A Case for Scaling Applications to Many-core with OS Clustering

(reviewed by Oleg Iegorov and Jander Nascimento)

University Joseph Fourier

December 11, 2011

Roadmap

Process management

Sharing file

The prototype

Evaluation

Our evaluation

Scaling Operating Systems For Many Cores

Nowaday's tendencies:

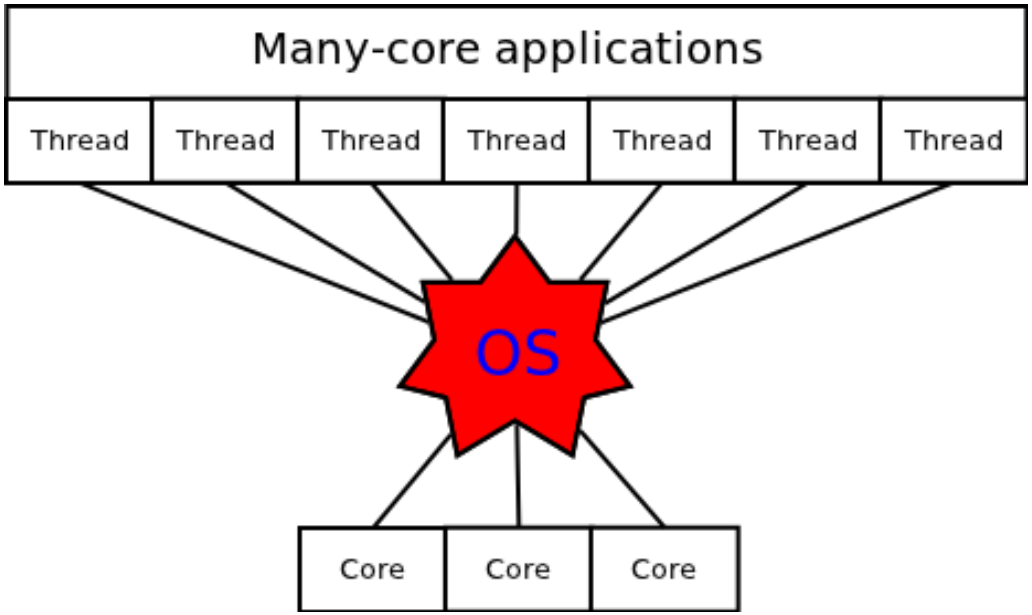
- the number of CPU cores in a system increases;
- parallel applications use fine-grained components;

Scaling Operating Systems For Many Cores

Nowaday's tendencies:

- the number of CPU cores in a system increases;
- parallel applications use fine-grained components;

OS becomes a bottleneck!



Scaling Operating Systems For Many Cores

Two main approaches:

1. designing new OSes
   - distribute independent kernels on multiple cores;
   - reduce resource sharing & improve data locality;
2. refining commodity kernels
   - design new data structures to reduce the contention on shared data structures;
   - *interesting fact:* Linux kernel can efficiently scale to 48 cores!

OS clustering approach

New approach proposed by this paper: **OS clustering**

- cluster multiple OSes atop a VMM;
- several OSes can serve one application;

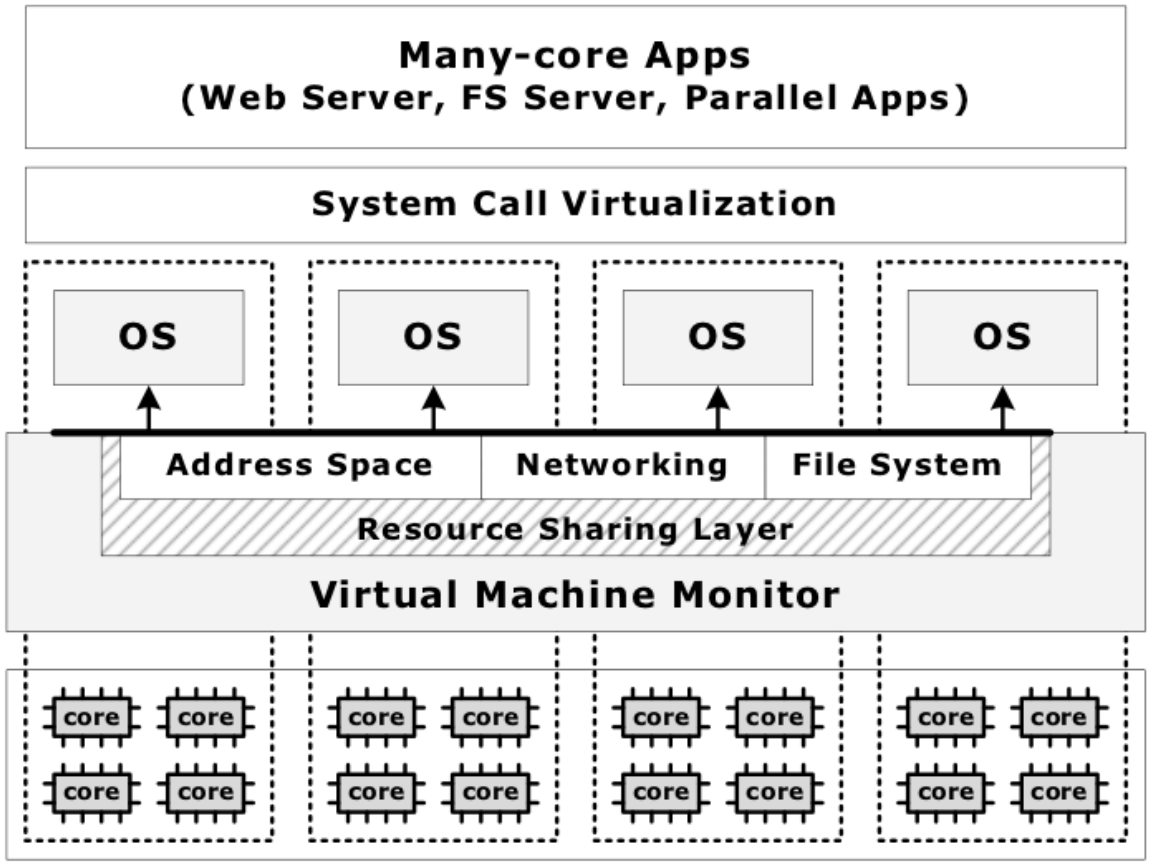
Motivation

1. commodity kernels scale well with small # of CPU cores;
2. one VMM can efficiently consolidate multiple OSes;

OS clustering approach

New approach proposed by this paper: **OS clustering**

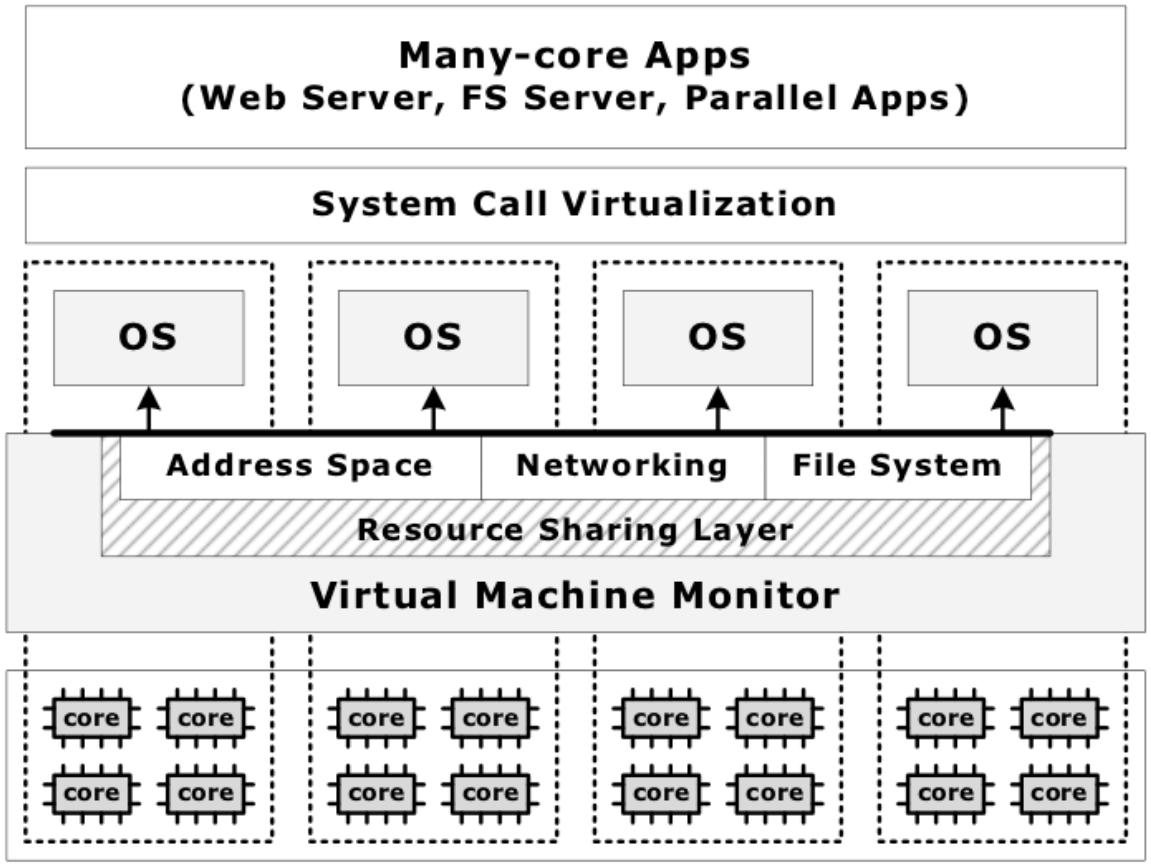
- cluster multiple OSes atop a VMM;
- several OSes can serve one application;



OS clustering approach

Results

- single OS manages fewer cores;
- resource contention is mitigated.

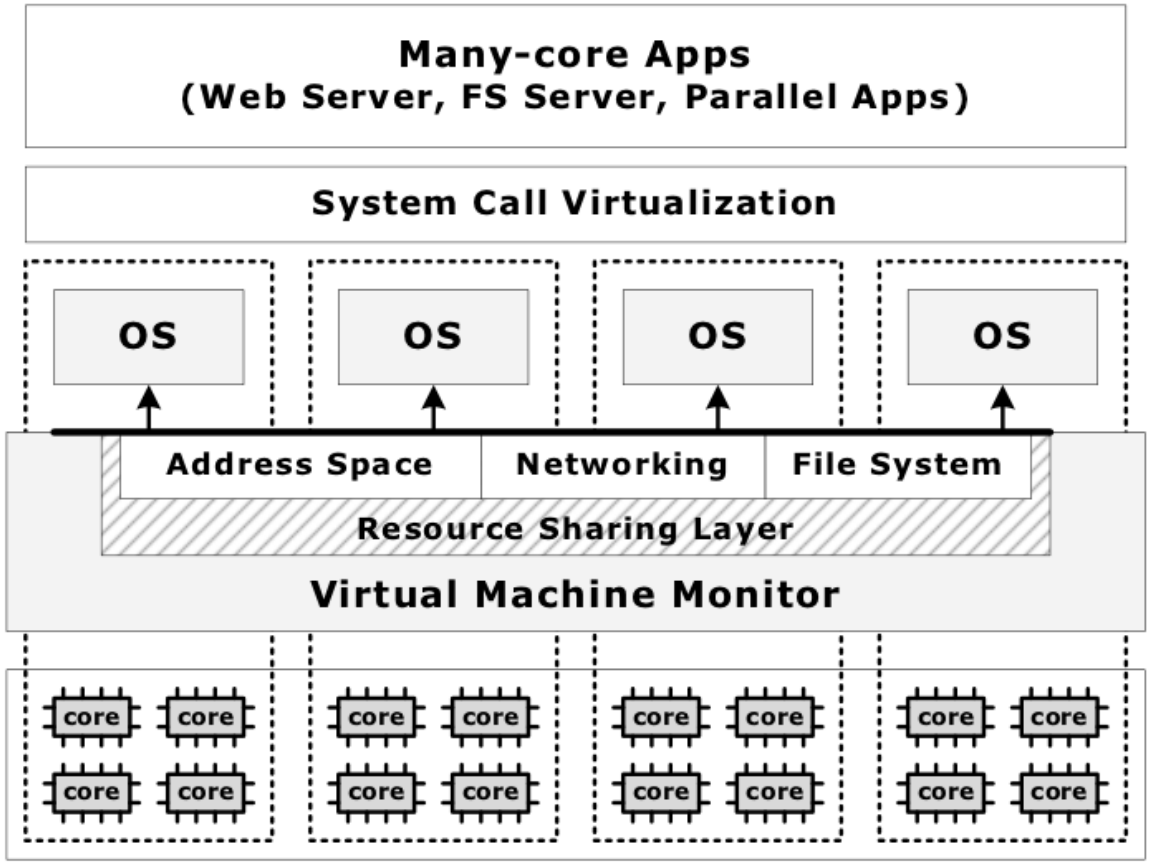


What's next?

The implemented system was called **Cerberus**

- overview of Cerberus;
- more implementation details;
- evaluation results

Overview of Cerberus architecture



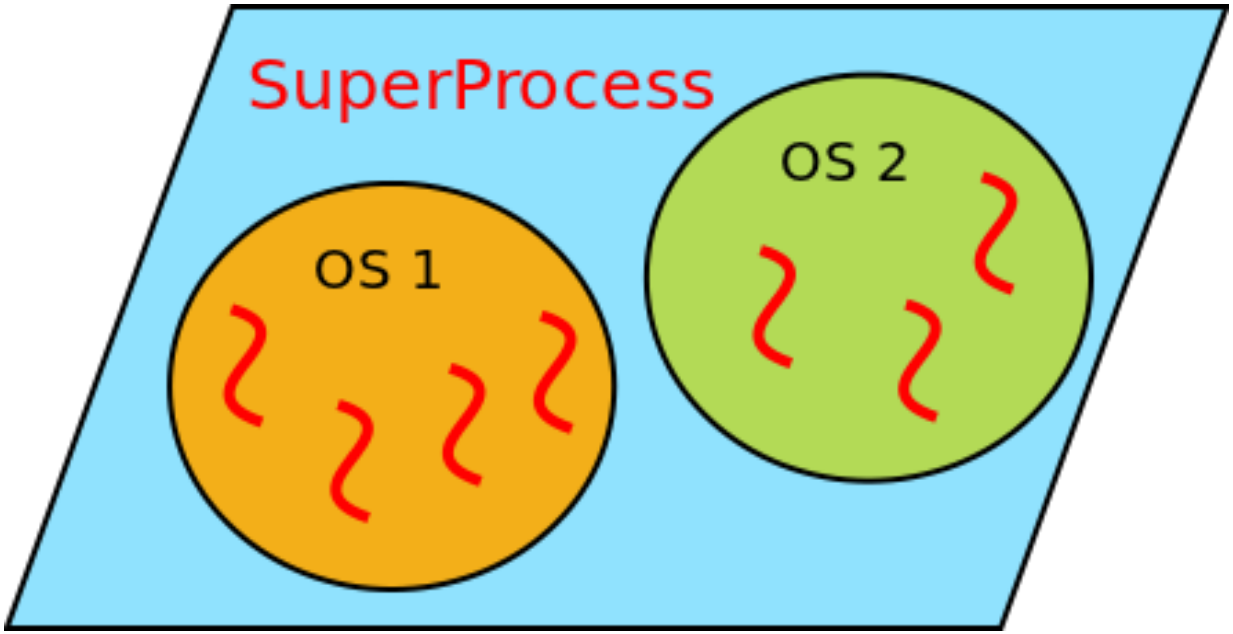
System call virtualization

Why it is needed?

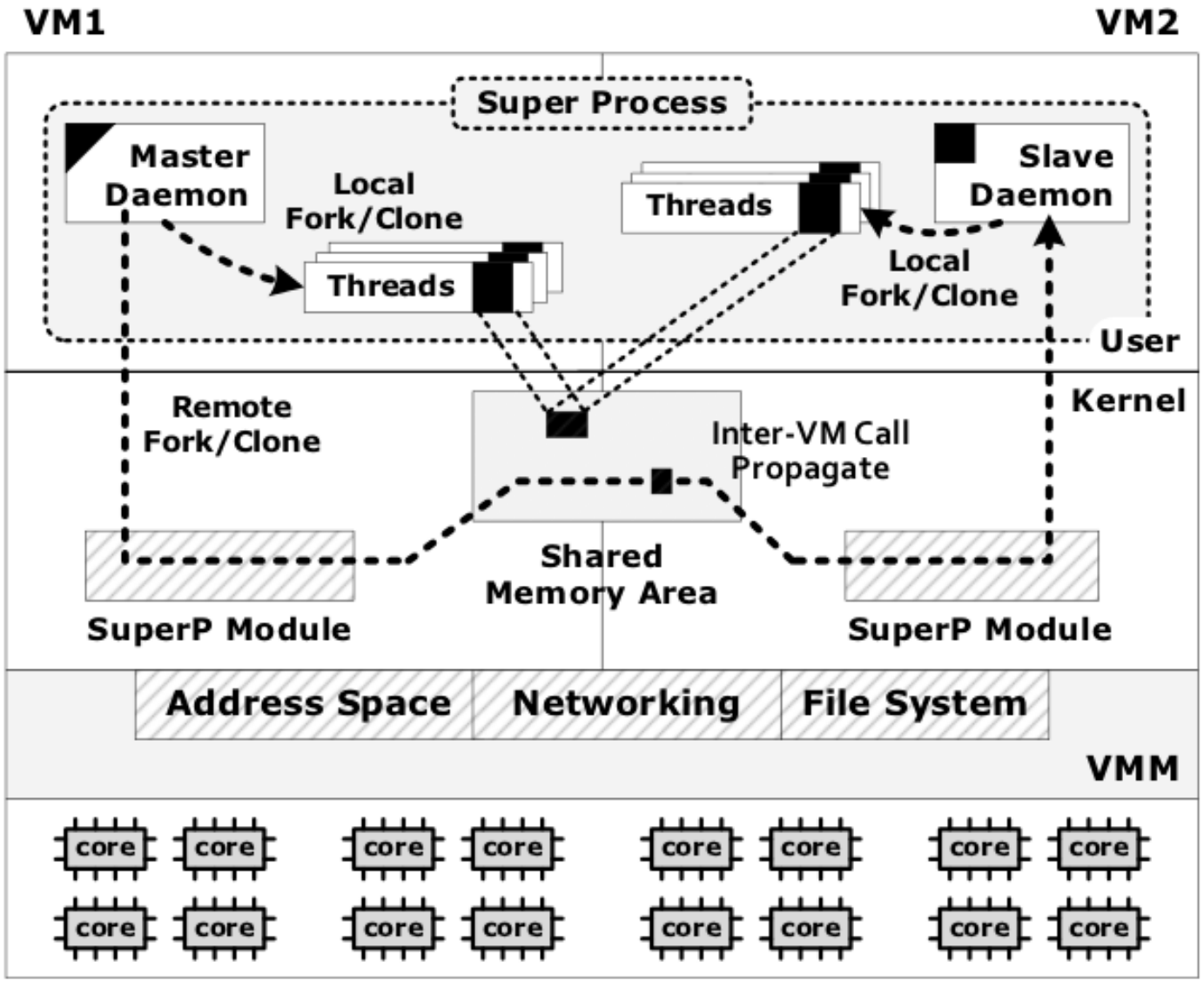
- all syscalls should be intercepted;
- allow processes/threads of one application to run on several OSes;
- transparent to the user (POSIX API)

System call virtualization

The notion of **SuperProcess**

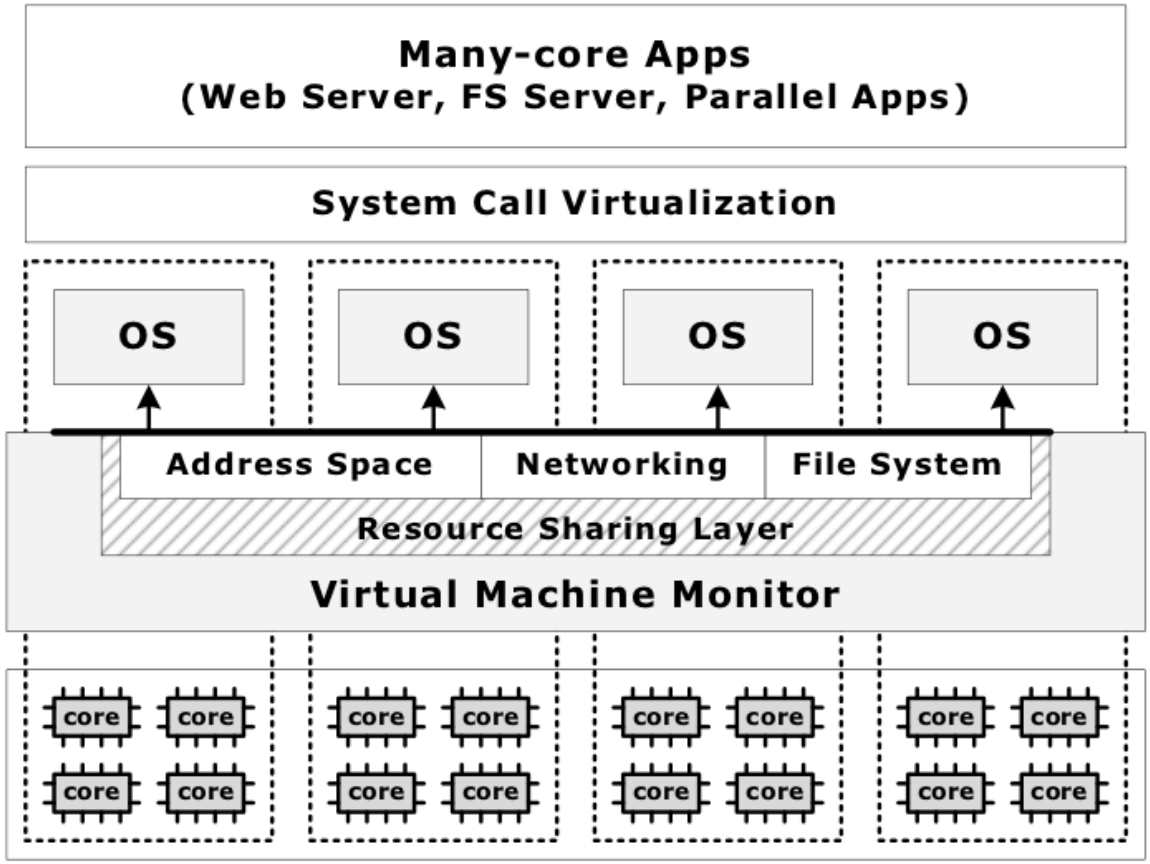


System call virtualization

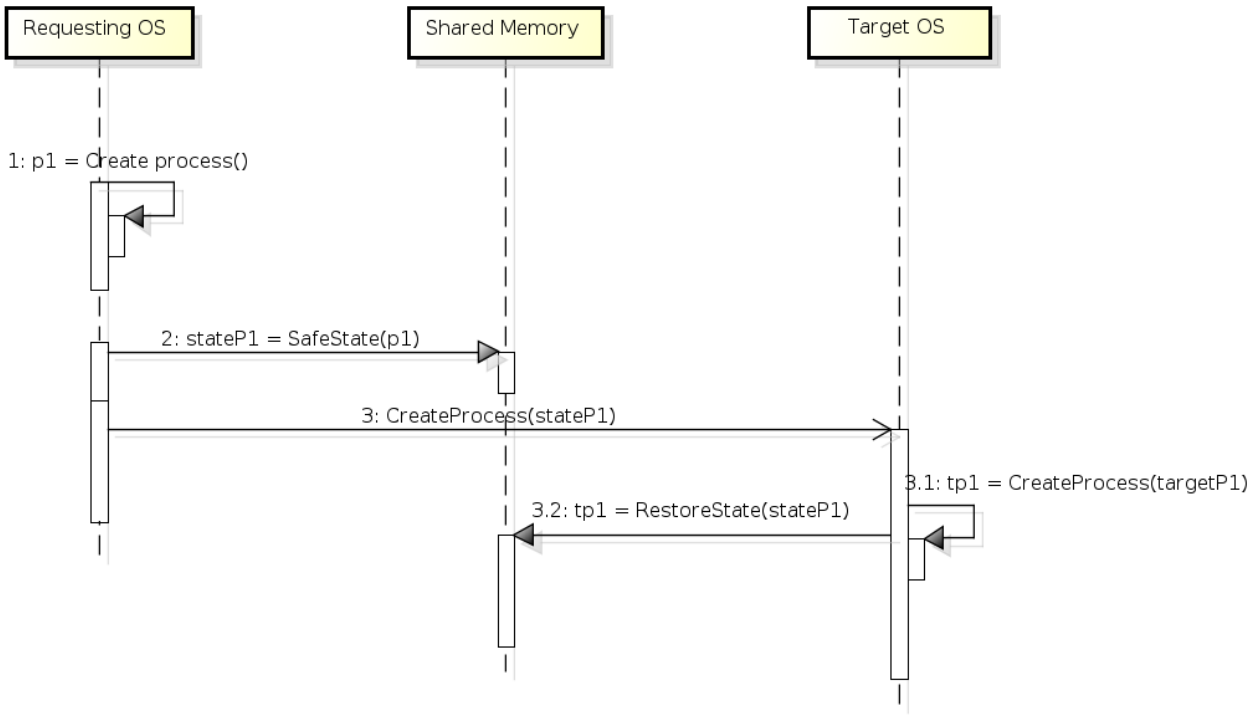


Resource Sharing

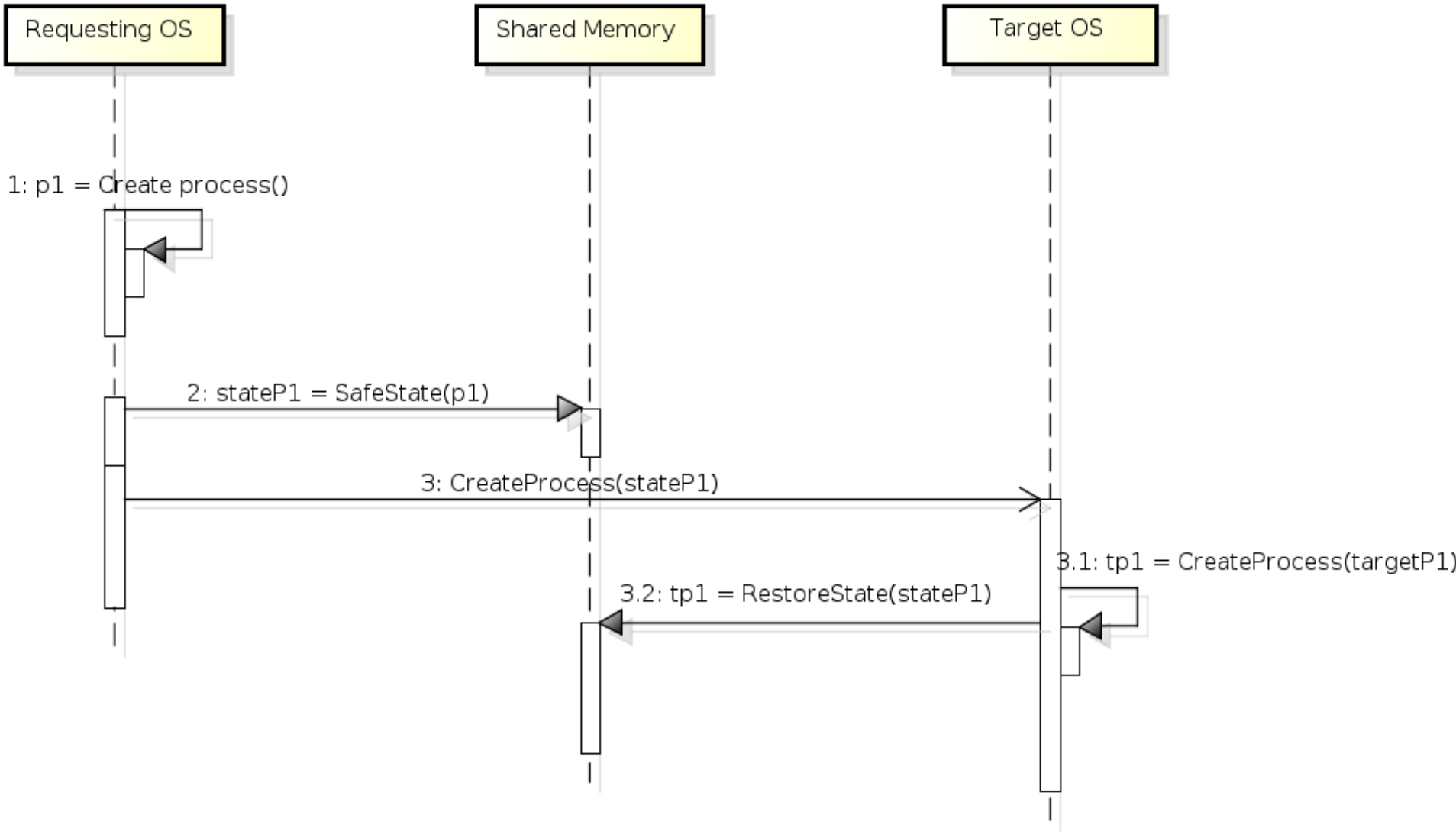
- processes/threads of one application must share some resources;
- VMMs are designed to isolate VMs from each other;
- *resource sharing layer* must modify VMM's code



Remote process



- Create local process
- Saves the state of the process
- Request target OS to create a process
- Target OS restores the state


Requesting OS
Shared Memory
Target OS
1: p1 = Create process()
2: stateP1 = SafeState(p1)
3: CreateProcess(stateP1)
3.1: tp1 = CreateProcess(targetP1)
3.2: tp1 = RestoreState(stateP1)

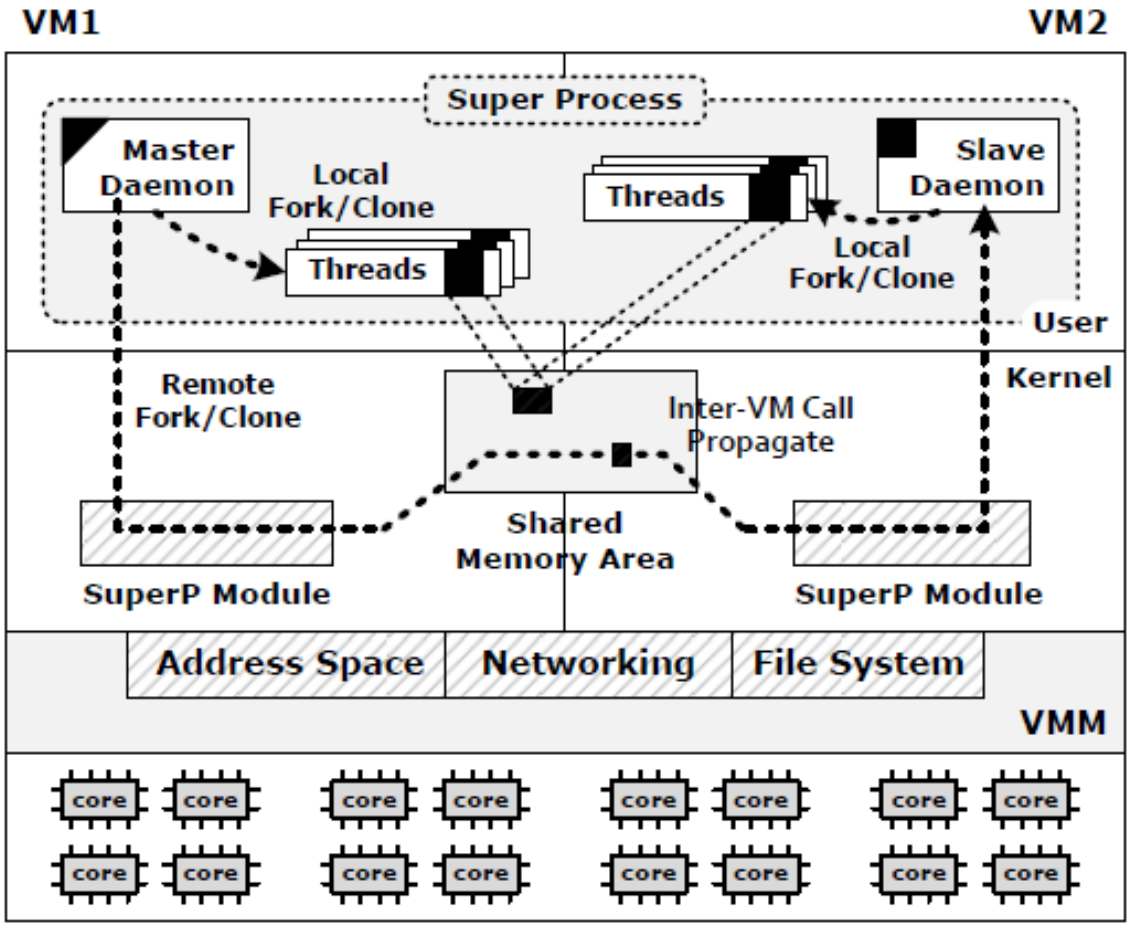
Performance techniques

- Only certain levels of the page tables are shared
- Use domains to address page faults
- Avoid conflicting by enforcing update memory state

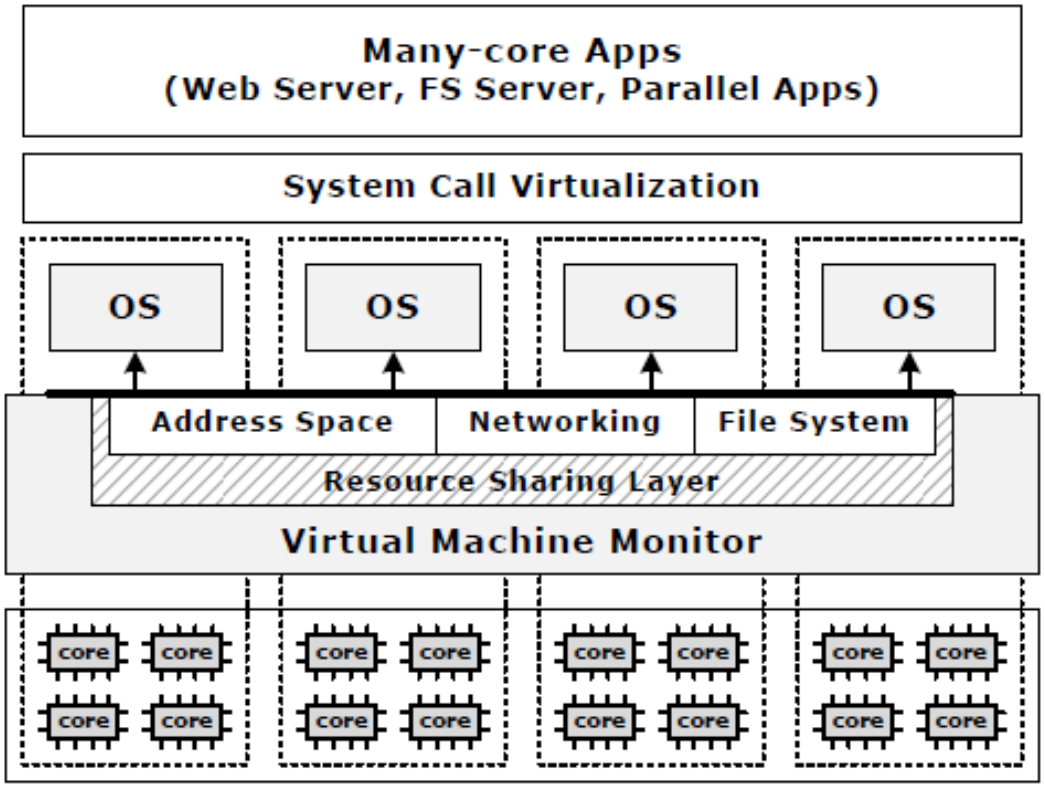
Shared Data Segment

- Not common for multiprocess application
- Very common for multi-threaded

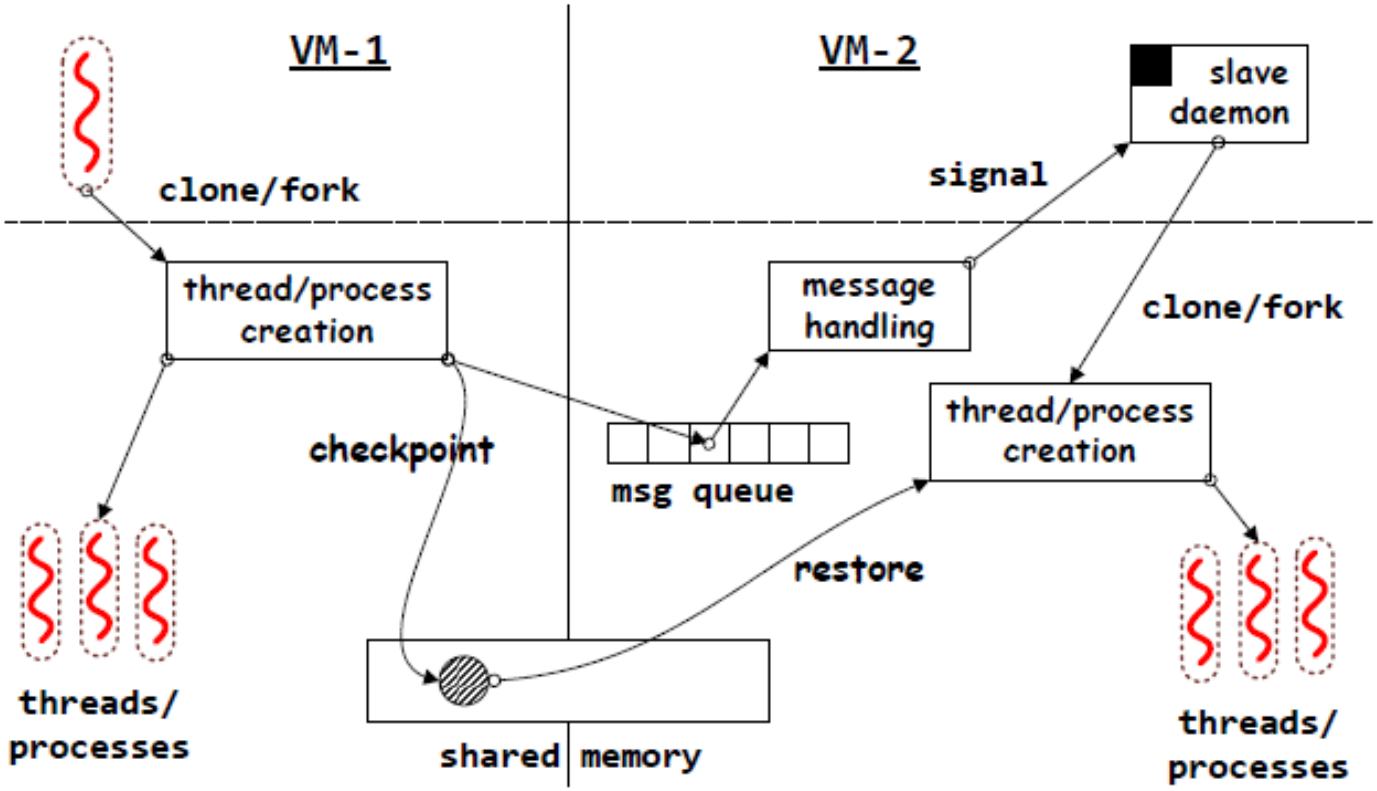
Cerberus arch



Syscall virtualization



Fork and Clone



Common approach

Use existent network file systems:

- NFS

Issues?

- Centralized
- Rather slow

Solution?

Sharing file

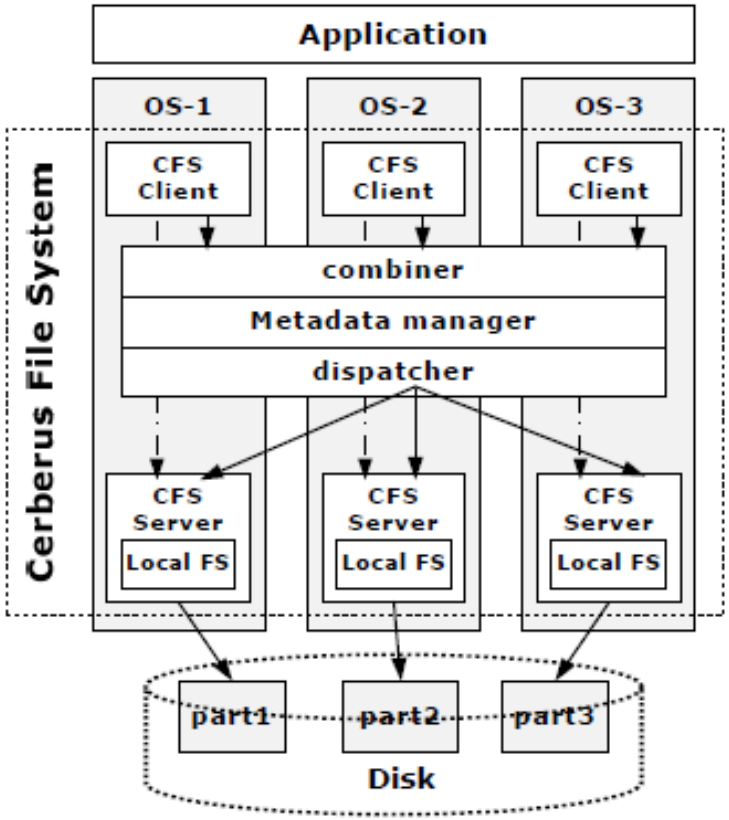
Question

How share data files efficiently?

The answer might be:

- Hybrid solution: local and remote file access.
- Basic security scheme: public or private file.

CFS arch


Application
OS-1
OS-2
OS-3
CFS Client
CFS Client
CFS Client
Cerberus File System
combiner
Metadata manager
dispatcher
CFS Server
Local FS
CFS Server
Local FS
CFS Server
Local FS
part1
part2
part3
Disk

Prototype

Tech overview:

- based on Xen
- Uses compare and swap to look for gaps
- using shadow mode page table
- re-implement a subset of POSIX calls
- Using P2M

Hardware overview

- AMD 48 cores
- 4 network cards

Downside

Some negative points of the paper:

- Do not show the performance of CFS in operations like: writing and seeking
- Do not explain how solve the problem of centralization for load balance
- Do not show the numbers for acceptable performance
- Code is not available