ORC: Increasing Cloud Memory Density via Object Reuse with Capabilities

17th USENIX Symposium on Operating Systems Design and Implementation(OSDI'23)

汇报人：易正烨

1.1 Background-memory density

- 高内存密度(High Memory Density)
  - 云服务提供商希望在有限的内存中尽可能多地部署用户实例

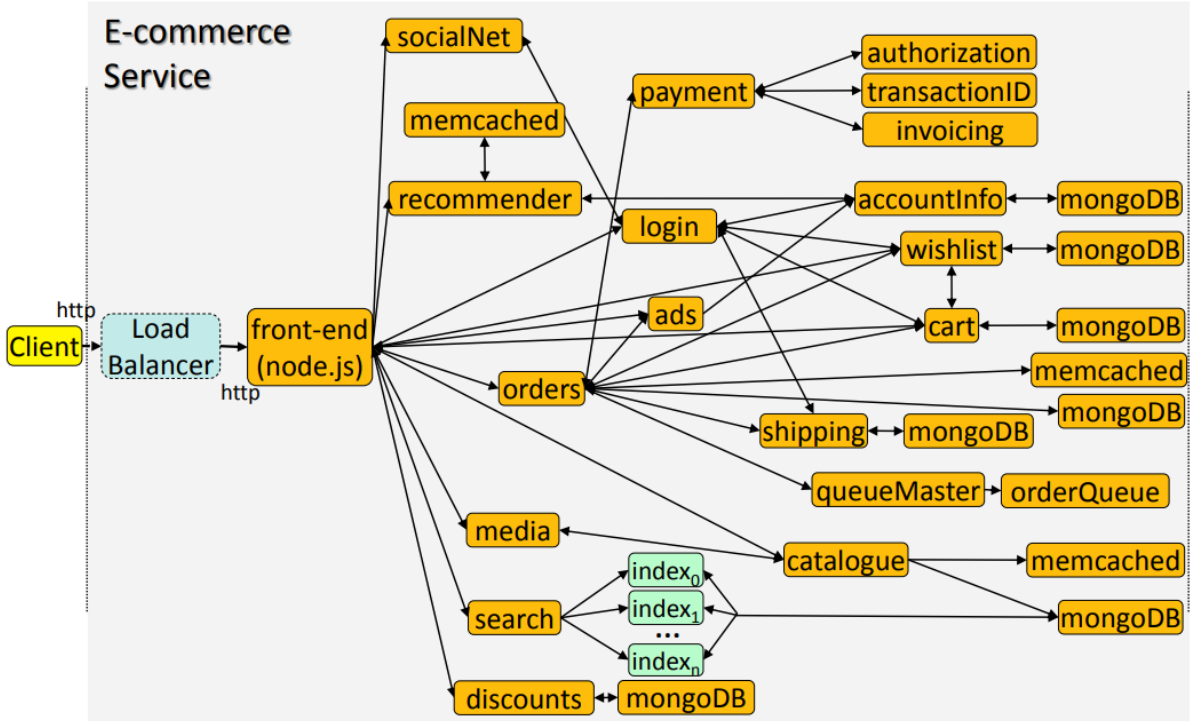
- 内存冗余删除(Memory De-Duplicating)
  - 删除同一机器上不同虚拟机、容器和进程间内容相同的内存页面

- 隔离性(Isolation)
  - 云中需要通过隔离来保证不同实例间的安全运行，有ISA级别强隔离、软件层级隔离等

1.2 Background-memory duplication

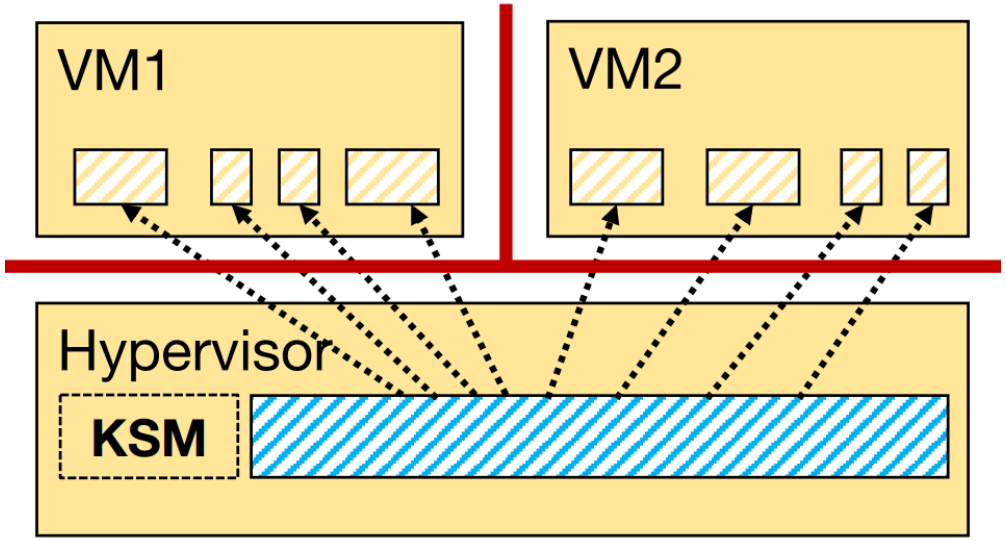
- **内存冗余在云中广泛存在**
  - 不同虚拟机/容器间可能是相同的OS内核
  - 不同虚拟机/容器上可能运行相同的应用
  - 不同应用可能在相同的框架上构建

- **内存冗余在微服务中广泛存在**
  - 不同进程间存在重复冗余的实例，而且往往单个实例就很大(如右图中，存在7个MangoDB实例，一个实例占到了640MB)



1.3 Background-memory share

- **操作系统托管的内存共享—容器**
  - OS为共享内存提供了用户空间抽象，加载器可以跨进程映射相同的二进制对象，利用用户级信息在加载时消除冗余，无额外运行时开销。
  - **破坏了隔离性，复杂的TCB有安全性风险**



- **Hypervisor托管的内存共享—虚拟机**
  - 通过ISA上暴露的虚拟化接口，在运行时定期扫描内存识别冗余页面，并将冗余的物理页面重新映射到新的物理页面上。
  - 典型代表：VMWare ESX、KVM

  - **页面扫描影响平均性能和尾延迟**
  - **无法获得应用语义，只能依赖扫描**
  - **易受到跨虚拟机的侧信道攻击**

1.4 Background-memory capability

- **CHERI：Capability Hardware Enhanced RISC Instructions**
  - 使用新指令、寄存器和其他硬件原语扩展了传统的ISA
  - 可实现细粒度内存共享与安全保护、可扩展的软件隔离

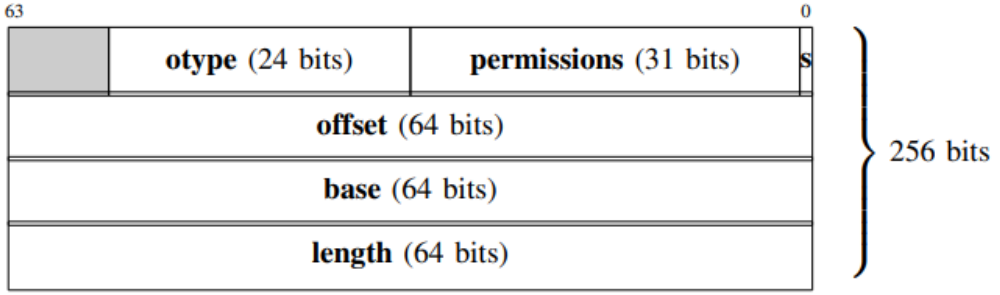
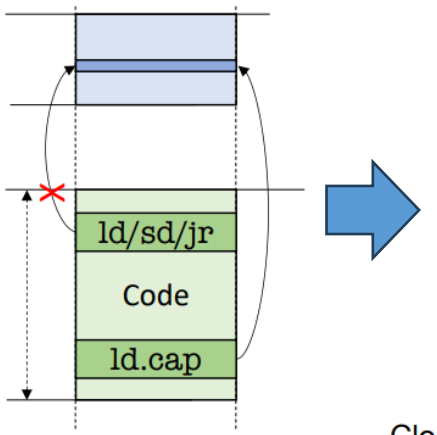
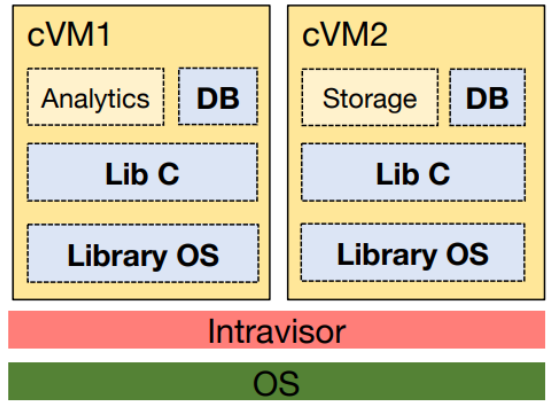


Fig. 3. ISA-level representation of a 256-bit CHERI capability

- **Capability ← Fat Pointer**
  - 原始内存地址+访问权限+边界+指针有效性元数据
  - 一种不可伪造的授权标记，授予在执行上下文中执行操作的特定权限
  - **可以通过使用具有相同边界的Capability来构建了一个隔离的分区**





Cloud stack built on top of cap-based compartments
- Lightweight isolation and sharing
- Shared single address space → Control over components

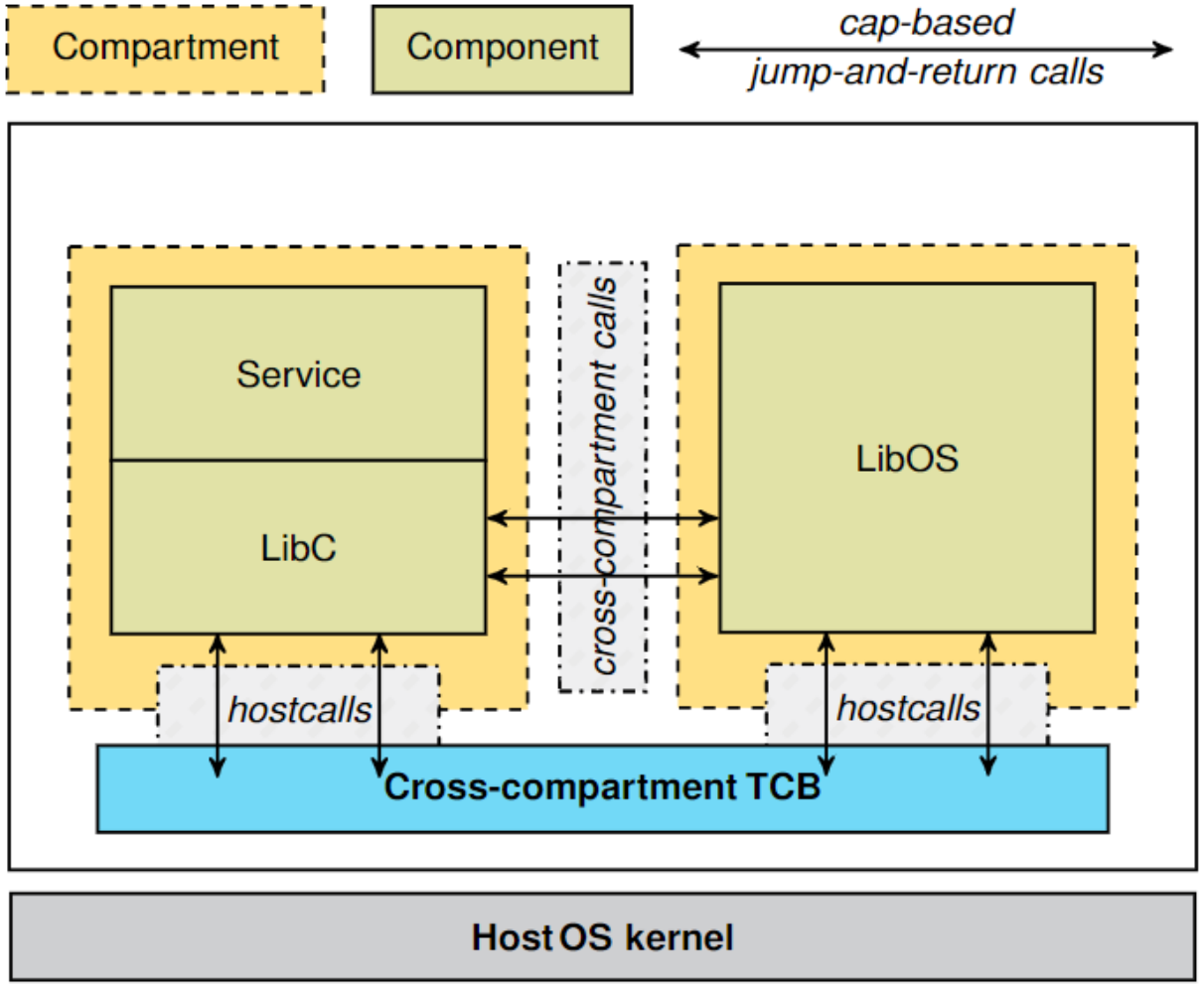
[1] R. N. M. Watson *et al.*, 'CHERI: A Hybrid Capability-System Architecture for Scalable Software Compartmentalization', in *2015 IEEE Symposium on Security and Privacy, SP 2015, San Jose, CA, USA, May 17-21, 2015*, 2015, pp. 20–37.
[2]V. A. Sartakov, L. Vilanova, D. M. Eyers, T. Shinagawa, and P. R. Pietzuch, 'CAP-VMs: Capability-Based Isolation and Sharing in the Cloud', in *16th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2022, Carlsbad, CA, USA, July 11-13, 2022*, 2022, pp. 597–612.

1.5 Background-Efficiency and security

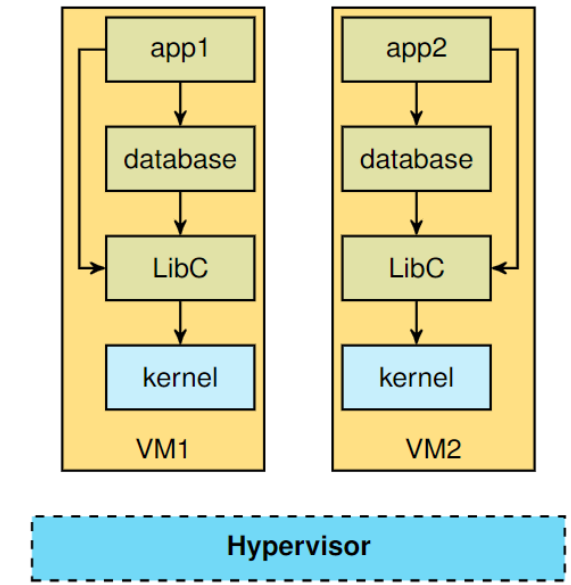
- **最小可信基+强隔离**
  - 保证租户使用同时减小攻击界面面
- **低性能开销**
  - 不应影响系统其他任务正常执行
- **细粒度共享**
  - 支持共享任意大小对象，共享正确性高

↓

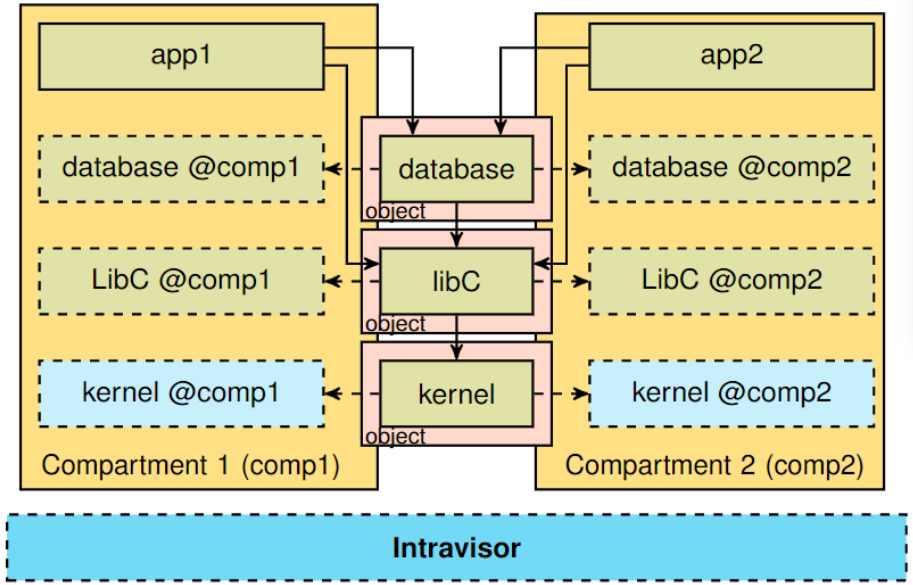
- ORC中仅对**代码**和**只读数据**共享



2.1 Design



(a) VM-based stack with intrinsic object duplication

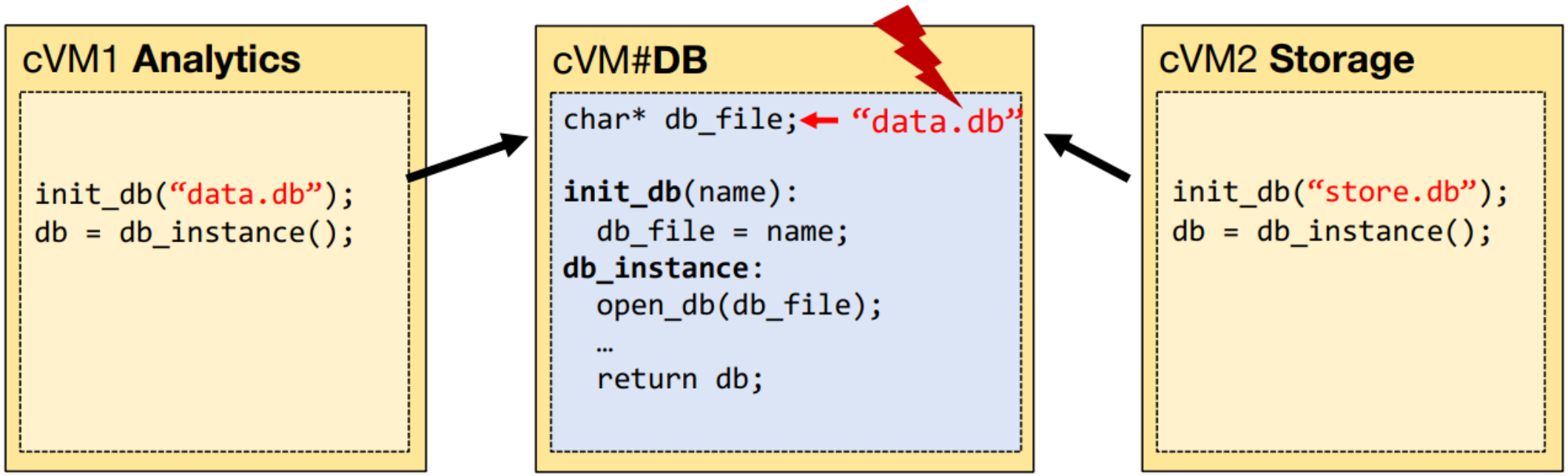


(b) ORC-based stack with explicitly shared objects

■ **ORC：Object Reuse with Capabilities**

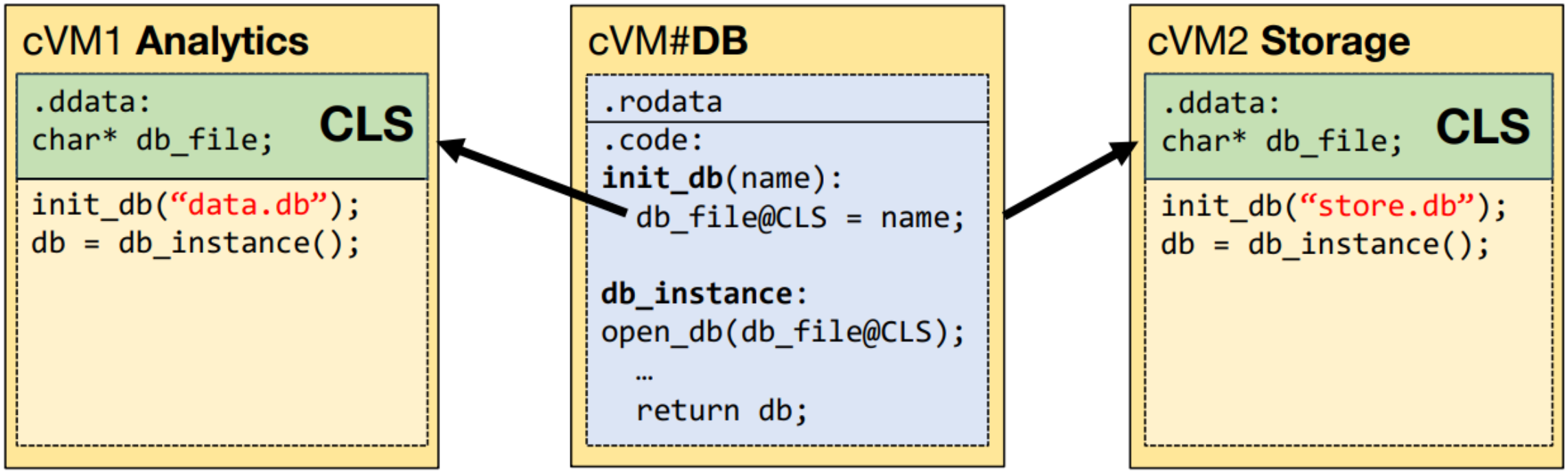
- 利用 CHERI 架构特性实现软件隔离和显式二进制对象共享，使物理内存密度提升的同时消除重复数据删除的性能开销
- 隔离区使用共享页表部署，隔离区内OS实例只能访问自己的地址空间，通过capability跨区访问共享对象
- 共享对象运行时无法修改，隔离区通过编译来构建共享对象的本地可修改实例

2.2 Design



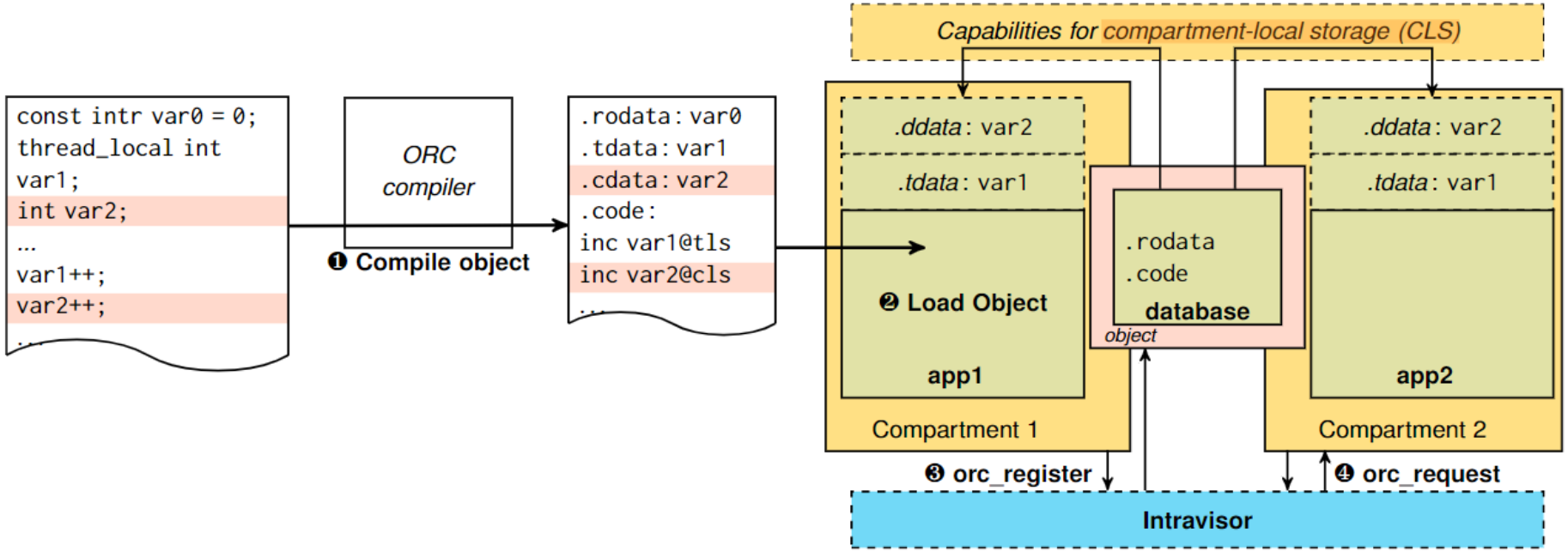
- 共享的对象在不同的示例中具有不同状态，新来实例具备不同状态时无法安全共享

2.3 Design-compartment local storage



- ORC将全局共享对象的状态保存在隔离区的CLS中，作为私用副本
- 代码和只读对象则为共享对象
- CLS借鉴了TLS(Thread local storage)，基于LLVM pass实现，即对编译后的中间表示(IR)上做进一步操作，把对非共享变量的引用替换为函数__cls_get_addr

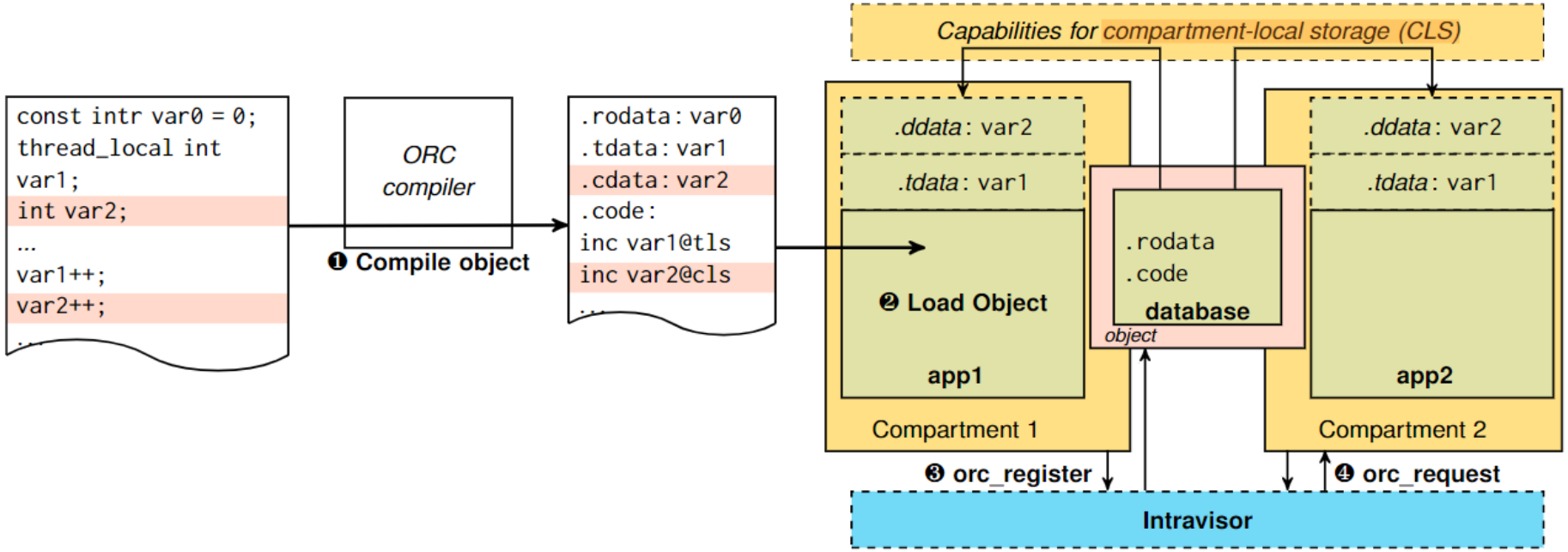
2.4 Architecture



■ **ORC工作流程1**

- ORC编译潜在对象
  - 将对象的可写状态移动到隔离区的本地内存空间中(CLS)，如图中的var2
  - 代码、和常量作为共享对象，如图中的var0
- 隔离区使用自己的ld.so为CLS中对象内容分配内存空间并加载

2.5 Architecture



■ ORC工作流程2

- 隔离区向Intravisor注册共享对象(orc_register)
  - 将对象的capability和代码引用列表传递给Intravisor
  - Intravisor将对象复制到自己创建得隔离区中，并其计算哈希值

■ ORC工作流程3

- 其他隔离区向Intravisor请求共享对象(orc_request)
  - Intravisor会对比共享对象Hash值和请求对象Hash值
  - 如果相同则释放该对象的Capability，请求的隔离区在CLS中分配内存，通过ld.so加载对象内容并引用

3.1 Evaluation

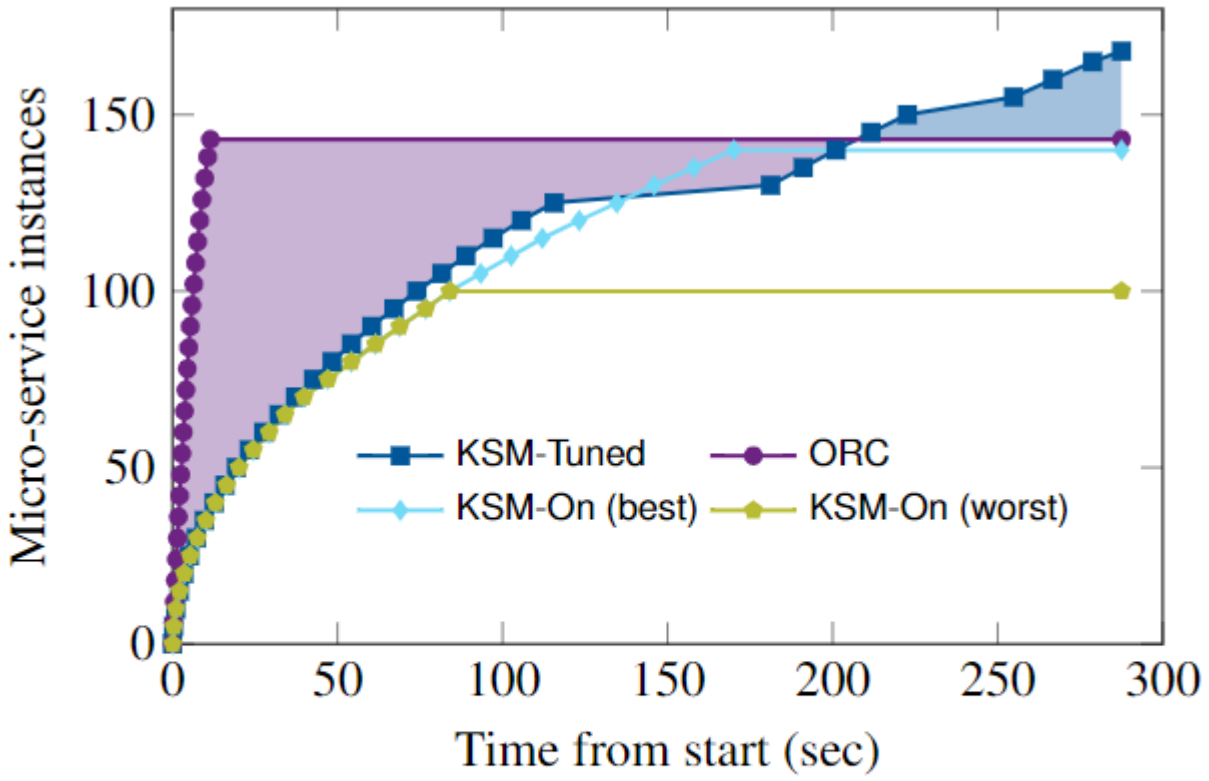


Fig. 4: Throughput over time when de-duplication of the video transcoding micro-service happens (The shaded area indicates the difference between ORC and KSM-Tuned, which is the best-performing KSM configuration.)

- 工作负载：实时视频转码微服务，使用Ffmped执行转码
- 测试平台：搭载Armv8-A CPU的Morello开发板，支持CHERI架构
- KSM-Tuned：每60s检查可用内存占比，低于20%时工作
- KSM-On：在最短时间内实现良好的内存效率
- 结果：ORC 中的实例创建和重复数据删除上比 KSM 的效率更高。尽管最终KSM-Tuned内存密度好于ORC，但是达峰时间太长，不利于云中实时任务的处理。

3.2 Evaluation

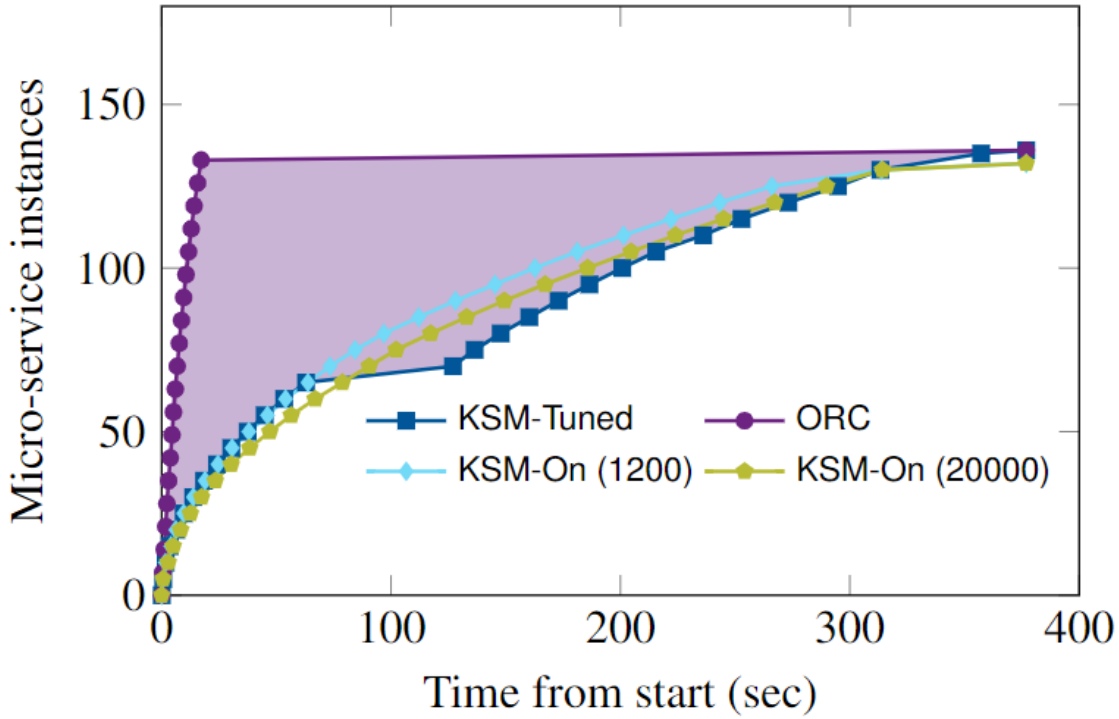


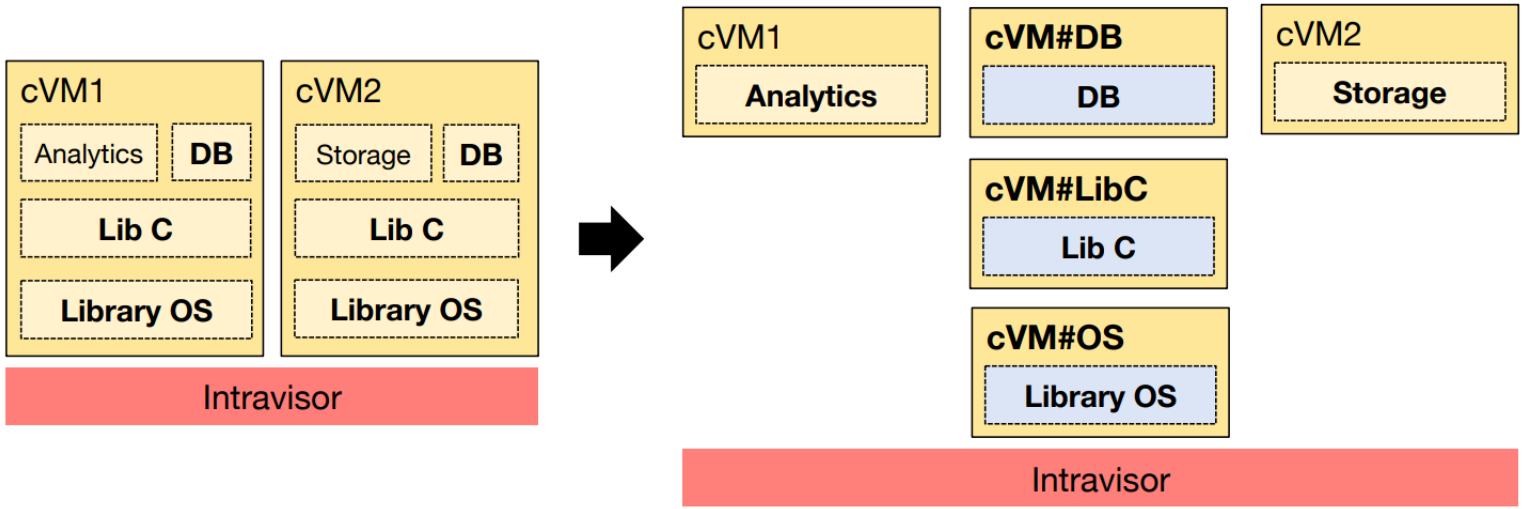
Fig. 5: Throughput over time when de-duplication of the video transcoding micro-service happens using larger binaries (The shaded area indicates the difference between ORC and KSM-Tuned.)

Tab. 1: Impact of memory de-duplication on Redis tail latency

	set requests		get requests	
	p50	p99	p50	p99
Linux	0.5 ms	9.7 ms	0.5 ms	9.3 ms
Linux+KSM	0.5 ms	20.9 ms	0.5 ms	20.8 ms
CheriBSD	2.1 ms	3.7 ms	2.2 ms	3.6 ms
CheriBSD+CC	2.1 ms	4.2 ms	2.1 ms	4.3 ms
CheriBSD+CC+ORC	2.1 ms	4.9 ms	2.1 ms	4.8 ms

- KSM页面重复数据删除开销导致p99延迟增加2倍以上;
- ORC的二进制对象共享将开销降低到12%-17%。

4 Conclusion



- 虚拟机/容器的内存中有许多相似/相同的内容
- 传统的基于MMU/Hypervisor的管理方法效率低：页面扫描和去重开销较大
- ORC实现了带有语义级别的对象复用和memory capability的内存去重，具备更小的开销和更高的吞吐量

Q&A

Thanks!