YASMIN

(*Yet Another Shared Memory for Intra-Node framework*)

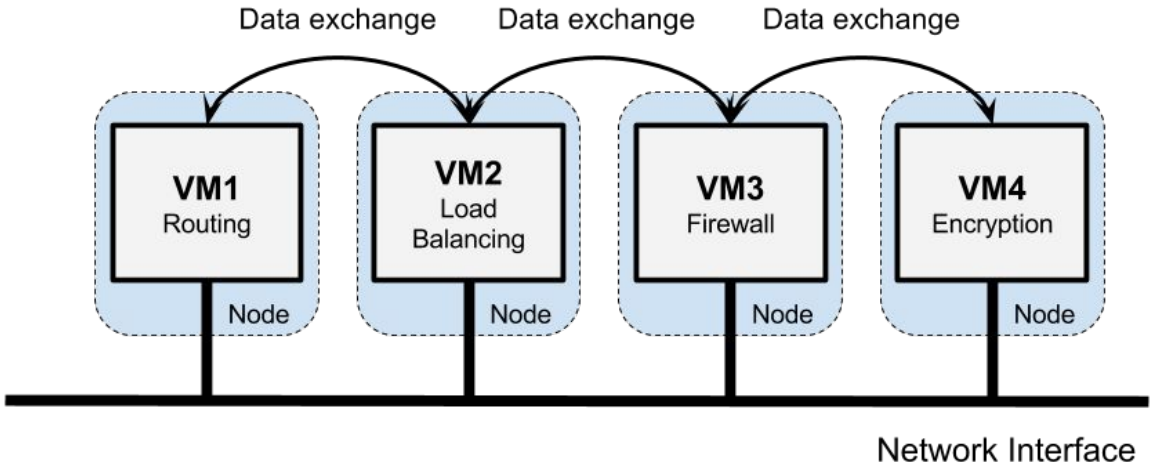
Efficient Intra-Node Communication Using Generic Sockets

Michalis Rozis, Stefanos Gerangelos, and Nectarios Koziris
National Technical University of Athens
Computing Systems, Laboratory

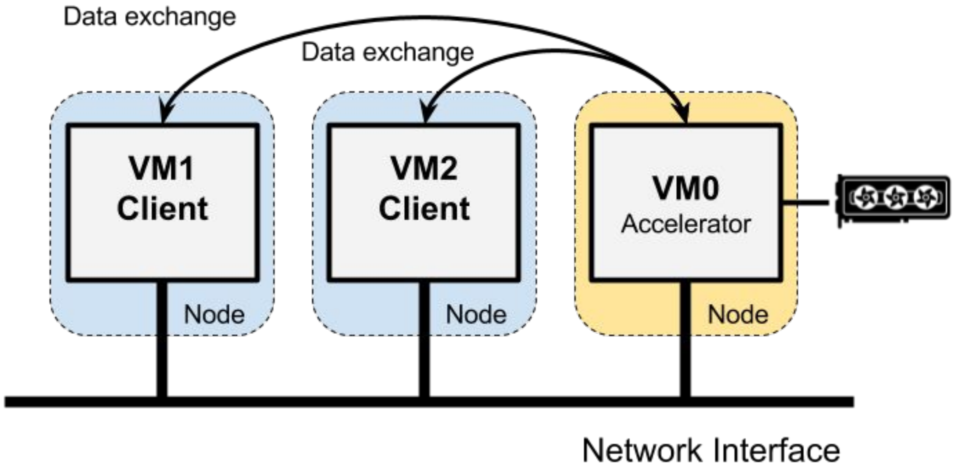
What’s the problem?

- VMs exchange large amount of data. For example:
  - MapReduce workloads running in many VMs.
  - HPC applications
  - Network Function Virtualisation (NFV) such as routing, encryption, load balancing etc.
  - VMs access shared device (accelerators, crypto devices etc)
- VMs can be placed in different nodes
  - Data path through network
- VMs can be placed in the same node
  - Data path through memory
  - Memory speed > Network

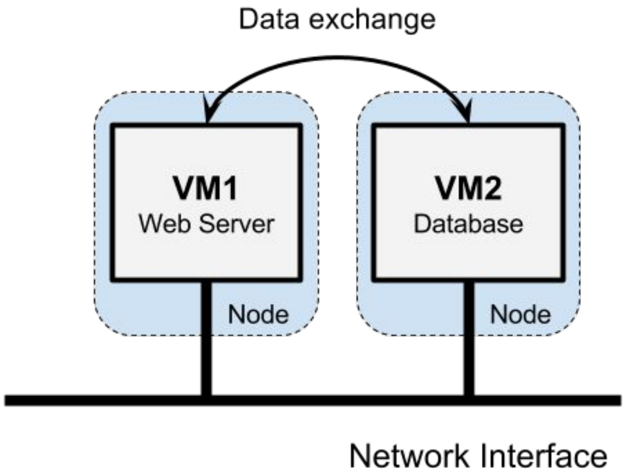
Example: NFV


Data exchange
Data exchange
Data exchange
VM1
Routing
Node
VM2
Load Balancing
Node
VM3
Firewall
Node
VM4
Encryption
Node
Network Interface

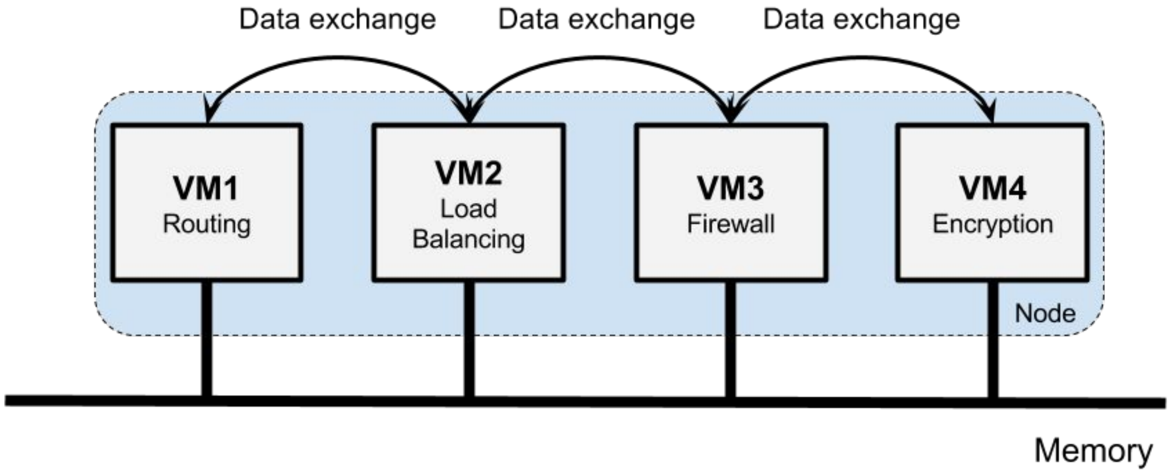
Example: Shared GPU



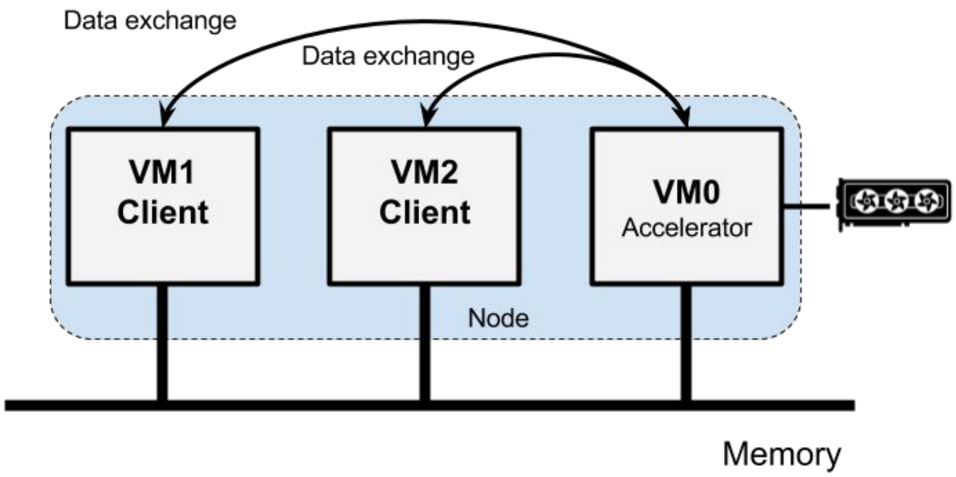
Example: Web Server



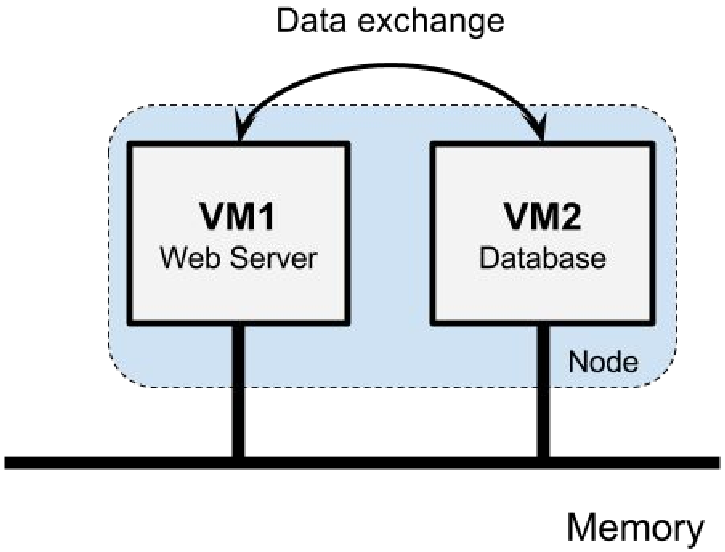
Example: Co-located NFV VMs



Example: Co-located VMs - shared GPU



Example: Co-located VMs - Web Server



Efficient Intra-node Communication

- Place VMs in the same node

- Make communication efficient (latency, throughput).

- Applications need to be unaware of execution environment.

Build on the Xen hypervisor

- Widely used
- Lightweight type-1
- Paravirtualized Guests
- Exploit page sharing techniques

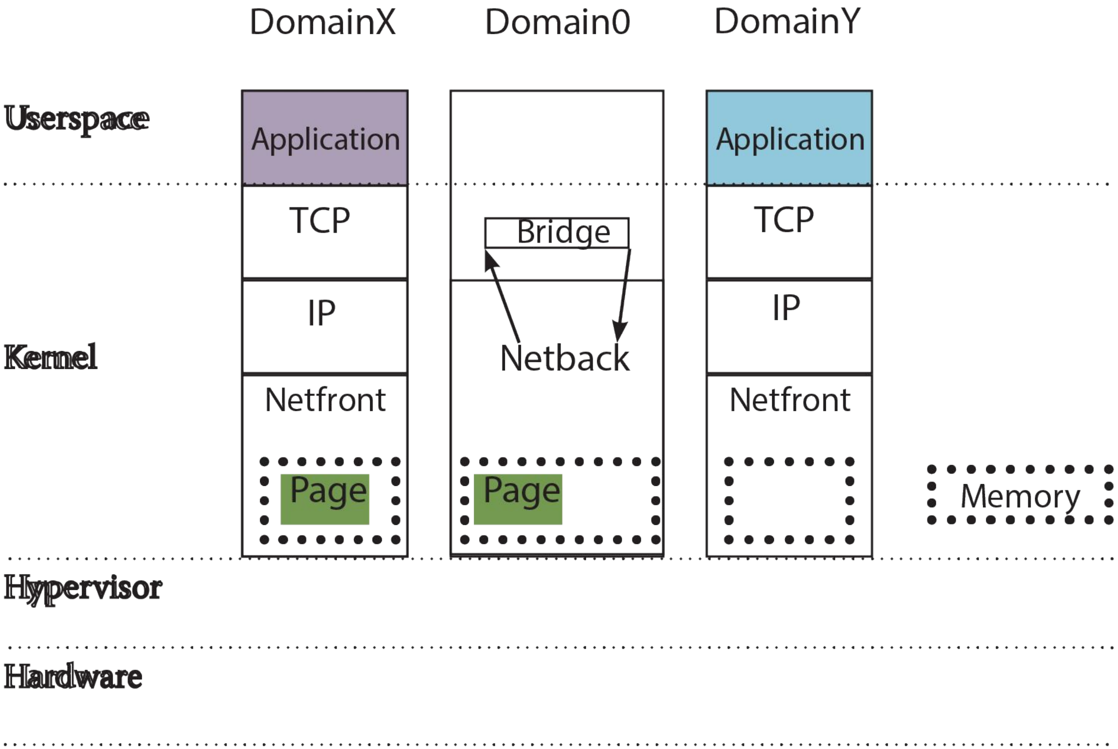
Xen features

- Grant table → map and share pages
  - Granter (Dom1):
    - Allocates page
    - Grants page access to foreign domain (Dom2)
    - Returns grant-table index
  - Grantee (Dom2):
    - Allocates page
    - Maps page using grant-table index

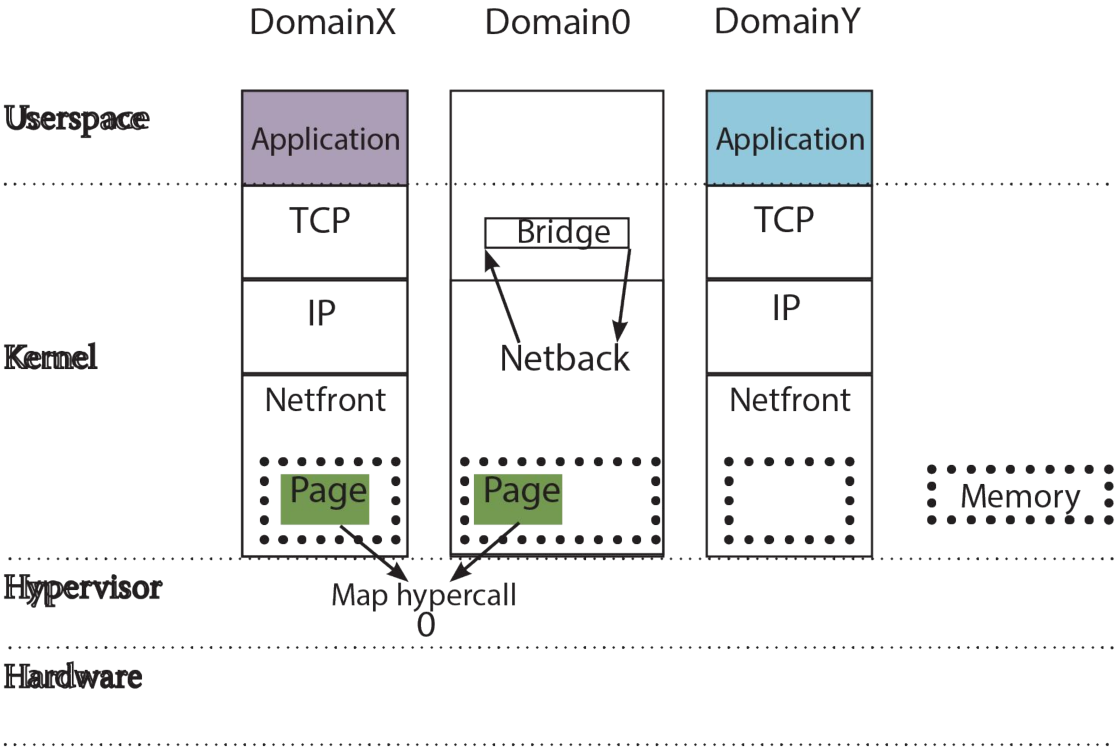
Xen features

- Event channel → Virtual interrupt mechanism
    - Dom1:
        - New channel with foreign domain
        - Returns local channel port
        - Registers interrupt handler
    - Dom2:
        - Registers channel using Dom1's local channel port
        - Registers interrupt handler
    - Invoke interrupts between VMs.

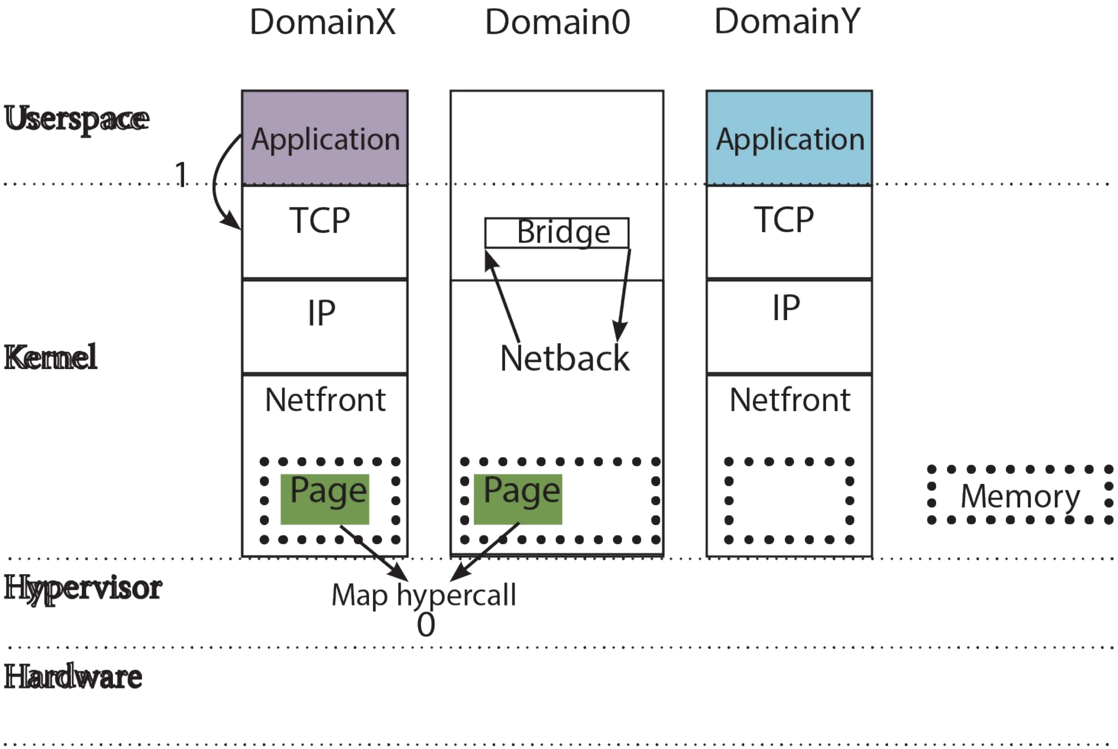
Xen netback / netfront



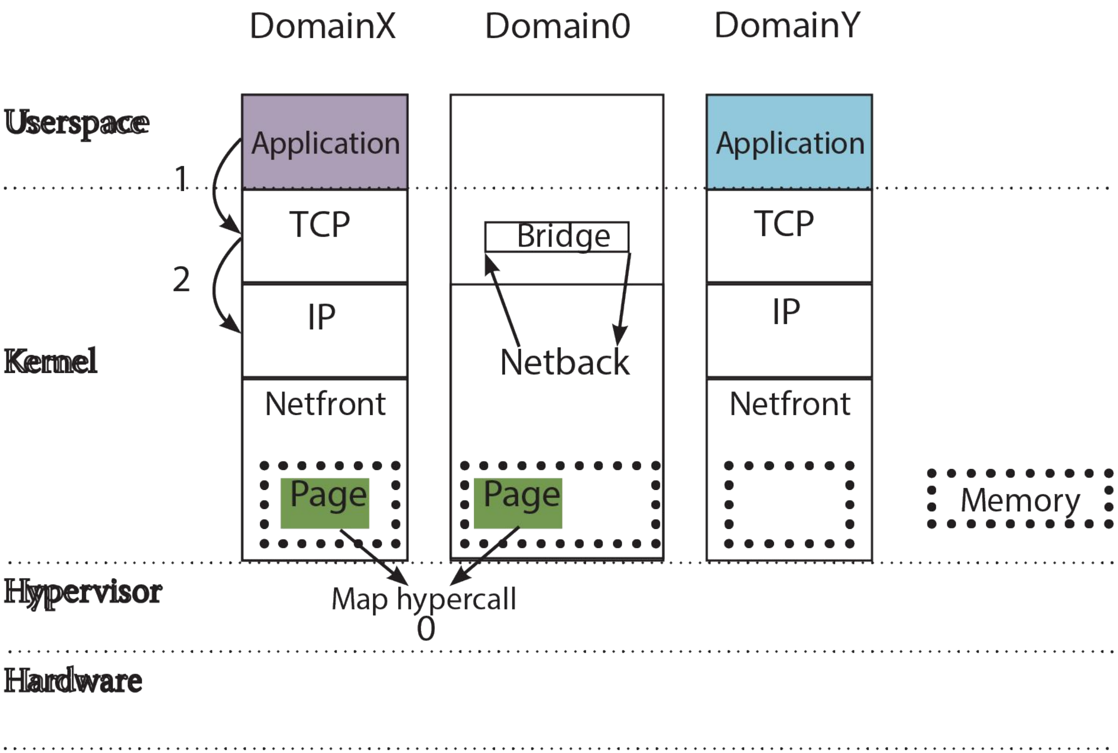
Xen netback / netfront



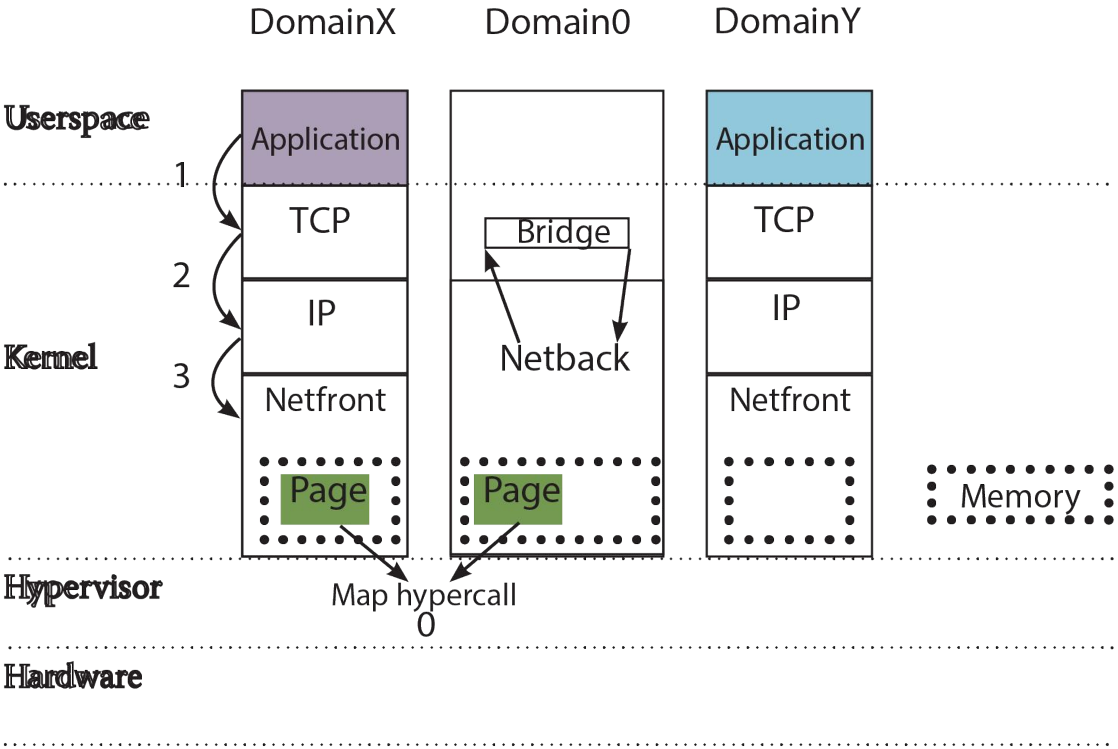
Xen netback / netfront



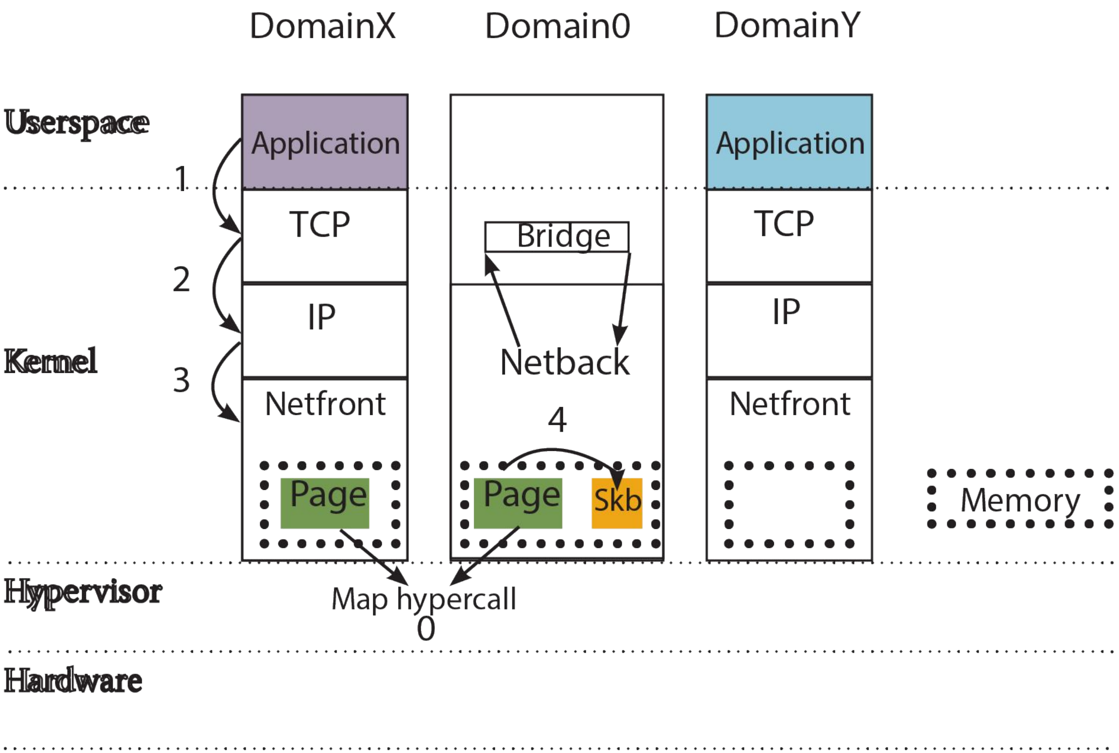
Xen netback / netfront



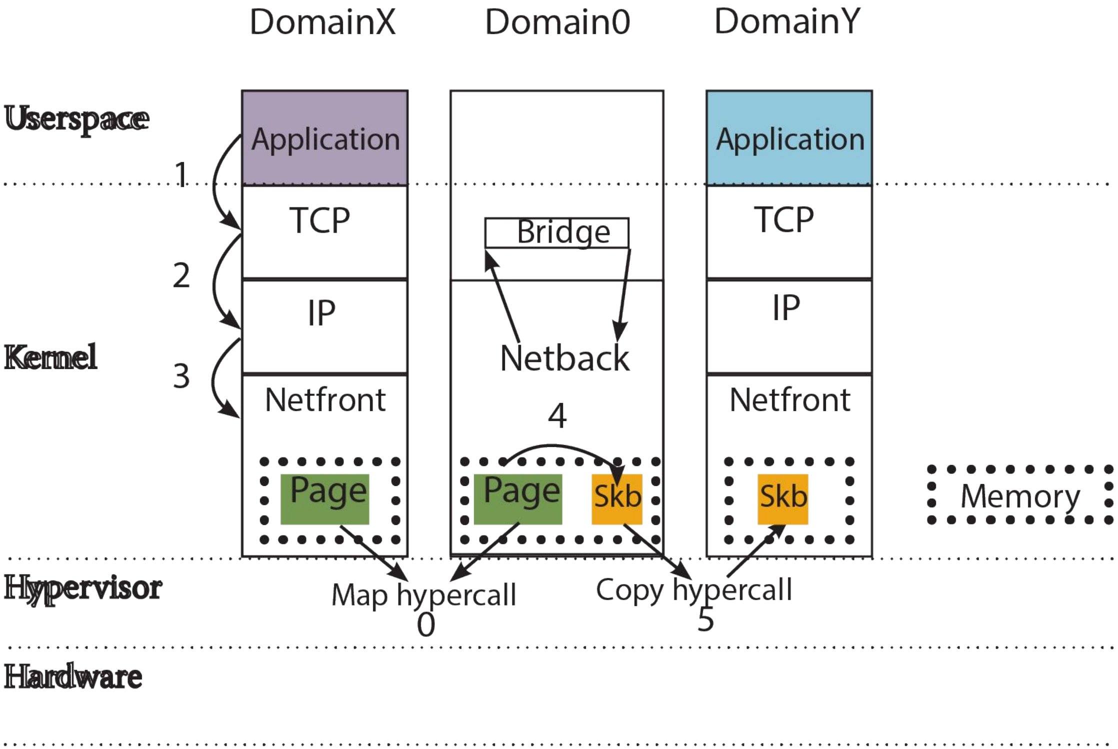
Xen netback / netfront



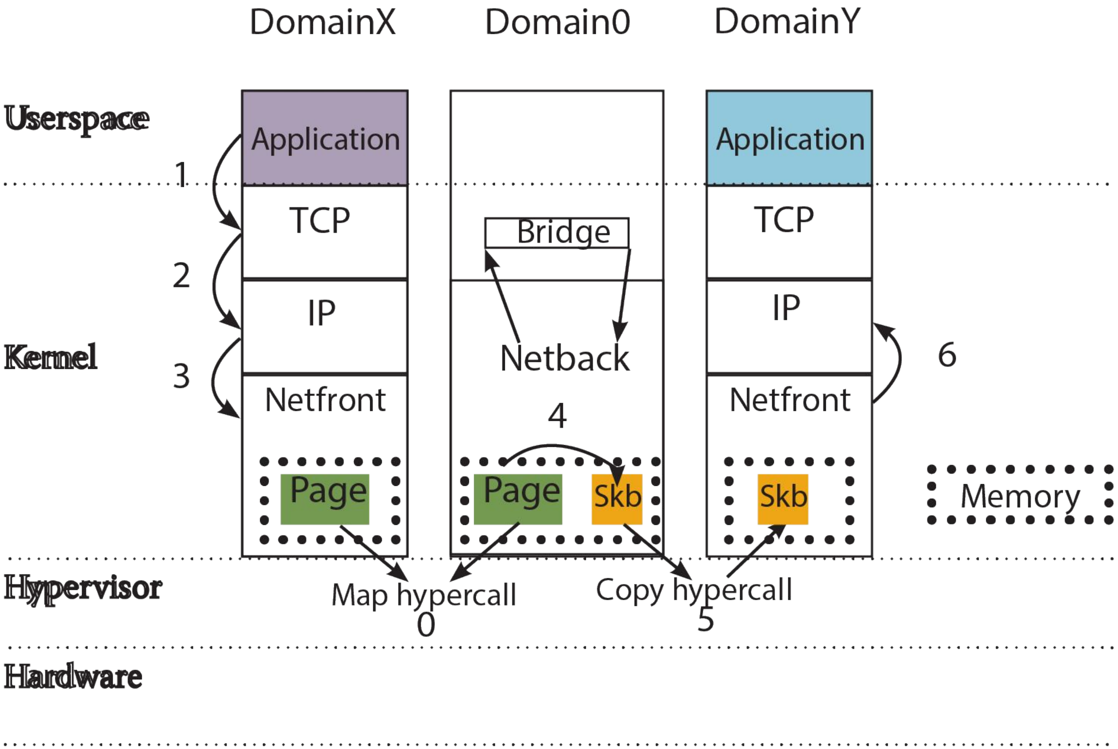
Xen netback / netfront



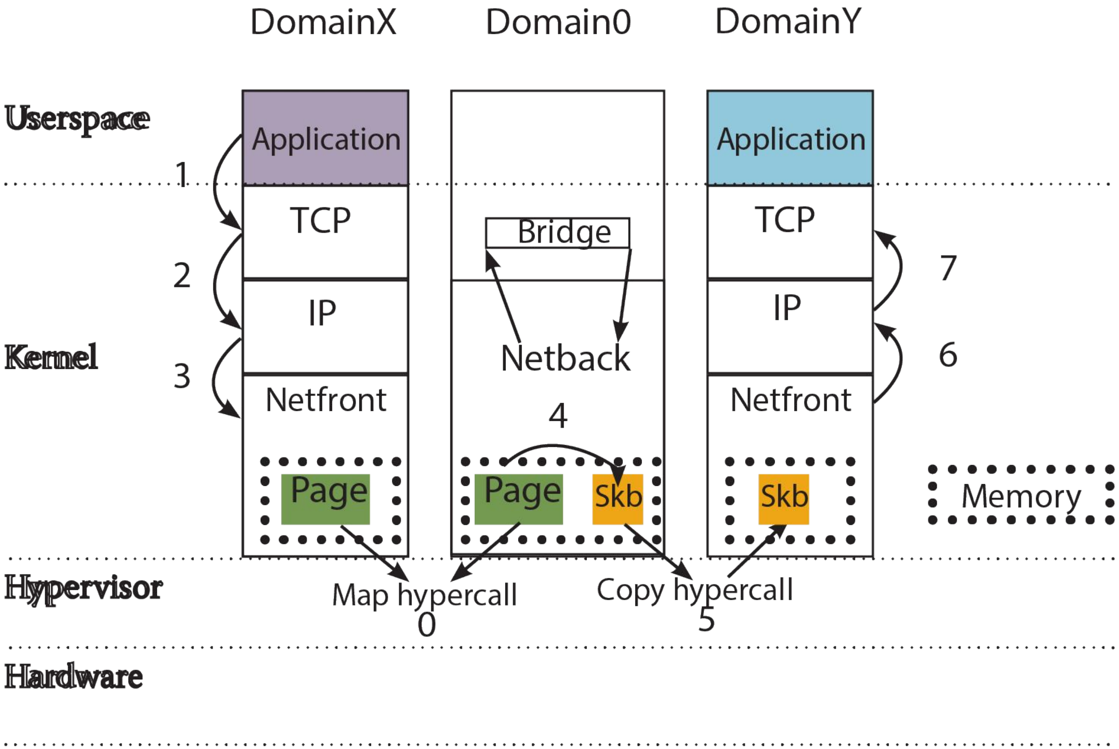
Xen netback / netfront



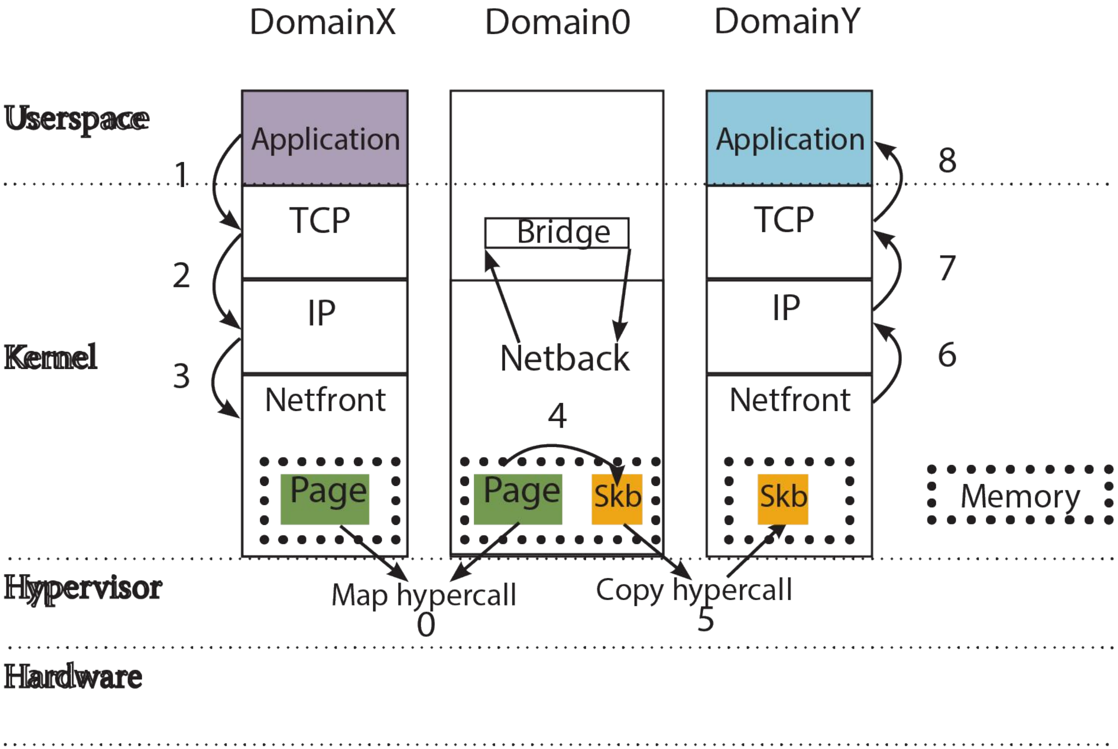
Xen netback / netfront



Xen netback / netfront



Xen netback / netfront



Standard intra-node communication:

- Slow: Copies between VMs and control domain
- Non - scalable:
  - Different applications between one pair: same interface
  - Different pair of VMs: all traffic through control domain.

✓Leave Control Domain alone.

How?

Idea:

- Page Sharing between connecting VMs
- Endpoints responsible for setup and breakdown of channel.
- Sockets
- Packet send/receive through network → copies to/from memory

Related work

- Concept of page sharing with Sockets connections
    - New Socket API
    - Message copying by hypervisor
    - Packet capture and process (netfilter)

  Plenty of proposals use page sharing but...

- Hypervisor modification
- Code refactoring
- Recompilation
- TCP/IP

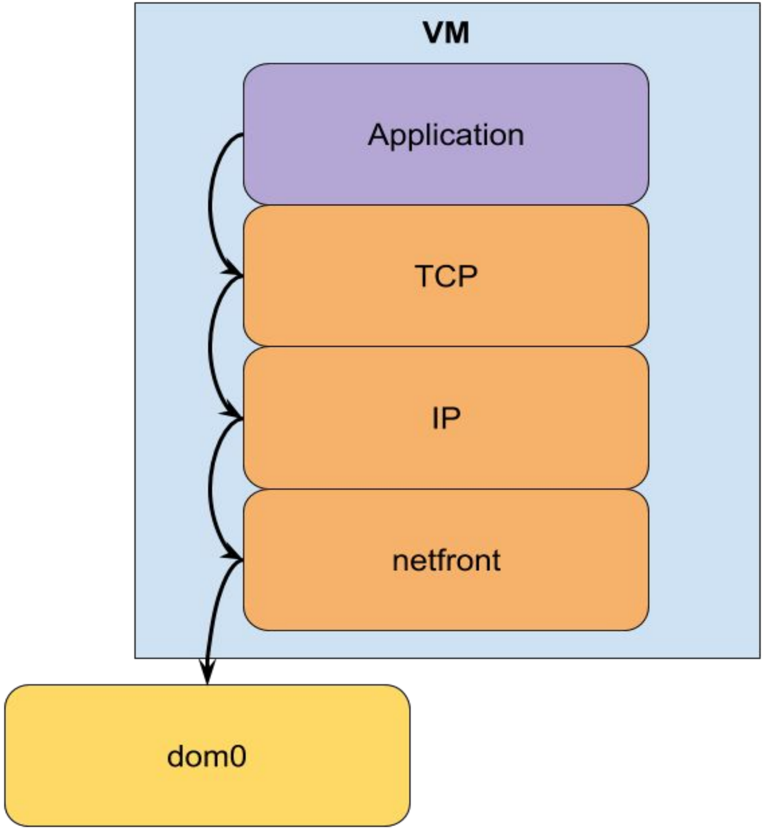
Compatibility issues!

What’s new with YASMIN?

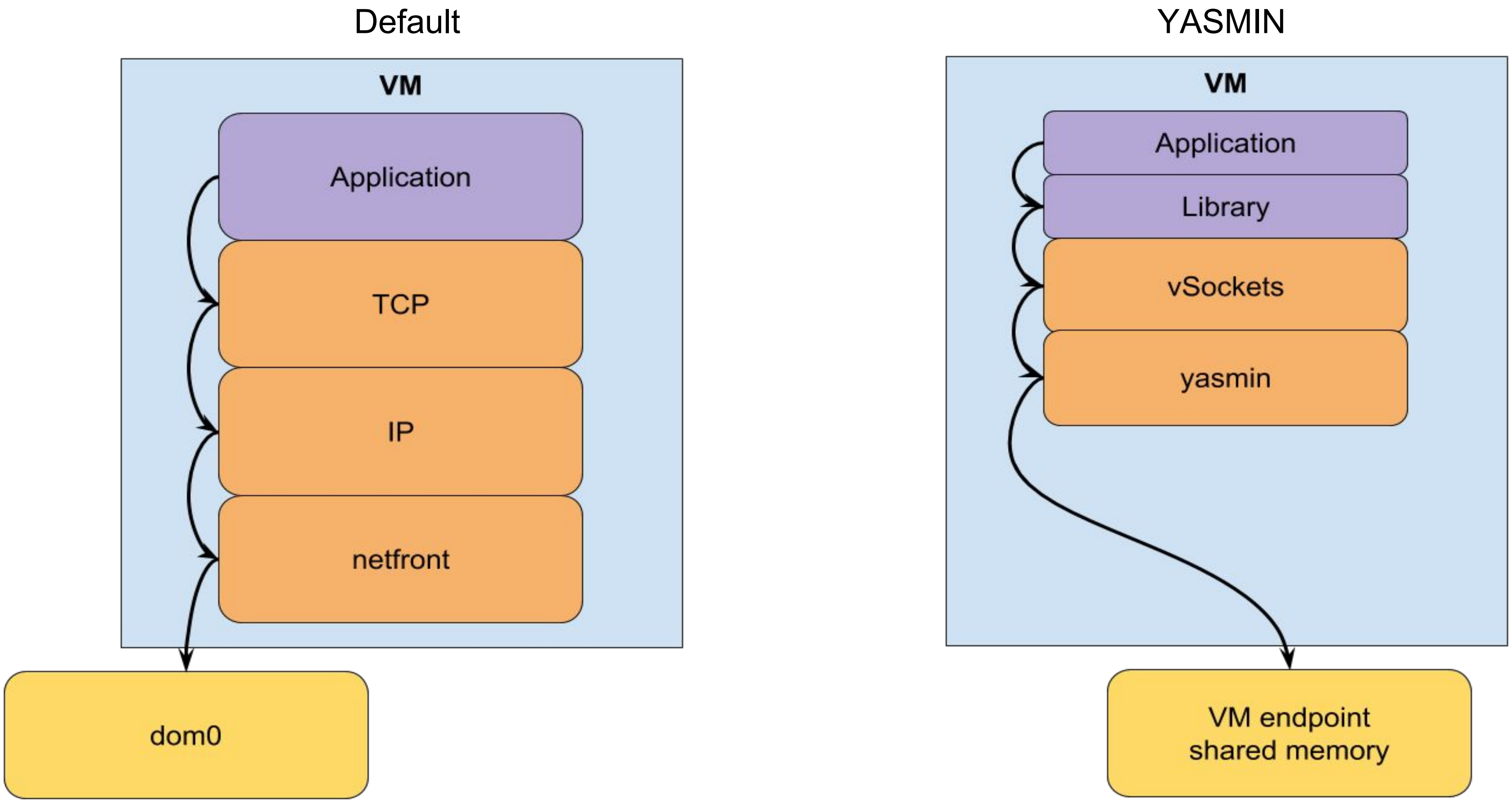
- Compatibility very important
  - Wiseman said just stick to POSIX Sockets
- Why always pay TCP/IP fee? Do we really need this? Take a shortcut!
- Build a new protocol from scratch?
  - Use vSockets (AF_VSOCK) with a brand new transport layer
  - VMWare vSockets protocol (domain ID & connection port)

Design overview

Default



Design overview

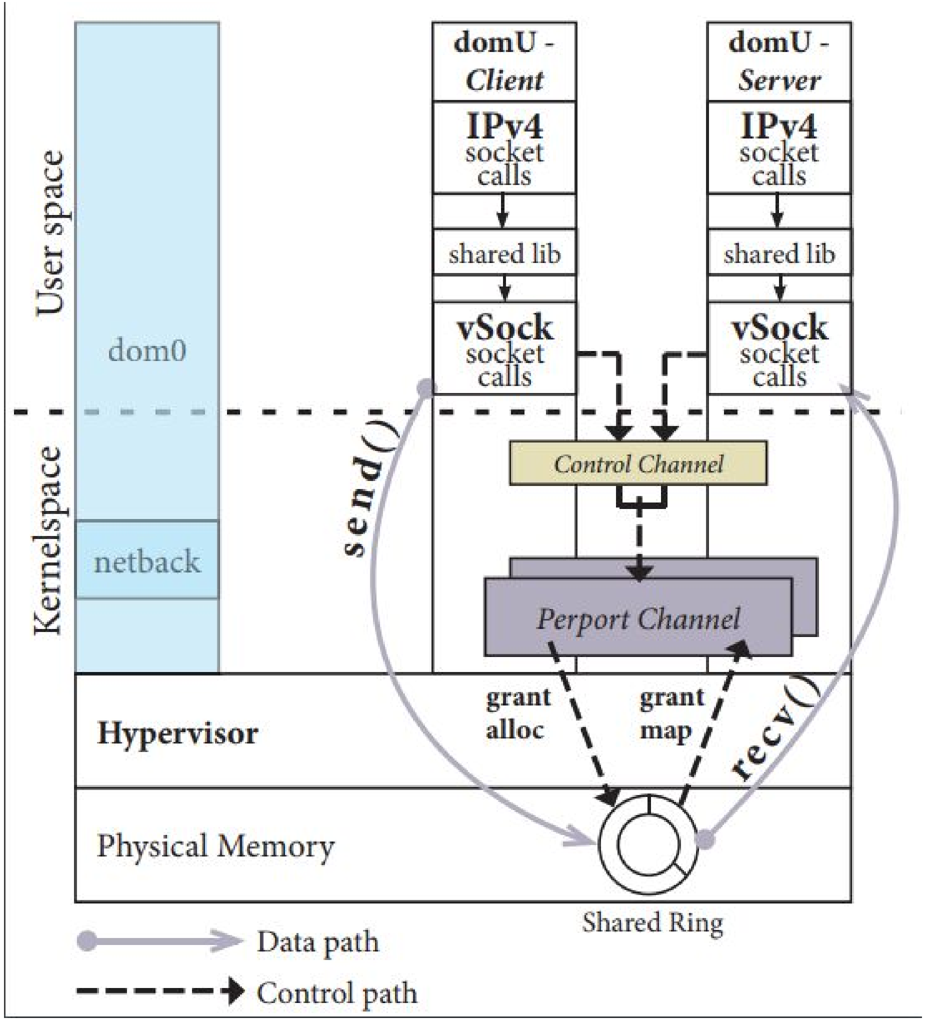


Details

- TCP/IP socket calls translated to vSockets calls (AF_VSOCK)
- *hosts* file with IPv4 - domID mappings
- One control channel connecting VM pairs
- One data channel for connecting socket pairs (*persocket* channel)

Channel setup

- Grant-table mechanism for shared pages (grant access & map)
- Event-channel for packet notification
- Producer - consumer ring (no locks) in shared memory
- copy_to/from_user() for actual data transfer


User space
Kernelspace
dom0
netback
domU - Client
IPv4 socket calls
shared lib
vSock socket calls
domU - Server
IPv4 socket calls
shared lib
vSock socket calls
Control Channel
Perport Channel
send()
recv()
grant alloc
grant map
Hypervisor
Physical Memory
Shared Ring
Data path
Control path

Evaluation

Use microbenchmarks (Iperf, NetPIPE, STREAM)

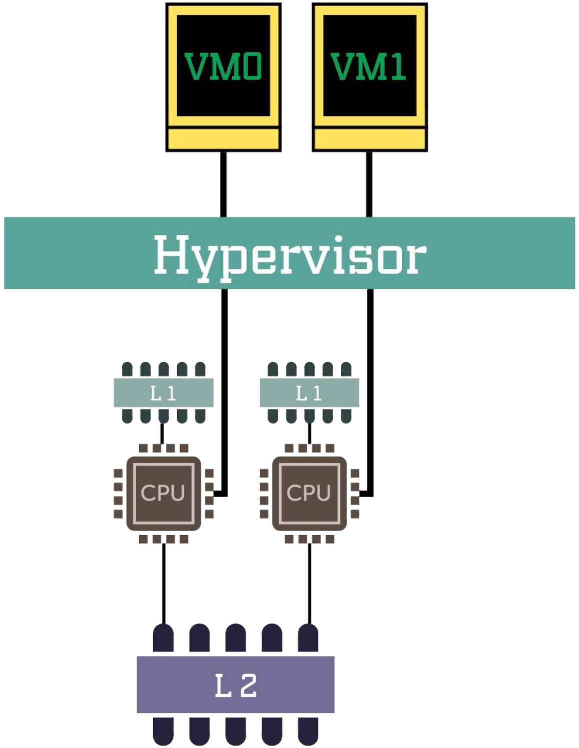
- Latency
- Bandwidth
- Scalability

Baseline

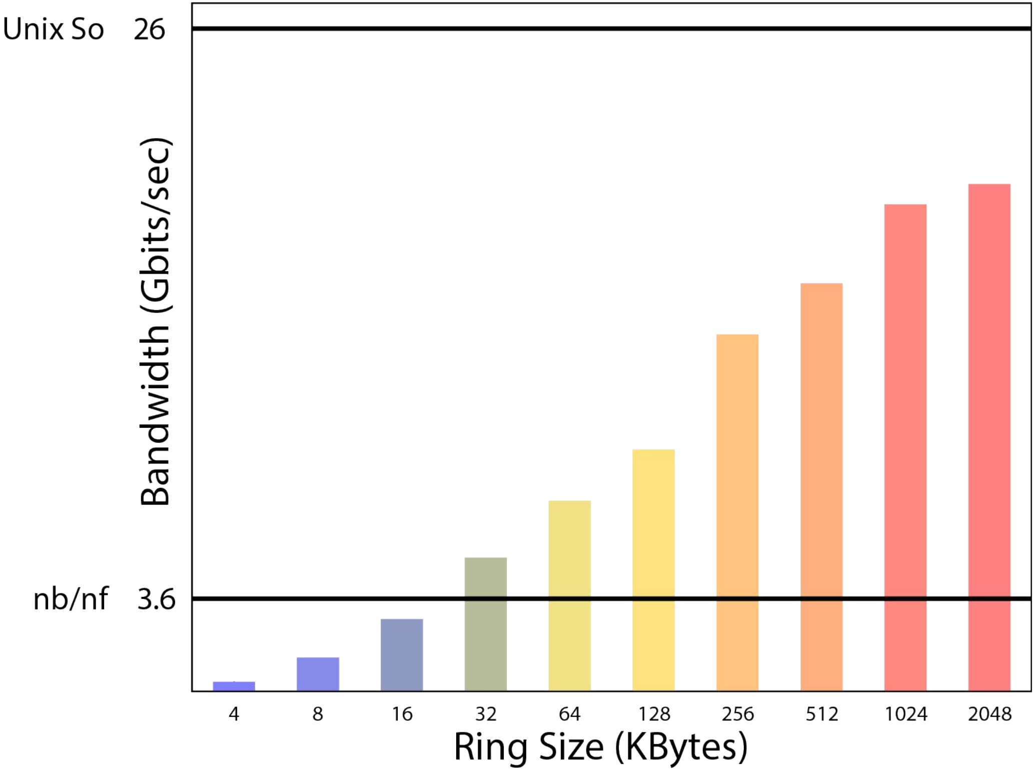
Compare to:

- netback / netfront
- Unix Domain Sockets
- Memory bandwidth

Evaluation Setup



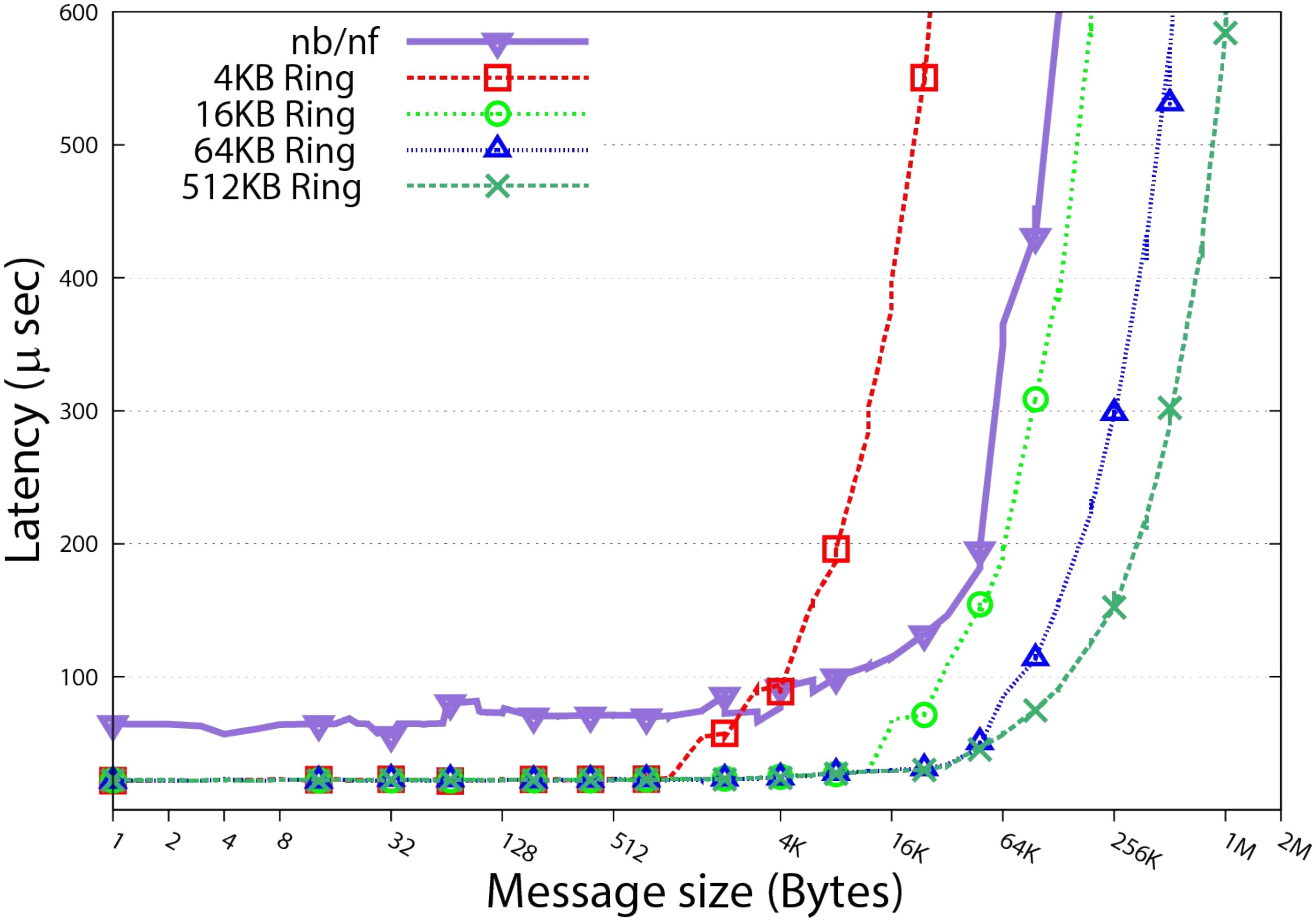
Proper Ring Size

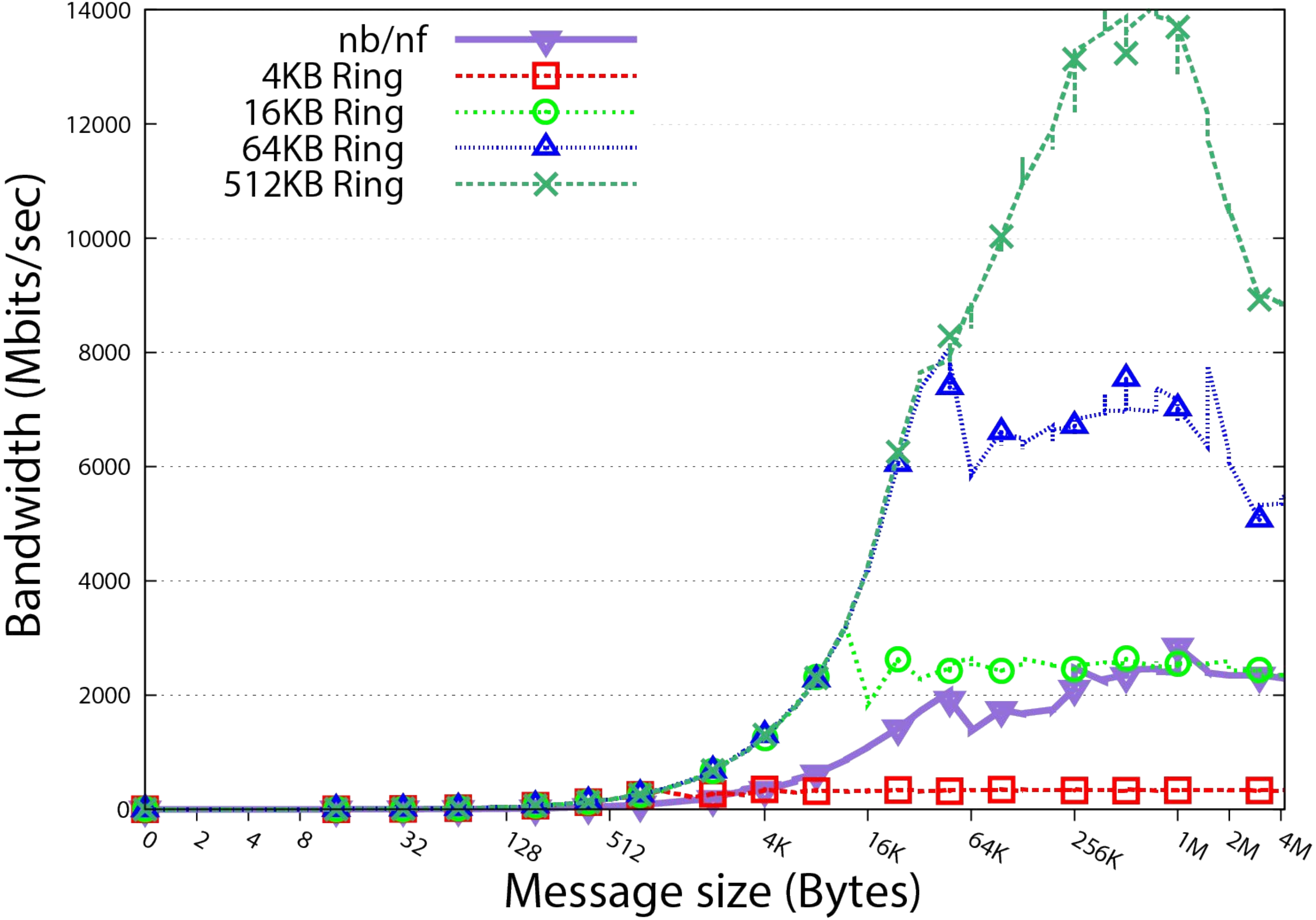


Latency - throughput

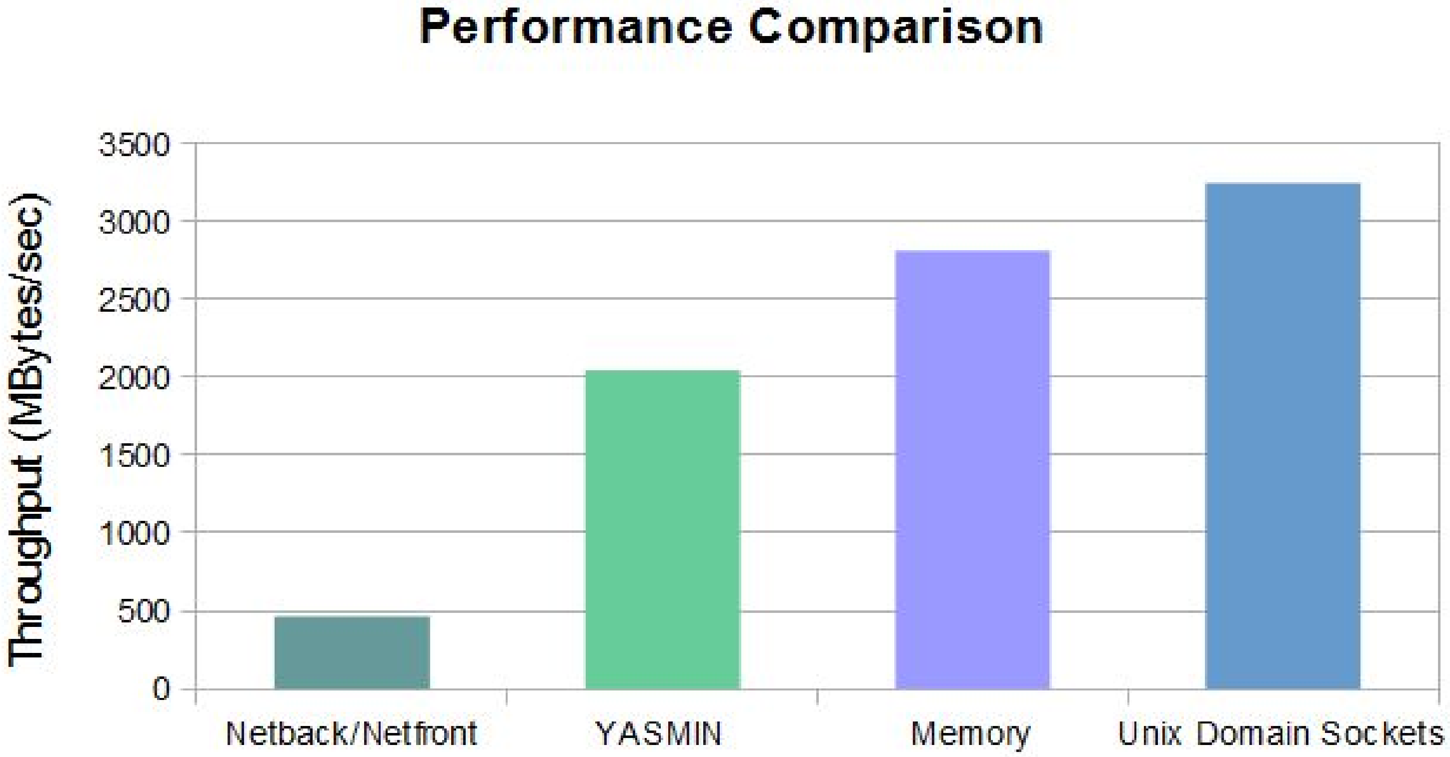
For 512kB ring size (128 pages):

- Latency reduction up to 65% (netback/netfront)
  - YASMIN @22μsec
  - netback/netfront @64μsec
- Throughput increase by 4.4x (netback/netfront)
  - netback/netfront @ 463MBytes/sec
  - YASMIN @2048MBytes/sec
  - Memory@2813MBytes/sec
  - UNIX Domain Sockets @3250MBytes/sec

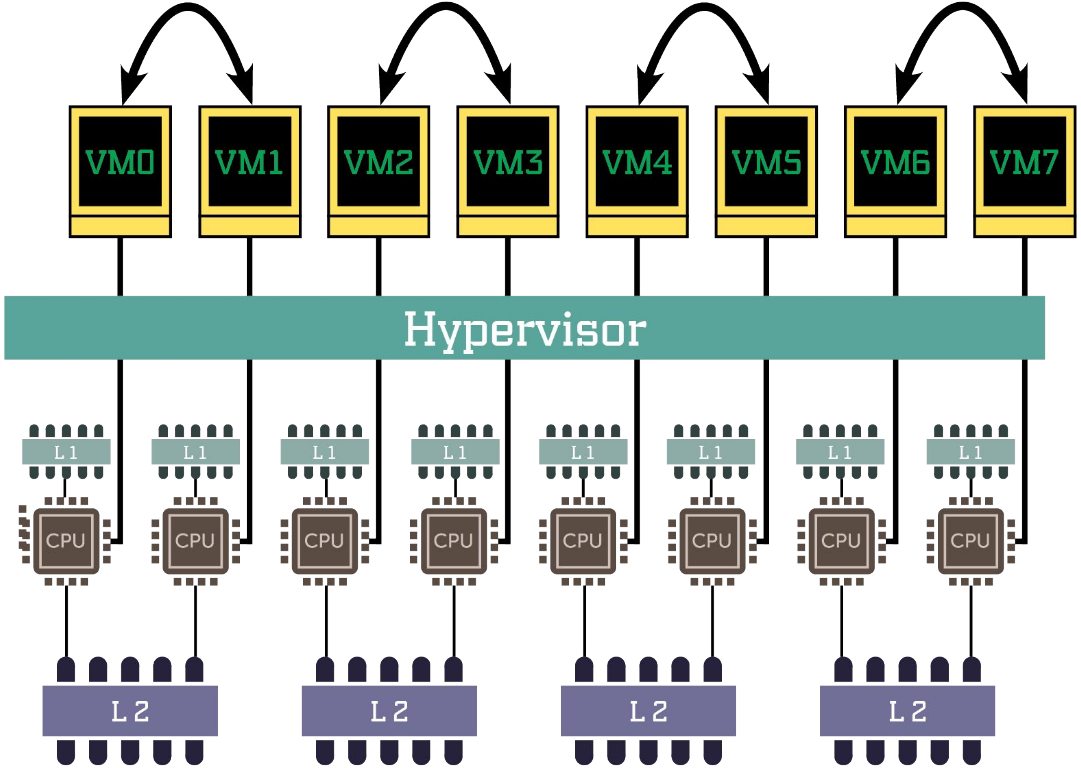

nb/nf
4KB Ring
16KB Ring
64KB Ring
512KB Ring
Latency (µ sec)
600
500
400
300
200
100
1
2
4
8
32
128
512
4K
16K
64K
256K
1M
2M
Message size (Bytes)

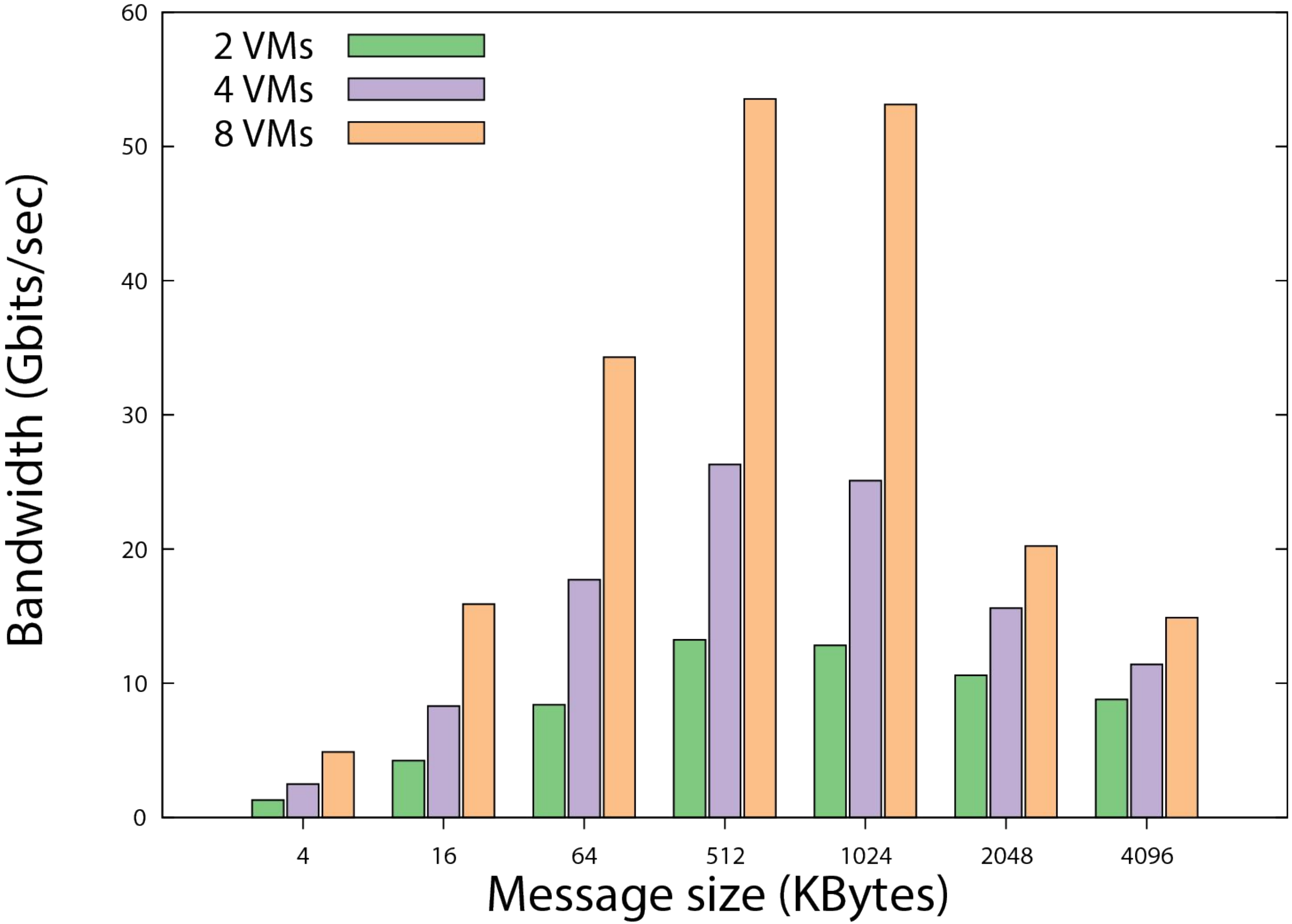


nb/nf
4KB Ring
16KB Ring
64KB Ring
512KB Ring
Bandwidth (Mbits/sec)
14000
12000
10000
8000
6000
4000
2000
0
Message size (Bytes)
0
2
4
8
32
128
512
4K
16K
64K
256K
1M
2M
4M


Performance Comparison
Throughput (MBytes/sec)
0
500
1000
1500
2000
2500
3000
3500
Netback/Netfront
YASMIN
Memory
Unix Domain Sockets

Scalability setup:




2 VMs
4 VMs
8 VMs
Bandwidth (Gbits/sec)
60
50
40
30
20
10
0
4
16
64
512
1024
2048
4096
Message size (KBytes)

Conclusion

- VMs exchange data → Run in same physical node
- POSIX Sockets - Retain compatibility
  - No refactoring
  - No recompiling
  - No hypervisor modification
- Choose transport layer on the fly
- Exploit shared memory
- Bypass TCP/IP
- Latency - Throughput - Scalability

Future work

- Test on real-life applications (NFV infrastructure)
- Validate caching effects
- Enhance hosts file mechanism