High performance protocols in practice

Simon Chatterjee

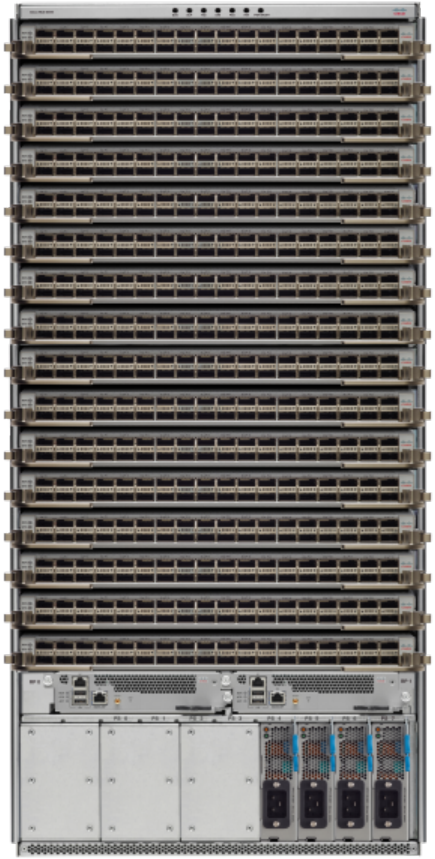
Lead Architect, Service Provider Software

What’s this about

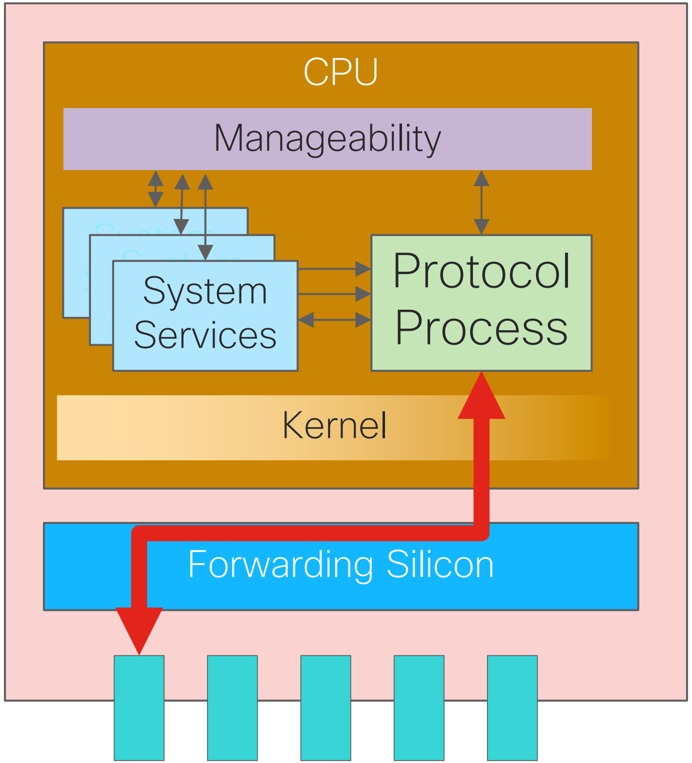
- I’ve been building big routers for a while
- Back when I started, industrial protocol implementations looked pretty similar to the open source equivalents
- Now more divergence on average
- Not claiming novelty, but interesting to me

Context

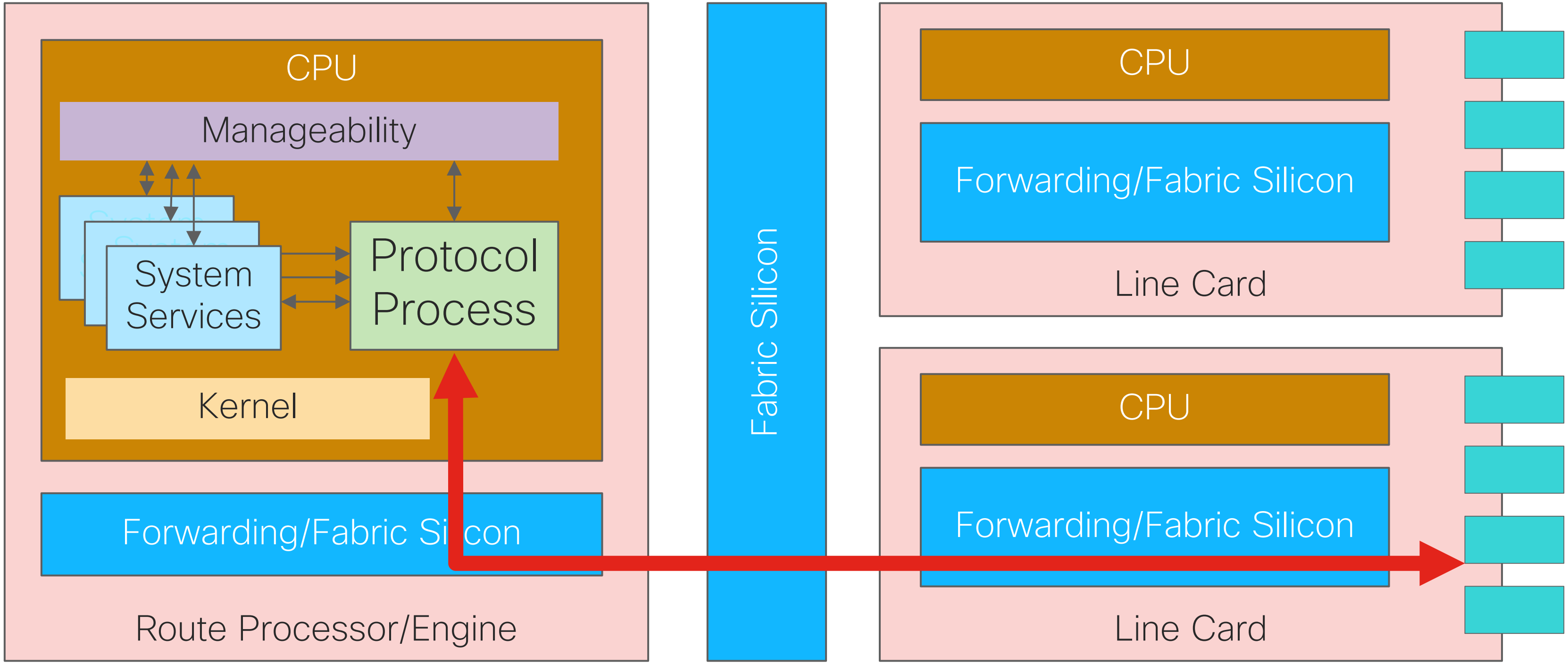
- Try to focus engineering effort on as few “performance” protocols as possible
- “Simple” keepalives (BFD for L3, CFM for L2)
  - As fast as 3.3ms x 3
  - More commonly 10ms x 3
  - Can have many protocol peers
- Precision time protocol (IEEE 1588)
  - 128 syncs per second actually means 640 pps
  - 5000 cells => 3.2 million pps steady state
- Others still creep in (eg VRRP)



Classical protocol implementations



Classical protocol implementations



Pros/cons of the classical model

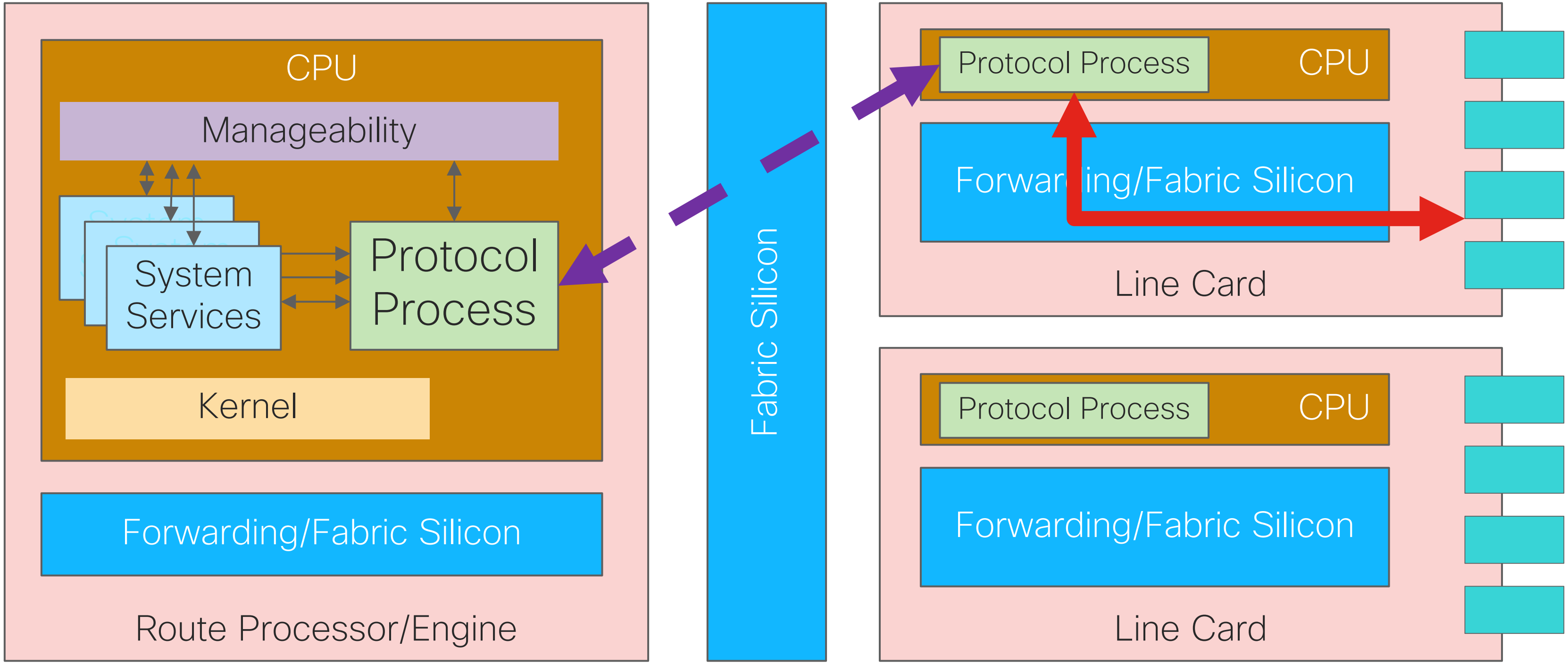
Good

- Simple
- Familiar sockets

Bad

- Scale limitations
- Hard to do make low latencies reliable
- Hardware failovers kill fast keepalives
  - 4-6 seconds (ISIS, MSTP) just about possible
- Makes the single process very complex
  - Big manageability queries, configuration changes, in-place software upgrades, …

Simple distribution



Pros/cons of simple distribution

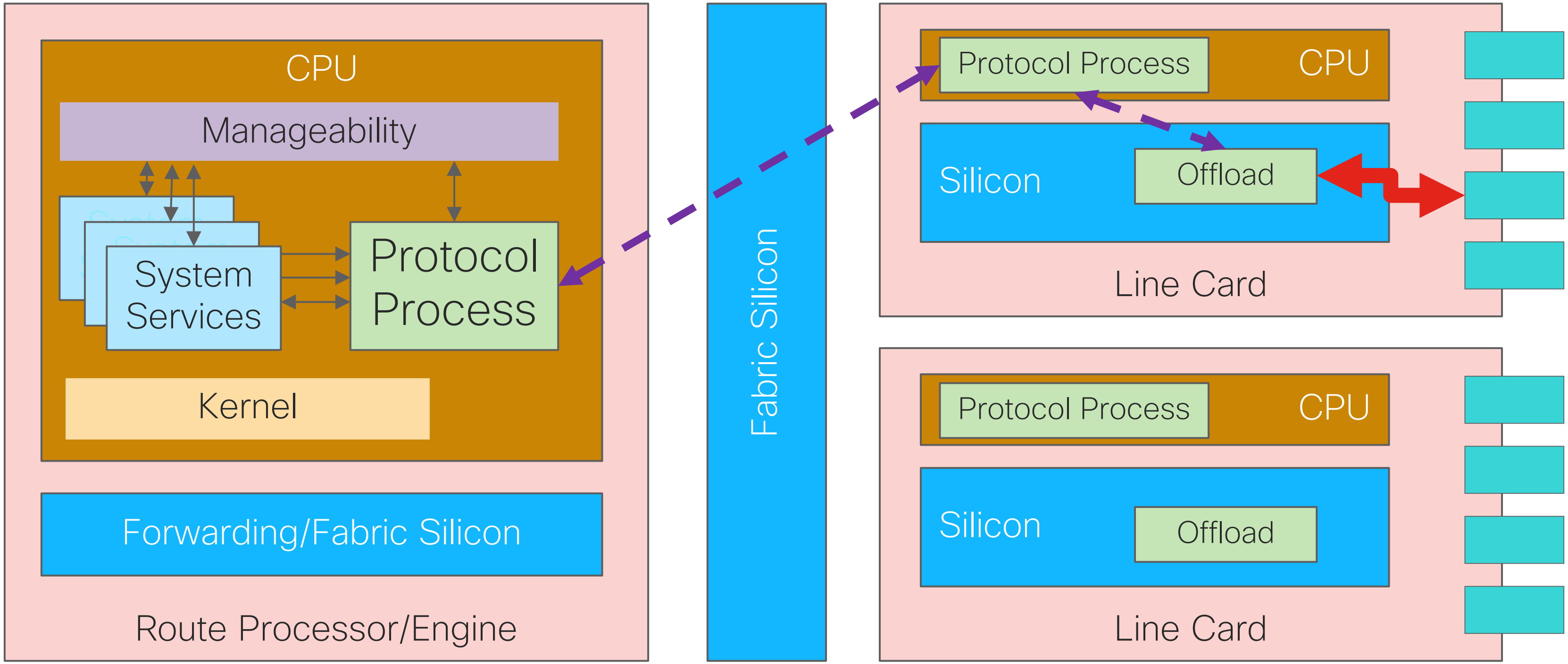
Good

- Linear scaling when it works
  - Despite weedy LC CPUs
- RP failovers non-impactful
- Potentially lots of code re-use
  - If system infra already distributed

Bad

- Not all protocol sessions are tied to a specific LC
  - Eg peer defined just by IP address (or Link Aggregation Group, or L2 bridge, or tunnel, or …)
  - Sometimes cheesy mitigations work
- Multiple high-pps protocol processes are hard to make reliable
- Software upgrades still impactful

Hardware offload



Pros/cons of hardware offload

Good

- Easy to do low latency (eg 3.3ms) keepalives reliably
- Doesn’t degrade with multiple protocol processes
- In principle lets you hitlessly upgrade the software

Bad

- Doesn’t really solve any other problems
- Usually has disappointing scale and feature limitations in practice
- Doesn’t work for anything more than the dumbest of keepalives

Feels so close…

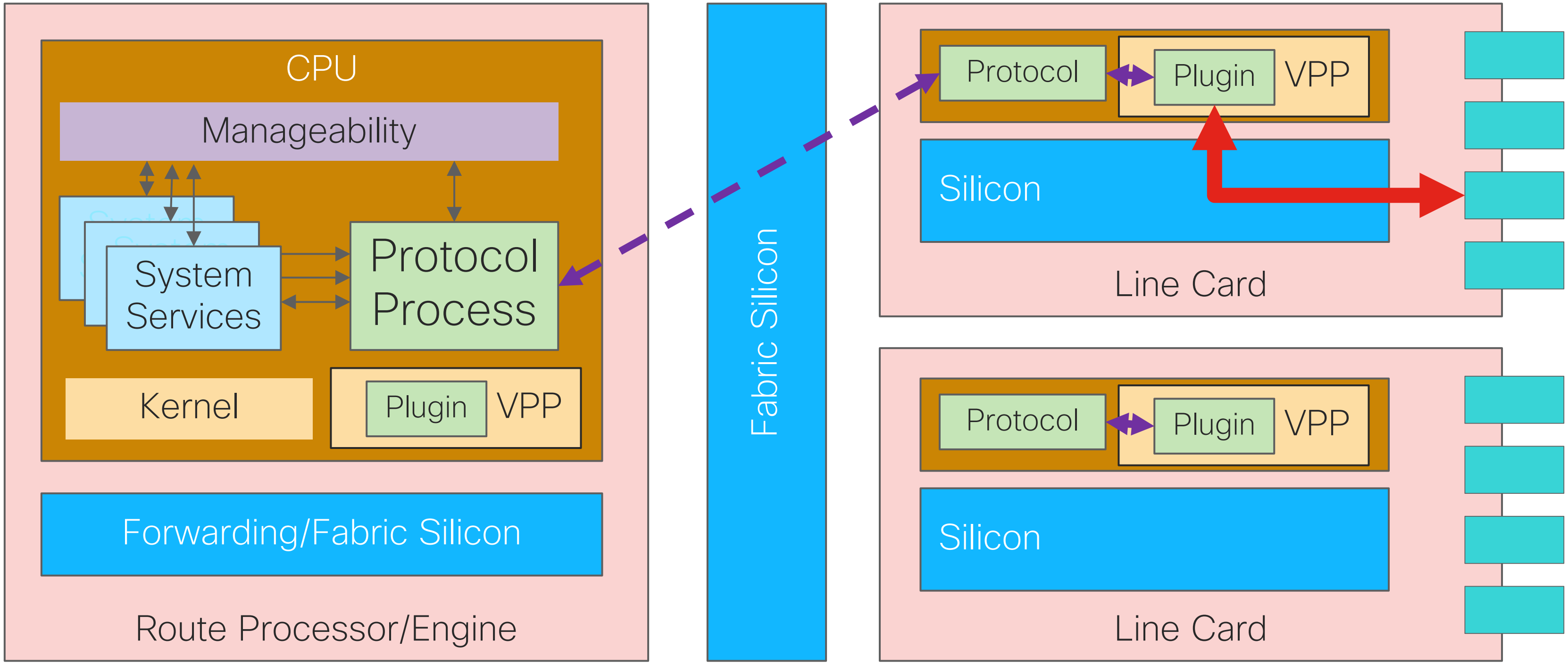
“Powerful but reduced complexity” element for reliable scale and low latency

“Smart but weedy”element for advanced protocol features without real-time guarantees

Vector Packet Processing

- VPP gives us “software microcode”
- Applying careful cache and memory access management lets us do limited packet processing in software at very high rates
- Lets us split protocols into a “fast path” (plugin within the single VPP process) and a “slow path” (dealing with non-keepalive packets and anything complex)
- fd.io is the third-generation of a technique refined in production for well over ten years

VPP offload


CPU
Manageability
System Services
Protocol Process
Kernel
Plugin
VPP
Forwarding/Fabric Silicon
Route Processor/Engine
Fabric Silicon
Protocol
Plugin
VPP
Silicon
Line Card
Protocol
Plugin
VPP
Silicon
Line Card

Pros/cons of VPP offload

Good

- A single VPP process (with a single high-frequency timer wheel) localises all intense activity into a single CPU/OS/cache-friendly workload
- Plugins can accommodate complex offload logic (eg precision time)
- Most upgrades hit the "complex" process not the offload plugin
- Scales much higher than h/w offload(!)

Bad

- Still need to do work for protocols not tied to individual cards
  - Hot standby VPP plugins ready to take over, often with duplicate packet sends

Things to ponder

- I always reach first for this model now (even for pizza boxes)
  - Clickbait alternative title: “Sockets are dead!”
- High performance always requires measurement
  - eg cost of “sending a bit” vs “creating a bit”
- If you’re designing a production protocol, please consider how the messaging splits across these two functions
  - A clean split really helps
  - Bonus points: a message for “My higher-level control function is going away for a few seconds, please chill out a bit” and “Two senders are ok”

cisco