

Problem / Overview

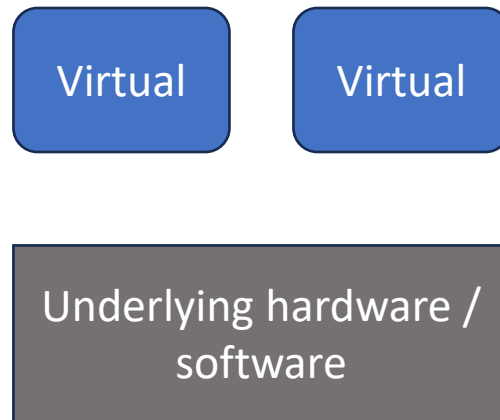
Course: Networking Principles in Practice – Linux Networking
Module: Virtual Networking in Linux



University of Colorado **Boulder**

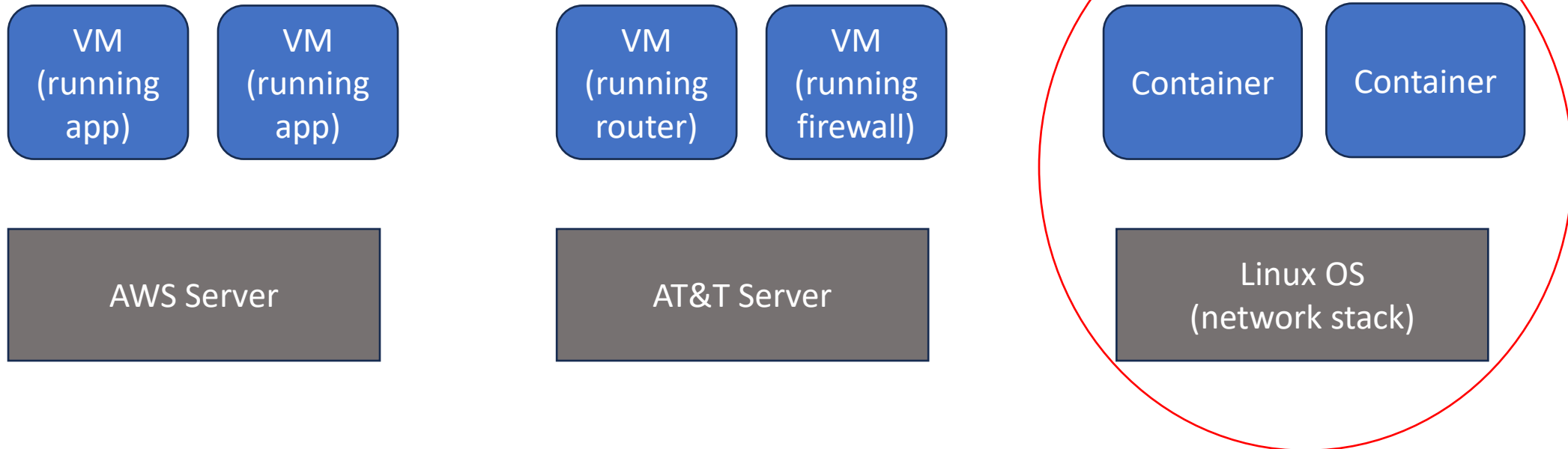
What is Virtualization?

- Technology for abstracting the underlying hardware / software
- Benefits:
 - More efficient use of resources
 - Simplified deployment by isolating configuration



Examples of Virtualization

- Cloud Computing
- Network Function Virtualization
- Virtualizing the network stack in Linux <= our focus



Outline

- Namespaces
- ip netns
- Networking Between Namespaces
- Docker Networking



University of Colorado **Boulder**

Network Namespaces

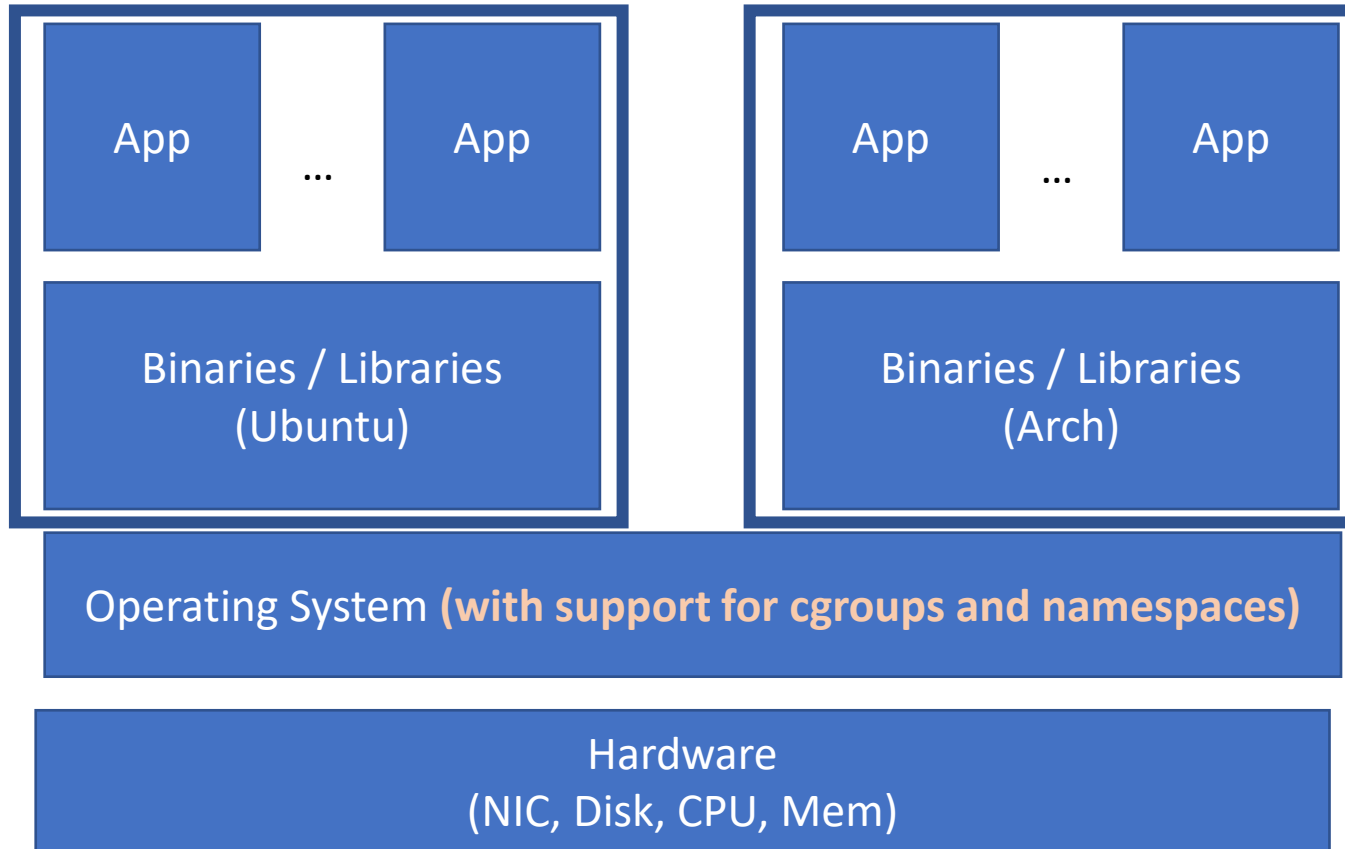
Course: Networking Principles in Practice – Linux Networking
Module: Virtual Networking in Linux



University of Colorado **Boulder**

Bare Metal -> Virtual Machines -> Containers

What if we don't need different OSes? Soln: Introduce isolation mechanisms into OS

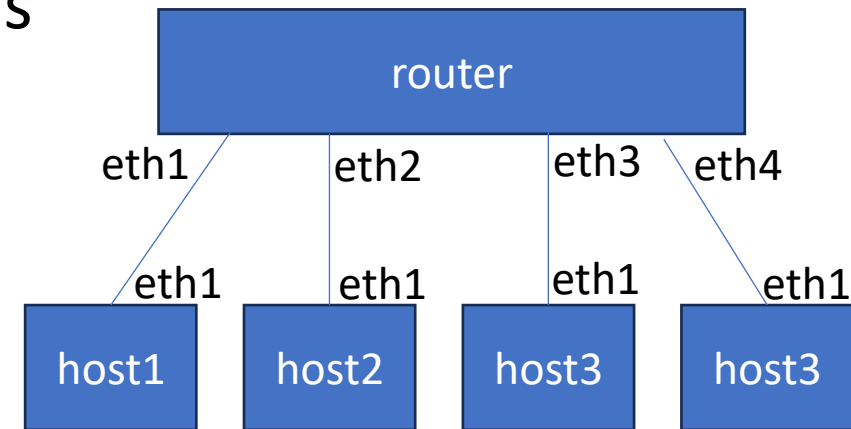


namespace - what resources and naming of those resources a process sees (file descriptors, IP addresses)

cgroup - (control group) groups processes and allocates resources (CPU, Memory) that the kernel enforces

Example Similar to Labs

- Containerlab Configuration file
- containerlab deploy – created docker containers
- docker exec commands inside of container



```
docker exec -it clab-demo-host1 ip link set dev eth1 address 22:33:22:44:55:44
```

```
docker exec -it clab-demo-router ip route add 10.0.2.0/24 dev eth2
```


namespaces

Creates isolation in the kernel that allows processes to have their own namespace for these resources.

Linux namespaces

Kernel maintains data structures on a per-process basis (file system, process IDs, etc.)

```
742
743 struct task_struct {
744     #ifdef CONFIG_THREAD_INFO_IN_TASK
745         /*
746          * For reasons of header soup (see current_thread_info()), this
747          * must be the first element of task_struct.
748          */
749         struct thread_info      thread_info;
750     #endif
751         unsigned int            __state;
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
```

```
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
```

<https://elixir.bootlin.com/linux/v6.6.7/source/include/linux/sched.h#L743>

<https://elixir.bootlin.com/linux/v6.6.7/source/include/linux/nsproxy.h#L31>

Data Structures in the Linux Kernel

```
struct net {  
...  
    struct netns_ipv4 ipv4;  
...  
    struct netns_nf nf;
```

The relevant one to us is the struct net, which contains all of the ipv4 forwarding data structures, the netfilter tables, etc.

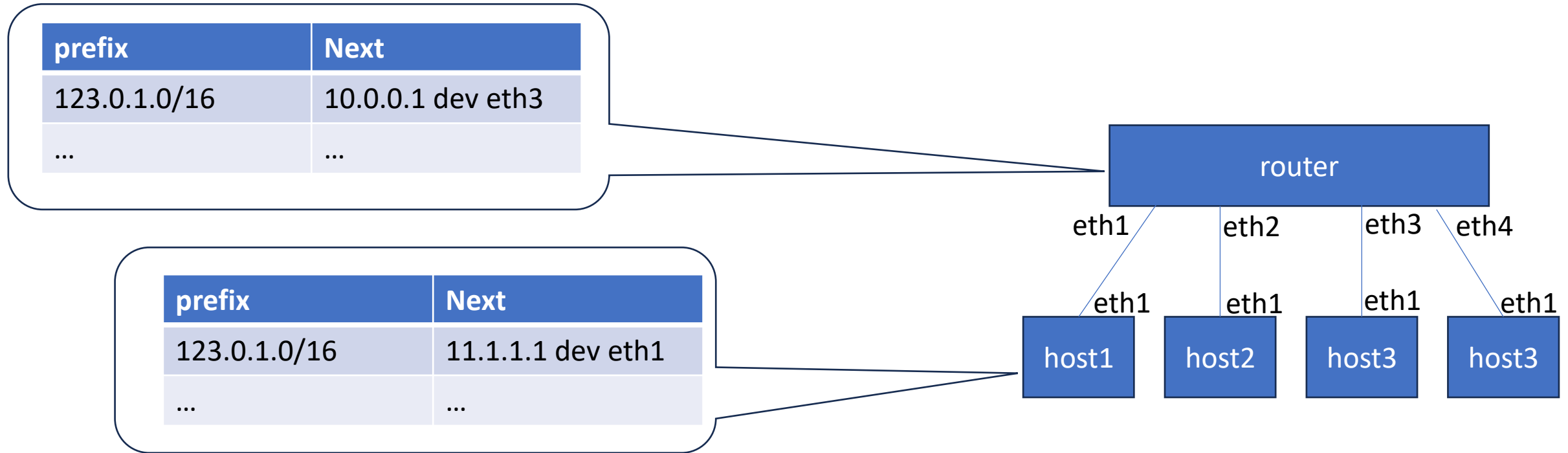
```
struct netns_ipv4 {  
  
#ifdef CONFIG_IP_MULTIPLE_TABLES  
    struct fib_rules_ops *rules_ops;  
    struct fib_table __rcu *fib_main;  
    struct fib_table __rcu *fib_default;  
    unsigned int fib_rules_require_fldissect;  
    bool fib_has_custom_rules;  
#endif  
    bool fib_has_custom_local_routes;  
    bool fib_offload_disabled;  
  
    struct hlist_head *fib_table_hash;  
    struct sock *fibnl;
```

https://elixir.bootlin.com/linux/v6.6.7/source/include/net/net_namespace.h#L61

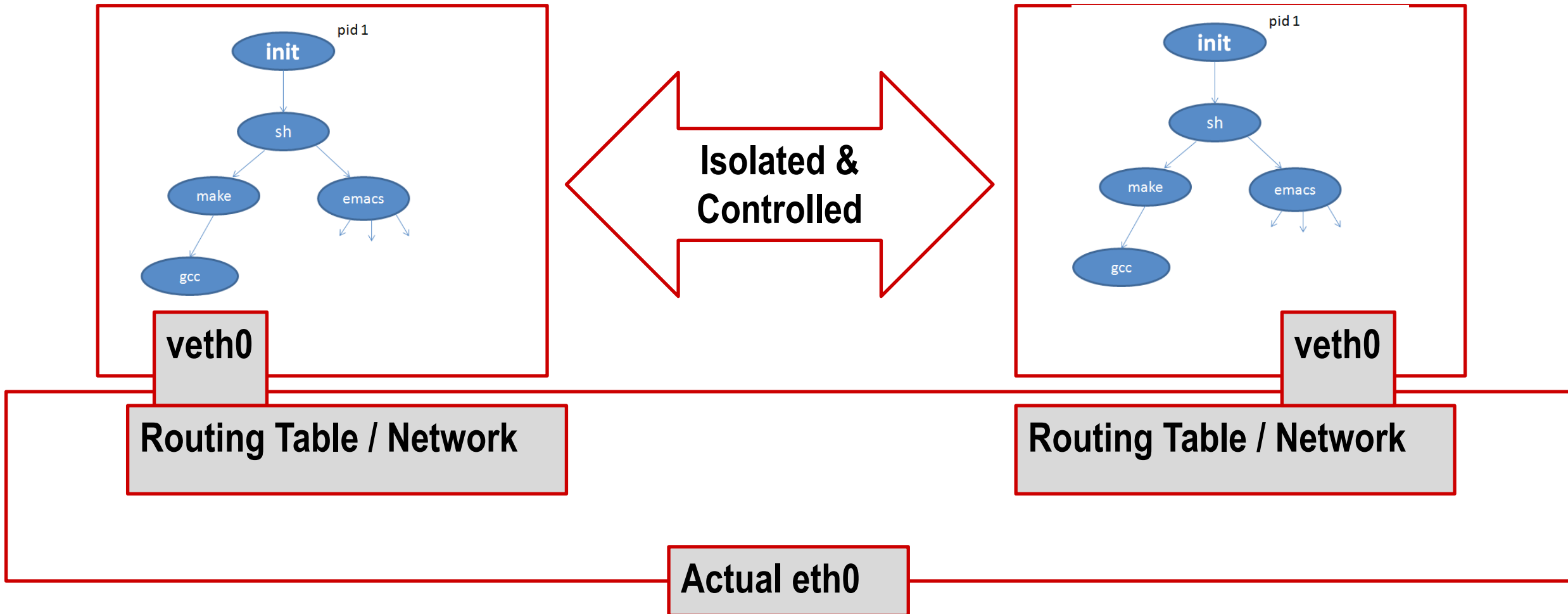
https://elixir.bootlin.com/linux/v6.6.7/source/include/net/netns_ipv4.h#L44

End Result:

Each namespace Gets its own set of tables



Virtualization using Namespaces





University of Colorado **Boulder**

ip netns

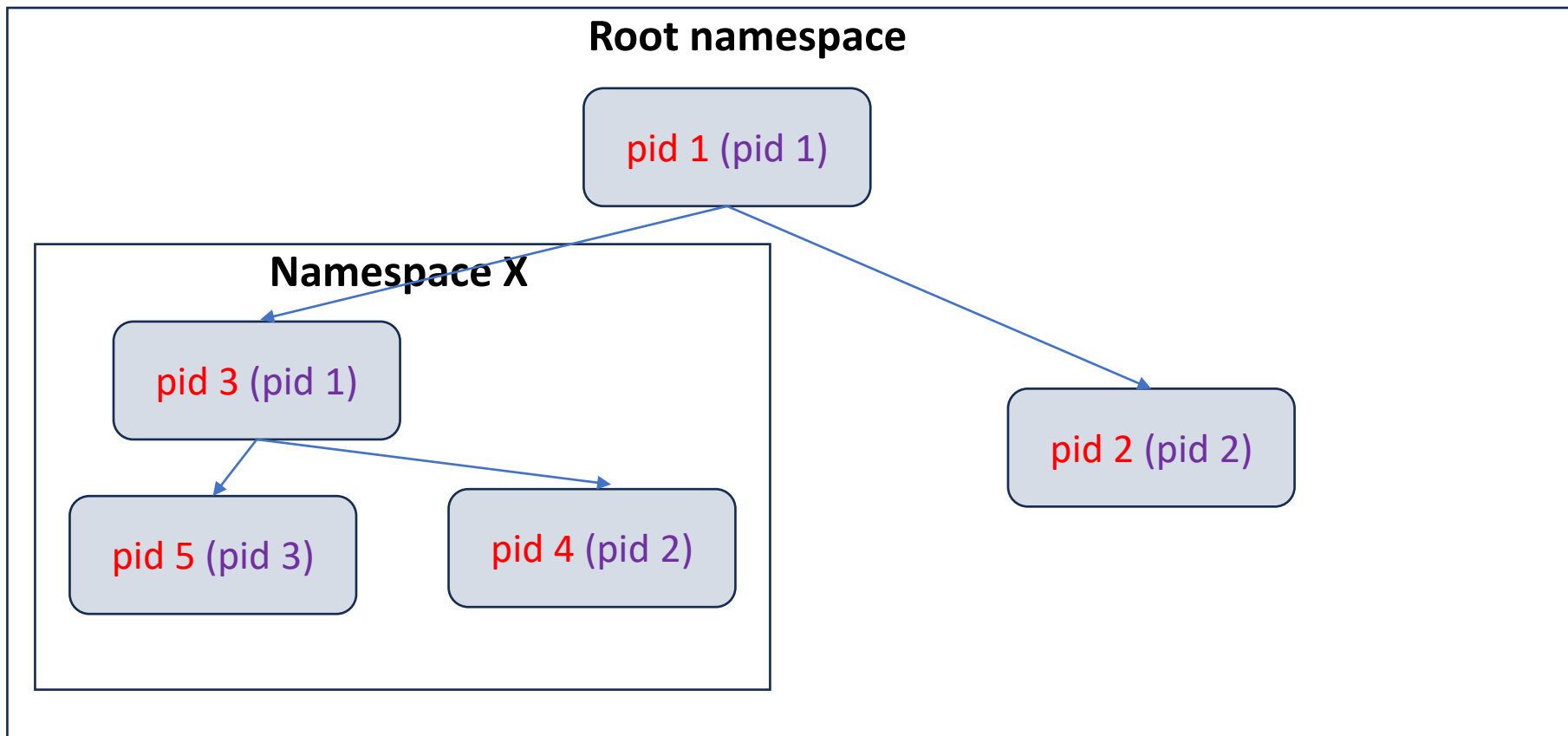
Course: Networking Principles in Practice – Linux Networking
Module: Virtual Networking in Linux



University of Colorado **Boulder**

Process Relationship to Namespaces

- Processes Inherit from parent (starting with root/default namespace)



ip netns

IP-NETNS(8)

Linux

IP-NETNS(8)

NAME [top](#)

`ip-netns` - process network namespace management

SYNOPSIS [top](#)

`ip [OPTIONS] netns { COMMAND | help }`

`ip netns [list]`

`ip netns add NETNSNAME`

`ip netns attach NETNSNAME PID`

`ip [-all] netns del [NETNSNAME]`

`ip netns set NETNSNAME NETNSID`

NETNSID := *auto* | *POSITIVE-INT*

...

<https://man7.org/linux/man-pages/man8/ip-netns.8.html>

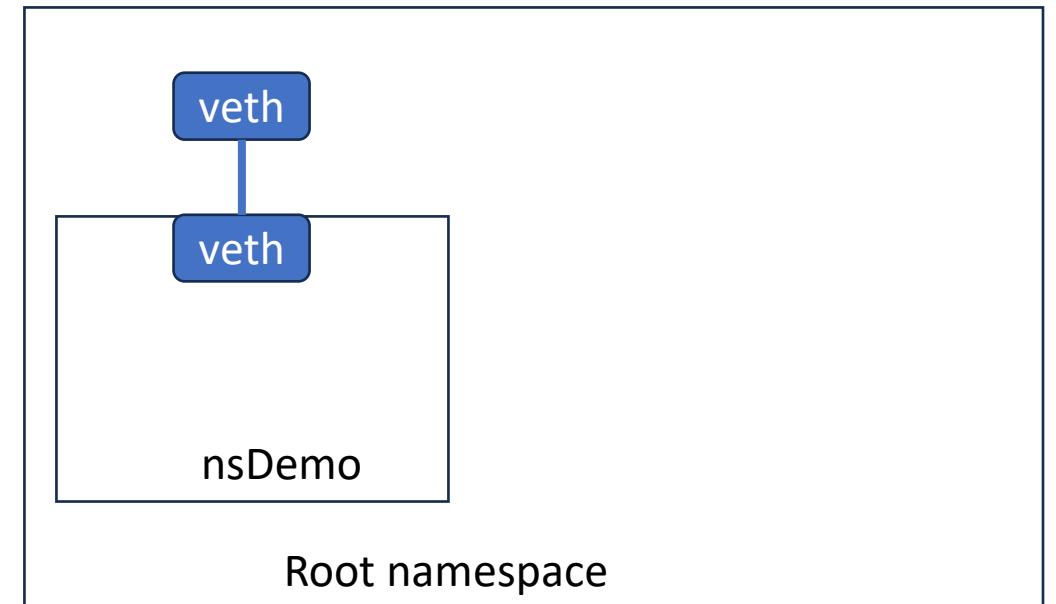
General Process

Setup:

- Create network namespace
- Create veth pair
- Attach veth devices to a namespace

Then you can:

- Execute commands inside namespace
- Use Linux networking
- ...



Setup: Create Network Namespace

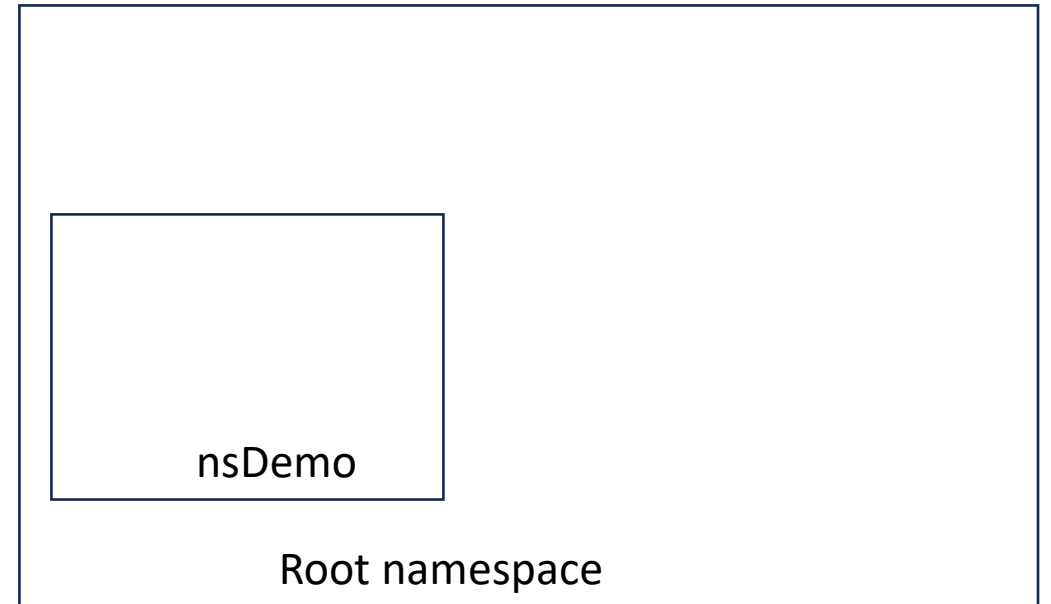
`ip netns add NAME`

- Create a new named network namespace

Example:

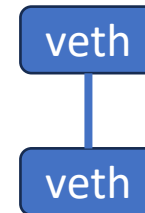
`ip netns add nsDemo`

`ip netns list`



veth devices

- veth - Virtual Ethernet device
- Always created in interconnected pairs
- Packets transmitted on one device in the pair are immediately received on the other device.



Setup: Create veth pair

```
ip link add <p1-name> type veth peer name <p2-name>
```

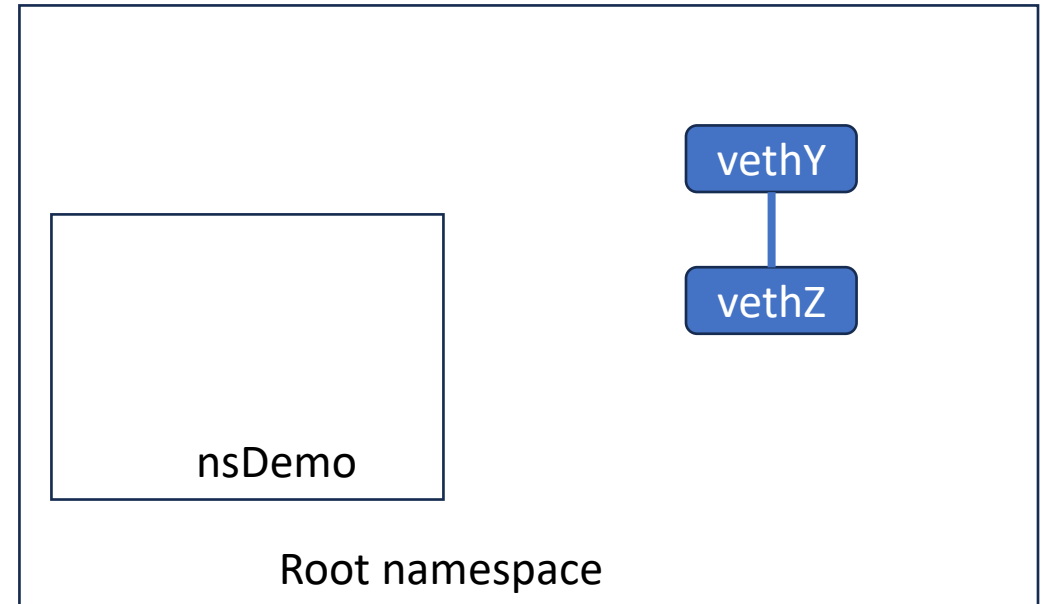
Example:

```
ip link add vethY type veth peer name vethZ
```

```
ip link # see both vethY and vethZ
```

```
ip link set dev vethY up
```

```
ip link set dev vethZ up
```



Finding the Peer

[ethtool\(8\)](#) can be used to find the peer of a **veth** network interface, using commands something like:

```
> ethtool -S vethY # Discover interface index of peer
```

```
NIC statistics:
```

```
peer_ifindex: 5
```

```
....
```

```
> ip link | grep '^5:' # Look up interface
```

```
5: vethZ@vethY: <BROADCAST,MULTICAST,M-DOWN> mtu 1500 qdisc ...
```

<https://man7.org/linux/man-pages/man4/veth.4.html>

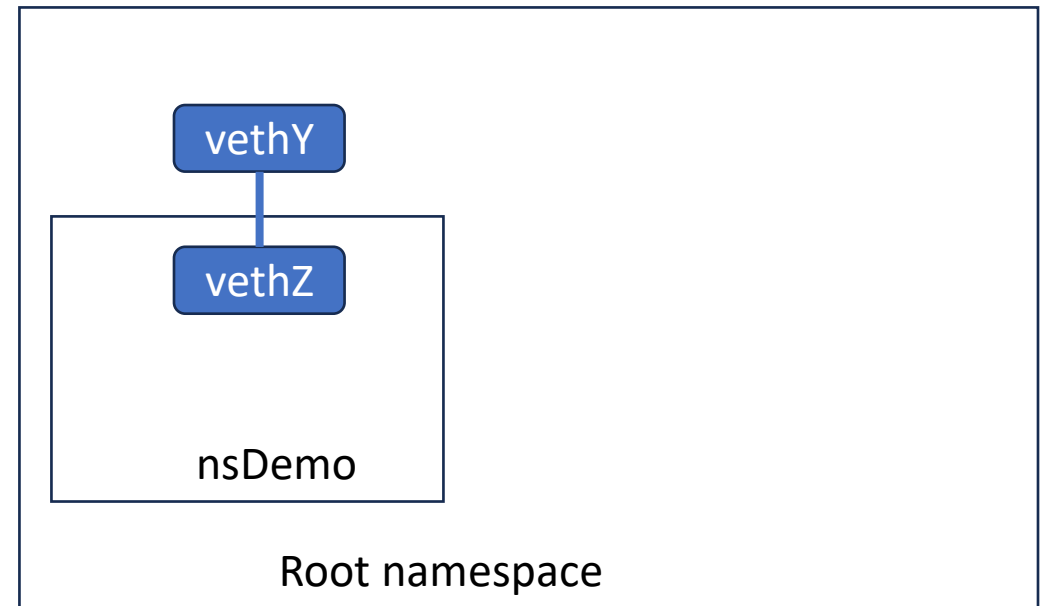
Setup: Attach to Namespace

```
ip link set <p2-name> netns <p2-ns>
```

Example:

```
ip link set vethZ netns nsDemo
```

```
ip link # note vethZ no longer shown
```



Can connect two namespaces

- Simply attach the other end of a veth pair to another namespace

Example:

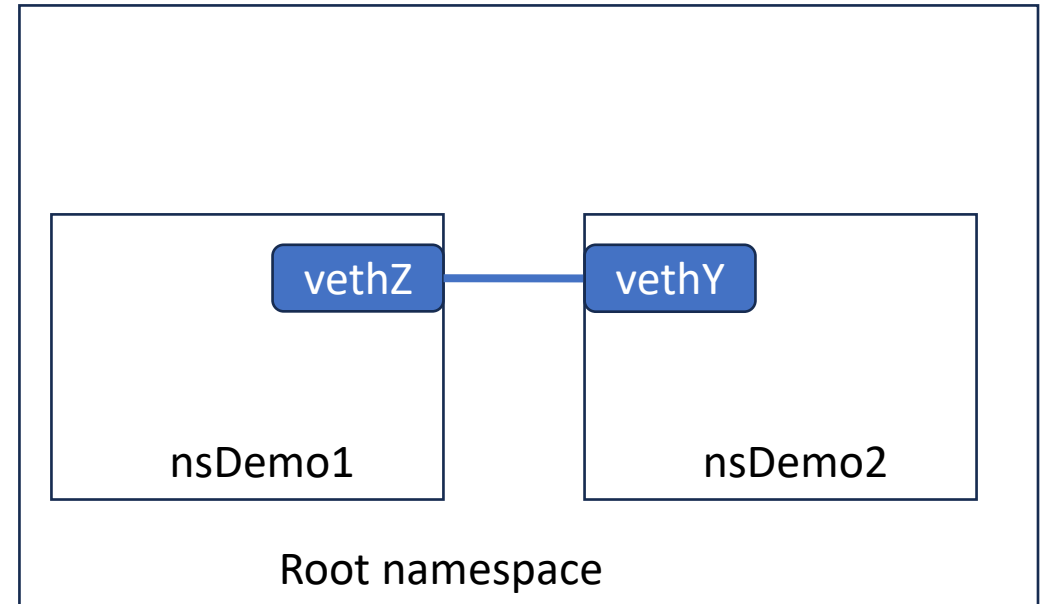
```
ip netns add nsDemo1
```

```
ip netns add nsDemo2
```

```
ip link add vethY type veth peer name vethZ
```

```
ip link set vethZ netns nsDemo1
```

```
ip link set vethY netns nsDemo2
```



After: Execute Commands in netns

- `ip netns exec <netns> <command>`

Example – set an address on each veth device, and ping between them:

```
ip netns exec nsDemo ip link
```

```
ip netns exec nsDemo ip addr add 10.10.10.10/24 dev vethZ
```

```
ip netns exec nsDemo ip link set vethZ up
```

```
ip addr add 10.10.10.11/24 dev vethY
```

```
ip netns exec nsDemo ping 10.10.10.11
```

```
ip netns exec nsDemo tcpdump -i vethZ
```

Other misc.

- Moving veth device out of a namespace

```
ip netns exec nsDemo ip link set vethZ netns 1 # 1 is default ns
```

- Note a `-n` option in ip commands (as alternative to `ip netns exec`):

```
ip -n <netns> <iproute2 command>
```

```
ip -n nsDemo route add 11.11.0.0/16 dev vethZ
```

```
ip -n nsDemo route
```

After: Linux Networking

- If you do “ip link” (in root namespace) you’ll see the veth devices that are in the root namespace.
- veth devices are just Ethernet devices.
- So, can do Linux networking stuff with them.
- We’ll explore more in the next lesson



University of Colorado **Boulder**

Networking Between Namespaces

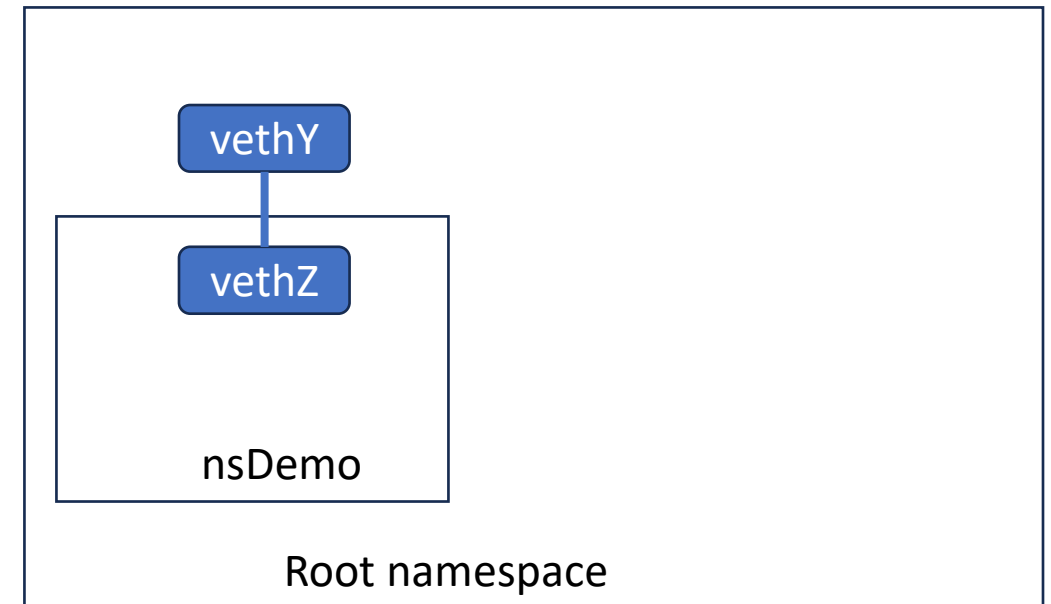
Course: Networking Principles in Practice – Linux Networking
Module: Virtual Networking in Linux



University of Colorado **Boulder**

Recap – Network Namespaces

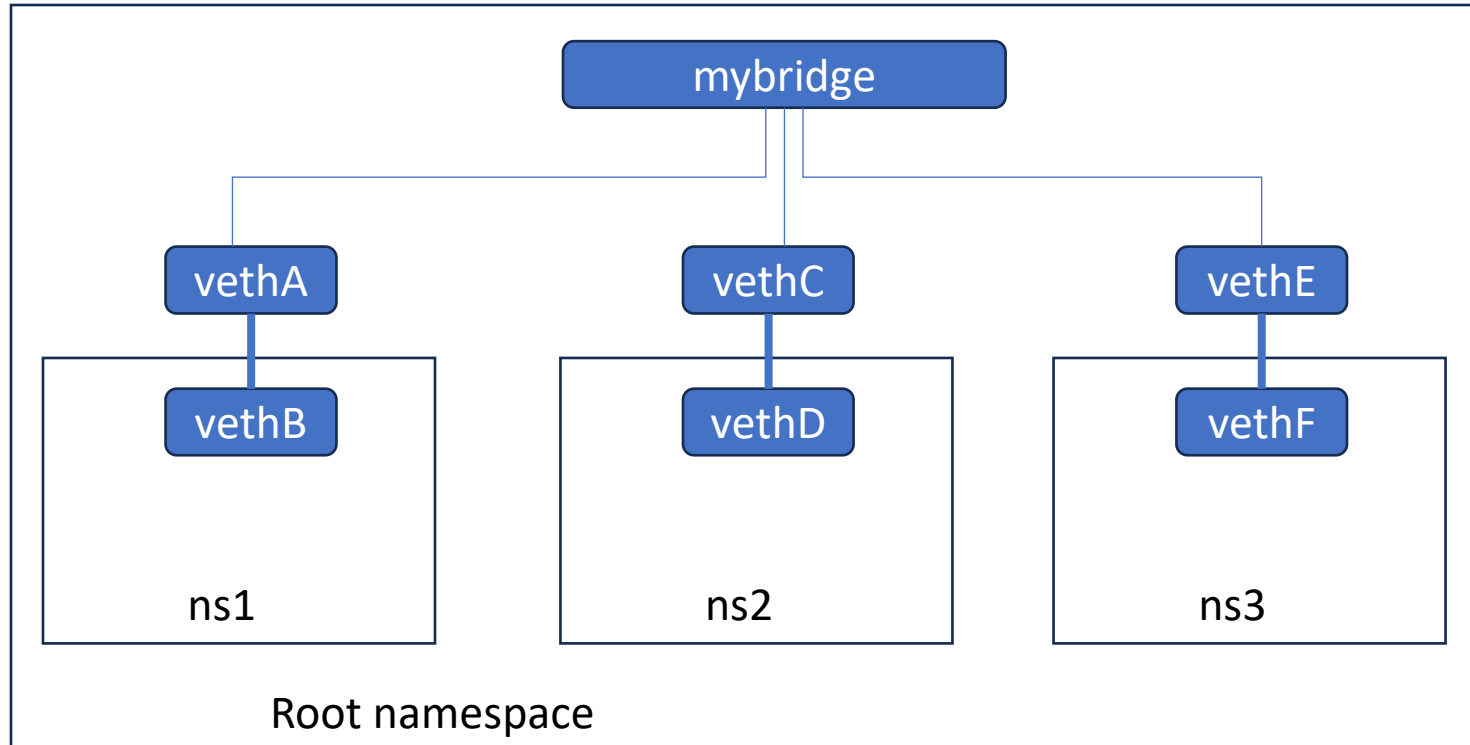
- Create network namespaces
`ip netns add nsDemo`
- Create veth pair
`ip link add vethY type veth peer name vethZ`
- Attach veth to network namespace
`ip link set vethZ netns nsDemo`
- Note: you'll also want to set the device state up, set an address and routes, etc.



Lab Setup

- Show vagrant file
- vagrant up
 - Brings up both VMs
- vagrant up node1
 - Brings up only node1
- Github link:
<https://github.com/eric-keller/npp-linux-04-virtual>

Case 1: Connecting Together w/ Bridge

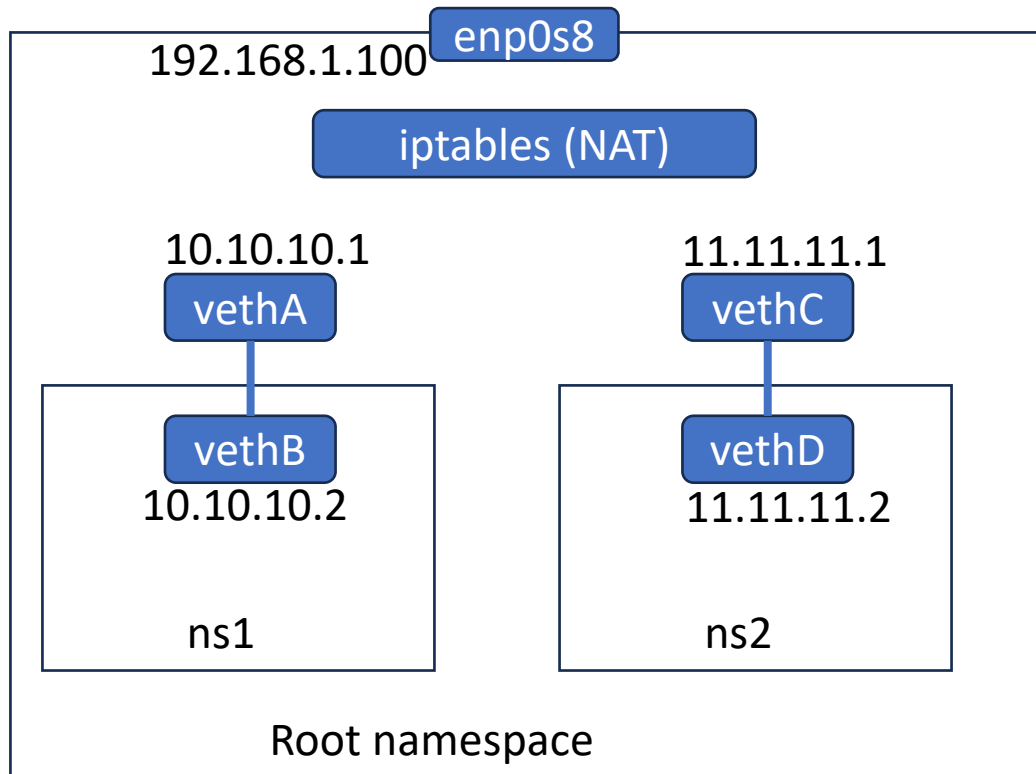


Switch to console

- Link to github

<https://github.com/eric-keller/npp-linux-04-virtual>

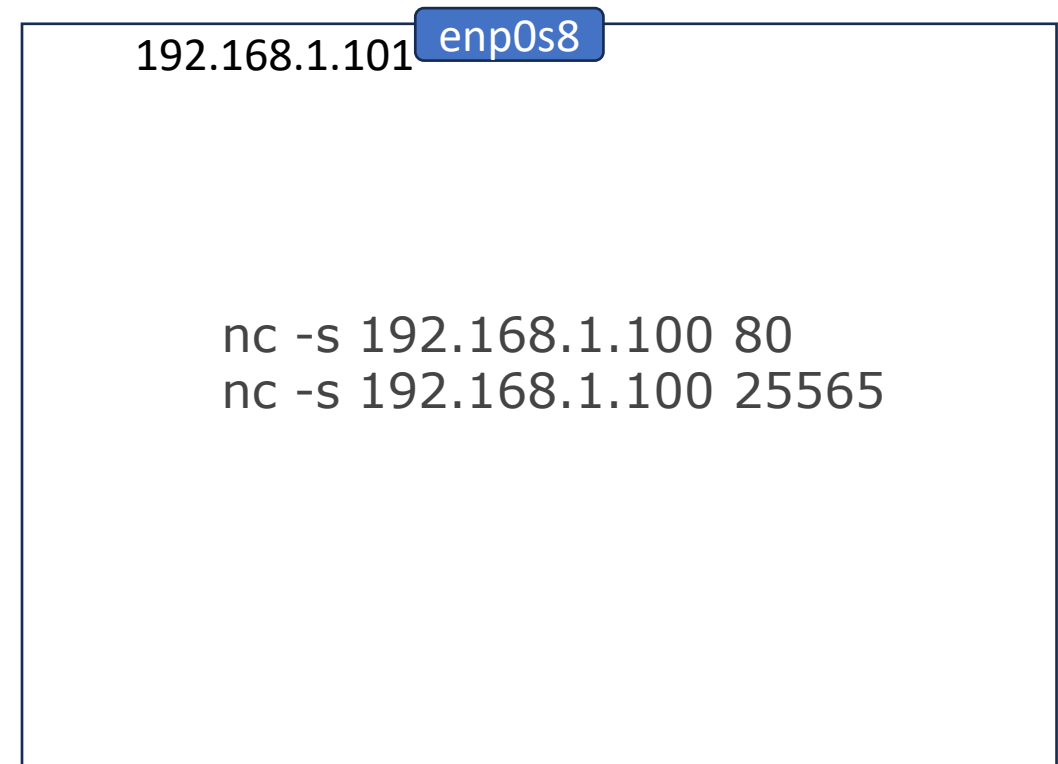
Case 2: Connecting To External World (e.g., port forwarding)



node1

In ns1 – run service on port 80

In ns2 – run service on port 25565



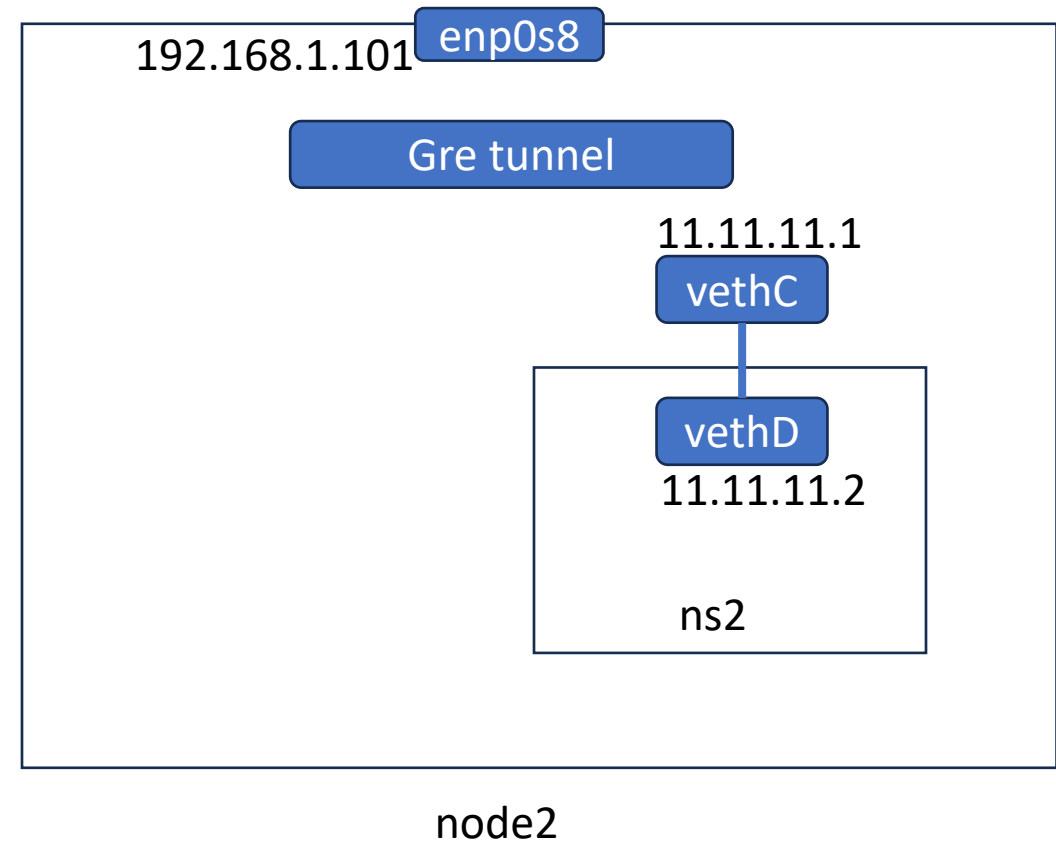
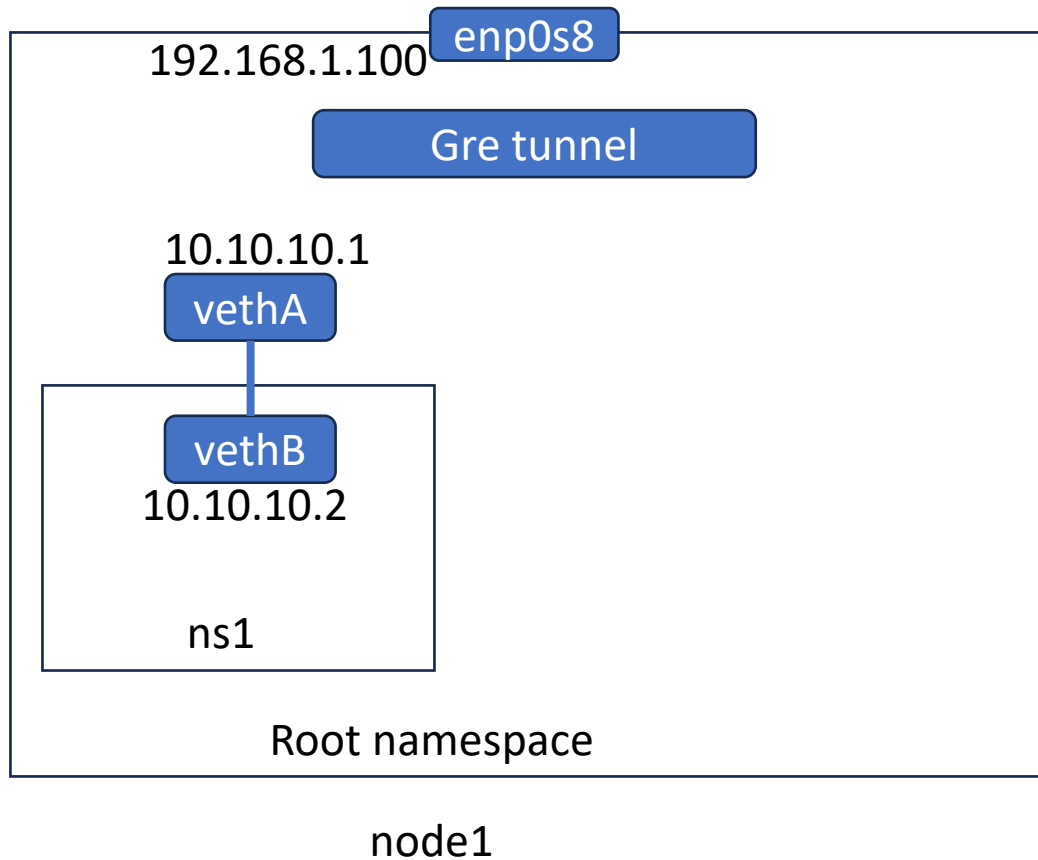
node2

Switch to console

- Link to github

<https://github.com/eric-keller/npp-linux-04-virtual>

Case 3: Extending network between machines (try on your own)



Try on your own

Lab Overview

- You are provided with a create and delete script that have been obfuscated.
(github link)
- Only need one node:
 - vagrant up node1
 - clear-fw.sh
 - ob-create-lab1-mod4.sh
 - Answer questions in Coursera
 - ob-del-lab1-mod4.sh
- Useful commands:
 - ip netns list, ip netns exec, ip route, ethtool, ip link, ping, tshark or tcpdump



University of Colorado **Boulder**

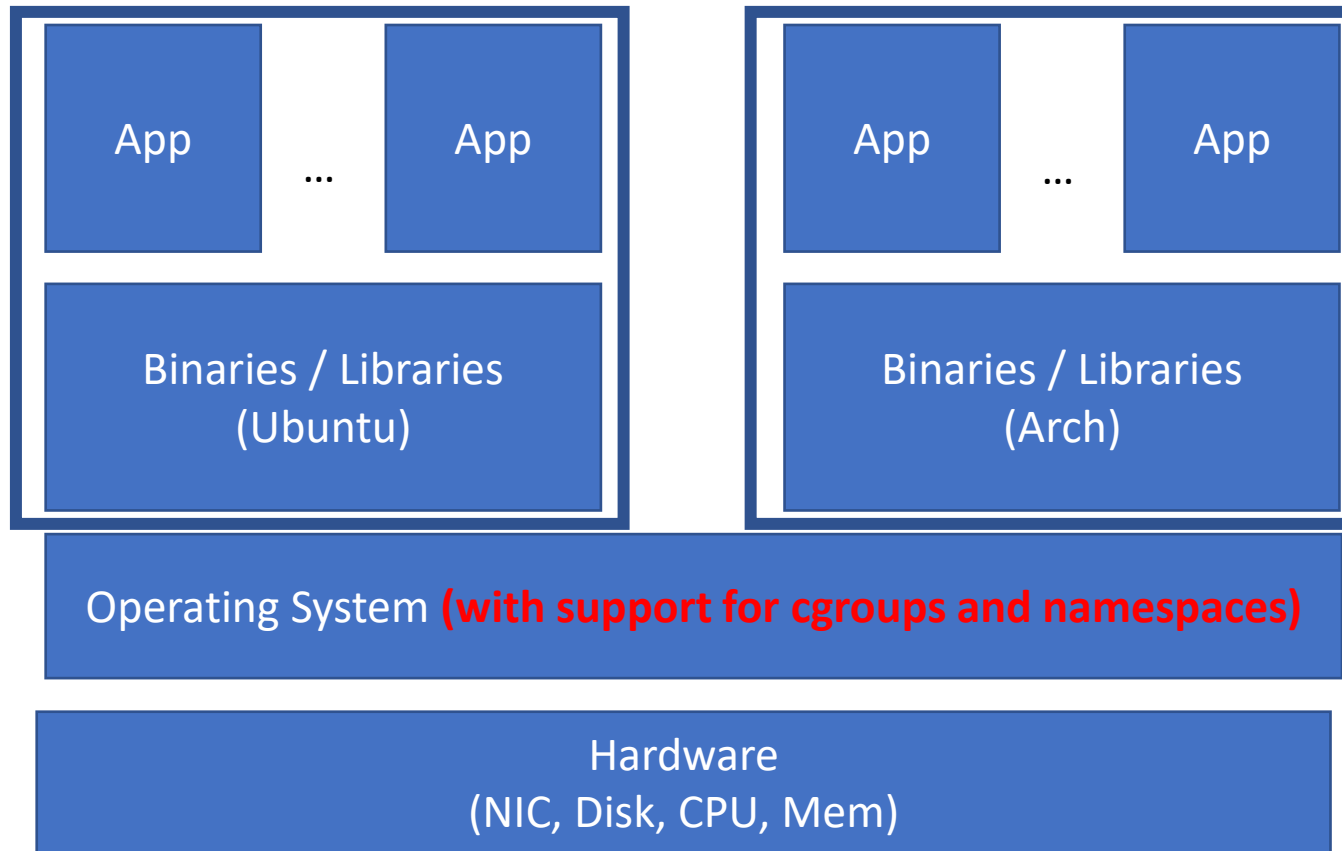
Docker Networking

Course: Networking Principles in Practice – Linux Networking
Module: Virtual Networking in Linux

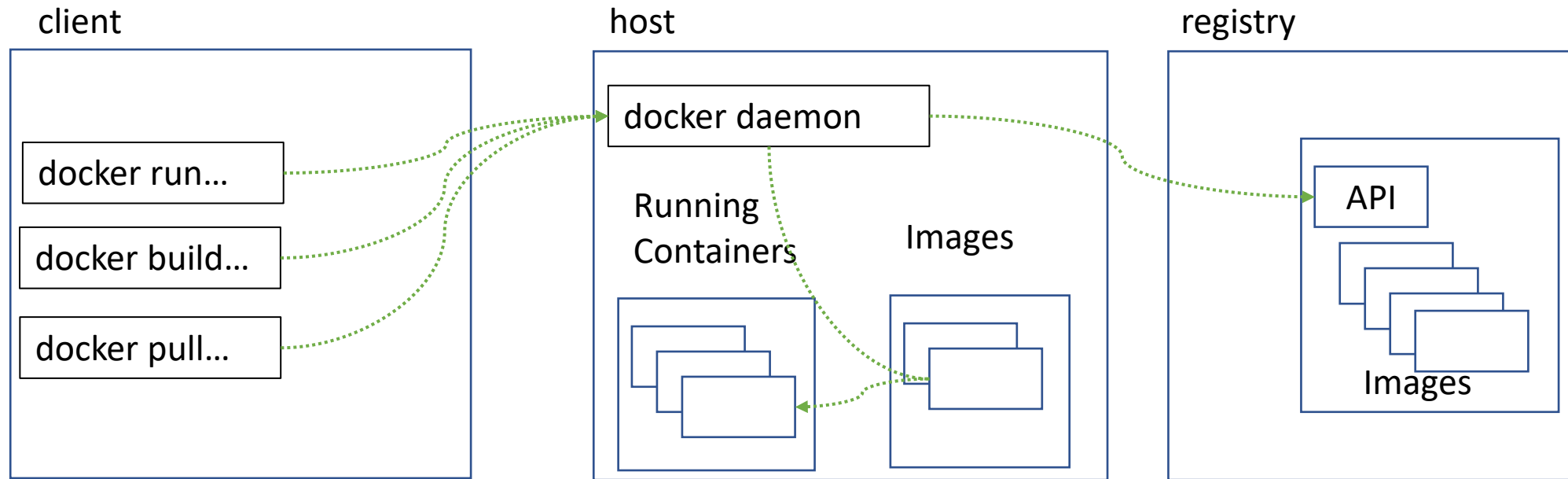


University of Colorado **Boulder**

Recap: Container



Recap: Docker



```
sudo docker run -ti --rm ubuntu:22.04 /bin/bash
```

Recap: Docker Containers

- A temporary filesystem
 - layered over an image
 - fully writable (copy on write)
 - disappears when End of Life
- A Network Stack
- A Process Group - one main process, with possible subprocesses (which exits when main process exits)

What Networking is Set Up?

- `sudo ip netns`
- `sudo ip link`
- `sudo docker run -d -rm --name nginx1 nginx`
- `sudo ip link`
- `sudo ip netns`

- Nothing is listed?

Docker / ip netns?

“ *ip netns ls* command looks up network namespaces file in the */var/run/netns* directory.

However, the **Docker daemon doesn't create a reference of the network namespace file in the */var/run/netns* directory after the creation.** Therefore, *ip netns ls* cannot resolve the network namespace file.”

<https://www.baeldung.com/linux/docker-network-namespace-invisible>

Soln – see `run-container.sh` (and `del-container.sh`)

Now, let's re-run ip link

- See the veth in the root namespace
 - `sudo ip link`
- Get the veth this is paired to (gives ID)
 - `ethtool -S <veth>`
- Look inside of container
 - `sudo ip netns exec <containername> ip link`
- There it is

Where does the veth traffic go (in root)?

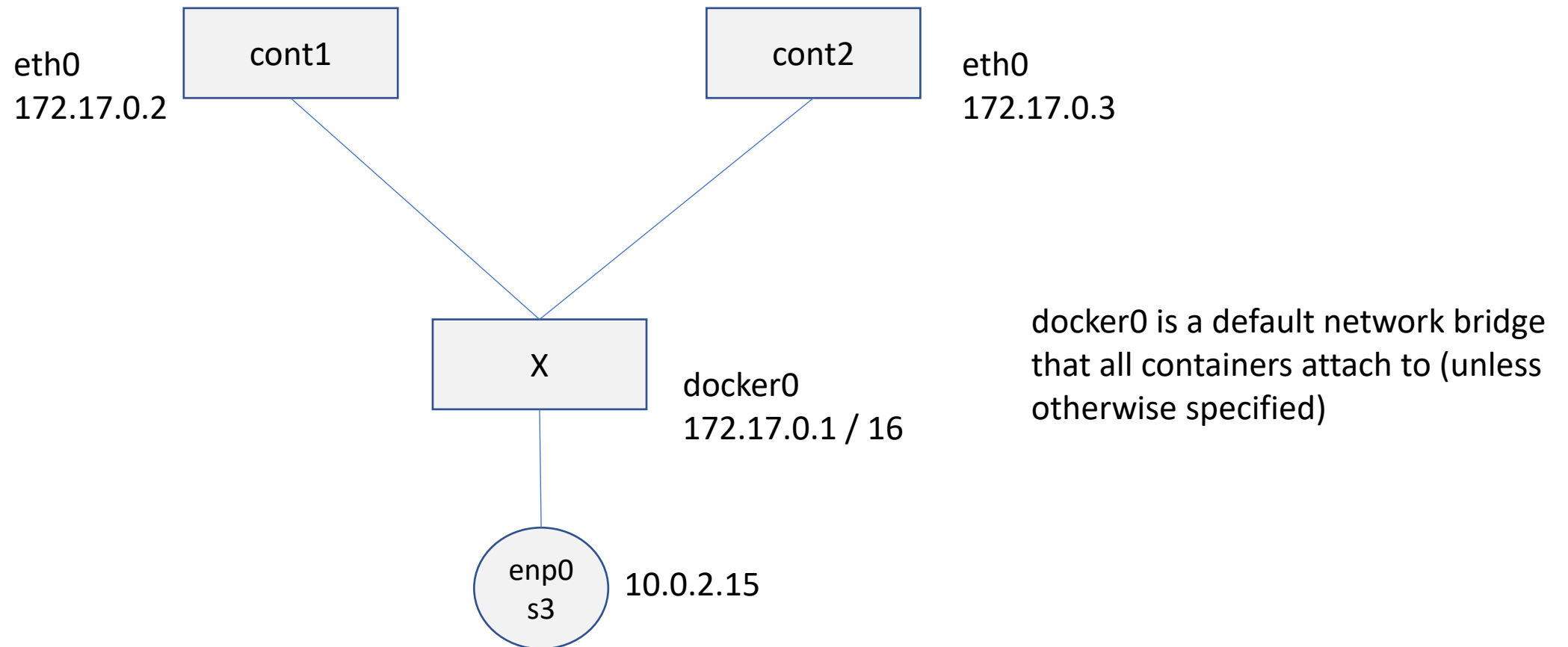
ip link

Sample output:

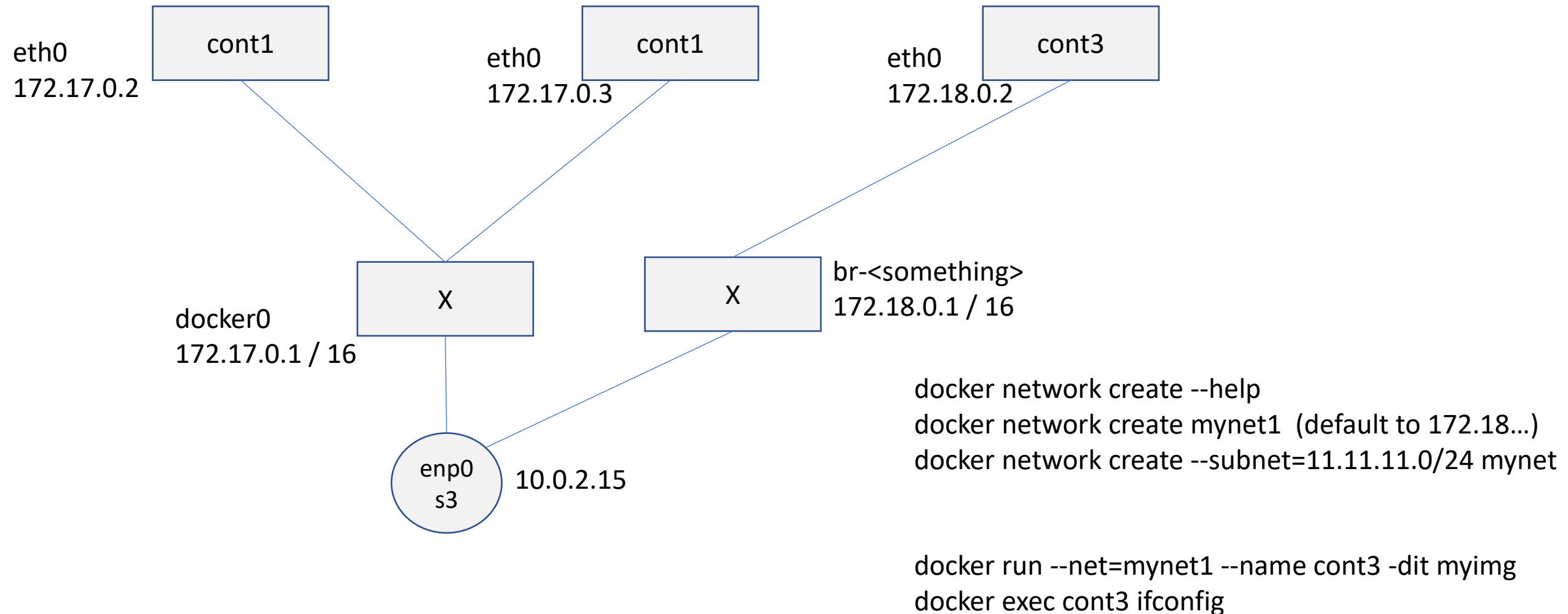
```
4: docker0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc  
noqueue state UP mode DEFAULT group default link/ether 02:42:ee:da:17:05  
brd ff:ff:ff:ff:ff:ff
```

```
84: veth34e2745@if83: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500  
qdisc noqueue master docker0 state UP mode DEFAULT group default  
link/ether 92:53:22:f2:24:6f brd ff:ff:ff:ff:ff:ff link-netnsid 2
```

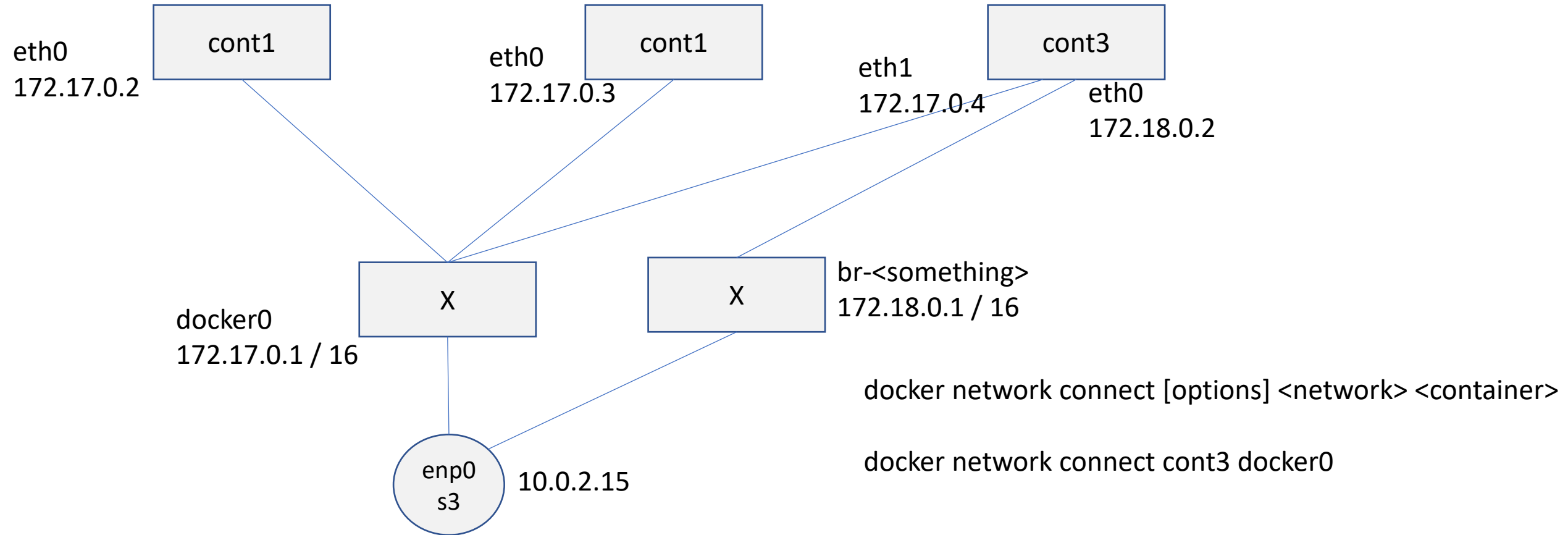
Docker Networking: docker0 bridge



Custom Networks in Docker

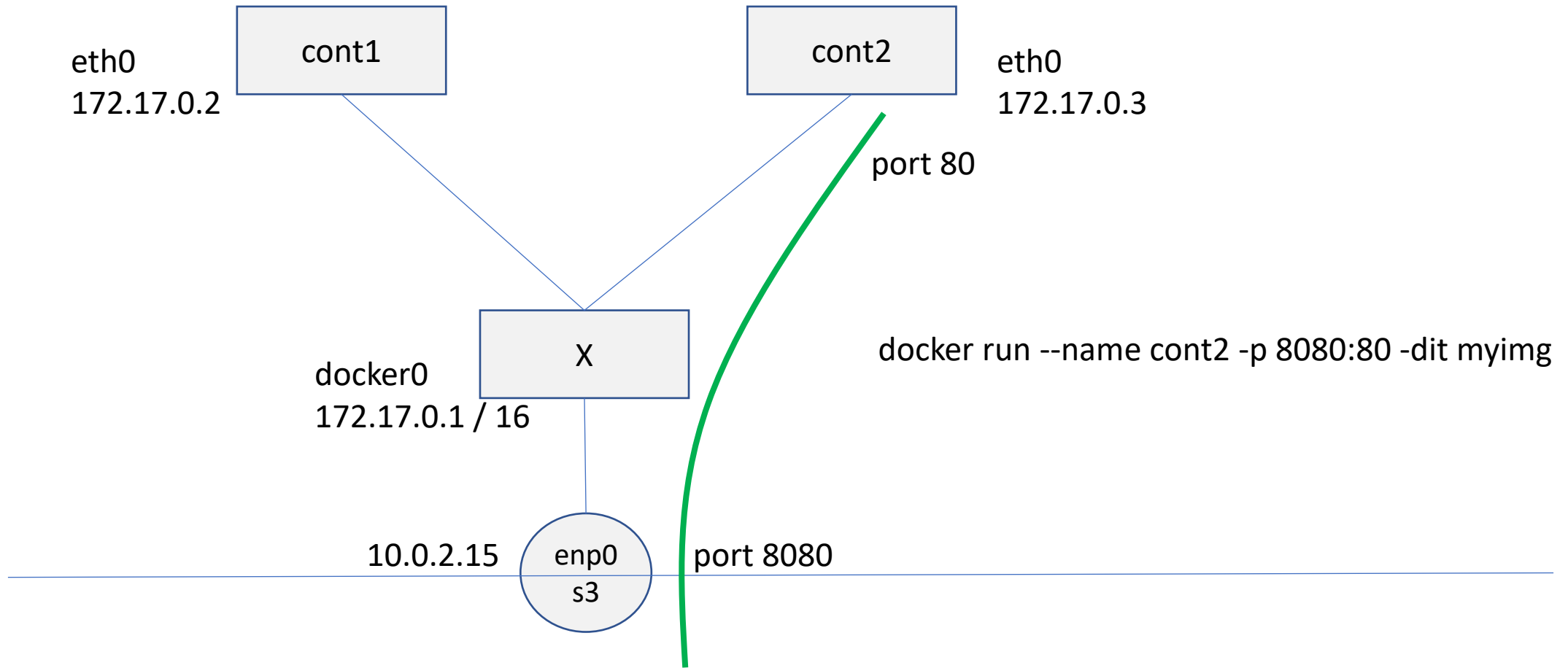


Attaching to Multiple Networks



Port Forwarding

`docker run -p <host_port>:<container_port> <image>`



Docker compose wrapper

What is it:

- YAML wrapper for docker commands (build, pull, run...)
- Also supports multiple containers

To install: <https://docs.docker.com/compose/install/>

To use: <https://docs.docker.com/compose/reference/>

`docker-compose --help` (build, run, ... - a lot of the same commands)

Example

docker-compose.yml

docker compose up

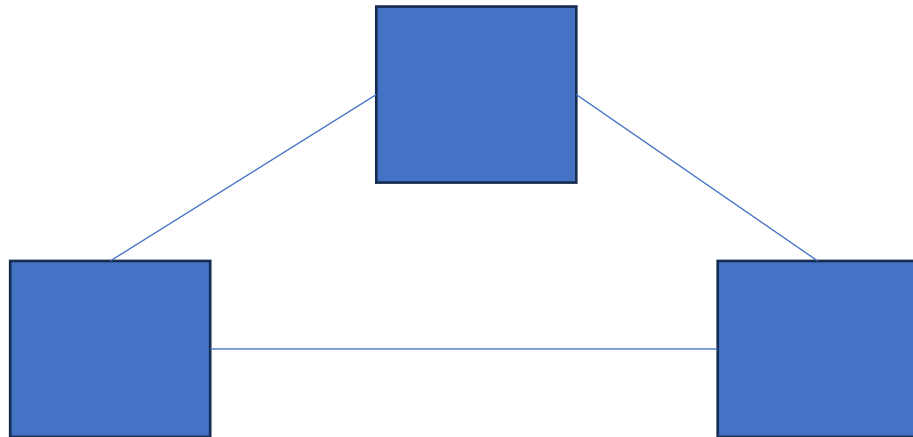
```
services:
  proxy:
    build: ./proxy
    networks:
      - frontend
  app:
    build: ./app
    networks:
      - frontend
      - backend
  db:
    image: postgres
    networks:
      - backend

networks:
  frontend:
    # Use a custom driver
    driver: custom-driver-1
  backend:
    # Use a custom driver which takes special options
    driver: custom-driver-2
    driver_opts:
      foo: "1"
      bar: "2"
```

<https://docs.docker.com/compose/networking/>

Practice Exercise

- Create a containerlab topology configuration for the following topology
- Inspect the network that was created
- Attempt to recreate it manually





University of Colorado **Boulder**