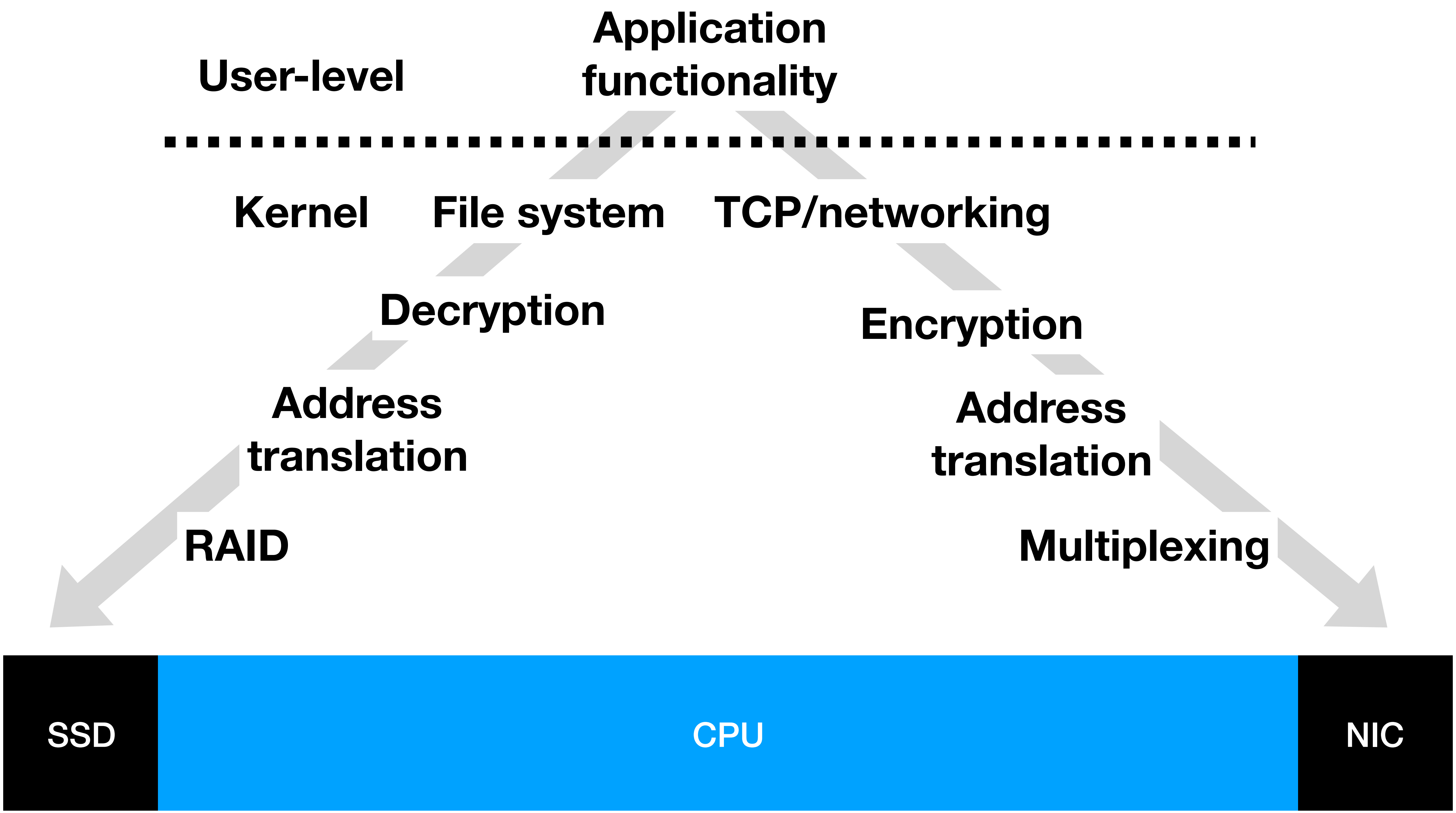
Next Generation Datacenter Operating Systems

A general-purpose
OS/application interface
for programmable hardware

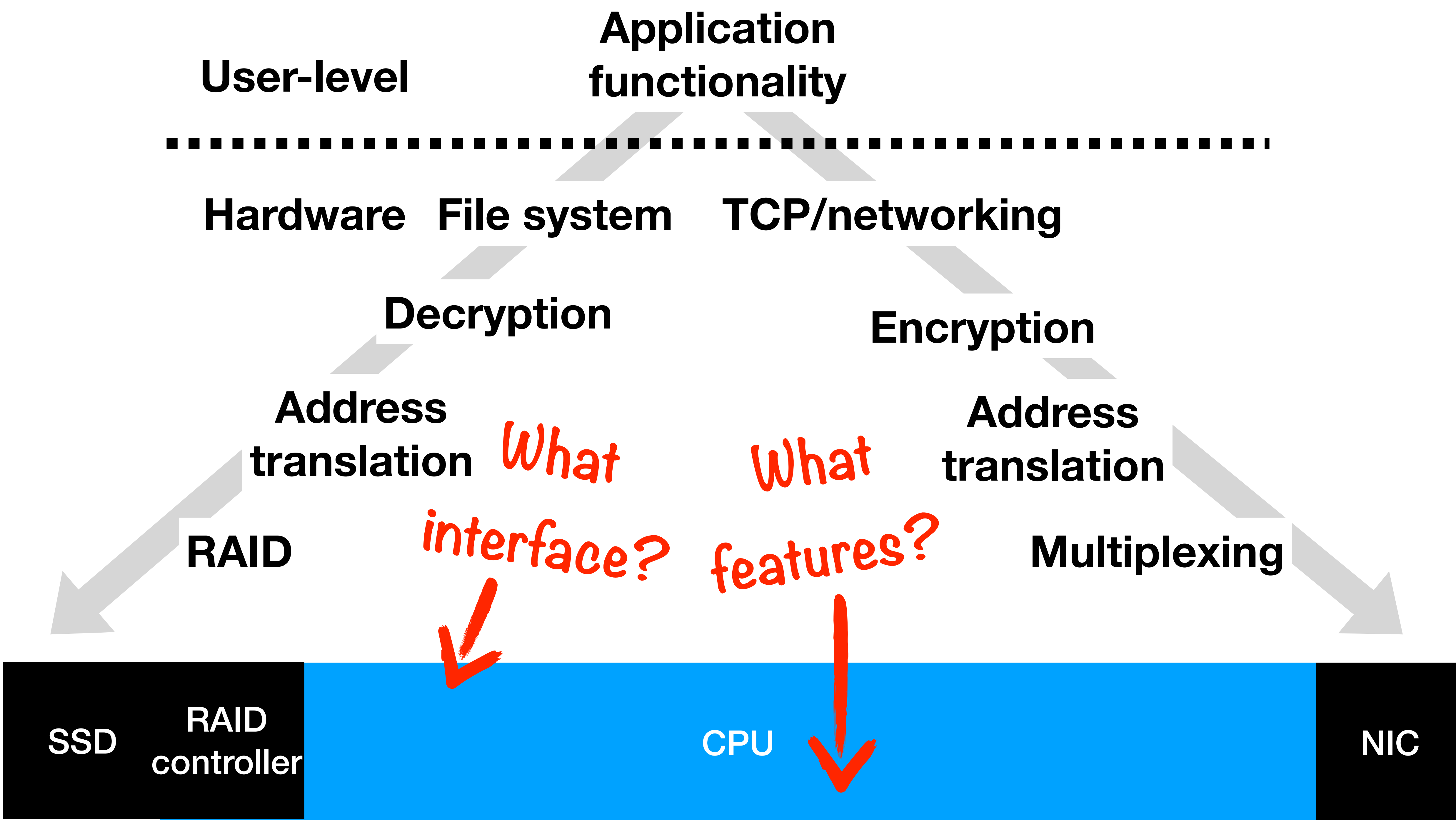
Devices on datacenter servers are getting faster while CPUs are not.

Insert Moore's Law here

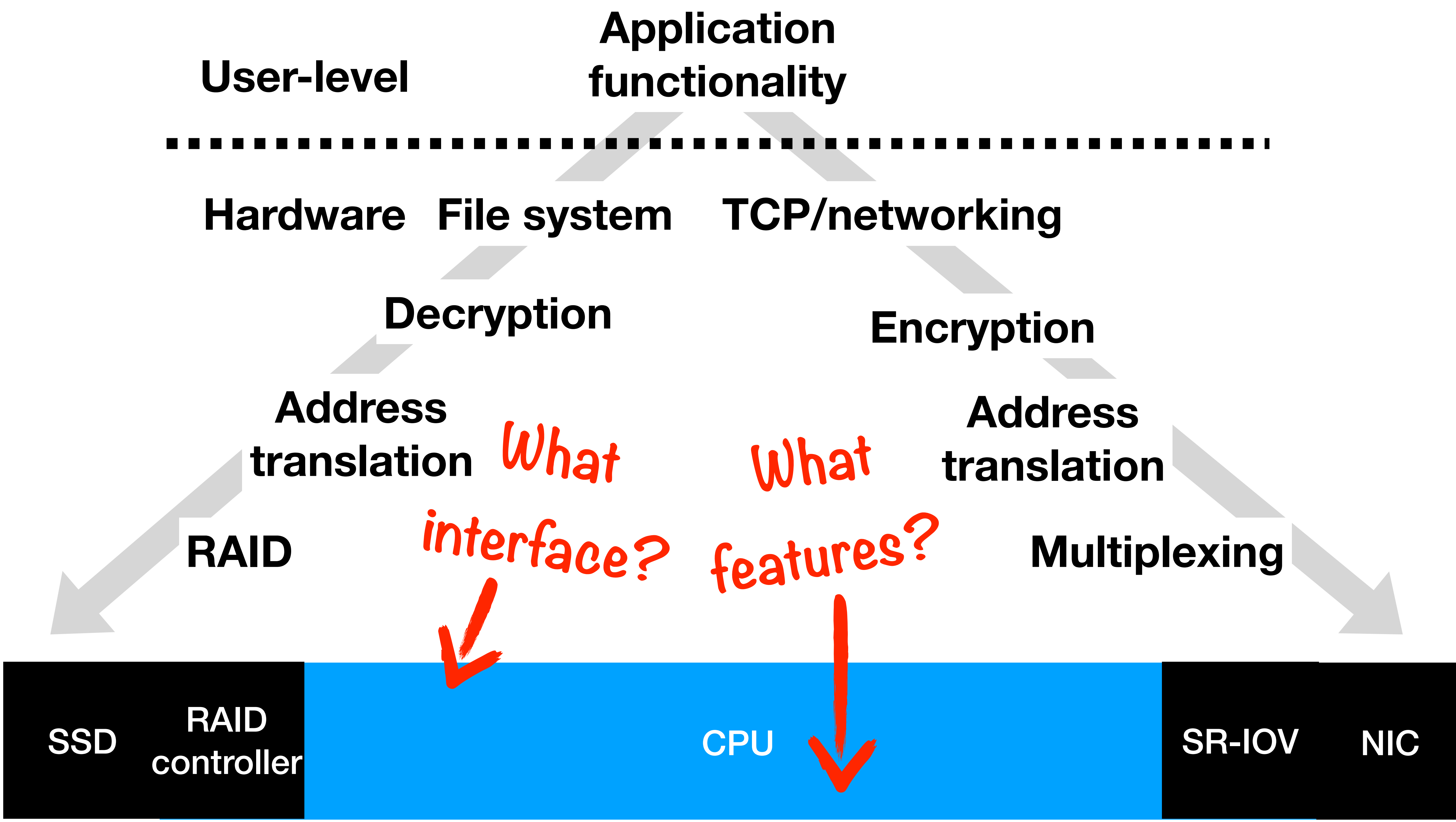
CPUs cannot keep up with demanding datacenter applications



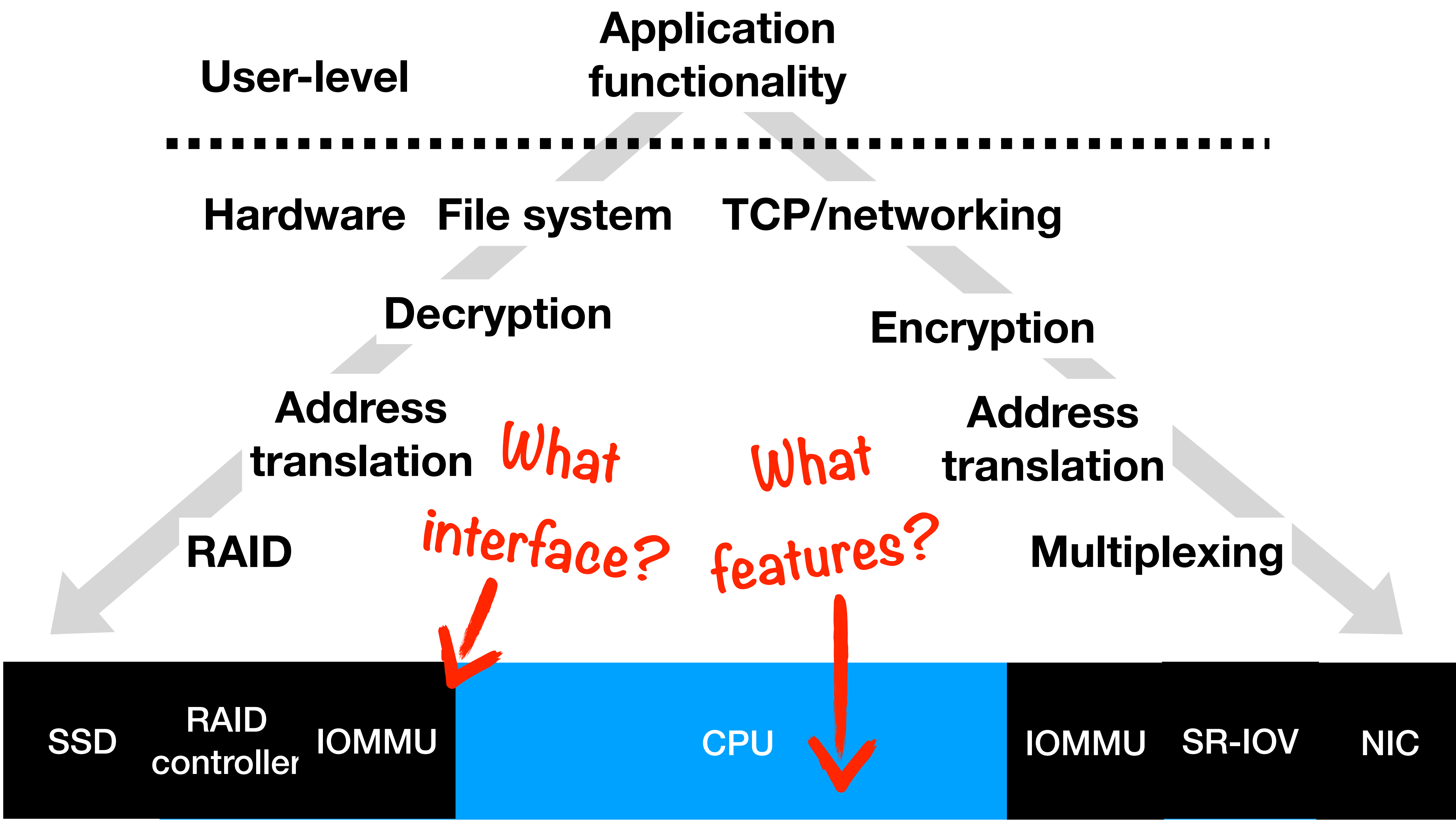
Hardware acceleration offloads functionality from CPU to devices



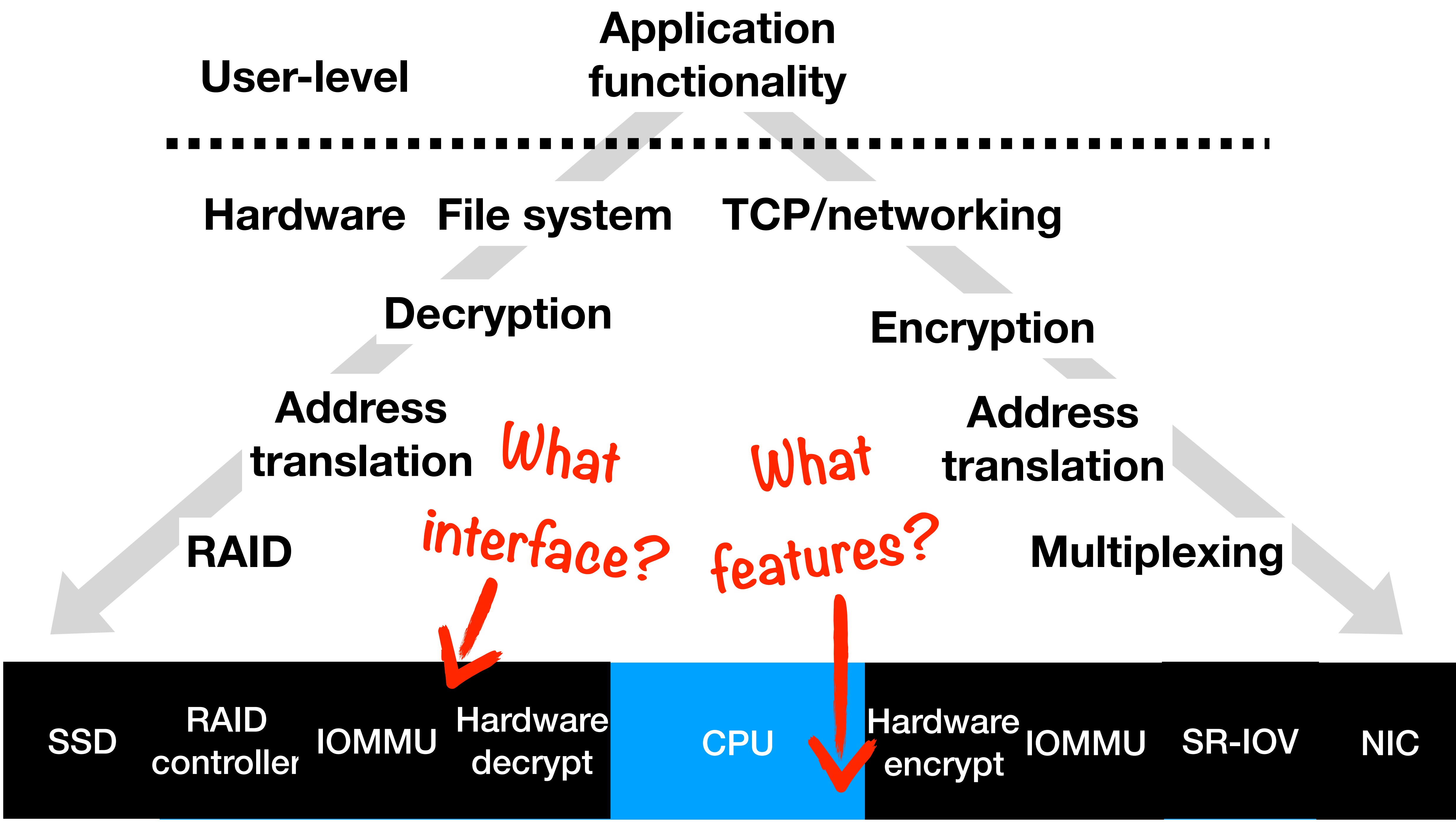
Hardware acceleration offloads functionality from CPU to devices



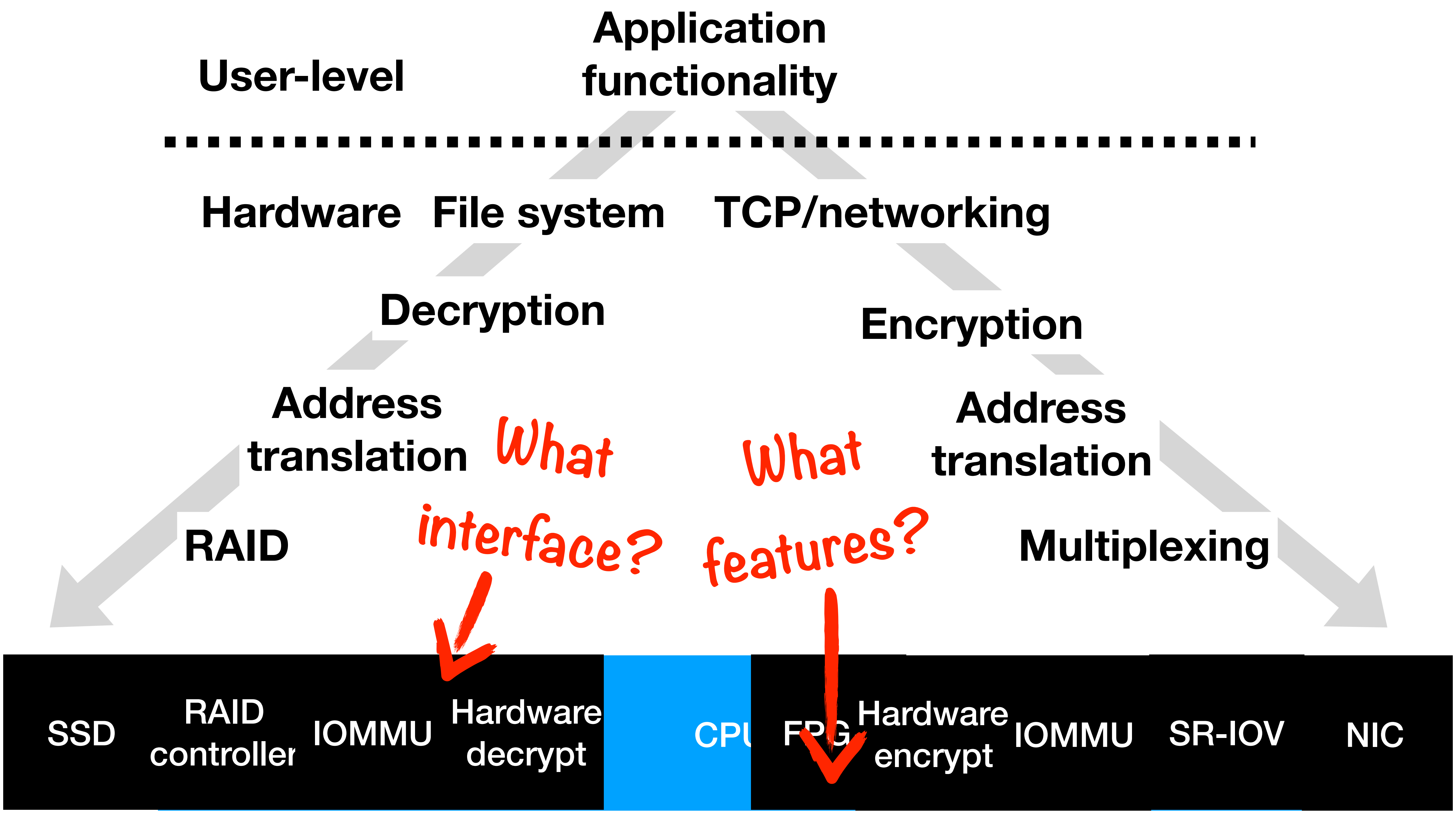
Hardware acceleration offloads functionality from CPU to devices



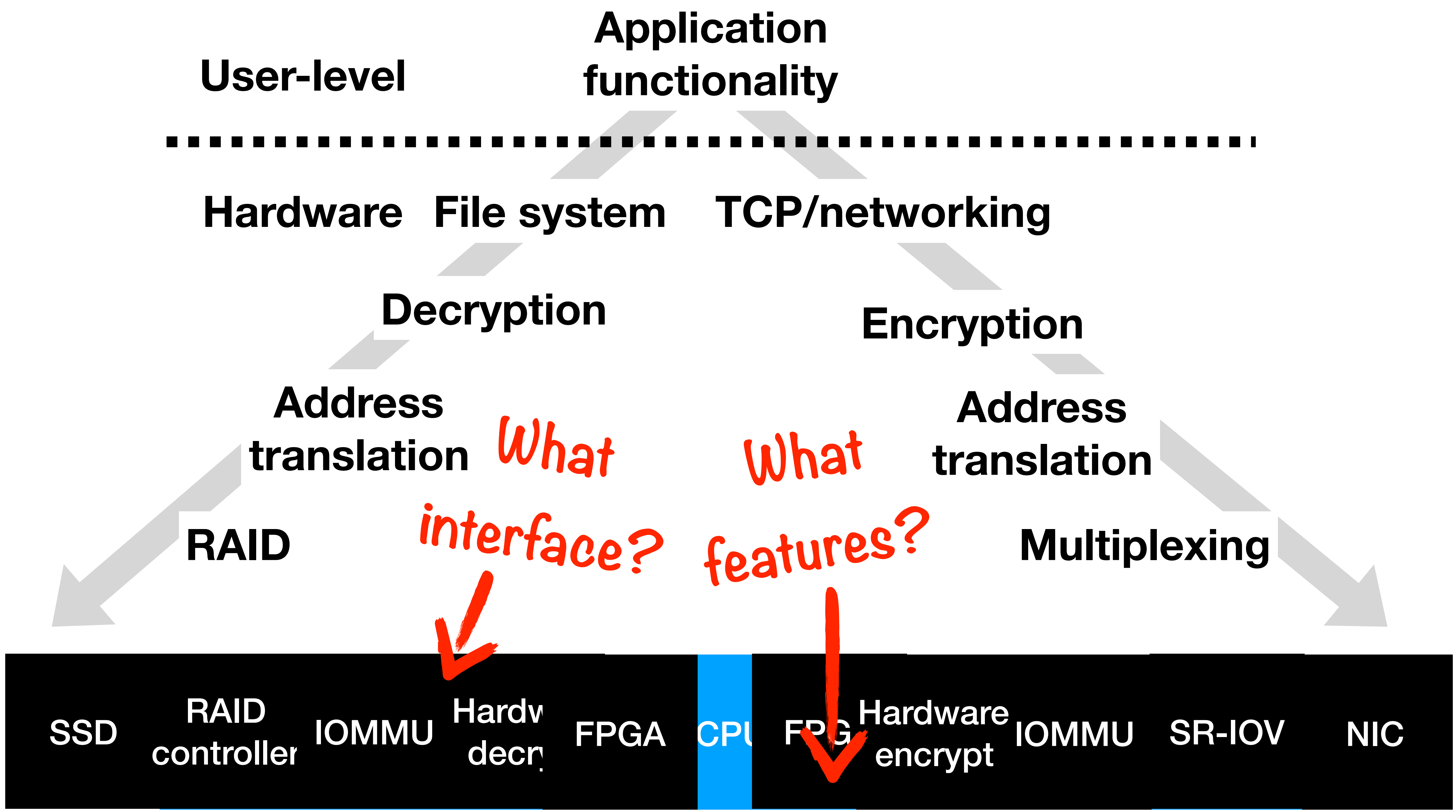
Hardware acceleration offloads functionality from CPU to devices



Hardware acceleration offloads functionality from CPU to devices



Hardware acceleration offloads functionality from CPU to devices



Acceleration hardware is changing at a rapid pace.

- The hardware functionality keeps changing (e.g., new NICs, new features).
- The hardware/software interface keeps changing (e.g., let’s put everything into the NIC! Let’s not!).
- New systems interfaces keep appearing.

How to build a demanding datacenter app in this world?

- Modify applications for every new technology/system.

How to build a demanding datacenter app in this world?

- ~~Modify applications for every new technology/system.~~

How to build a demanding datacenter app in this world?

- ~~Modify applications for every new technology/system.~~

- Require new hardware to support legacy interfaces (e.g., POSIX)

How to build a demanding datacenter app in this world?

- ~~Modify applications for every new technology/system.~~
- ~~Require new hardware to support legacy interfaces (e.g., POSIX)~~

How to build a demanding datacenter app in this world?

- ~~Modify applications for every new technology/system.~~
- ~~Require new hardware to support legacy interfaces (e.g., POSIX)~~
- Design new general systems/application interface for programmable hardware

Requirements for new systems interface

- Separate app from hardware to allow hardware to evolve independently
- Efficient for transferring data (not a serializing interface!)
- Efficient to implement in hardware or software

Everything is a ~~file~~ zero-copy queue

- Replaces file descriptors and pipes with queue descriptors and queues
- Is zero-copy for efficient app processing without cache pollution
- Offers insight into granularity for filtering, merging, sorting

Interface

```
qid = socket(domain, type, protocol)

qid = open(file)

qid = accept(qid)

insert(qid, *sga)

*sga = dequeue(qid)

qid = filter(qid, *filter_func)

qid = merge(qid, qid)

qid = sort(qid, *sort_func)

(sga = scatter gather array = list of bufs)
```

Use cases

- File/storage server
- Memcached
- Meta-data server
- Replicated service
- Others?

Available Hardware (MSR only)

	programmable?	programming mode	features
RDMA	static		direct memory access
RDMA	partially programmable	firmware changes	
RDMA	fully programmable	FPGA	
SSD	fully programmable	start-up	SR-IOV (end 2018)
SSD	remote access	FPGA	

Fast Context Switching for Fine-grained Process Scheduling

High-performance datacenter apps make poor use of CPUs

- Existing solution is to pin threads, which under-utilizes the CPU for bursty workloads.
- No multi-tenancy to even out bursts.
- CPU to go into a lower power mode in between bursts, increasing tail latency

New CPU hardware has lowered the cost of context switches

- Faster switches between rings
- Tagged TLBs
- Partitioned caches

Making it feasible to schedule the CPU for shorter periods.

Datacenter applications have natural interrupt points

- High-performance datacenter apps are typically request processors
- Less data shared between requests (measure this!)
- OS has no insight into these points

Cooperative scheduling will perform better than interrupts or polling

Request-based process scheduling

- Can potentially be done without modifying app (e.g., changing libevent)
- Allow even high performance apps to share a CPU
- Lower (tail) latency for everyone
- Better performance isolation

Exactly once RPC hardware