CS252 Graduate Computer Architecture
Fall 2015
Lecture 18: I/O

Krste Asanovic
`krste@eecs.berkeley.edu`
`http://inst.eecs.berkeley.edu/~cs252/fa15`

(I/O) Input/Output

Computers useless without I/O

- Over time, literally thousands of forms of computer I/O: punch cards to brain interfaces

Broad categories:

- Secondary/Tertiary storage (flash/disk/tape)
- Network (Ethernet, WiFi, Bluetooth, LTE)
- Human-machine interfaces (keyboard, mouse, touchscreen, graphics, audio, video, neural,…)
- Printers (line, laser, inkjet, photo, 3D, …)
- Sensors (process control, GPS, heartrate, …)
- Actuators (valves, robots, car brakes, …)

Mix of I/O devices is highly application-dependent

Interfacing to I/O Devices

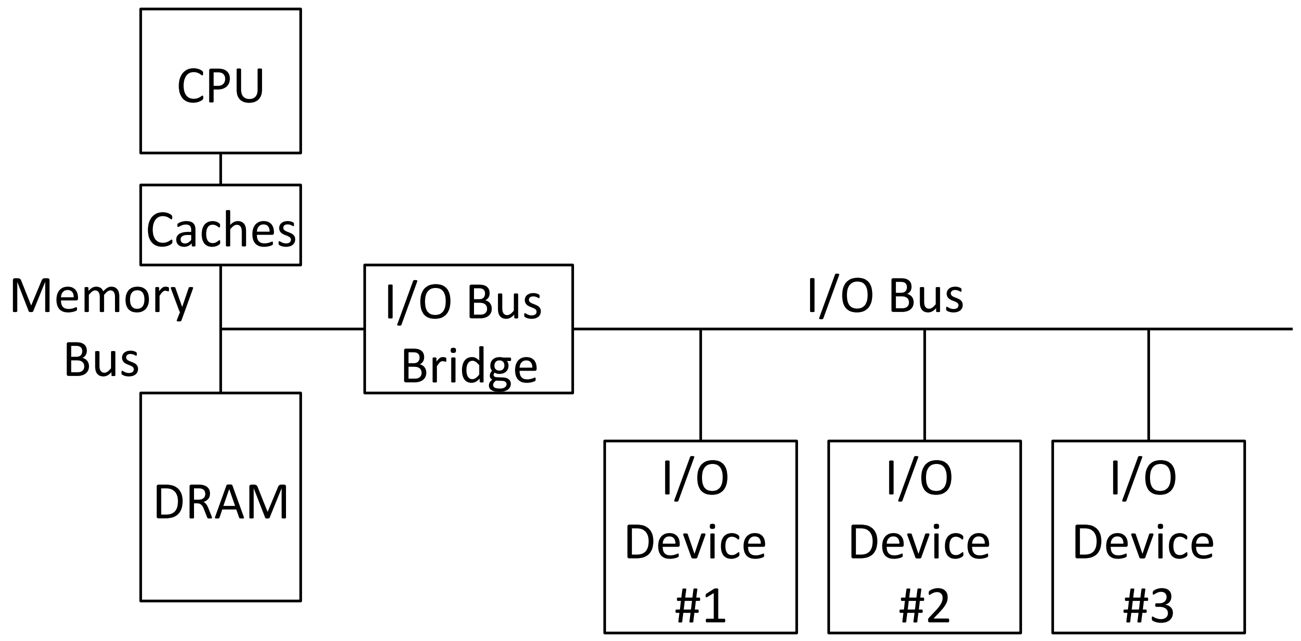
Two general strategies

- Memory-mapped
  - I/O devices appear as memory locations to processor
  - Reads and writes to I/O device locations configure I/O and transfer data (using either programmed I/O or DMA)
- I/O channels
  - Architecture specifies commands to execute I/O commands over defined channels
  - I/O channel structure can be layered over memory-mapped device structure
- In addition to data transfer, have to define synchronization method
  - Polling: CPU checks status bits
  - Interrupts: Device interrupts CPU on event

Memory-Mapped I/O

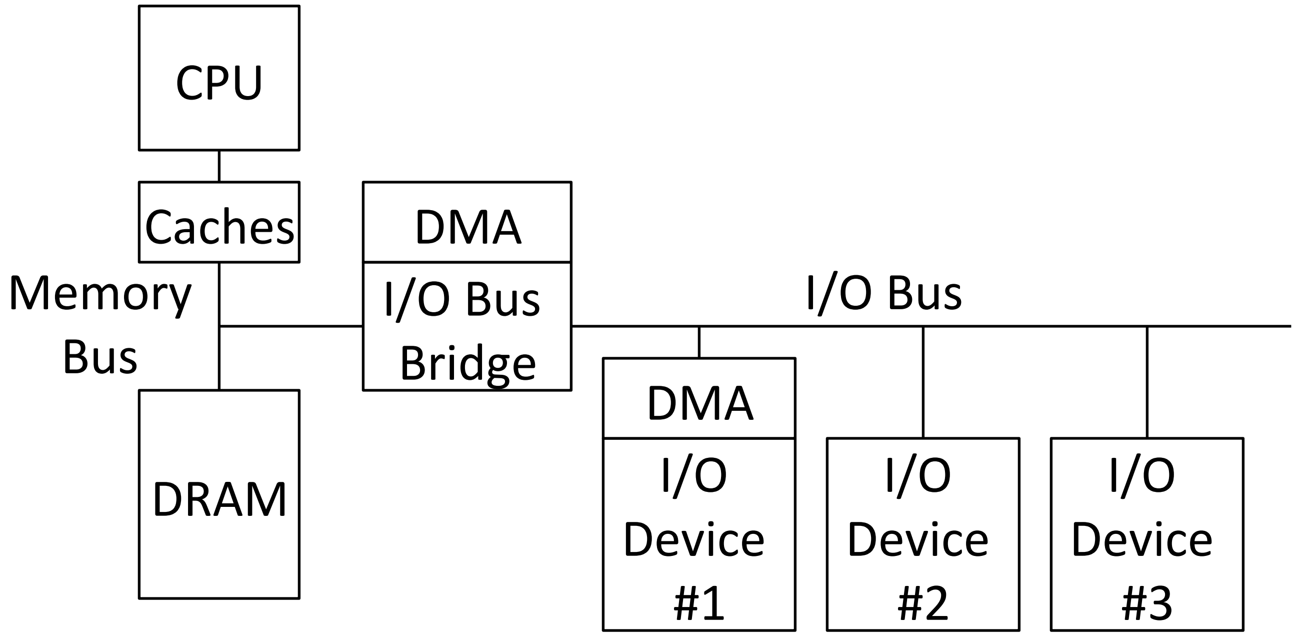
- Programmed I/O uses CPU to control I/O device using load and store instructions, with address specifying device register to access
  - Load and store can have side effect on device
- Usually, only privileged code can access I/O devices directly, to provide secure multiprogramming
  - System calls sometimes provided for application to open and reserve a device for exclusive access
- Processors provide "uncached" loads and stores to prevent caching of device registers
  - Usually indicated by bits in page table entries or by reserving portions of physical address space

Simple I/O Bus Structure



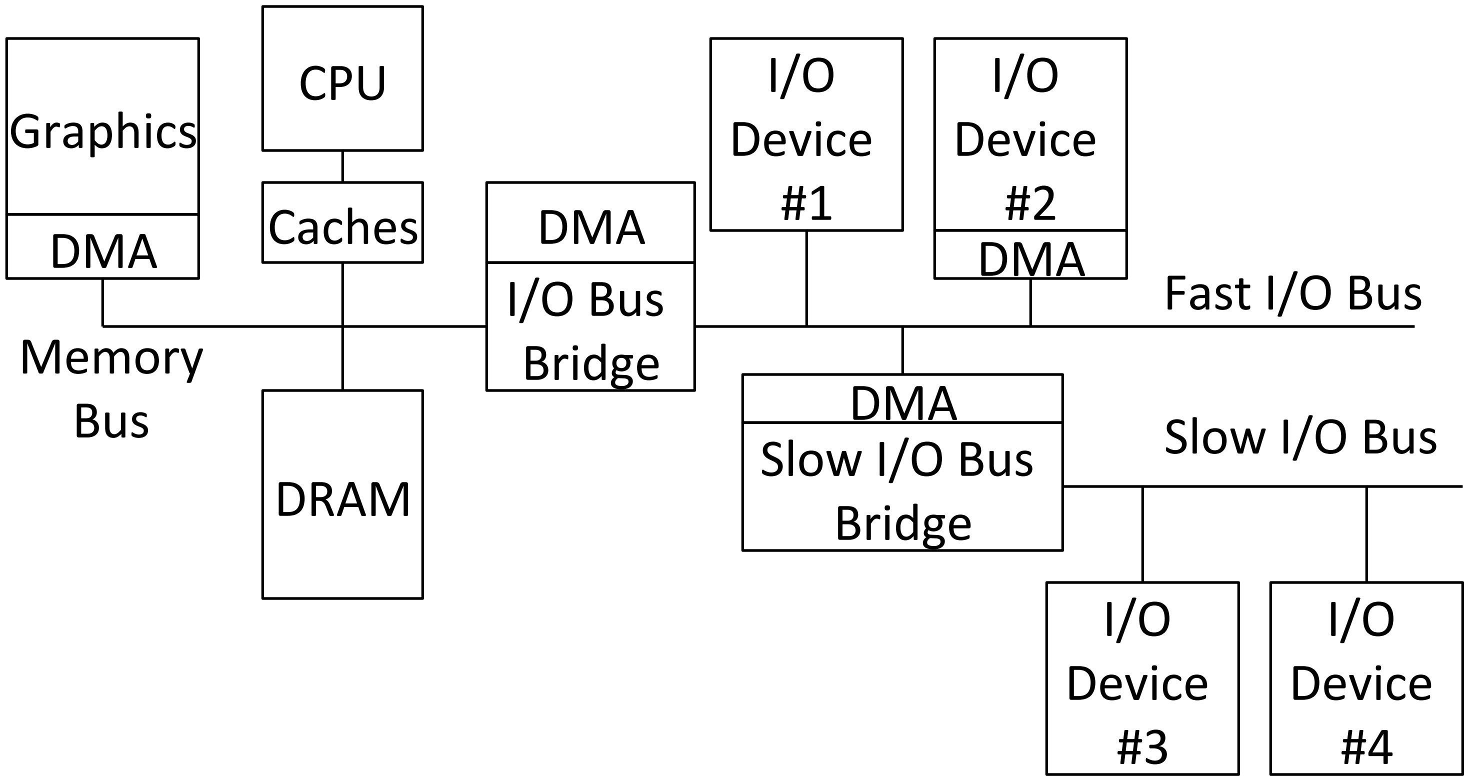
- Some range of physical memory addresses map to I/O bus devices
- I/O bus bridge reduces loading on critical CPU-DRAM bus
- Devices can be “slaves”, only responding to I/O bus requests
- Devices can be “masters”, initiating I/O bus transfers

DMA (Direct Memory Access)



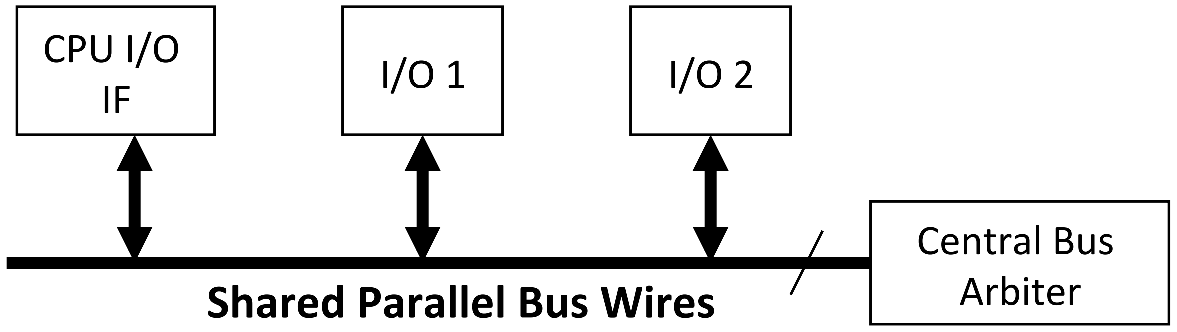
- DMA engines offload CPU by autonomously transferring data between I/O device and main memory. Interrupt/poll for done
  - DMA programmed through memory-mapped registers
  - Some systems use dedicated processors inside DMA engines
- Often, many separate DMA engines in modern systems
  - Centralized in I/O bridge (usually supporting multiple concurrent channels to different devices), works on slave-only I/O busses
  - Directly attached to each peripheral (if I/O bus supports mastering)

More Complex Bus Structures



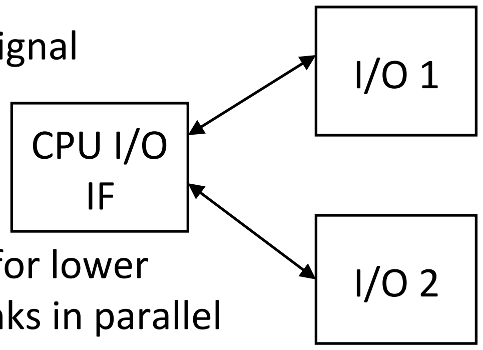
- Match speed of I/O connection to device demands
  - Special direct connection for graphics
  - Fast I/O bus for disk drives, ethernet
  - Slow I/O bus for keyboard, mouse, touchscreen
    - Reduces load on fast I/O bus + less bus logic needed on device

Move from Parallel to Serial I/O Off-chip



- Parallel bus clock rate limited by clock skew across long bus (~100MHz)
- High power to drive large number of loaded bus lines
- Central bus arbiter adds latency to each transaction, sharing limits throughput
- Expensive parallel connectors and backplanes/cables (all devices pay costs)
- Examples: VMEbus, Sbus, ISA bus, PCI, SCSI, IDE

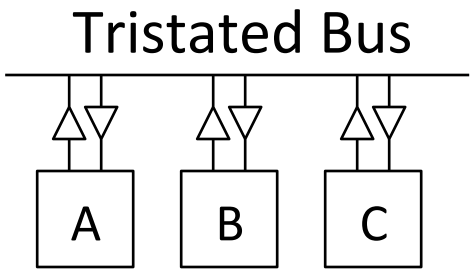
Dedicated Point-to-point Serial Links

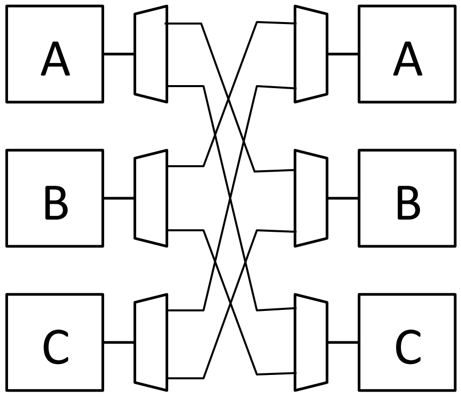


- Point-to-point links run at multi-gigabit speed using advanced clock/signal encoding (requires lots of circuitry at each end)
- Lower power since only one well-behaved load
- Multiple simultaneous transfers
- Cheap cables and connectors (trade greater endpoint transistor cost for lower physical wiring cost), customize bandwidth per device using multiple links in parallel
- Examples: Ethernet, Infiniband, PCI Express, SATA, USB, Firewire, etc.

Move from Bus to Crossbar On-Chip

- Busses evolved in era where wires were expensive and had to be shared
- Bus tristate drivers problematic in standard cell flows, so replace with combinational muxes
- Crossbar exploits density of on-chip wiring, allows multiple simultaneous transactions





Crossbar

I/O and Memory Mapping

- I/O busses can be coherent or not
  - Non-coherent simpler, but might require flushing caches or only non-cacheable accesses (much slower on modern processors)
  - Some I/O systems can cache coherently also (SGI Origin)
- I/O can use virtual addresses
  - Simplifies DMA into user address space, otherwise contiguous user segment needs scatter/gather by DMA engine
  - Provides protection from bad device drivers
  - Adds complexity to I/O device

Interrupts versus Polling

Two ways to detect I/O device status:

- Interrupts
  - +No CPU overhead until event
  - –Large context-switch overhead on each event (trap flushes pipeline, disturbs current working set in cache/TLB)
  - –Can happen at awkward time
- Polling
  - – CPU overhead on every poll
  - – Difficult to insert in all code
  - +Can control when handler occurs, reduce working set hit

- Hybrid approach:
  - – Interrupt on first event, keep polling in kernel until sure no more events, then back to interrupts

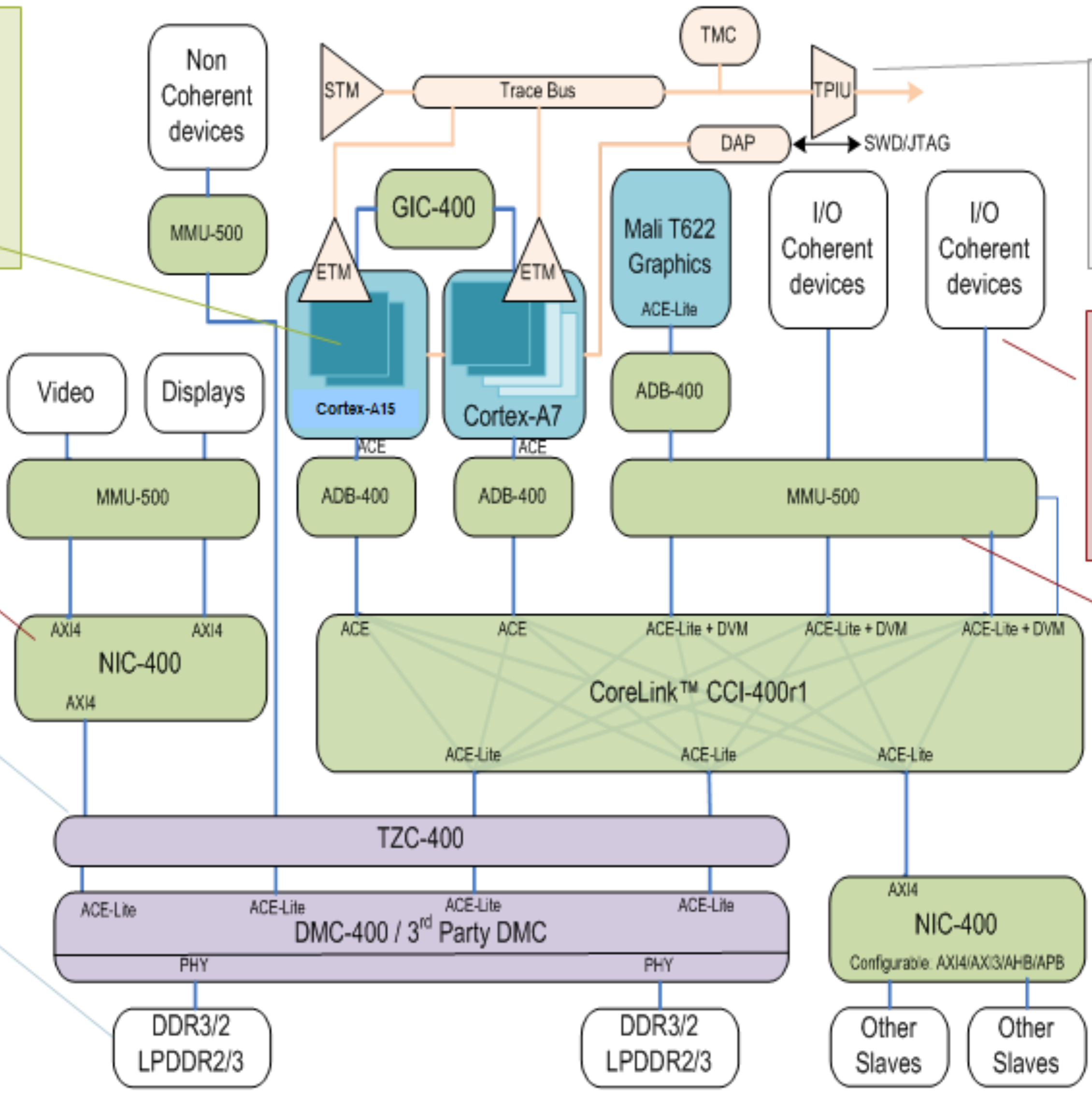
Example ARM SoC Structure

Range of Topologies:
big.LITTLE
- Dual A15 + Quad A7
- Dual A15 + Dual A7

(Optional) Real-time traffic on a separate NIC AXI4 bus into DMC

TZC provides secure and protected regions of Memory

DRAM
12-17GB/s
e.g.
2 * x32 LPDDR3



CoreSight debug & trace IP for flexible profiling and optimization of complex systems

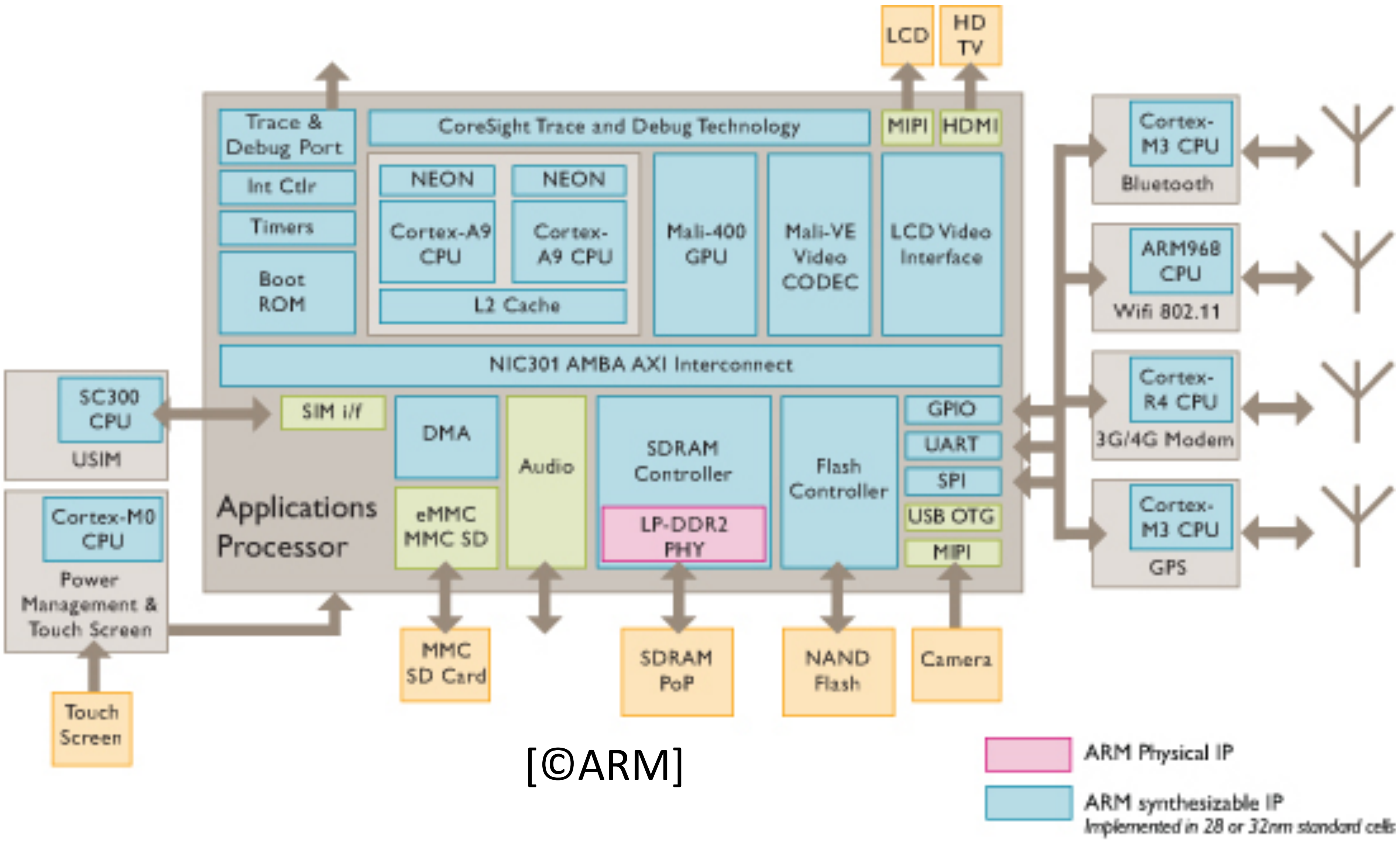
I/O coherent masters that share data with the CPU
e.g. 2D accelerator or security / encryption module

MMU enables a common memory view for all SoC components

CCI-400 provides hardware coherency to simplify software

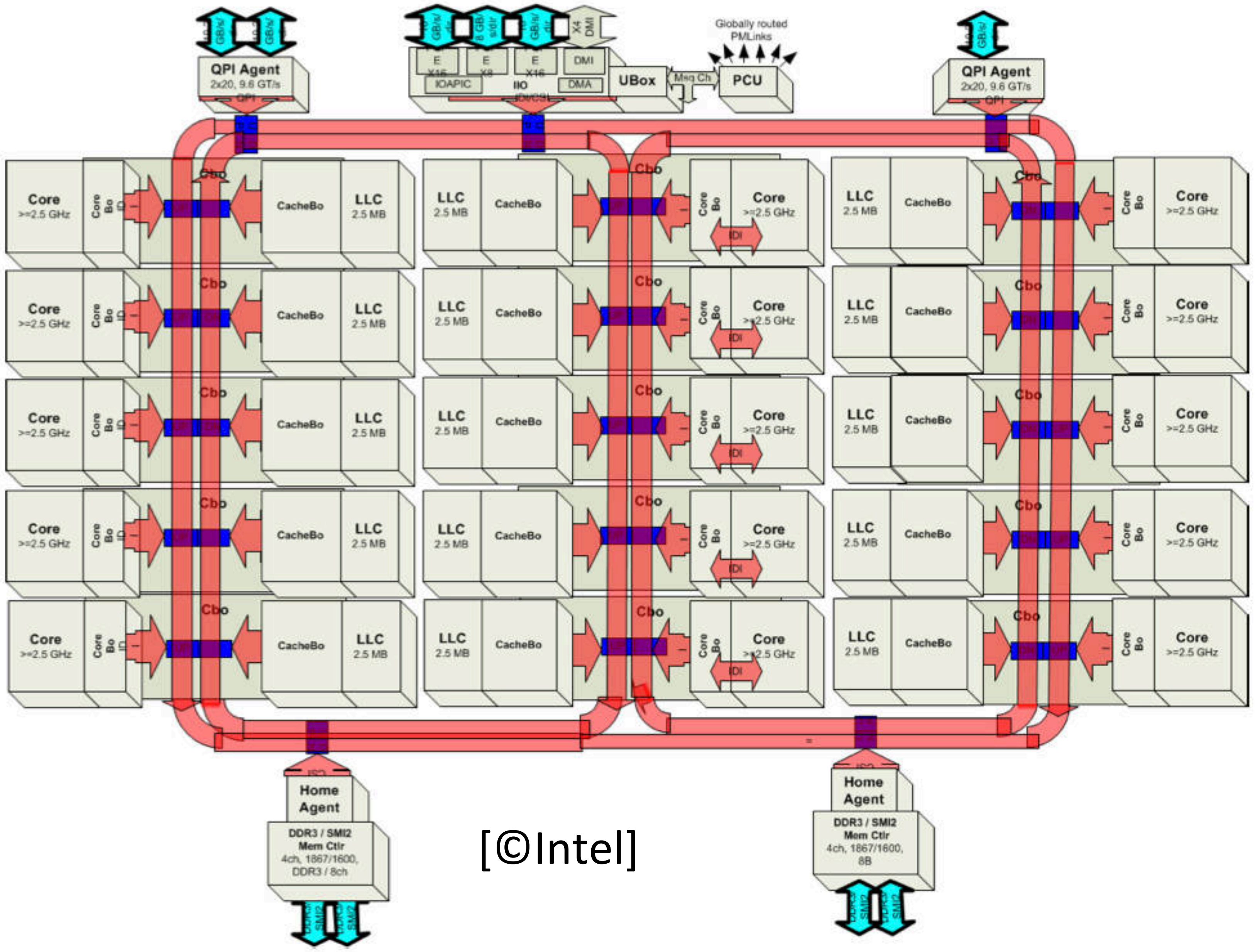
[©ARM]

ARM Sample Smartphone Diagram



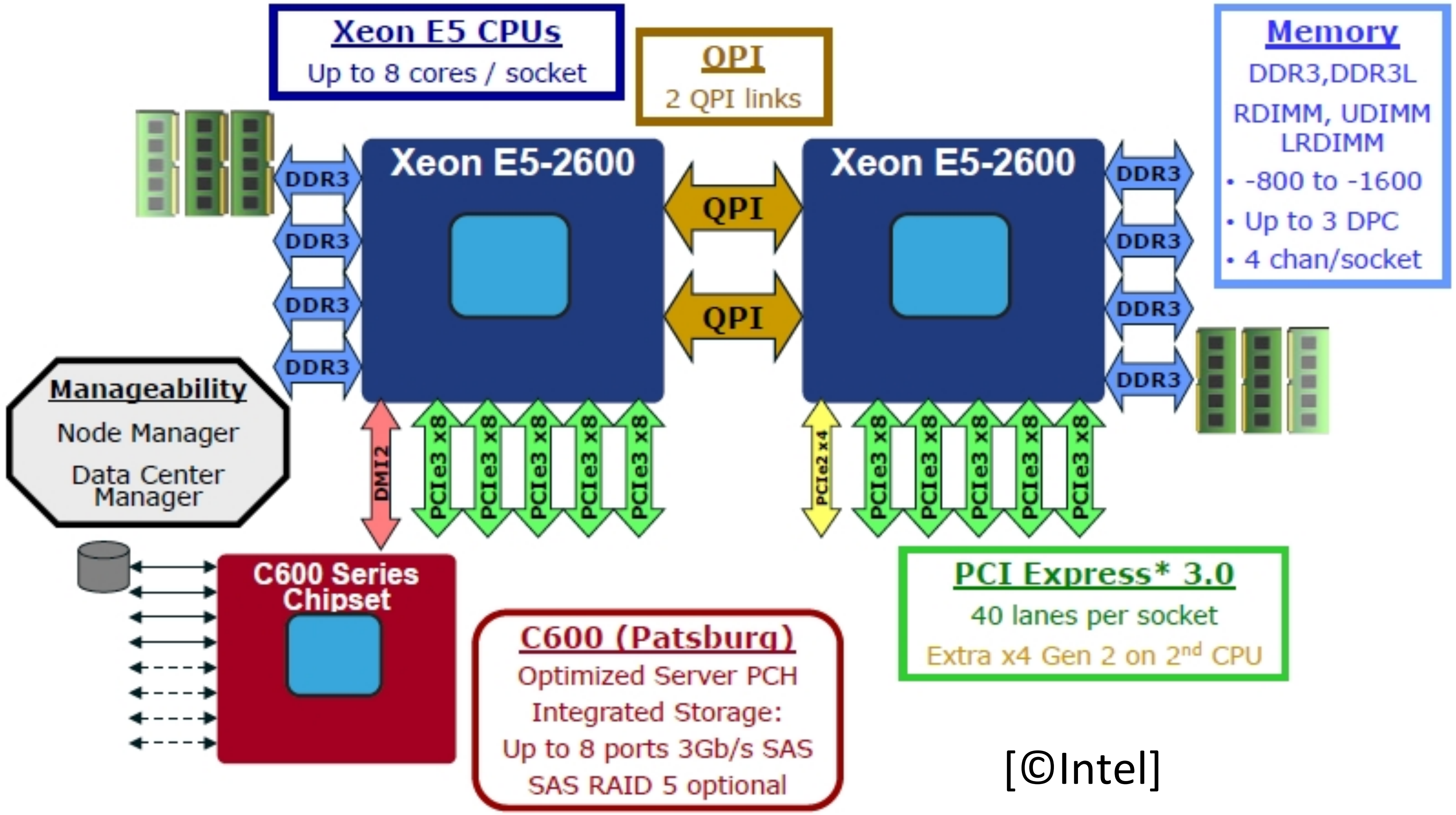
[©ARM]

Intel Ivy Bridge Server Chip I/O



[©Intel]

Intel Romley Server Platform



[©Intel]

Acknowledgements

- This course is partly inspired by previous MIT 6.823 and Berkeley CS252 computer architecture courses created by my collaborators and colleagues:
  - Arvind (MIT)
  - Joel Emer (Intel/MIT)
  - James Hoe (CMU)
  - John Kubiatowicz (UCB)
  - David Patterson (UCB)