NIL-MIGRATION

(Nearly Instantaneous Live Migration of Virtual Machines, Containers, and Processes)
BoF: VM Live Migration over CXL memory

LSF/MM/BPF Summit 2023
dragan@stancevic.com

BOF PREAMBLE

- Will give high-level overview
- Would like to discuss/brainstorm ideas
- Just starting to tackle this problem, don't have all the answers yet.
- Project assumes CXL 3.0 spec devices, with shared memory features
- Using VM as an example, but concept could be applied to containers and processes

dragan@stancevic.com

THEORETICAL GOAL

- Traditional LM: pre-copy, quiesce, 2nd-copy, un-quiesce
- nil-migration: migrate_pages, pass by reference/the handoff
- Remove the need to freeze memory (quiesce, 2nd-copy), by redirecting where you take page faults
- Aim to make VM live migration as instantaneous as multitasking
- VM migration could occur in-between allocated processor slices
- When a VCPU uses up its allocated processor slice on the source hypervisor, the next time slice the VCPU receives would be on the target hypervisor.

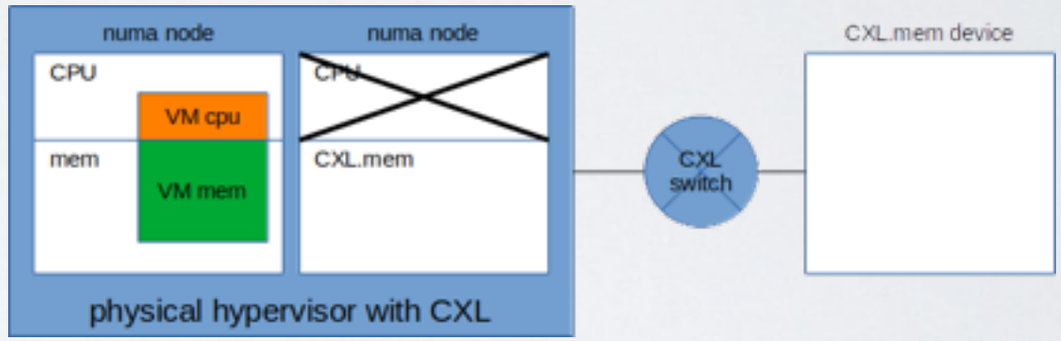
dragan@stancevic.com

TOPOLOGY

- Traditional CPU+memory on one NUMA node

- CXL.mem device; seen as a separate compute-less NUMA node

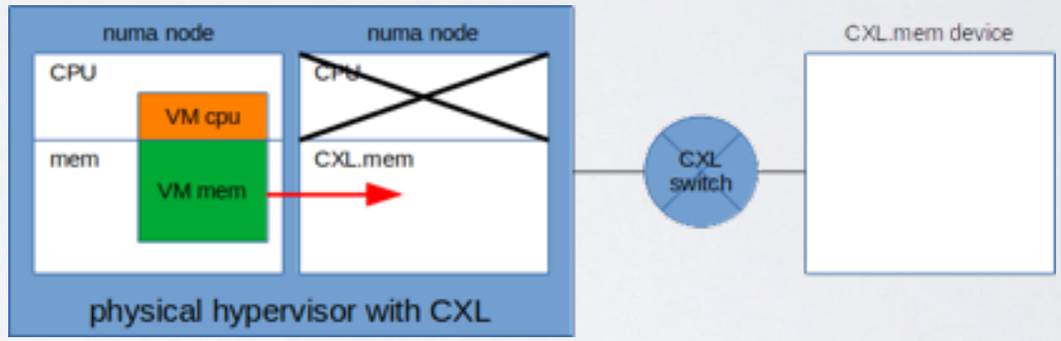
- Connected via CXL switch



dragan@stancevic.com

FIRST STEP - DECOUPLE

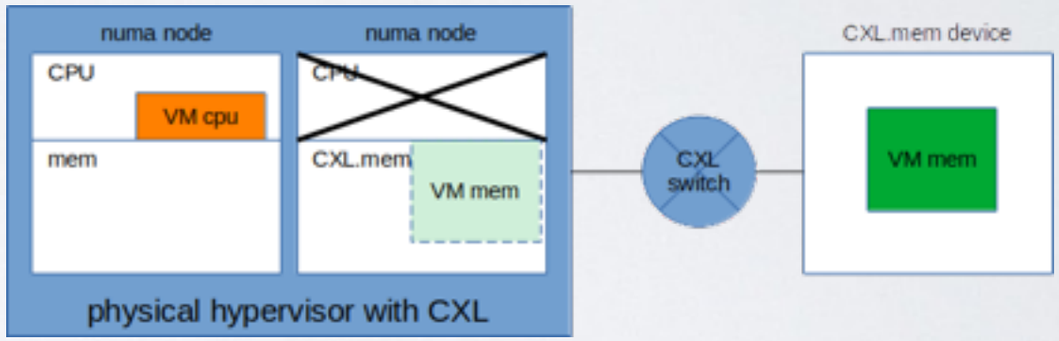
- Decouple the VM memory from the VM compute
- Migrate the VM memory onto one of the compute-less NUMA nodes.
- migrate_pages(), etc
- VM is live and executing



dragan@stancevic.com

DECOUPLE COMPLETE

- The VM cpu is executed on a traditional NUMA node
- VM memory footprint is on the cpu-less NUMA node
- VCPU loads and stores are redirected to the CXL.mem device
- At this point, the VM memory is physically stored on a CXL.mem device connected to our hypervisor through a CXL switch
- Use set_mempolicy and Co to prevent pages from migrating back out of CXL.mem
- VM is live and executing

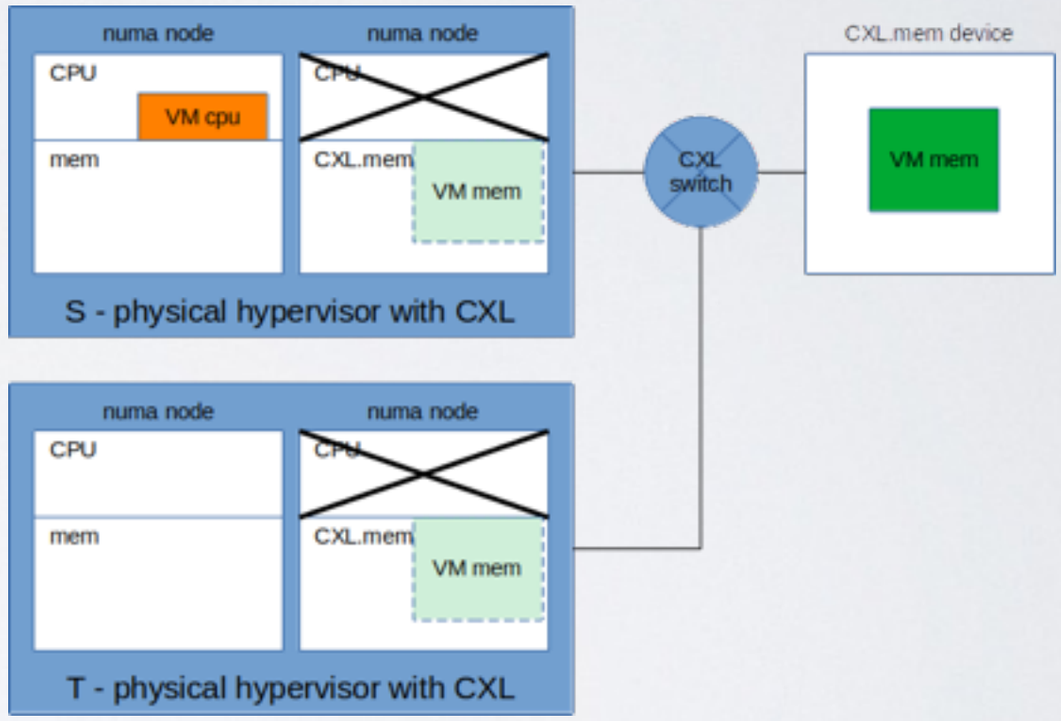


dragan@stancevic.com

TOPOLOGY - SHARED

- Add additional hypervisor accessing the same CXL.mem device where our VM is, through the same CXL switch.

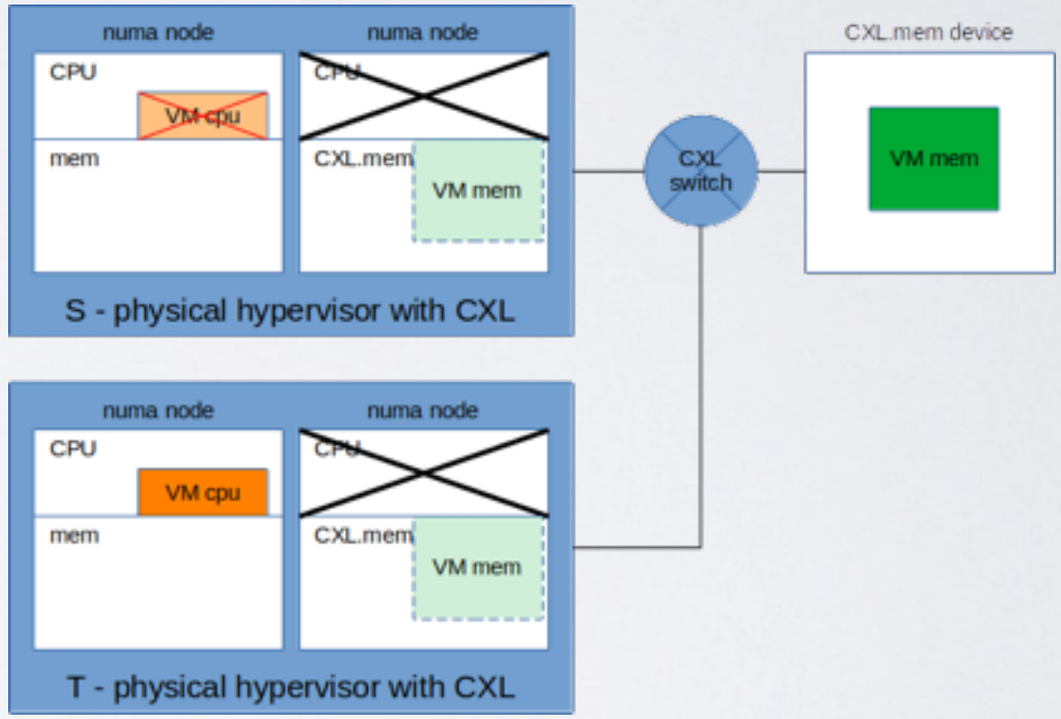
- We'll call this new hypervisor the target(T) hypervisor, and the original one we'll call the source(S).



dragan@stancevic.com

THE HANDOFF

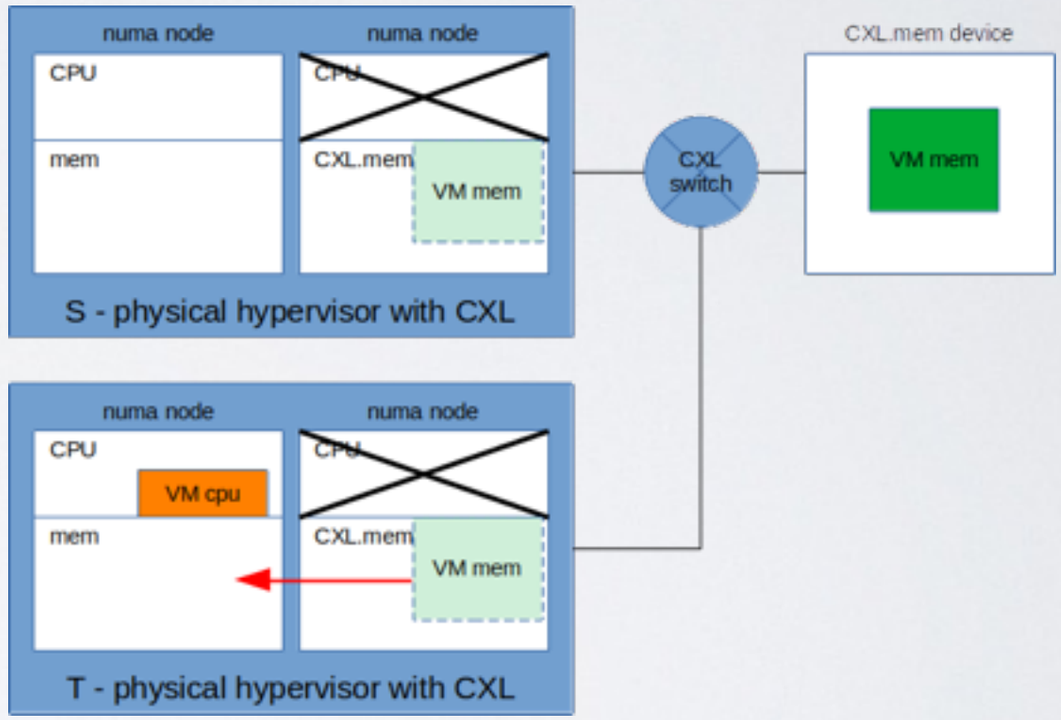
- The next step of nil-migration is to "recreate" the compute portion of the VM on the target hypervisor and destroy it on the source.
- In a perfect world, this would be as simple as a process context switch
- When the VCPU uses up it's processor time slice on the source, the context is stored in memory.
- And the context is simply restored on the target hypervisor
- The hypervisors would need a way to cooperate on the shared memory.
- There would be a need for some kind of in-memory handoff ABI between the hypervisors



dragan@stancevic.com

FINAL STEP - RECOUPLE

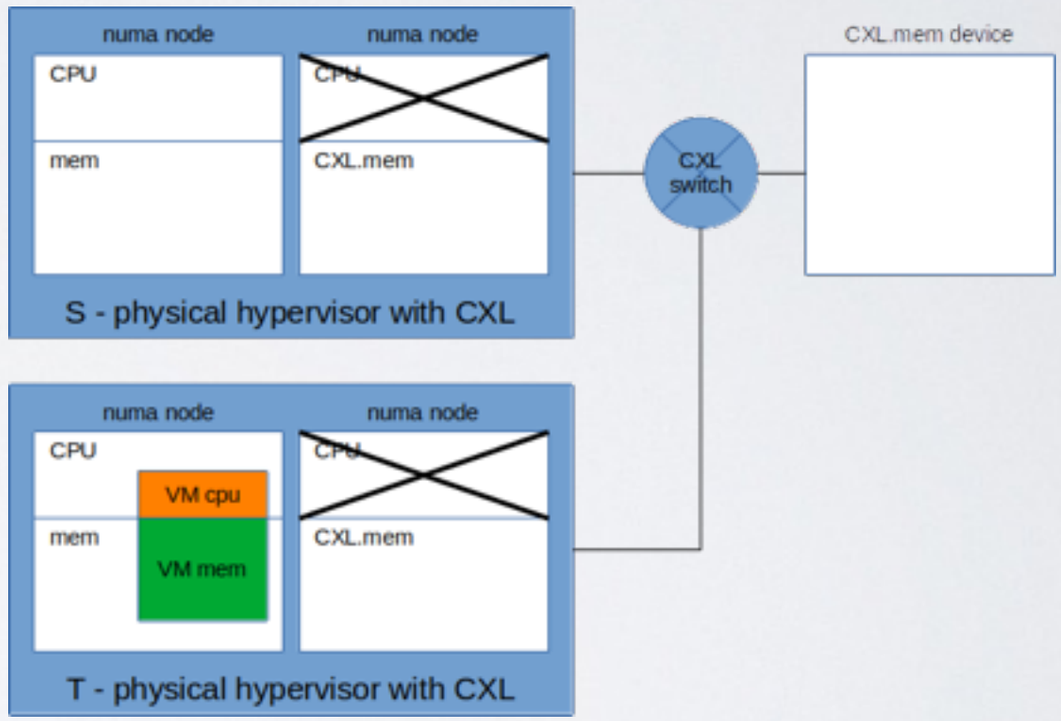
- After the handoff, the VM is executing on the target hypervisor and the VM memory footprint is on the compute-less NUMA node backed by the CXL.mem device.
- At this point, the VM compute and memory are still split between the traditional NUMA node and the compute-less NUMA node. We want to merge them back again.
- Migrate the VM memory onto one of the traditional NUMA nodes (CPU+memory)
- migrate_pages(), etc
- VM is live and executing



dragan@stancevic.com

RECOUPLE COMPLETE

- The VM compute and memory are now on a traditional numa node of the target hypervisor.
- VM is live and executing



dragan@stancevic.com

PROJECT INFO

- nil-migration is in its early stages of design/development, all development and prototyping are/will be done using qemu.
- As of this moment, CXL aspects needed for nil-migration to function are not fully implemented neither in the kernel or qemu.
- homepage https://nil-migration.org/
- nil-migration@lists.linux.dev mailing list hosted at https://subspace.kernel.org/lists.linux.dev.html
- Cc-ing: linux-cxl@vger.kernel.org and/or linux-mm@kvack.org

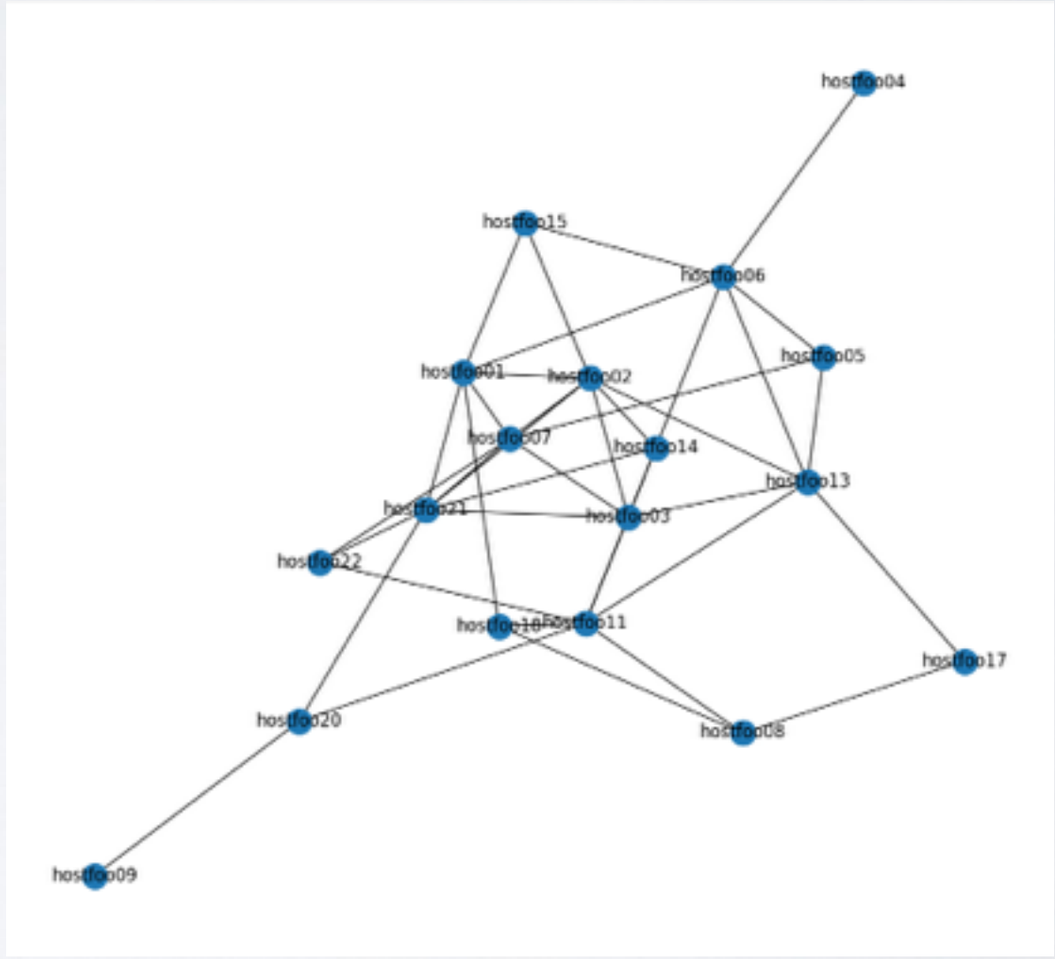
dragan@stancevic.com

DISCUSSION

EXTRA SLIDES

RACK TOPOLOGY

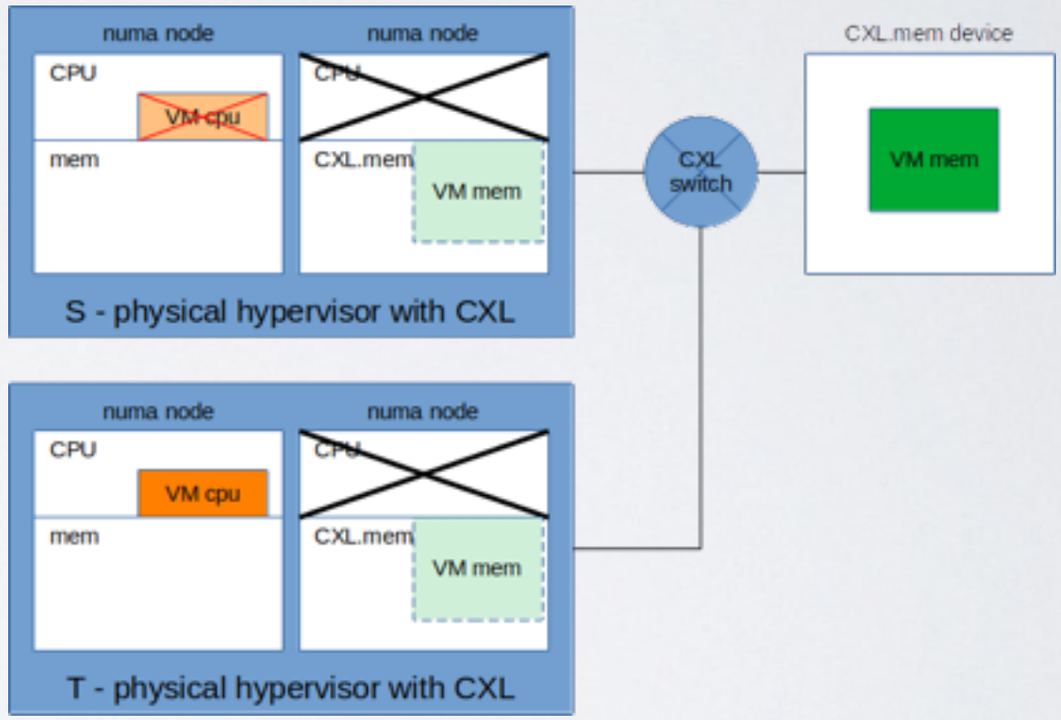
- Multiple memory devices per rack, connected to various hypervisors to form a hypervisor traversal graph
- A VM would migrate across a single hop, or a few hops to reach it's destination hypervisor
- For the lack of better word, this would be your "migration namespace" to migrate the VM across the rack.
- The critical connections in the graph are hostfoo04 and hostfoo09, and those you'd use if you want to pop the VM into a different "migration namespace", for example a different rack or maybe even a pod



dragan@stancevic.com

HYPERVISOR CLUSTERING

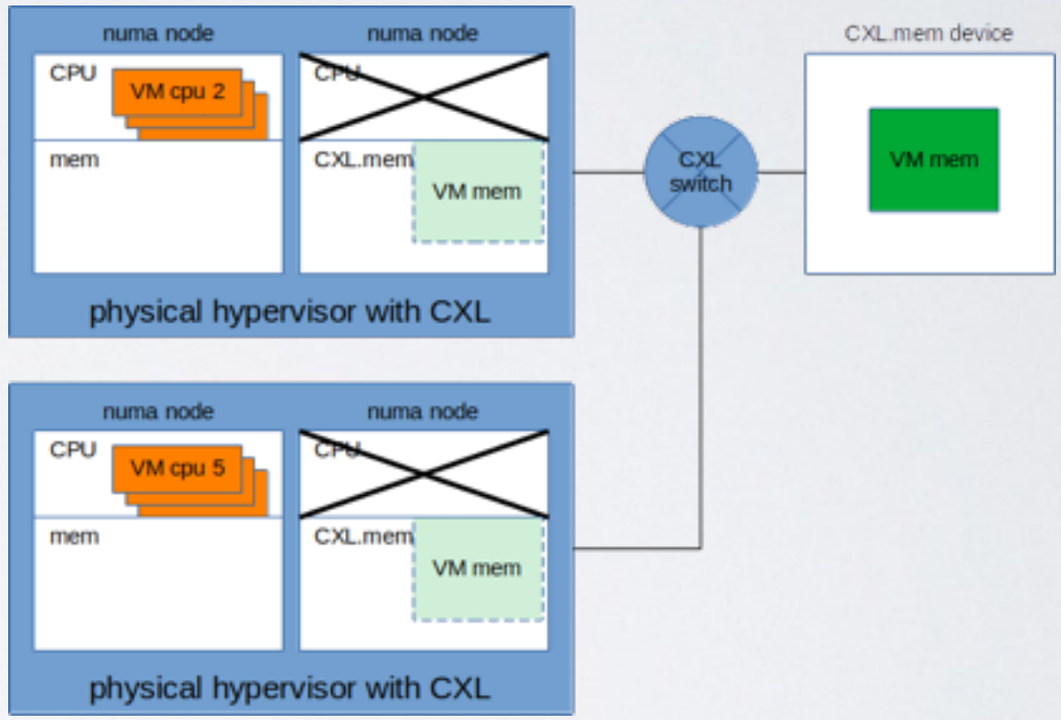
- Another aspect of nil-migration applicability is Hypervisor Clustering, where in case of a crash of hypervisor S, hypervisor T can take over the execution of the VM



dragan@stancevic.com

LOAD BALANCING

- Hypervisor clustering could also potentially be used for quasi-real-time CPU load distribution or CPU disaggregation.
- One hypervisor could be running one set of the VM's virtual CPUs, and a different hypervisor could run another set of virtual CPUs from the same VM.
- Since memory now becomes a discrete component, you are on the flip side disaggregating CPU as well
- You can dynamically attach more CPU power to this discrete shared memory by adding additional hypervisors, in essence creating a sort of hybrid of both vertical and horizontal expansion.



dragan@stancevic.com