JD.COM 京东

容器底层技术

田琪

JD.COM 京东

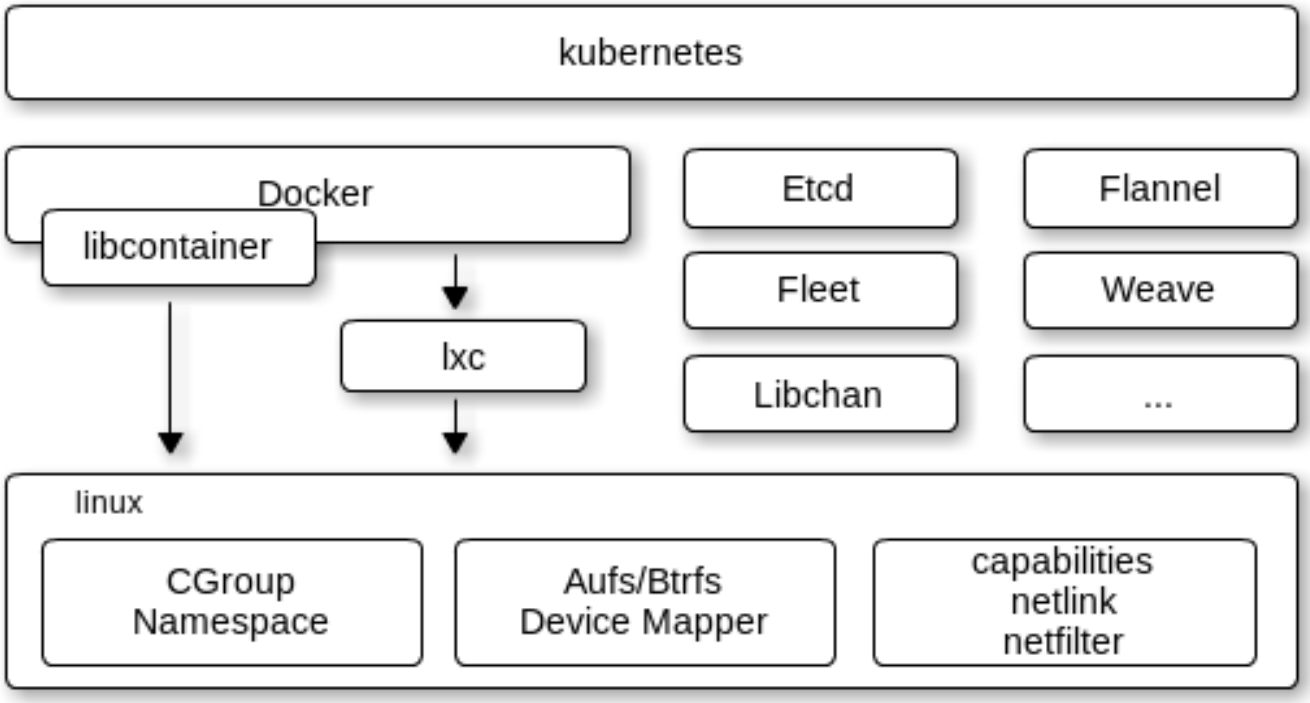
Agenda

JD.COM 京东

- 容器系统整体架构
- 内核 Namespace 机制
- 内核 CGroup 机制
- 内核 Device Mapper 机制
- DM Thin Provision Target 应用
- Docker Registry 及镜像存储
- 关于 Go

容器系统整体架构

JD.COM 京东

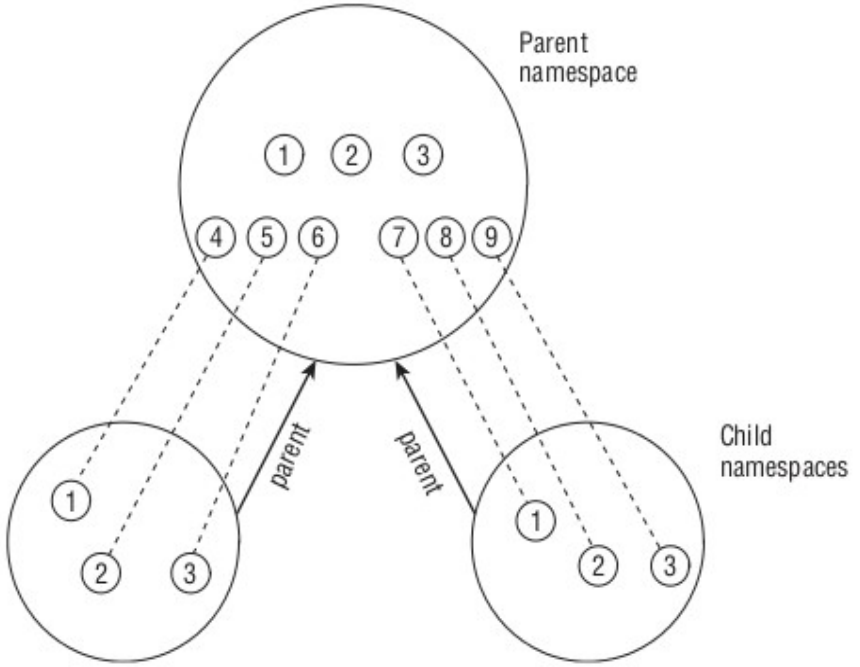


- 提供进程级别的资源隔离
- 为进程提供不同的命名空间视图
- 无 hypervisor 层，区别于 KVM,Xen 等虚拟化技术
- 从 Kernel 2.4 版本引入 mnt namespace~3.8 引入 user namespace 仍然持续发展中

Namespace 概念及当前支持的种类

JD.COM 京东

- mnt (Mount points)
- pid (Processes)
- net (Network stack)
- ipc (System V IPC)
- uts (Hostname)
- user (UIDS)



JD.COM 京东

- 创建新进程及 namespace

```
int clone(int (*fn)(void *), void *child_stack,
          int flags, void *arg, ...
          /* pid_t *ptid, struct user_desc *tls, pid_t *ctid */ );
```

- 加入当前进程到新建 namespace 中

```
int unshare(int flags);
```

- 改变当前进程的 namespace

```
int setns(int fd, int nstype);
```

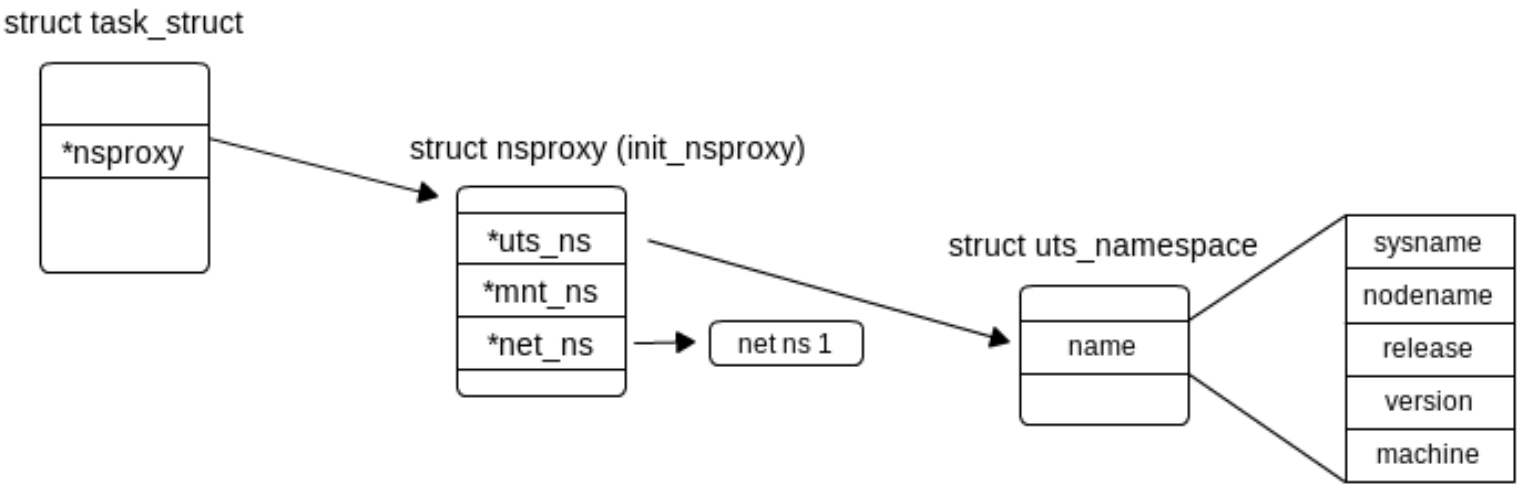
flags 取值及含义

JD.COM 京东

CLONE_NEWNS	2.4.19	CAP_SYS_ADMIN
CLONE_NEWUTS	2.6.19	CAP_SYS_ADMIN
CLONE_NEWIPC	2.6.19	CAP_SYS_ADMIN
CLONE_NEWPID	2.6.24	CAP_SYS_ADMIN
CLONE_NEWNET	2.6.29	CAP_SYS_ADMIN
CLONE_NEWUSER	3.8	No capability is required

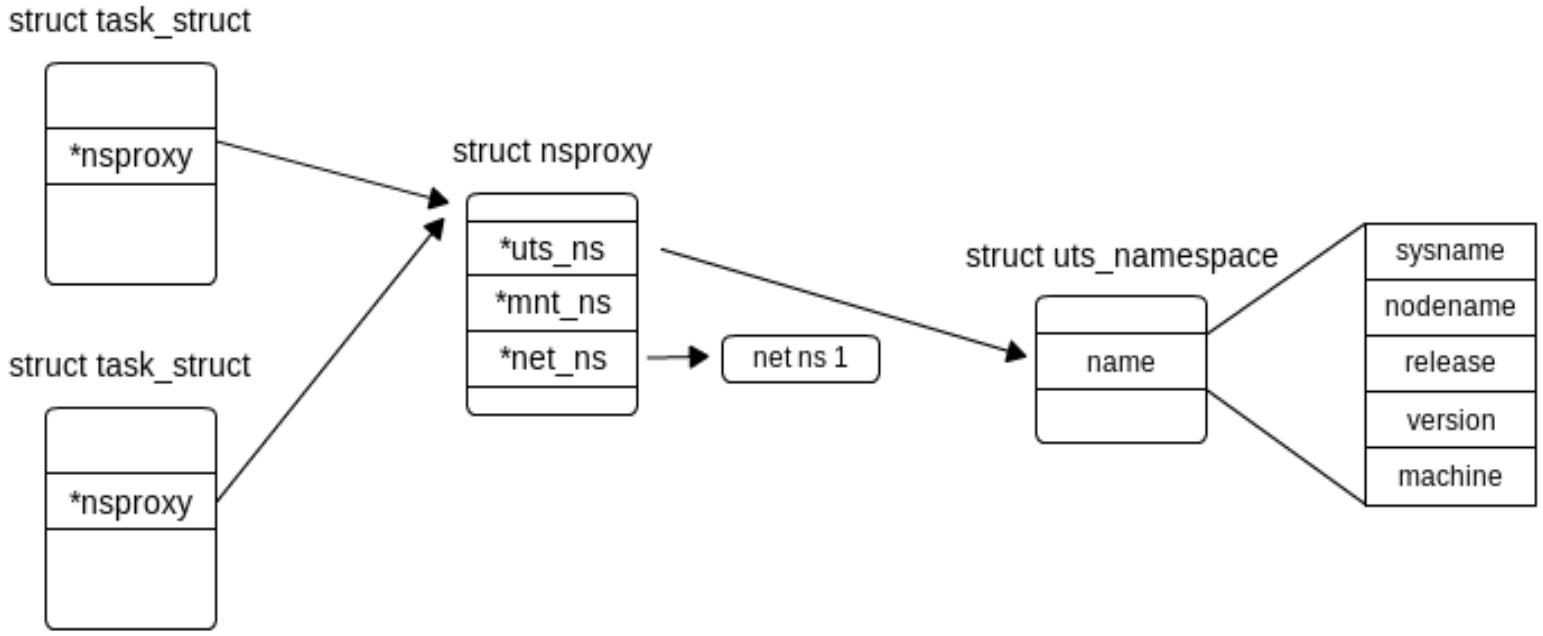
Namespace 内核实现 - 初始状态

JD.COM 京东



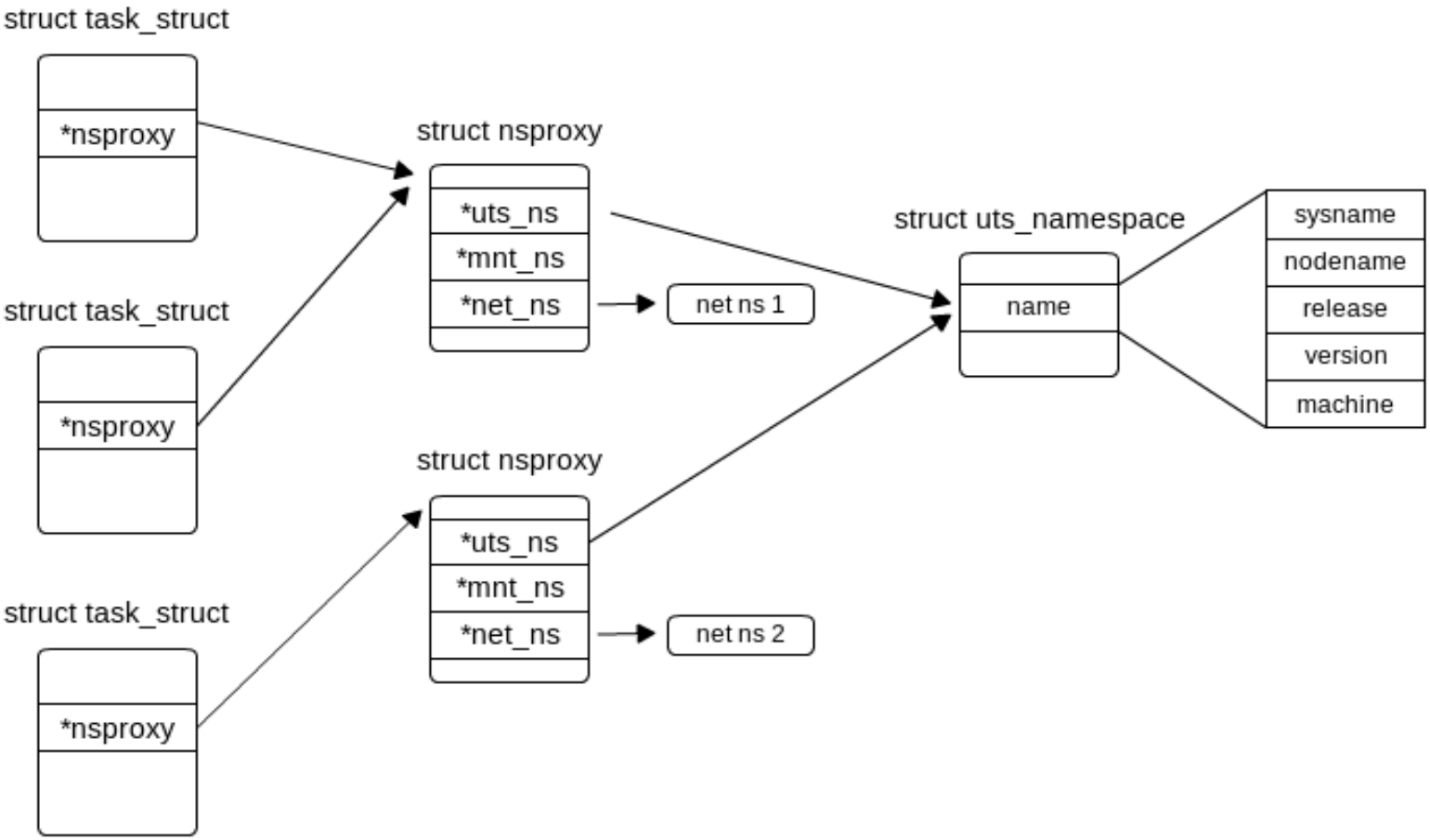
Namespace 内核实现 - after normal fork

JD.COM 京东



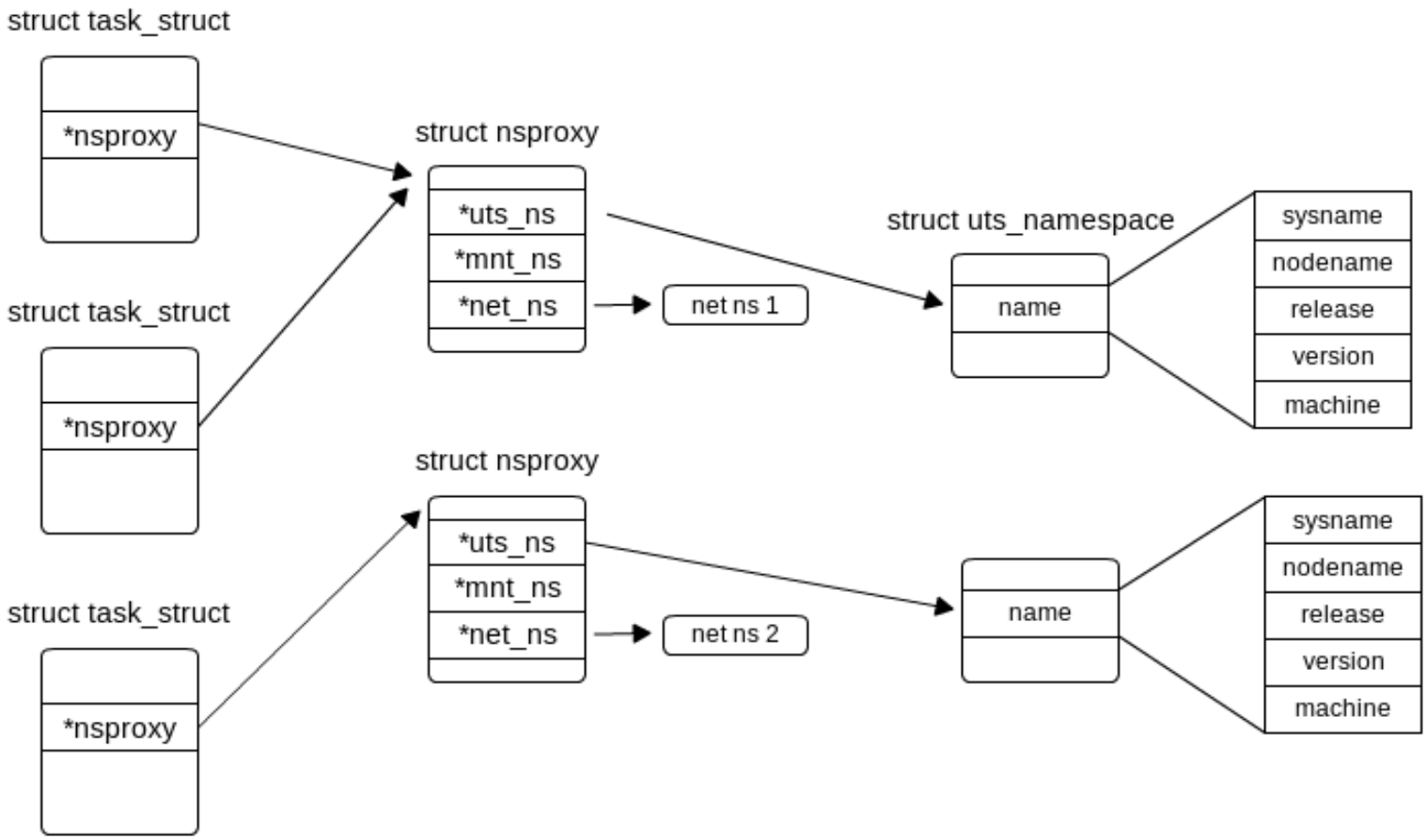
Namespace 内核实现 - clone a new ns

JD.COM 京东



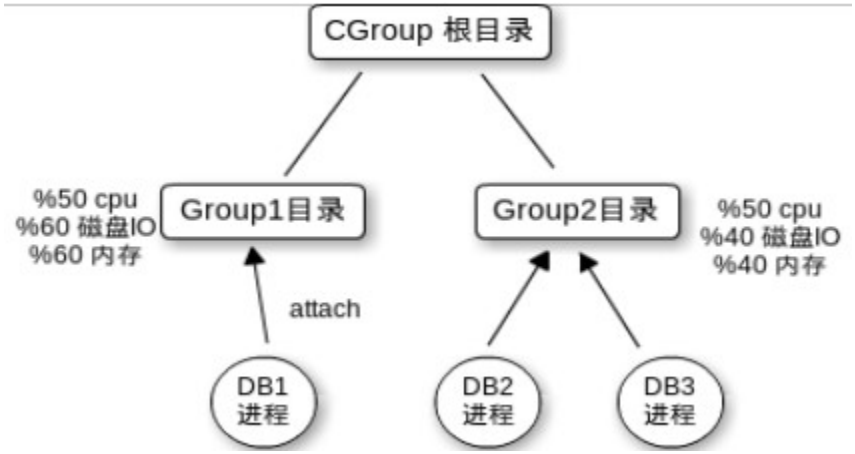
Namespace 内核实现 - after setns on task

JD.COM 京东



JD.COM 京东

- 提供进程的资源管理功能
- 资源管理主要涉及内存 ,CPU,IO 等
- 不依赖于 Namespace ，可单独使用
- 管理功能通过 VFS 接口暴露
- CGroups 提供通用框架，各子系统负责实现



CGroups 文件系统

JD.COM 京东

block子系统 CFQ策略文件

- blkio.weight_device
- blkio.weight
- blkio.leaf_weight_device
- blkio.leaf_weight
- blkio.time[_recursive]
- blkio.sectors[_recursive]
- blkio.io_merged[_recursive]
- blkio.io_queued[_recursive]
- blkio.io_wait_time[_recursive]
- blkio.io_serviced[_recursive]
- blkio.io_service_time[_recursive]
- blkio.io_service_bytes[_recursive]

cpu 子系统相关文件

- cpu.shares
- cpu.cfs_quota_us
- cpu.cfs_period_us
- cpu.rt_runtime_us
- cpu.rt_period_us
- cpu.stat

block子系统 throttle策略文件

- blkio.throttle.read_bps_device
- blkio.throttle.write_bps_device
- blkio.throttle.read_iops_device
- blkio.throttle.write_iops_device
- blkio.throttle.io_service_bytes
- blkio.throttle.io_serviced

block子系统框架产生文件

- blkio.reset_stats

cpu accounting 子系统相关文件

- cpuacct.usage
- cpuacct.stat
- cpuacct.usage_percpu

security 子系统文件

- devices.allow
- devices.deny
- devices.list

cpuset 子系统相关文件

- cpuset.cpu_exclusive
- cpuset.cpus
- cpuset.mem_exclusive
- cpuset.mem_hardwall
- cpuset.memory_migrate
- cpuset.memory_pressure
- cpuset.memory_pressure_enabled
- cpuset.memory_spread_page
- cpuset.memory_spread_slab
- cpuset.mems
- cpuset.sched_load_balance
- cpuset.sched_relax_domain_level

CGroup 框架相关文件

- cgroup.clone_children
- cgroup.event_control
- cgroup.procs
- notify_on_release
- release_agent
- tasks
- cgroup.sane_behavior

memory 子系统相关文件

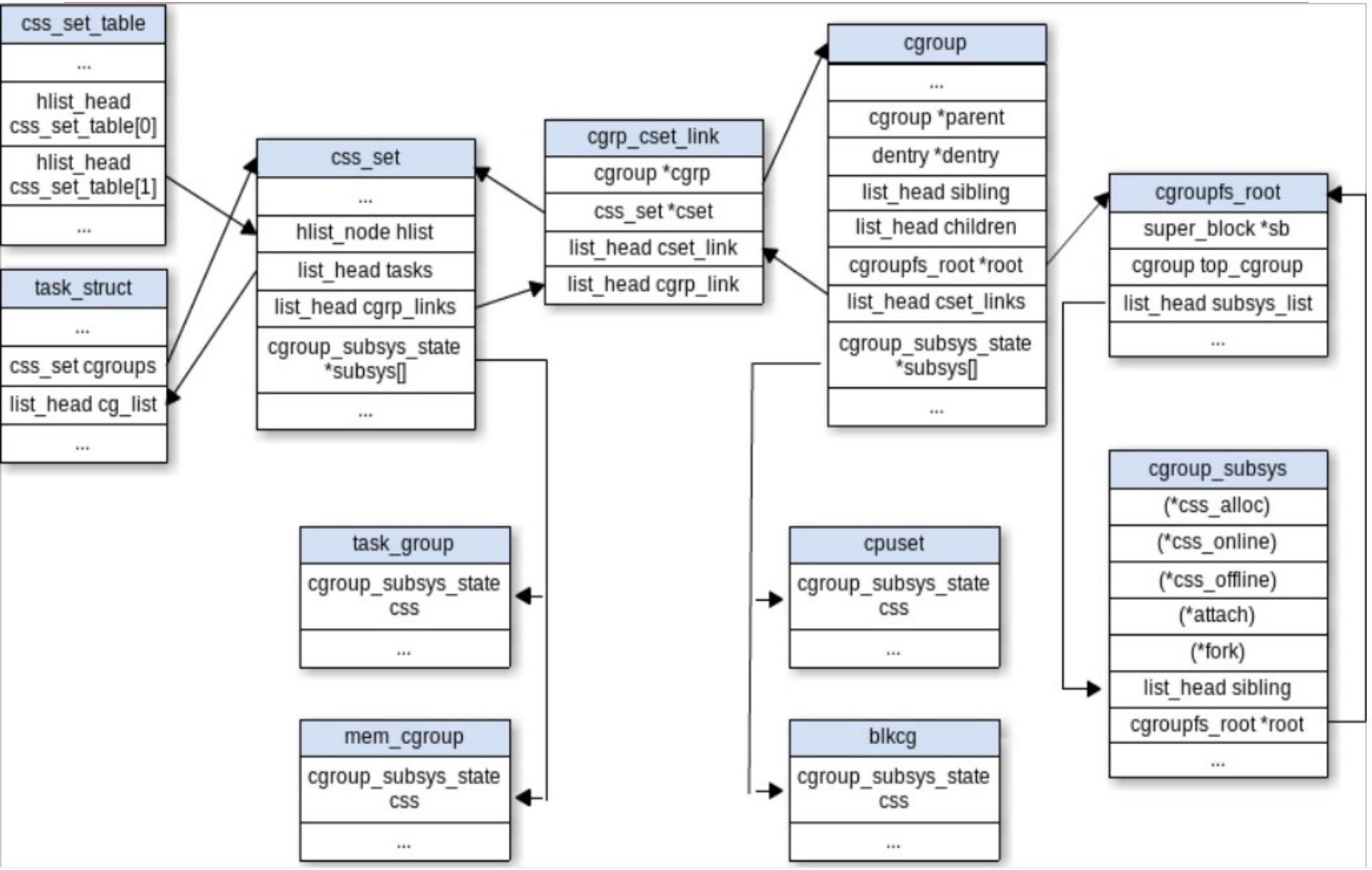
- memory.failcnt
- memory.force_empty
- memory.limit_in_bytes
- memory.max_usage_in_bytes
- memory.move_charge_at_immigrate
- memory.numa_stat
- memory.oom_control
- memory.pressure_level
- memory.soft_limit_in_bytes
- memory.use_hierarchy
- memory.swappiness
- memory.usage_in_bytes
- memory.stat

hugetlb 相关文件

- hugetlb.2MB.failcnt
- hugetlb.2MB.limit_in_bytes
- hugetlb.2MB.max_usage_in_bytes
- hugetlb.2MB.usage_in_bytes

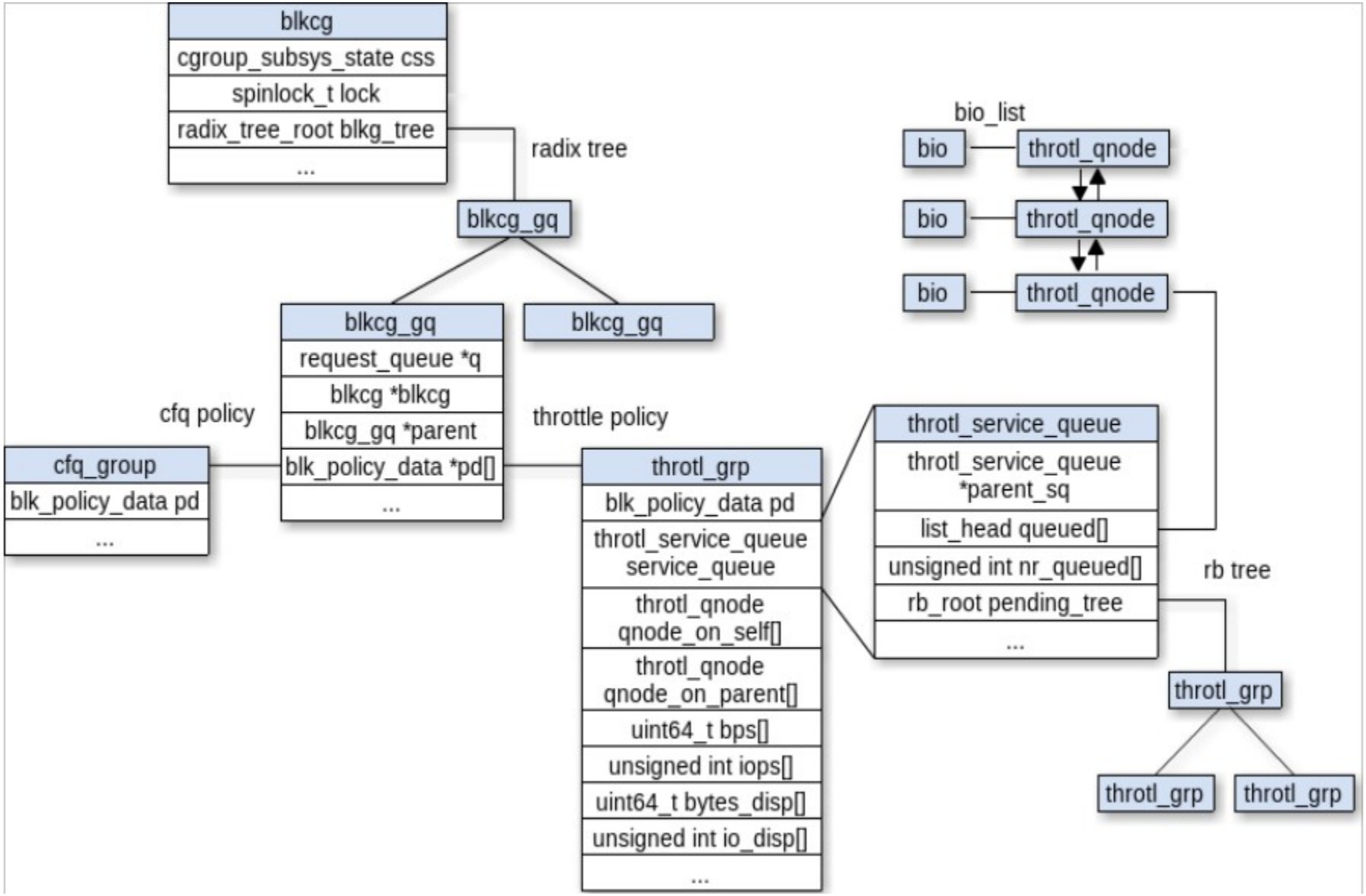
CGroups 框架实现

JD.COM 京东



CGroups IO 限流子系统实现

JD.COM 京东



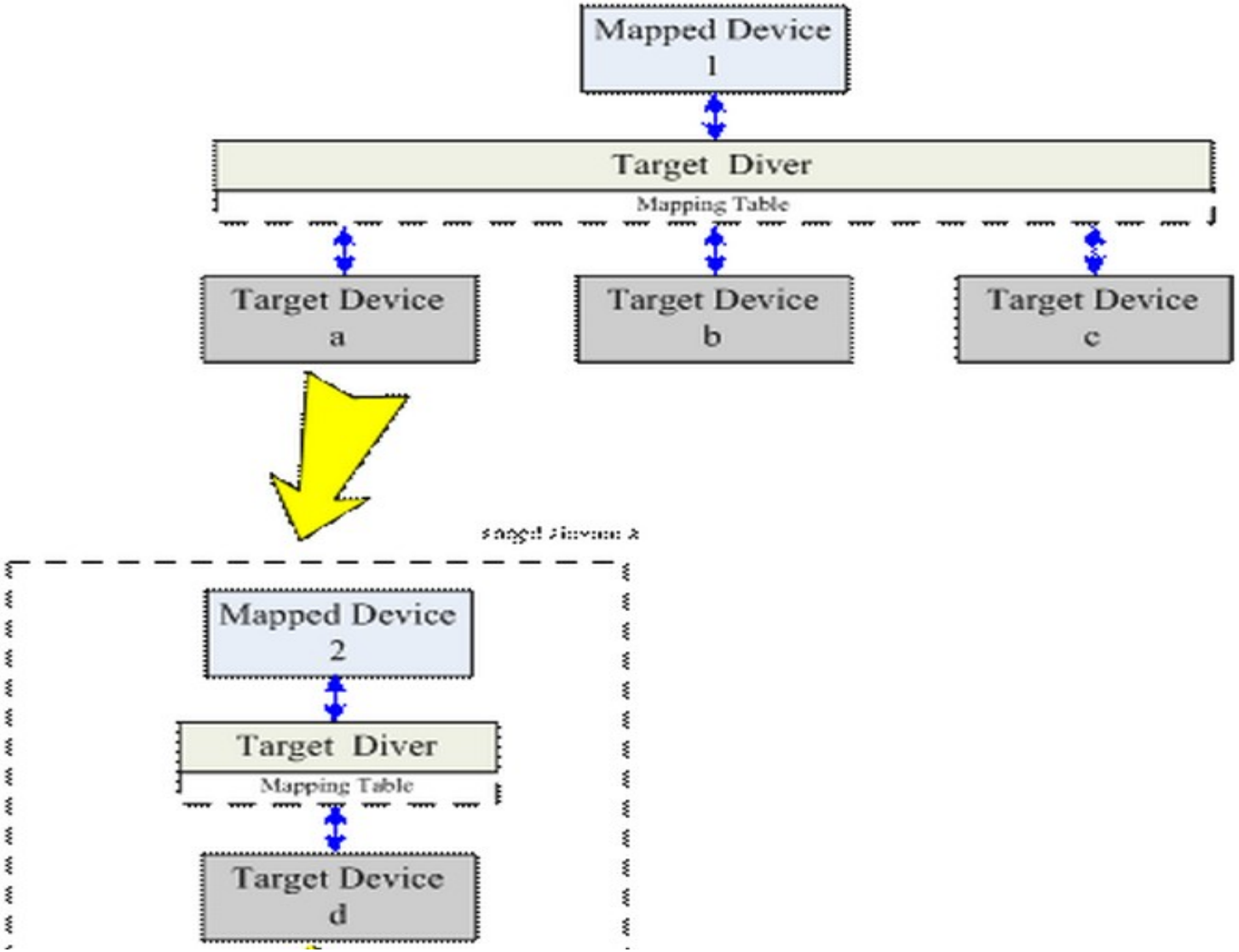
- Buffer IO 无法准确控制
- 带宽控制只能 CFQ 调度器，不适合高速硬件
- 厂商 Flash 卡驱动很可能根本不走 CFQ
- 通用块层只提供限流策略，缺少弹性
- 混合存储或快照支持引入 DM 情况会更加复杂
- 内核通用块层的改写对现有 CGroup 机制的影响

JD.COM 京东

- DM 框架为上层应用提供了丰富的设备映射及 IO 策略方面的支持
- LVM,flashcache,soft-raid 等均基于 DM 实现
- Docker 存储端实现之一也是 DM - thin provision
- 上层通过 dmsetup 工具或 libdevmapper 库使用

Device Mapper 原理

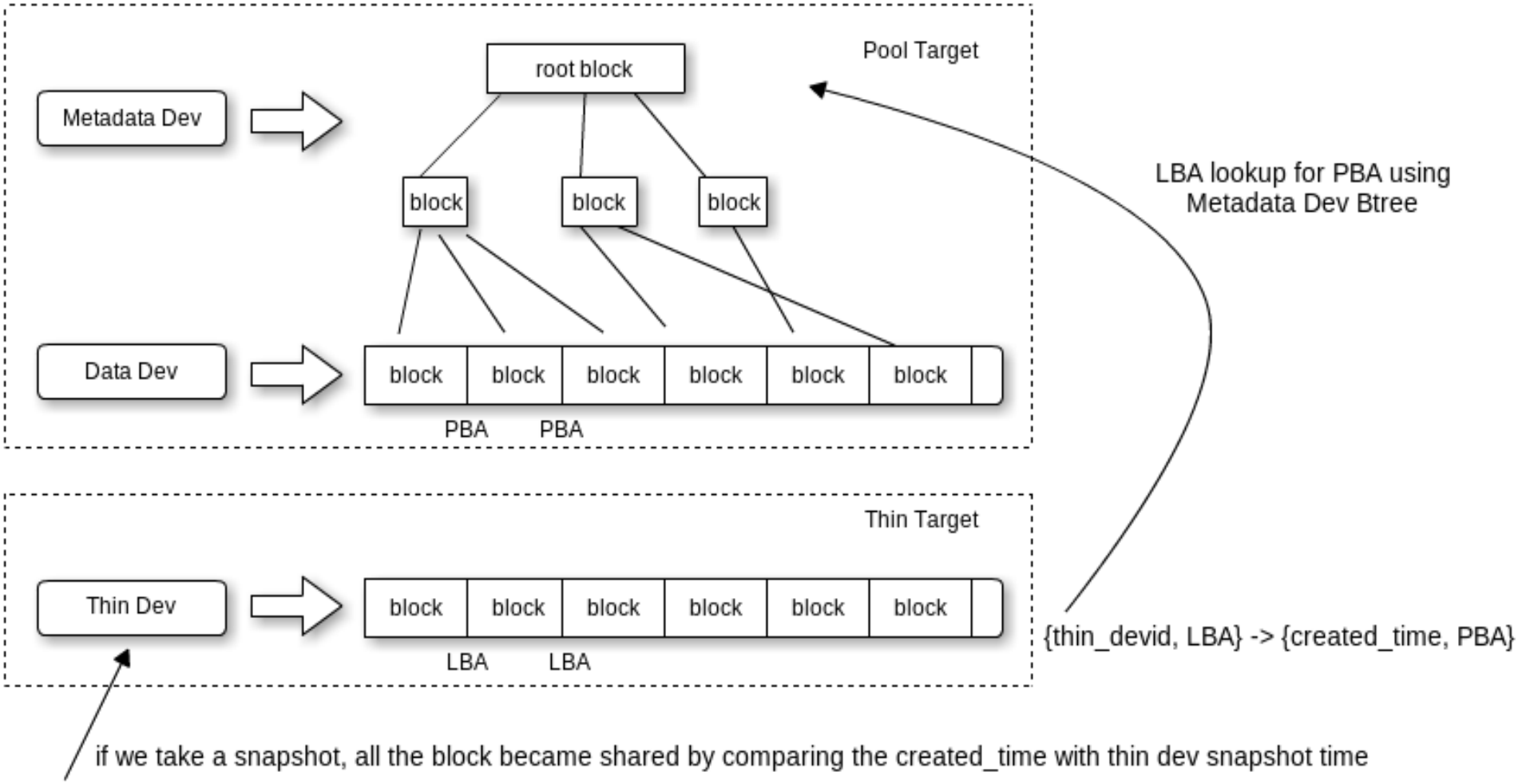
JD.COM 京东



- 由 thin-pool 及 thin 两个 dm target 共同组成
- thin-pool 管理整个资源池的分配， thin 设备及快照管理工作
- thin 向上提供虚拟设备供使用，数据块写时由 thin-pool 分配
- thin-pool 通过 btree 管理 thin 设备 {thin-devid, LBA} -> {time,FBA} 关系
- 根据 thin 设备的最后一次 snapshot 时间与 time 比较判断是否是 shared block
- 实际所有 thin 设备的数据都写入到 thin-pool 管理的 datadev 中， thin 设备只是虚拟抽象

DM thin provision

JD.COM 京东



thin-provision 使用 - 创建 thin-pool

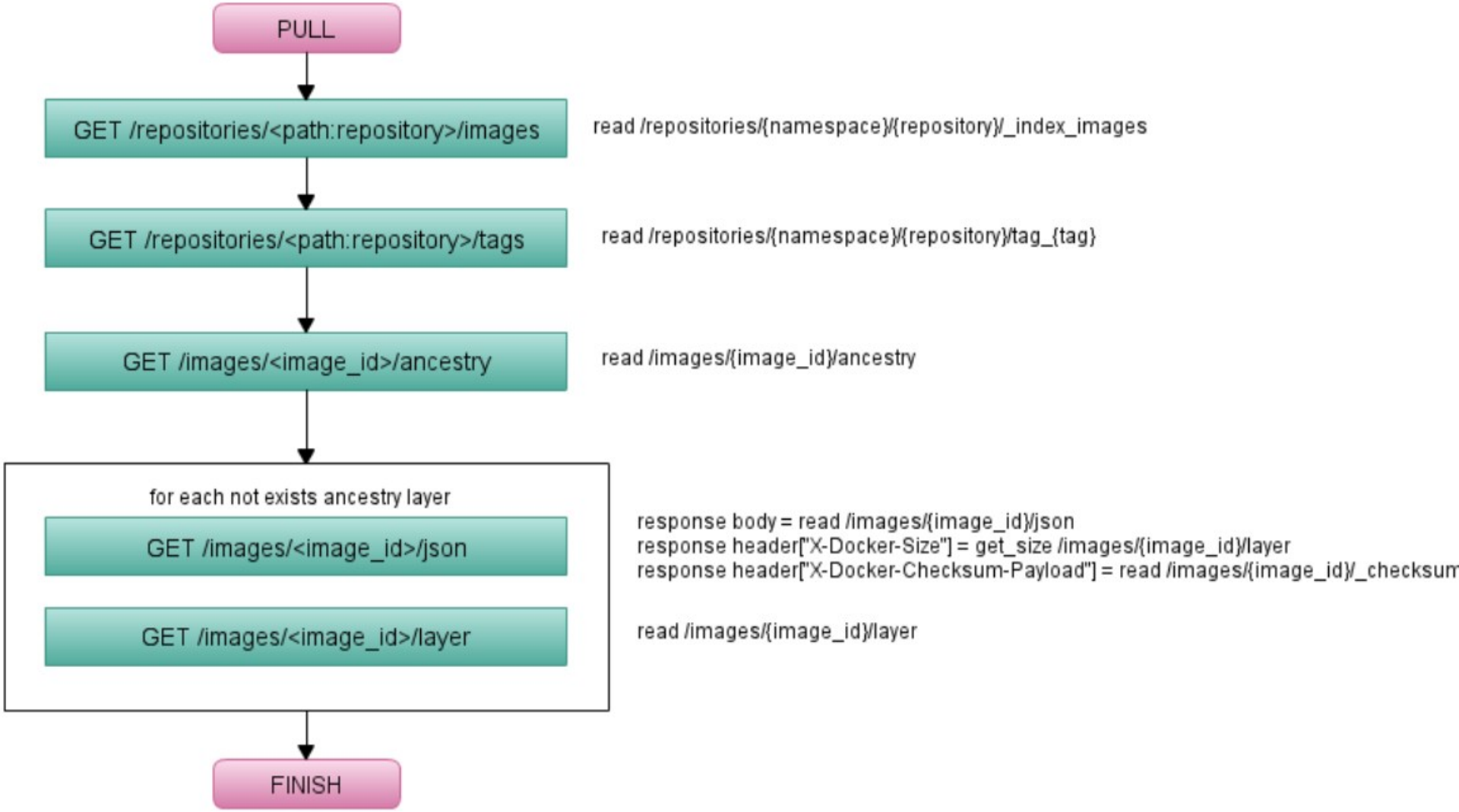
JD.COM 京东

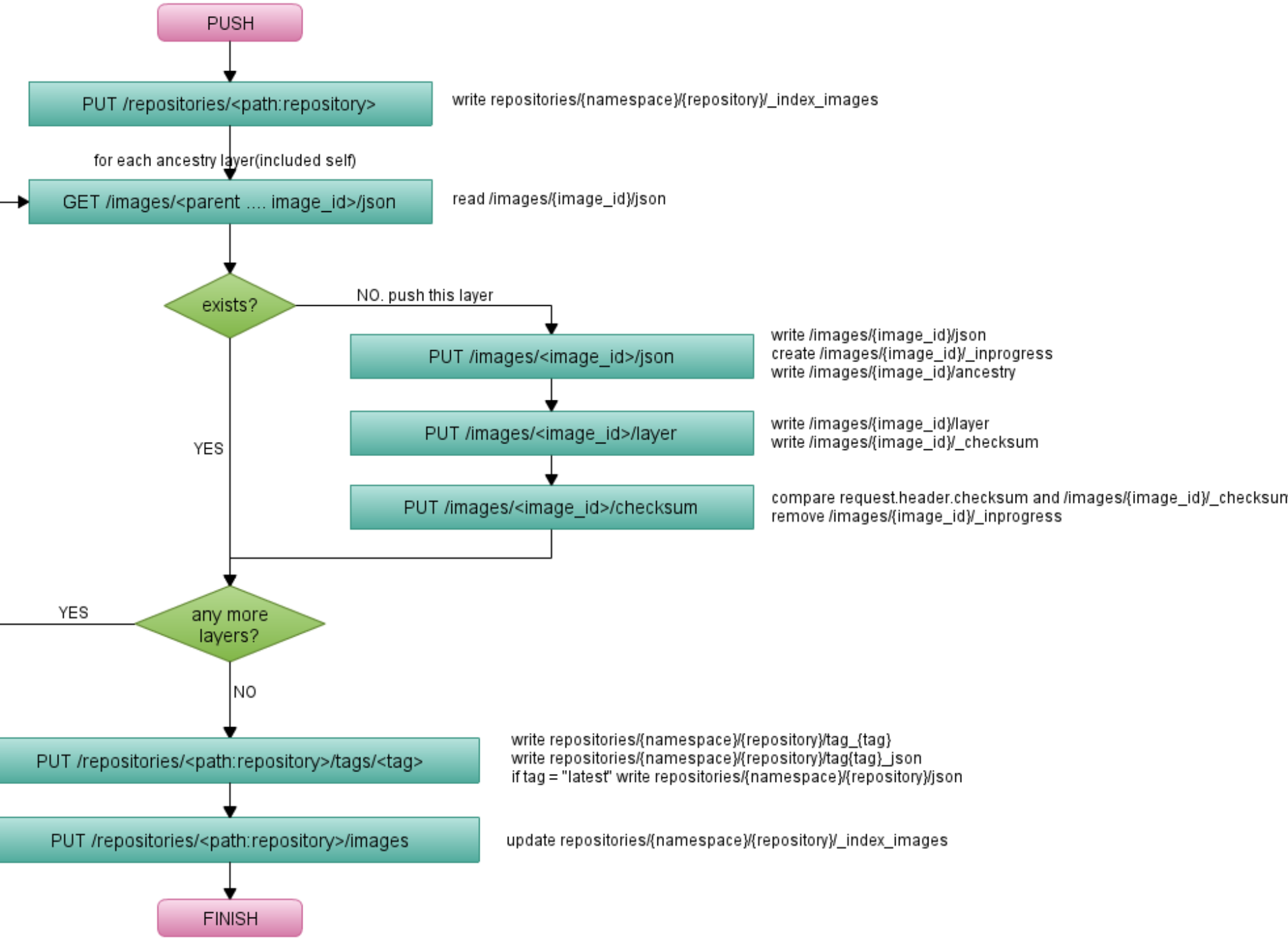
```
root@test:~# dd if=/dev/zero of=metadata bs=1024k count=128
root@test:~# dd if=/dev/zero of=data bs=1024k count=1024
root@test:~# losetup /dev/loop7 metadata
root@test:~# losetup /dev/loop6 data


dmsetup create my_thin_pool --table "0 2097152 thin-pool /dev/loop7 /dev/loop6 128 512"

root@test:~# ll /dev/mapper/my_thin_pool
lrwxrwxrwx 1 root root 7 Dec  8 17:40 /dev/mapper/my_thin_pool -> ../dm-1

root@test:~# dmsetup message /dev/mapper/my_thin_pool 0 "create_thin 0"

root@test:~# dmsetup create my_thin --table "0 65536 thin /dev/mapper/my_thin_pool 0"

root@test:~# ll /dev/mapper/my_thin*
lrwxrwxrwx 1 root root 7 Dec  8 17:49 /dev/mapper/my_thin -> ../dm-2
lrwxrwxrwx 1 root root 7 Dec  8 17:40 /dev/mapper/my_thin_pool -> ../dm-1
```

- Docker 支持 Aufs,DM,Btrfs 等
- Aufs 没有进入主线内核 - which means never
- Btrfs 目前没有 production ready
- Docker 代码对 DM 的使用写死较多，需手工调整
- 由于 DM 基于设备层，对上层文件系统 layer Diff 无法直接支持， Docker 手工比对文件实现
- 不停容器的情况，理论上无法保障一致性

Docker registry 镜像存储

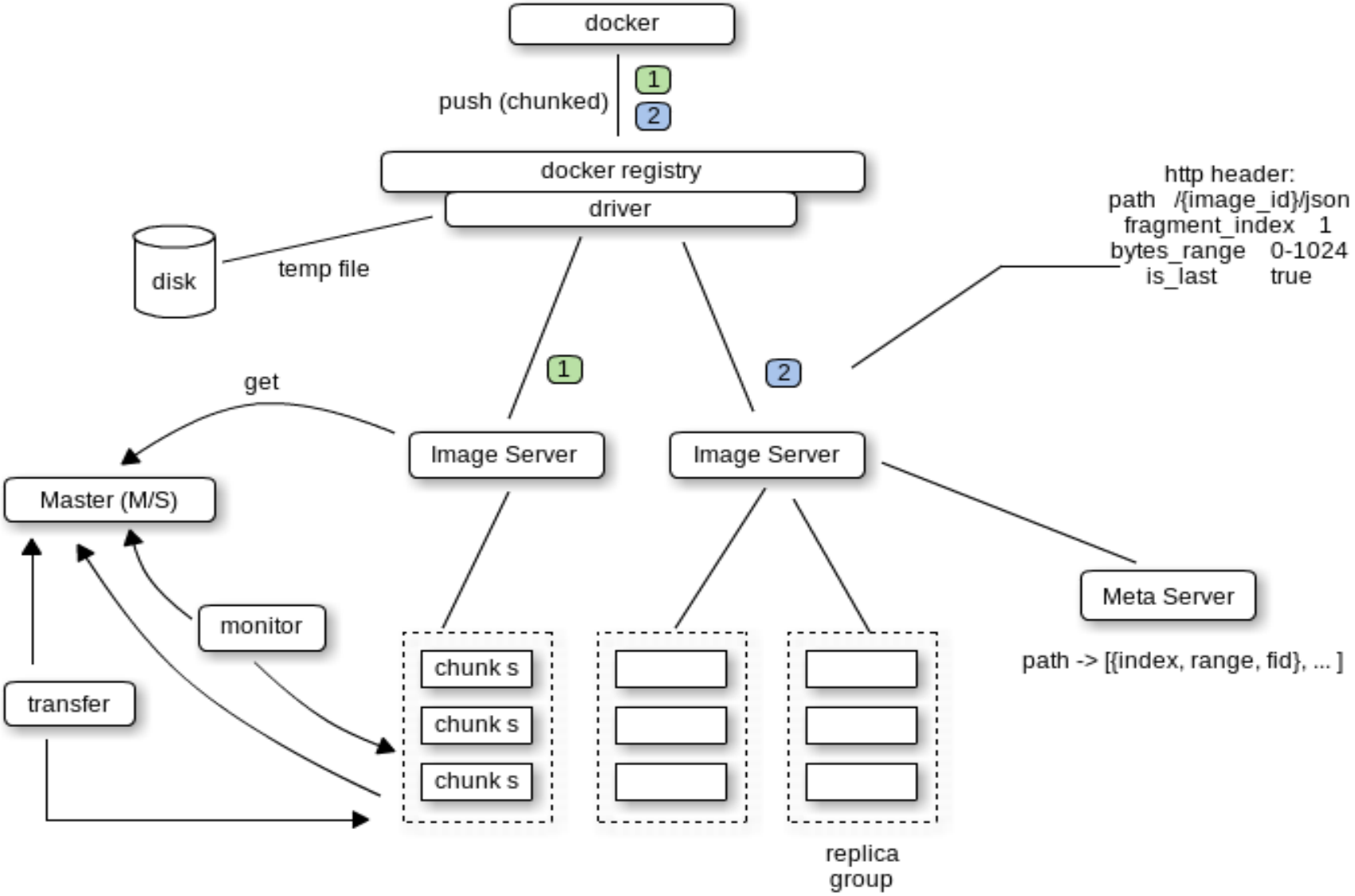
JD.COM 京东




PUSH
PUT /repositories/<path:repository>
write repositories/{namespace}/{repository}/_index_images
for each ancestry layer(included self)
GET /images/<parent .... image_id>/json
read /images/{image_id}/json
exists?
NO. push this layer
PUT /images/<image_id>/json
write /images/{image_id}/json
create /images/{image_id}/_inprogress
write /images/{image_id}/ancestry
PUT /images/<image_id>/layer
write /images/{image_id}/layer
write /images/{image_id}/_checksum
PUT /images/<image_id>/checksum
compare request.header.checksum and /images/{image_id}/_checksum
remove /images/{image_id}/_inprogress
YES
YES
any more layers?
NO
PUT /repositories/<path:repository>/tags/<tag>
write repositories/{namespace}/{repository}/tag_{tag}
write repositories/{namespace}/{repository}/tag{tag}_json
if tag = "latest" write repositories/{namespace}/{repository}/json
PUT /repositories/<path:repository>/images
update repositories/{namespace}/{repository}/_index_images
FINISH

我们的镜像存储系统

JD.COM 京东



blah~,blah~,blah~

- 当前 Namespace 功能仍不完善，需要更多的隔离
- CGroup 资源配额在 IO 方面仍存在一些问题
- Docker 存储端仍需做一些选择或工作
- 选择 DM thin-provision 需要手工创建 thin-pool
- 基于 DM thin-provision 的不停容器的镜像生成可能存在一致性问题
- Docker Registry Storage 需要选择开源或定制开发

谢谢

微博： @ 摇摆巴赫

微信： swingbach