Efficiently and Precisely Locating Memory Leaks and Bloat

Hound: C/C++ Leak & Bloat Detector

Leak & Bloat ?

- Leak
  - Reachability leaks(GCable, unreachable)
  - Staleness leaks(unused, reachable)
- Bloat
  - Unnecessary excess memory consumption

Problem: Memory Inefficiency

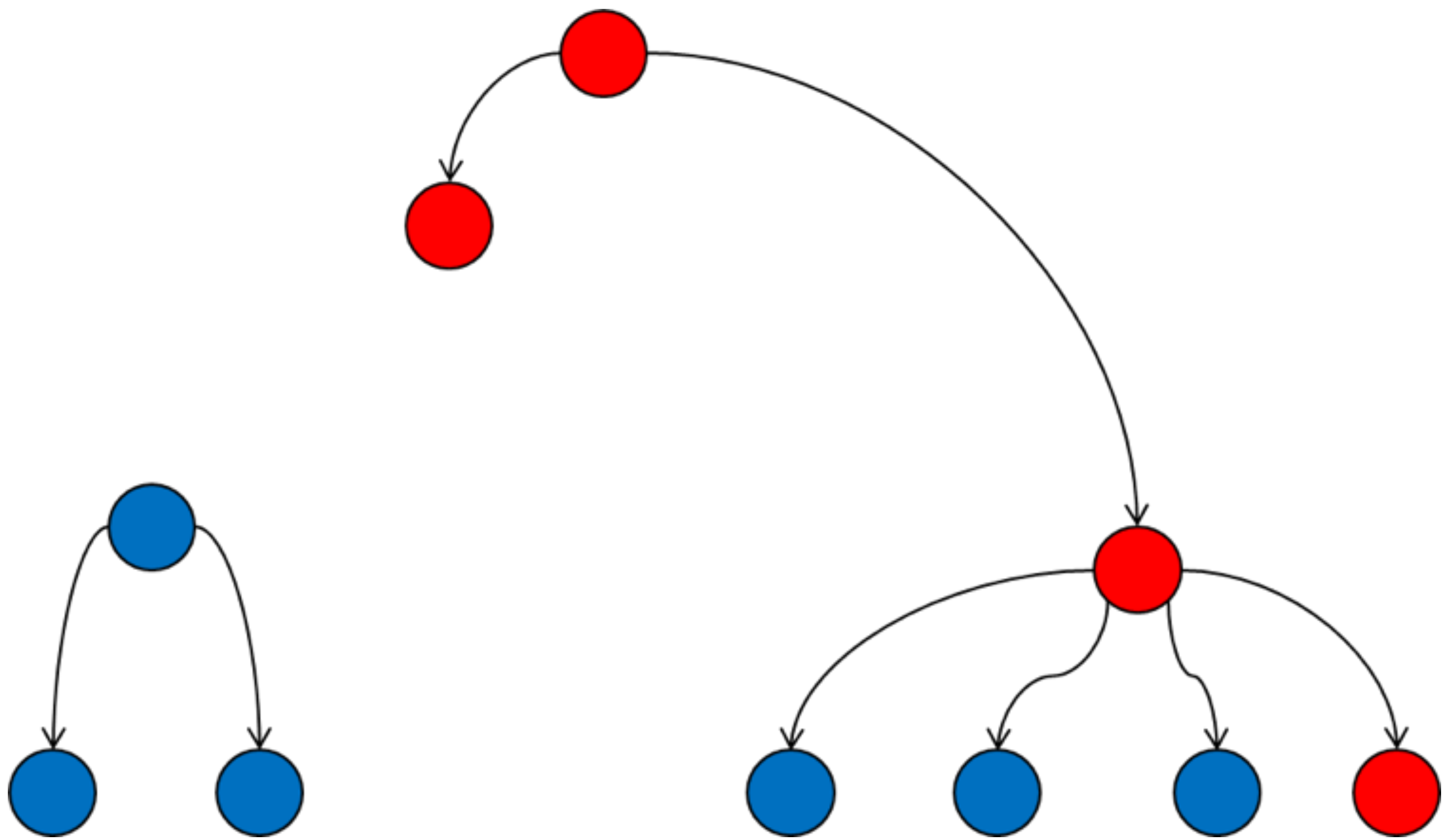
- Jan/Feb 2008, over 150 leak-related bugs were reported

- Ideal world: Collect high-precision leak reports from real programs with low-overhead

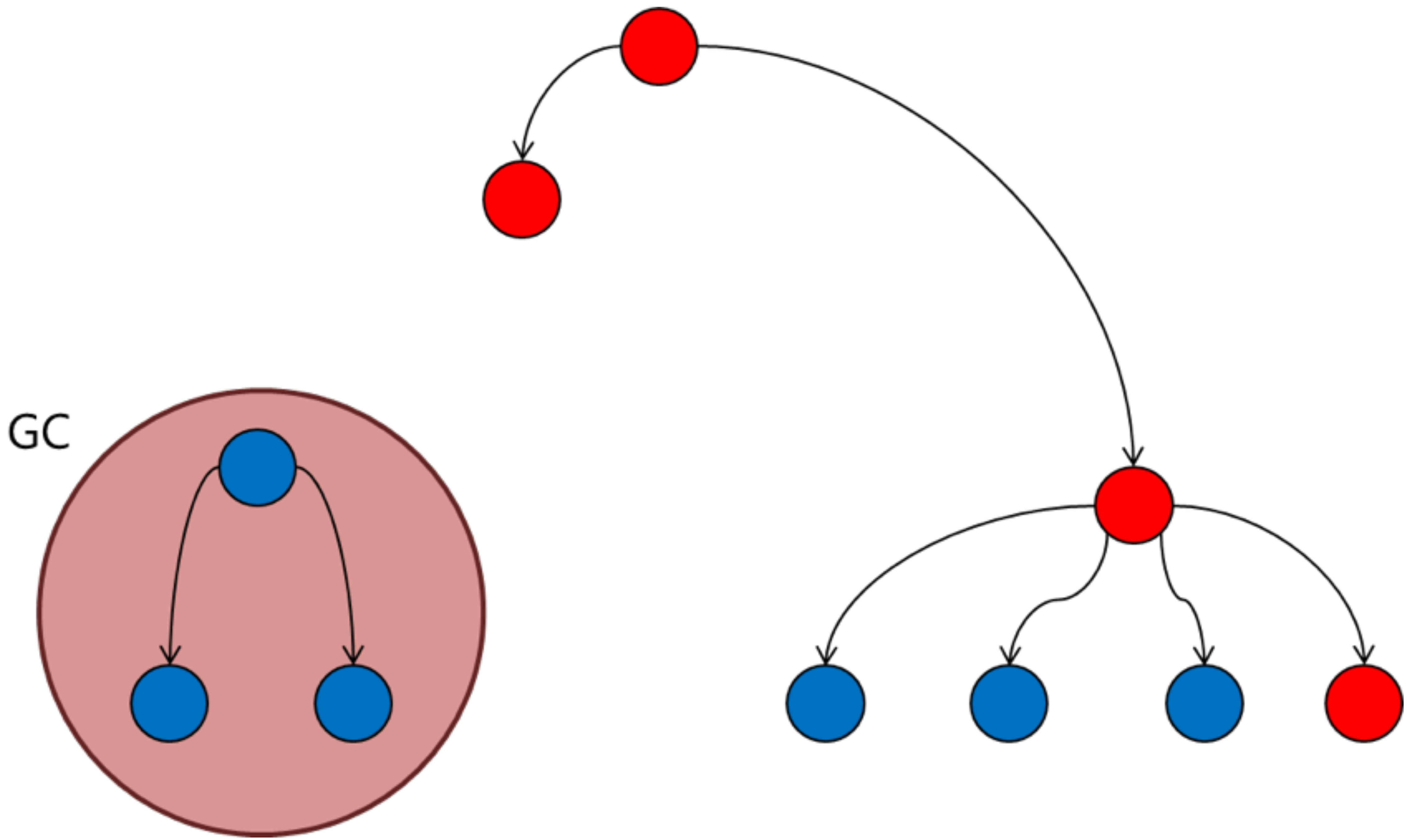
Hound: as a solution

- No false positive
- Data Sampling
  - Context-sensitive memory allocation
  - Age-segregated memory allocation
- Virtual compaction

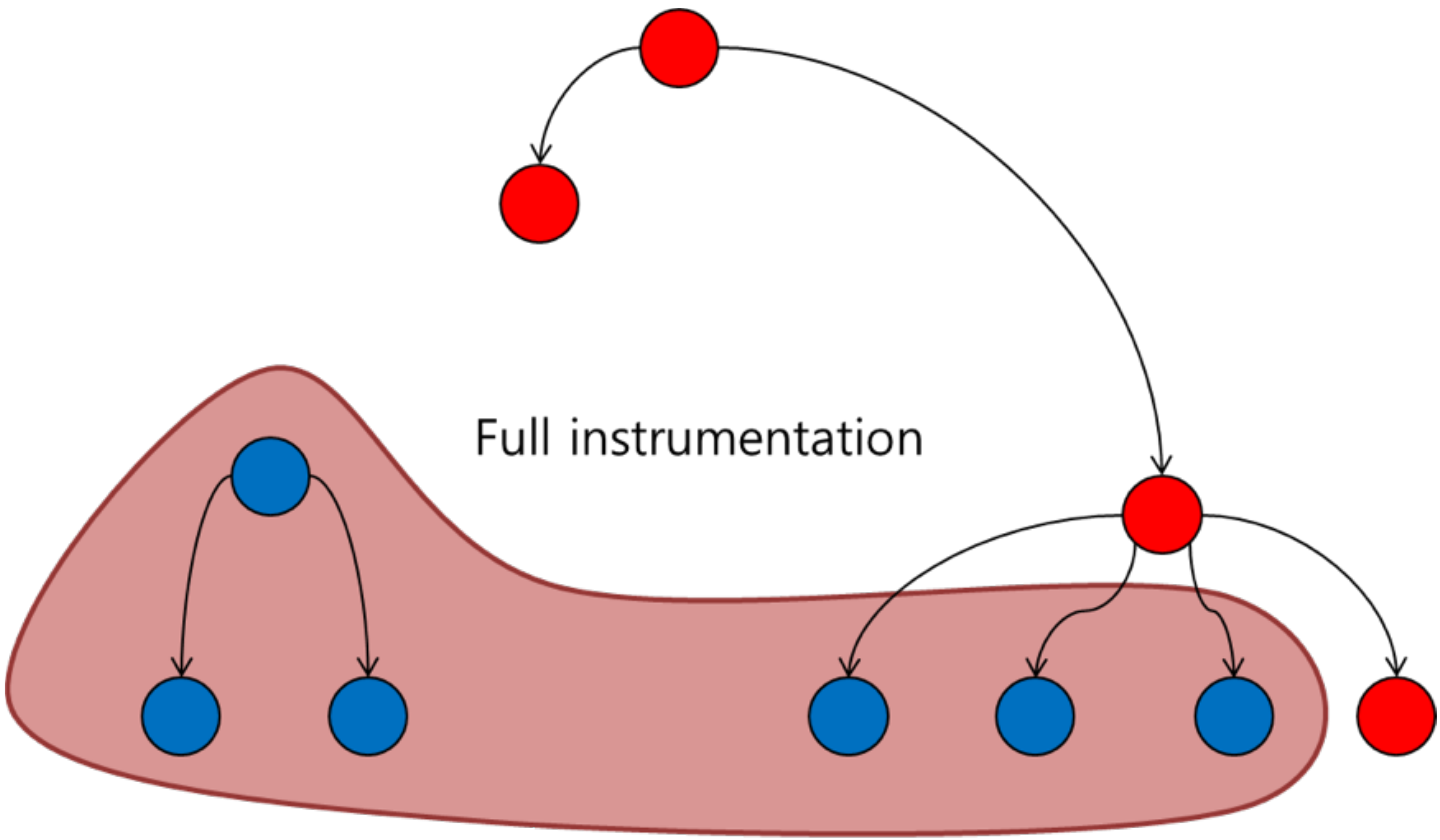
Why Data Sampling?



Why Data Sampling? cont'd

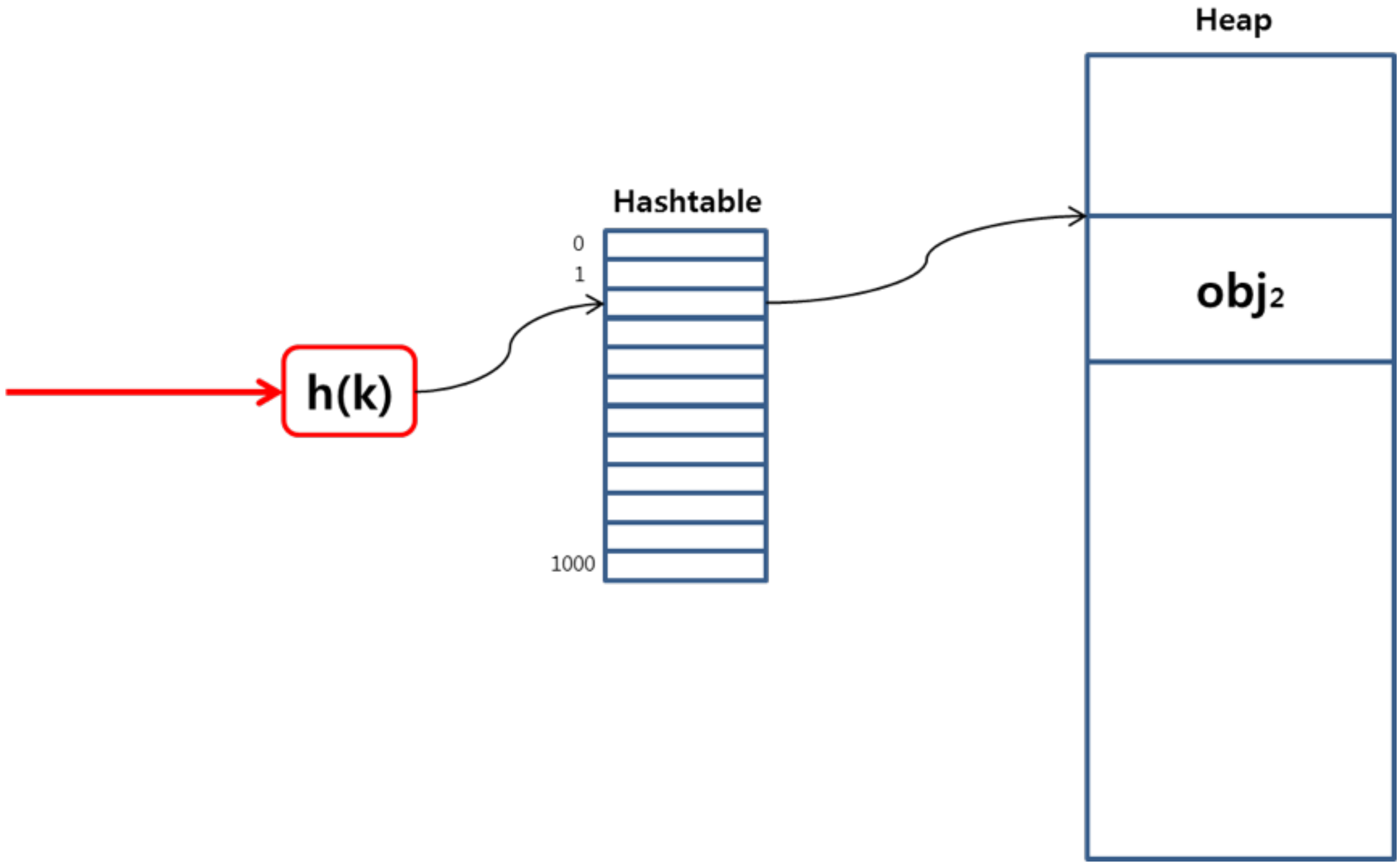


Why Data Sampling? cont'd

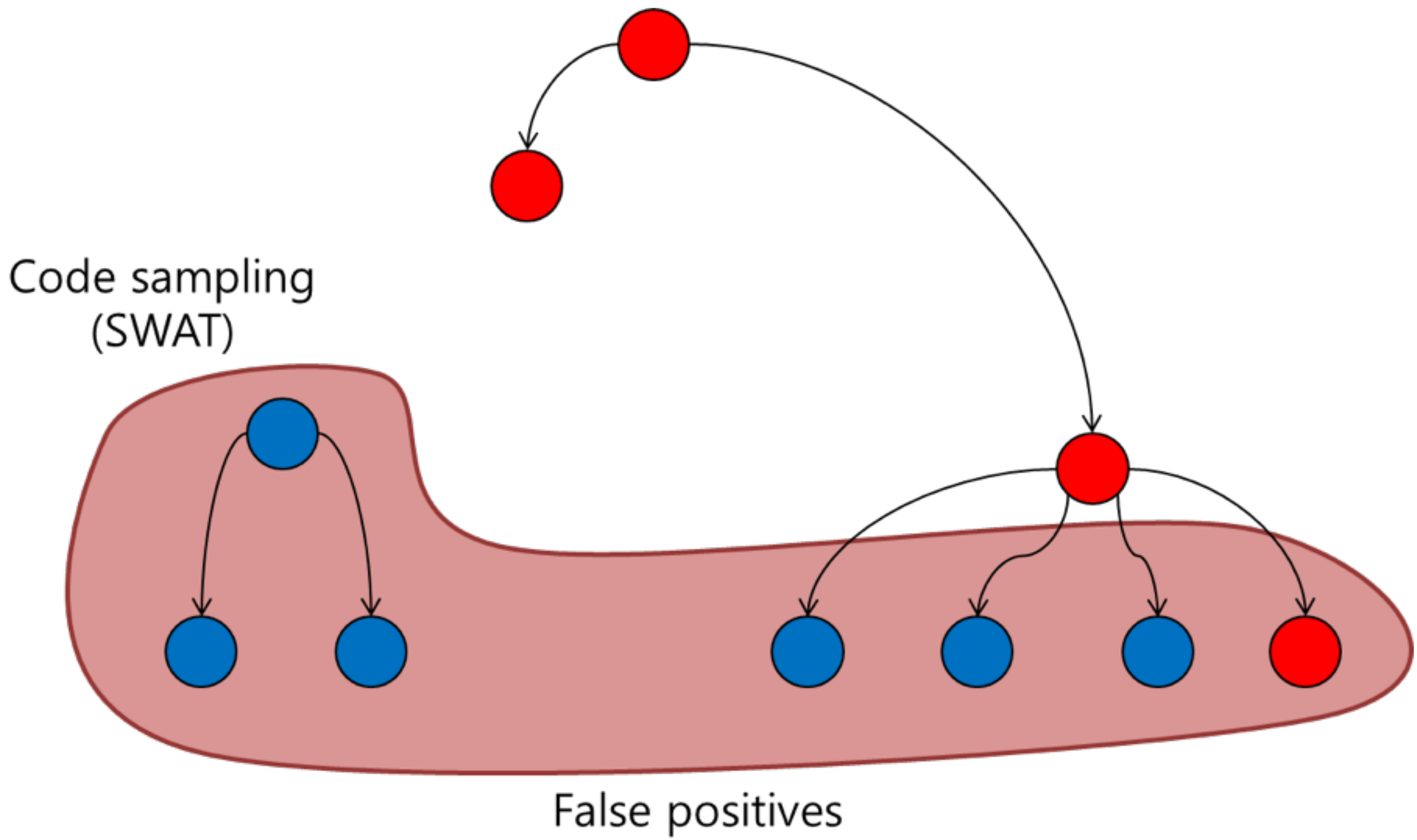


High runtime overhead (100X)

Why Data Sampling? cont'd

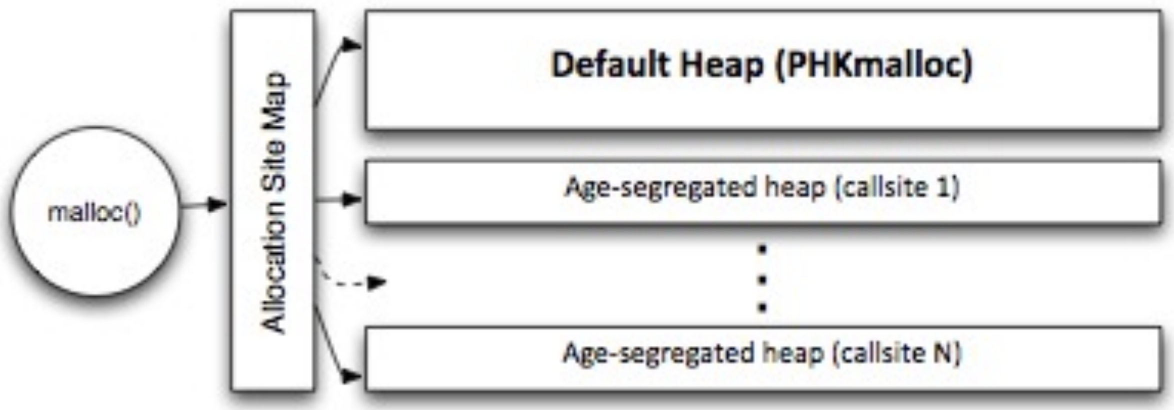


Why Data Sampling? cont'd



How to Data Sampling?

- A novel memory manager
- Segregate along 2-D
  - allocation sites
  - age



Site Segregation

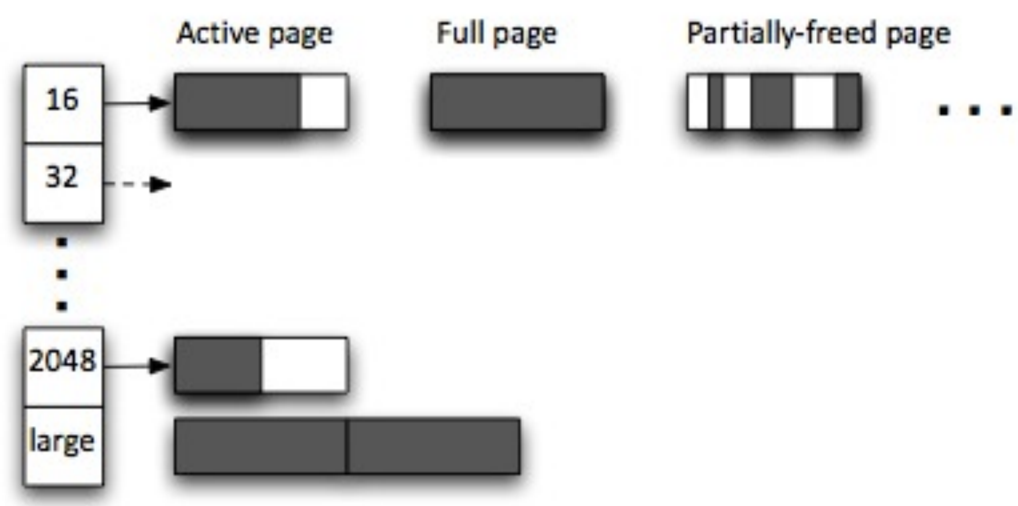
```
void * houndmalloc (size_t size) {
// compute hash of calling context.

int context = getContextHash();

Metadata * m = getMetadata(context);

// one more object allocated. m->liveCount++;

// use the age-segregated heap to
// satisfy the request, if possible.

if (m->getAgeHeap() != NULL) {

return m->getAgeHeap()->malloc (size); }

else if (m->getLiveCount() >= 64) {

    // make a new heap.
    m->initAgeHeap();
return m->getAgeHeap()->malloc (size); }

else {

// still below threshold:
// get memory from standard allocator.

return phkmalloc_with_header (size, context);

} }
```

To reduce memory overhead

- ♀ Add an extra header word
- ♀ Initiate a new age-heap for a site only exceeding a threshold, currently 64
- ♀ Otherwise, allocate objects to a conventional heap

Age Segregation

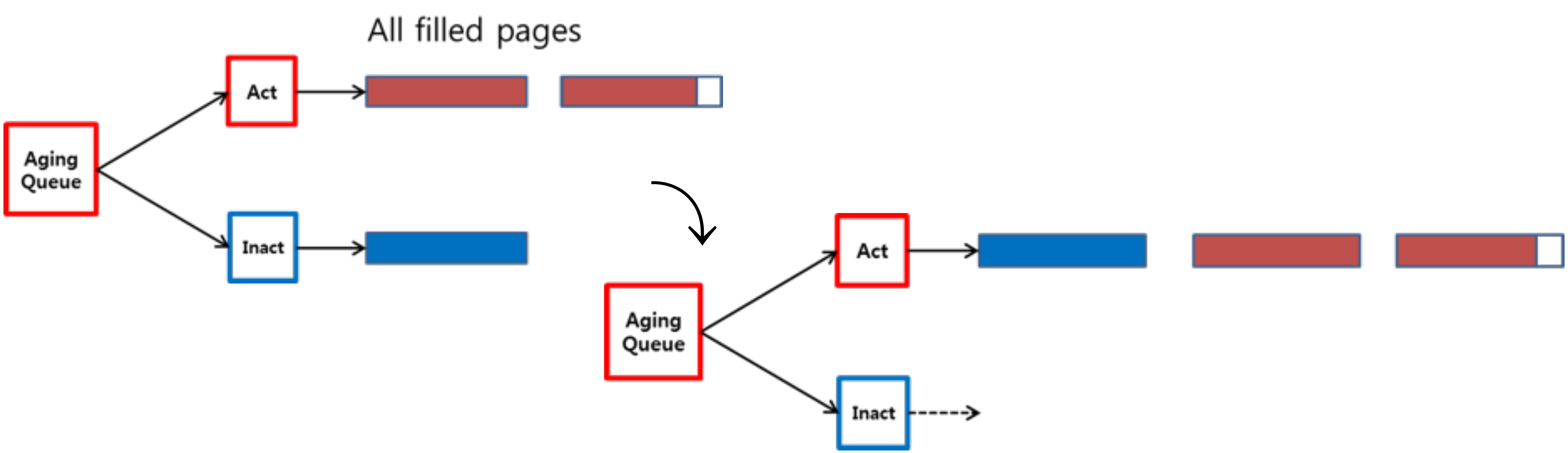
```
if (!h->activePage || h->activePage->bump ==  h->activePage->endOfPage) {
void * page = getNewPage();
PageEntry * e = createPageEntry (page);
    e->bump = page;
    e->endOfPage = page + PAGE_SIZE;
    e->inUse = 0;
    e->heap = h;
    h->activePage = e;
 }
```



Each heap organizes as a collection of pages.(for each size class)
Each page is an array of fixed-sized object slots.
Meta data for each page(bump pointer, # of live objects, a bitmap tracks slots with live objects)

Age Segregation cont'd

- Keeps all filled pages on aging queue and protects pages on the queue
- Due to cost, cannot protect all pages



Age Segregation cont'd

- The size of inactive list is controlled adaptively
  - Low runtime overhead & Maximize Useful Info
- Re-evaluate size every 1/8 CPU time
  - Page faults > 1.5% of total CPU time → Dec
  - Page faults < 0.5% of total CPU time → Inc
- Low runtime overhead >> useful info

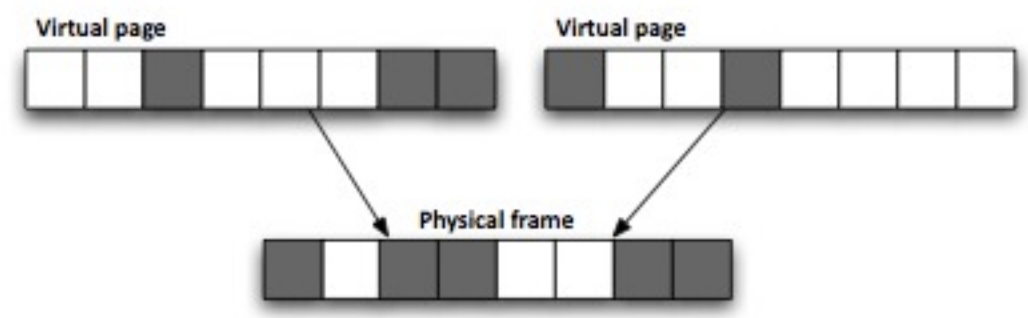
Increase: $P_I = P_I + \max(\min(P_A, P_I)/32, 8)$
Decrease: $P_I = P_I - \max(\min(P_A, P_I)/8, 8)$

Virtual Compaction

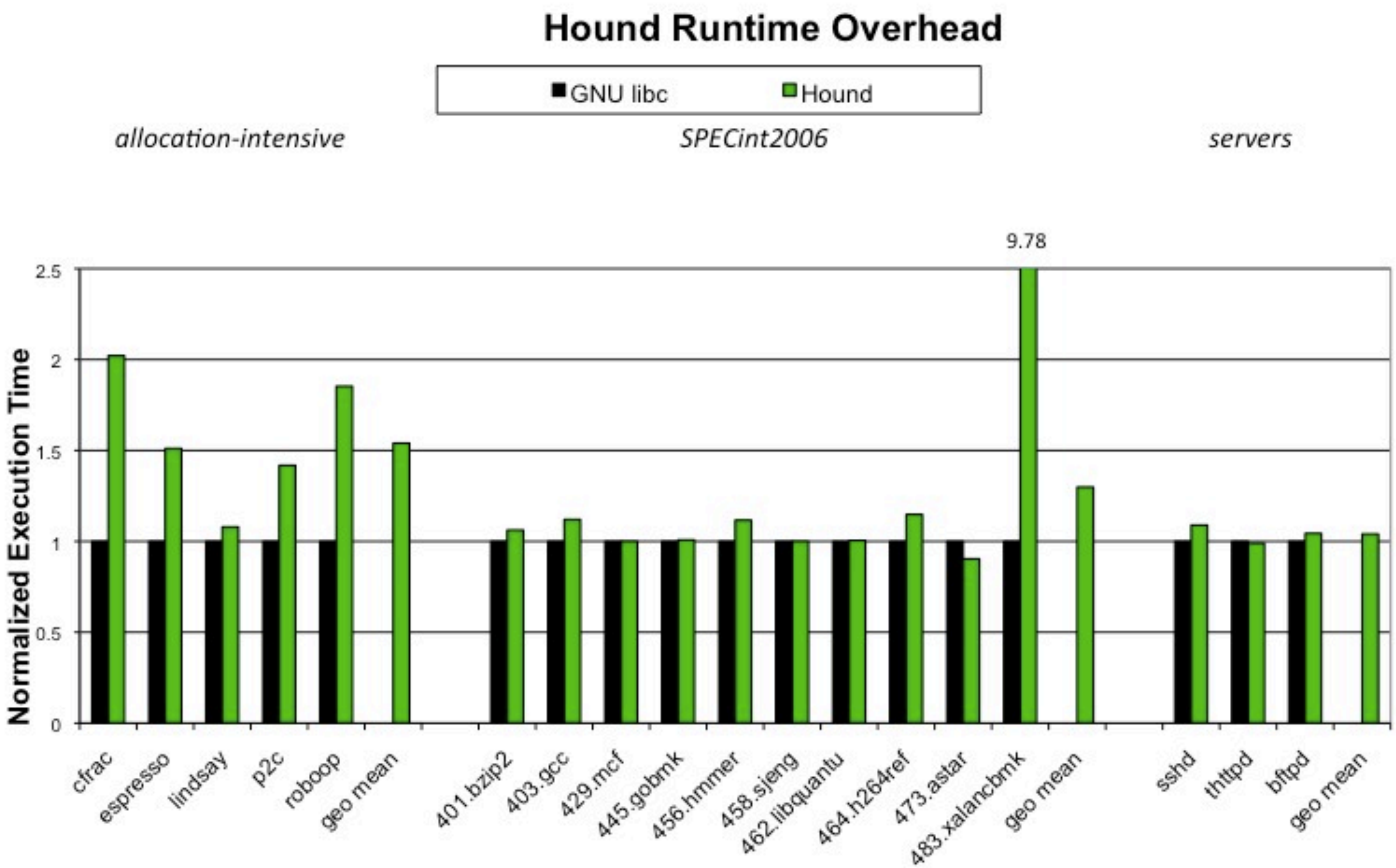
- Why?
  - High fragmentation(potential)
  - Recycles memory from age-segregated heaps only when pages become empty
- To Whom?
  - Toward same sized pages
  - Only for pages have less than 50% occupancy

Virtual Compaction cont'd

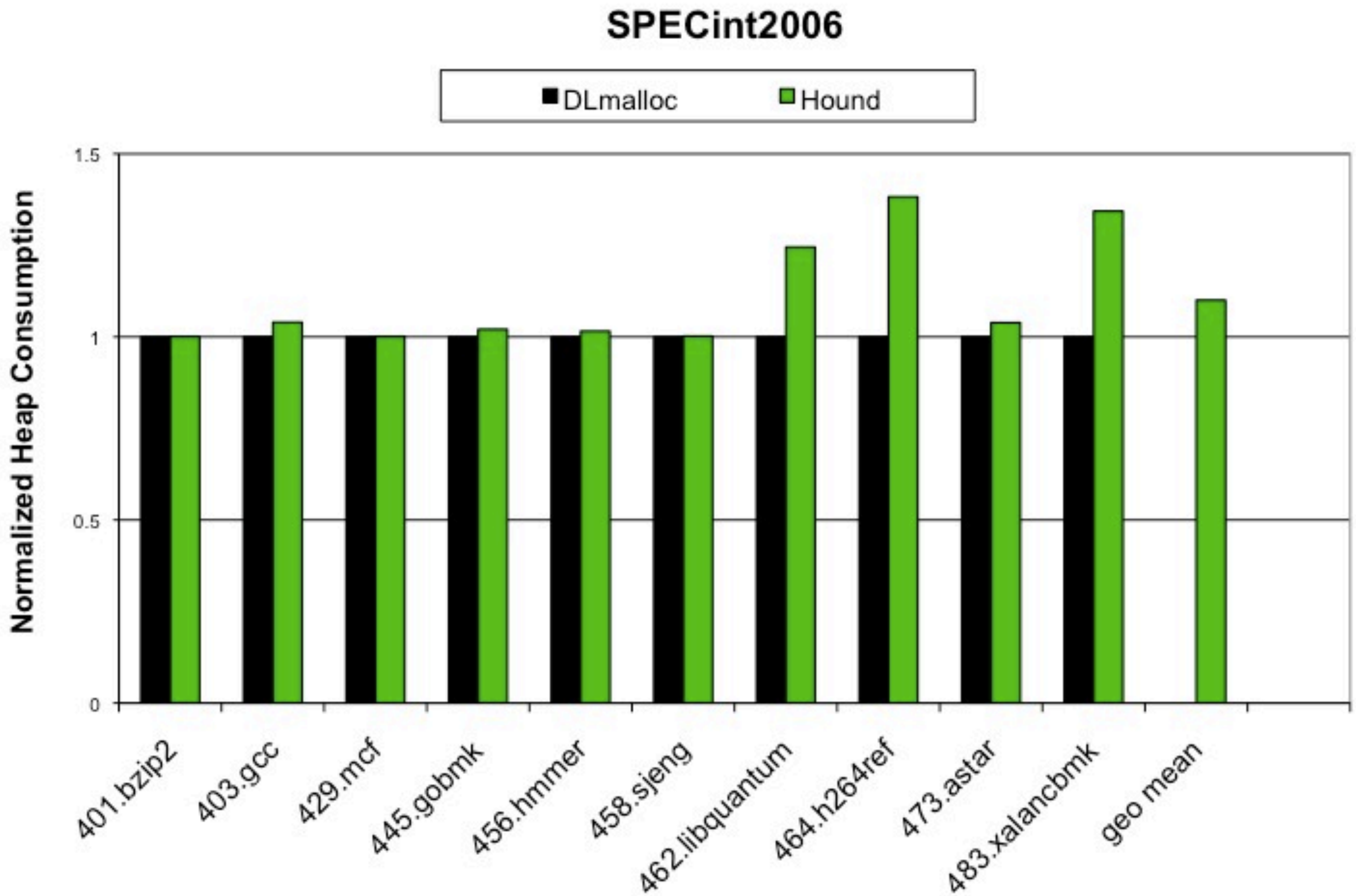
- How?
  - Performing a bitwise AND of several candidate pages' object bitmap
  - Merge them onto a single physical page (mremap call)
  - Remap target(physical) page to both virtual pages



Hound Runtime Overhead



Hound Memory Overhead

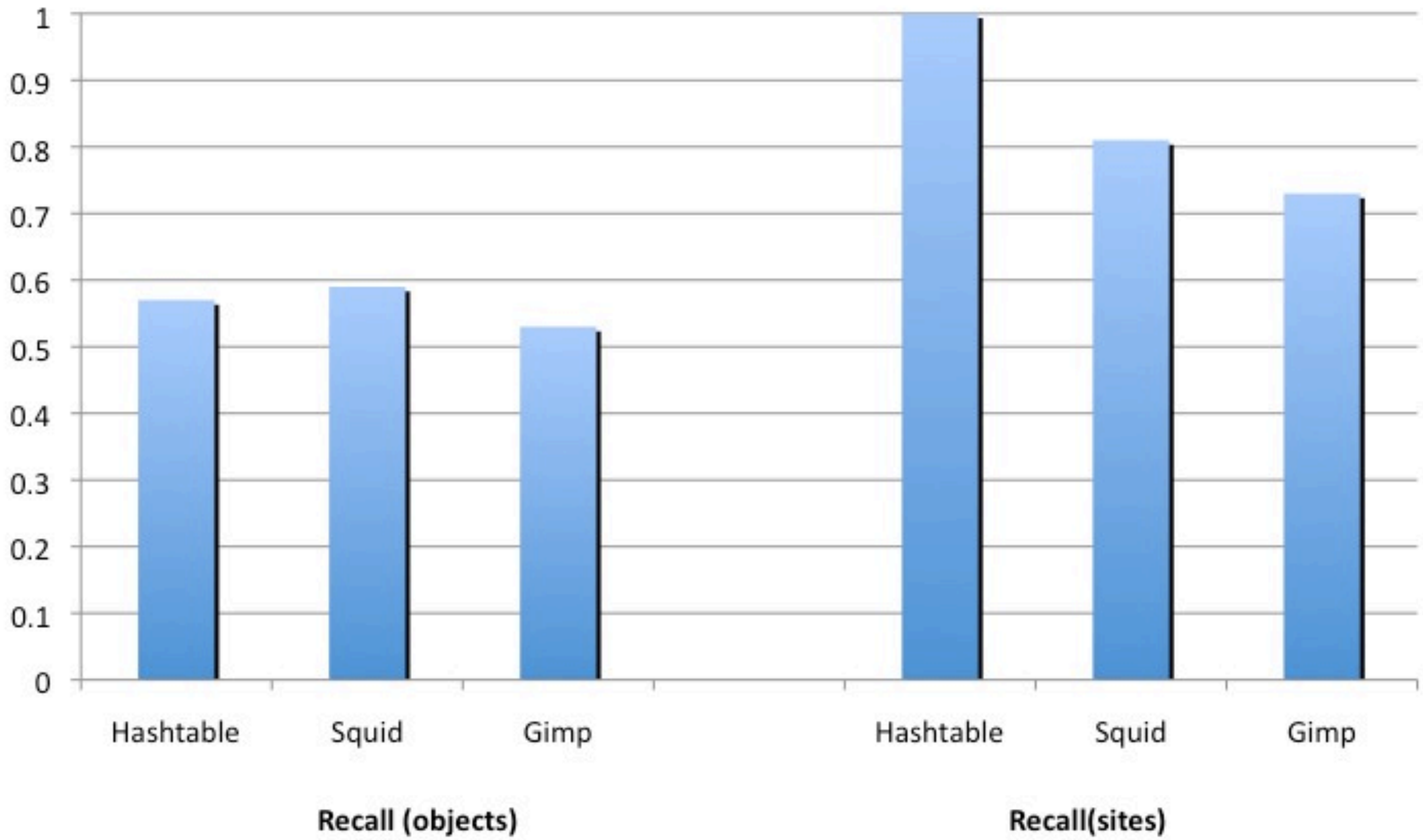


Staleness Computation Accuracy

- Recall(measure the quality of classifier)
  - true positives / (true positives + false negatives)
  - e.g.
    - Consider a report identifies *1* allocation site as the source stale data
    - If this report failed to identify *9* other sites that had stale data
    - recall = *0.1*

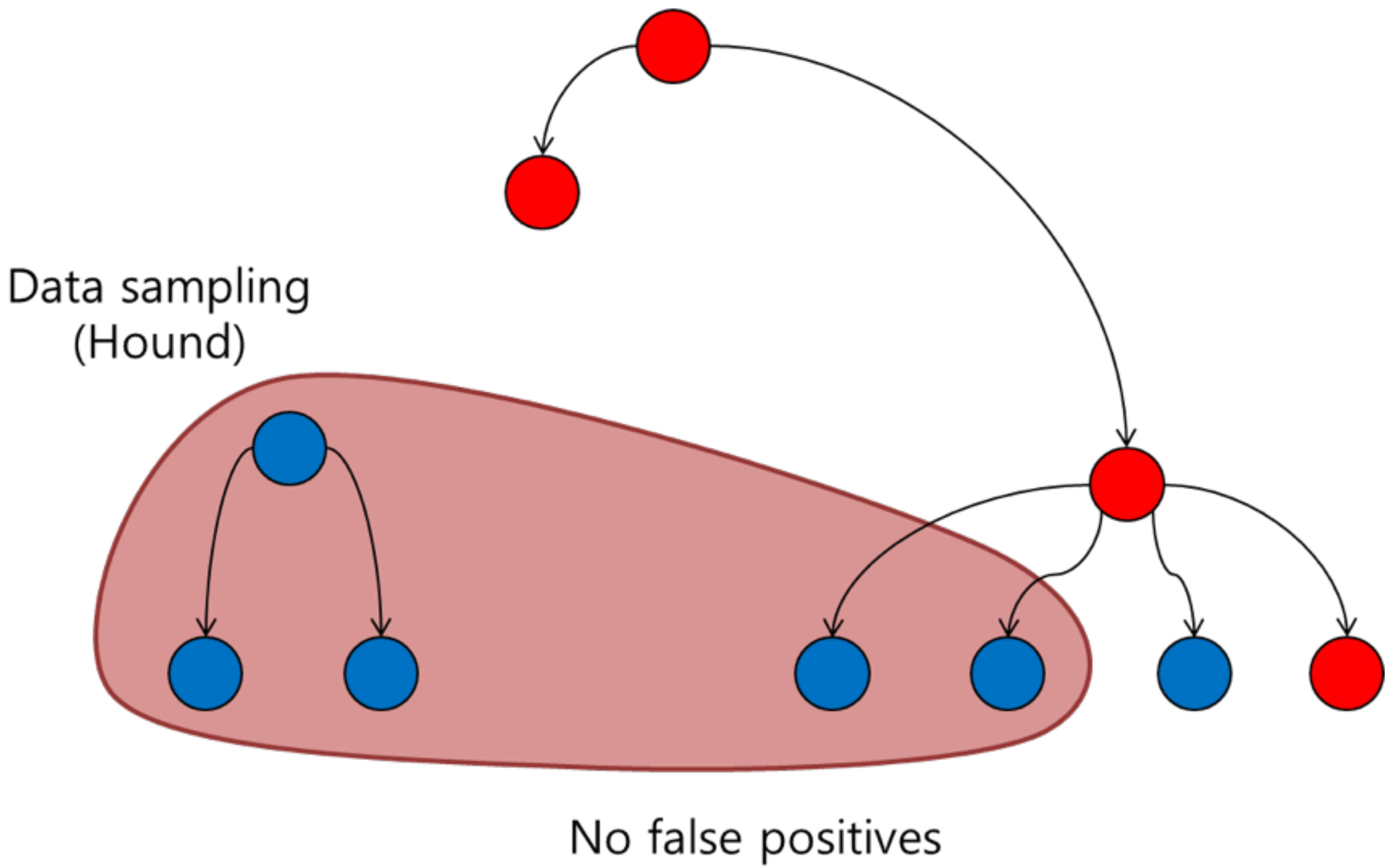
Staleness Computation Accuracy cont'd

Only underestimate!



No false positives; ~30% false negatives

Appropriate result



Questions for Discussion

- Can we appropriately combine two results from Hound and SWAT to make a more decent result?