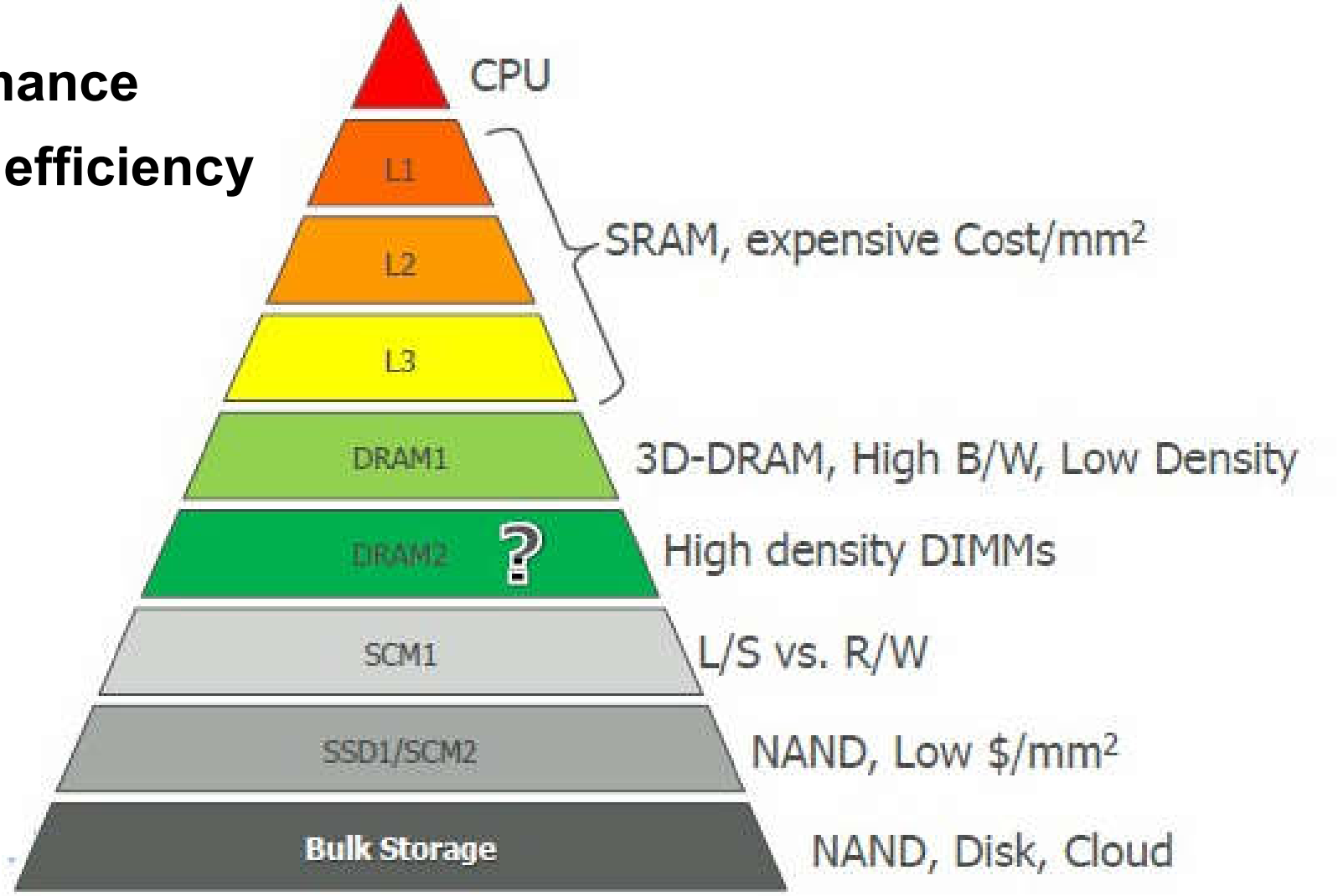
Understanding Memory Systems

Memory hierarchy, improves

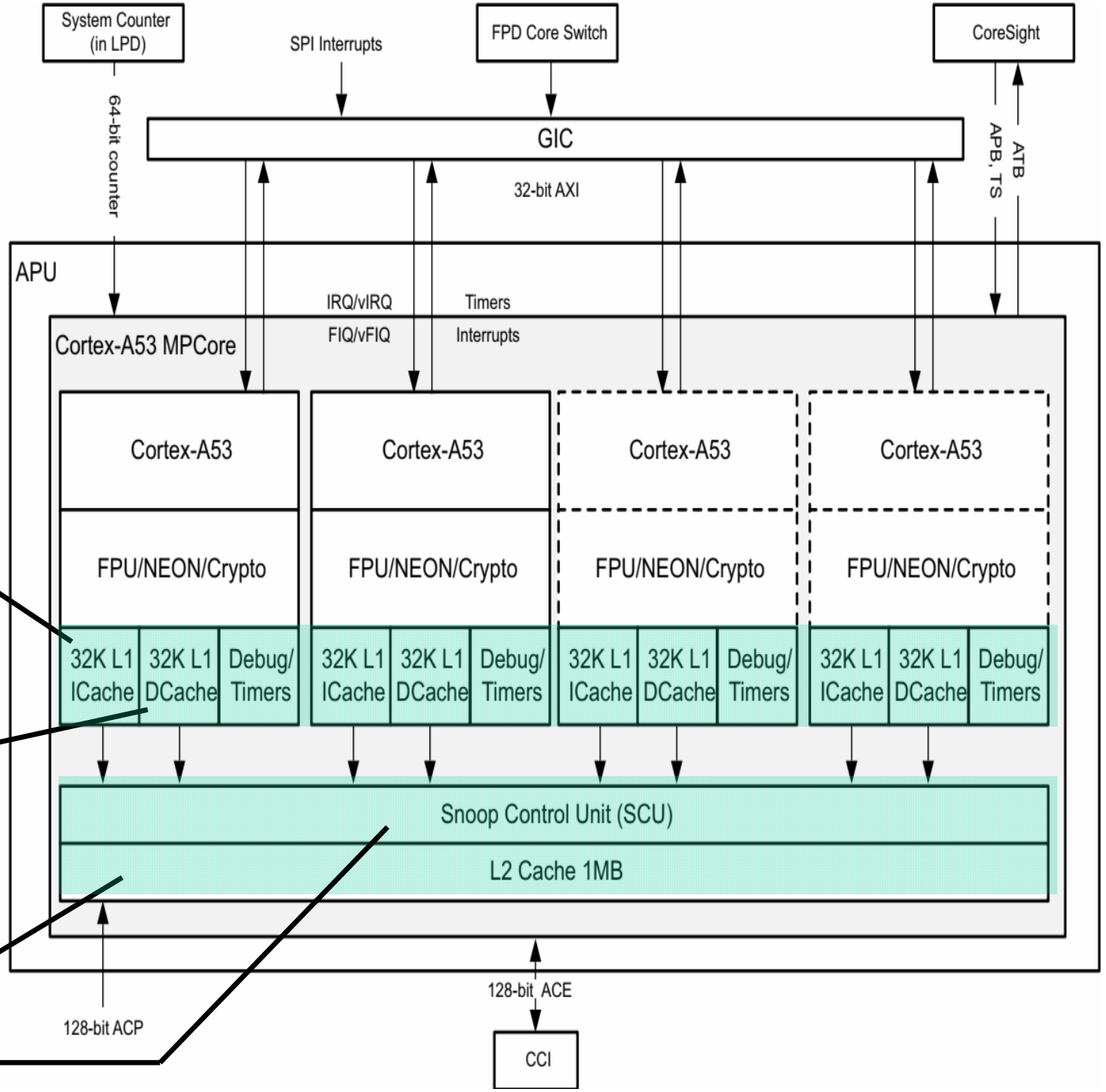
- **cost**
- **performance**
- **energy efficiency**



Caches (e.g. ARM)

Memory hierarchy of a Xilinx Zynq-US+ SoC:

- 4 cores, each with
  - 32K L1 ICache (2-way set associative)
  - 32K L1 DCache (4-way set associative)
- 1MB shared L2 Cache (16-way set associative)
- MESI Snooping



How long do memory accesses take?

This depends on various factors:

- **Where is the data located**? (hit latency: 1 cycles L1, 2-4 cycles L2)
- **In what state is the cache**? (200+ clock cycles for a cache miss)
- **In what state is the TLB[1)]?** (TLB: Translation Lookaside Buffer)

  Do we need a page table walk? (typical 1K+ CPU cycles)
- **What are the other CPU cores (or other accelerators) doing**?
- **Which CPU features are currently used?**

  (out-of-order Execution (OoO), Branch Prediction, Prefetch Unit)
- **Is the DDR RAM going through a refresh cycle**? (~300ns for DDR4)

- **Message: very difficult to use for real time systems!**

What impacts the cache state?

Impact on the cache from:

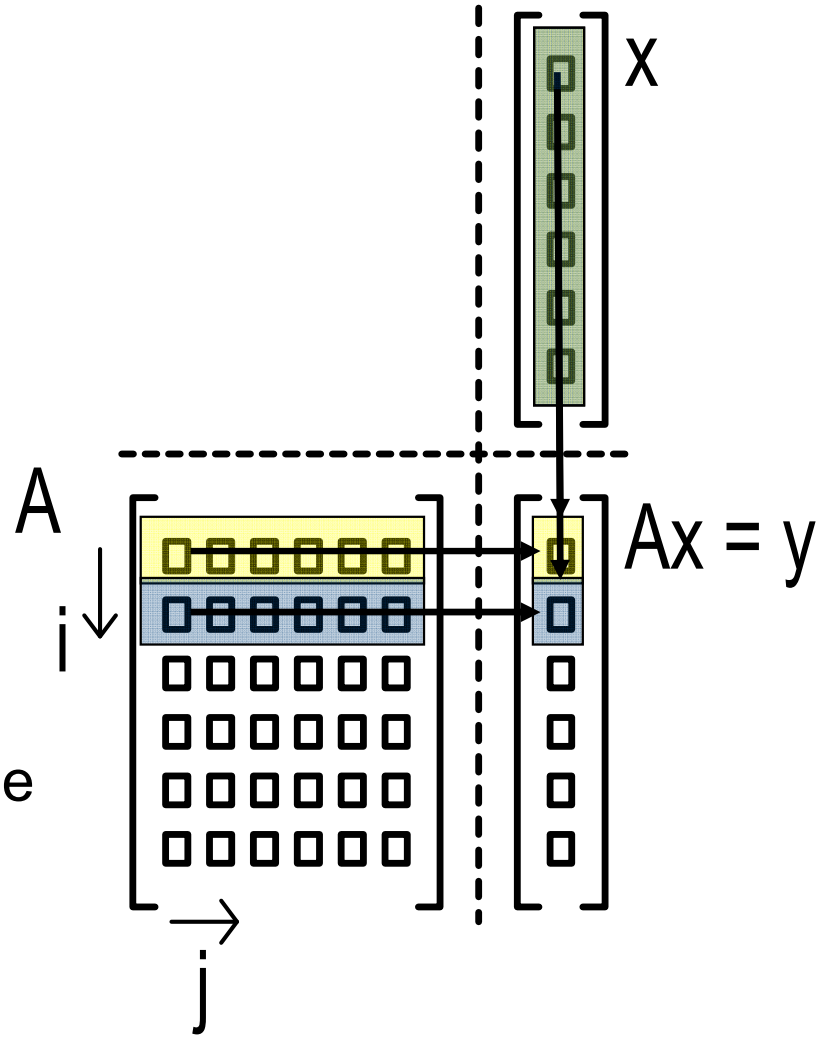
- **Intratask interference:** When the currently used data (working set) does not fit into cache (if your currently problem cannot use the cache)
- **Intracore/Intertask interference:** If the data inside a cache was evicted (e.g. from a context switch or for serving an IRQ).
  In this case, data has to be loaded again on resume.
- **Intercore interference:** describes the cross-interaction from multiple cores that share a cache (e.g. L2 on larger ARM SoC is shared)
  Warning: can result in ping-pong cache pollution effect

  → core underbooking (using less than the available cores) can improve performance

How long do memory accesses take?

Example: Matrix-Vektor Multiplication:

```
// for each row of matrix A
for ( i = 0; i < N; i++ )
    // compute dot product
    for ( j = 0; j < N; j++ )
        y[i] +=  A[ i ][ j ] * x [ j ];
```



- **Intratask interference:**
  x does not fit entirely into cache
- **Intracore/Intertask interference:**
  Another process evicts x from the cache
- **Intercore interference:**
  Anoter core evicts x from the cache

→ Think about how this example would look like for matrix multiplication!

Can we predict the worst case?

- A good worst case estimate should not be too pessimistic!
  (or we may waste compute performance)
- The estimate should not be too optimistic!
  (or we may overload the system or miss deadlines (real time!))

Problem: Memory access latency can range from 1 - 4000+ clock cycles

https://www.jblopen.com/arm-cortex-a-interrupt-latency/

Worst case estimates are possible for relative simple models
(single core, no virtual memory, …) but tough for general systems

→ (still) quite some research performed in this space

Are we doomed?

Dedicated memory mapping (OS), cache-aware code, hardware

Hardware Support

The ARM SoC provides a Real-Time Processing Unit (RPU)

RPUs comprise Tighly-coupled Memories (TCU)
also called scratchpads or managed caches