RAMCloud Design Review

Recovery

Ryan Stutsman

April 1, 2010

Overview

- **Master Recovery**
  - 2-Phase
  - Partitioned
- **Failures**
  - Backups
  - Rack/Switch
  - Power
  - Datacenter

Implications of Single Copy in Memory

- **Problem: Unavailability**
  - If master crashes unavailable until read from disks on backups
  - Read 64 GB from one disk? **10 minutes**

- **Use scale to get low-latency recovery**
  - Lots of disk heads, NICs, CPUs
  - **Our goal**: recover in **1-2 seconds**
    - Is this good enough?
  - Applications just see “hiccups”

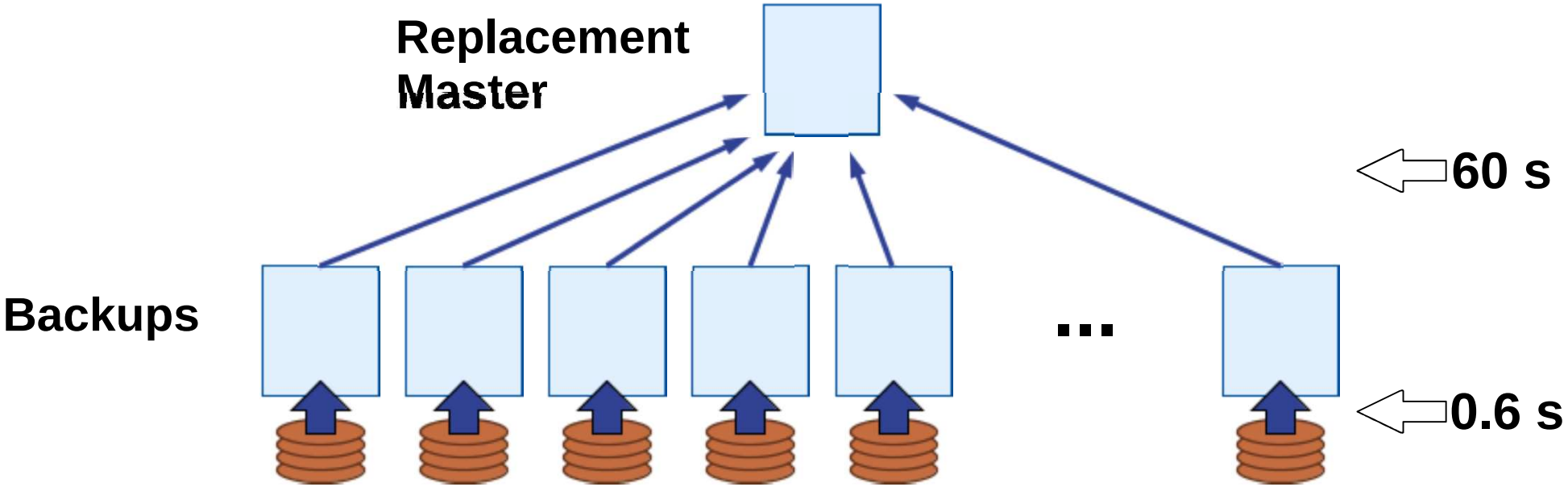
Fast Recovery

- **Problem: Disk bottleneck for recovery**
- **Idea: Leverage many spindles to recover quickly**
  - Data broadly scattered throughout backups
    - Not just great write throughput
    - Take advantage of read throughput
- **Reincarnate masters**
  - With same tables
  - With same indexes
  - Preserves locality

Fast Recovery: The Problem

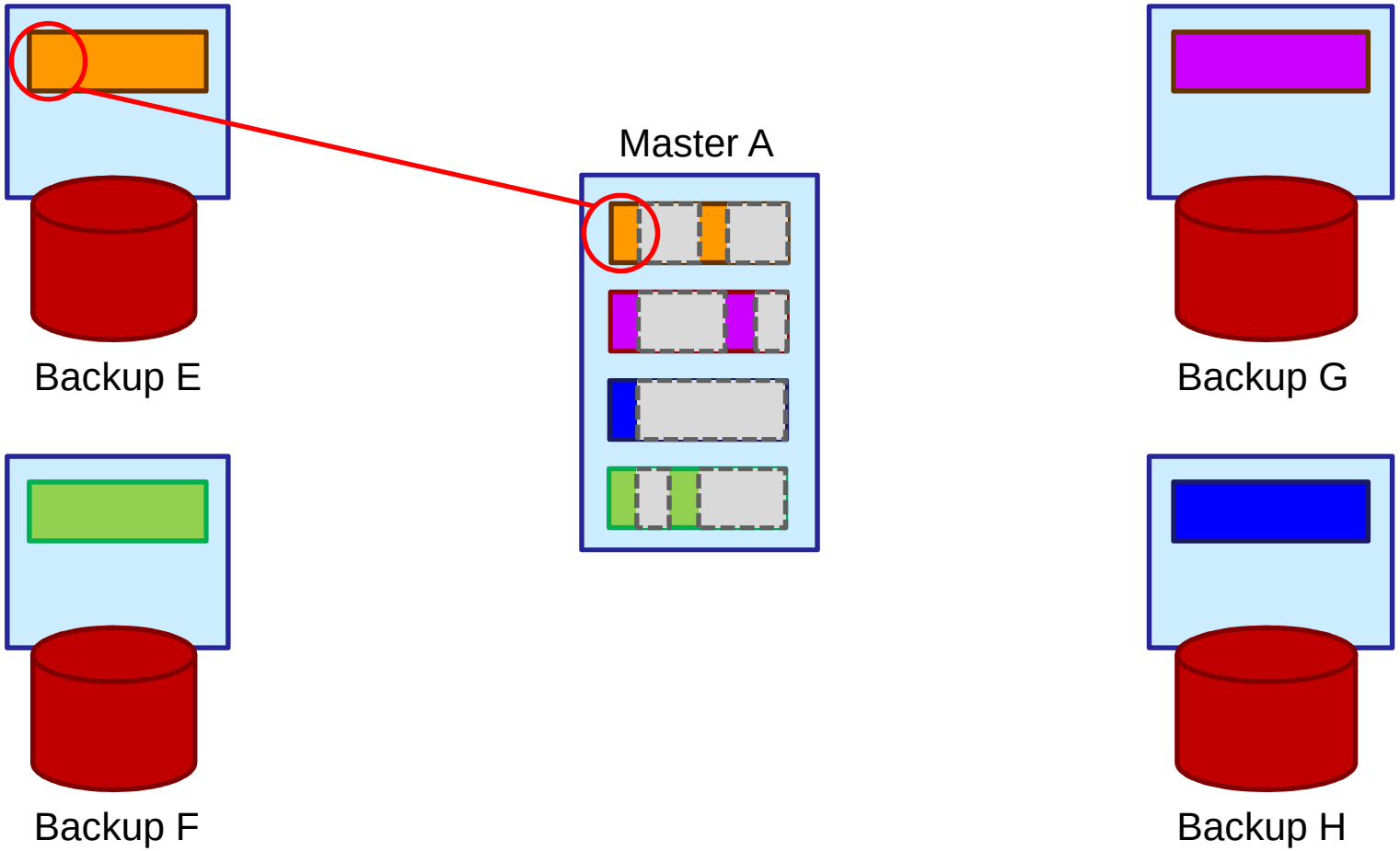
- **After crash, all backups read disks in parallel**
  (64 GB/1000 backups @ 100 MB/sec = **0.6 sec, great!**)

- **Collect all backup data on replacement master**
  (64 GB/10Gbit/sec ~ **60 sec: too slow!**)
  Problem: Network is now the bottleneck!



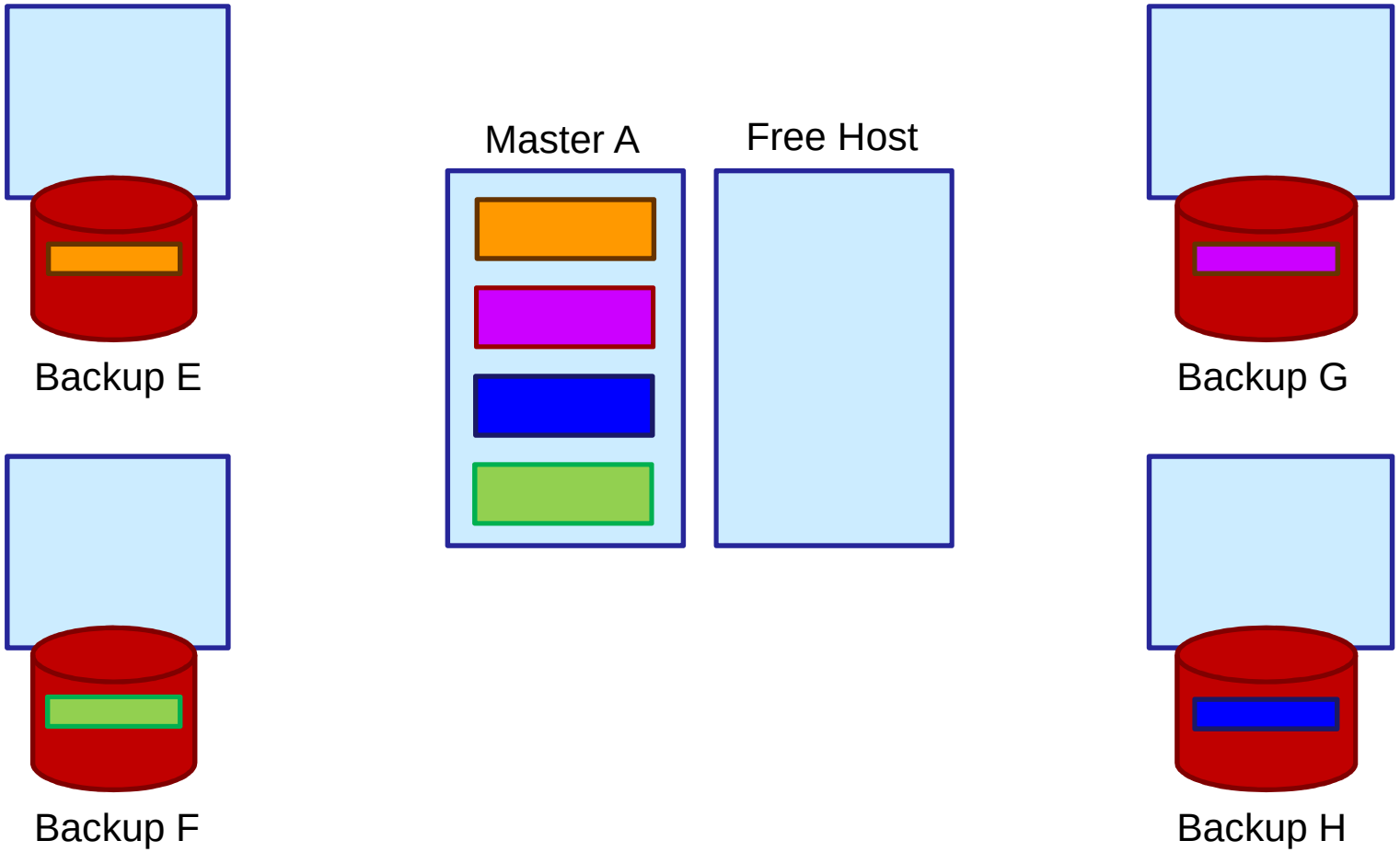
2-Phase Recovery

- **Idea: Data already in memory on backups, just need to know where**



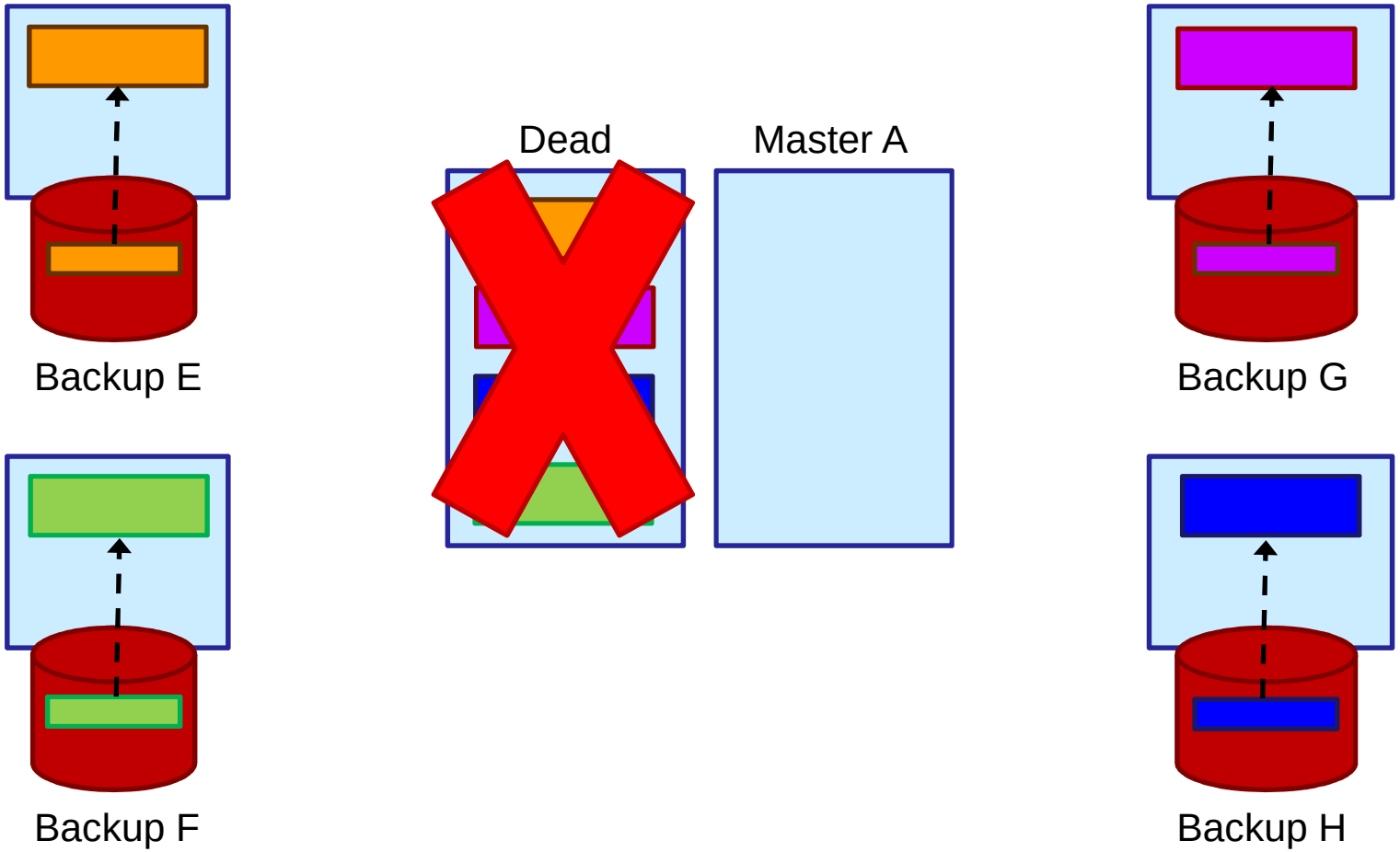
2-Phase Recovery

- **Phase #1: Recover location info (< 1s)**



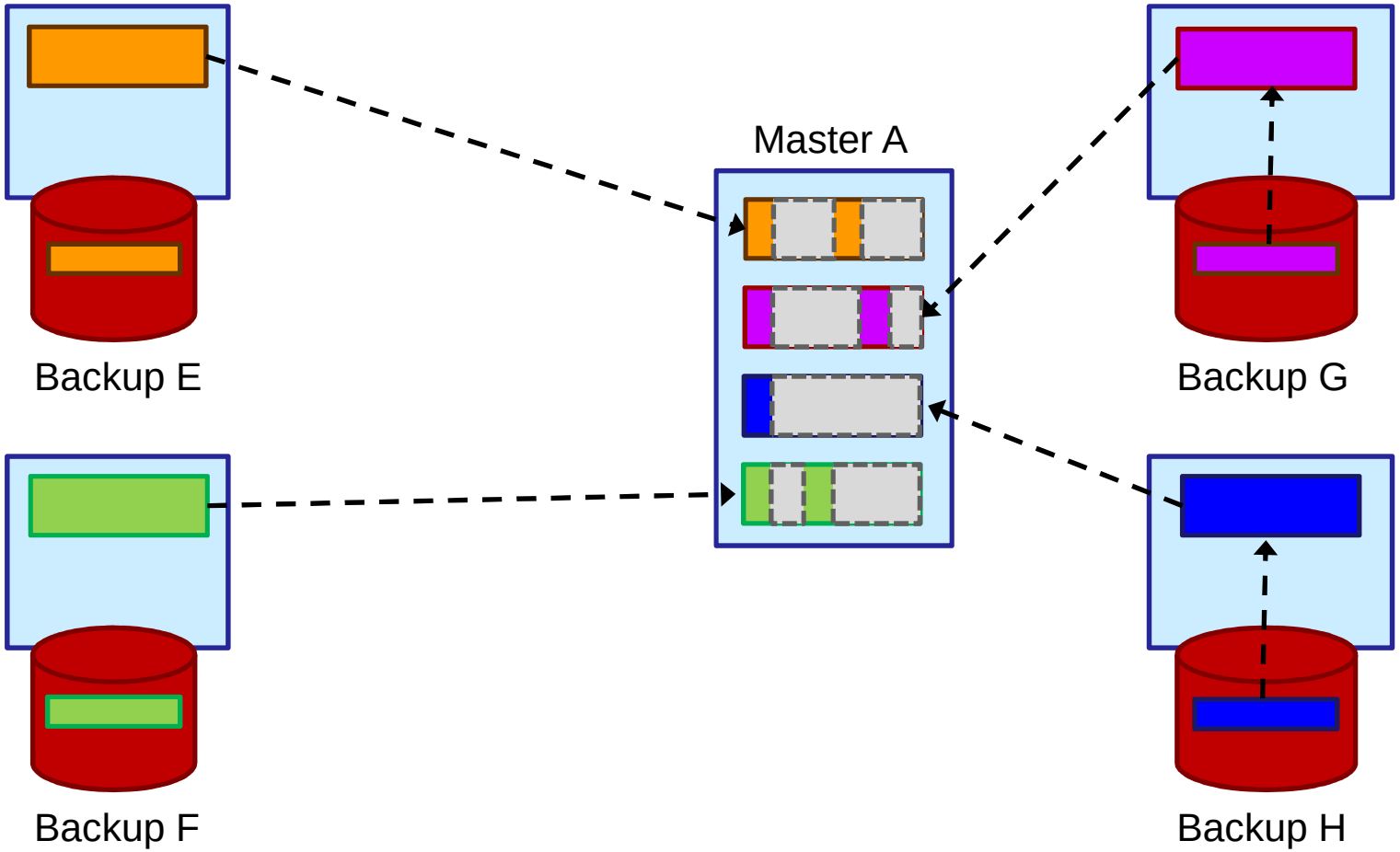
2-Phase Recovery

- **Phase #1: Recover location info (< 1s)**
  - Read all data into memories of backups



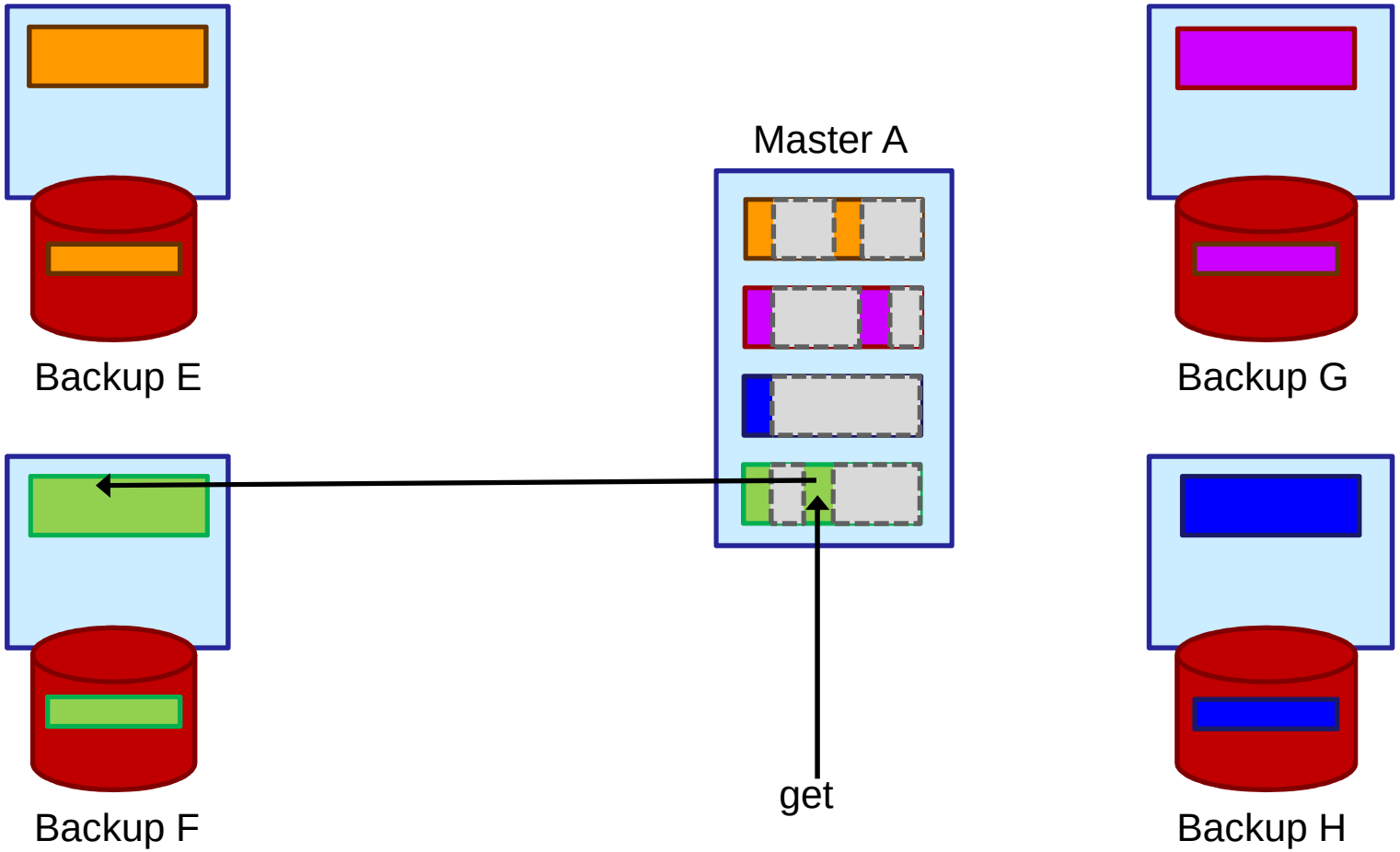
2-Phase Recovery

- **Phase #1: Recover location info (< 1s)**
  - Read all data into memories of backups
  - Send only location info to replacement master



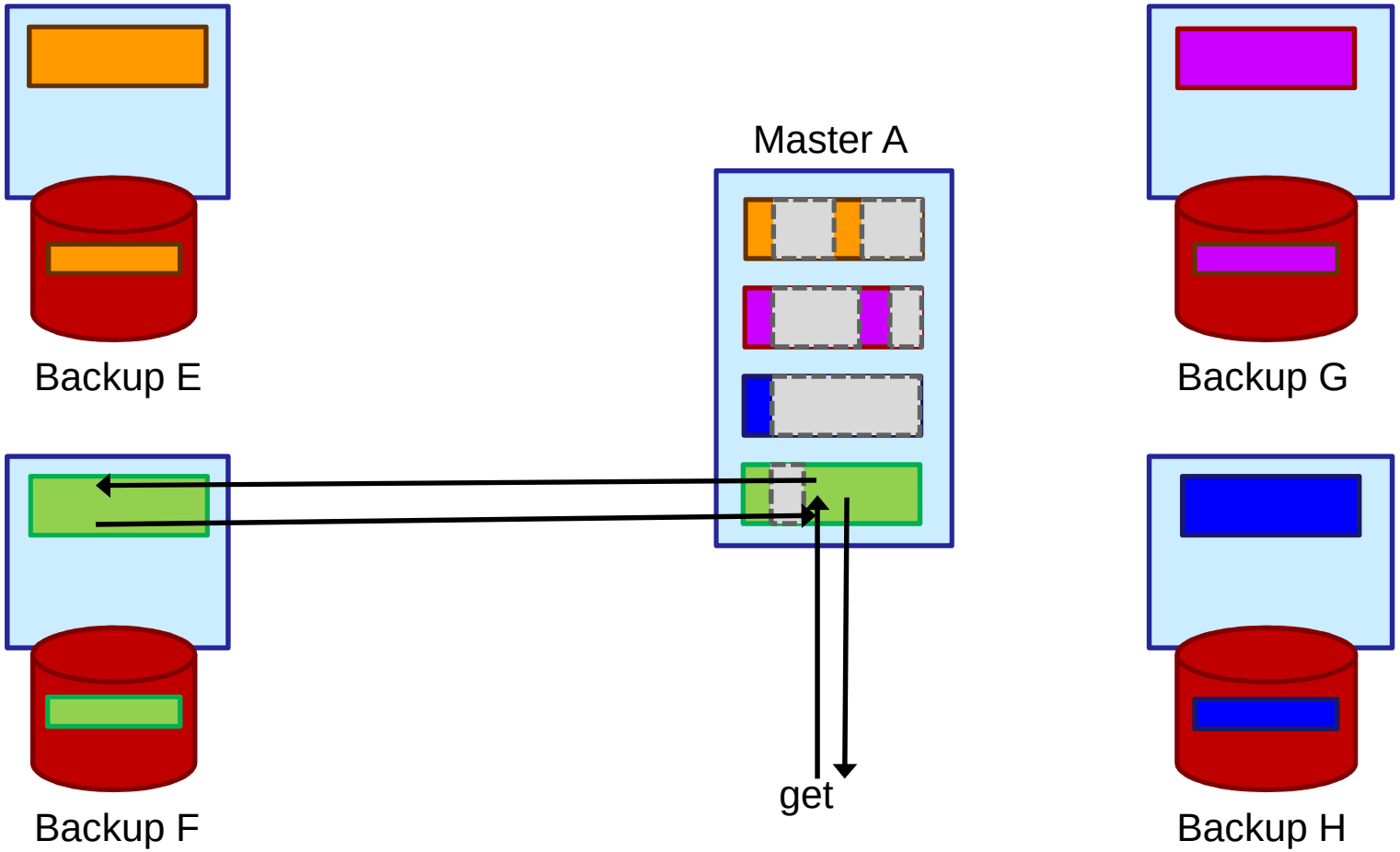
2-Phase Recovery

- **Phase #2: Proxy & Recover Full Data (~60s)**
  - **System resumes operation:**
    - Fetch on demand from backups
    - 1 extra round trip on first read of an object
    - Writes are full speed



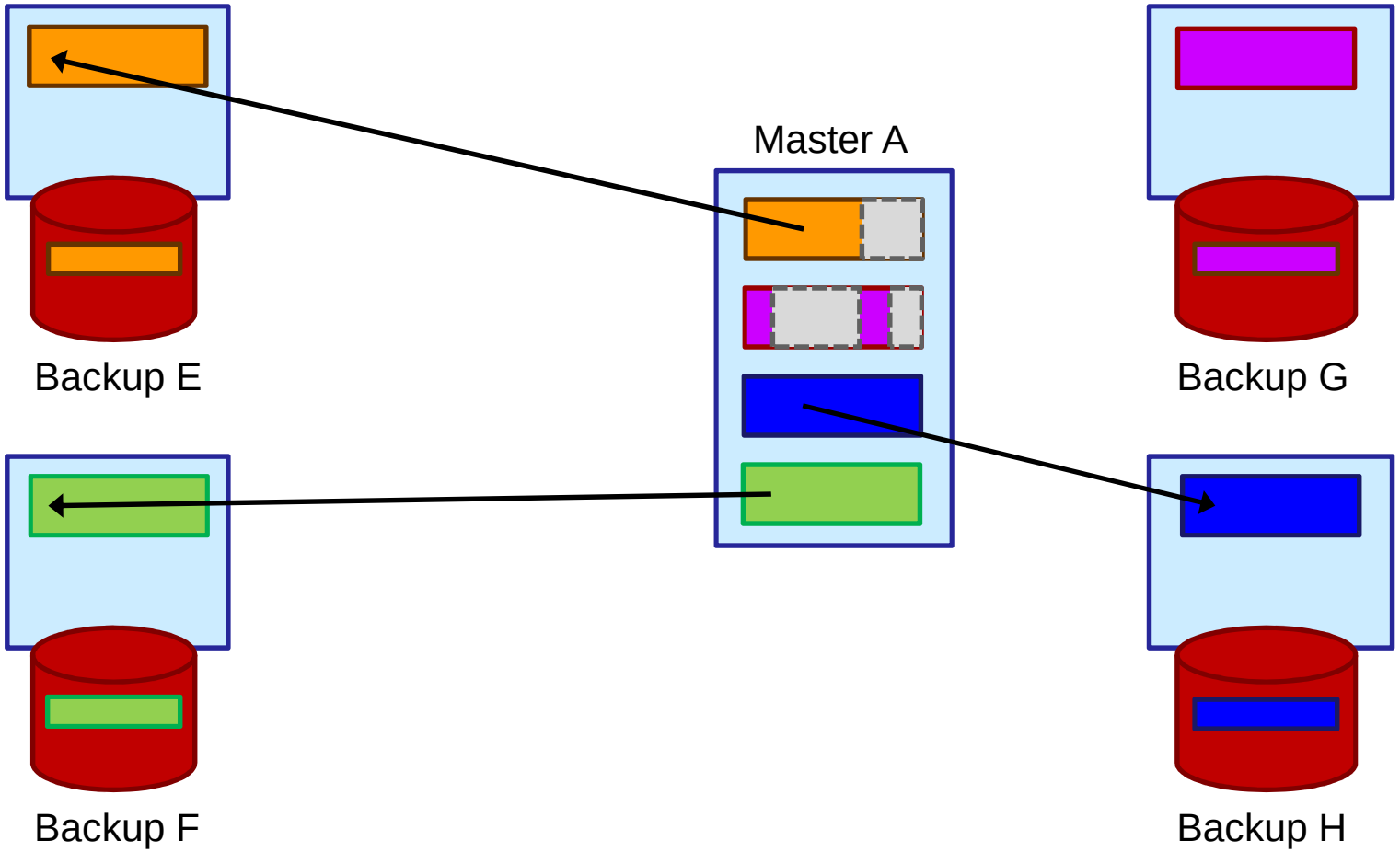
2-Phase Recovery

- **Phase #2: Proxy & Recover Full Data (~60s)**
  - **System resumes operation:**
    - Fetch on demand from backups
    - 1 extra round trip on first read of an object
    - Writes are full speed



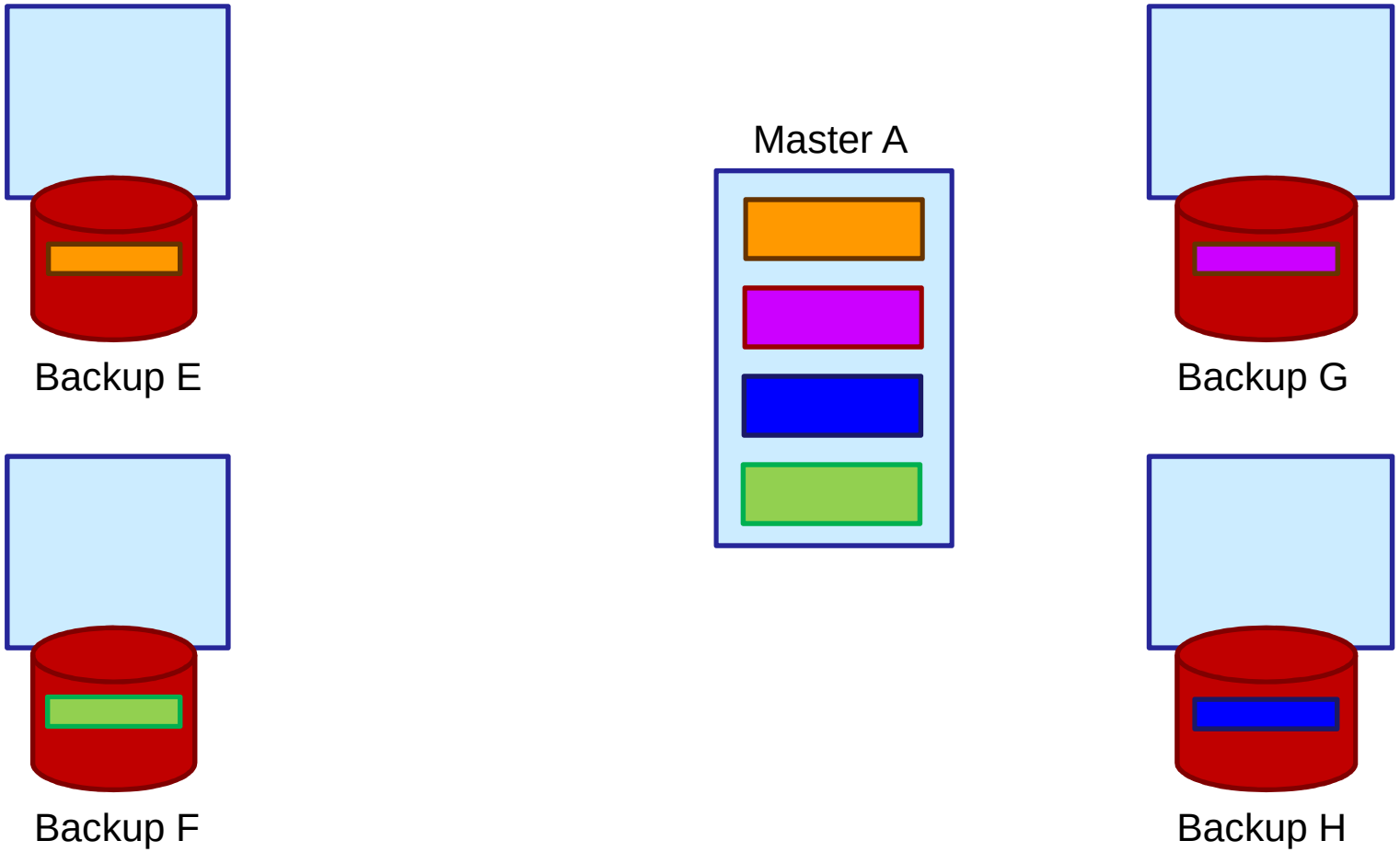
2-Phase Recovery

- **Phase #2: Proxy & Recover Full Data (~60s)**
  - Transfer data from backups in between servicing requests



2-Phase Recovery

- **Performance normal after Phase #2 completes**



2-Phase Recovery: Thoughts

- ✓ **Recovers locality by recovering machines**
- ✗ **Need to talk to all hosts**
  - Because backup data for a single master is on all machines
  - How bad is this?
- ✗ **Doesn't deal with heterogeneity**
  - Machine is the unit of recovery
  - Can only recover a machine to one with more capacity
- **Bi-modal Utilization**
  - Must retain pool of empty hosts

2-Phase Recovery: Problem

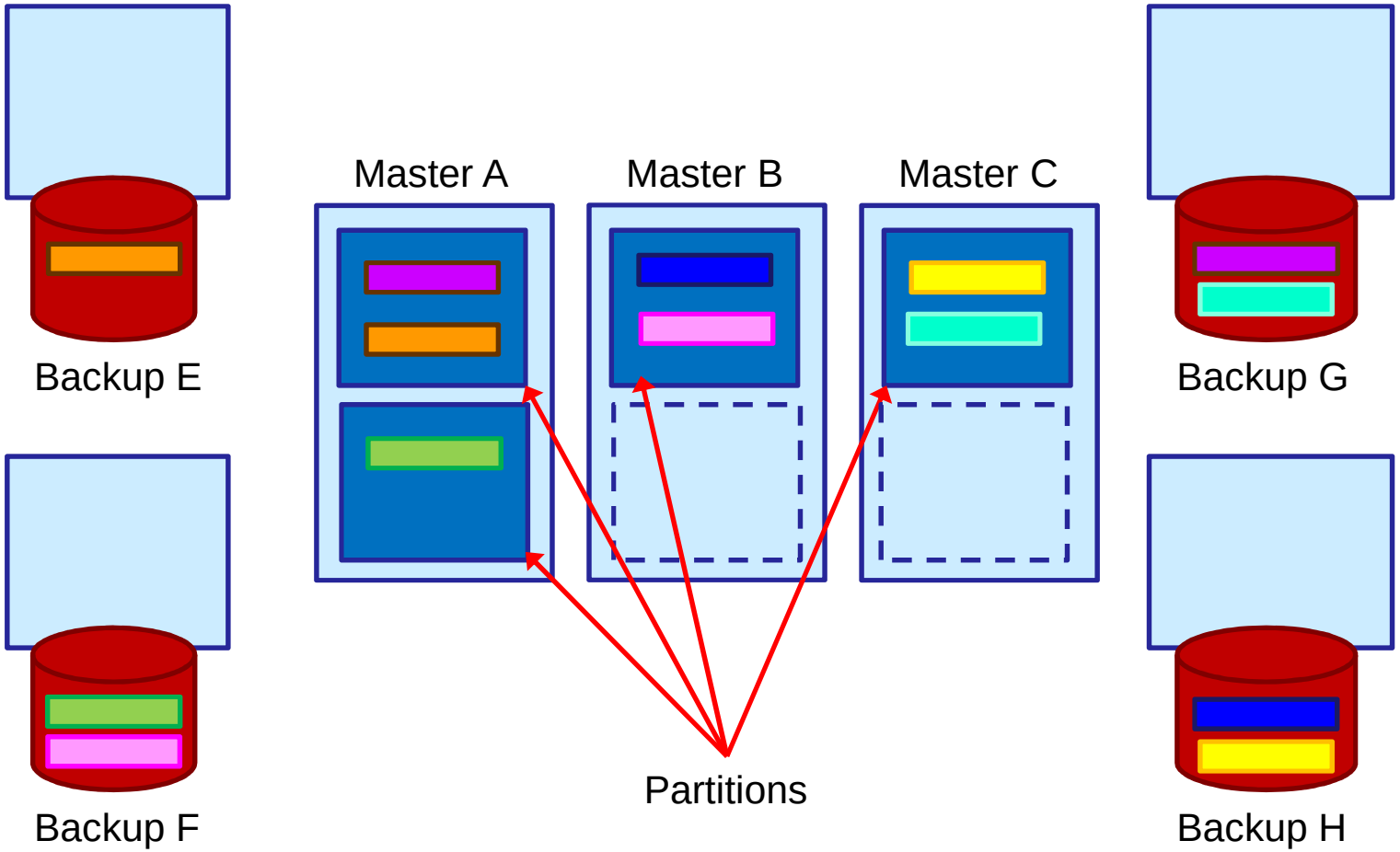
- ✘ **Hashtable inserts become the new bottleneck**
  - Phase #1 must place all objects in the hashtable
  - Master can have 64 million 1 KB objects
  - Hashtable can sustain about 10 million inserts/s
  - 6.4s is over our budget
  - Can use additional cores, but objects could be even smaller

- **Unsure of a way to recover exact master quickly**
  - Constrained by both CPU and NIC
  - Recovery to single host is a bottleneck

- **Another way?**

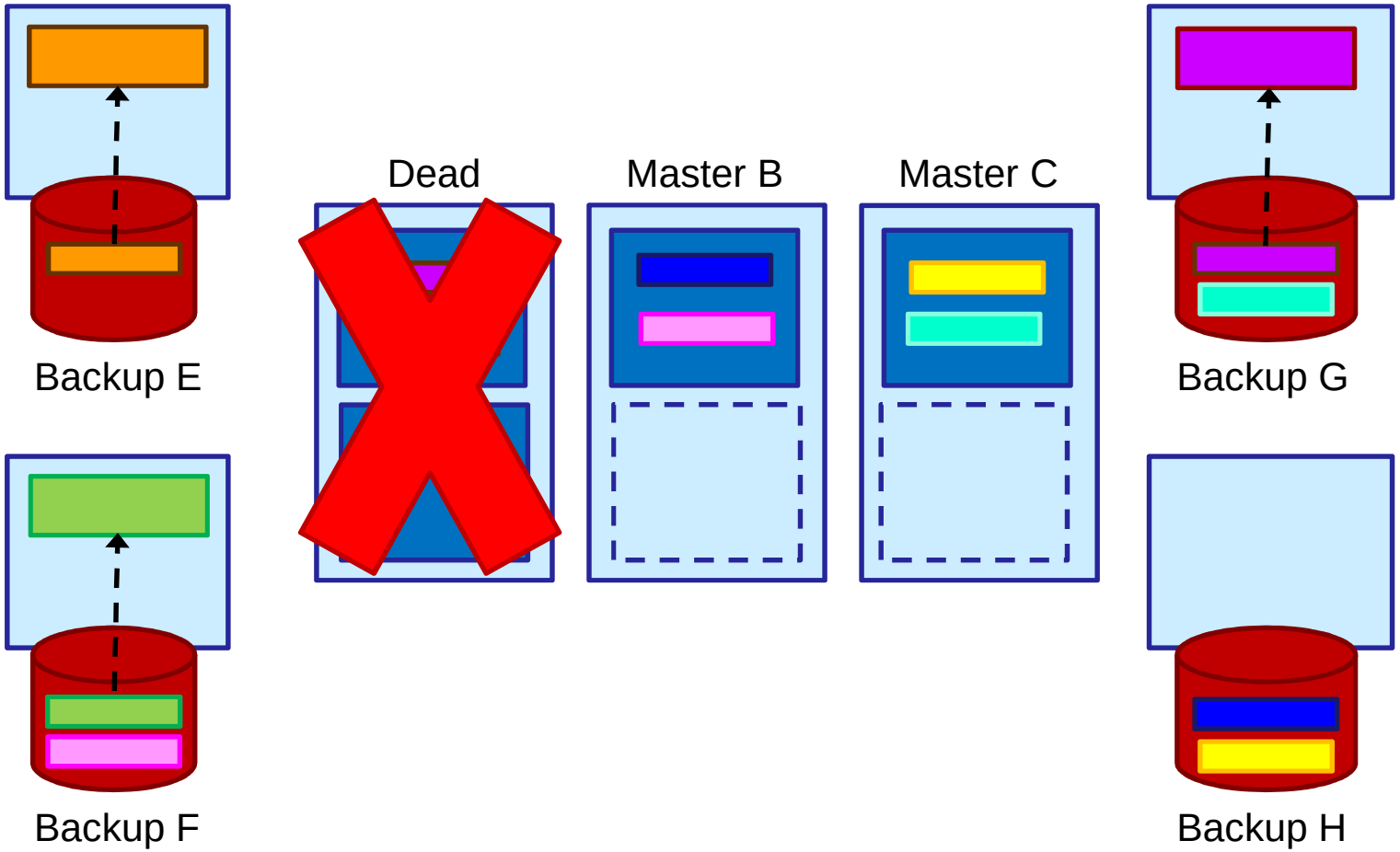
Partitioned Recovery

- **Idea: Leverage many hosts to overcome bottleneck**
  - Problem is machines are large so divide them into partitions
  - Recover each partition to a different master
  - Just like a machine
    - Contains any number of tables, table fragments, indexes, etc.



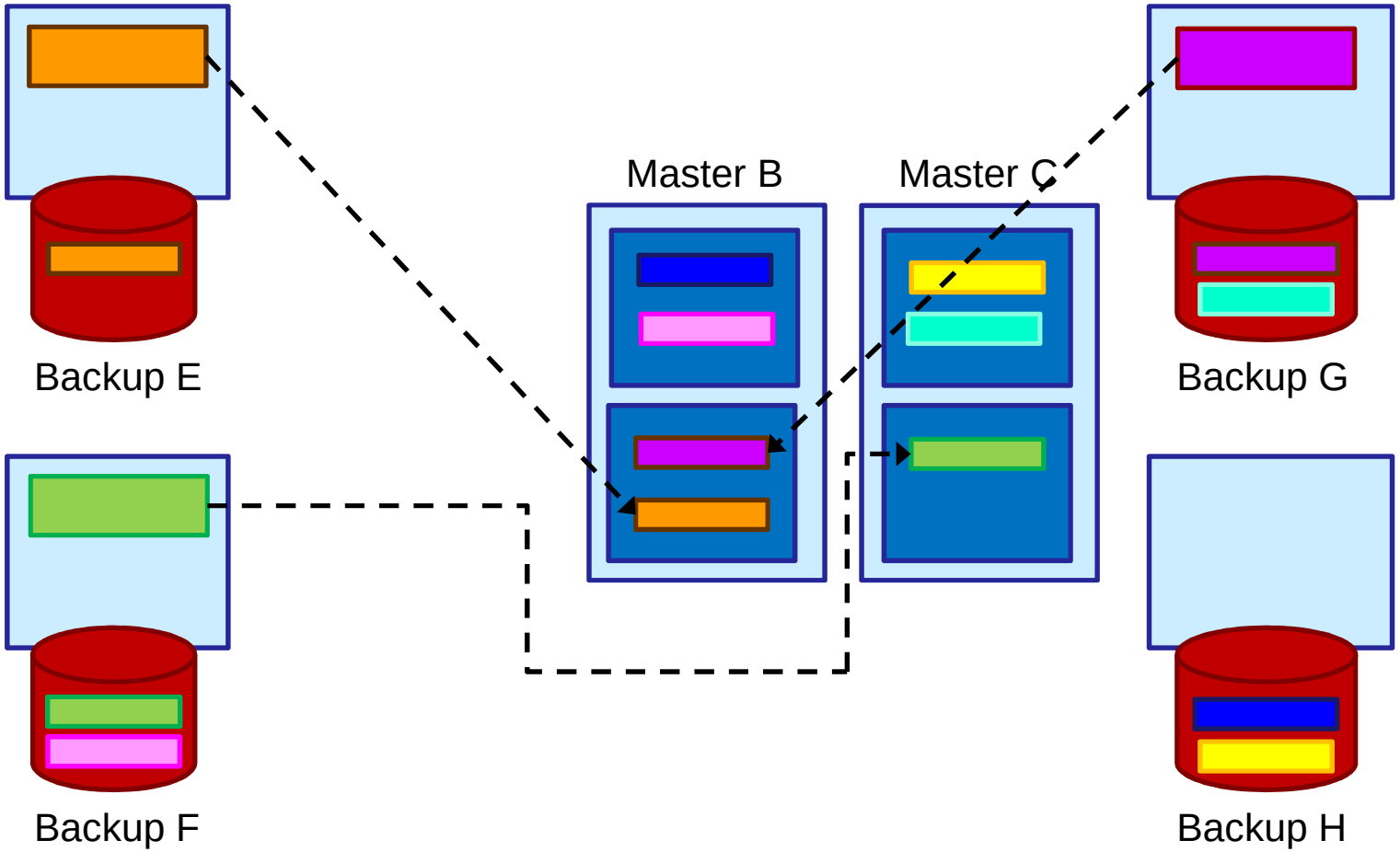
Partitioned Recovery

- **Load data from disks**



Partitioned Recovery

- **Reconstitute partitions on many hosts**
- **64 GB / 100 partitions = 640 MB**
- **640 MB / 10 GBit/s = 0.6s for full recovery**

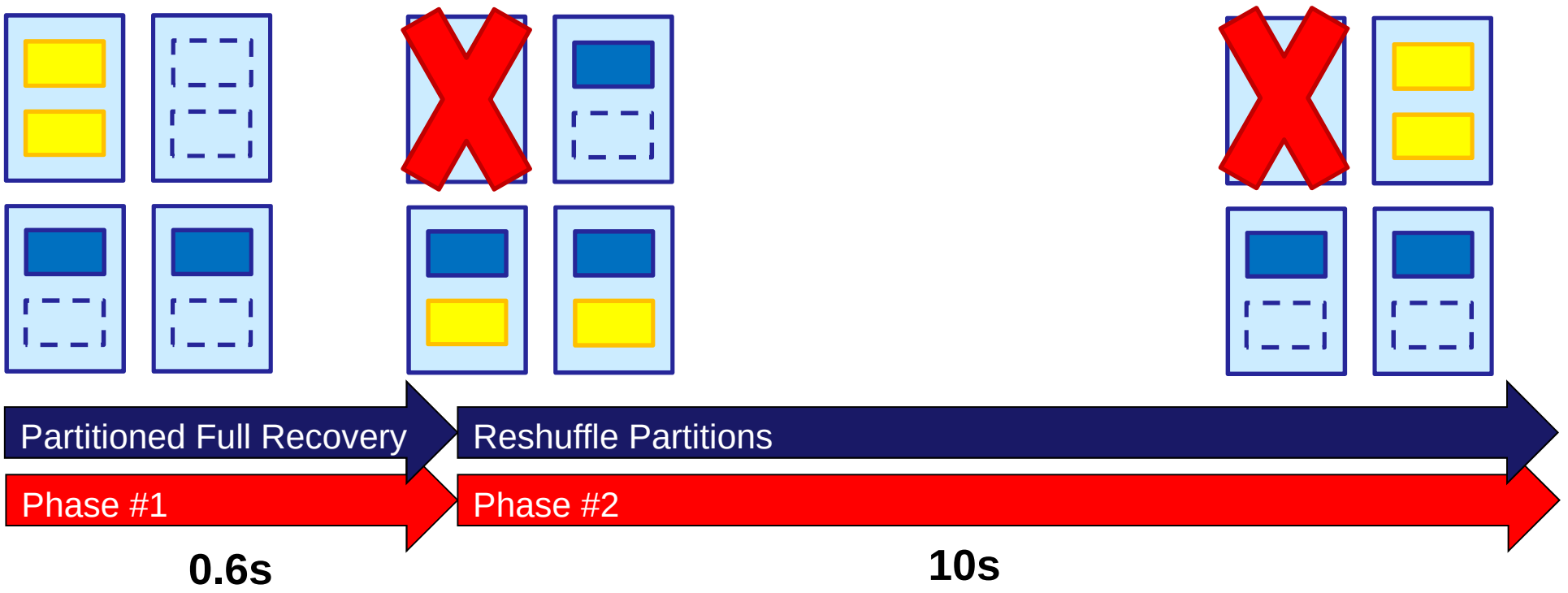


Partitioned Recovery: Thoughts

- ✓ **It works: meets availability goals**
  - o Can tune time by adjusting partition size
- ✓ **Helps with heterogeneity**
  - o Unit of recovery is no longer a machine
- ✗ **Increases host/partition related metadata**
  - o Partitions still large enough to provide app locality
- ✗ **Need to talk to all hosts**
- • **No hot spares**

Partitioned Recovery: Thoughts

- **Does not recover locality**
  - But, no worse than 2-Phase
  - Partitioned approach recovers as fast as Phase #1
  - Can restore locality as fast as Phase #2



Failures: Backups

- **On backup failure the coordinator broadcasts**
  - More information during coordinator discussion
- **All masters check their live data**
- **If objects were on that backup rewrite elsewhere (from RAM)**

- **No recovery of backups themselves**

Failures: Racks/Switches

- **Careful selection of backup locations**
  - Write backups for objects to other racks
    - As each other
    - As the master
  - Changes as masters recover
    - Can move between racks
  - Masters fix this on recovery

- **Rack failures handled the same as machine failures**
  - Consider all the machines in the rack dead

- **Question: Minimum RAMCloud that can sustain an entire rack failure and meet recovery goal?**
  - 100 partitions to recover a single machine in 0.6s
  - 50 dead * 100 partitions, need 5000 machines to make 0.6s
  - Don’t pack storage servers in racks, mix with app servers

Failures: Power

- **Problem: Objects are buffered temporarily in RAM**
  - Even after the put has returned as successful to the application
- **Solution: All hosts have on-board battery backup**
- **Flush all dirty objects on fluctuation**
  - Any battery should be easily sufficient for this
  - Each master has *r* active buffers on backups
  - Buffers are 8MB, for 3 disk copies
    - 3 * 8MB takes 0.24s to flush

- **Question: Cost effective way to get 10-20s of power?**

- **No battery?**
  - Deal with lower consistency
  - Synchronous writes

Failures: Datacenter

- **Durability guaranteed by disks, no availability**
  - Modulo nuclear attacks
- **No cross-DC replication in version 1**
  - Latency can't be reconciled with consistency
  - Aggregate write bandwidth of 1000 host RAMCloud
    - 100 MB/s * 1000 = 1 Tbit/s
- **Application level replication will do much better**
  - Application can batch writes
  - Application understands consistency needs
- **Is this something we need to support?**

Summary

- **Use scale in two ways to achieve availability**
  - Scatter reads to overcome disk bottleneck
  - Scatter rebuilding to overcome CPU and network bottlenecks
- **Scale driving lower-latency**
- **Remaining Issue: How do we get information we need for recovery?**
  - Every master recovery involves all backups

Discussion