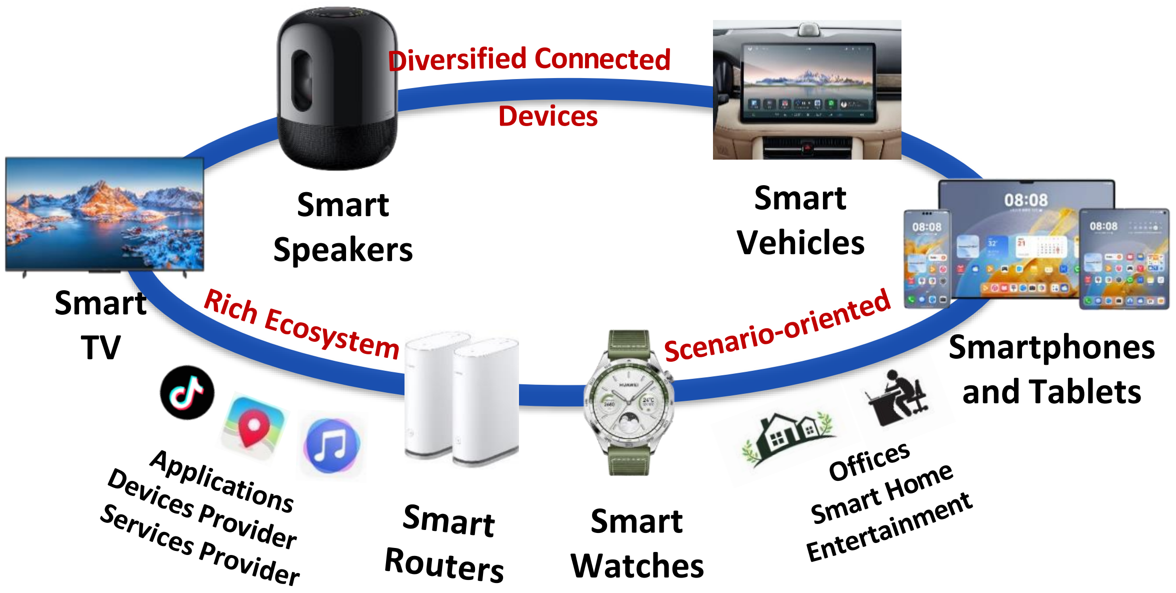
Microkernel Goes General: Performance and Compatibility in the ***HongMeng*** Production Microkernel

*Haibo Chen[1,2], Xie Miao[1], Ning Jia[1], Nan Wang[1], Yu Li[1], **Nian Liu[1]**, Yutao Liu[1], Fei Wang[1], Qiang Huang[1], Kun Li[1], Hongyang Yang[1], Hui Wang[1], Jie Yin[1], Yu Peng[1], and Fengwei Xu[1]*

[1]Huawei Central Software Institute [2]Shanghai Jiao Tong University

Revisiting **Microkernels** in an **Emerging Connected Intelligent World**



Requirement for OSes

✓ ***Stringent Security Requirements***

High-level Industrial Certifications

Protect Sensitive User Data

✓ ***Require Specialized Optimizations***

Full-system Optimizations

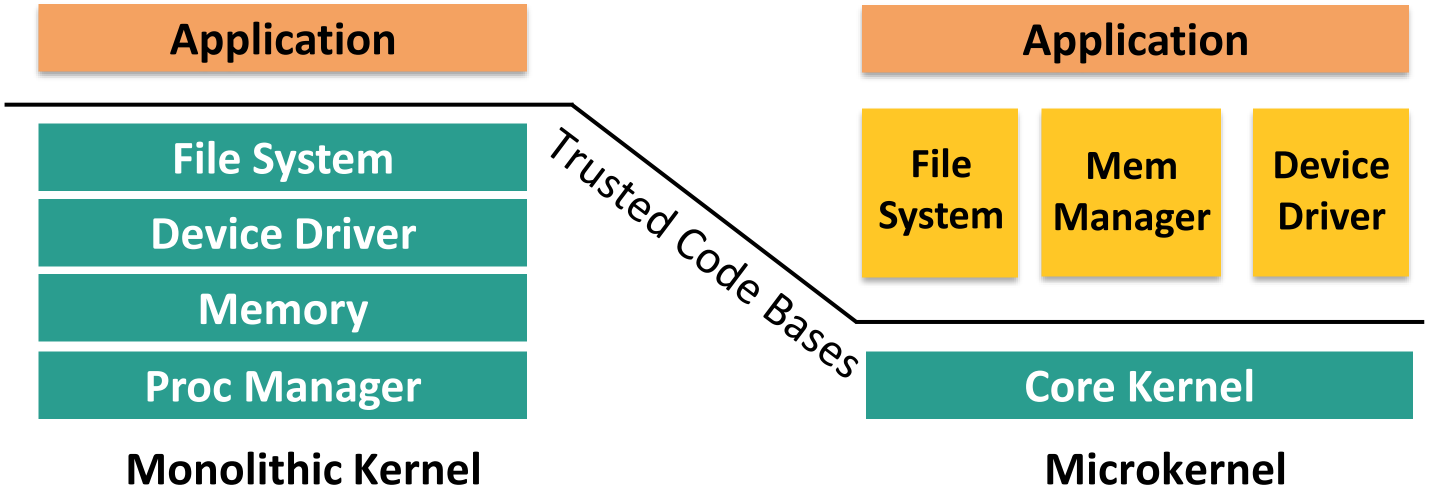
HW & SW Co-design

✓ ***Fast Evolution***

Fast Time to Market, R&D

Hard to Meet High Security Demands with Monolithic Kernels

- Reduction of trusted code bases
- **~70% of 1000 CVEs in the last 4 years** [1] can avoid by proper isolation [2,3]
- Difficult to satisfy high-level industry certifications [4]

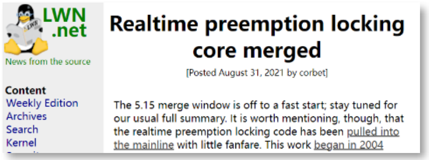


[1] Analyzing the Linux CVEs in recent 4 years https://cve.mitre.org/.
[2] Elton Lum. Study Confirms That Microkernel Is Inherently More Secure.
[3] Simon Biggs, et al. The jury is in: Monolithic OS design is flawed: Microkernel-based designs improve security. APSys '18.
[4] Mark Pitchford. Using Linux with critical applications: Like mixing oil and water?

Tightly-coupled Modules in Monolithic Kernels Impede Specialization

- Hard & costly to apply **domain-specific strategies**, e.*g.*, *QoS-aware allocation*
- E.g., Took over **10 years** for PREEMPT-RT patch set to be **partially merged** [1,2,3]

Began in 2004 [1]



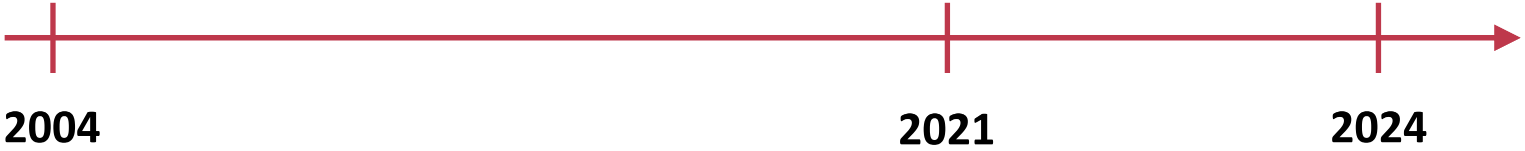
Realtime preemption locking core merged

[Posted August 31, 2021 by corbet]

The 5.15 merge window is off to a fast start; stay tuned for our usual full summary. It is worth mentioning, though, that the realtime preemption locking code has been pulled into the mainline with little fanfare. This work began in 2004

The remarkable ***preemption core locking*** was merged [2]

Still has not been merged into mainline Linux [3]



Timeline of **PREEMPT_RT** patch

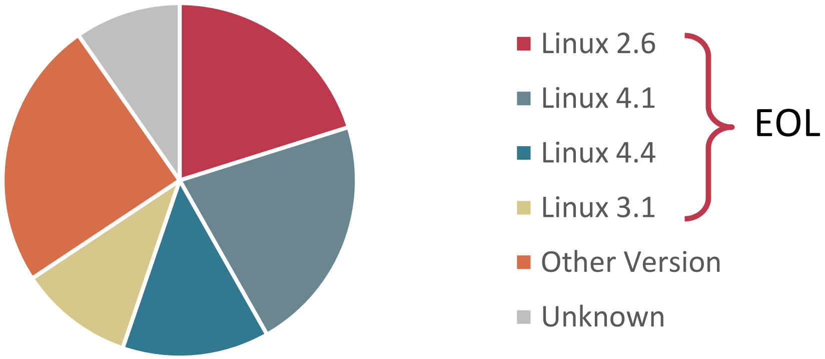
[1] Jonathan Corbet. Approaches to realtime Linux. https://lwn.net/Articles/106010/
[2] Jonathan Corbet. Realtime preemption locking core merged. https://lwn.net/Articles/867919/
[3] Jonathan Corbet. Jonathan Corbet. The real realtime preemption end game. https://lwn.net/Articles/951337/

Evolving Custom Code with Upstream Linux is Costly

- Synchronizing with upstream for **security patches** is expensive

  Require large-scale performance regression testing, even rewriting

- No surprise to see massive amounts of products in market run **Linux 2.6** [1]



Linux versions in 122 models of on-stock routers from 7 different vendors in 2022 [1].

All top 4** mostly-used Linux versions have already reached **EOL.

[1] *Johannes vom Dorp René Helmke. Home Router Security Report 2022.*

Emerging Scenarios Pose Challenges for Microkernels

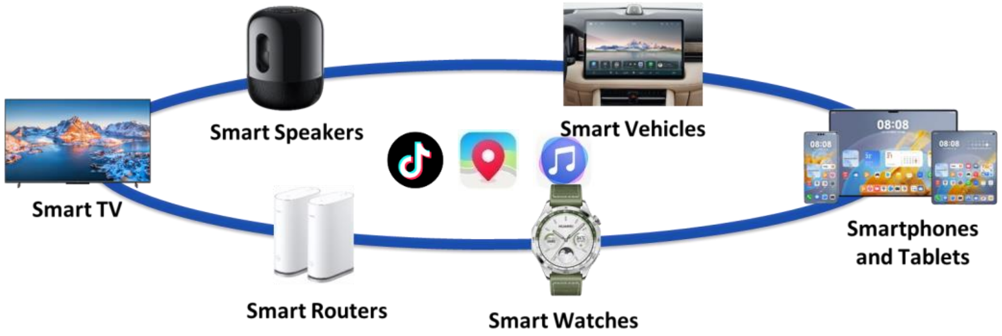
Most SOTA microkernels target

	Domain-specific Scenarios		*Emerging Scenarios*
Software Ecosystem	**Pre-determined, source-available**	→	**Open ecosystem, distributed in binary form**
Resources Management	**Pre-partitioned, self-managed**	→	**Coordinated, globalized management**
Performance Requirement	**Value security more**	→	**Prioritize performance, also emphasize security**

Routers & Switches

Robotic Arms

Wireless Modem Chips



Contributions: Microkernel Goes General with HongMeng Kernel

- **Revisiting microkernel design** for emerging scenarios

  Identifies the unsolved performance and compatibility challenges

- **HongMeng** production microkernel
  - ✓ *Retains minimality principle*

    Maintains most benefits of microkernels

  - ✓ *Provides structural supports*

    Addresses the performance and compatibility challenges

- Implemented and deployed in massive production
- Typically with **improved performance** over Linux

Outline

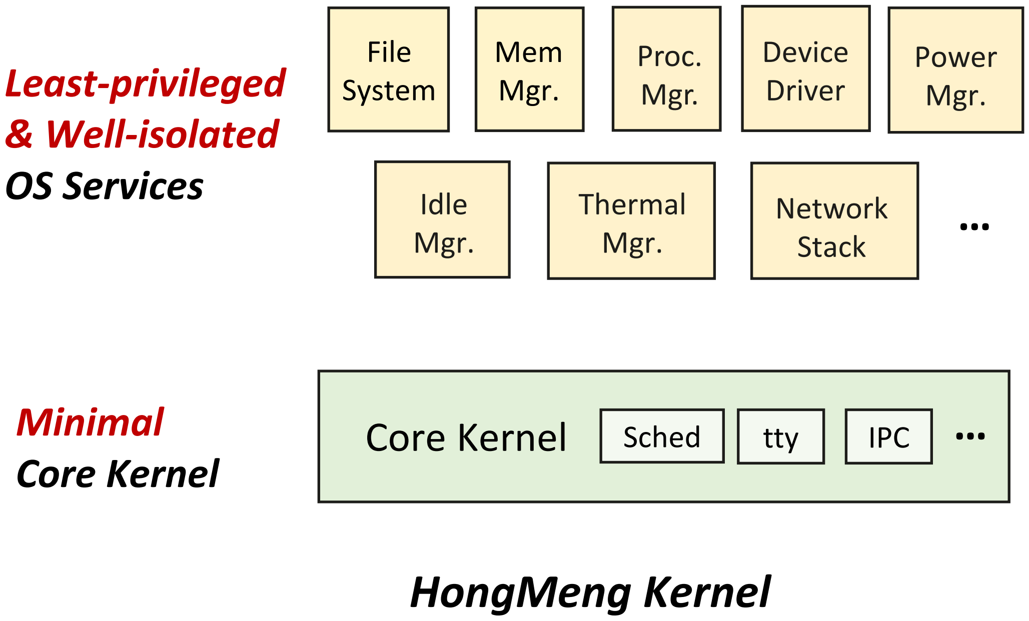
- ✓ **Revisiting Microkernel for Going General**
- ✓ Implementation and Performance
- ✓ Lessons and Experiences

Revisiting Conventional Wisdoms in Microkernels

	Conventional Wisdoms	Problems	HongMeng Kernel
Minimality	**Minimal Core Kernel**	**N/A**	**Retains Minimality**
IPC/Isolation			
Service Partitioning			
Access Control			
Interface			
Drivers			

Revisiting **Minimality**

Retaining Minimality To Preserve Microkernels' Benefits



- **Minimal** core kernel

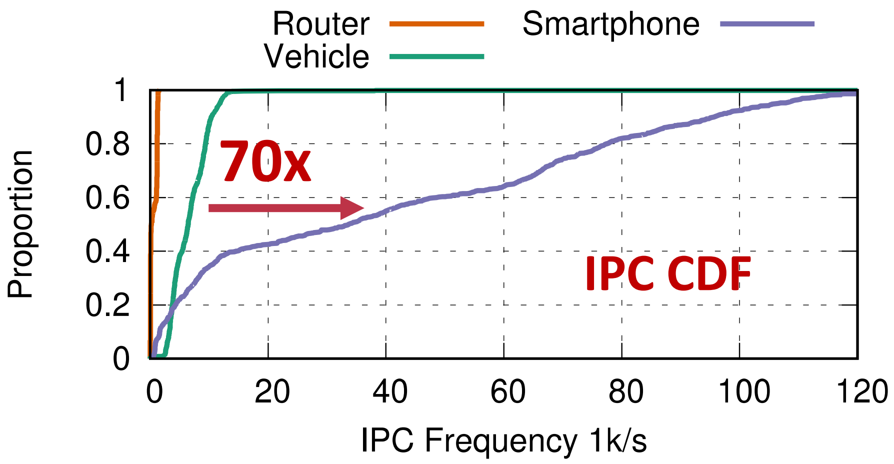
  Scheduler, IPC, access control, essential drivers like tty
- **Fine-grained** access control
- **Decoupled, least-privilege, and well-isolated** OS services

Revisiting Conventional Wisdoms in Microkernels

	Conventional Wisdoms	Problems	HongMeng Kernel
Minimality	Minimal Core Kernel	N/A	Retains Minimality
IPC/Isolation	**All Services at Userspace**	**Overly Strong Considering High IPC Frequency**	**Isolation Classes**
Service Partitioning			
Access Control			
Interface			
Drivers			

Rapidly Increased IPC Frequency Amplifies Performance Degradation

IPC Frequency CDF in Various Scenarios

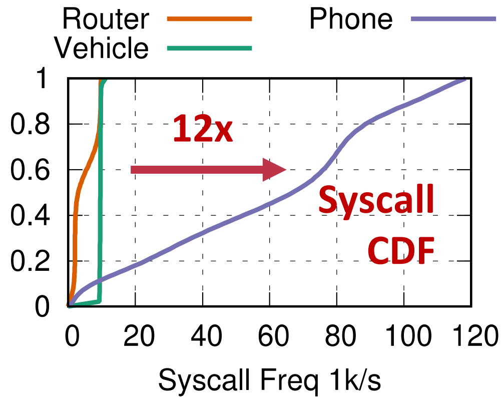


70x

IPC CDF

In-production Typical Usage

***High IPC frequency** leads to **2x to 3x** performance degradation in phones*



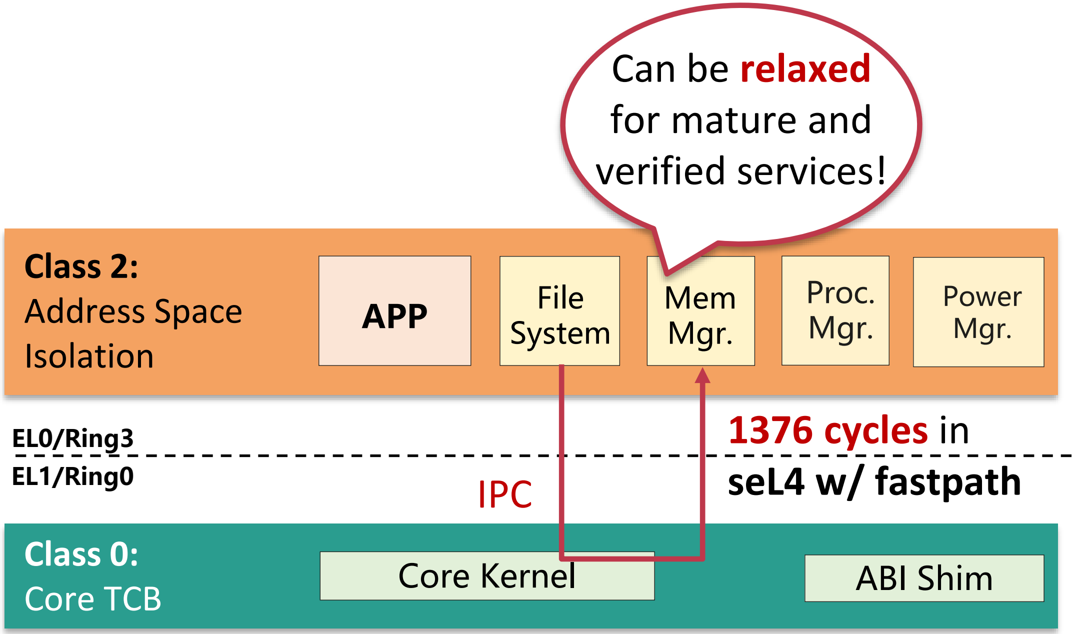
12x

Syscall CDF

Partly due to the high syscall frequency.

Revisiting
IPC

Isolation Classes Tailor Isolation for Services and Scenarios


Can be **relaxed** for mature and verified services!
Class 2: Address Space Isolation
APP
File System
Mem Mgr.
Proc. Mgr.
Power Mgr.
EL0/Ring3
EL1/Ring0
IPC
1376 cycles in **seL4 w/ fastpath**
Class 0: Core TCB
Core Kernel
ABI Shim

Differentiated Isolation Classes** in **HongMeng

- **Relax isolation** for trusted services
- **Classify services** and **define isolation**

***Class 2:** Address Space Isolation*

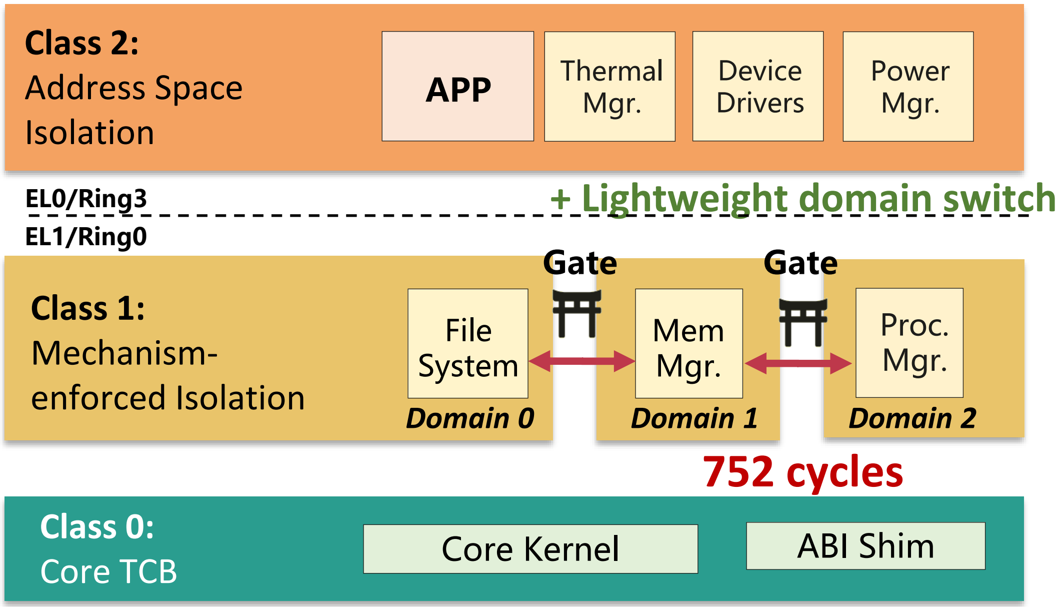
+ Address space switches
+ Privilege-level switches

> 50% Overhead

***Class 0:** Trusted Code Bases*

- *No isolation is enforced*

How Does Class 1 Relax Isolation and Speedup IPC?



Class 2:
Address Space Isolation

APP | Thermal Mgr. | Device Drivers | Power Mgr.

EL0/Ring3

+ Lightweight domain switch

EL1/Ring0

Class 1:
Mechanism-enforced Isolation

Gate | Gate

File System — ***Domain 0***

Mem Mgr. — ***Domain 1***

Proc. Mgr. — ***Domain 2***

752 cycles

Class 0:
Core TCB

Core Kernel | ABI Shim

Differentiated Isolation Classes *in* ***HongMeng***

Class 1: Mechanism-enforced *Isolation*

- ***IPC only Involves Lightweight Domain Switches***

  1376 Cycles => 752 Cycles

- ***Restrict Cross Domain Accesses***

  Intel PKS or ARM watchpoint

- ***Forbid Privileged Instructions***

  Lightweight CFI + secure monitor

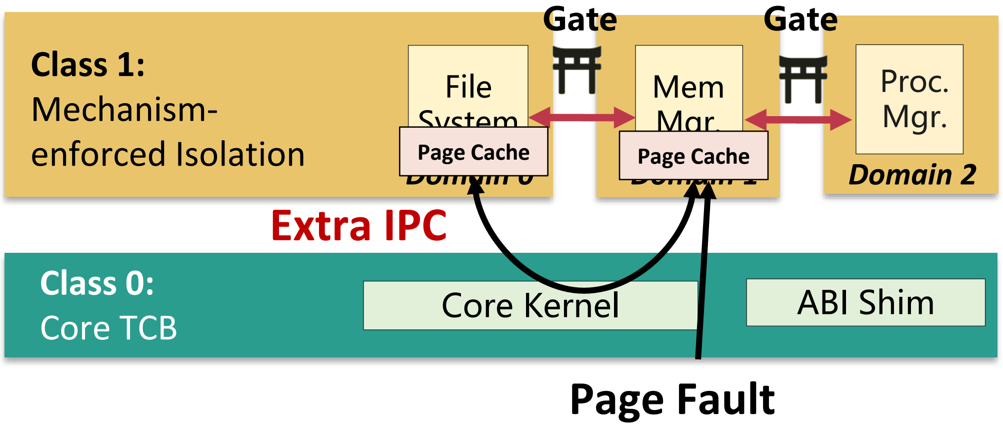
- ***Threat Model***

  Additional attack surfaces

Revisiting Conventional Wisdoms in Microkernels

	Conventional Wisdoms	**Problems**	**HongMeng Kernel**
Minimality	Minimal Core Kernel	N/A	Retains Minimality
IPC/Isolation	All Services at Userspace	Overly Strong Considering High IPC Frequency	Isolation Classes
Service Partitioning	**Static Multi-server**	**State Double Bookkeeping**	**Flexible Composition**
Access Control			
Interface			
Drivers			

Multi-server Design Causes State Double Bookkeeping



Class 1:
Mechanism-enforced Isolation

File System

Page Cache

Domain 0

Gate

Mem Mgr.

Page Cache

Domain 1

Gate

Proc. Mgr.

Domain 2

Extra IPC

Class 0:
Core TCB

Core Kernel

ABI Shim

Page Fault

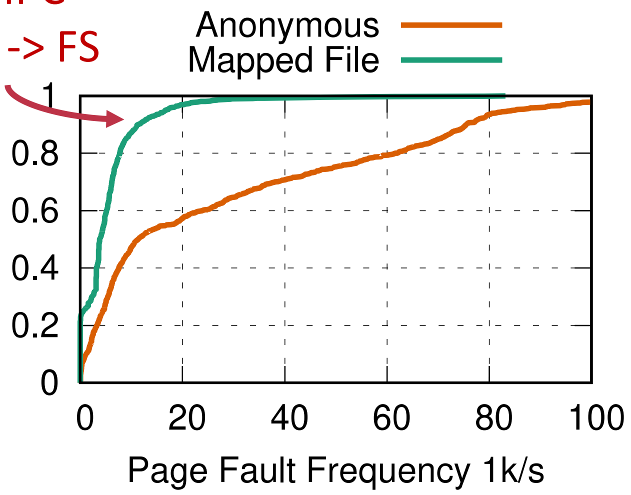
***Paging for mapped files** is **2x slower** than Linux*

Double-bookkeeping of Page Cache

Higher IPC frequency and memory overhead

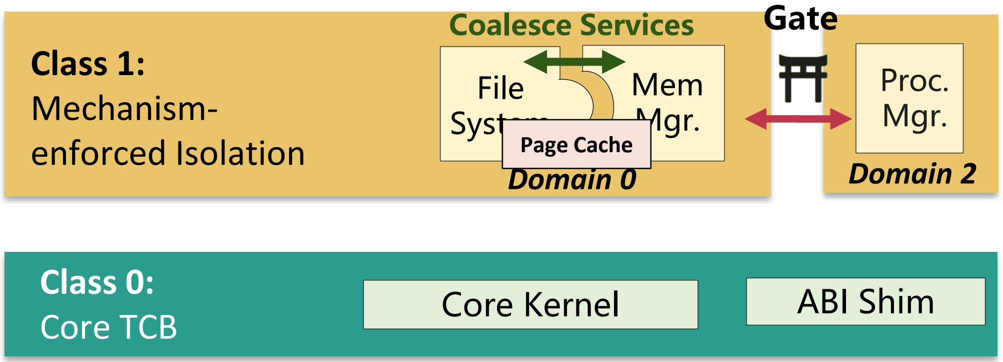
Minor page fault CDF in Smartphones

Extra IPC
Mem -> FS



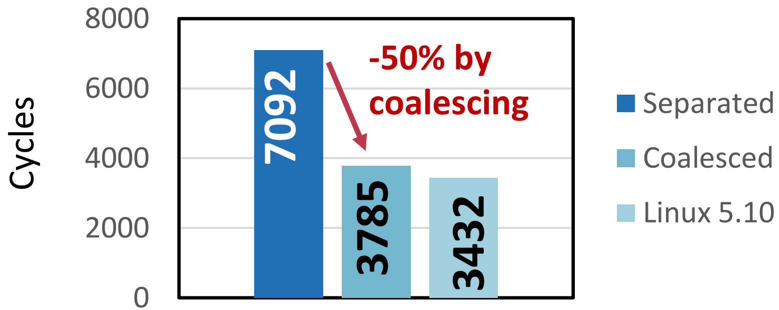
In-production Typical Usage

Coalescing Coupled Services in Performance-critical Scenarios



Coalescing Coupled Services** in **Smartphones

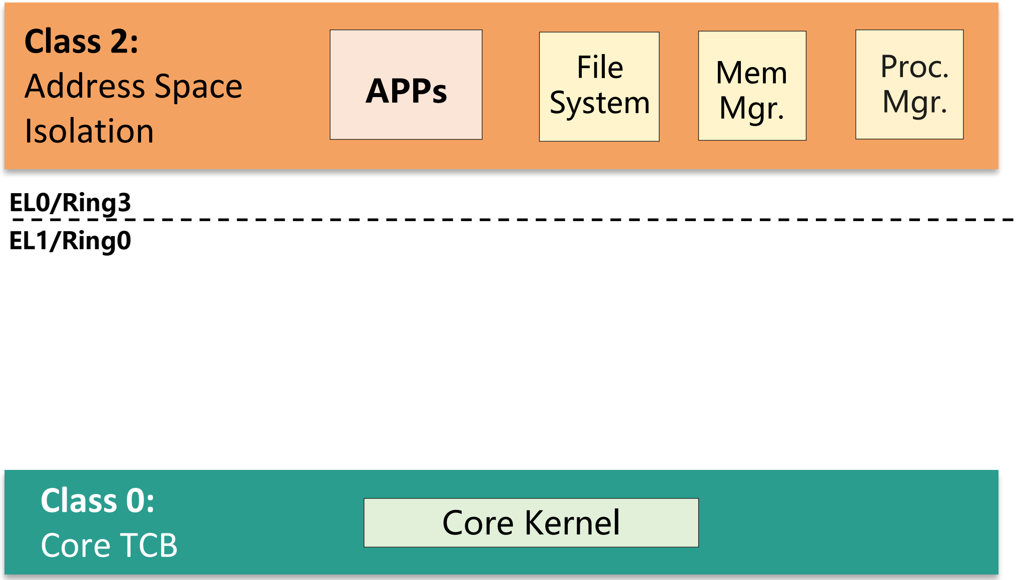
- **Coalescing** coupled services
  - *Reducing IPC frequency*
  - *Eliminating double-bookkeeping*



***Page Fault Latency** of mapped files*

Revisiting Partitioning

Flexibly Assemble the System for Various Scenarios



HongMeng Kernel** in **Routers** and **Secure OS (TEE)

- Service coalescing and isolation classes are configurable **during deployment**

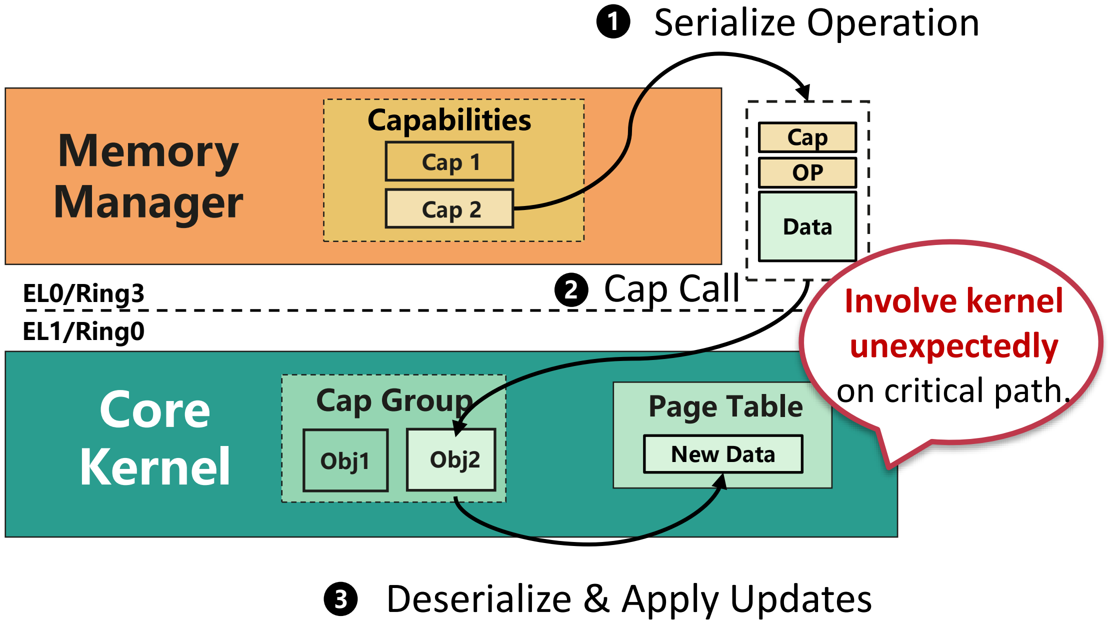
  Accommodate various scenarios

- **Separate or enforce stronger isolation** when **new attack emerges**

Revisiting Conventional Wisdoms in Microkernels

	Conventional Wisdoms	Problems	HongMeng Kernel
Minimality	Minimal Core Kernel	N/A	Retains Minimality
IPC/Isolation	All Services at Userspace	Overly Strong Considering High IPC Frequency	Isolation Classes
Service Partitioning	Static Multi-server	State Double Bookkeeping	Flexible Composition
Access Control	**Capabilities**	**Hide Kernel Objects**	**Address Tokens**
Interface			
Drivers			

Why are Capabilities Slow When Updating Objects?



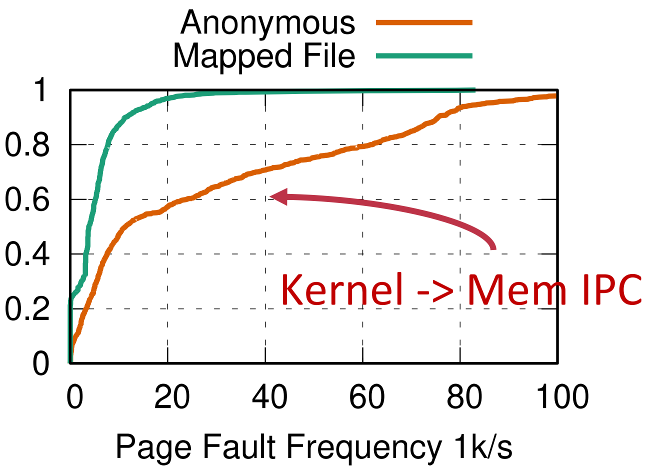
Involve kernel unexpectedly on critical path.

***Capability-based** Access Control*

Overhead of Capabilities

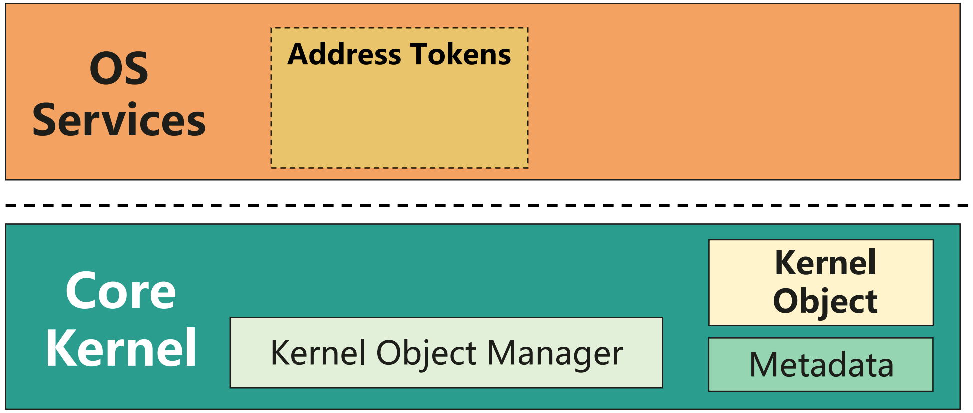
- + Serialize operation
- + Privilege level switch
- + Referring the cap table
- + Deserialize operation

Minor page fault CDF in Smartphones



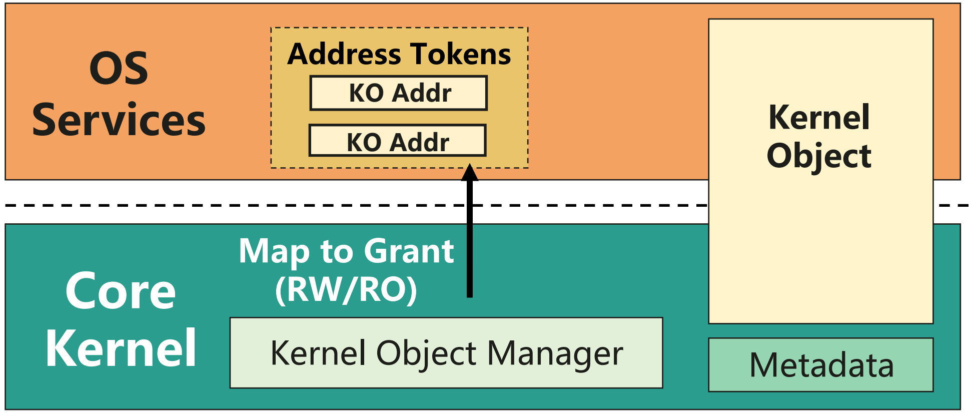
Address Tokens Use Addresses as Tokens

- Token: Slot id -> **Address**



Address-token-based *Access Control in HongMeng Kernel*

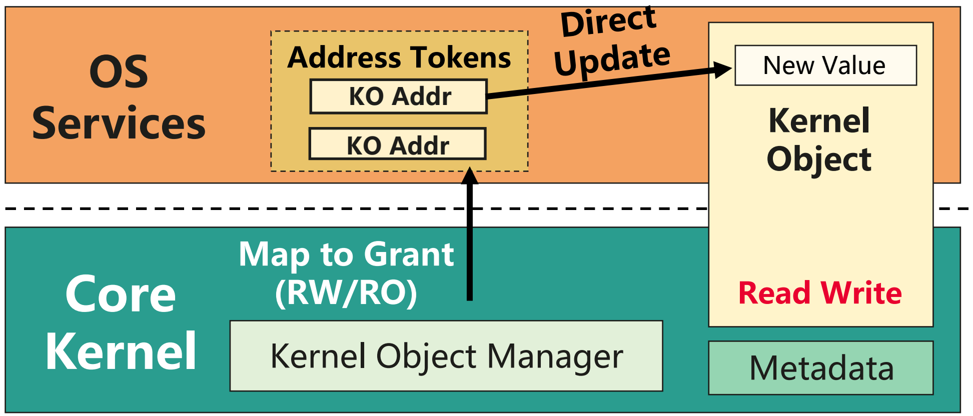
Mapping to OS Services for Granting Objects



***Address-token-based** Access Control in HongMeng Kernel*

- Token: Slot id -> **Address**
- Map to grant, Unmap to revoke
- **Read object directly** w/o kernel involvement

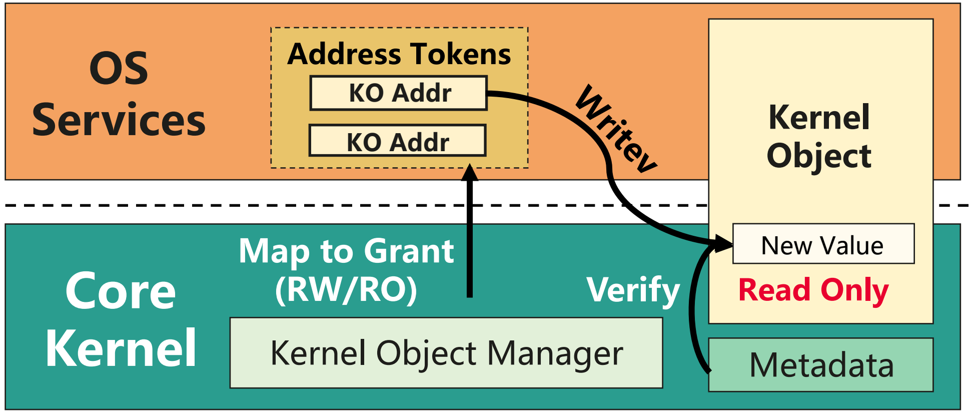
Bypassing the Core Kernel When Updating RW Objects



***Address-token-based** Access Control*
in HongMeng Kernel

- Token: Slot id -> **Address**
- Map to grant, Unmap to revoke
- **Read object directly** w/o kernel involvement
  - **RW**: Direct accesses to restricted obj (for security)

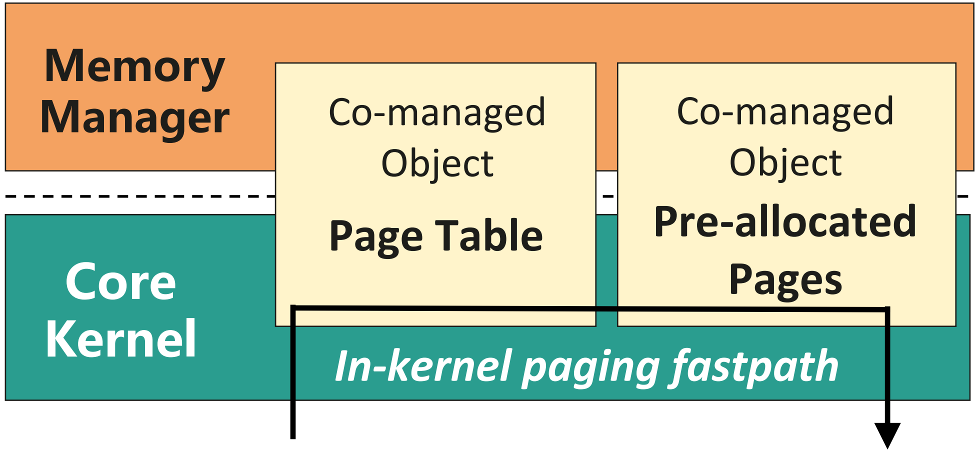
Eliminating Serialization When Updating RO Objects



***Address-token-based** Access Control in HongMeng Kernel*

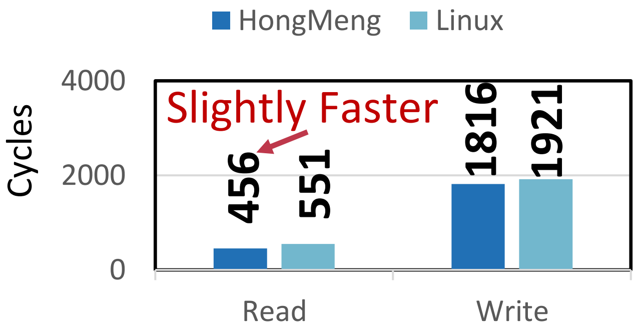
- Token: Slot id -> **Address**
- Map to grant, Unmap to revoke
- **Read object directly** w/o kernel involvement
  - **RW**: Direct accesses to restricted obj (for security)
  - **RO**: Writev syscall + verify permission in kernel

Address Tokens Enable Efficient Objects Co-management



Policy-free Kernel Paging
Enabled by Address Tokens

- Enables efficient co-management
  - Performant **policy-free kernel paging**



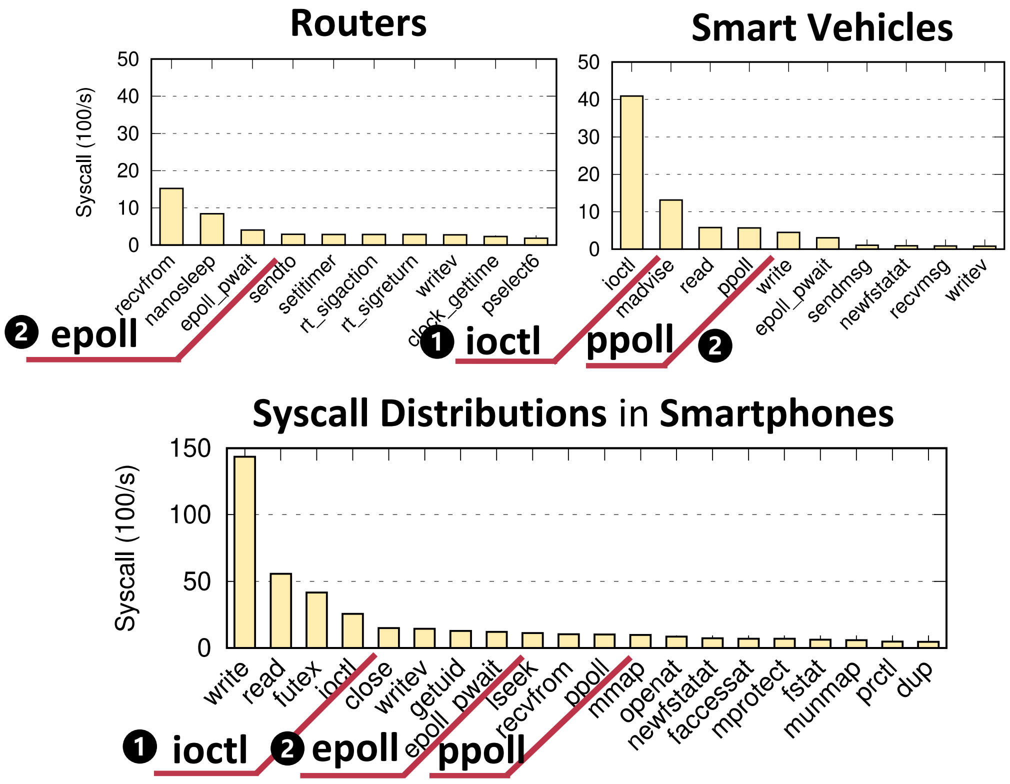
Latency of paging

  - Efficient implementations of functions like **poll**

Revisiting Conventional Wisdoms in Microkernels

	Conventional Wisdoms	Problems	HongMeng Kernel
Minimality	Minimal Core Kernel	N/A	Retains Minimality
IPC/Isolation	All Services at Userspace	Overly Strong Considering High IPC Frequency	Isolation Classes
Service Partitioning	Static Multi-server	State Double Bookkeeping	Flexible Composition
Access Control	Capabilities	Hide Kernel Objects	Address Tokens
Interface	**(subset-)POSIX**	**Require More than POSIX**	**ABI-compliant Shim**
Drivers			

POSIX-compliant is Not Enough for an Open Ecosystem



In-production Typical Usage

❶ Eco-compatible Requires More Than POSIX

Use ioctl to extend system API

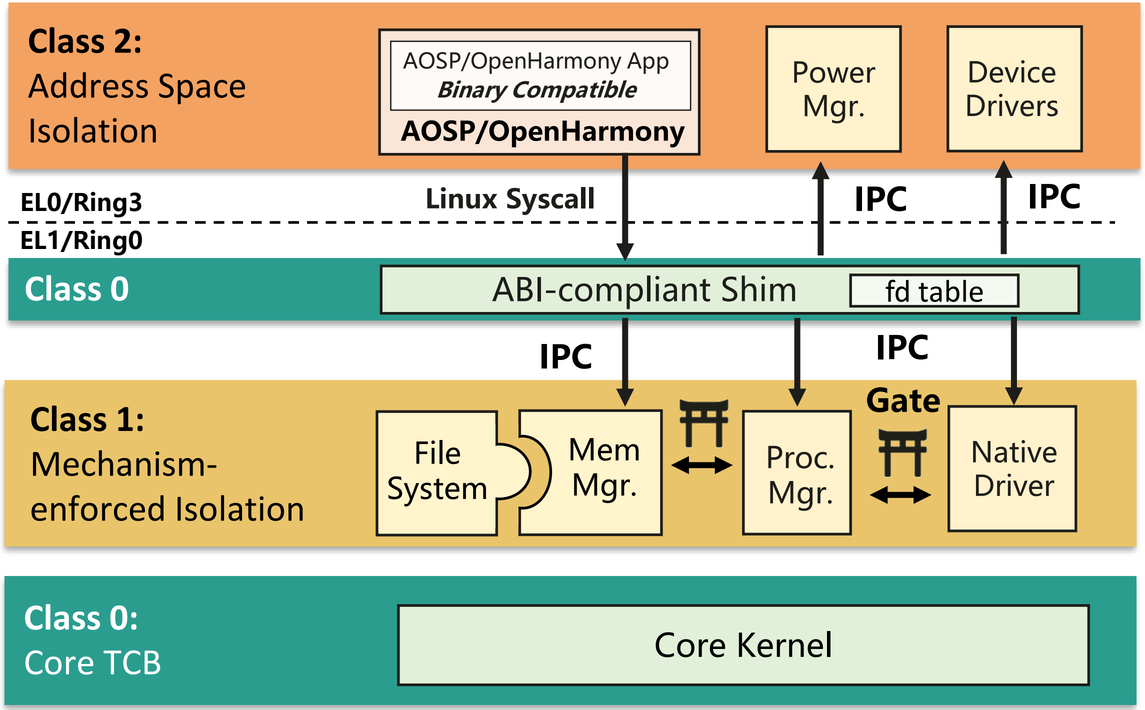
❷ No Central Repository for Global States

For example, file descriptor (fd), poll list Distributed in different services

Hard to implement epoll, fork efficiently

Revisiting Interface

Achieving Linux Binary Compatible via ABI-compliant Shim



- **ABI-compliant Shim**

  Redirect Linux syscall

- **Central Repository**

  For global states like the file descriptor

  Efficient Implementation of poll

- **Supports Complex Frameworks**

  OpenHarmony & AOSP

Revisiting Conventional Wisdoms in Microkernels

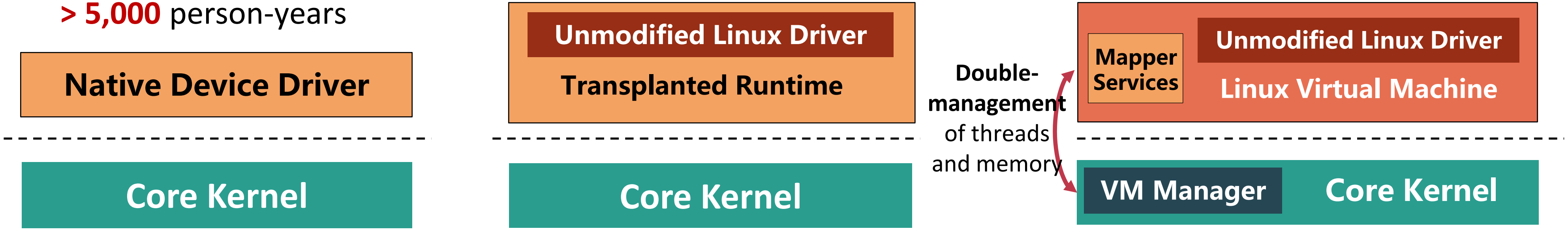
	Conventional Wisdoms	Problems	HongMeng Kernel
Minimality	Minimal Core Kernel	N/A	Retains Minimality
IPC/Isolation	All Services at Userspace	Overly Strong Considering High IPC Frequency	Isolation Classes
Service Partitioning	Static Multi-server	State Double Bookkeeping	Flexible Composition
Access Control	Capabilities	Hide Kernel Objects	Address Tokens
Interface	(subset-)POSIX	Require More than POSIX	ABI-compliant Shim
Drivers	**VM/Transplanting**	**Require Performant Reuse**	**Driver Containers**

Massive Drivers Require Performant Reuse

700+ drivers are required by vehicles and smartphones to function correctly

Rewriting Drivers | **Transplanting Runtime** [1] | **VM-based Methods** [2]

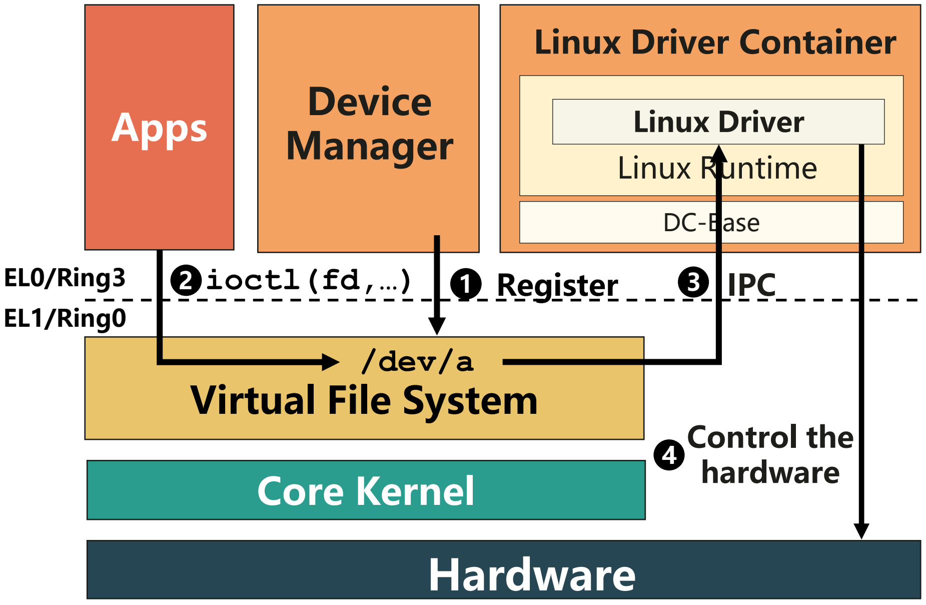
> 5,000 person-years



Engineering Issue | **Compatibility & Engineering Issues** | **Performance Issue**

[1] *Weisbach, Hannes. DDEKit Approach for Linux User Space Drivers*
[2] *LeVasseur, Joshua, et al. Unmodified Device Driver Reuse and Improved System Dependability via Virtual Machines. OSDI ’04.*

How Do Driver Containers Reuse Linux Device Drivers?



***Driver Containers** in HongMeng*

- **Provides a Linux runtime at userspace** for unmodified Linux drivers

  Similar to LKL/UML [1,2] but targets driver reuse
- **DC-base** redirects **necessary KAPIs**

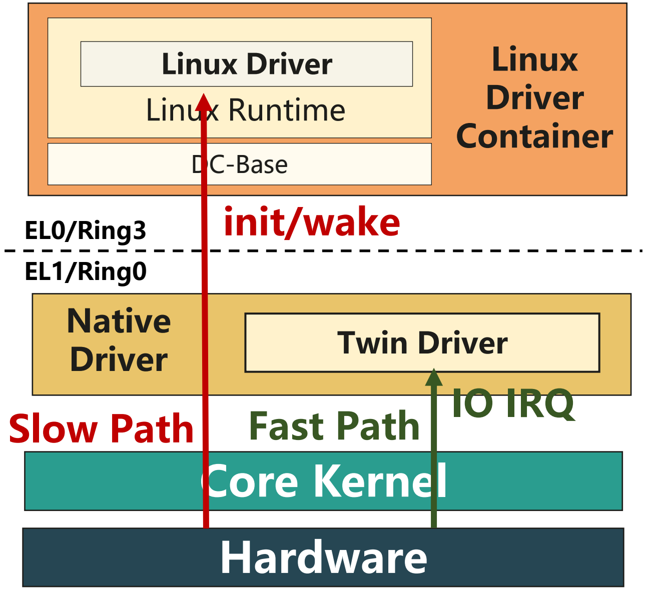
  Kthread, Kernel Memory

  Forbids double management
- **Minor modifications** to upgrade (< 100 changes from 4.19 to 5.10)

[1] *Octavian Purdila, et al. LKL: The Linux kernel library. RoEduNet '10*

[2] *Jeff Dike. User-mode Linux.*

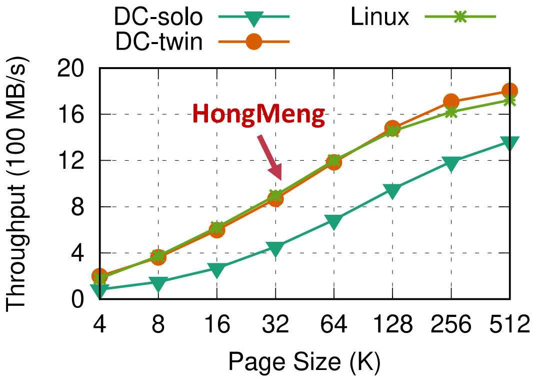
Improving Performance via Control/Data Plane Separation



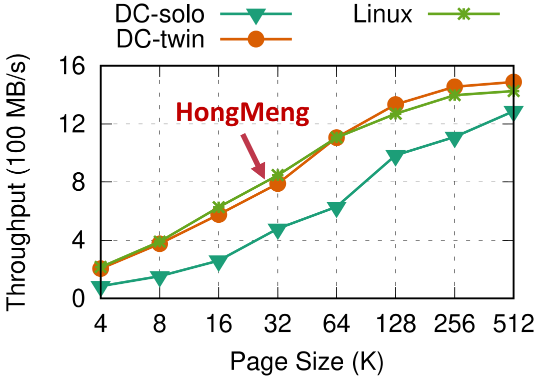
***Driver Containers** in HongMeng*

- Cumbersome Procedure (init/wake) in userspace
- **I/O requests** to **rewritten twin drivers**

Can be configured to use weaker isolation



BIO Read



BIO Write

Outline

- ✓ Revisiting Microkernel for Going General
- ✓ **Implementation and Performance**
- ✓ Lessons and Experiences

Implementation and Deployment of HongMeng Kernel

- Core kernel **~90 thousands LoC**, OS services over 1 million LoC, written in subset C
- Deployed in **tens of millions of devices**

OS Kernel for Routers/Switches

OS Kernel for Smart Vehicles

OS Kernel for Smartphones/Tablets

- Same codebase with different configurations
- Certified with CC-EAL6+ (security) and ASIL-D (safety)

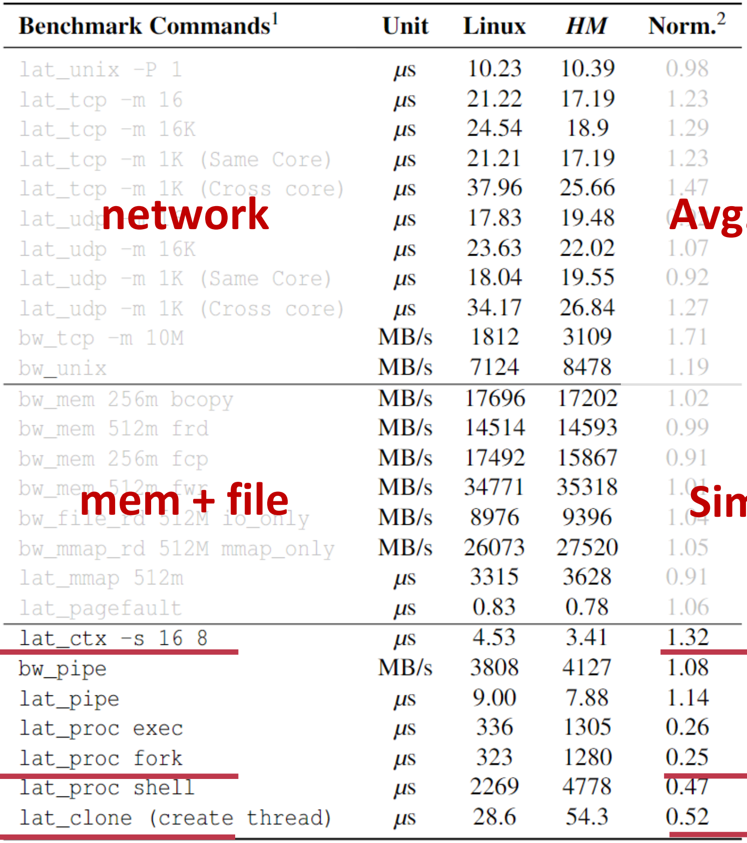
Performance Comparison in Micro-benchmark

Q1: How well does HongMeng perform compared to Linux in **micro-benchmark**?

LMBench Results

- Improved network
- Improved context switches
- Similar memory and file operations
- Issues with Fork/Clone

Can be accelerated through parallelism



Benchmark Commands[1]	Unit	Linux	*HM*	Norm.[2]
`lat_unix -P 1`	µs	10.23	10.39	0.98
`lat_tcp -m 16`	µs	21.22	17.19	1.23
`lat_tcp -m 16K`	µs	24.54	18.9	1.29
`lat_tcp -m 1K (Same Core)`	µs	21.21	17.19	1.23
`lat_tcp -m 1K (Cross core)`	µs	37.96	25.66	1.47
`lat_udp -m 16`	µs	17.83	19.48	0.92
`lat_udp -m 16K`	µs	23.63	22.02	1.07
`lat_udp -m 1K (Same Core)`	µs	18.04	19.55	0.92
`lat_udp -m 1K (Cross core)`	µs	34.17	26.84	1.27
`bw_tcp -m 10M`	MB/s	1812	3109	1.71
`bw_unix`	MB/s	7124	8478	1.19
`bw_mem 256m bcopy`	MB/s	17696	17202	1.02
`bw_mem 512m frd`	MB/s	14514	14593	0.99
`bw_mem 256m fcp`	MB/s	17492	15867	0.91
`bw_mem 512m fwr`	MB/s	34771	35318	1.01
`bw_file_rd 512M io_only`	MB/s	8976	9396	1.04
`bw_mmap_rd 512M mmap_only`	MB/s	26073	27520	1.05
`lat_mmap 512m`	µs	3315	3628	0.91
`lat_pagefault`	µs	0.83	0.78	1.06
`lat_ctx -s 16 8`	µs	4.53	3.41	1.32
`bw_pipe`	MB/s	3808	4127	1.08
`lat_pipe`	µs	9.00	7.88	1.14
`lat_proc exec`	µs	336	1305	0.26
`lat_proc fork`	µs	323	1280	0.25
`lat_proc shell`	µs	2269	4778	0.47
`lat_clone (create thread)`	µs	28.6	54.3	0.52

network

Avg. +21%

mem + file

Similar

Context Switches

+32%

Fork

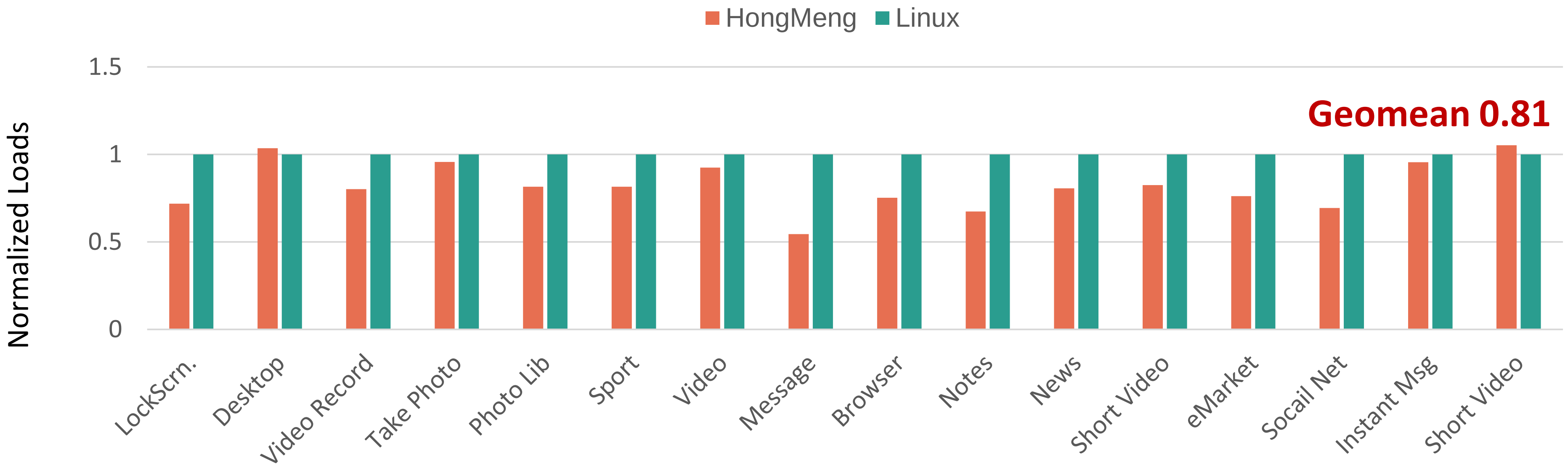
-75%

Clone

-48%

Load Comparison in Typical Use Case

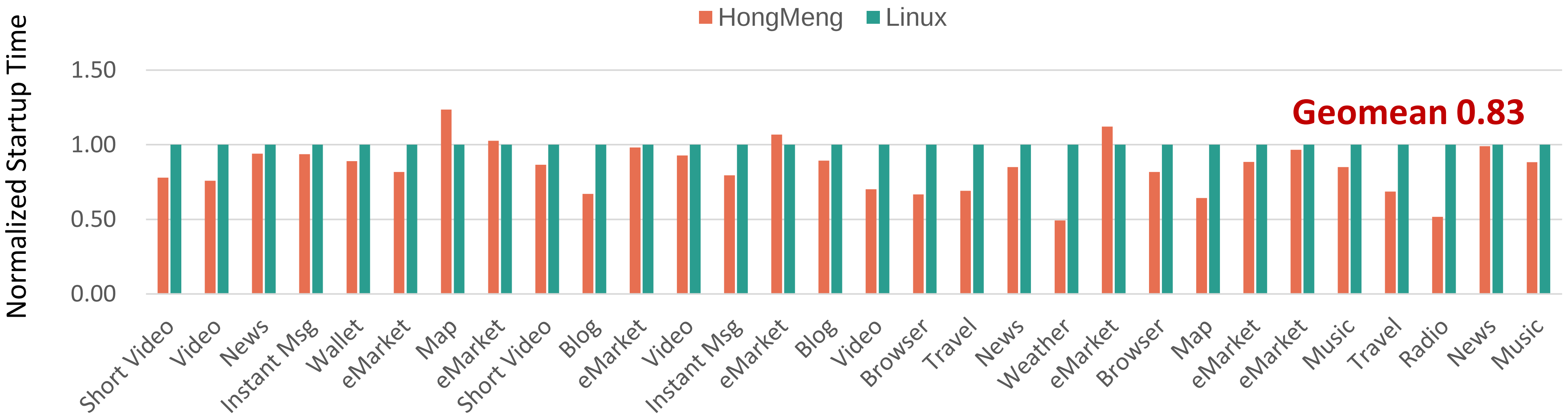
Q2: Will microkernel architecture lead to **a higher load**?



19% lighter loads in typical scenarios (the less the better)

End-to-End Comparison of Startup Time and Frame Drops

Q3: How does HongMeng perform in **real-world scenarios** compared to Linux?



17% shorter app startup time in top30 applications (the less the better)

10% less frame drops in typical usage lasting 24 hours

Outline

- ✓ Revisiting Microkernel for Going General
- ✓ Implementation and Performance
- ✓ **Lessons and Experiences**

Being Compatible at First, then Nativize Gradually

Being compatible is a crucial first step for commercial deployment

Products require unified codebase for various platforms

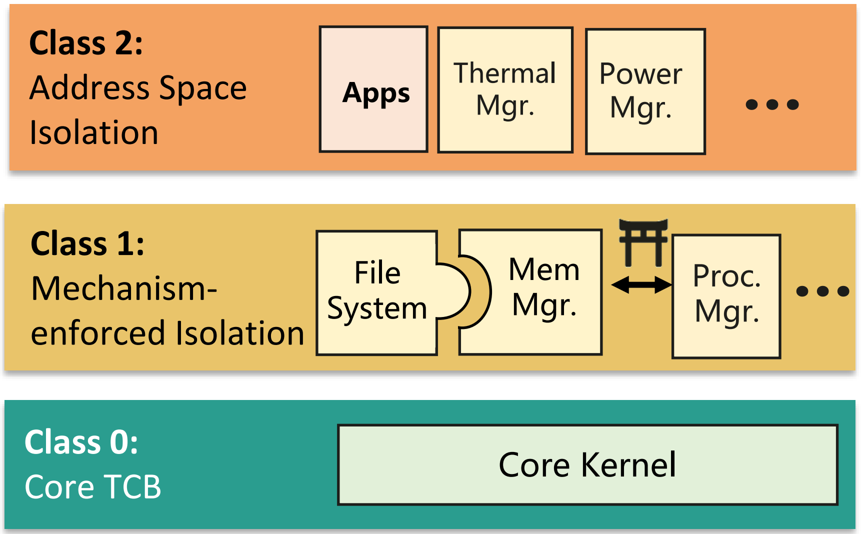
OpenHarmony

Linux

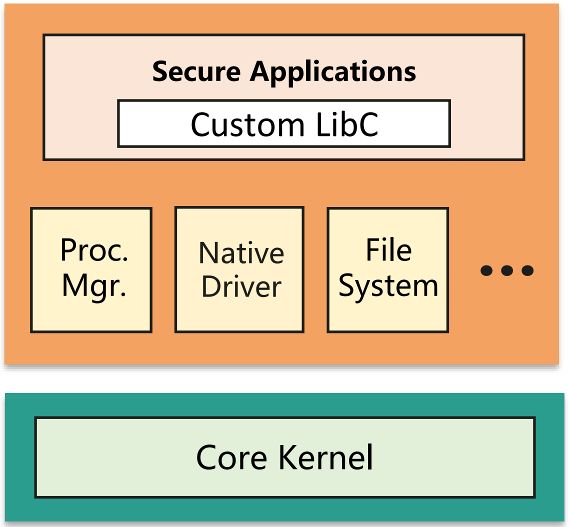
HongMeng

Third-party apps & drivers are distributed in binary form

Configurable Composition is Critical for Cross-Scenario Deployment



***HongMeng Kernel** in Smartphones*



***HongMeng Kernel** in Secure OS*
Trusted Execution Environment (TEE)

• • •

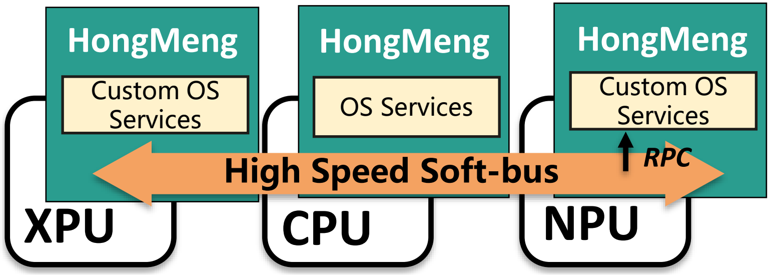
Other Scenarios

*Same codebase with **different configurations***

Future Work: Accommodating Heterogenous Hardware

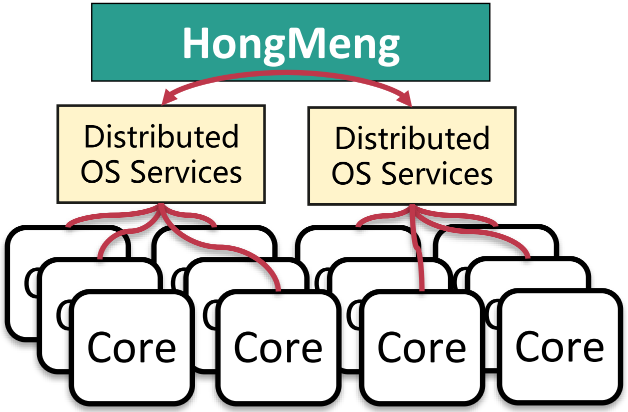
Heterogenous Processing Unit

Serves as a solid starting point for applying heterogenous-oriented architectures [1,2] in production heterogeneous systems



Non-Cache-Coherence Manycore

Scales out software with distributed/partitioned OS services



[1] Baumann, Andrew, et al. The multikernel: a new OS architecture for scalable multicore systems. SOSP '09
[2] Shan, Yizhou, et al. LegoOS: A disseminated, distributed {OS} for hardware resource disaggregation. OSDI '18

Conclusions

- **HongMeng** general-purpose microkernel
- **Retaining Minimality**

  Minimal core kernel with decoupled, well-isolated, least-privileged OS services
- **Prioritizing Performance**

  Structural supports includes isolation classes, flexible composition, and address tokens
- **Maximizing Compatibility**

  Achieves Linux ABI compliance and performant driver reuse
- Deployed in production and typically with **improved performance**