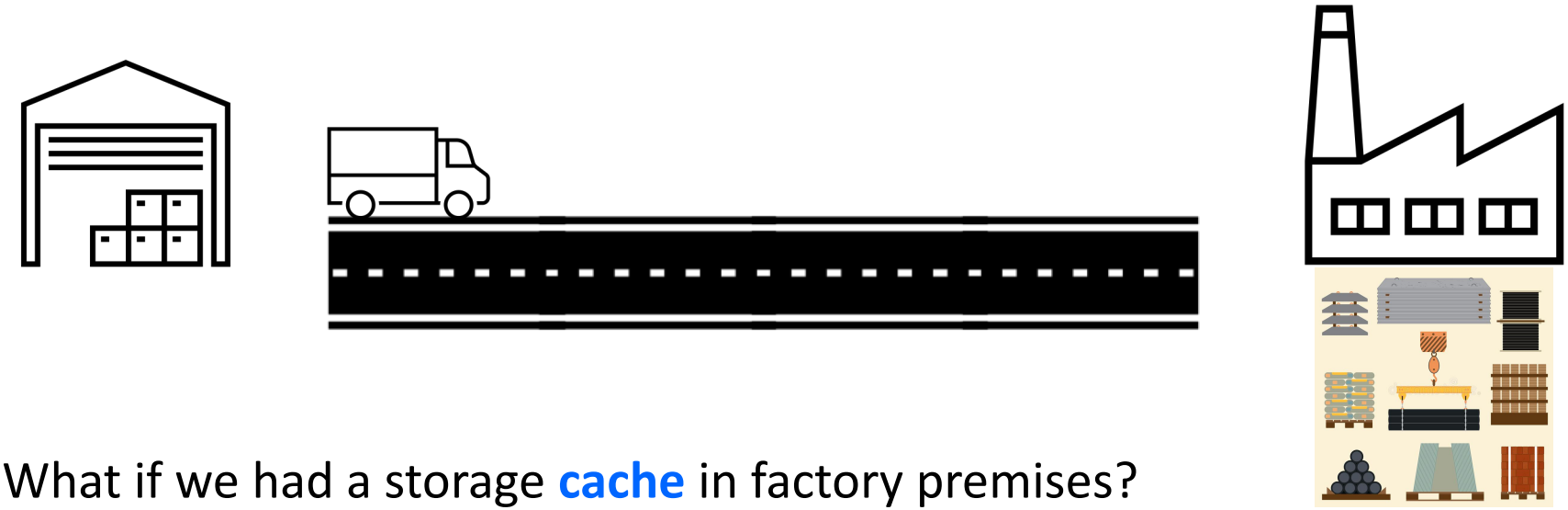


Source: SC16 Invited Talk ""Memory Bandwidth and System Balance in HPC Systems" by John D. McCalpin

- FLOPS = floating point operations per second (CPU speed)

University of Pittsburgh

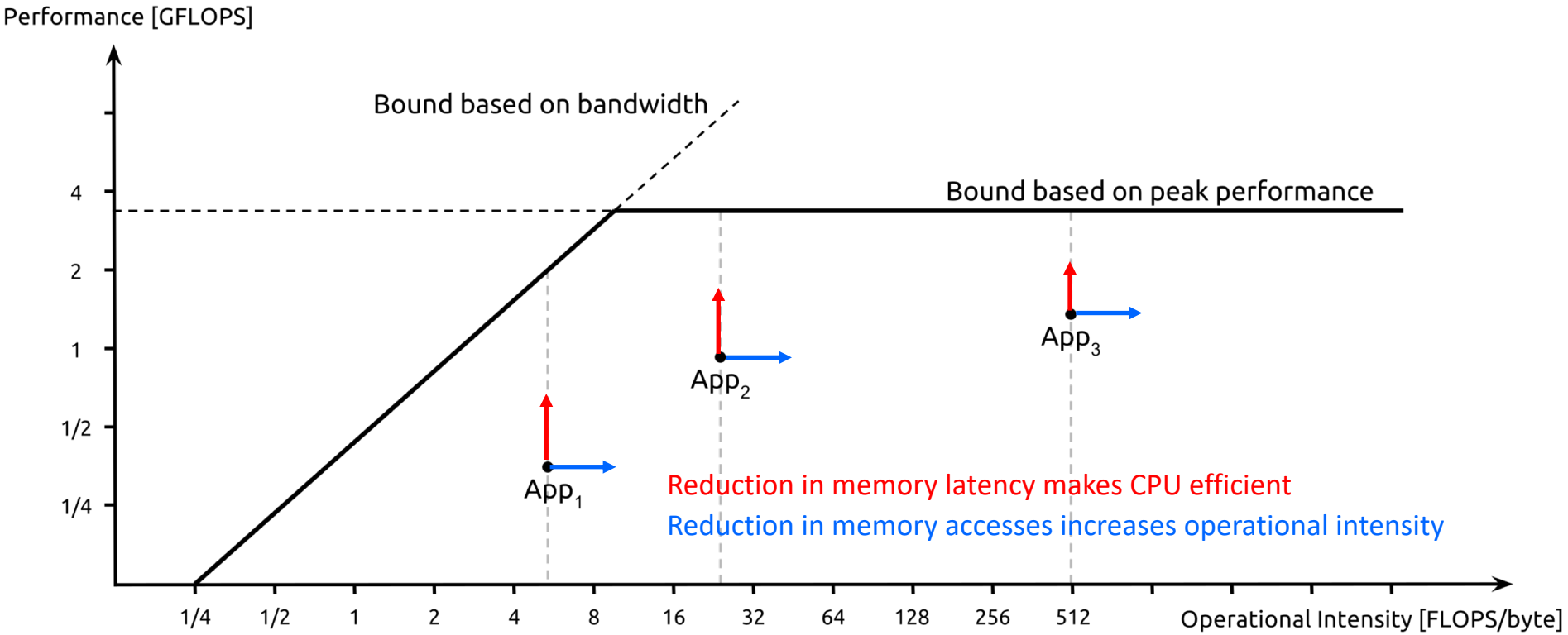
Cache: storage in CPU for frequently accessed data



- What if we had a storage **cache** in factory premises?
  - Bring in a truckload of material on an order (more than needed)
  - Store the rest of the material in cache for future use
- Then, both memory latency and bandwidth problems are alleviated
  - Latency: access is quick when material is found in the cache
  - Bandwidth: less need to go back and forth to the warehouse

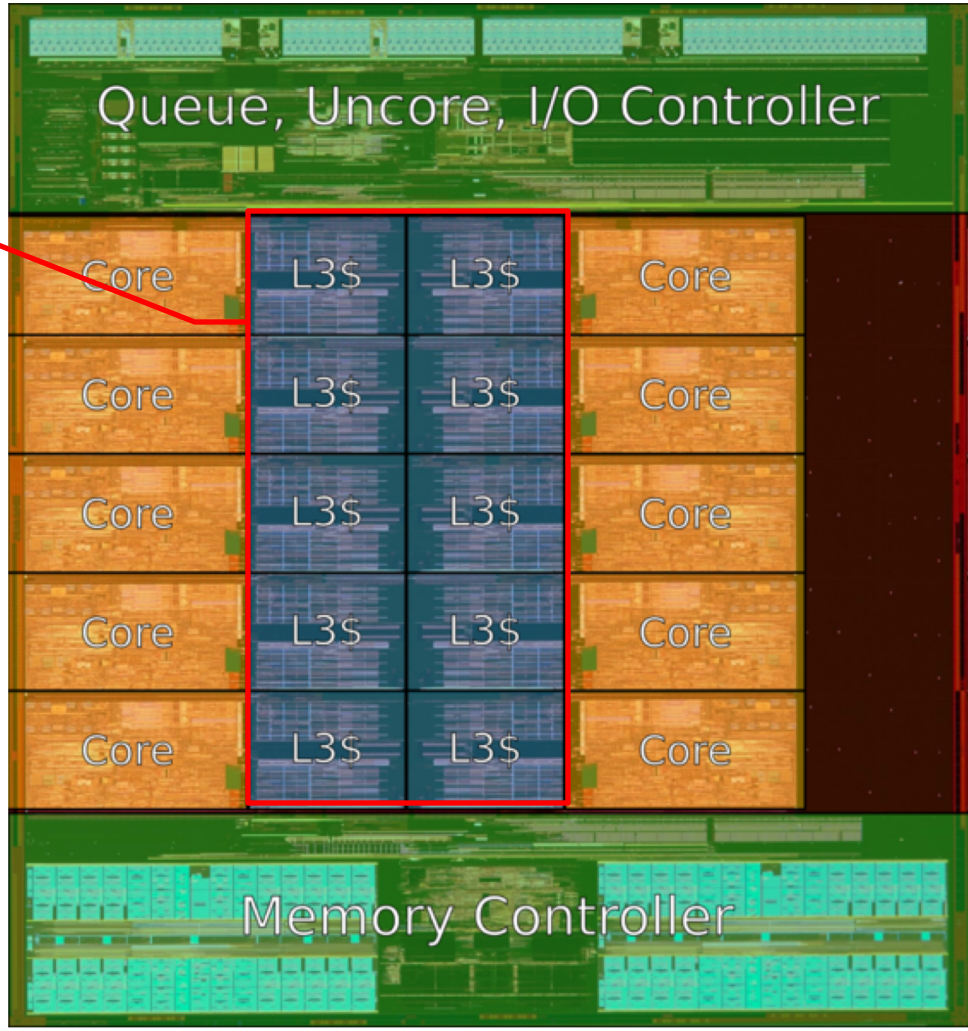
Effect of Cache on Roofline Model

- With caching, apps shift upwards and rightwards



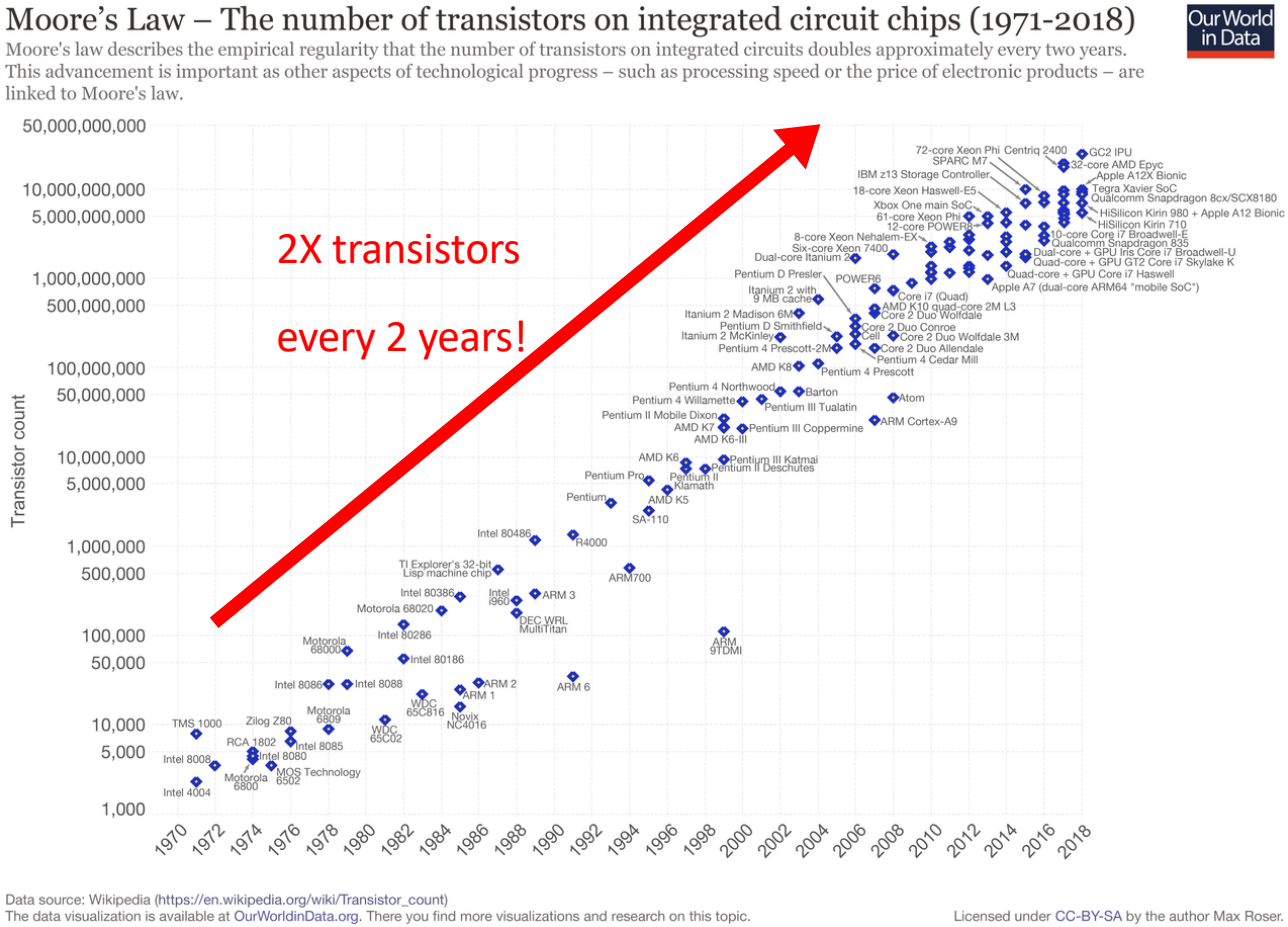
Caches have become bigger and bigger

- As the memory wall got higher and higher, caches got bigger and bigger
- On the right is a diagram of the Intel Xeon Broadwell CPU
  - Caches take up almost as much real estate as cores!
  - A memory access is that painful (10~100X slower)



How to design a shirt factory
Part 2: Worker Utilization

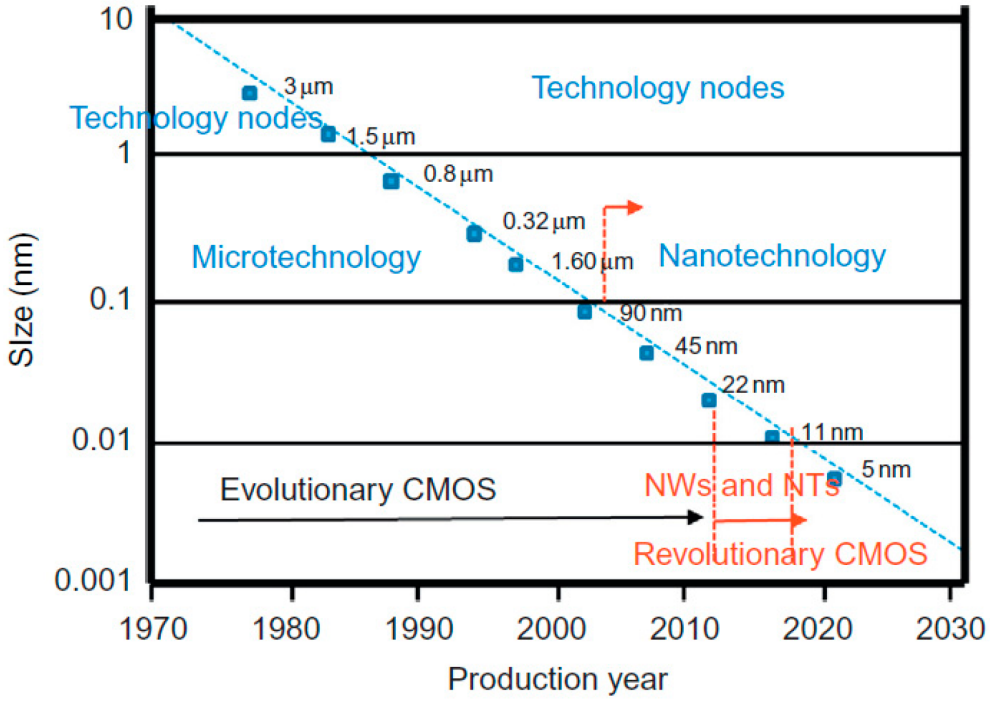
Moore’s Law



Data source: Wikipedia (https://en.wikipedia.org/wiki/Transistor_count)
The data visualization is available at OurWorldinData.org. There you find more visualizations and research on this topic.

University of Pittsburgh

Miniaturization of Transistors



Data source: Radamson, H.H.; He, X.; Zhang, Q.; Liu, J.; Cui, H.; Xiang, J.; Kong, Z.; Xiong, W.; Li, J.; Gao, J.; Yang, H.; Gu, S.; Zhao, X.; Du, Y.; Yu, J.; Wang, G. Miniaturization of CMOS. *Micromachines* **2019**, *10*, 293.

- Moore's Law has been driven by transistor miniaturization

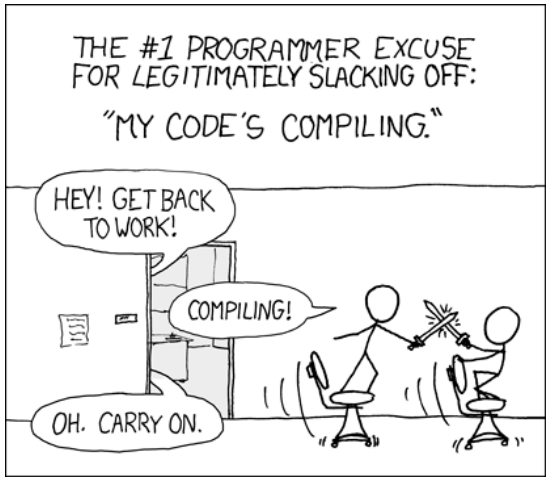
University of Pittsburgh

Our factory is bursting with workers!

- Transistors are cobbled together to create **Functional Units**
  - Things like Add Unit, Multiply Unit, Load Unit, Store Unit

- With billions of transistors, CPUs can contain many Functional Units

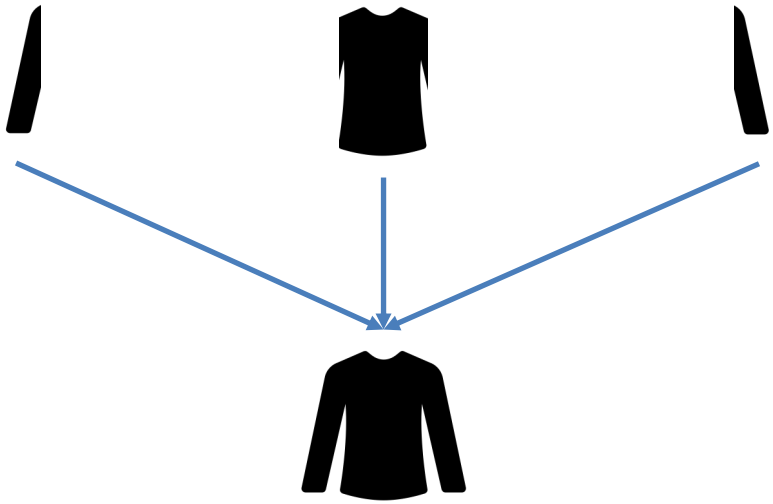
- What is the problem with a factory with a lot of workers?
  - It is hard to keep all the workers productive!
  - Workers always find an excuse to slack…
  - Same goes for Functional Units

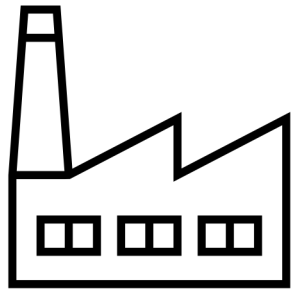


Shirt manufacturing instructions

- Let's assume one man hour for each instruction.
- How many workers would you hire?

- **Data Dependence Graph** for shirt manufacturing



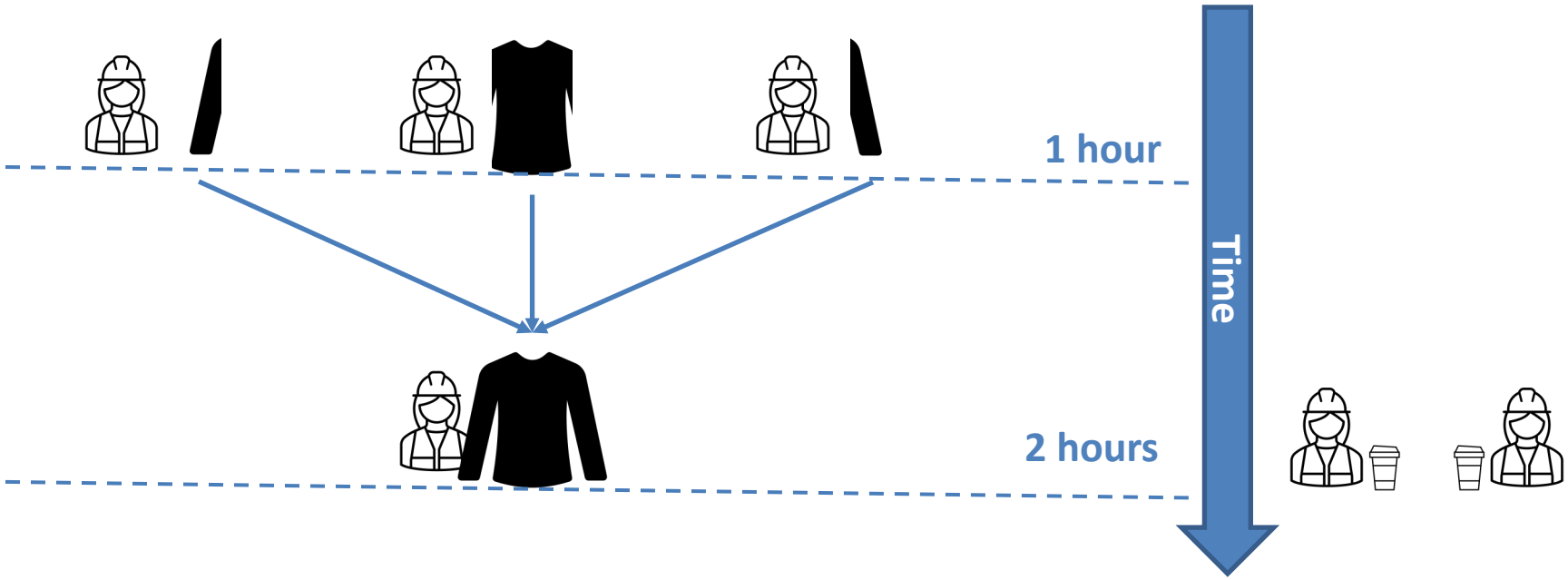


[Instructions]

1. Cut fabric for left sleeve
2. Cut fabric for right sleeve
3. Cut fabric for body piece
4. Stitch pieces together

3 workers = fastest but low utilization

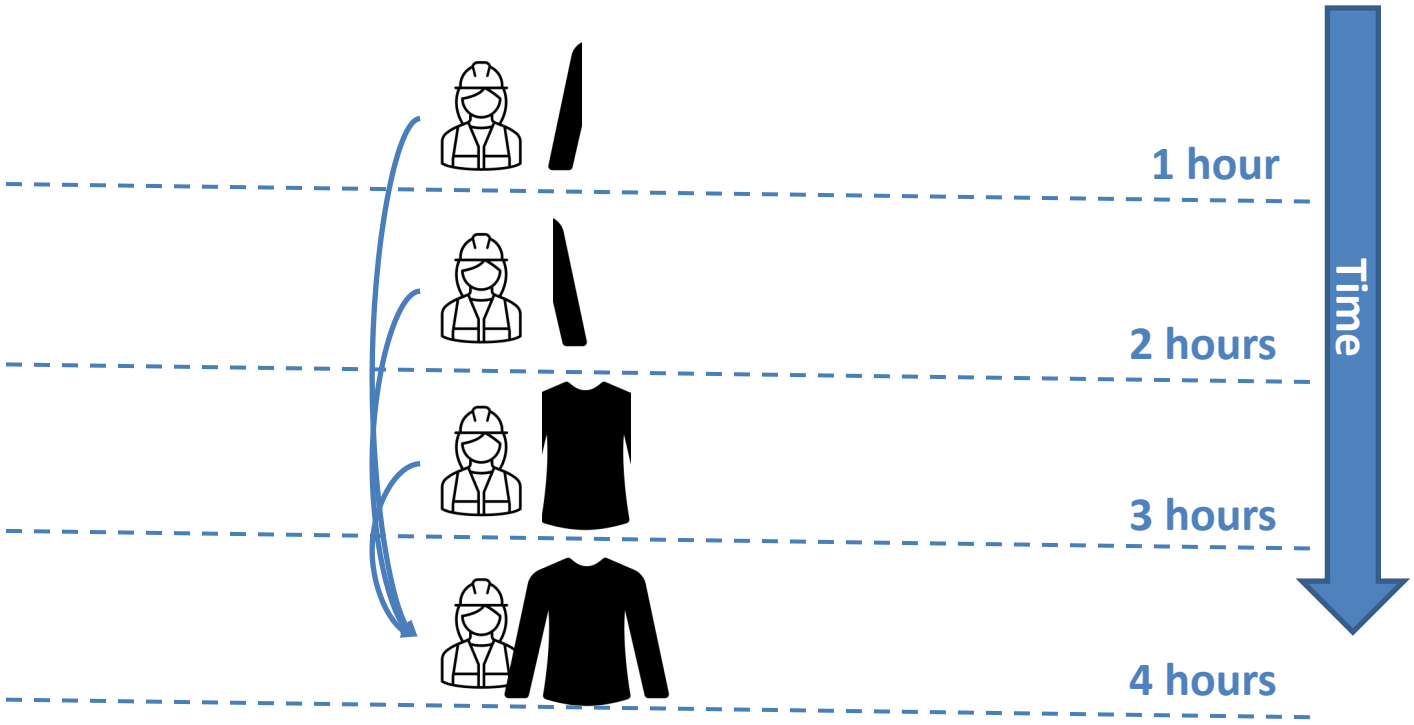
- Can produce a shirt in just 2 hours:



- During the 2nd hour, 2 workers were slacking over coffee. Oh no!
- Utilization = 4 steps / 6 worker-hours * 100 ≈ 67%

1 worker = slow but highest utilization

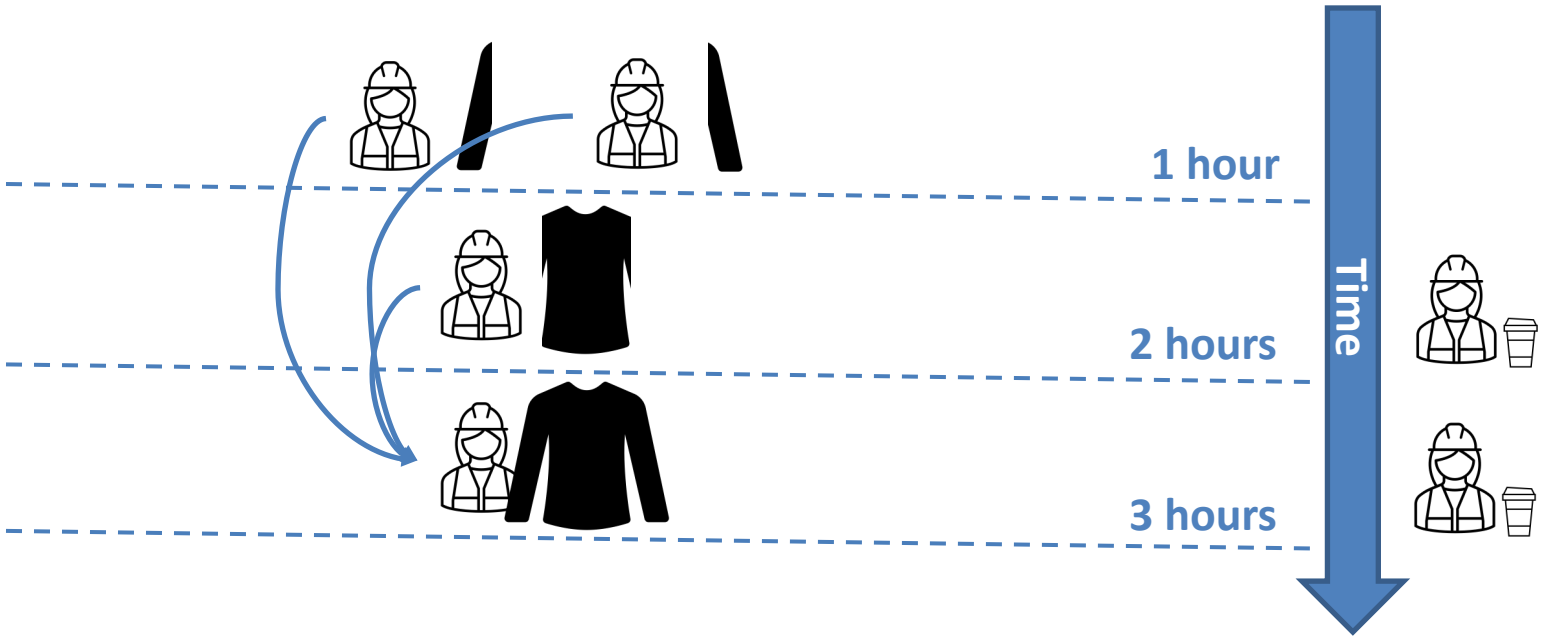
- Needs 4 hours to produce a shirt, but no slacking!



- Utilization = 4 steps / 4 worker-hours * 100 = 100%

2 workers = mediocre speed and low utilization

- Needs 3 hours with some slacking:



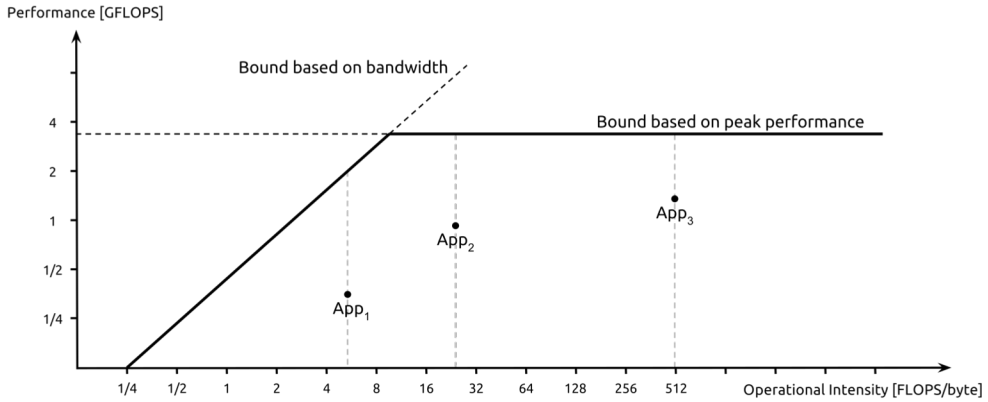
- Utilization = 4 steps / 6 worker-hours * 100 ≈ 67%
- Same utilization as 3 workers but just slower. Fail!

What are the implications of low utilization?

- Factory: leads to high cost for producing the same shirt
- CPU: leads to **high energy use** for executing the same instructions
  - Functional units still use energy while slacking off

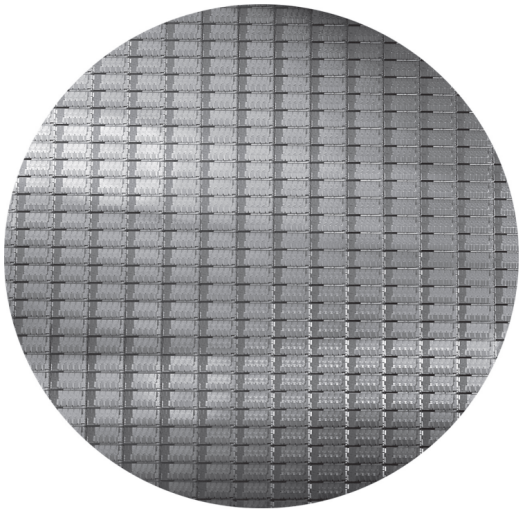
- So, should we go with 3 workers or 1 worker? It depends!
  - 3 workers = Gaming CPU that runs fast but uses lots of power
  - 1 worker = Mobile CPU that runs slower but uses less battery life

- Low utilization is what prevents apps from reaching full potential

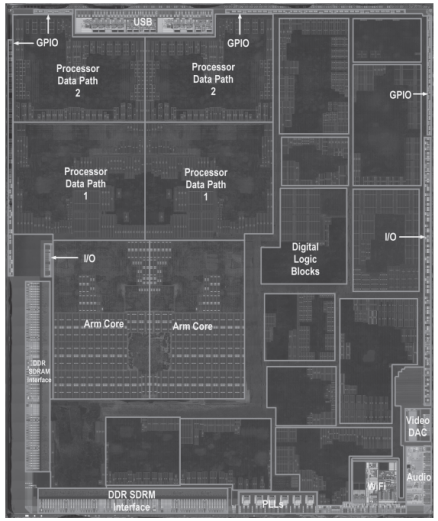


Scope of Class

- Physical layer is beyond the scope of the class

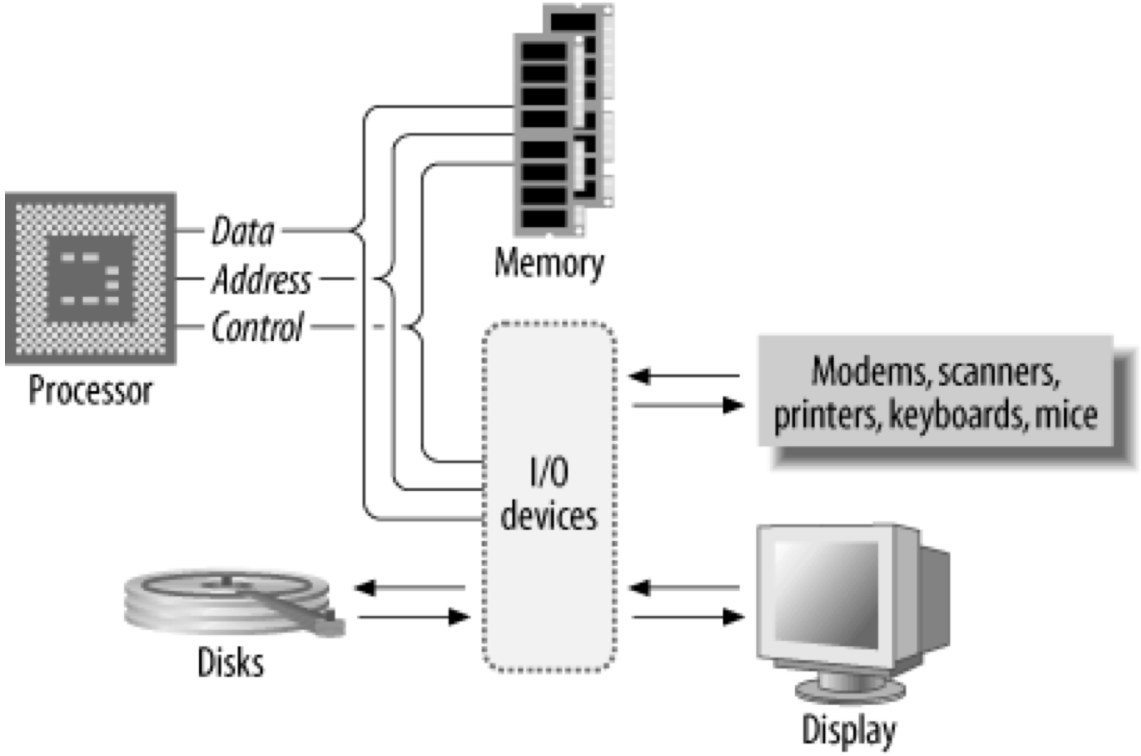


- We will focus mostly on processor organization
  - And how performance goals are achieved



Scope of Class

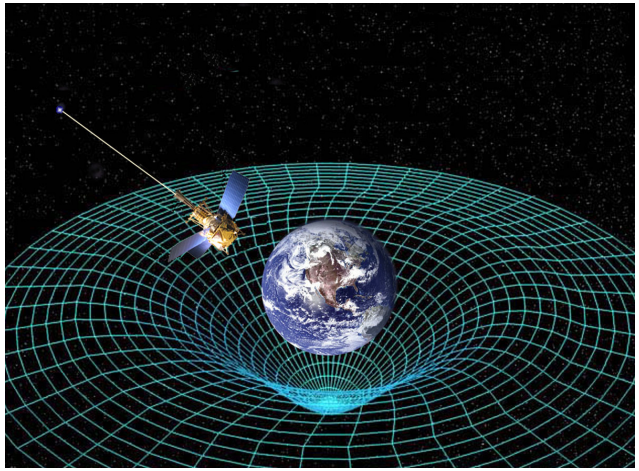
- Computer architecture is part of system architecture



- Other components beside processor is beyond the scope

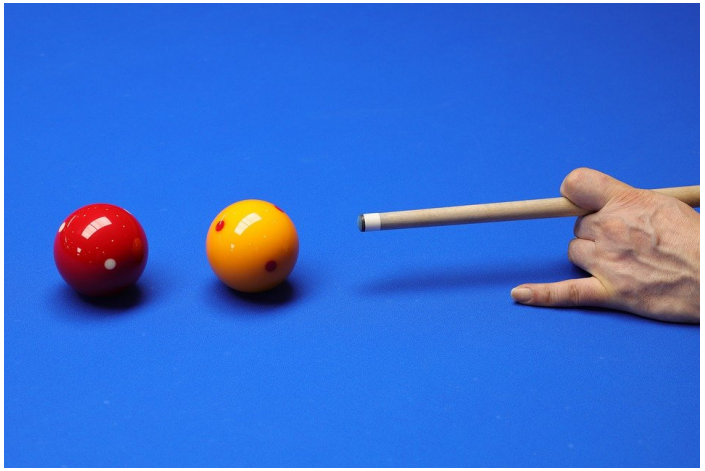
Two Forces on Computer Architecture

1. Application pull



- Market forces pull architecture towards popular applications

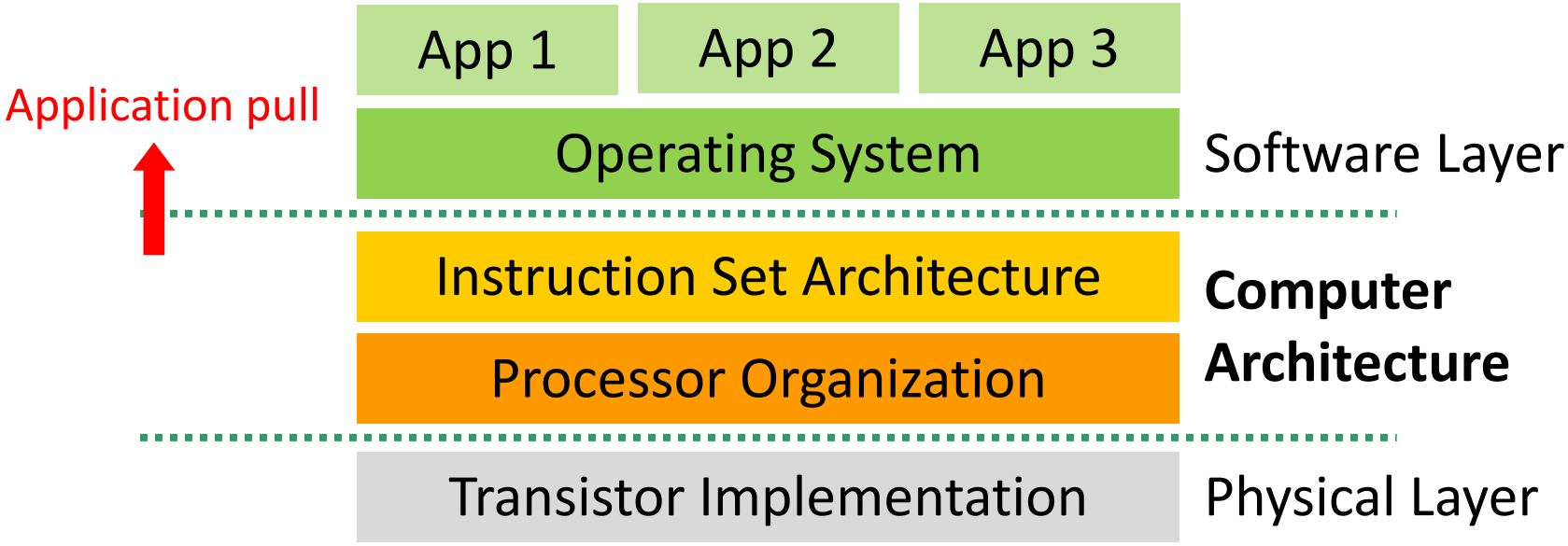
2. Technology push



- Advances in silicon technology push architecture to change

Application Pull

- Different applications pull in different directions



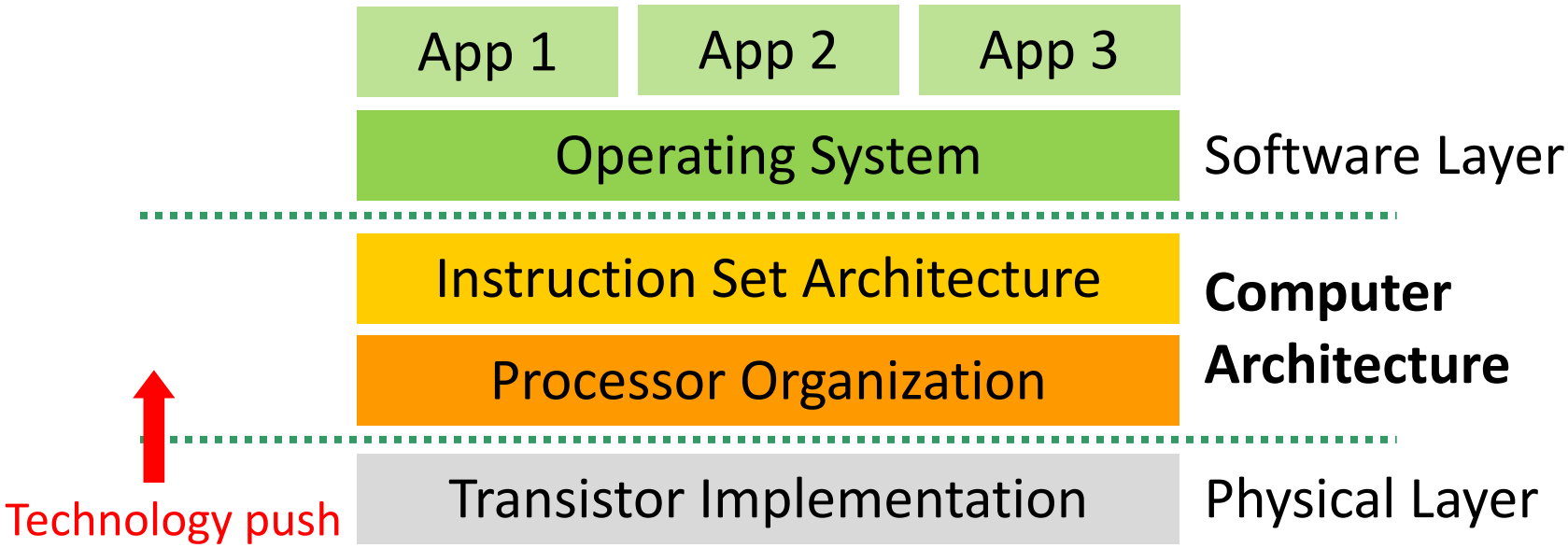
  - Real-time app (e.g. Game): *Short latency*
  - Server app: *High throughput*
  - Mobile app: *High energy-efficiency* (battery life)
  - Mission critical app: *High reliability*
- An app typically has multiple goals that are important

Application Pull

- Some goals can be incompatible
  - E.g. Speed and energy-efficiency are incompatible
    - ➢ Running is faster than walking but uses more energy
    - ➢ A Ferrari is faster than a Prius but has worse fuel efficiency
  - E.g. Reliability is incompatible with many other goals
    - ➢ If you use redundancy, you use twice the amount of energy
- Even when sharing a goal, apps have unique needs
  - Scientific apps need lots of *floating point units* to go fast
  - Database apps need lots of *memory cache* to go fast
- An architecture is a compromise among all the apps
  - When app achieves market critical mass, designs diverge (Mobile chips / Server chips / GPUs / TPUs diverged)
  - Sometimes even ISAs diverge (GPUs and TPUs)

Technology Push

- Trends in technology pushes architecture too



  - Trends can be *advances* in technology
  - Trends can be *constraints* technology couldn't overcome
- "Technology" in CPU design refers to the physical layer
  - Manufacturing technology used for transistor implementation