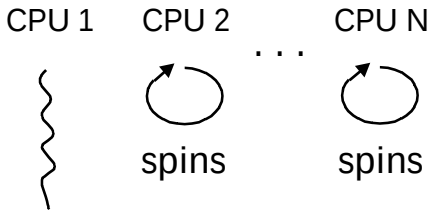
UDI Performance

Performance & Scalability of the UDI Driver Model

Synchronization Model Comparisons

In the common case where a driver has significant per-instance state that needs to be protected, legacy drivers will typically hold a spinlock across the driver path:

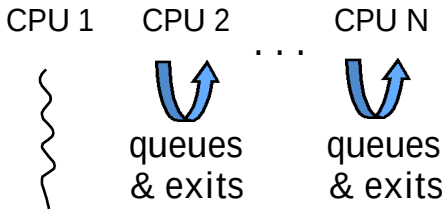
Legacy Model



Spinlock is held across the driver path.

- If multiple CPUs enter the driver near the same time, one will obtain the instance lock, the others will spin.

UDI Model



allowing the CPUs to do other useful work.

Spinlock only held for a short time.

In UDI, instead of holding the spinlock across the driver path, it's held a short time to set "busy" or queue.

Region Synchronization Scalability & Correctness

- The UDI model
  - Provides deterministic lock held times --> scalability & correctness
    - Improves scalability via decreased spinlock contention
    - Eliminates corner case paths that produce long spinlock held & spin times.
  - Eliminates cascading effects on lock held times caused by holding spinlocks across external service calls (which in turn grab their own spinlocks, further holding off the caller's spinlock).
  - Provides simplified locking (in the UDI environment) that's analyzable for correctness.

Region Synchronization Parallelism (1/2)

- UDI has instance-independence

  --> Parallelism among instances of a driver is a given.

- Parallelism within a driver instance:
  - To protect per-instance state, legacy drivers typically grab a spinlock at entry & release it at exit.
  - The UDI model serializes across the same driver path, so provides similar parallelism capability, but also provides:
    - increased parallelism to the degree that spinlock contention is reduced
    - driver paths in UDI are not prolonged by “long” service calls; such calls have callbacks, allowing for additional parallelism between the driver and the environment.

Region Synchronization Parallelism (2/2)

- Parallelism in a driver is gated by
  - the programming model of the driver/device, and the per-instance state on which it operates,
  - and can be gated by platform-specific processor affinity algorithms or requirements (e.g., putting requests on the same CPU as the corresponding completion interrupt can result in limiting the driver to per-instance parallelism when there's only one interrupt source).
- Multiple regions for additional parallelism:
  - Where the driver/device programming model allows for independent paths (e.g., independent transmit and receive paths), additional regions per instance can be created in the UDI model for additional parallelism.