



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

Xiang Song

Work with

Haibo Chen, Rong Chen,  
Yuanxuan Wang and Binyu Zang

Parallel Processing Institute, Fudan University

# Since a Long Time Ago ...



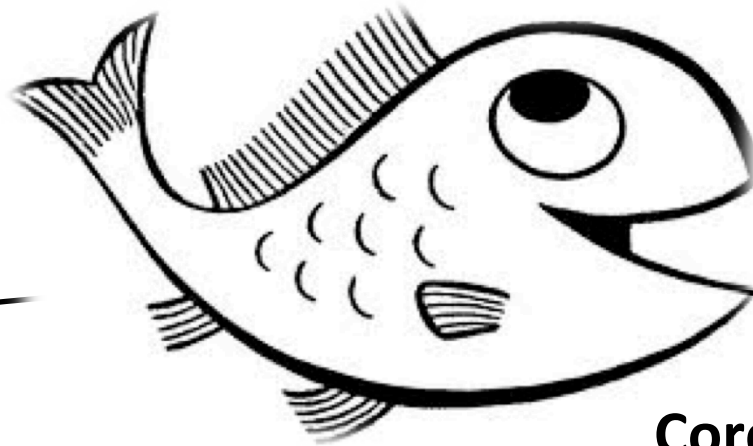
GNU/Linux

# Land is growing ...



**Multi-core**

# People proposed to raise fish



**Corey, fos, Barrelfish**



# Debate

But...

People are refining gnu  
to make it more  
delicious than before

Fish appears to be  
delicious for some  
people

**RCU**

McKenney,  
et.al.

**Sloppy**

**Counter**

Boyd-Wickizer,  
et.al.

2002

2010

**Barrelfish**

Baumann,  
et.al.

**Corey**

Boyd-Wickizer,  
et.al.

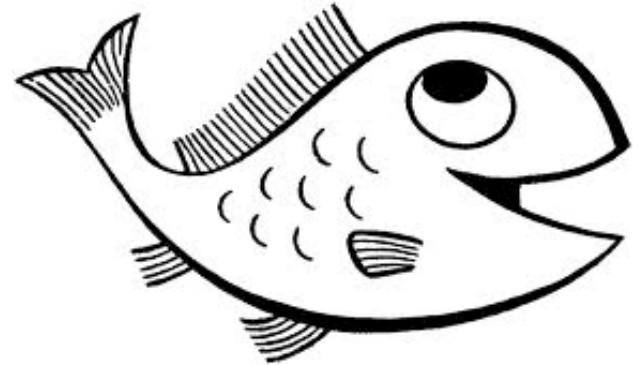
2009

2008

**fos**

Wentzlaff,  
et.al.

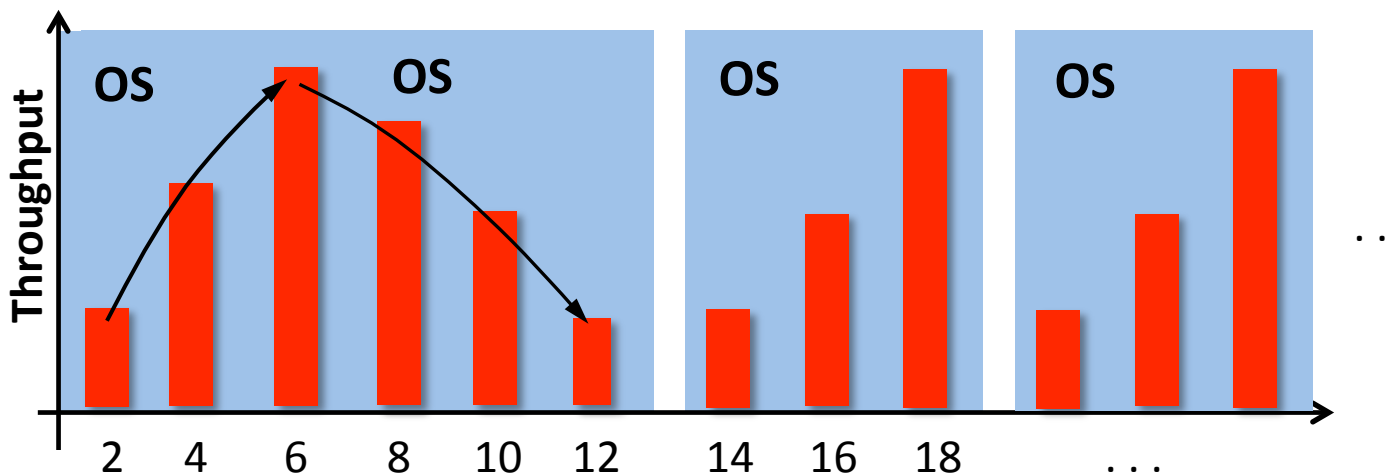
# Fish or Centaur's Choice





# Hypothesis of Cerberus

Commodity OS **scale well** with a **small number of cores** for many applications





# Cerberus

## Basic idea

- *Clustering* multiple OSes atop a VMM
- Serve one application with *POSIX interface*

## Goal

- Evaluate a middle point of improving scalability of commodity OSes
- Backward compatible to POSIX interface



# Challenges

- An app's resources spans across multiple OSes
  - Process/thread, address space, fs, network
- Should Provide
  - Distributed process/thread management
  - Single shared-memory interface
  - Efficient resource sharing

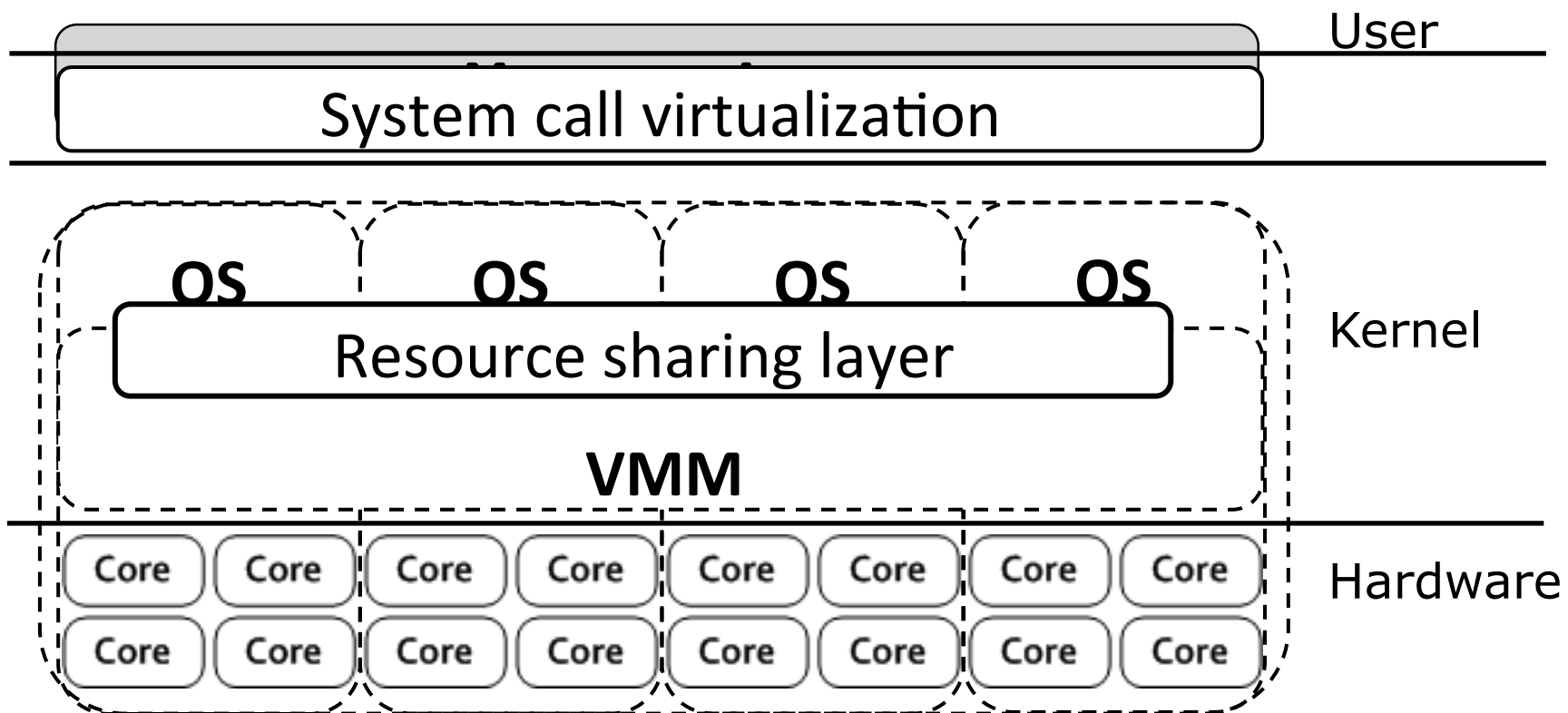


# Outline

- Cerberus Architecture
- Challenges & Solutions
- Implementation
- Evaluation

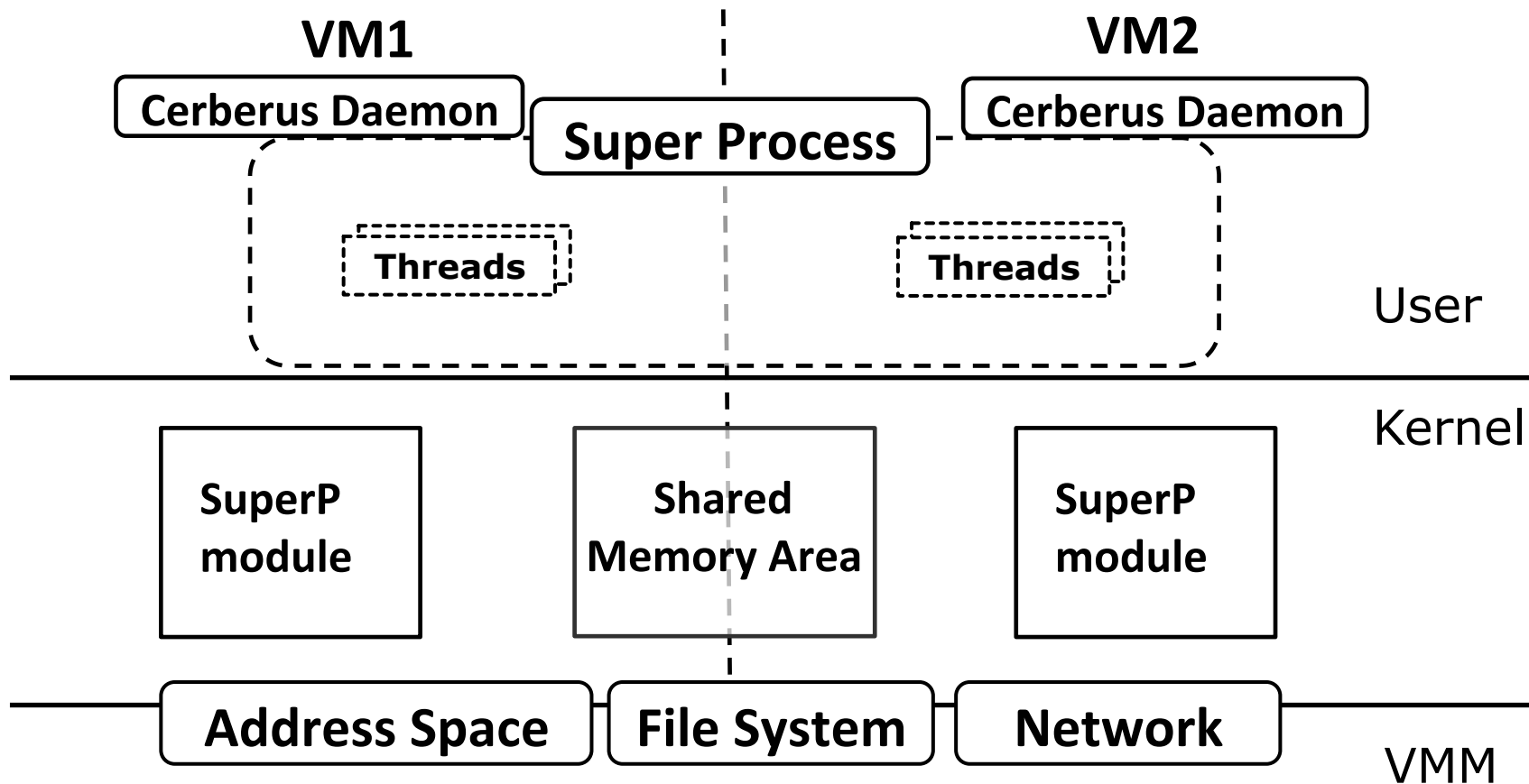


# Cerberus Architecture





# Detailed Architecture





# Outline

- Cerberus Architecture
- **Challenges & Solutions**
- Implementation
- Evaluation



# Challenge A

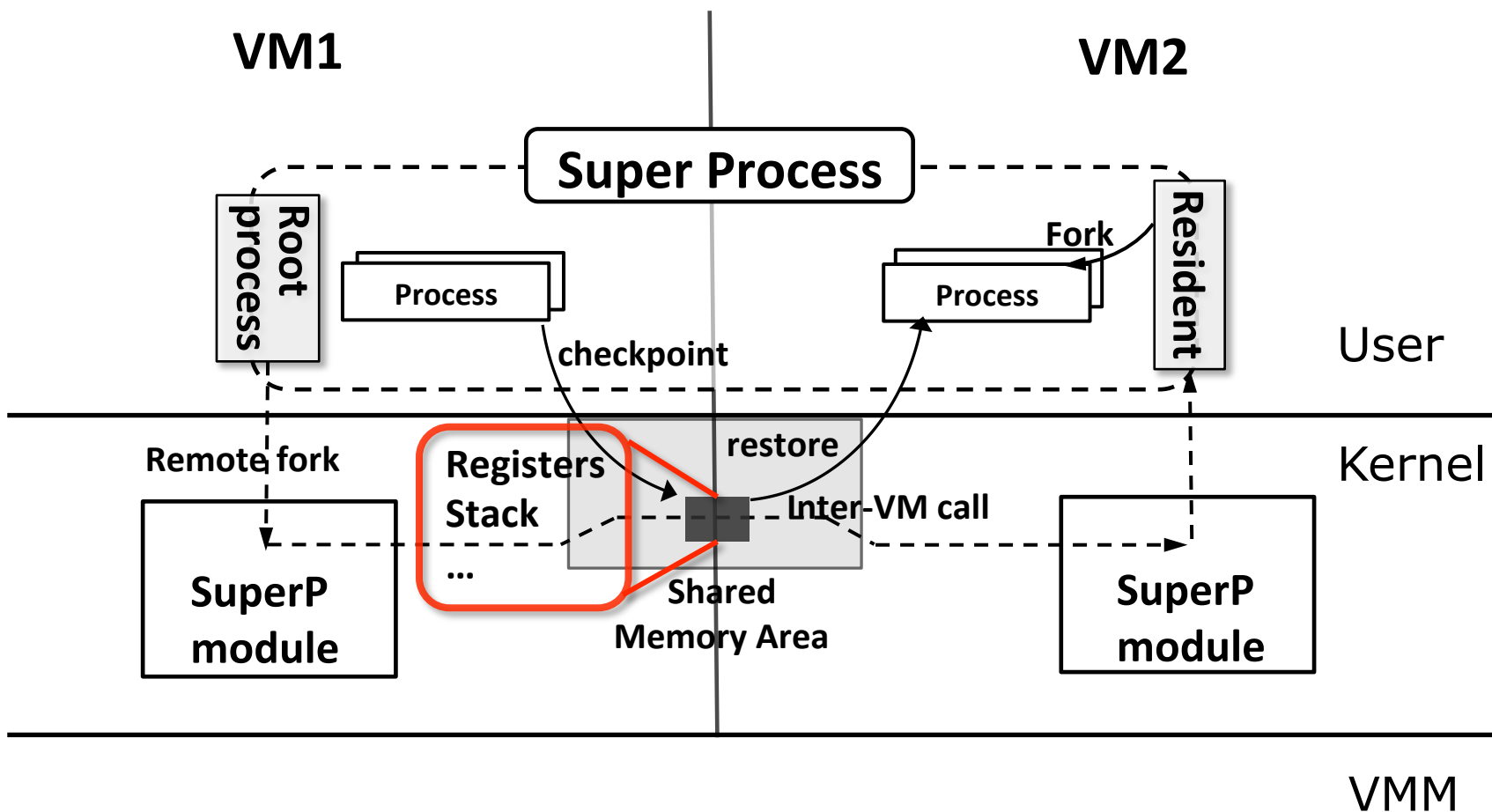
Need to run an app on a cluster of VMs

How does Cerberus spawn a thread/process on a remote VM?



# Support Super Process

## Remote Process/Thread Spawning





# Challenge B

We have threads/processes on different VMs

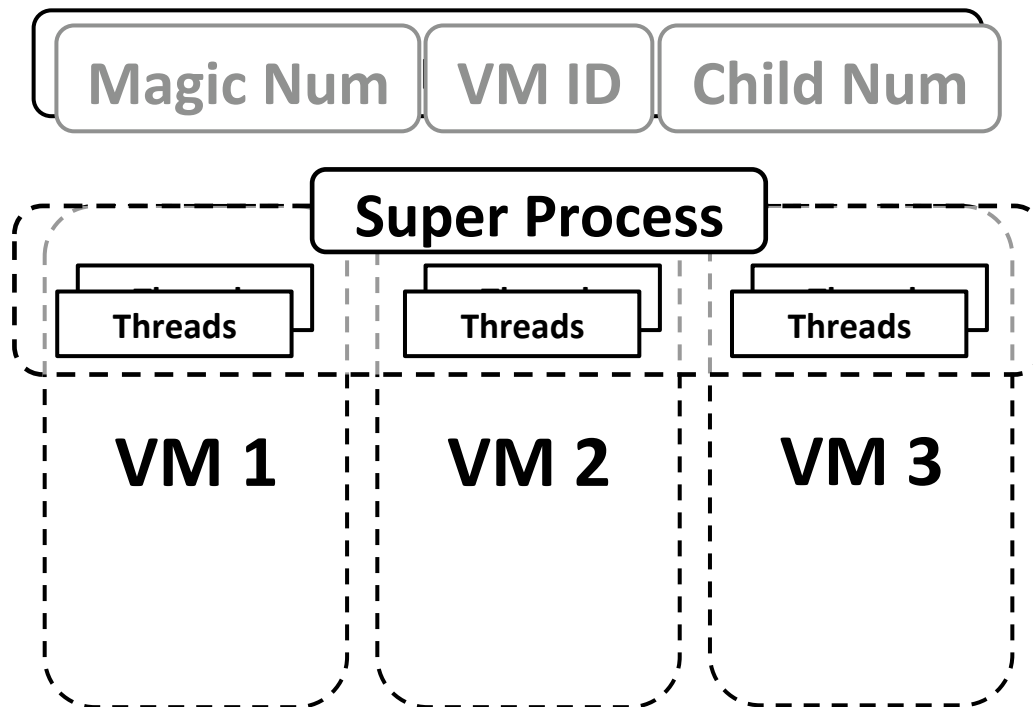
How does Cerberus identify a thread/process?



# Support Super Process

## Process management

- Virtual PID for each process/thread





# Challenge C

Threads/processes distributed across VMs, they need to cooperate

How does Cerberus virtualize system calls?



# System call virtualization

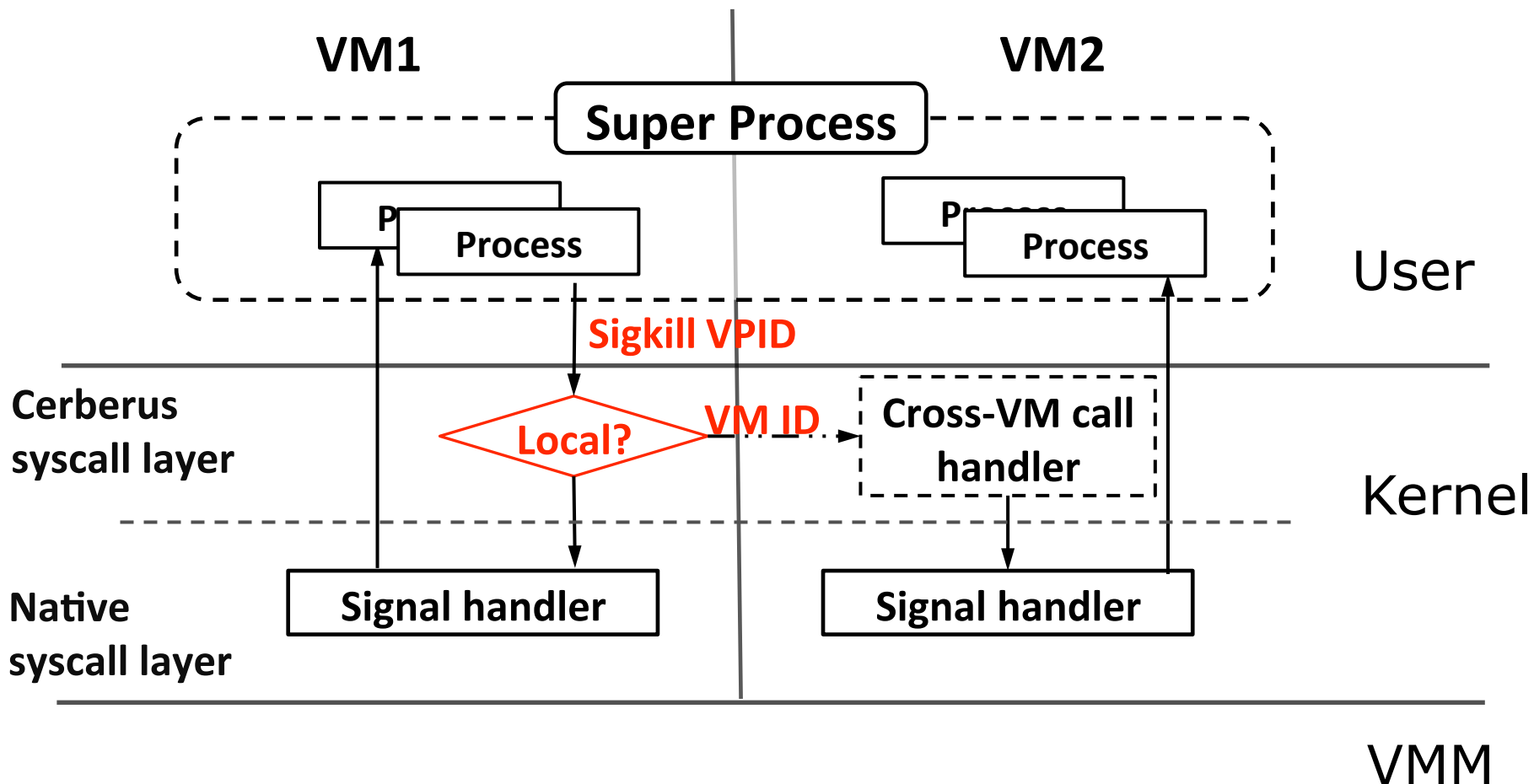
## Two types of system calls

- Access **local** state or stateless
  - Native syscall handler (e.g., `gettimeofday`)
- Access or modify **global** state
  - Cerberus syscall handler (e.g., `signal`)



# System call virtualization

## Example: Signal





# Challenge D

Threads/processes distributed among multiple VMs need to share data

How does Cerberus provide address space sharing?



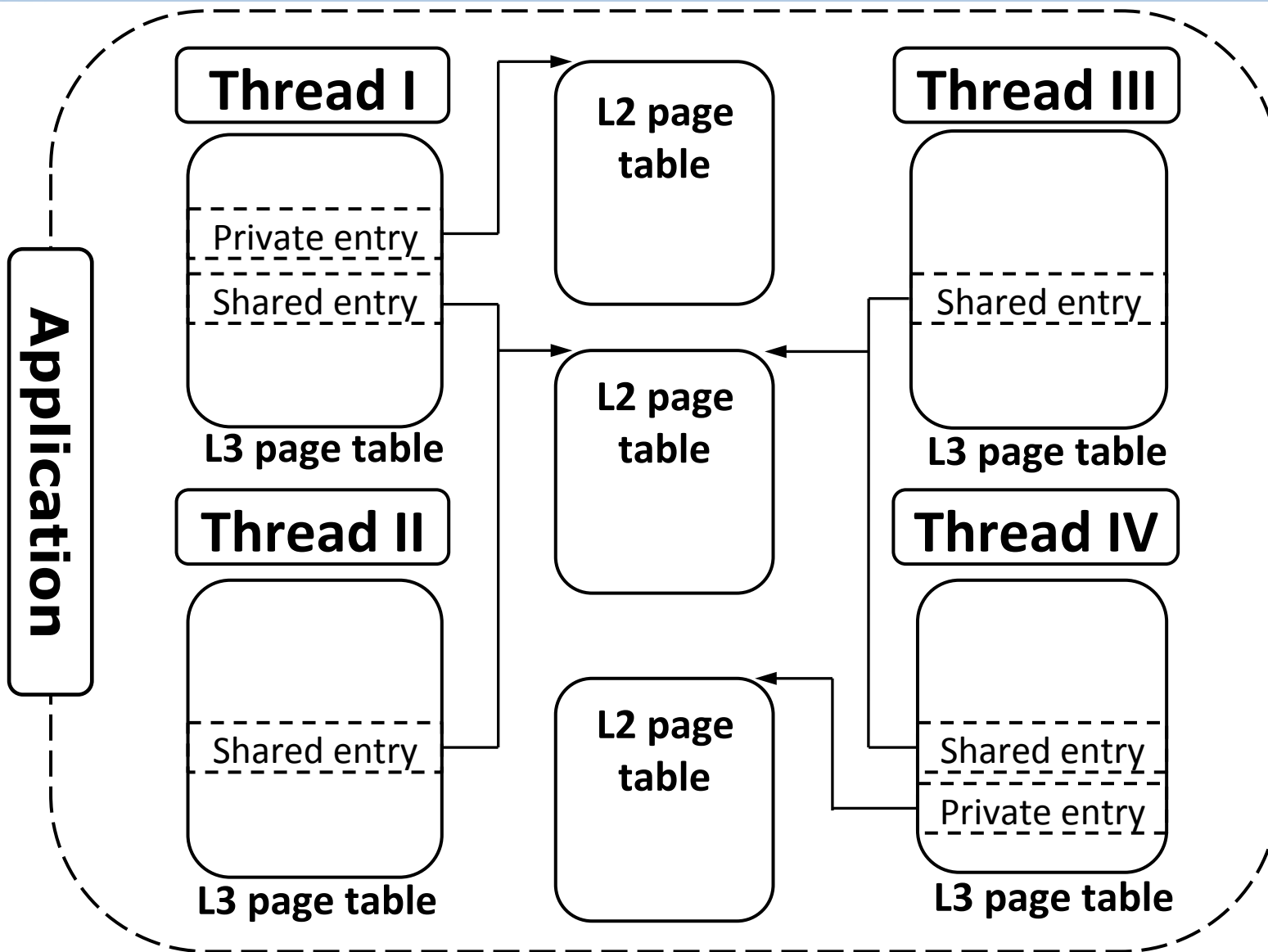
# Address space sharing

## Sharing a sub-tree of page table

– Corey (Boyd-Wickizer, et.al.)



# Address space sharing





# Challenge E

We have threads/processes sharing resources

How does Cerberus provide  
resource sharing?



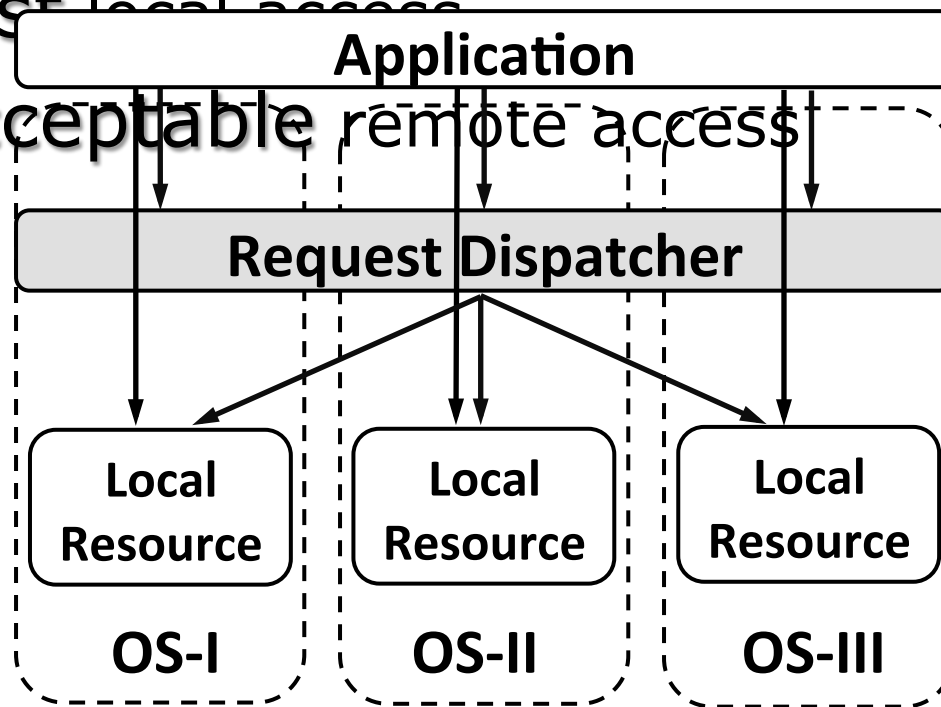
# Support Resource Sharing

## File system and Network sharing

– Most of accesses will be local

- Fast local access

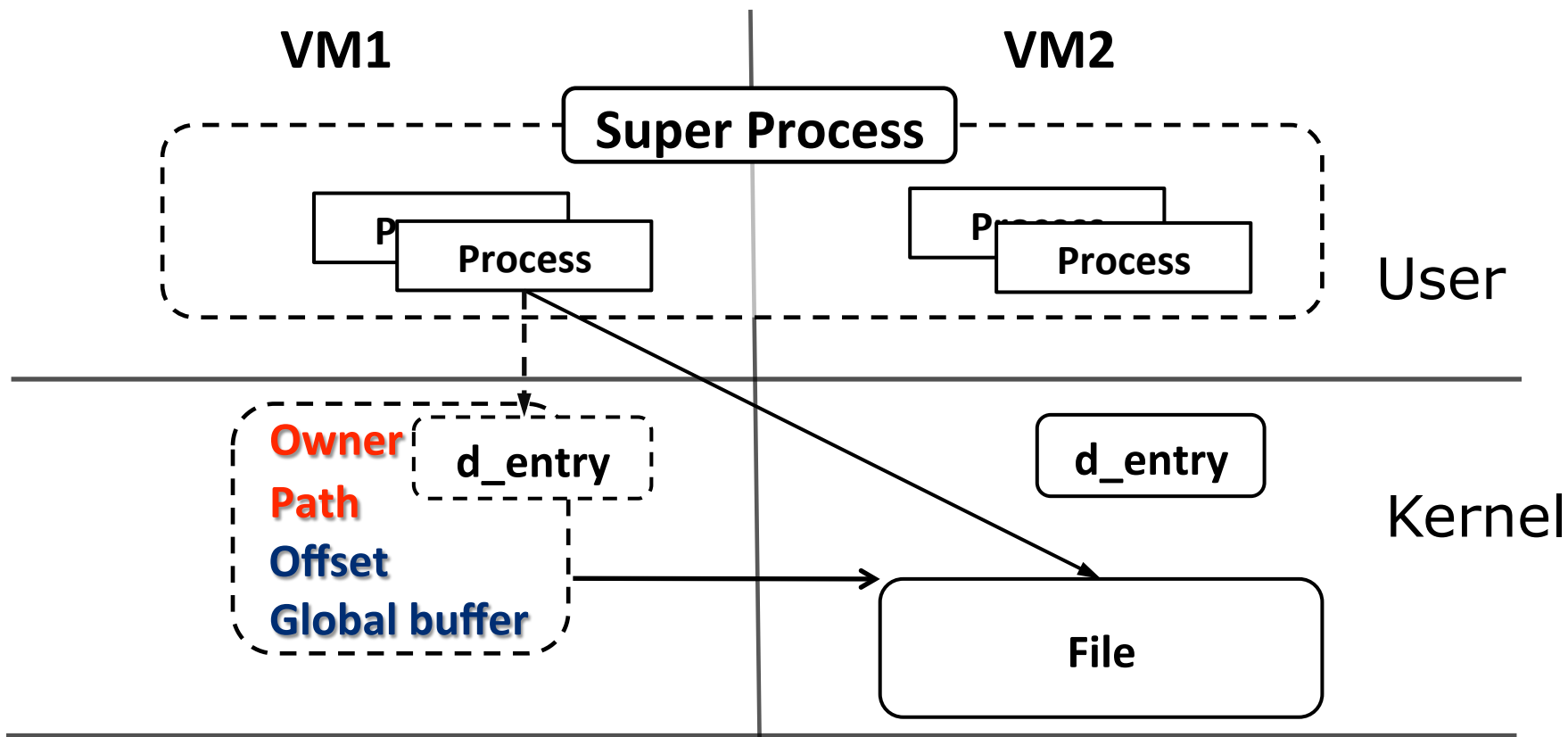
- Acceptable remote access





# Support Resource Sharing

- Example – remote file read





# Outline

- Cerberus Architecture
- Challenges & Solutions
- **Implementation**
- Evaluation



# Implementation

Based on Xen 3.3.0 + para-virtual VM

Add 1,800 lines in Xen VMM

- Management of super process
- Efficient data sharing

8,800 lines for SuperP module

- Support for super process
- Resource sharing
- Other support code



# Implementation

## Virtualize 35 POSIX system calls

	<b>Examples</b>
<b>Process/thread creation and exit</b>	fork, clone
<b>Process/thread communication</b>	futex, signal
<b>Memory management</b>	brk, mmap
<b>Network operations</b>	socket, connect
<b>File system operations</b>	open, read
<b>Security</b>	unhandled
<b>Real time signal</b>	unhandled
<b>Debugging</b>	unhandled
<b>Kernel modules</b>	unhandled



# Outline

- Cerberus Architecture
- Challenges & Solutions
- Implementation
- **Evaluation**



# Experimental Setup

## Software

### Benchmarks

- Histogram
- Dbench

## Hardware

### AMD Machine

48 Core (8 X 6) Opteron

Oprofile/Xenoprof



# Evaluation environment

## Three systems

- Linux 2.6.18
- Xen-Linux Dom0
- Cerberus
  - 1-core/domain
  - 2-core/domain



# Application Benchmarks

## Evaluation on the AMD machine

### – Histogram

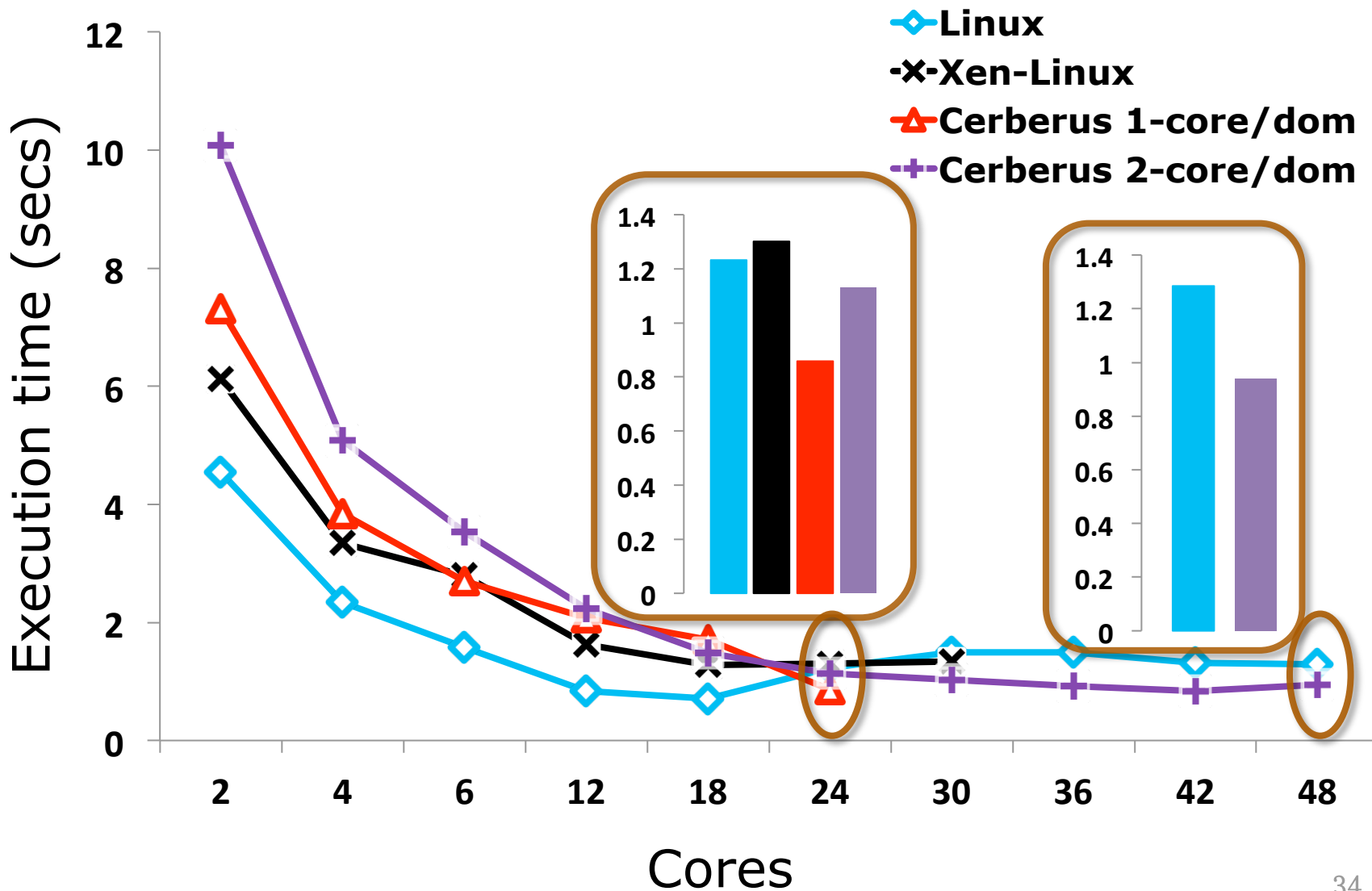
- 4 GByte input in ramfs
- 1 thread/core

### – Dbench

- 1 client/core



# histogram





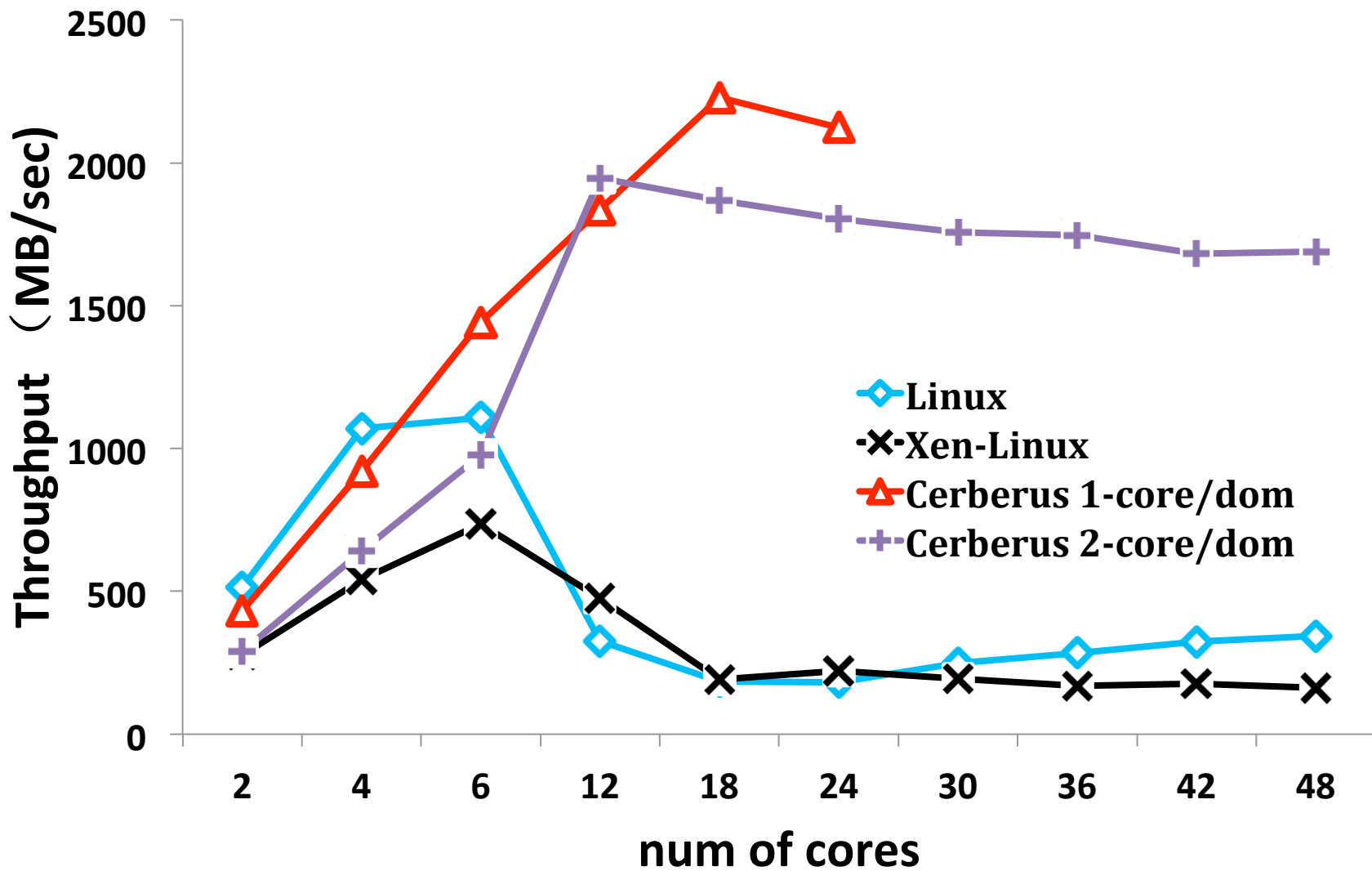
# Histogram analysis

- Top 3 hottest functions

	<b>Top 3 functions</b>	<b>Percentage</b>
Native Linux 48 threads	__up_read	<b>38.6%</b>
	__down_read_trylock	<b>35.9%</b>
	calc_hist	<b>8.3%</b>
Cerberus 2-core/VM 48 threads	calc_hist	<b>22.5%</b>
	sh x86 emulate cmpxchg guest 2	<b>8.9%</b>
	/xen-unknown	<b>8.3%</b>



# Dbench





# Dbench analysis

- Top 3 hottest functions

	<b>Top 3 functions</b>	<b>Percentage</b>
Native Linux 48 processes	ext3_test_allocatable	<b>66.6%</b>
	bitmap_search_next_usable_block	<b>18.2%</b>
	journal_dirty_metadata	<b>0.02%</b>
Cerberus 2-core/VM 48 processes	sh_x86_emulate_cmpxchg_guest_2	<b>11.2%</b>
	/xen-unknown	<b>8.67%</b>
	sh_x86_emulate_write_guest_2	<b>5.2%</b>



# Limitations

## Scenarios, Cerberus won't profit

- Applications that clone a number of short-lived, intensively communicating threads/processes
- Applications with frequent remote resource accesses
- Applications with frequent small-size memory mapping operations



# Conclusion

## Cerberus system

- Provide applications with the familiar *POSIX* programming interface
- Improve scalability of commodity OSes

## Experiments on Cerberus

- Some applications can *scale better* on Cerberus



# Future Work

- Cerberus without a VMM?
  - VMM incurs non-trivial overhead, it is not essentially necessary
  - A small dedicated thin software layer is sufficient
- Adjust Linux to make it cooperate with Cerberus
  - Minimize overheads associated with forks/mmmaps/signals

# Thanks

## Cerberus

See the ***past*** and ***present***

Looking for the ***future***

## Questions?



Parallel Processing Institute

<http://ppi.fudan.edu.cn>



# Backup



# Cerberus with more cores per domain

- The performance grows down when the number of cores per domain increases
  - The Shadow paging mode
  - Overhead introduced by the virtual layer



# Micro Benchmarks

- Remote fork/clone

	<b>1 process/thread</b>	<b>24 processes/threads</b>
Fork	5.40ms	31.77ms
Clone	3.21ms	30.79ms

- Cross-VM message passing

- 10.24 $\mu$ s for inner-chip
- 11.34  $\mu$ s for inter-chip



# Micro Benchmarks

- Local operations vs. remote operations
  - Sending signal
  - Sending and receiving packages
  - Read files



# Micro Benchmarks

- Local operations vs. remote operations
  - Sending signal
    - Cost of ping-ponging 1000 signals

	<b>Intel machine</b>	<b>AMD machine</b>
Native Linux	7.9ms	4.0ms
Xen-Linux	38.7ms	74.1ms
Cerberus local	43.1ms	72.3ms
Cerberus remote	25.8ms	45.0ms

- Read files



# Micro Benchmarks

- Local operations vs. remote operations
  - Sending and receiving packages
    - Cost of ping-ponging one packet 1000 times

	<b>Local host</b>	<b>Remote host</b>
Native Linux	11.25ms	125.5ms
Xen-Linux	42.9ms	132.6ms
Cerberus local	43.1ms	131.8ms
Cerberus remote	87.1ms	154.7ms



# Application Benchmarks

## Apache and Memcached

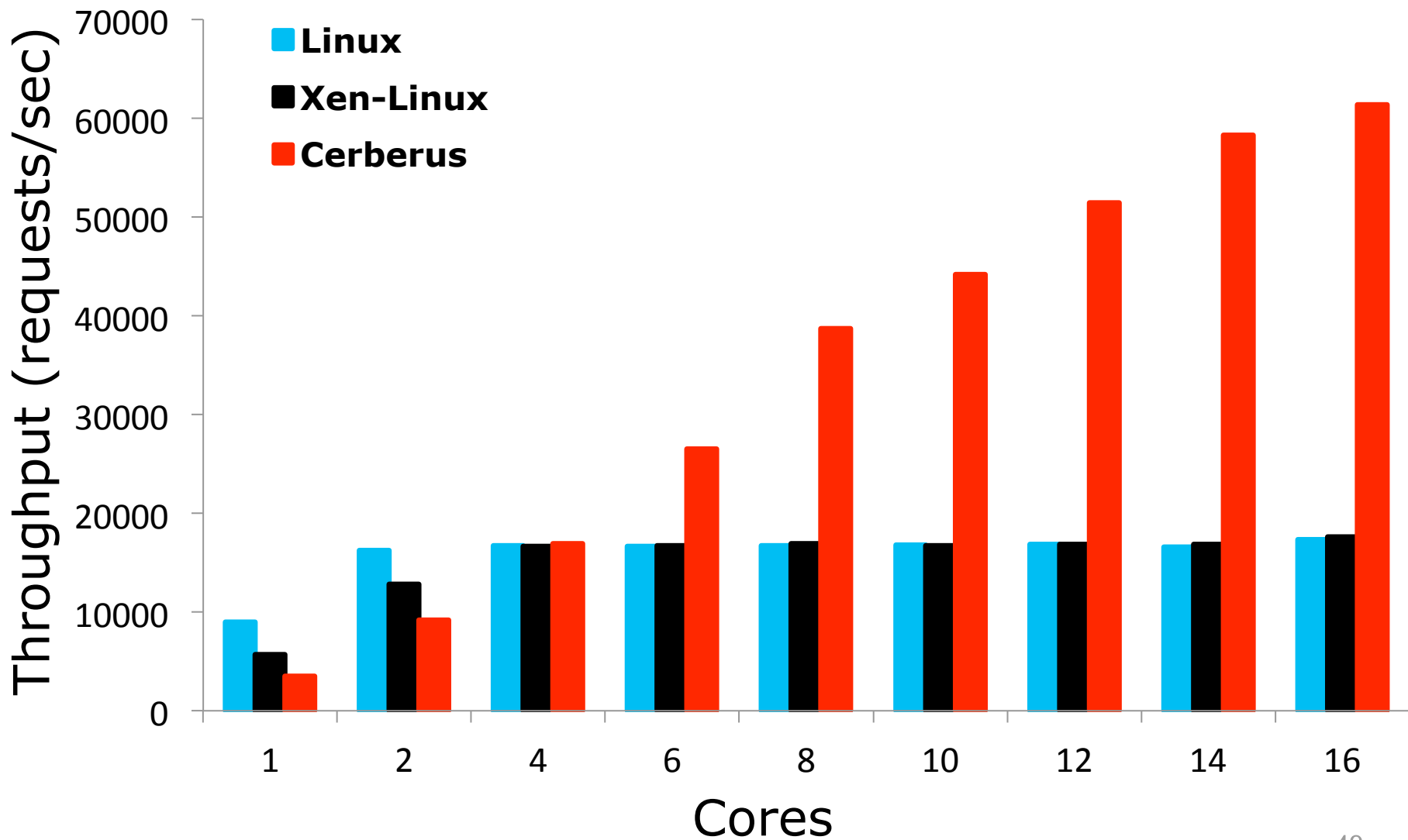
- A pool of 16 dual-core client machine

## Cerberus configuration

- 2 core/domain
- 1 NIC/domain
- 2 instances/domain

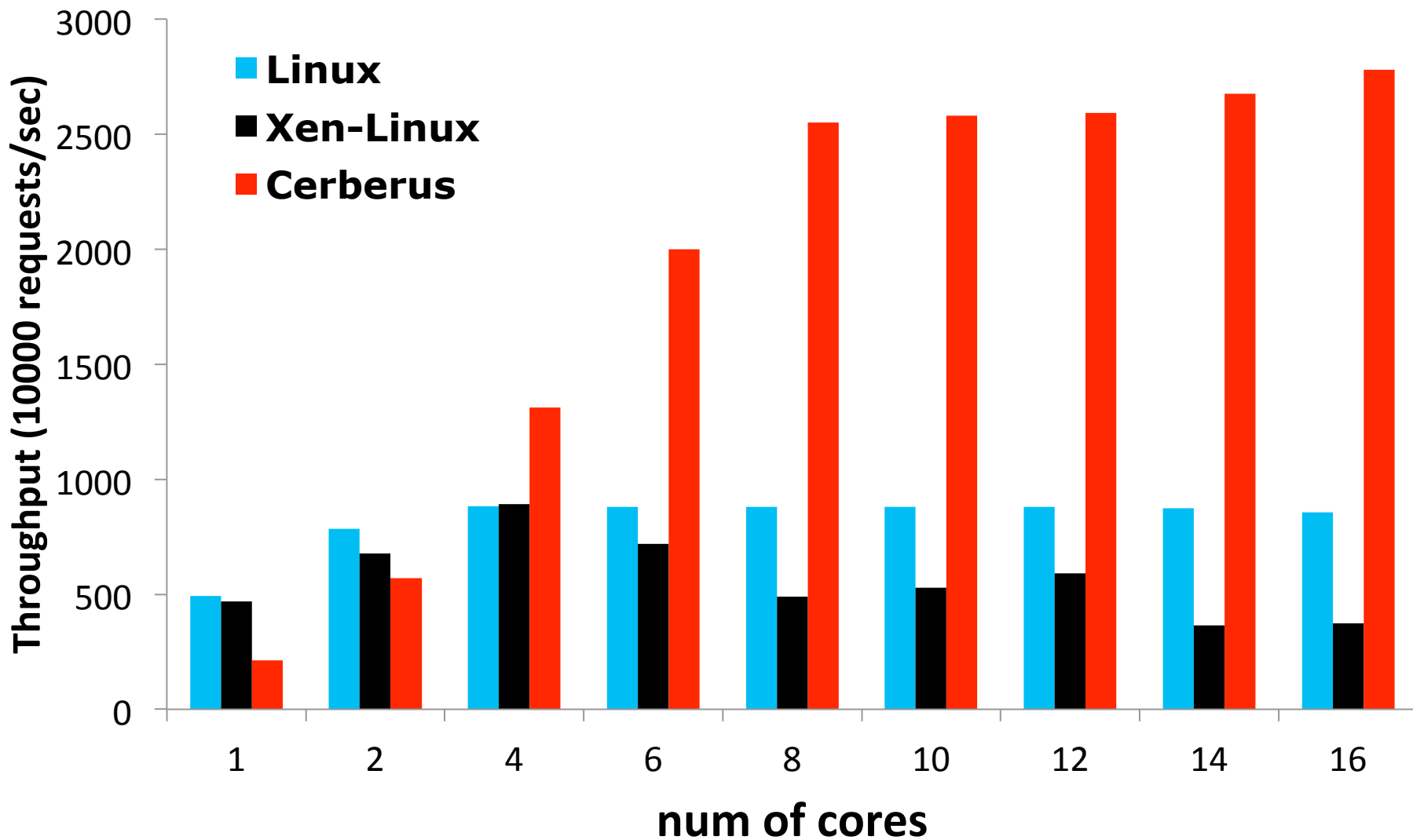


# Apache





# Memcached





# Apache performance

- The profiling result shows
  - The CPU utilization much lower in 16-core than in 1-core
    - 38.9% for Linux
    - 41.8% for Xen-Linux