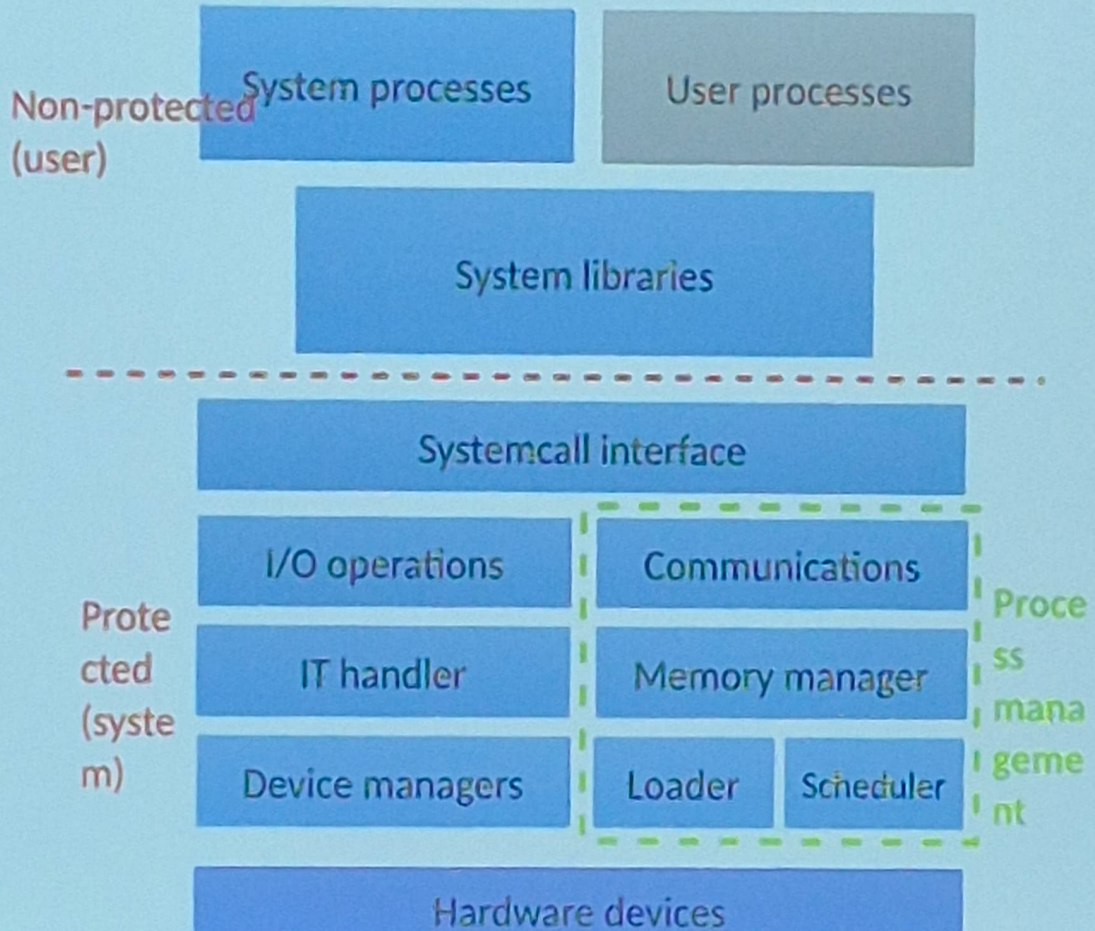
The kernel

- Controls user-mode processes
  - life-cycle management (creation, operation, termination)
  - event management (passes hardware and software events to processes)
  - provides common services to simplify software development

- Manages resources
  - set ups hardware elements
  - provides functions to access them
  - handles their events (e.g. interrupts)
  - resolves conflicts, allows simultaneous access

- Keeps the system reliable and secure
  - protects resources from programming errors and malicious requests
  - separates processes from each-other to protect them
  - provides security functions to user-level programs

The main blocks of the OS and the kernel (recap)

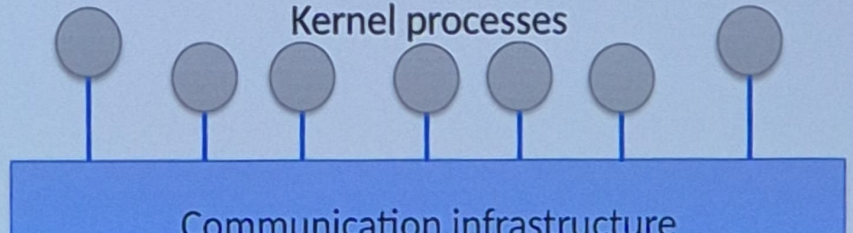


What's the problem with kernel structures?

- When did the TV say?
  - Don't turn me off, 220 important updates are pending
  - Needs reboot, because updates were performed – while watching a movie
  - Pay €400 or all of your channels will be encoded
- When can a vehicle control system crash?
  - Because a dirty CD is inserted
  - One of the components are changed during a maintainance
- Why do such phenomena occur when using an OS?
  - Complex systems, numerous devices and functions
    - Real and imaginary urges to develop more functions
  - Monolithic kernels are typical
    - One mistake causes the whole system to struggle or crash
    - Hard to isolate and repair (debug) the problems
    - Hard to maintain the integrity of the system
    - A bug can cause security weaknesses (for the whole system)
  - Programmers are not super humans
    - 1 small bug / 100 lines of code is typical

What can be done to amend the situation?

- Isolate the sensitive parts, keeping the monolithic structure
  - Most of the problems are caused by device drivers
    - Isolate them: sandboxing
    - Armored OS: wrap the device driver functions in a protective function which is able to detect problems and, for example stop the driver function
    - A user-mode agent managing the detected problems
  - Decompose the system using virtualization (see later in this semester)
    - Multiple virtualization methods
    - Causing performance drop
  - These techniques are often used in the current OS-s
- Throw away the monolithic structure
  - Build the kernel as a distributed system (workers and communication)
  - Only the most essential functions use kernel mode



The concept of microkernel

- Distributed system in general
  - Consists of independent units (computational and storage)
  - It is transparent to the user, the only differences are in the internal operation
  - Can be distributed physically
- The microkernel as a distributed system
  - A kernel mode task manager is necessary to perform the distributed tasks
    - Tasks: memory management and scheduling
    - Distributed: the workers are communicating and cooperating
    - (optionally the most relevant device drivers)
  - Individual programs implementing the kernel's other tasks
    - Running in user mode
    - Separated from each other, like every other user task
- Pros and Cons (see Tanenbaum-Torvalds debate)
  - Flexibility: multiple API-s together, dynamic expansion
  - Reliability: only a small section of the code has to be „good" (may be verified by formal methods)
  - Fault tolerant: errors in the user mode programs can be handled by kernel mode section
  - Using the right programming patterns are mandatory: modular coding, interfaces

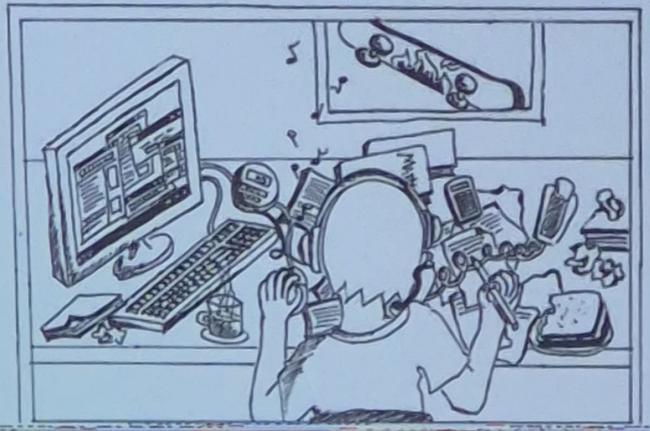
Second generation microkernels

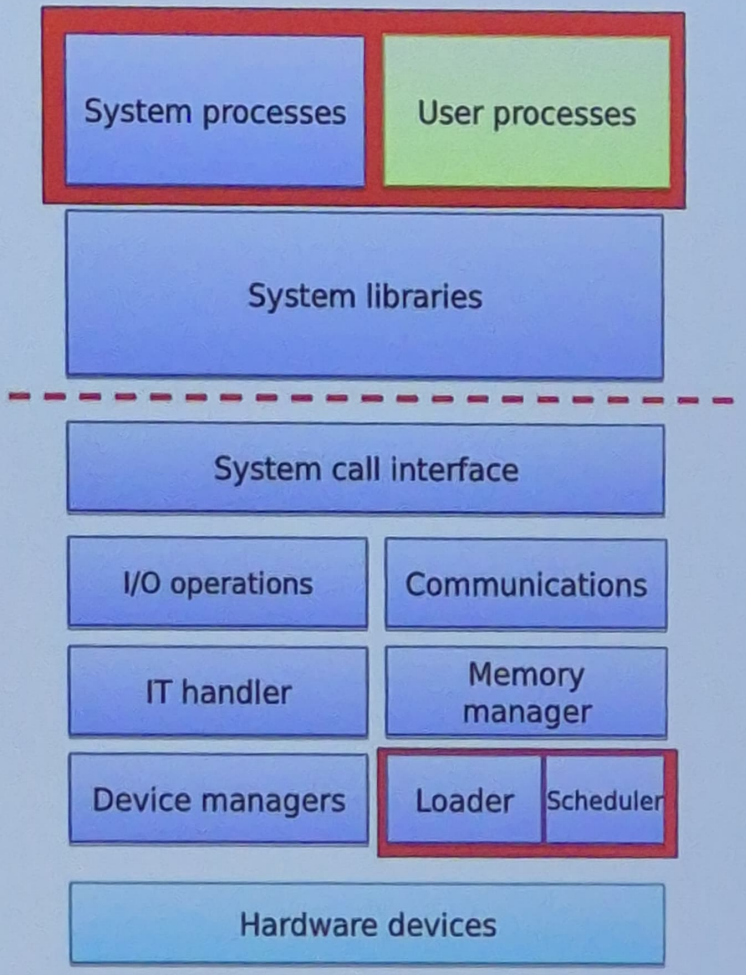
- IPC instead of RPC
  - **RPC** (Remote Procedure Call) are forwarded as
  - **IPC** (Inter Process Communication) messages
  - Which are forwarded by the communication infrastructure/subsystem
  - This method is really slow, compared to system calls
- The second generation microkernel improves the IPC speed
  - **Exokernel**: minimized kernel, simple and fast system calls
  - **L4 microkernel**: very fast IPC (may be forwarded trhough CPU registers)
  - 10-20 times faster than classic microkernel
  - Very few kernel functions (e.g. L4 provides 7 functions)
  - The protected kernel section is small (5-15K LoC)
  - HW dependencies are more important
  - The small kernel makes possible the **formal modeling and verification**
  - **Multiserver**: more than one server are running on the same microkernel
  - **Hybrid kernel**: monolithic kernel over a microkernel
    - OS X XNU: Mach microkernel + BSD UNIX hybrid kernel
    - Windows is containing microkernel elements, but it isn't microkernel based

The OS as a control program

The kernel as a control program – overview

- The Operating System
  - helps solving user's tasks
  - **control program**
  - **resource allocation**
- Expectations
  - handles multiple tasks
  - reliable, secure
  - meets users' requirements





What kind of tasks?

- All kinds of computers (from servers to small devices)
- → Tasks of all kinds → huge variety of software tools

Synaptic Package Manager

54628 packages listed, 2019 installed,

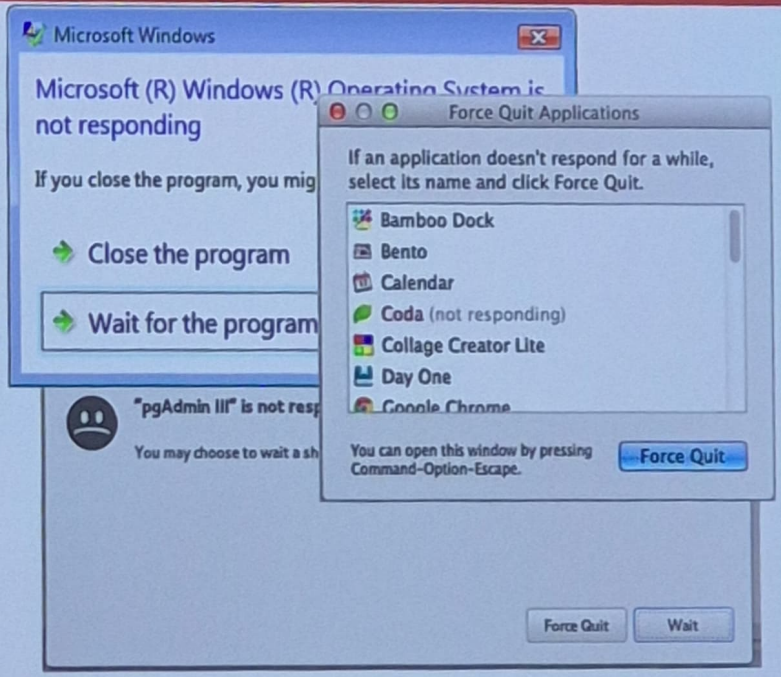
```
> yum list all | wc -l
22747
```

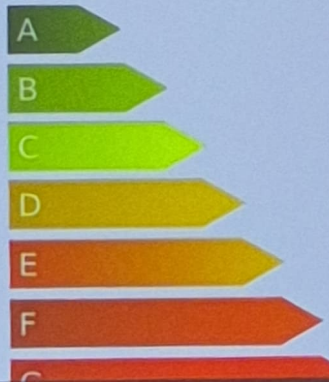
Let's try to characterize tasks

- I/O-bound
  - mostly waiting for I/O (reading, writing, events)
  - need less CPU time
  - examples: Web server, File storage, Email etc.
- CPU-bound
  - the CPU is their most required resource
  - need less I/O
  - example: simulations, mathematical algorithms, machine learning etc.
- Memory-intensive
  - need large amount memory
  - enough memory → CPU-bound
  - not enough → I/O-bound (swapping)
  - e.g. large matrix operations, document indexing and search, graph DB, etc.
- The are many others...
  - Real-time

User's expectations

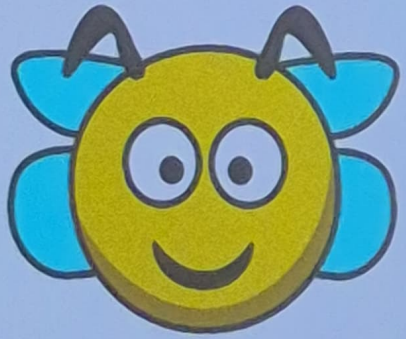
- Wait less
  - **waiting time**
  - **turnaround time**
  - **response time**
- Work efficiently
  - **CPU utilization**
  - **throughput**
  - **overhead**
- Be deterministic



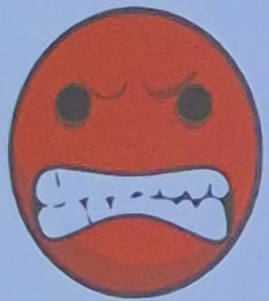


The optimal task execution system

- Ideally...
  - assures that all tasks are performed in time
  - minimizes wait and response times
  - maximizes the resource usage
  - has no overhead



- In practice...
  - some programs run slowly or even freeze
  - the OS require lot of resources
  - the battery depletes fast
  - sometimes even the entire OS freezes for a time
  - we can't answer calls on mobile
  - ...



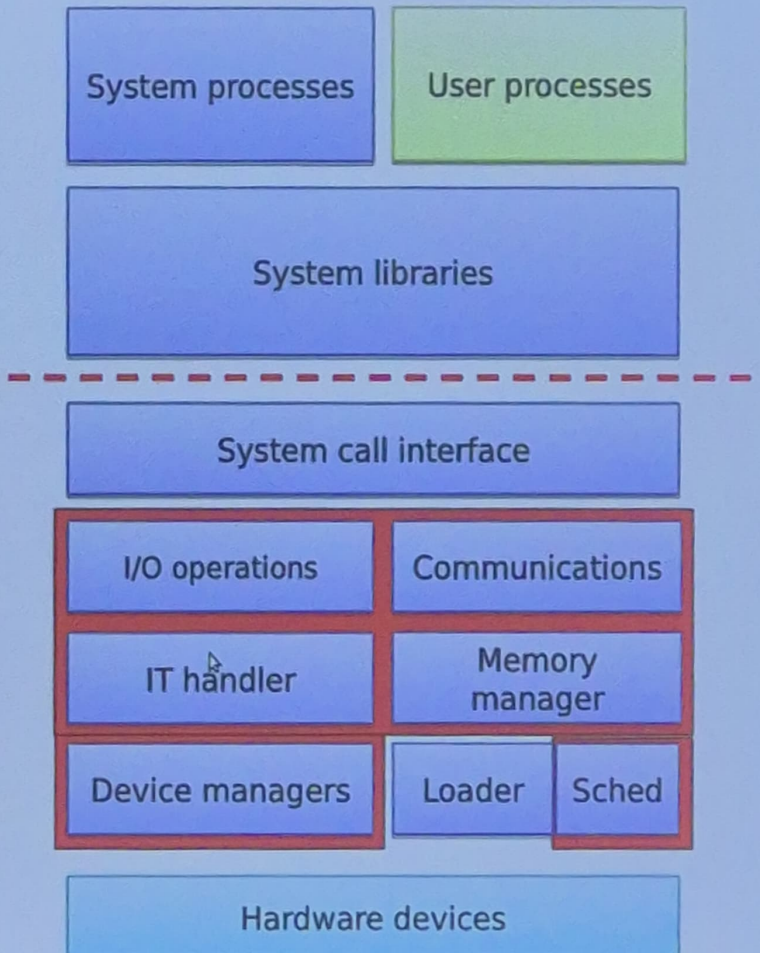
Why is it hard to design a good OS?

- We can't see into the future
  - what tasks are to come
  - what will be their characteristics
- There are many tasks running at the same time
  - they have different requirements
  - and different goals and optimums
  - sometimes the system collapses under the heavy load
- Tasks affect each-other
  - cooperation
  - competition
- There are errors
  - programming
  - hardware

The OS as a resource allocator

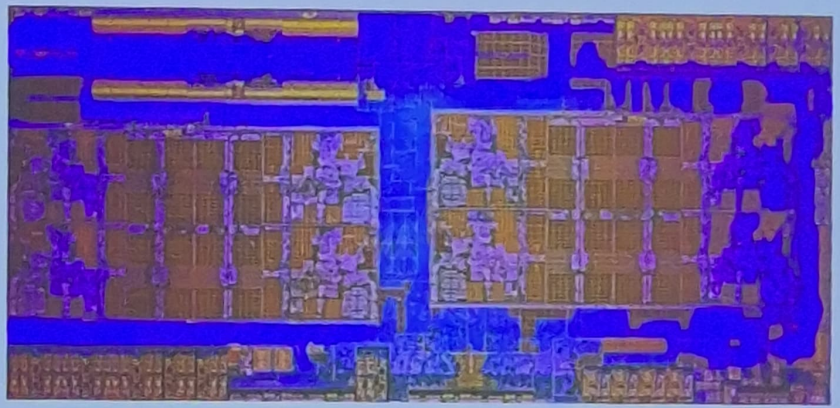
OS as a resource allocator – overview

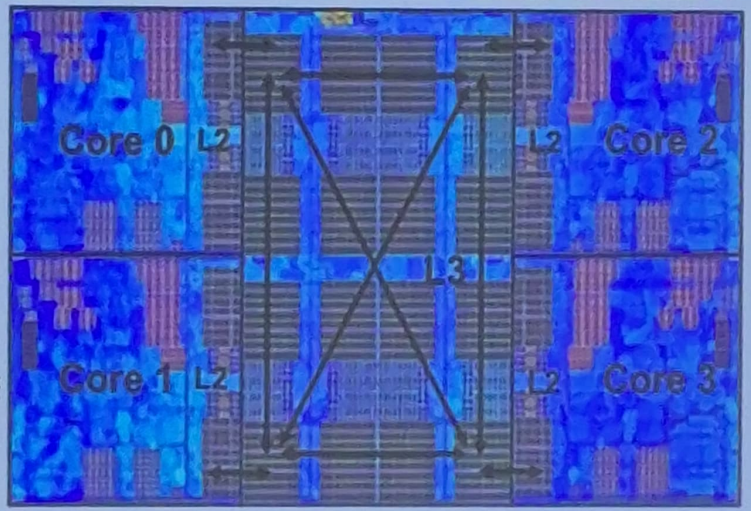
- The Operating System
  - helps solving user's tasks
  - control program
  - **resource allocation**
- Expectations
  - handles multiple tasks
  - reliable, secure
  - **highly utilizes resources**
- Resources
  - processing units (CPU, VGA)
  - system memory
  - storage systems
  - computer peripherals
  - other hardware components
  - software resources



AMD Ryzen

- 8 CPU core
  - 2 „Core Complex” (CCX)
  - Infinity Fabric interconnect
- Core complex
  - 4 CPU core (SMP)
  - 8MB L3 cache
- CPU core
  - 2 thread (SMT)
  - 512K L2 cache, 64K+32K L1 cac
- Many aspects for kernel programmers
  - How to handle CCX
  - Multithreading

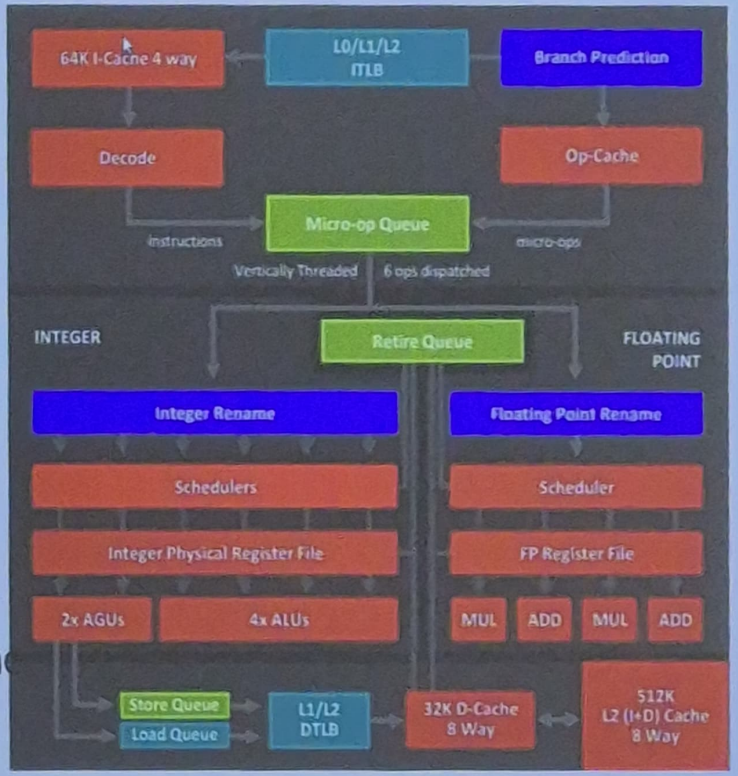




Source: AMD

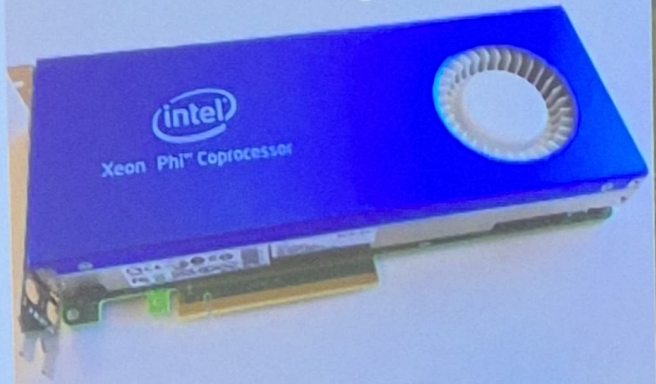
AMD Ryzen

- 8 CPU core
  - 2 „Core Complex” (CCX)
  - Infinity Fabric interconnect
- Core complex
  - 4 CPU core (SMP)
  - 8MB L3 cache
- CPU core
  - 2 thread (SMT)
  - 512K L2 cache, 64K+32K L1 cach
- Many aspects for kernel programmers
  - How to handle CCX
  - Multithreading

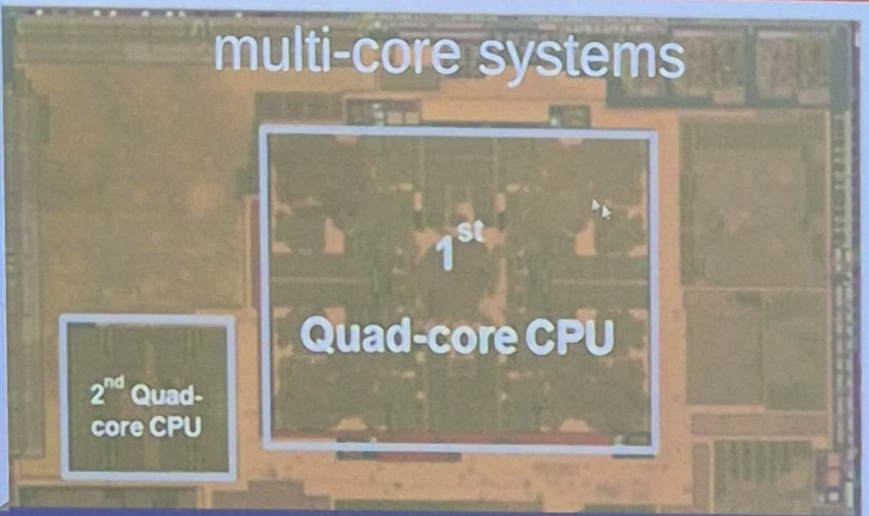


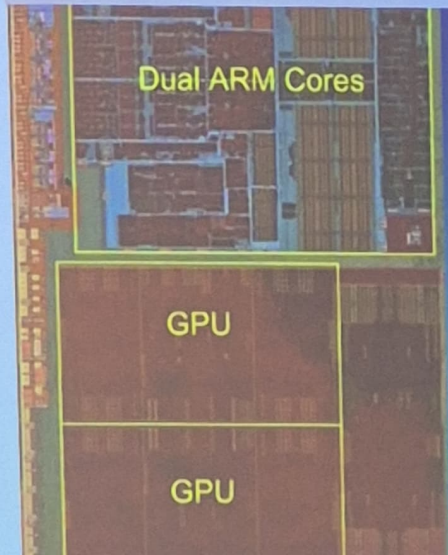
Source: AMD

Heterogeneous



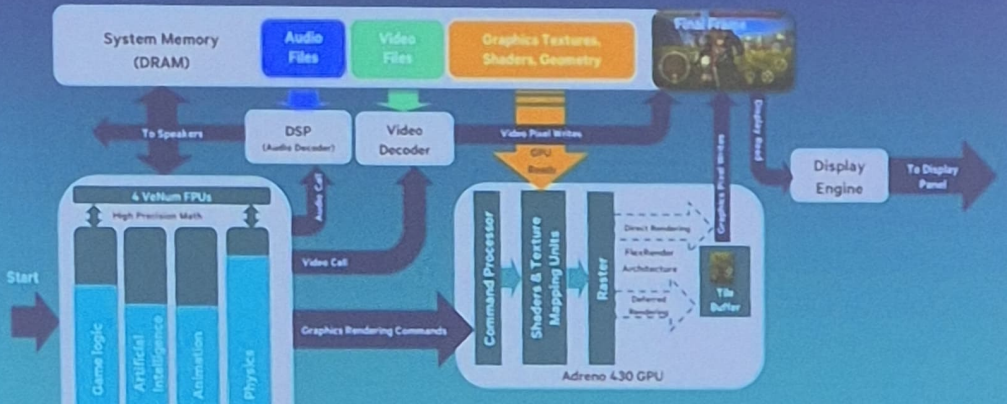
multi-core systems



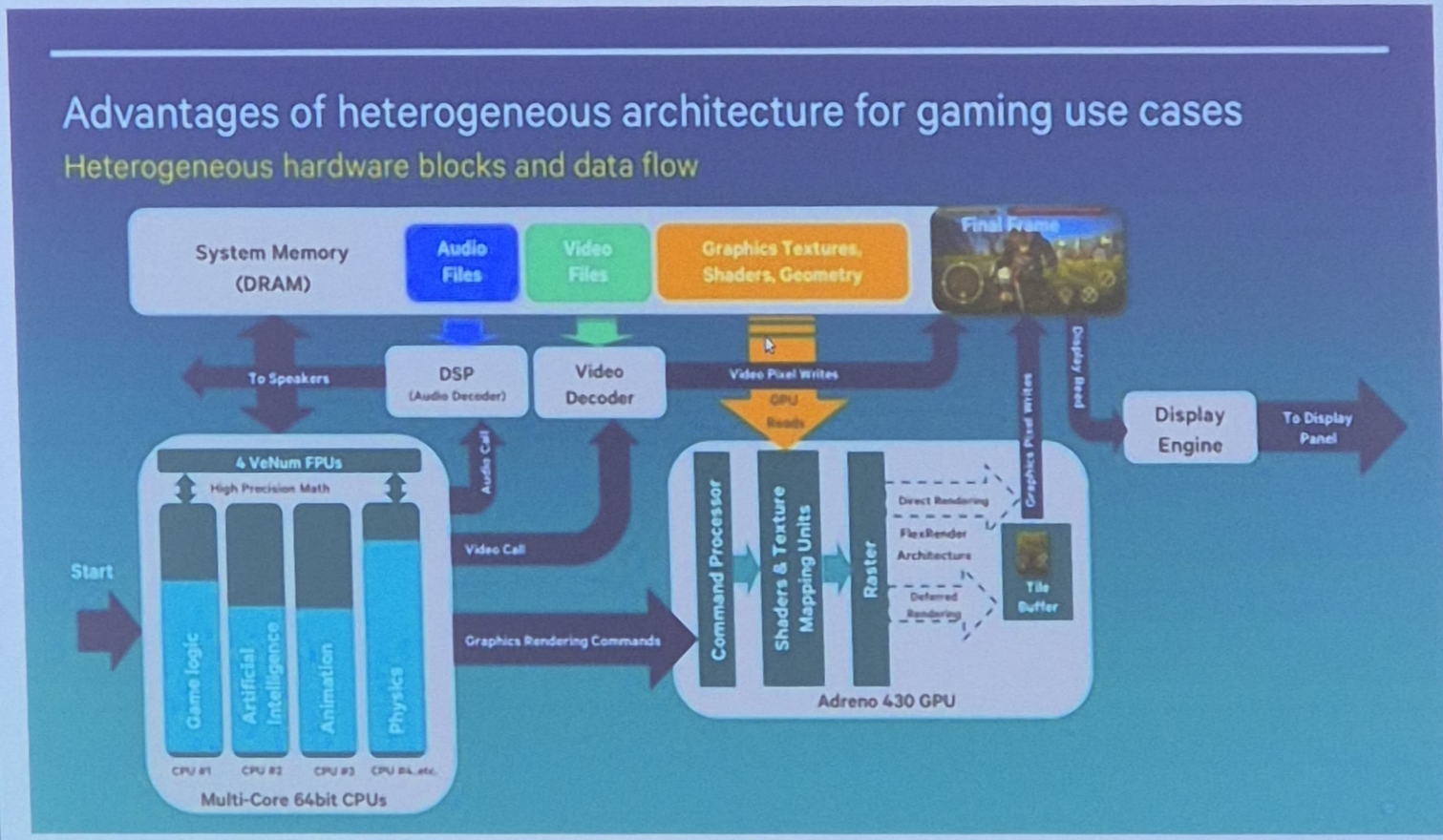


Advantages of heterogeneous architecture for gaming use cases

Heterogeneous hardware blocks and data flow



Assigning tasks to execution units



How to handle memory as a resource?

- Allocation
  - resource: physical memory and storage
  - requested by: user tasks and kernel
- Store tasks
  - program
  - data (dynamic and static)
- Provide memory for the kernel
  - program
  - administrative data
- Security and reliability
  - separation of users' tasks
  - error detection and handling
- Support data sharing