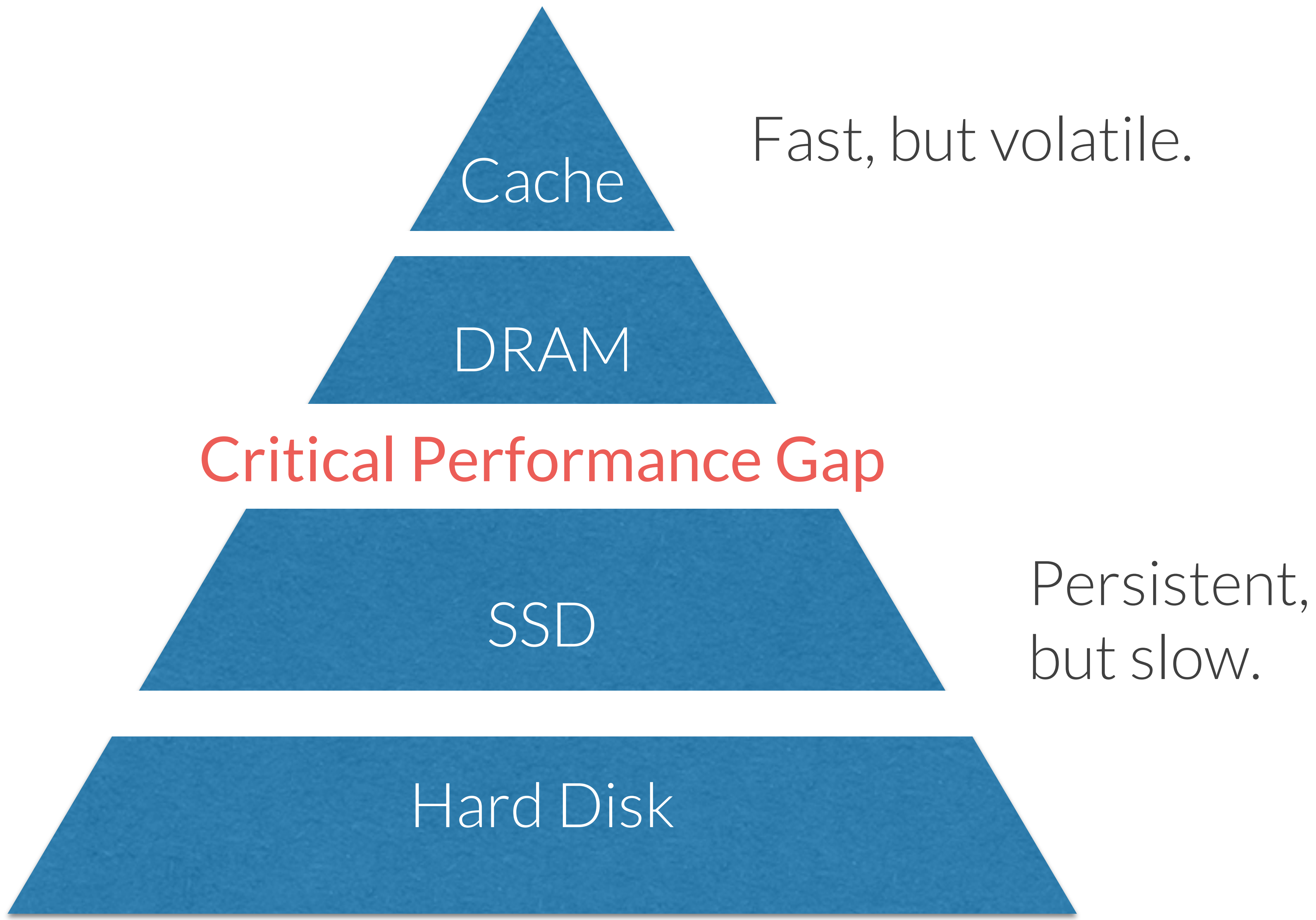
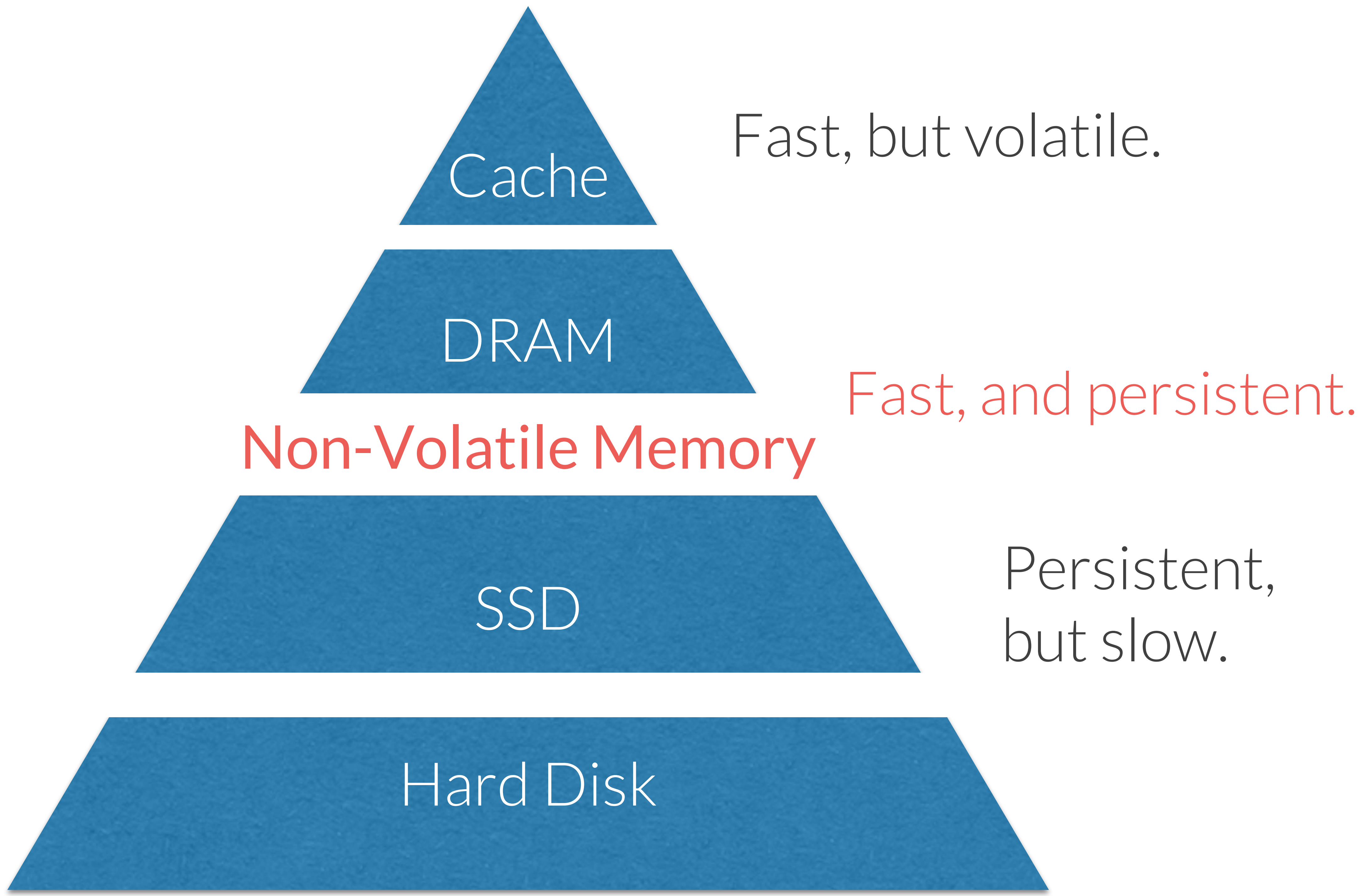
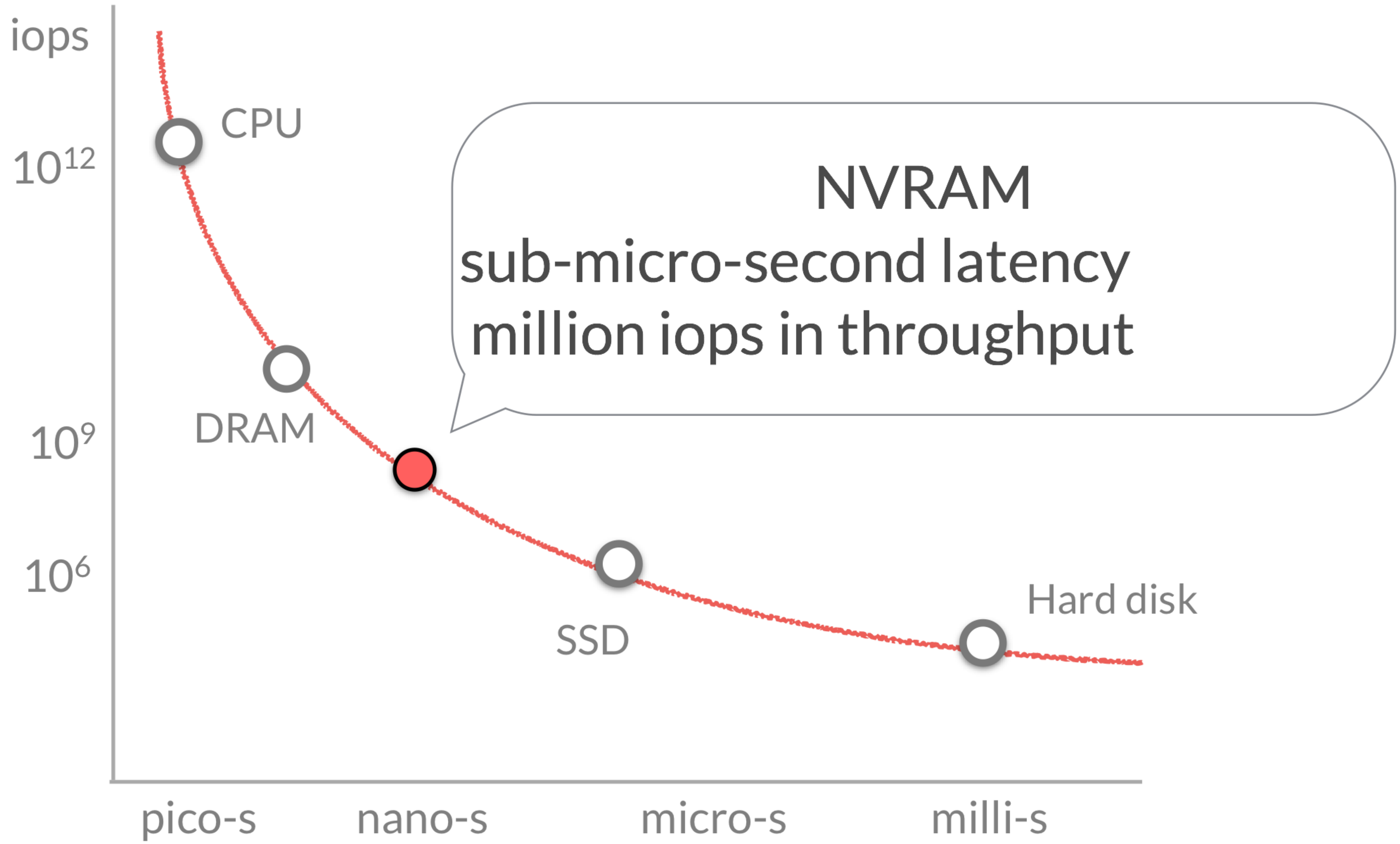
NVMove: Helping Programmers Move to Byte-based Persistence

Himanshu Chauhan

with
Irina Calciu, Vijay Chidambaram,
Eric Schkufza, Onur Mutlu, Pratap Subrahmanyam


Cache
DRAM
Critical Performance Gap
SSD
Hard Disk
Fast, but volatile.
Persistent,
but slow.


Cache
Fast, but volatile.
DRAM
Fast, and persistent.
Non-Volatile Memory
SSD
Persistent,
but slow.
Hard Disk



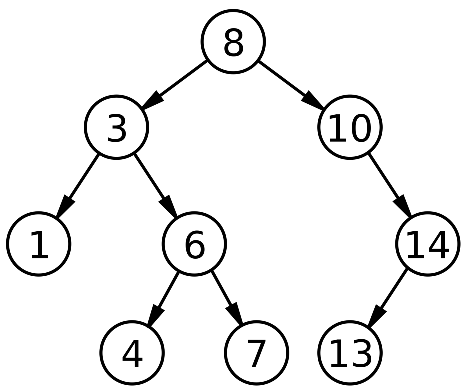
iops
CPU
10^{12}
DRAM
10^{9}
10^{6}
SSD
Hard disk
pico-s
nano-s
micro-s
milli-s
NVRAM
sub-micro-second latency
million iops in throughput

Persistent Programs

```
typedef struct {

} node
```

1. allocate from memory
2. data read/write + program logic
3. save to storage

Persistence Today



In-memory binary search tree

↓ Serialization

Flat Buffer

```
sprintf(buf, “%d:%s”, node->id, node->value)
```

↓

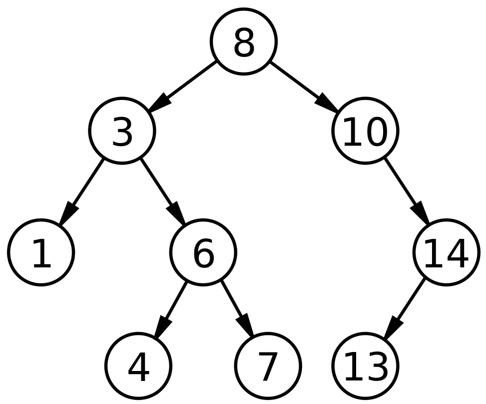
File

```
write(fd, buf, sizeof(buf))
fsync(fd)
```

↓ Block-sized Writes

Block-based Storage

Persistence with NVM



In-memory binary search tree

```
node->id = 10
pmemcopy(node->value, myvalue)

pmemobj_persist(node)
```

Byte-sized Writes

Byte-based NVM

Changing Persistence Code

Present

```
/* allocate from volatile memory*/
node n* = malloc(sizeof(…))

node->value = val //volatile
                    update
    …

/* persist to block-storage*
char *buf= malloc(sizeof(…))
int fd = open("data.db",O_WR
sprintf(buf,"…", node->id,
           node->value);
write(fd, buf, sizeof(buf));
```

NVM

```
/* allocate from non-volatile memory*/
de n* = pmalloc(sizeof(…))

de->value = val //persistent
                    update
…

flush cache and commit*/
ache_flush + __commit
```

Porting to NVM: Tedious

- Identify data structures that should be on NVM
- Update them in a consistent manner

Redis: simple key-value store (~50K LOC)

- Industrial effort to port Redis is on-going after two years
- Open-source effort to port Redis has minimum functionality

Changing Persistence Code

Present

```
/* allocate from volatile memory*/
node n* = malloc(sizeof(…))

node->value = val //volatile
                    update
    …

/* persist to block-storage
char *buf= malloc(sizeof(…)
int fd = open("data.db",O_W
sprintf(buf,"…", node->id,
           node->value);
write(fd, buf, sizeof(buf));
```

NVM

```
/* allocate from non-volatile memory*/
node n* = pmalloc(sizeof(…))

de->value = val //persistent
                  update
 …

 flush cache and commit*/
cache_flush + __commit
```

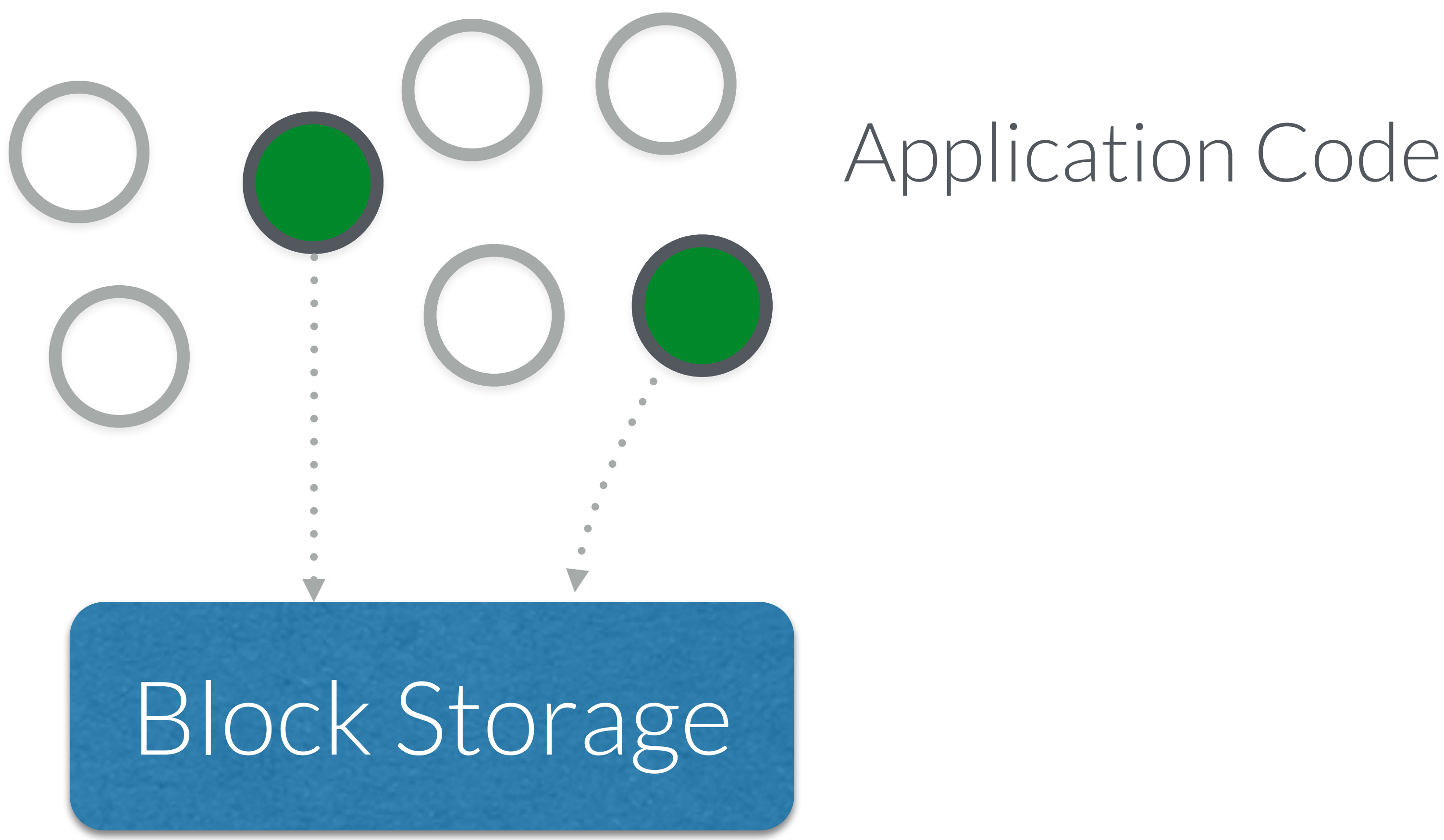
Goal:

Port existing applications to NVM with minimal programmer involvement.

DRAG CHUTE
OXYGEN
INSTRUMENTATION CONTROL
RECORD
ON
OFF
MOM.
OSC.
P.R.
XFT 3492 SER # 4
GUNS SIGHT CAMERA & RADAR
OFF
SIGHT CAMERA & RADAR
FIRE WARNING PUSH TO TEST
HYDRAULIC PRESSURE
SYS. 1
SYS. 2
UTILITY
RUD. ALT.
RADIO CALL
53-504
KNOTS
g UNITS
YAW
CLIMB
DIVE
PERCENT RPM
OIL PRESS
P.S.I.
ALT
MACH NUMBER
4 MIN TURN
EXHAUST
°CX100
CRUISE
ENG. PRESSURE RATIO
TAKE OFF
FUEL FLOW
FUEL QUANTITY
LBS x 100
FUEL QUANTITY PUSH TO TEST
RADIO COMPASS
SLIP D.C. ELECT.
A.C. ELEC
D.C. ELEC
EMERGENCY UP
ON
OFF
LANDING & TAXI LIGHT
UP
DOWN
LANDING GEAR POSITION
HORN CUTOUT
LANDING GEAR
EMERGENCY JETTISON
OPEN
OIL COOLER
FUEL PURGE
OFF
AIR START
ENG MASTER
OXYGEN REGULATOR
PRESSURE
FLOW

Persistent Types in Source

User defined source types (structs in C) that are persisted to block-storage.



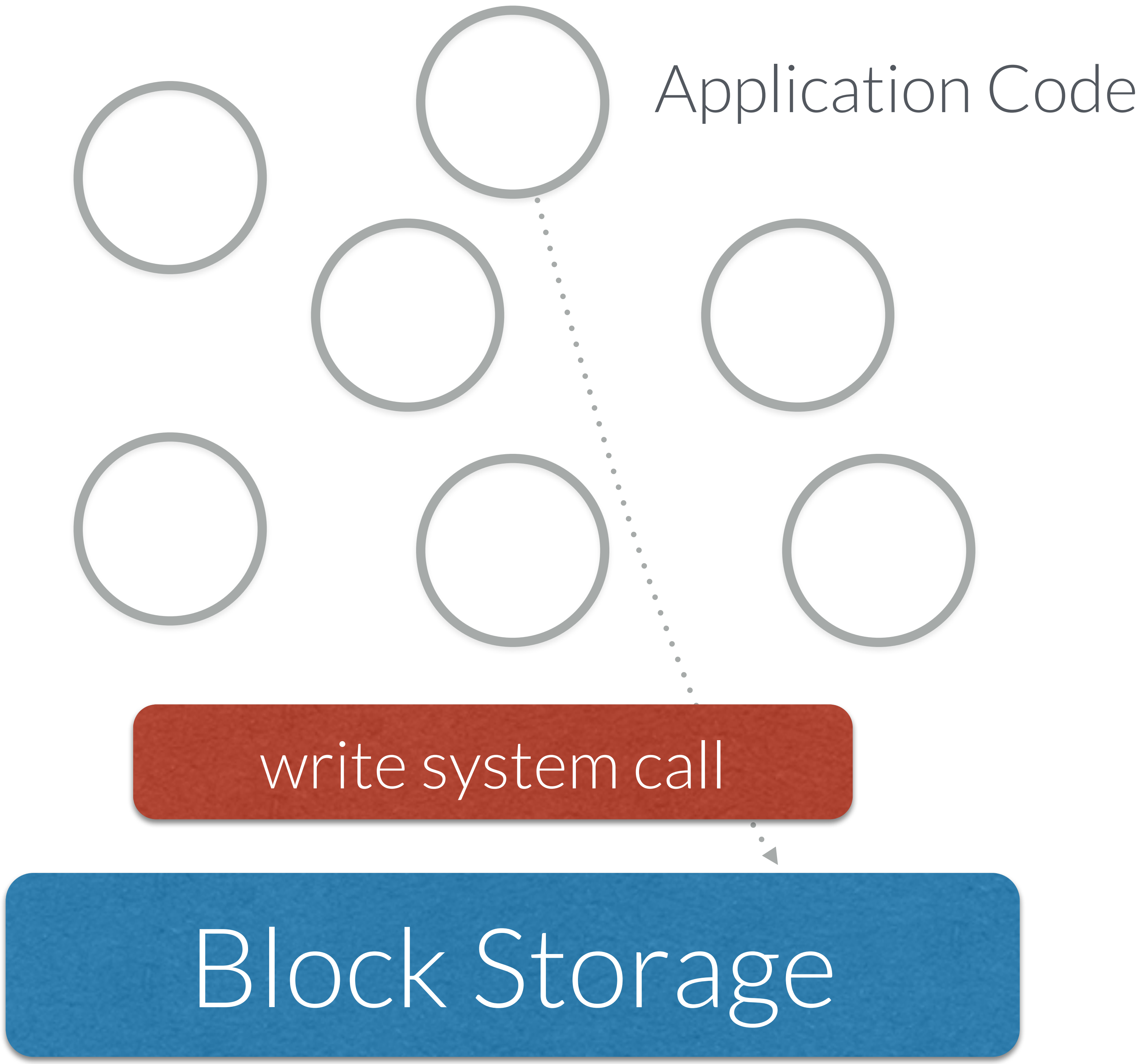
First Step:

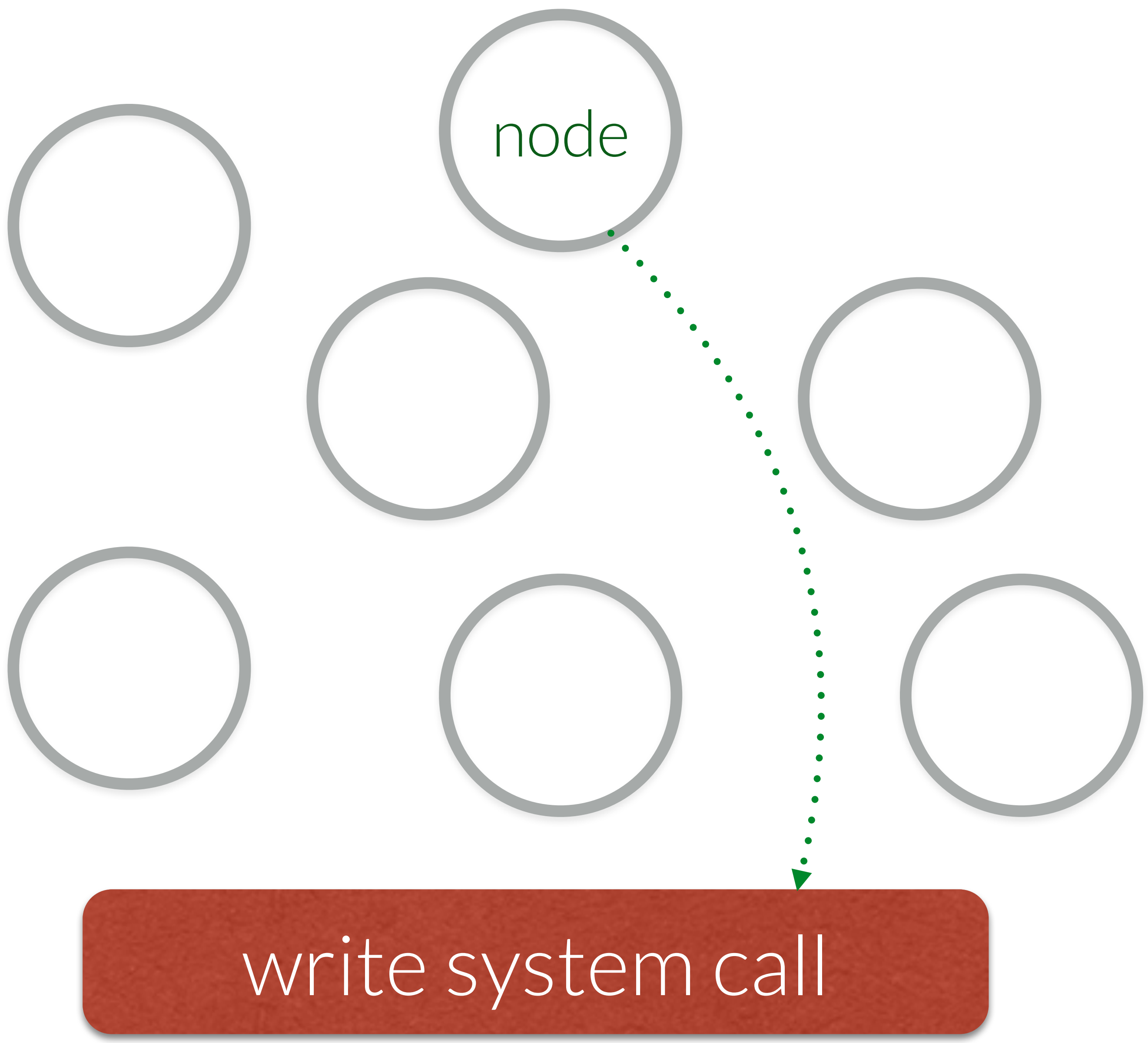
Identify persistent types in application source.

Solution: Static Analysis

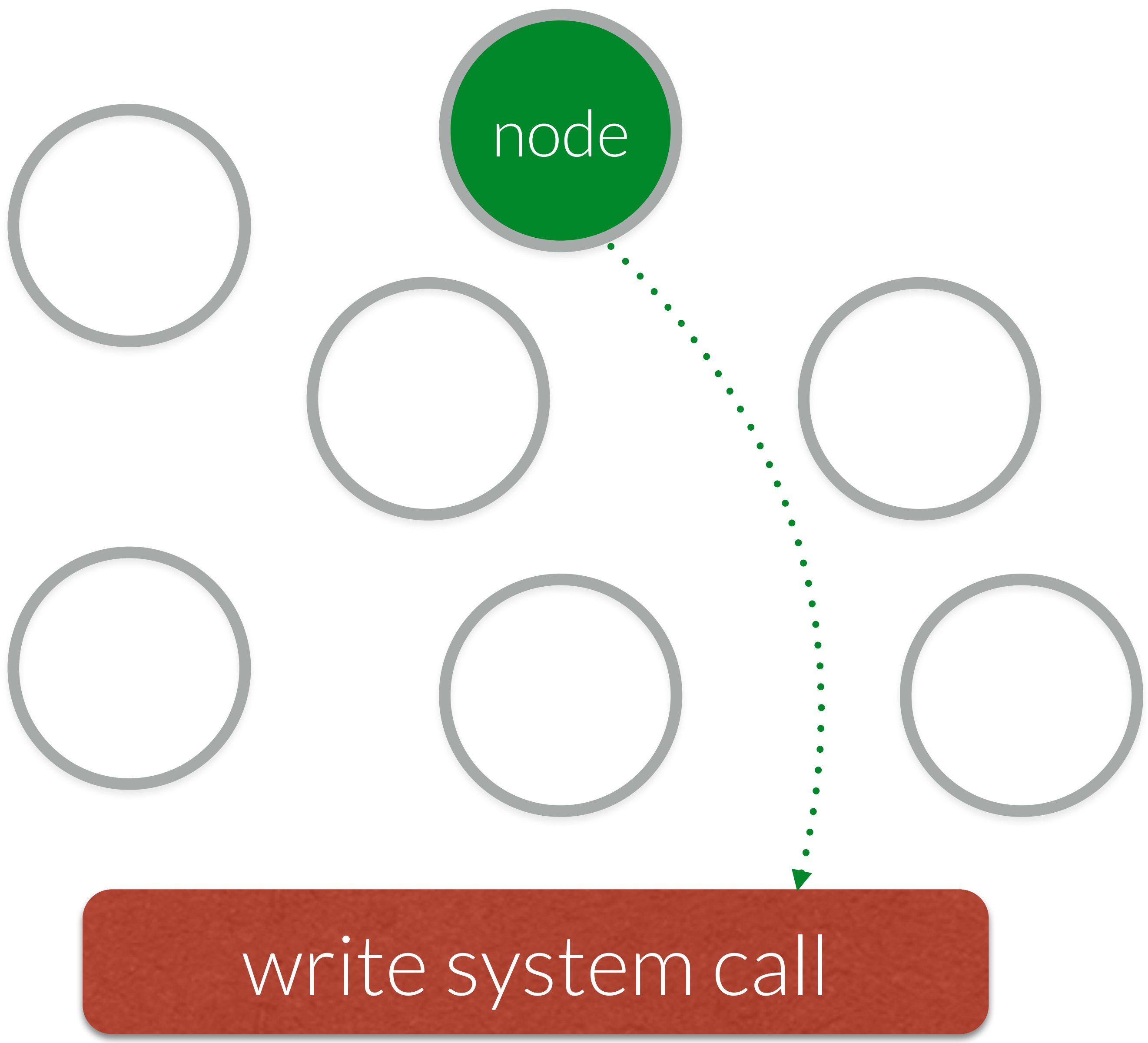
Current Focus: C

types = structs

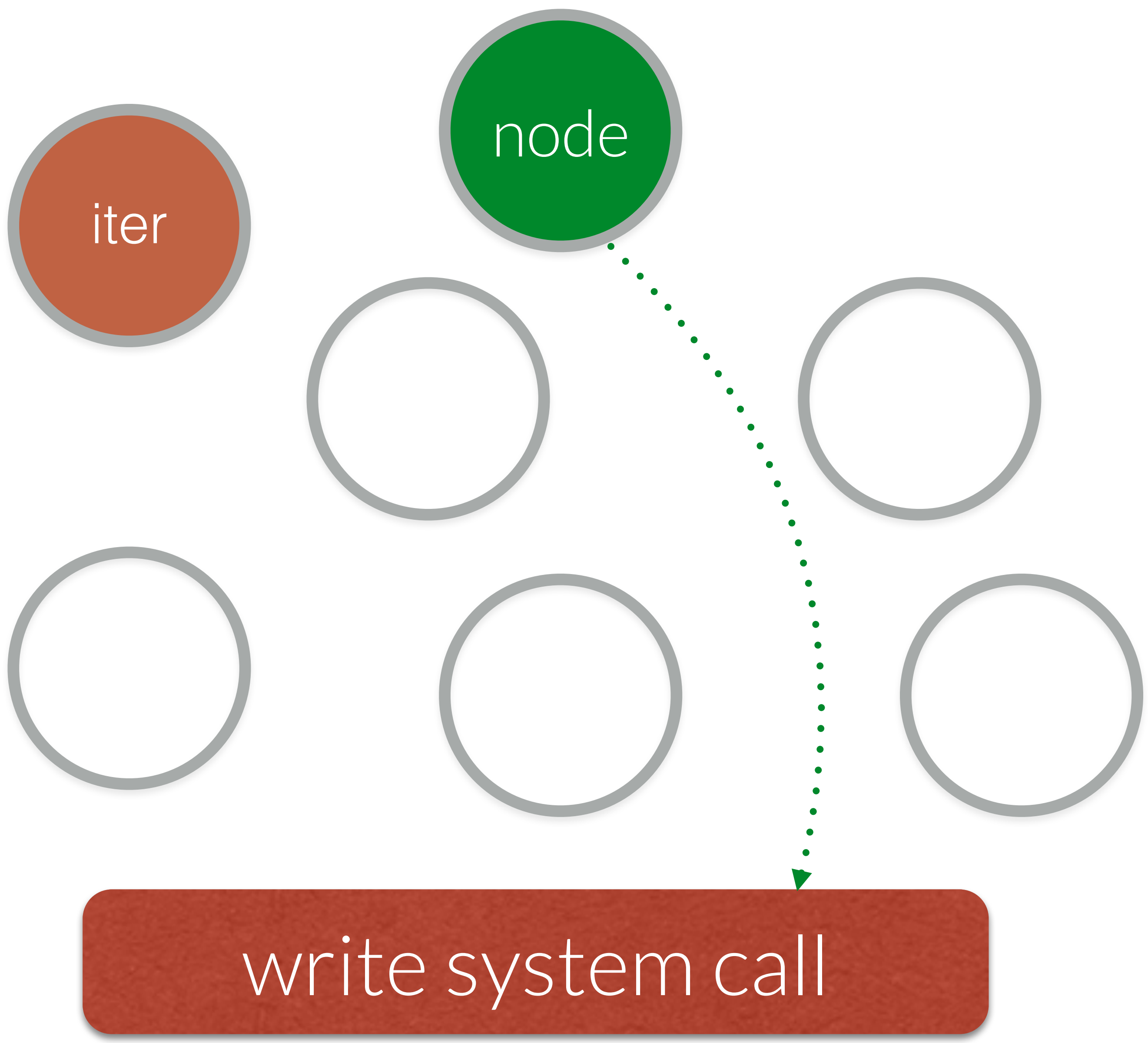

Application Code
write system call
Block Storage



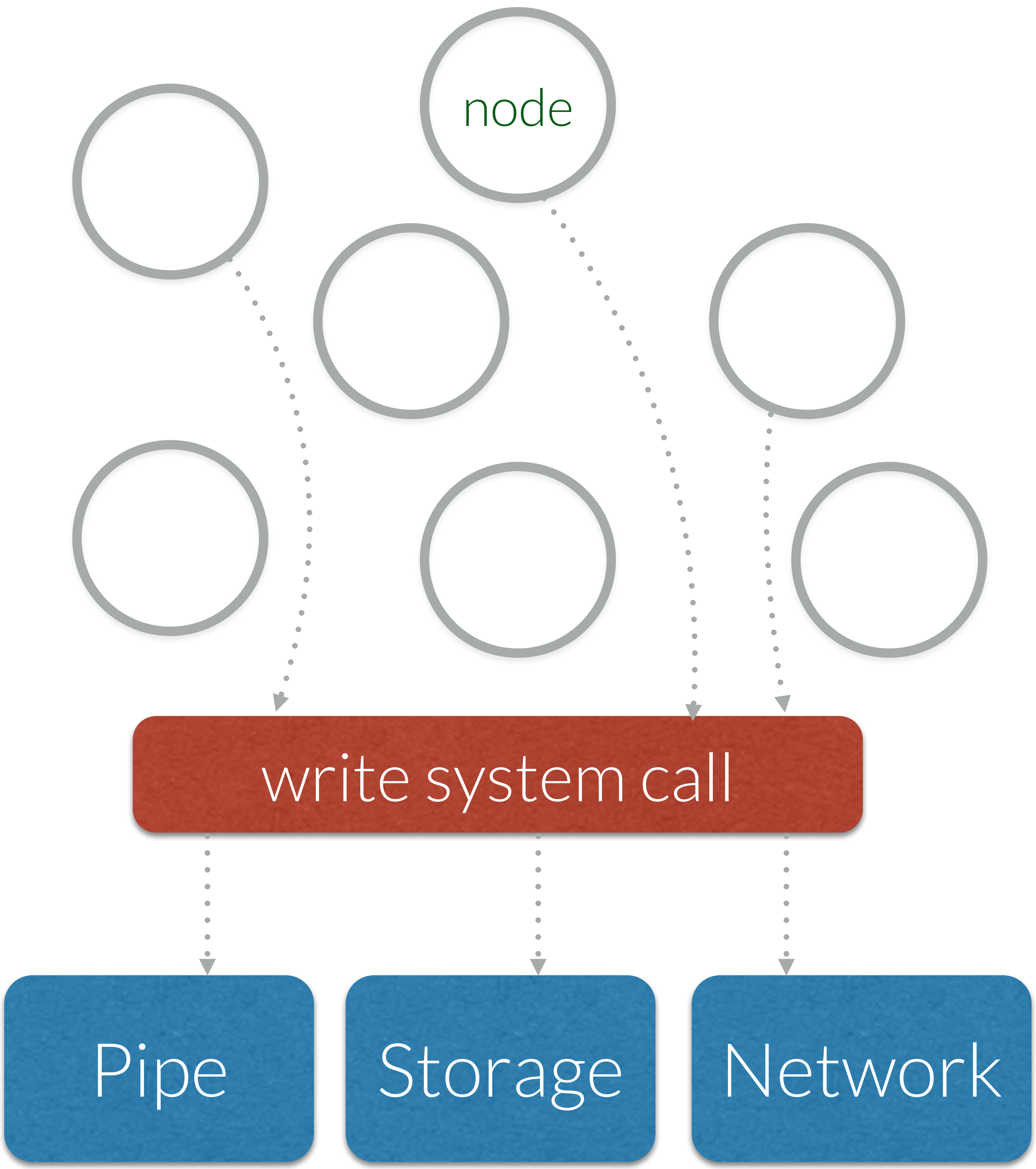

```
node *n = malloc(sizeof(node))

iter *it = malloc(sizeof(iter))

/* persist to block-storage*/
char *buf= malloc(...))
int fd = open(...)

sprintf(buf,"...",node->value)

write(fd, buf, ...)
```



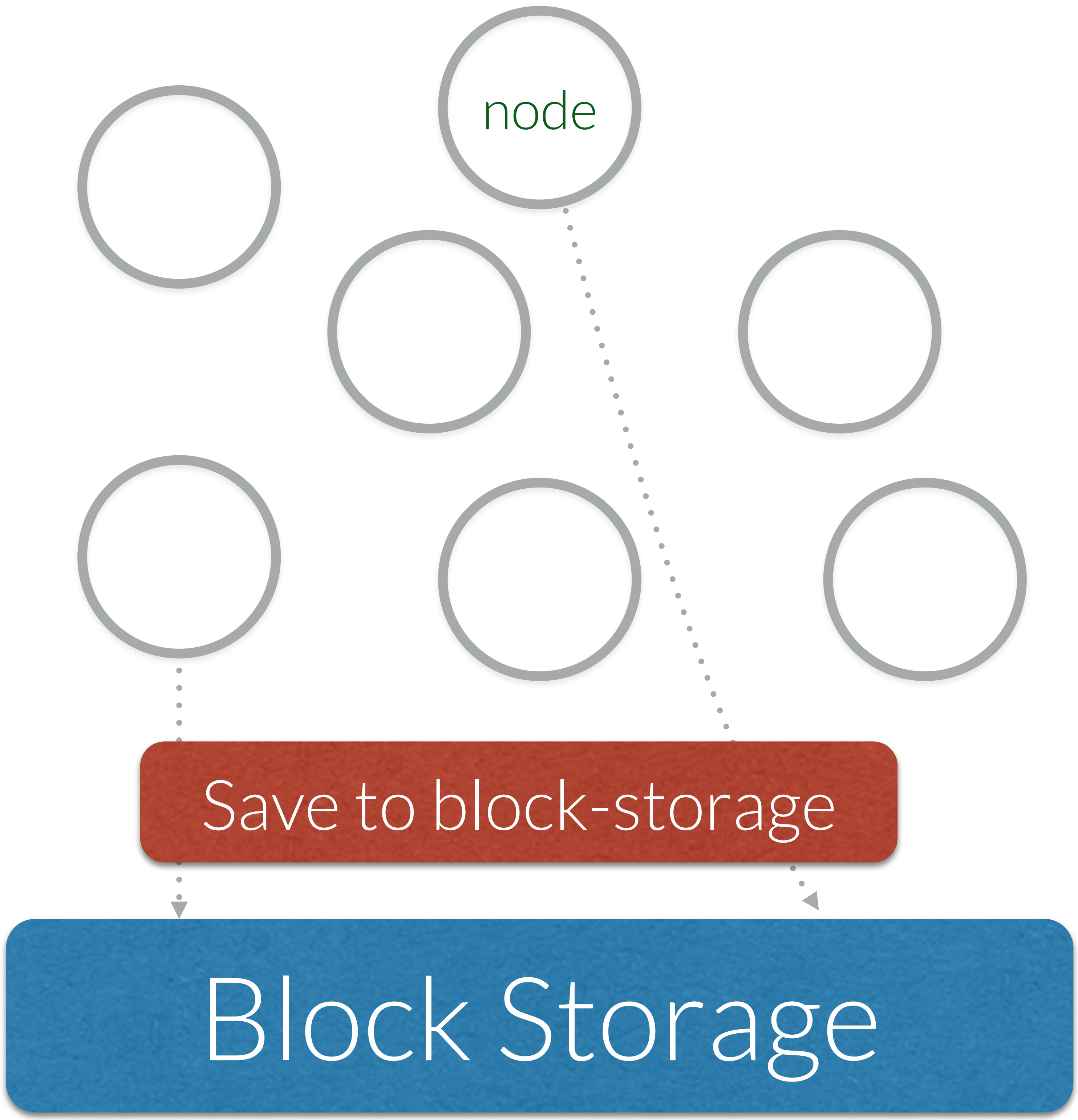

```
node *n = malloc(sizeof(node))

iter *it = malloc(sizeof(iter))

/* persist to block-storage*/
char *buf= malloc(...))
int fd = open(...)

sprintf(buf,"...",node->value)

write(fd, buf, ...)
```

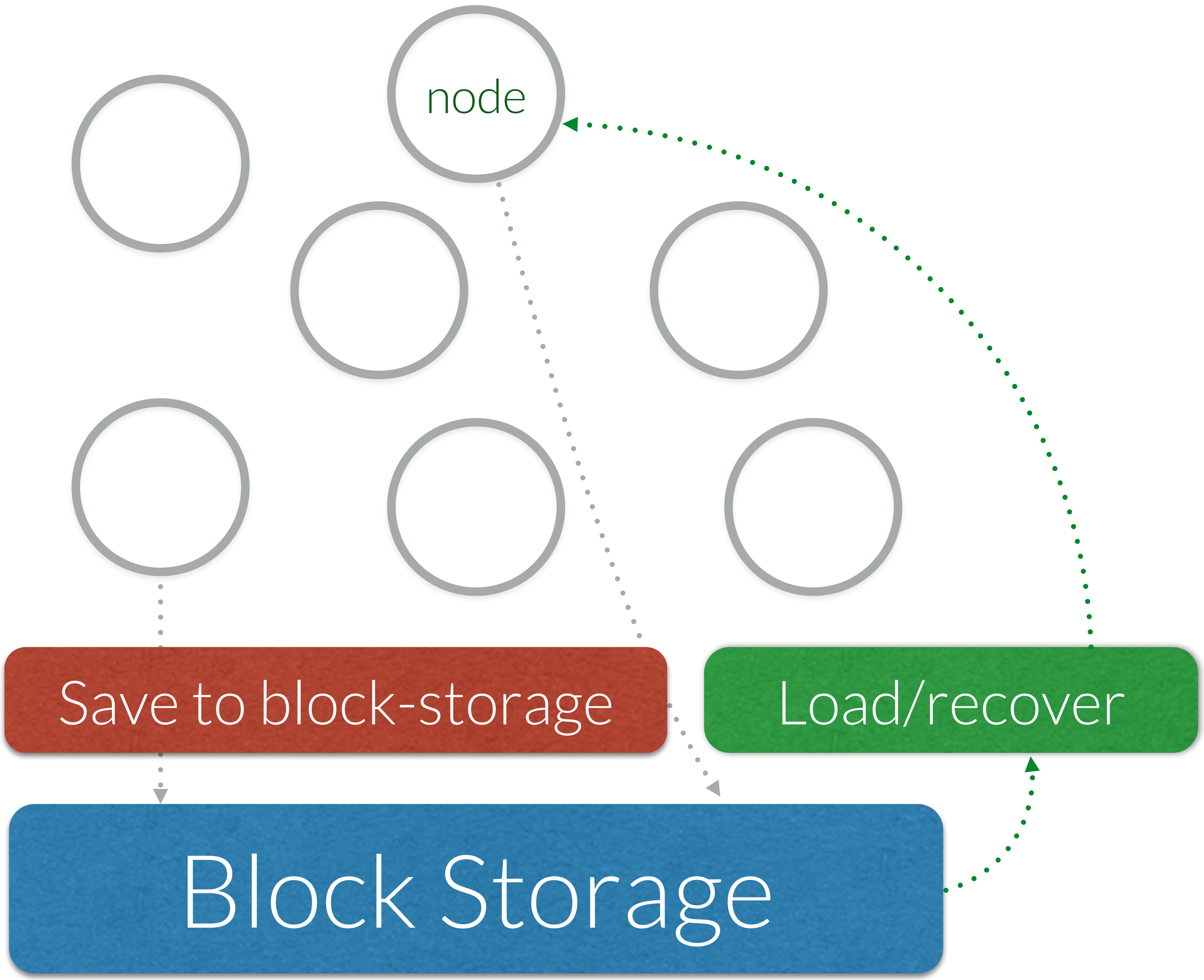



```
node *n = malloc(sizeof(node))

iter *it = malloc(sizeof(iter))

/* persist to block-storage*/
char *buf= malloc(...))
int fd = open(...)

sprintf(buf,"...",node->value)

write(fd, buf, ...)
```




```
/* write to network socket*/
…
write(socket, “404”, …)

/* write to error stream*/
…
write(stderr, “All is lost.”, …)

/* persist to block-storage*/
…
write(fd, buf, …)
```


node
Save to block-storage
Block Storage


node
Save to block-storage
Load/recover
Block Storage

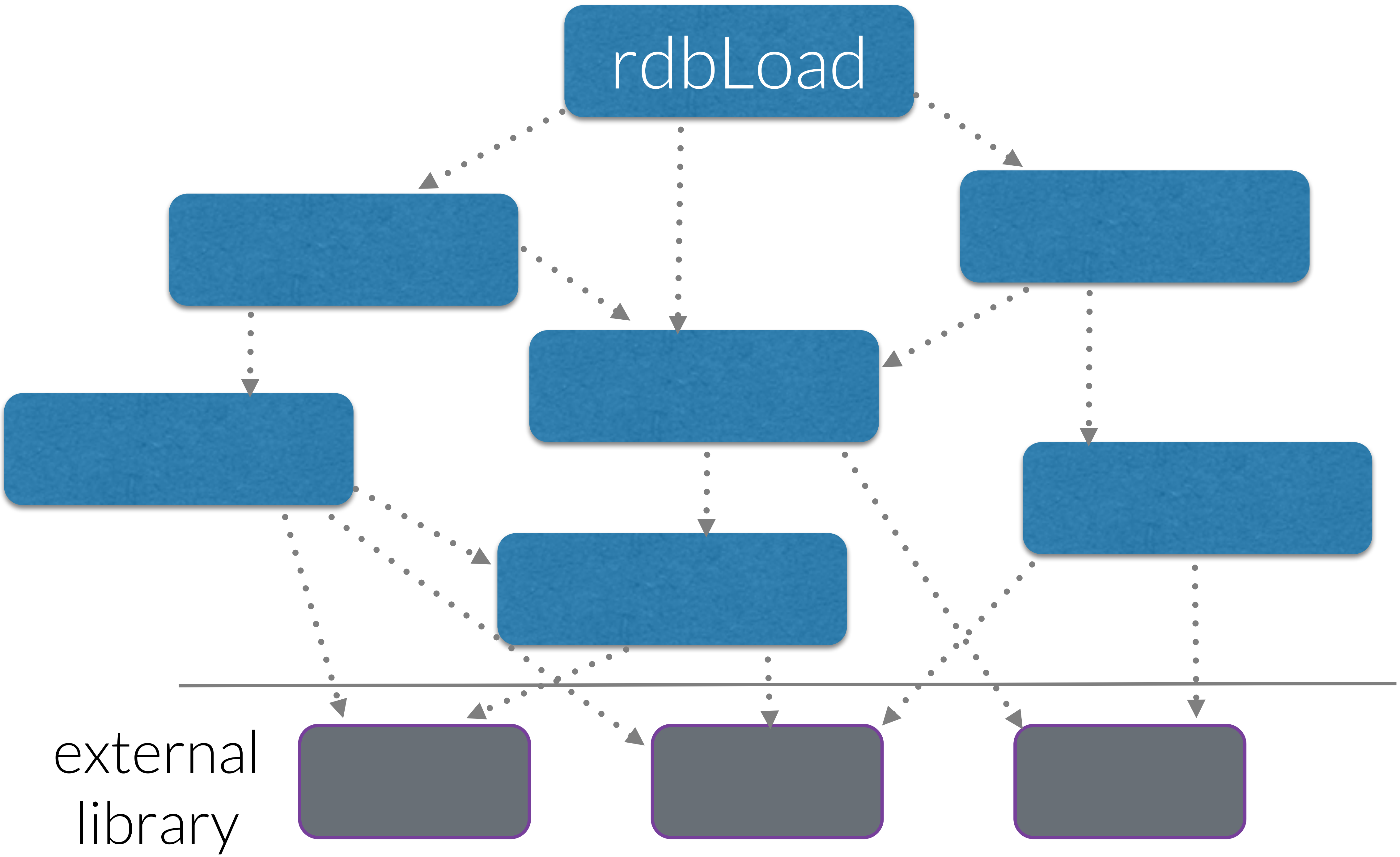
"rdbLoad" is the load/recovery function.

```c
int rdbLoad(char *filename) {
    uint32_t dbid;
    int type, rdbver;
    redisDb *db = server.db+0;
    char buf[1024];
    long long expiretime, now = mstime();
    FILE *fp;
    rio rdb;

    if ((fp = fopen(filename,"r")) == NULL) return C_ERR;

    rioInitWithFile(&rdb,fp);
    rdb.update_cksum = rdbLoadProgressCallback;
    rdb.max_processing_chunk = server.loading_process_events_interval_bytes;
    if (rioRead(&rdb,buf,9) == 0) goto eoferr;
    buf[9] = '\0';
    if (memcmp(buf,"REDIS",5) != 0) {
        fclose(fp);
        serverLog(LL_WARNING,"Wrong signature trying to load DB from file");
        errno = EINVAL;
        return C_ERR;
    }
    rdbver = atoi(buf+5);
```

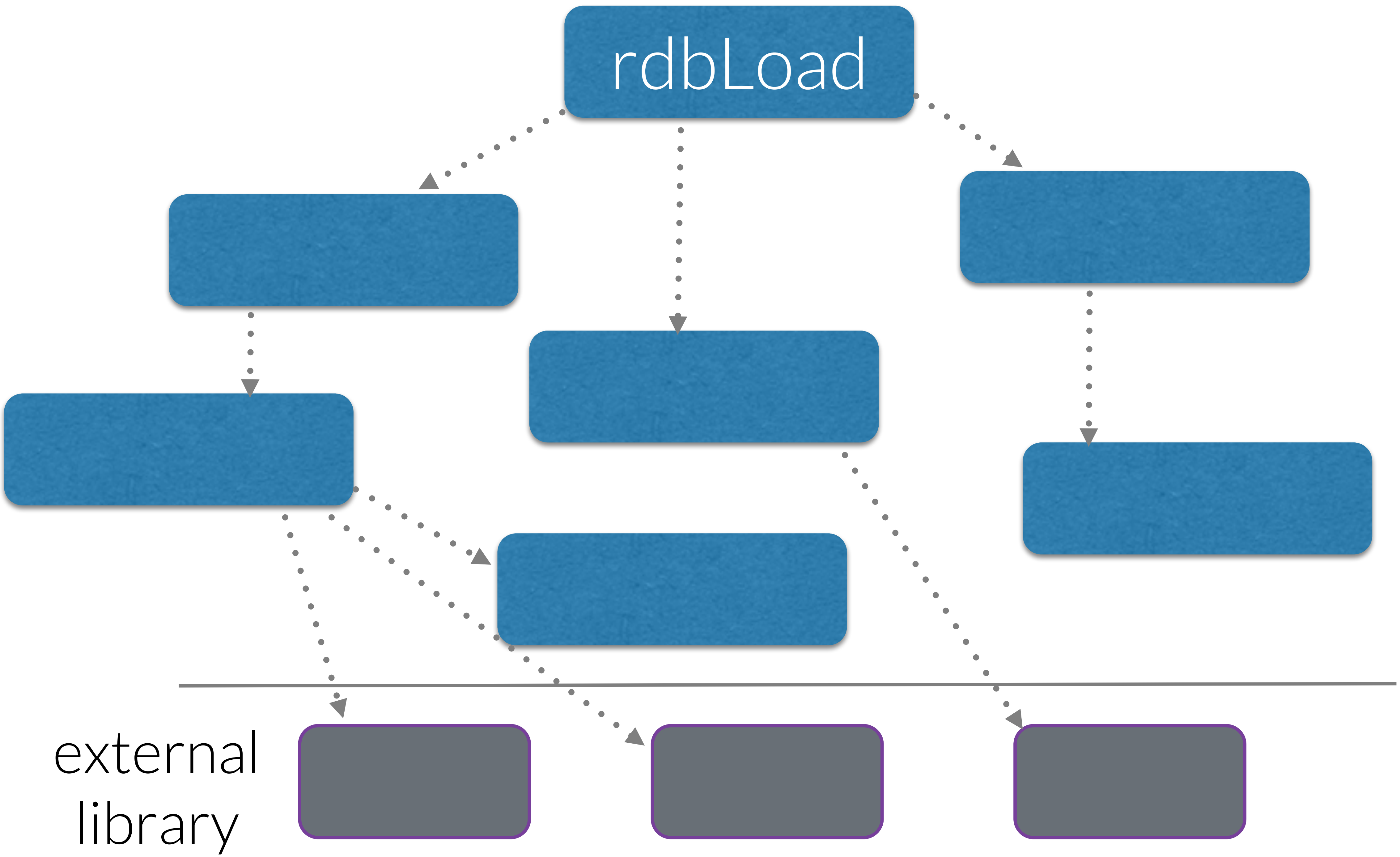
Mark every type that can be created during the recovery.

*if defined in application source.

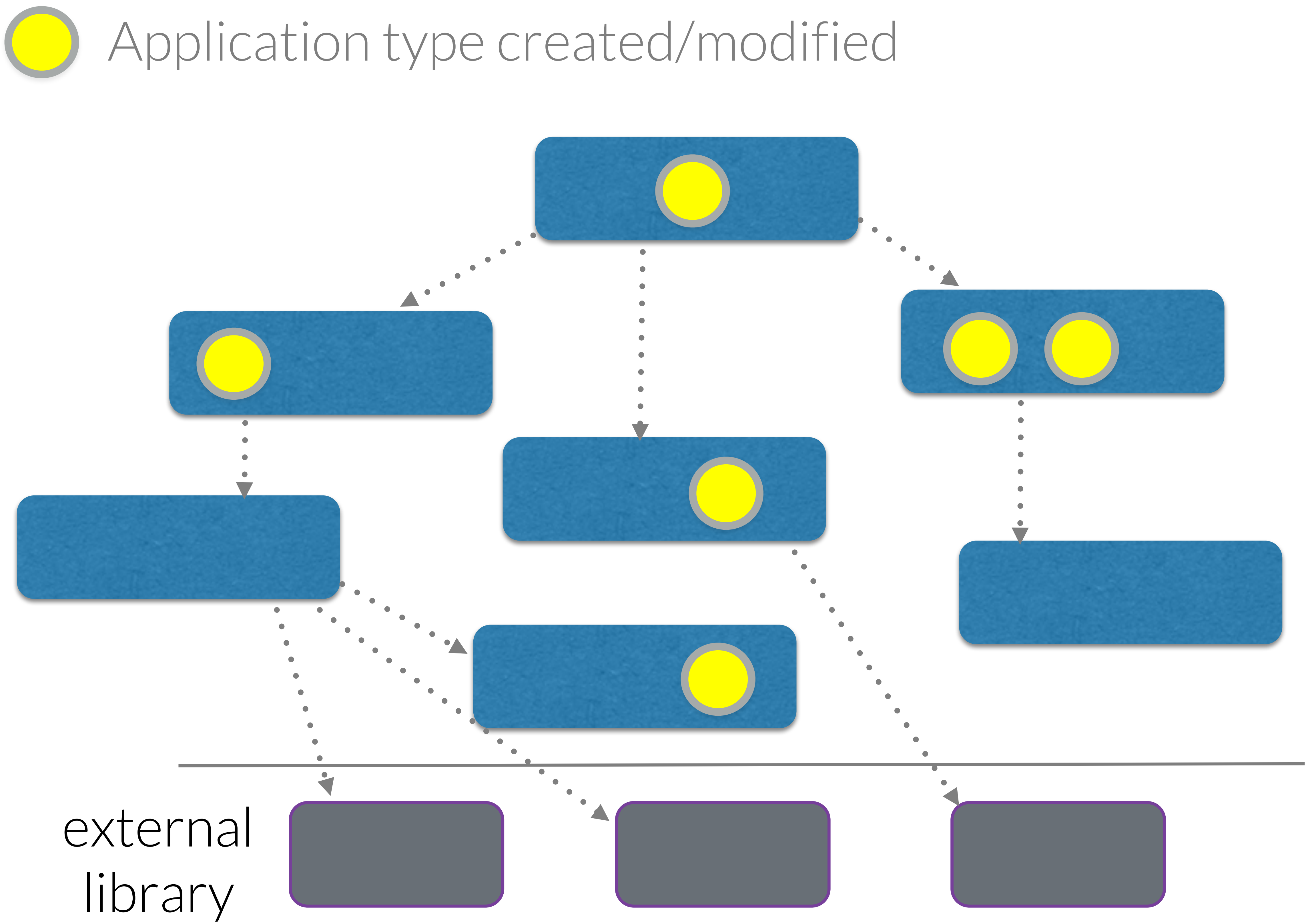
Call Graph from Load



BFS on Call Graph from Load



BFS on Call Graph from Load



NVMove Implementation

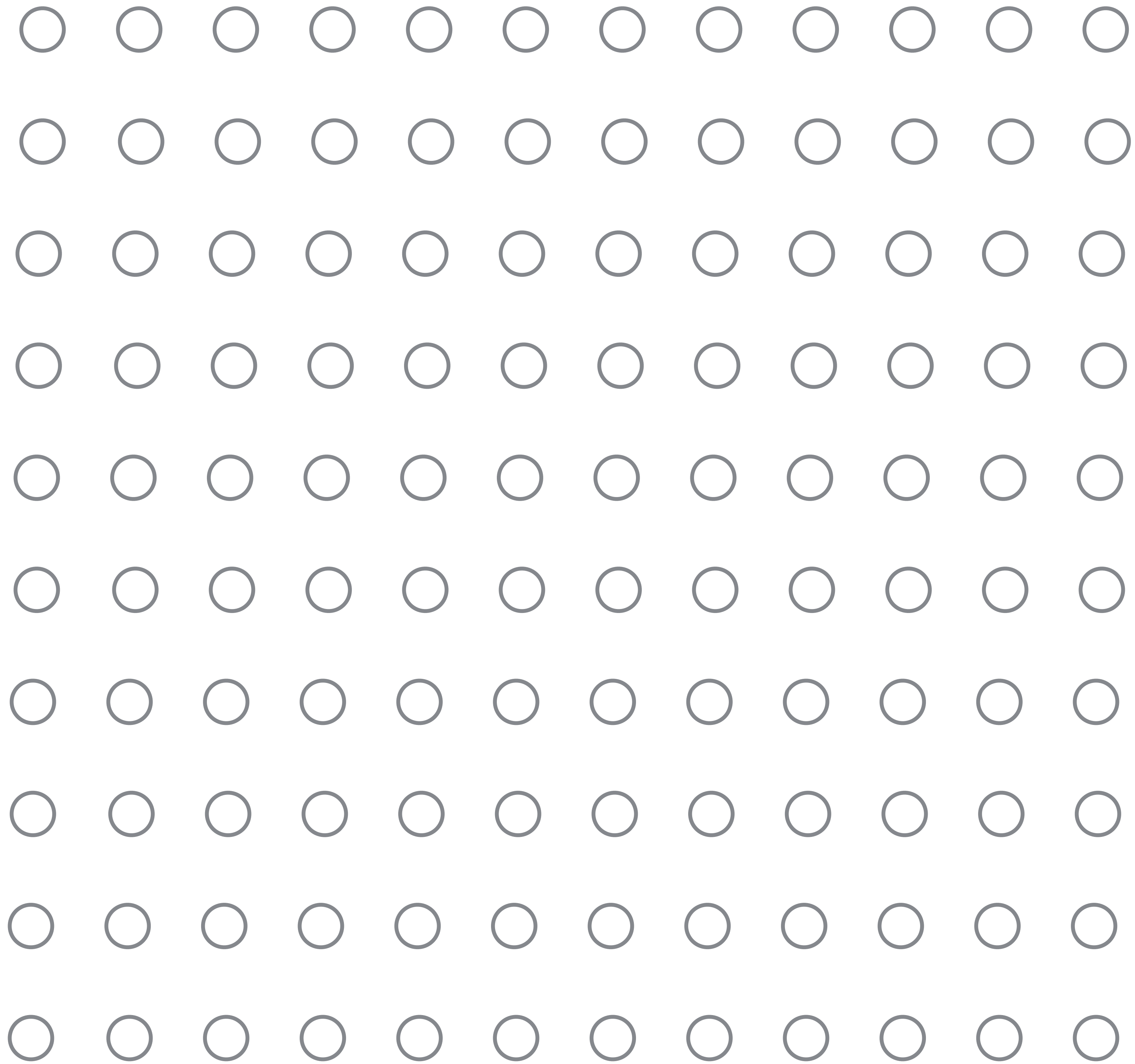
- Clang
  - Frontend Parsing
- Parse AST and Generate Call Graph
  - Find all statements that create/modify application types in graph
- Currently supports C applications

Evaluation

redis

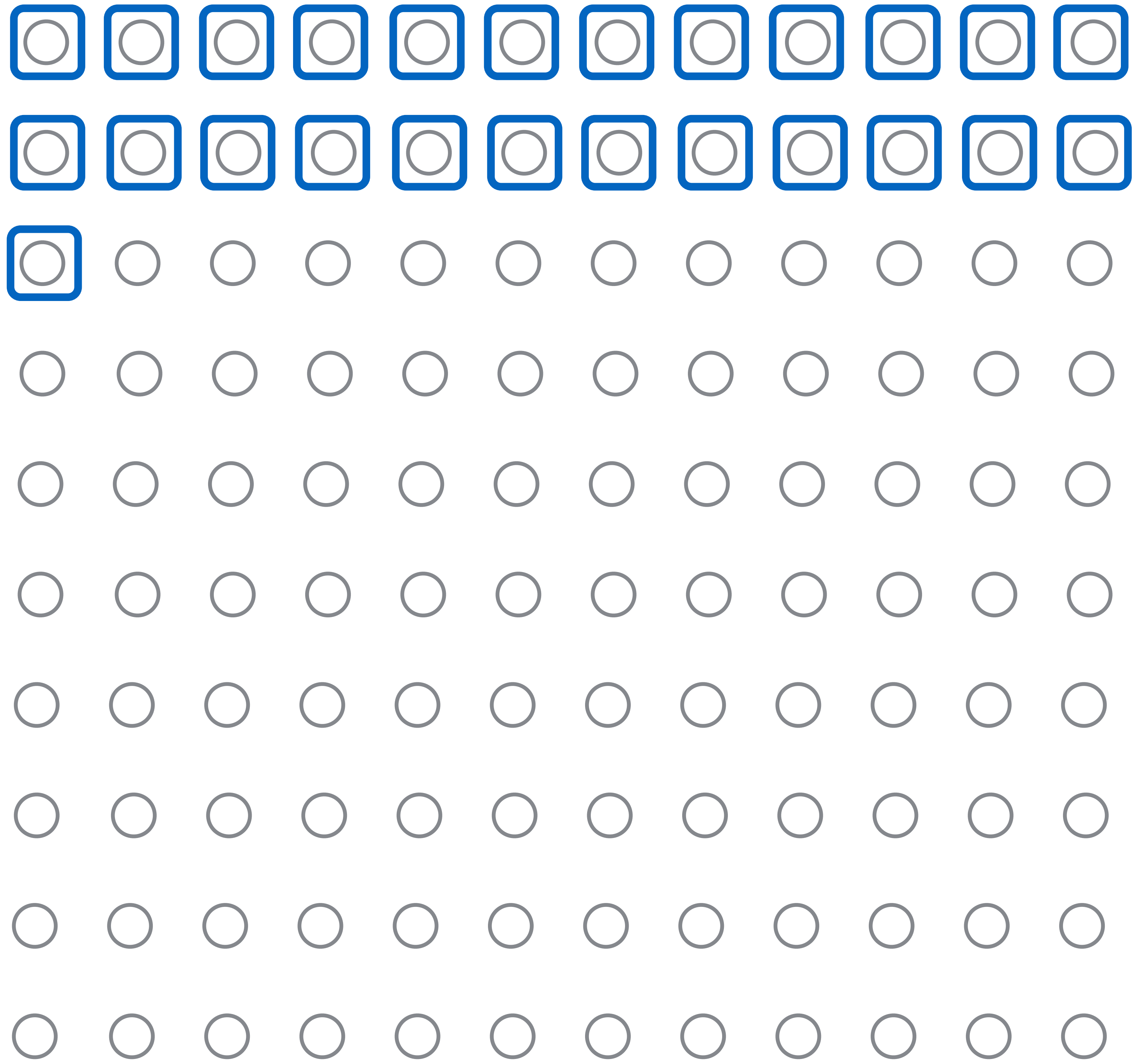
- In-memory data structure store
  - strings, hashes, lists, sets, indexes
- On-disk persistence
  - data-snapshots(RDB),
  - command-logging (AOF)
- ~50K lines-of-code

Identification Accuracy

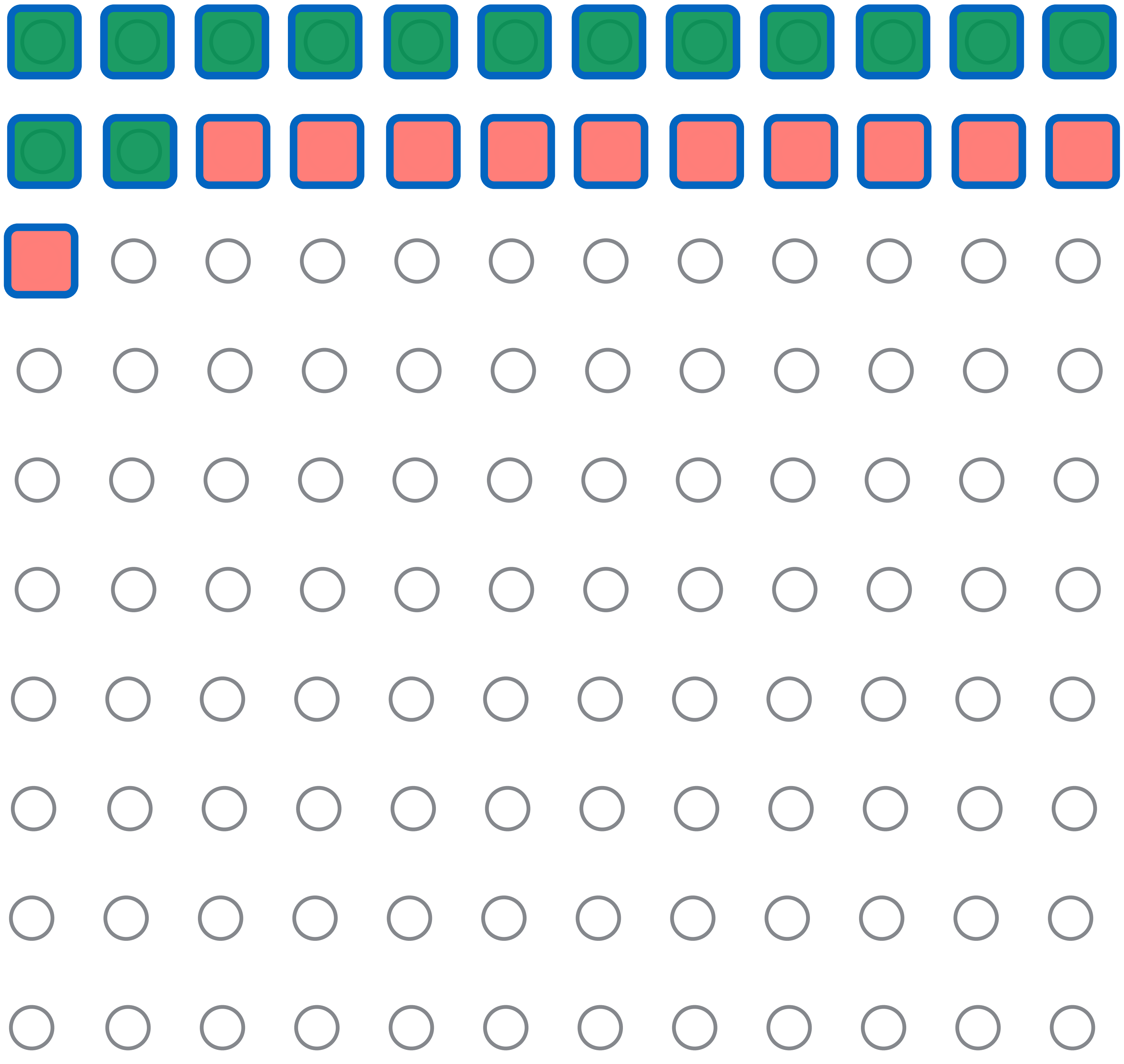


122 types (structs) in Redis Source

Identification Accuracy



Identification Accuracy



Identification Accuracy

Total types	122
NVMove identified persistent types	25
True positives (manually identified)	14
False positives	11
False negatives	0

Performance Impact

Redis Persistence

Snapshot (RDB)

- Data snapshot per second
- Not fully durable

Logging (AOF)

- Append each update command to a file
- Slow

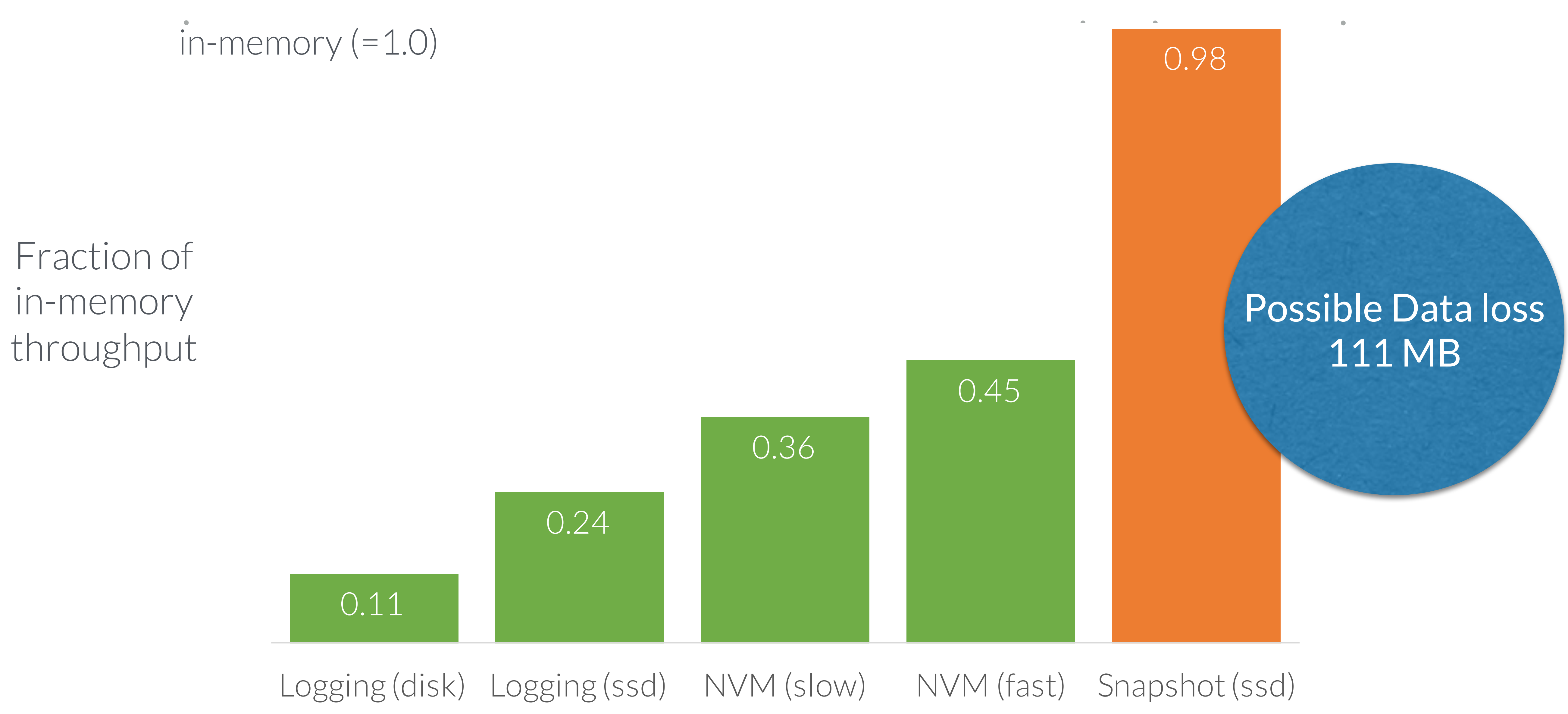
Both performed by forked background process.

NVM Emulation

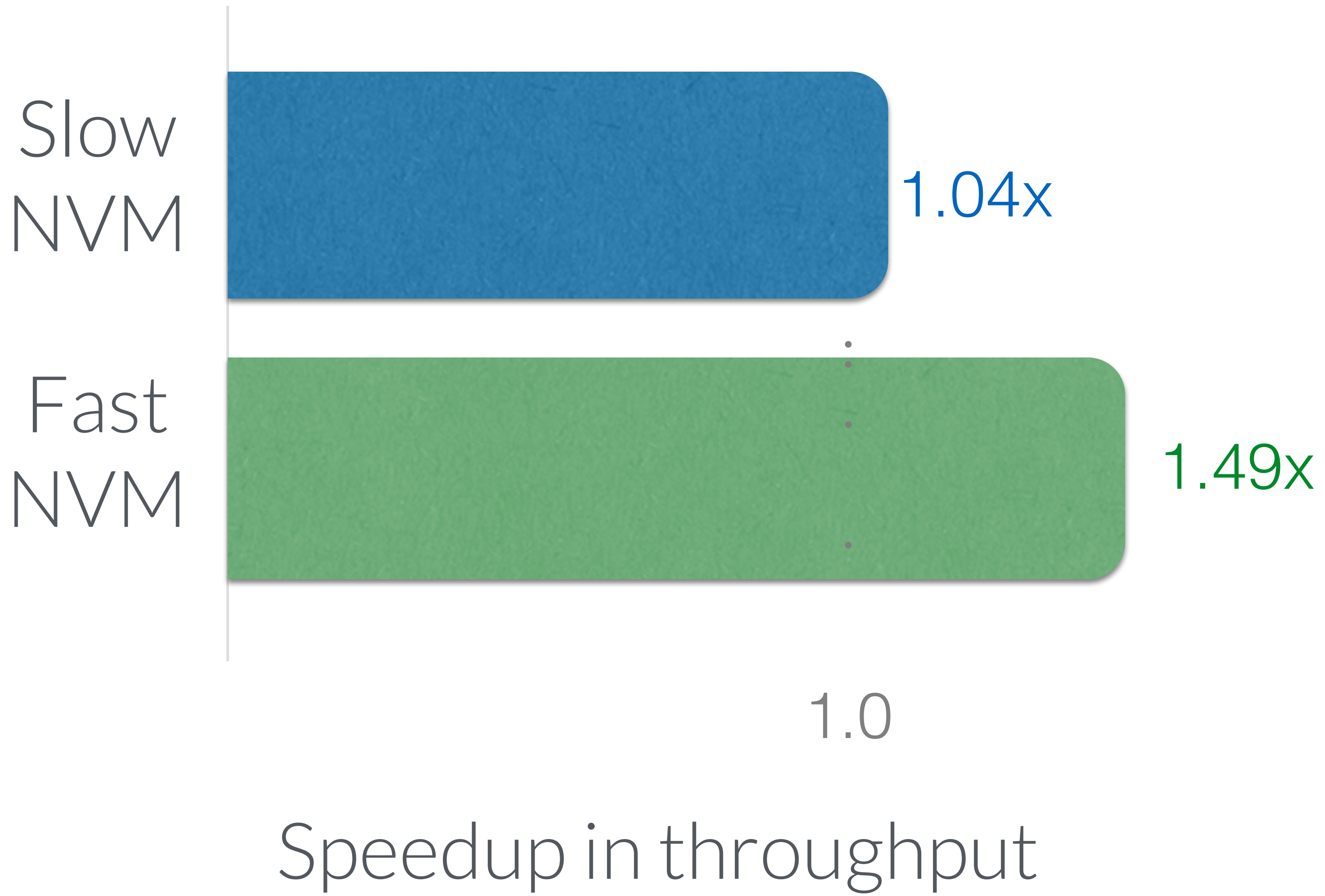
- Emulate allocation of NVMove identified types on NVM heap
  - Slow and Fast NVM
  - Inject delays for load/store of all NVM allocated types.
    - Worst-case performance estimate.
- Compare emulated NVM throughput against logging, and snapshot based persistence.

YCSB Benchmark Results

write-heavy (90% updated, 10% read ops)



Performance without False-Positives



First Step:

Identify persistent types in application source.

Next steps:

- Improve identification accuracy.
- Evaluate on other applications.

Backup

Throughputs (ops/sec)

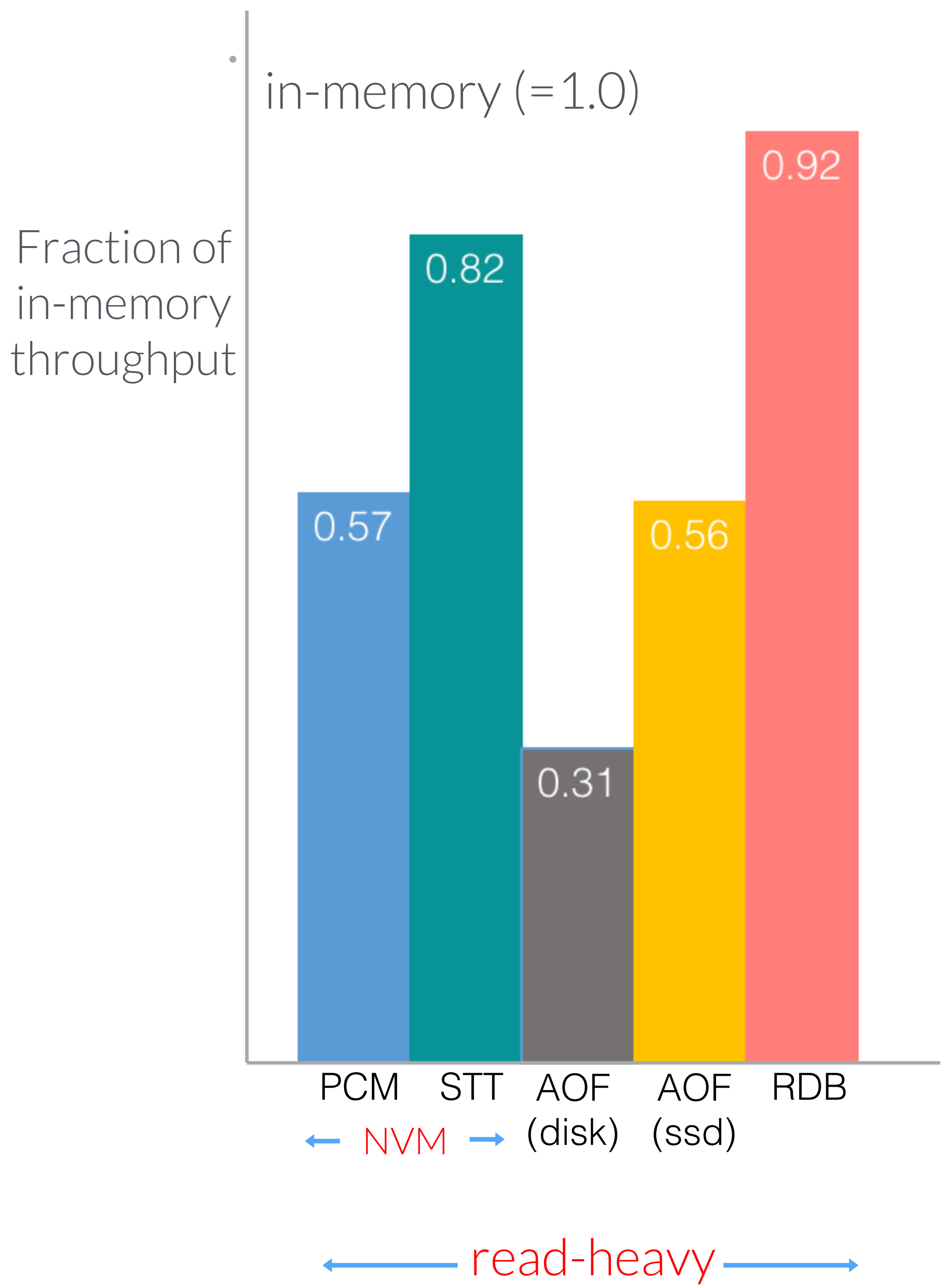
	readheavy	balance	writeheavy
PCM	28399	25,302	9759
STTRam	41213	38,048	12155
AoF (disk)	15634	6,457	2868
AoF (SSD)	27946	17,612	6605
RDB	46355	47,609	26605
Memory	50163	48,360	27156

NVM Emulation

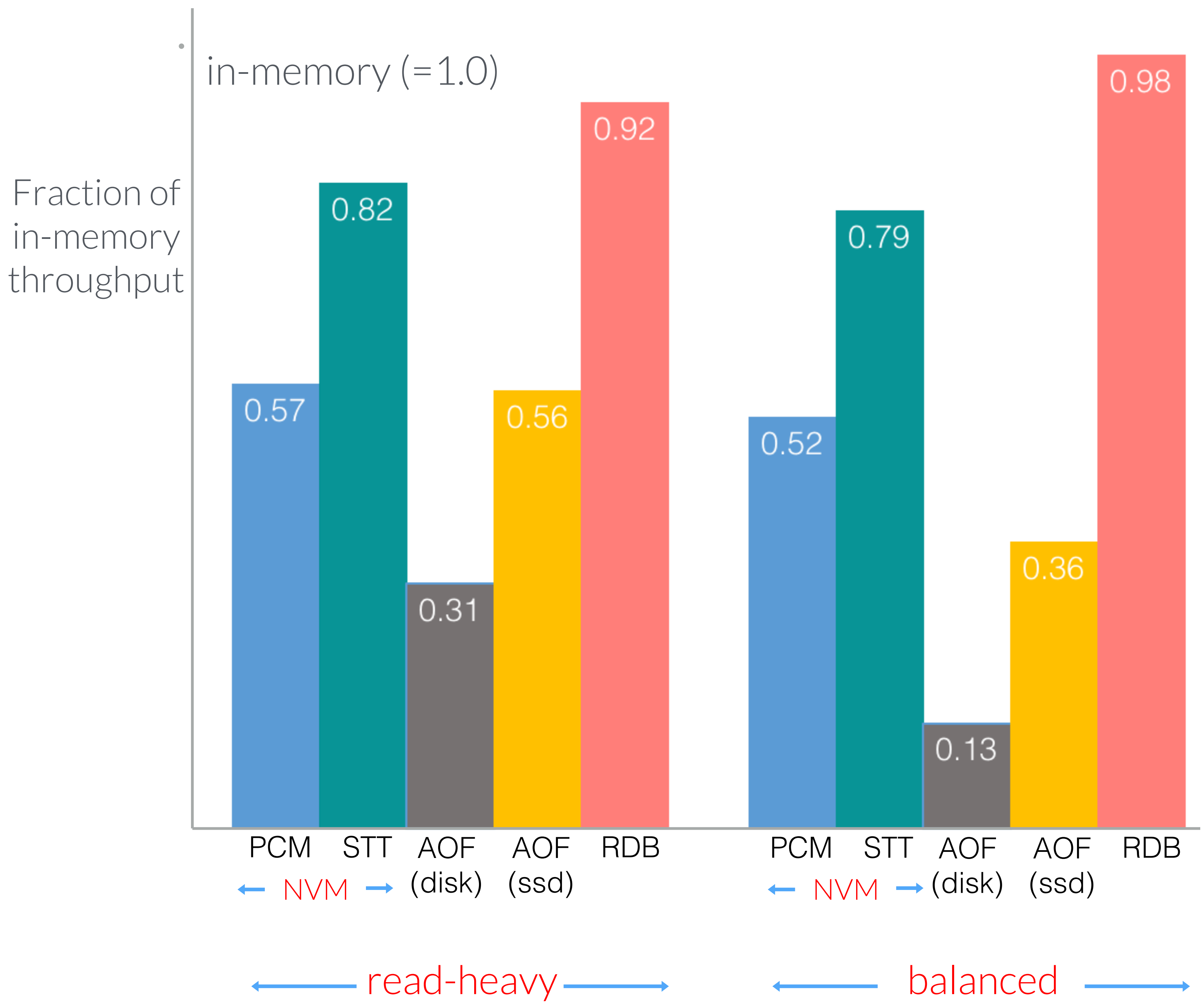
	Read Latency	Cache-line Flush Latency	PCOMMIT Latency
STT-RAM (Fast NVM)	100 ns	40 ns	200 ns
PCM (Slow NVM)	300 ns	40 ns	500 ns

*Xu & Swanson, NOVA: A Log-structured File System for Hybrid Volatile/Non-volatile Main Memories, FAST16.

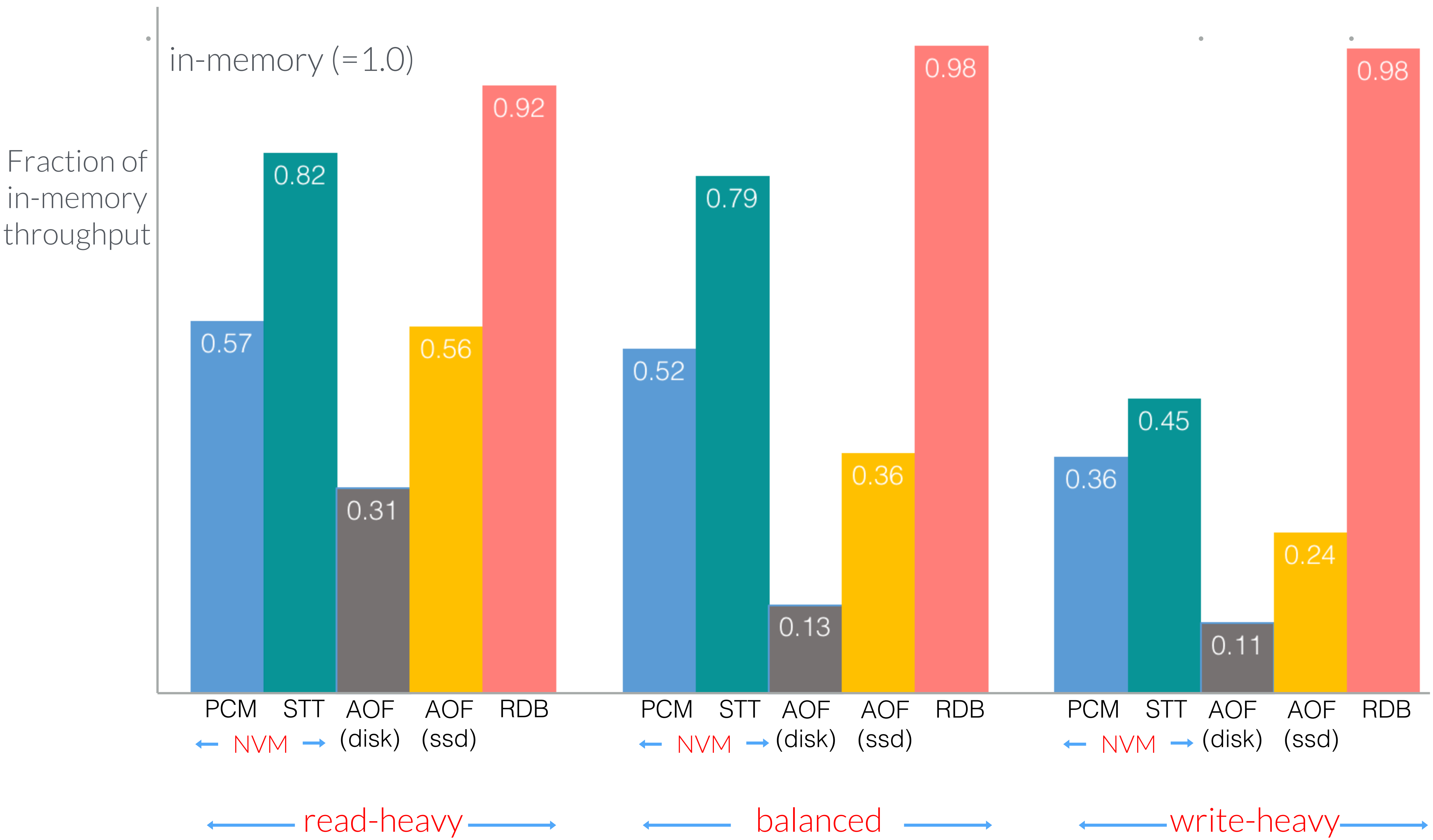
YCSB Benchmark Results



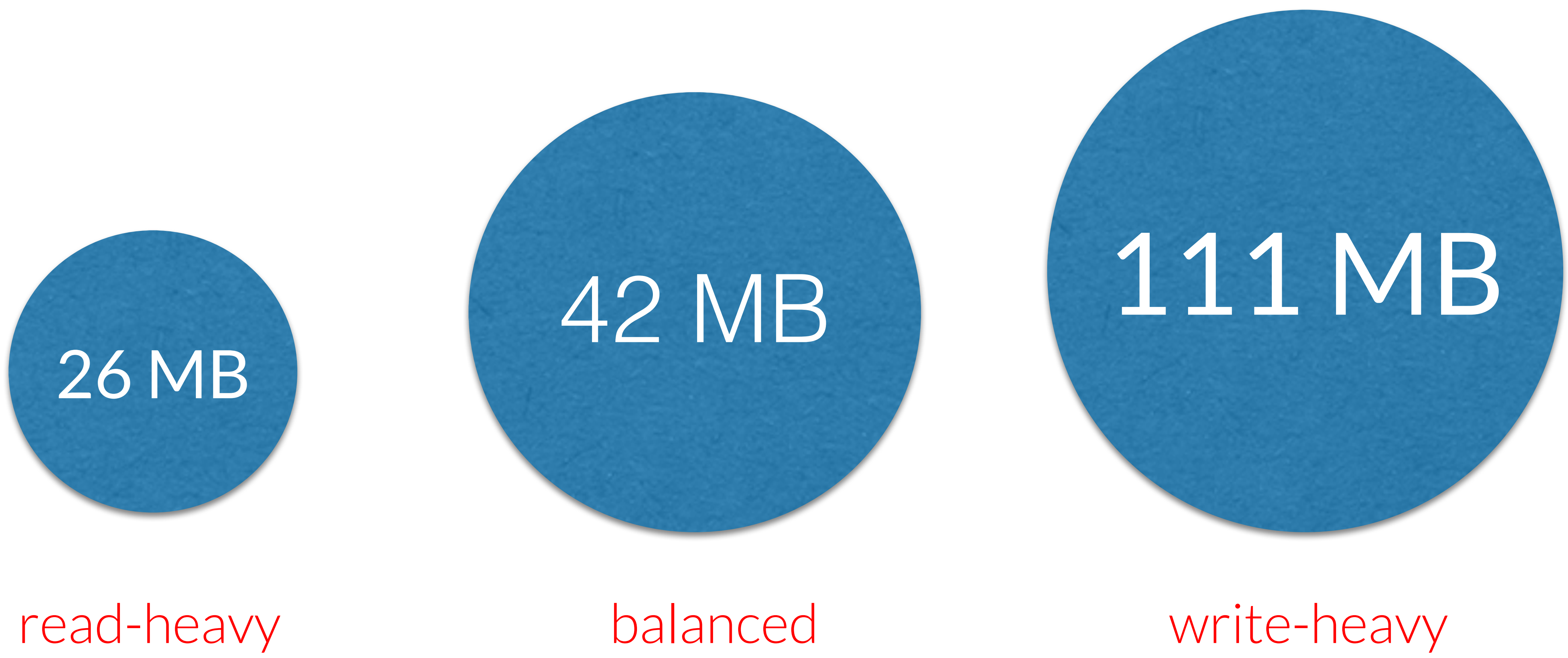
YCSB Benchmark Results



YCSB Benchmark Results



RDB Data Loss



Performance without False-Positives