ScaleCheck

A Single-Machine Approach for Discovering Scalability Bugs in Large Distributed Systems

Cesar A. Stuardo, Tanakorn Leesatapornwongsa*, Riza O. Suminto, Huan Ke, Jeffrey F. Lukman, Wei-Chiu Chuang+, Shan Lu and Haryadi S. Gunawi



THE UNIVERSITY OF CHICAGO

cloudera +

SAMSUNG RESEARCH AMERICA *

New Class of Bugs?

- "Classic" critical bugs
  - Concurrency
  - Configuration
  - Performance
  - Security
- Scalability bugs?

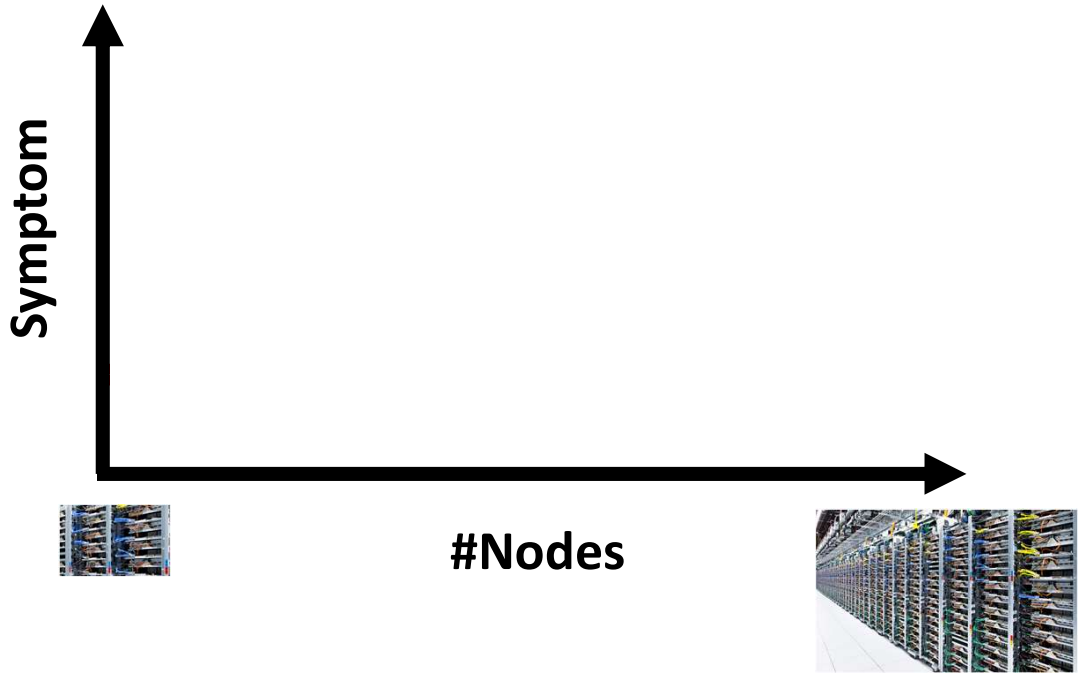


Scalability bugs?

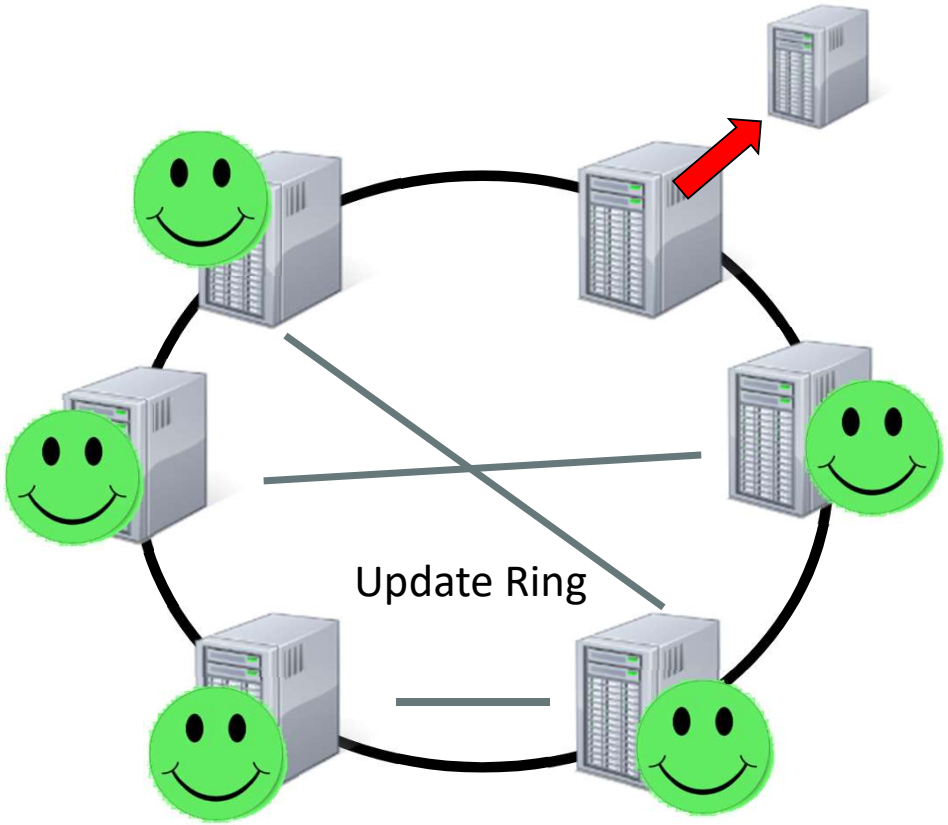
- Cassandra
  - Bug #3831: "start exploding when start reading **100-300 nodes**"
  - Bug #6127: "obvious symptom is with **1000 nodes**"
  - Bug #6409: "With **>500 nodes**, … have trouble"
- Hadoop/HDFS
  - Bug #1073: "**1600 data nodes**. randomWriter fails to run"
  - Bug #3990: "**multi-thousand node** cluster often results in [problem]"
- HBase:
  - Bug #10209: "with **200+ nodes**, [cluster] takes 50mins to start"
  - Bug #12139: "with **>200 nodes**, StochasticLoadBalancer doesn't work"
- Riak:
  - Bug #926: "**1000 node** cluster is a larger question"

Scalability Bugs

*Latent bugs whose symptoms **surface in large-scale deployments**, but not necessarily in small/medium-scale deployments*



An Example: Cassandra Bug #3831



An Example: Cassandra Bug #3831

In every node

```
for(i=0 to N)
  for(j=0 to N)
    for(k=0 to N)
```

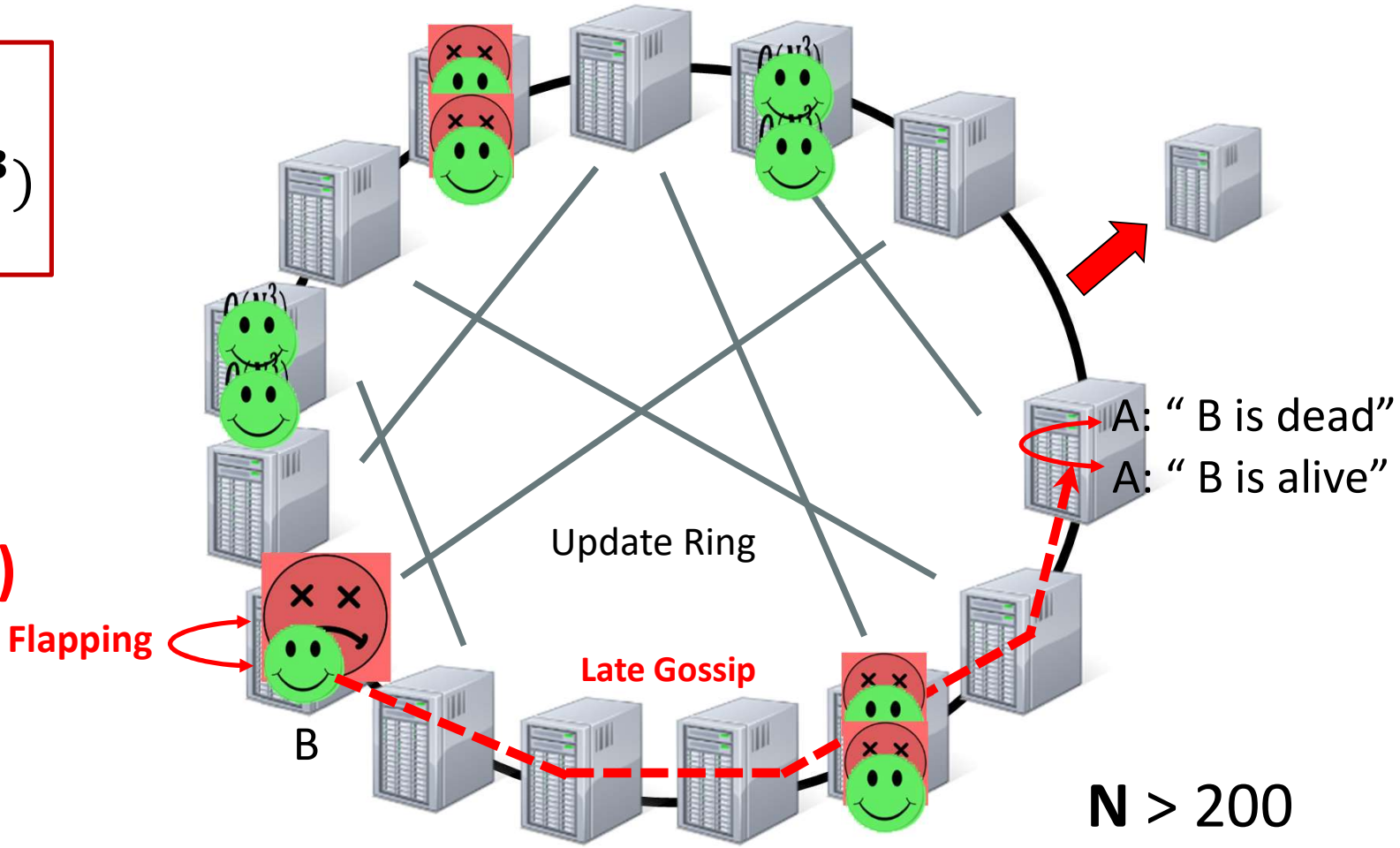
$O(N^3)$

CPU intensive

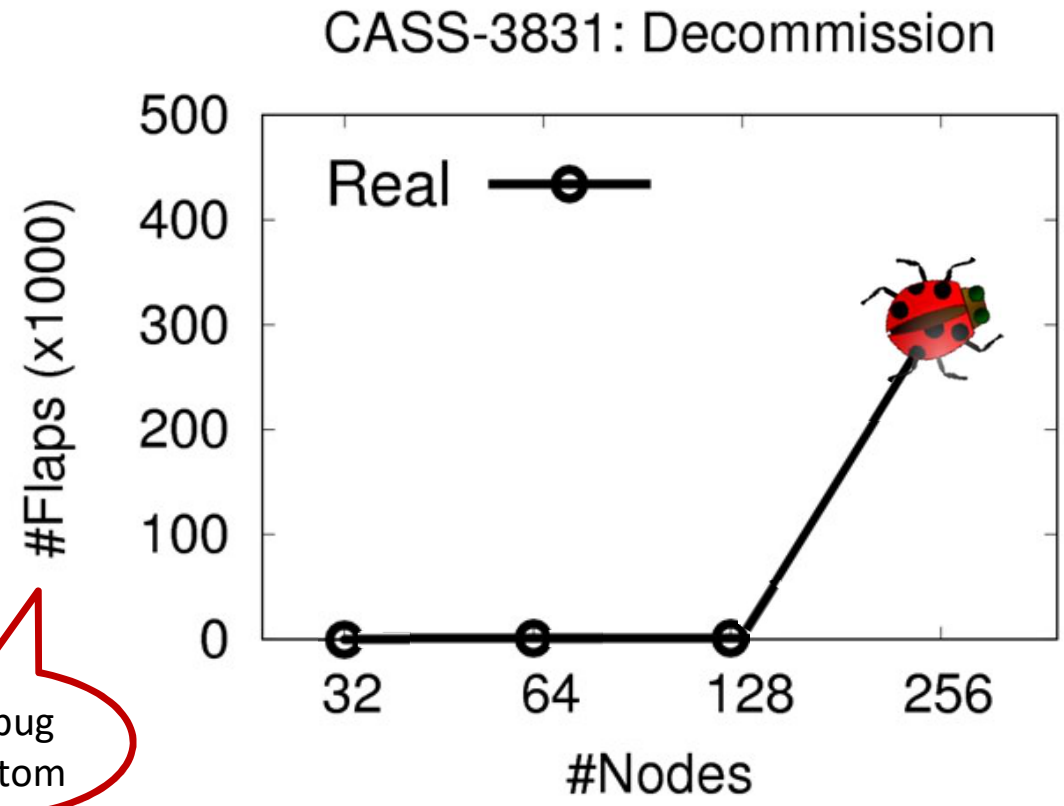
0.1 – 4 seconds

In all nodes!

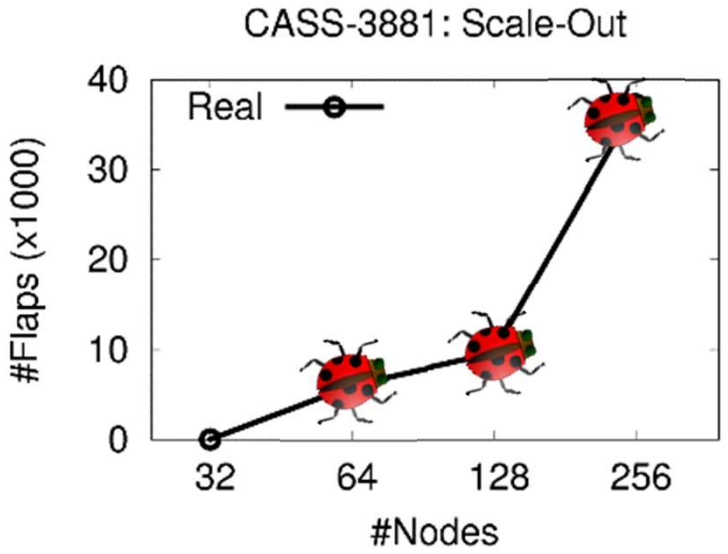
(unstable cluster)

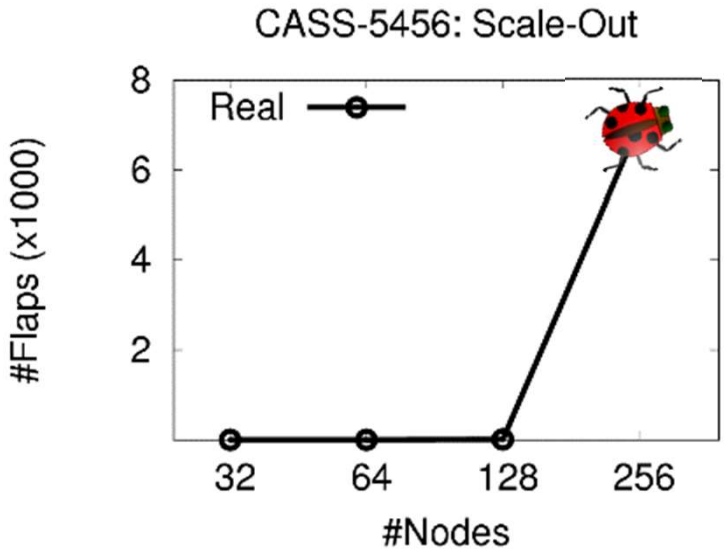


The "Flapping" Bug(s)

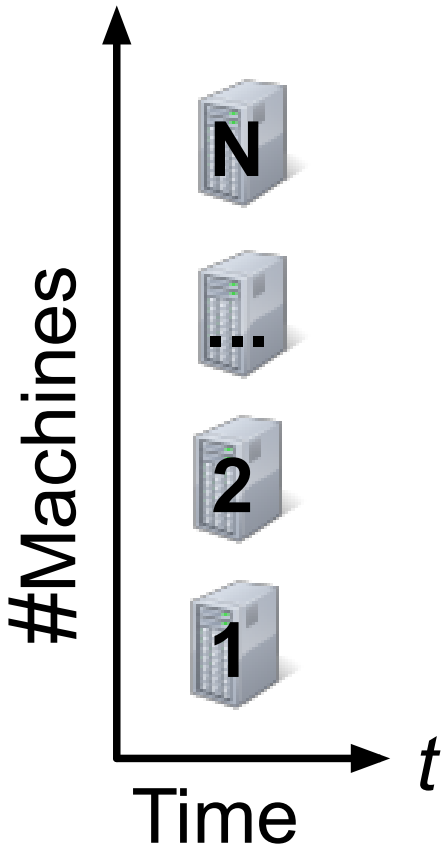


The bug symptom





Real-scale Testing



Q1

What are the **root causes** of scalability bugs?

Q2

Can we **do large-scale testing easily**, e.g. in one machine?

"For Apache Hadoop, **testing at thousand-node scale** has been one of the most effective ways of finding bugs,

but it[illegible]h **difficult** and **expensive**. It takes considerable [illegible] a large-scale cluster, much less

[illegible] thousands of dollars an hour, [illegible] **for the solo contributor**.

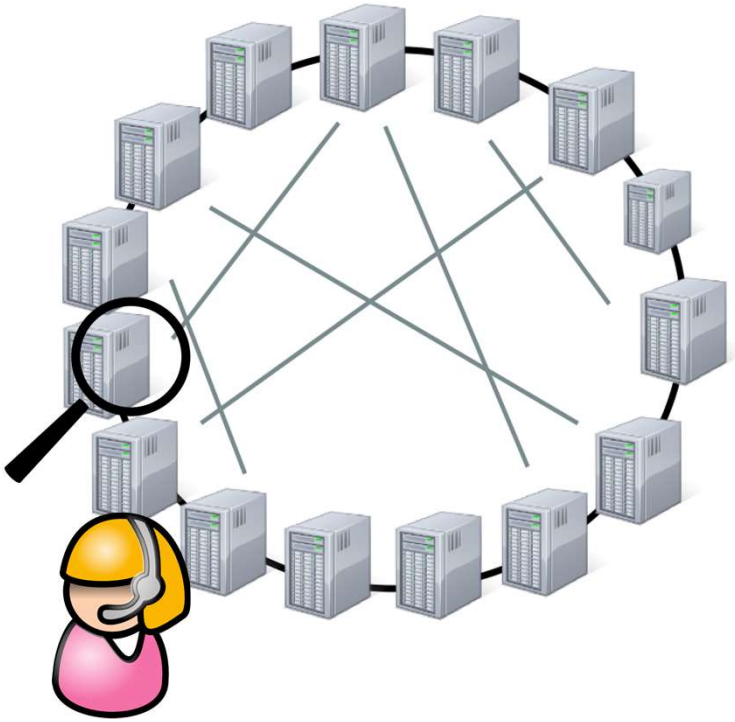
As it [illegible]s, we are heavily reliant on test clusters operated by [illegible]ing.

[illegible]s without requiring running a [illegible]remely useful"

— Andrew Wang (Cloudera and Apache Hadoop PMC Member and Committer)

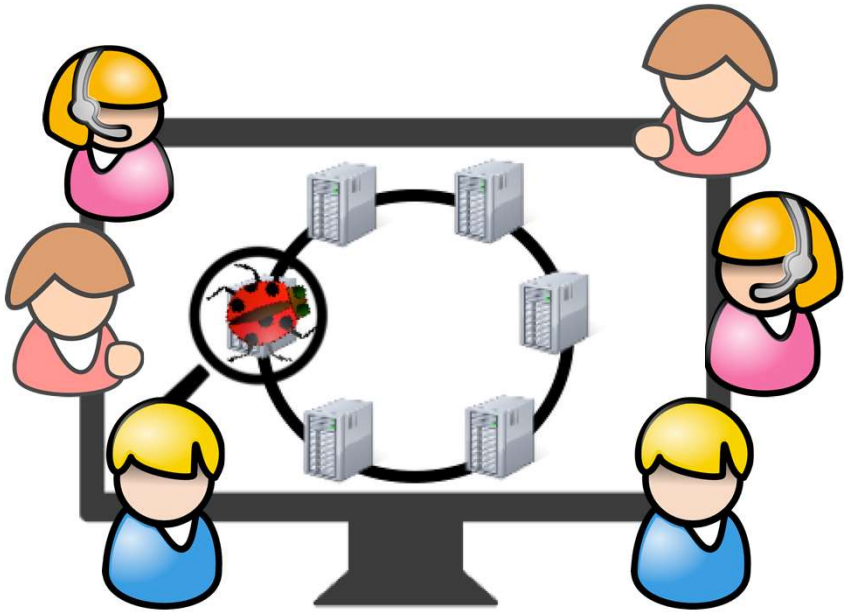
ScaleCheck

Real-scale testing



Discovering scalability bugs

Democratizing large scale testing



Q1

What are the **root causes** of scalability bugs?

Scale Dependent Loops

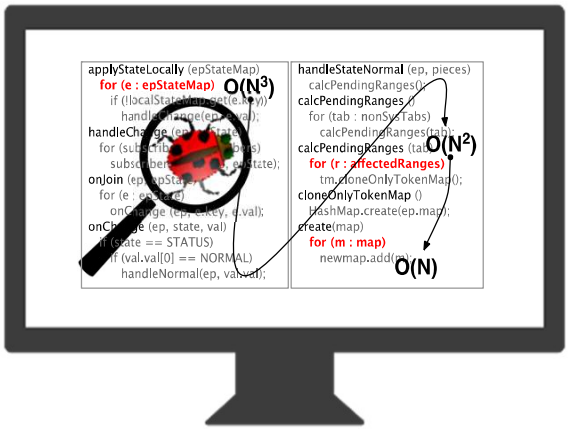
```
applyStateLocally (epStateMap)
  for (e : epStateMap)                    O(N^3)
    if (!localStateMap.get(e.key))
      handleChange(ep, e.val);
handleChange (ep, epState)
  for (subscriber : subscribers)
    subscriber.onJoin(ep, epState);
onJoin (ep, epState)
  for (e : epState)
    onChange (ep, e.key, e.val);
onChange (ep, state, val)
  if (state == STATUS)
    if (val.val[0] == NORMAL)
      handleNormal(ep, val.val);
```

```
handleStateNormal (ep, pieces)
  calcPendingRanges();
calcPendingRanges ()
  for (tab : nonSysTabs)                  O(N^2)
    calcPendingRanges(tab);
calcPendingRanges (tab)
  for (r : affectedRanges)
    tm.cloneOnlyTokenMap();
cloneOnlyTokenMap ()
  HashMap.create(ep.map);
create(map)
  for (m : map)
    newmap.add(m);                        O(N)
```

Loops iterating
scale dependent data structures

ScaleCheck

SFind



Automatically find *scale dependent data structures and loops*
by checking their ***in-memory growth tendency*** *over small scale experiments*

Q2

Can we **do large-scale testing easily**, e.g. in one machine?

Many Challenges!!!

Memory Overhead

Context switching

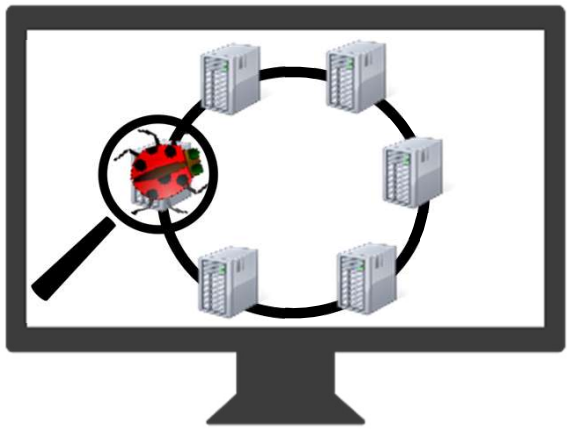
CPU contention

Network Overhead

Bad modularity

ScaleCheck

STest



SPC **S**ingle **P**rocess **C**luster

GEDA **G**lobal **E**vent-**D**riven **A**rchitecture

PIL **P**rocessing **Il**lusion

Large scale testing in a single machine *with little source code modification*

Our Results









- Integrated to **4** systems
- Scale tested **18** protocols
- Accurately reproduced **10** known scalability bugs
- Found **4** new bugs

- …..in just a single machine

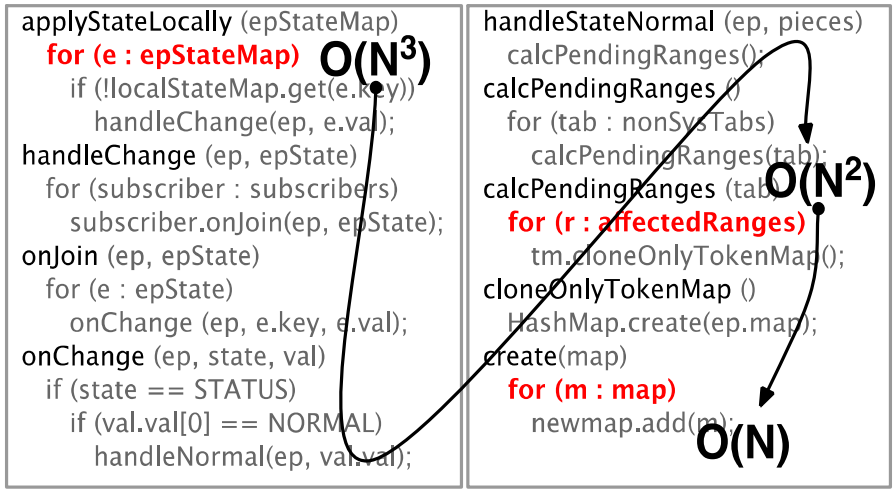
Outline

- Introduction
- SFind
- STest
- Evaluations and Conclusions

Our Frequent Root Cause…

Scale Dependent Loops

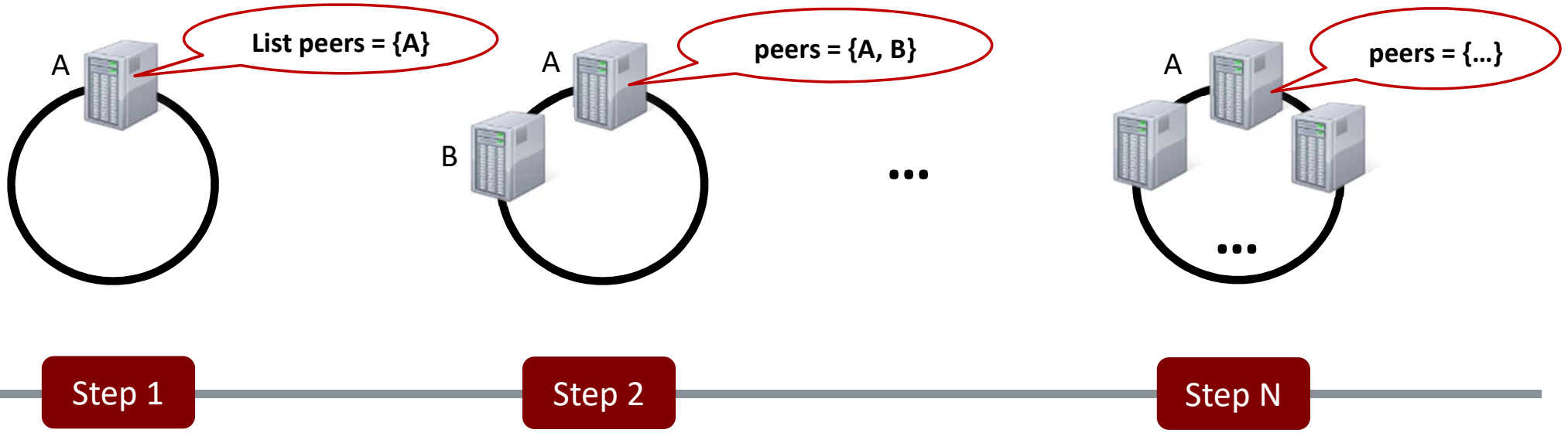
iterating scale dependent data structures



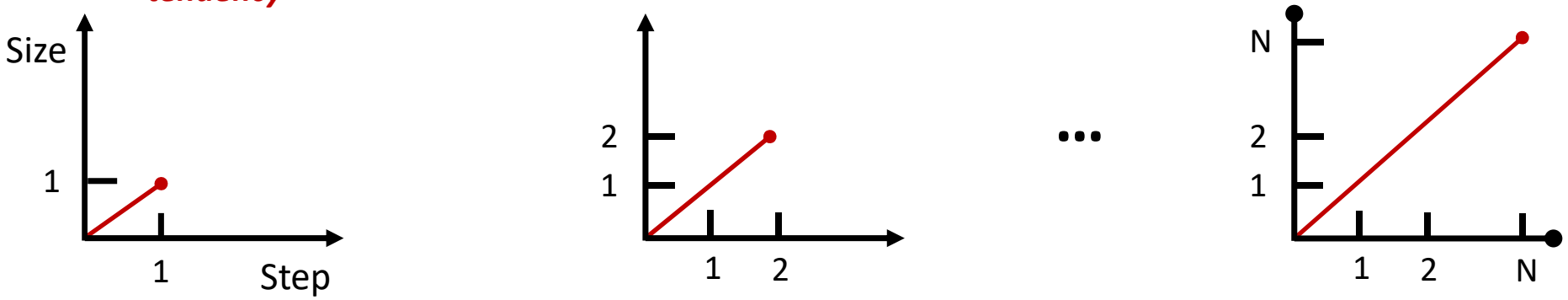
is not easy to find in…

- Any distributed system these days has…
  - > 100K LOC
  - > 100x source files
  - > 1000x maps, lists, queues, etc…
- We can find them manually…
  - */* … holds one entry per peer … */*
  - */* … maps partition to node… */*
  - */* … maps blocks to datanodes… */*
- How can we **find them automatically?**

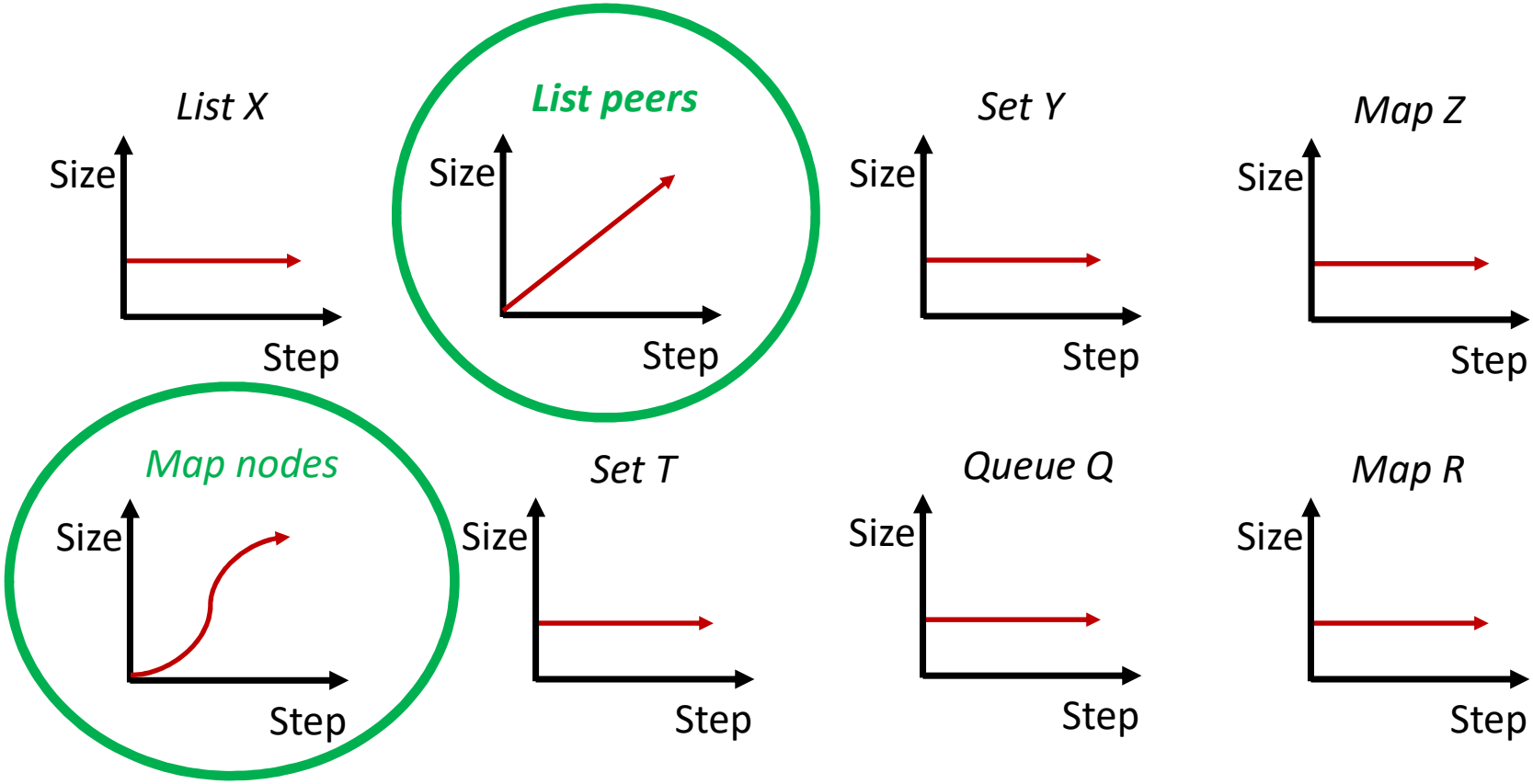
JVMTI and Reflection support



List "peers" ***growth tendency***



*Data structures that tend to grow when one or more dimensions grow are **scale dependent***

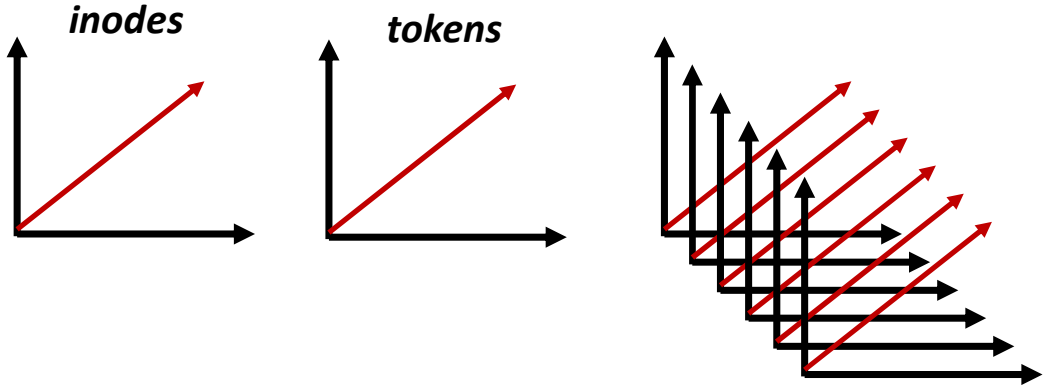


A's on heap data structures **growth tendency after N steps**

Finding **many** scale-dep data structures

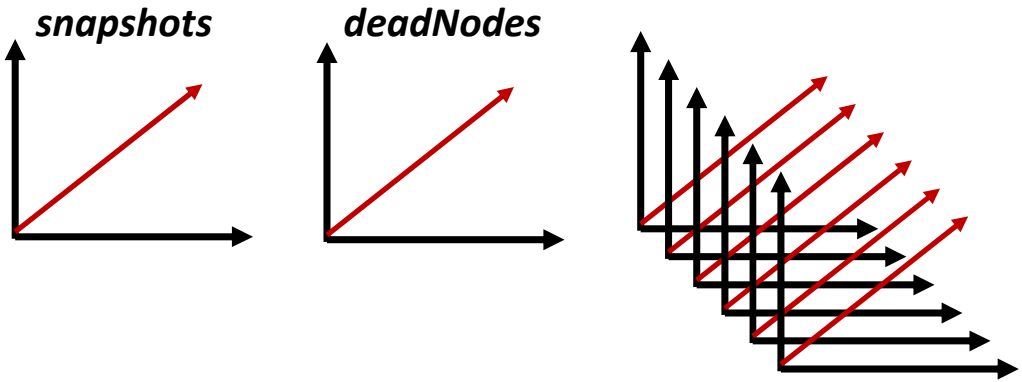
- Try out different **axes of scale**
  - Adding files
  - Adding partitions
  - tables, columns, files, directories…

26 data structures (Cassandra, HDFS)



- Try out different workloads
  - Snapshot directories
  - Removing nodes
  - migration, rebalance…

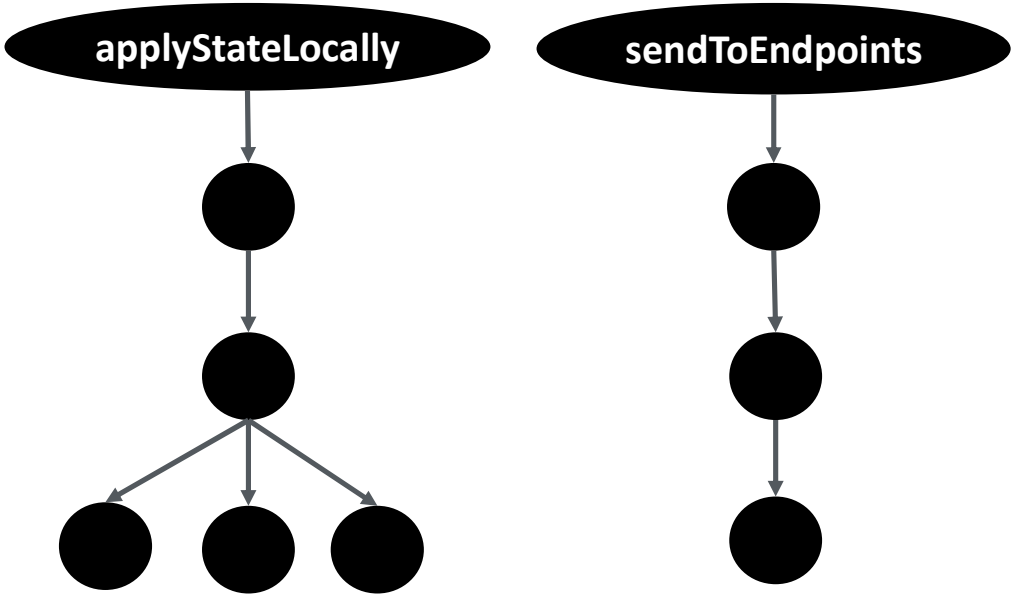
18 data structures (Cassandra, HDFS)



Prioritizing Scale-Testing

Map our data structures to loops using data-flow

Create **inter-procedural call graphs** for all methods



Priority	Method	Complexity	#IO	#Lock
?	sendToEndpoints	2	0	0
?	applyStateLocally	2	N^2	N^2

Prioritize harmful scalability bottlenecks

Priority	Method
1	applyStateLocally
2	sendToEndpoints

...

Outline

- Introduction
- SFind
- STest
- Evaluations and Conclusions

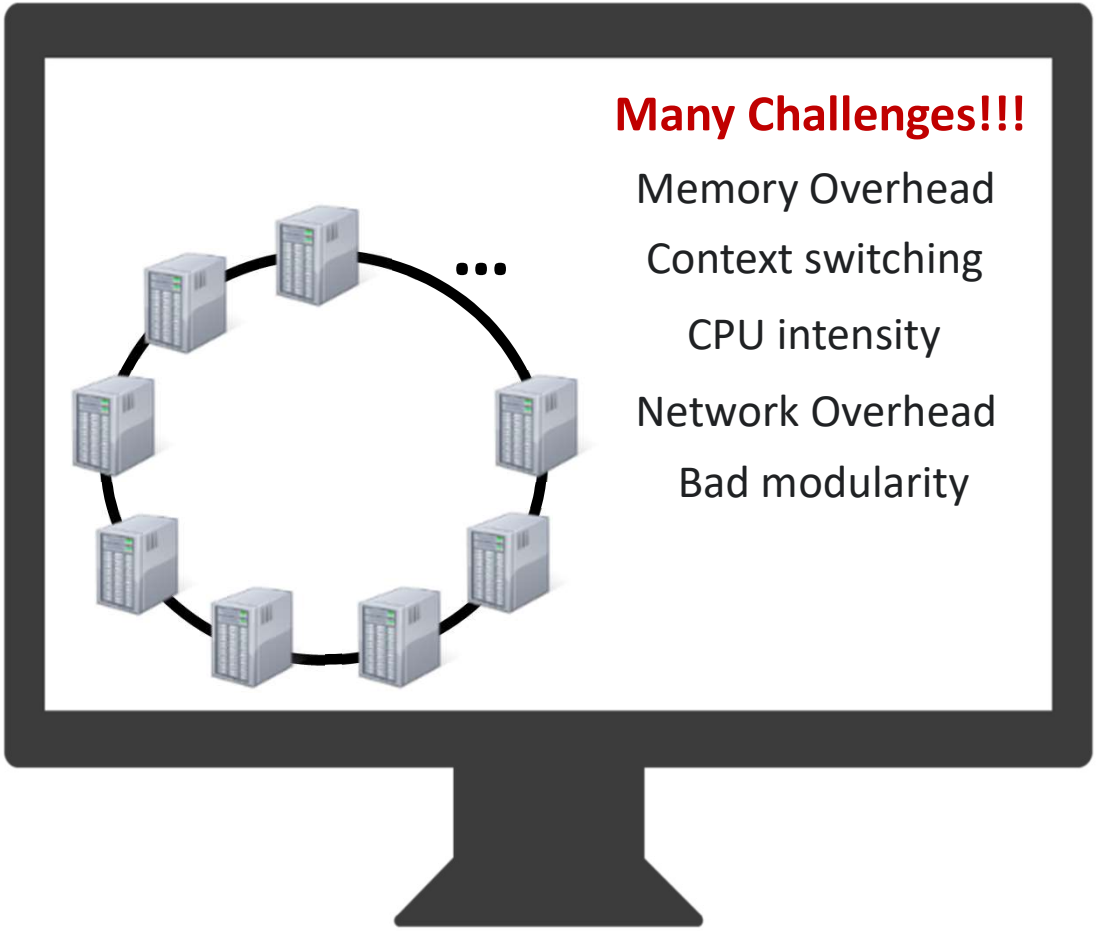
Democratizing Large Scale Testing with STest

Priority	Method
1	applyStateLocally

...

TestApplyStateLocally.java

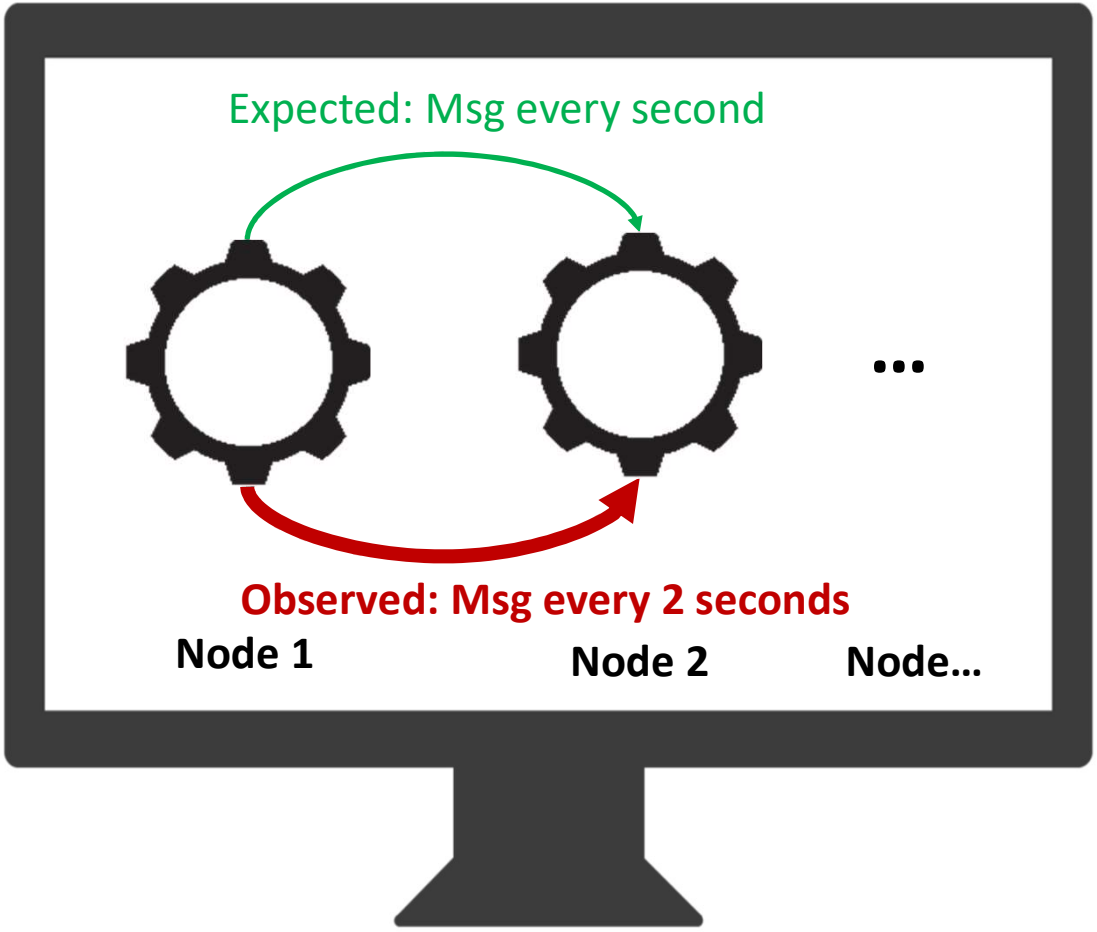
```
public void testApplyStateLocally(…){
    //…
}
```



Try #1: Naïve Packing (NP)

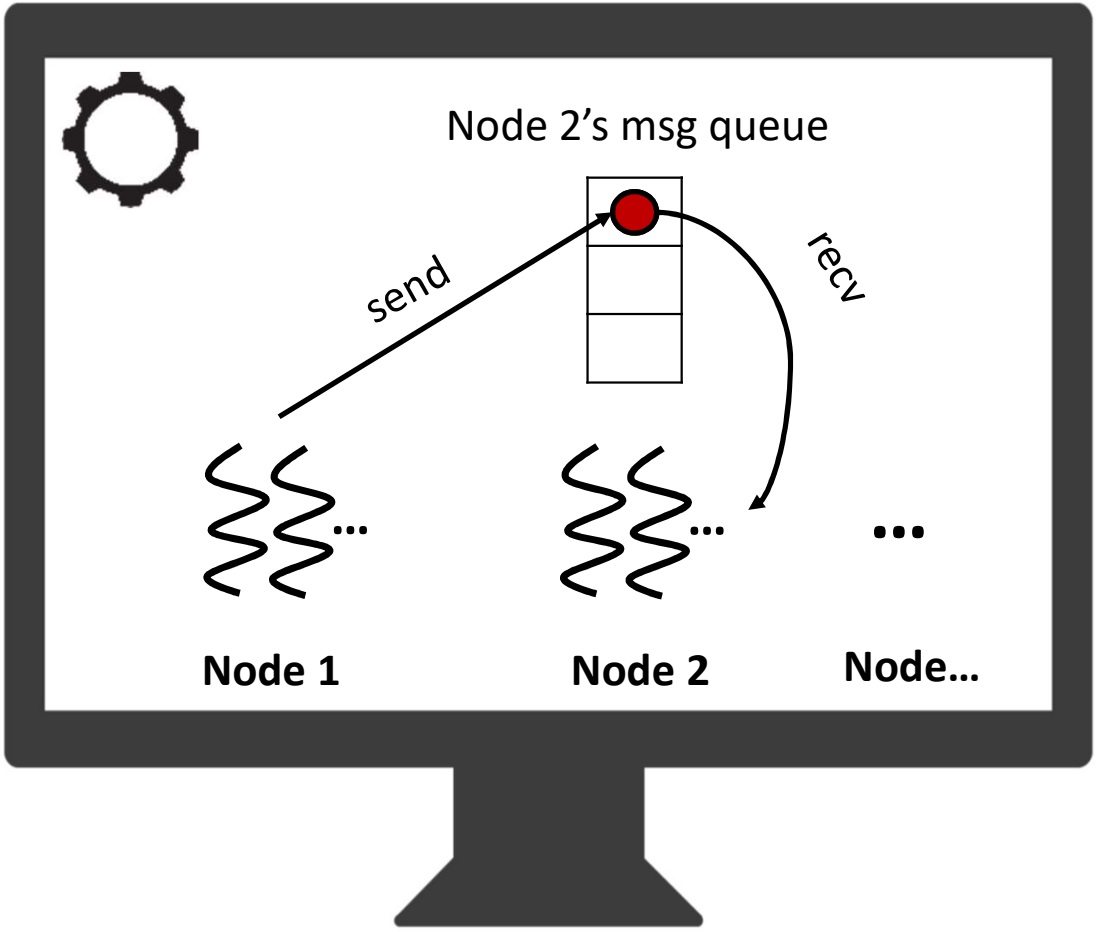
- Deploy **nodes as processes** in a single machine
  - One process = one node = one system's instance
- Main challenges
  - Per node runtime overhead
  - Process context switching
    - **Event Lateness** (e.g send message)
- Max Colocation factor: **50 nodes**

Prohibits large scale-testing!



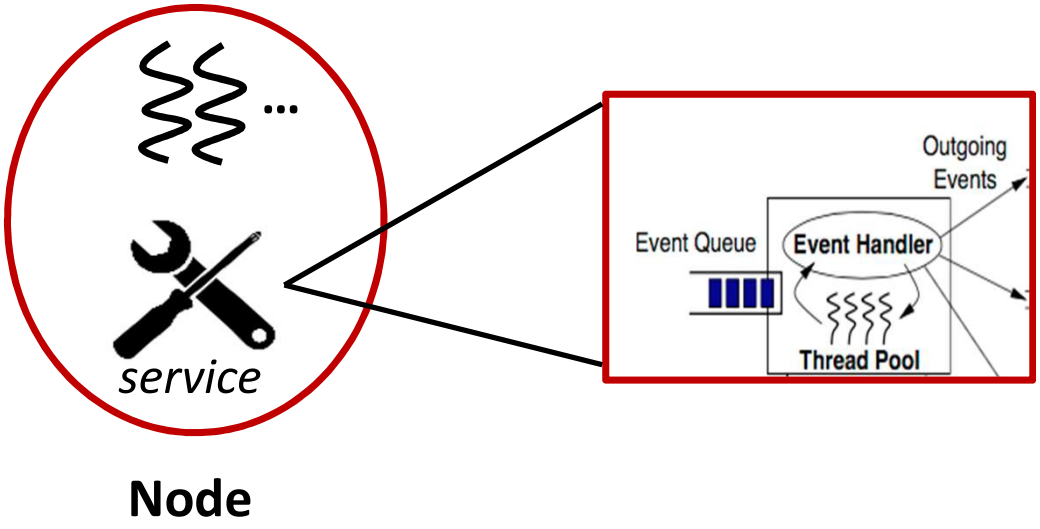
Try #2: Single Process Cluster (SPC)

- Deploy **nodes as** ~~processes~~ **threads** in a single process
- Challenges
  - Lack of modularity (e.g. bad design patterns)
    - Automatic per-node **isolation** (e.g. ClassLoader isolation)
  - Large amount of threads
    - Simplify inter-node communication
- Max Colocation factor: **120 nodes**

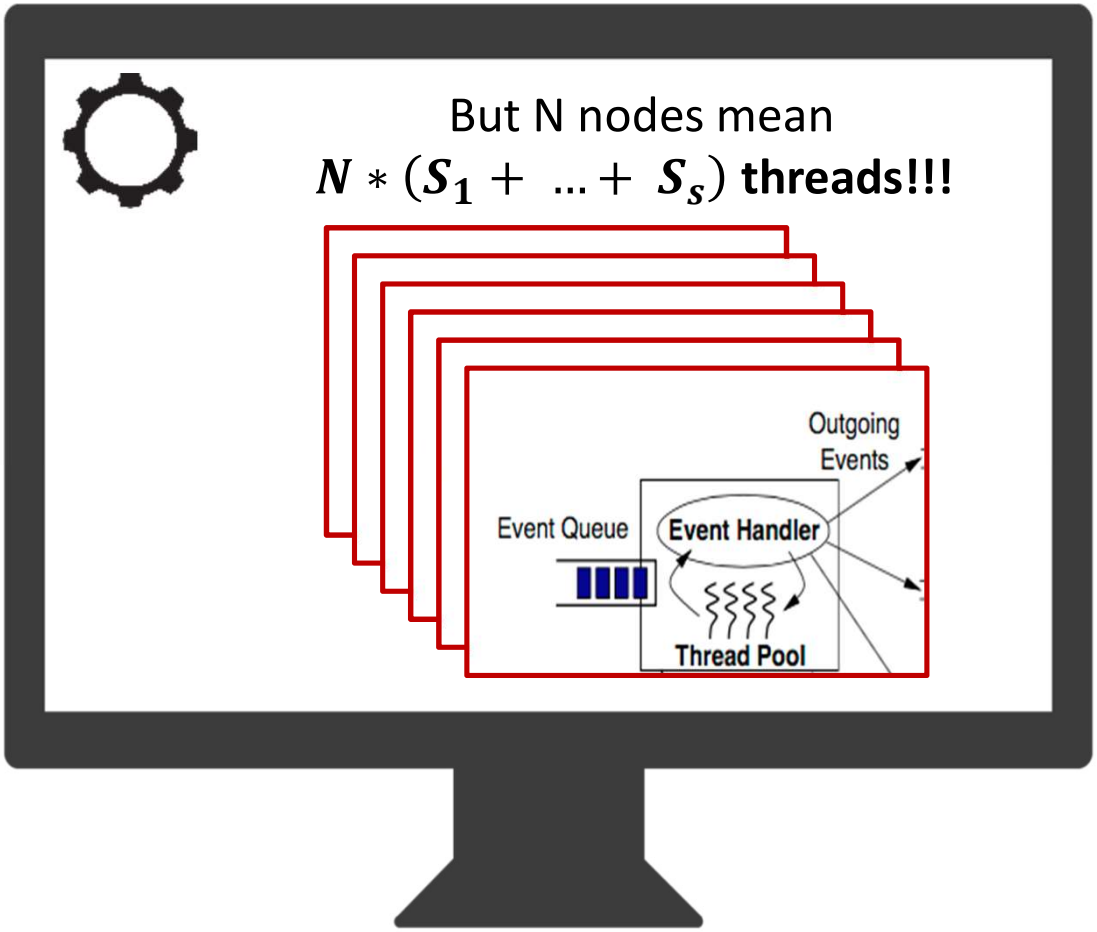


Per-Node Services

- Frequent Design pattern
  - Service = **worker pool + event queue**

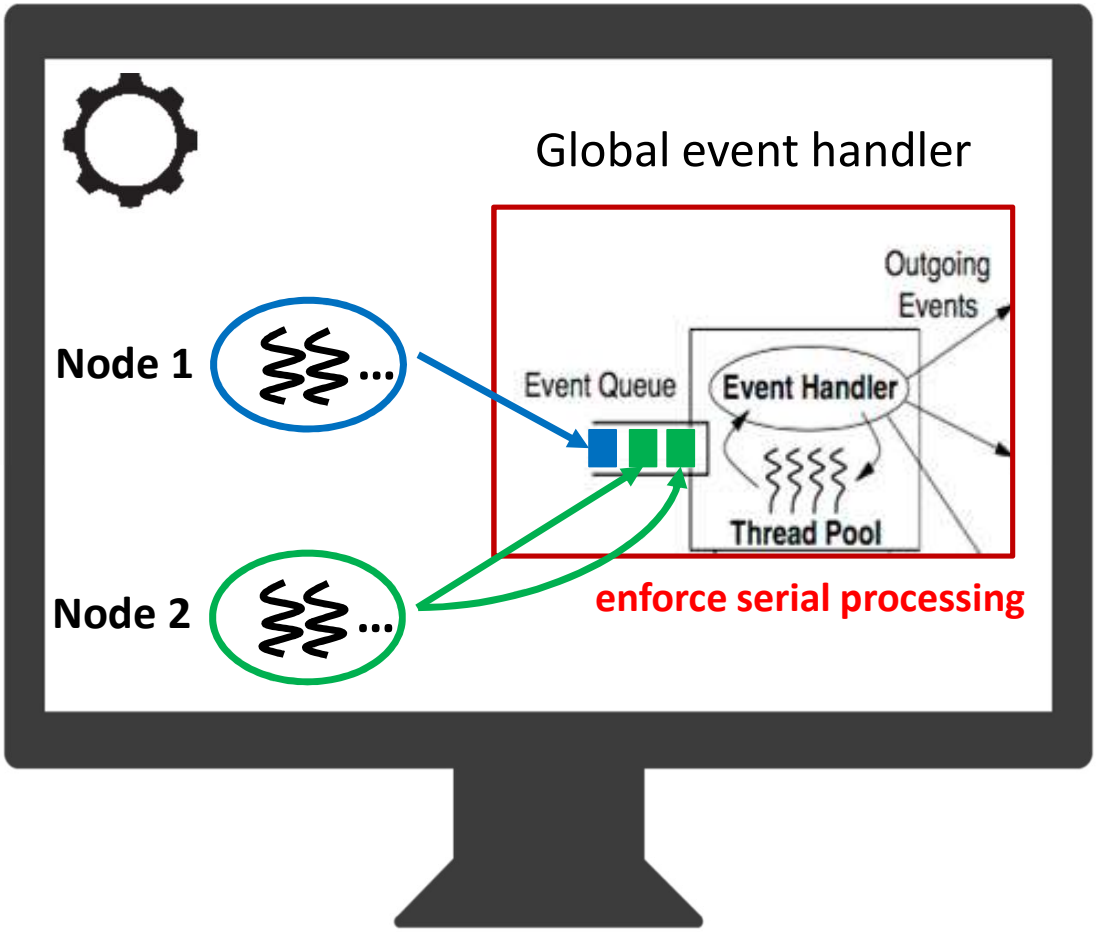


Node



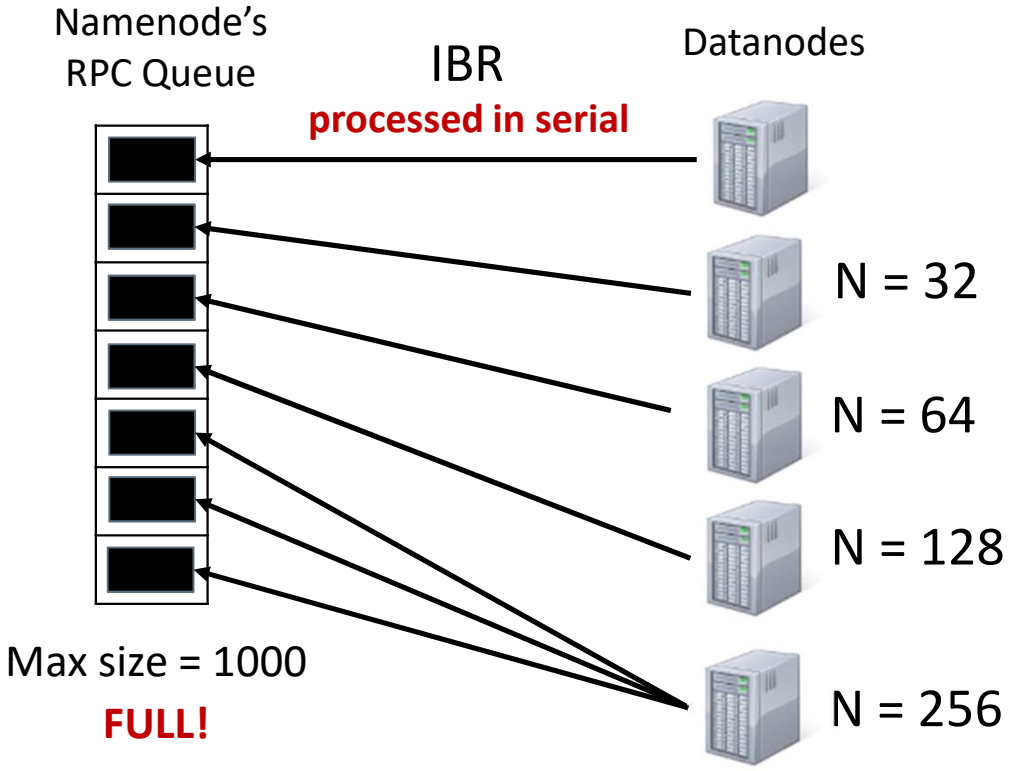
Try #3: Global Event Driven Architecture (GEDA)

- One **global event handler per service**
  - total # of threads = # of cores
- Identify **event constrains**
  - **Multi**-threaded versus **single**-threaded
- Max Colocation factor: **512 nodes**

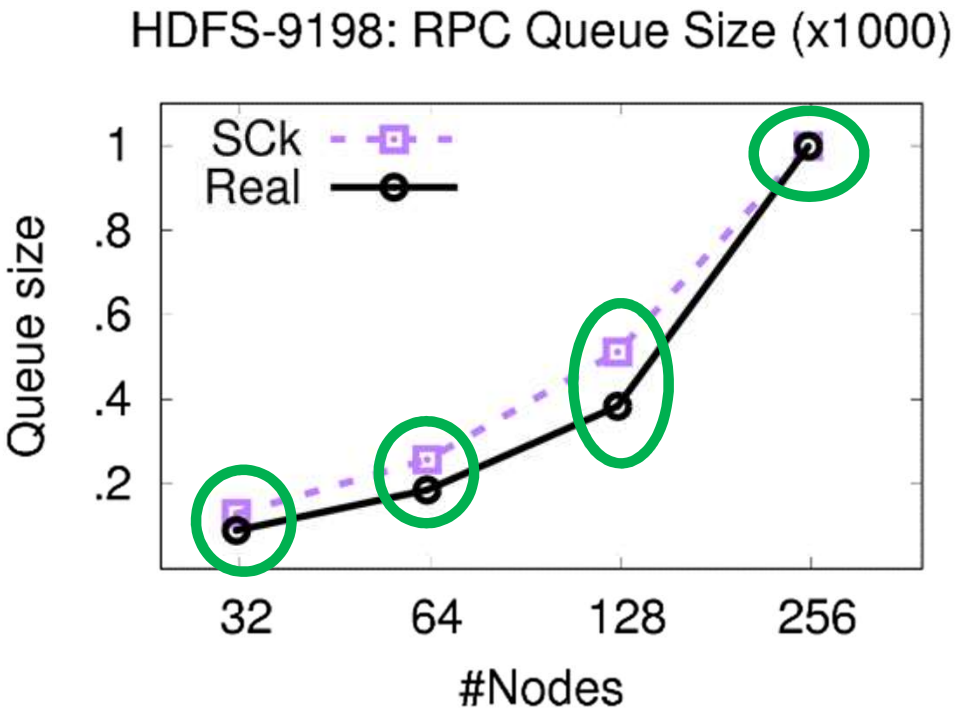


Using our Framework for HDFS-9188

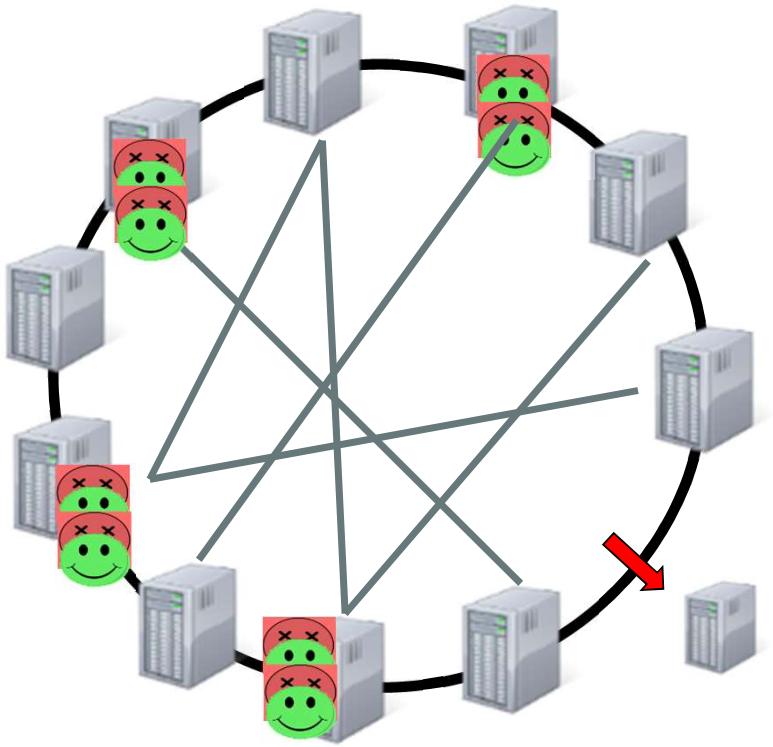
RPC Queue full = ***no other request can be served***



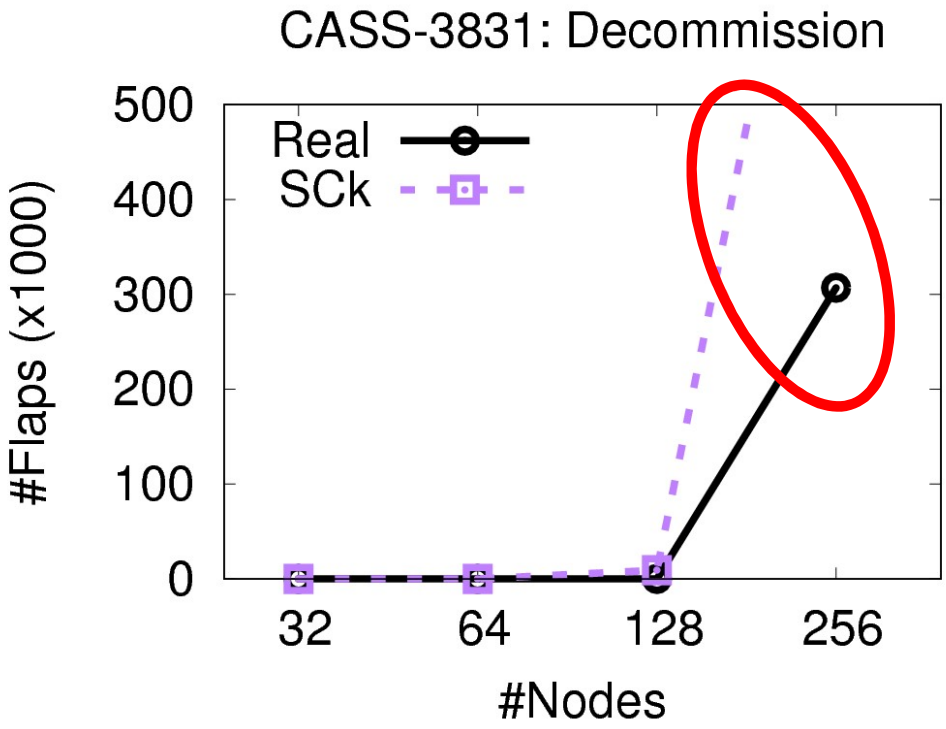
Accurate!



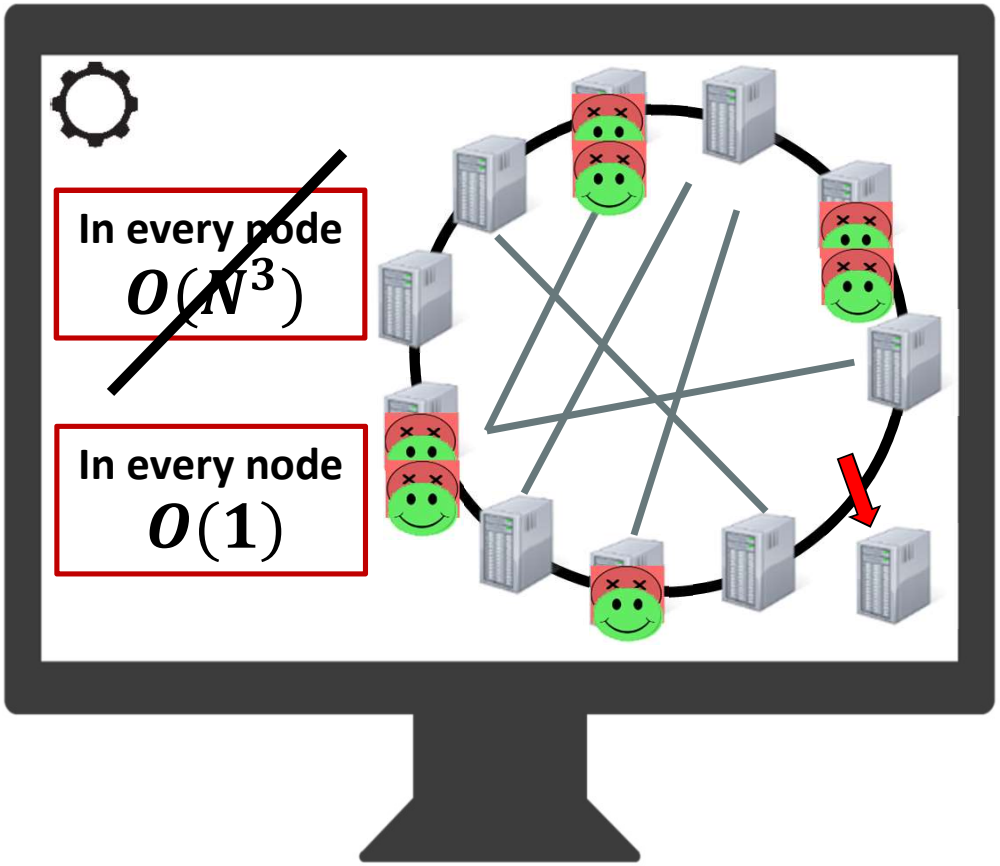
But Meanwhile in CASS-3831…



Inaccurate!



CPU Contention…



Try #4: Processing Illusion (PIL)

- **Replace CPU intensive** computation with **sleep**
- *"The key to computation is the execution time and eventual output"*
- What code blocks can be **safely** replaced by sleep?

(Easy) CPU intensive blocks

Non pertinent *operations that* ***do not affect*** *processing semantics*

```
function intense()
   for(…){
      for(…){
         for(…){
            write("hello!");
         }
      }
   }
```

$O(N^3)$

Can be PILed…

```
function intense()
   if(!ScaleCheck){
      for(…){
        for(…){
           for(…){
              write("hello!");
           }
        }
      }
   }
   else{
      t = getTime(…)
      sleep(t)
   }
```

Offline profiling

$O(1)$

(Hard) CPU intensive blocks

Pertinent *operations that* ***affect*** *processing semantics*

```
function relevantAndRelevant(input)
  for(…){
    for(…){
      for(…){
        modifyClusterState(…)
      }
    }
  }
```

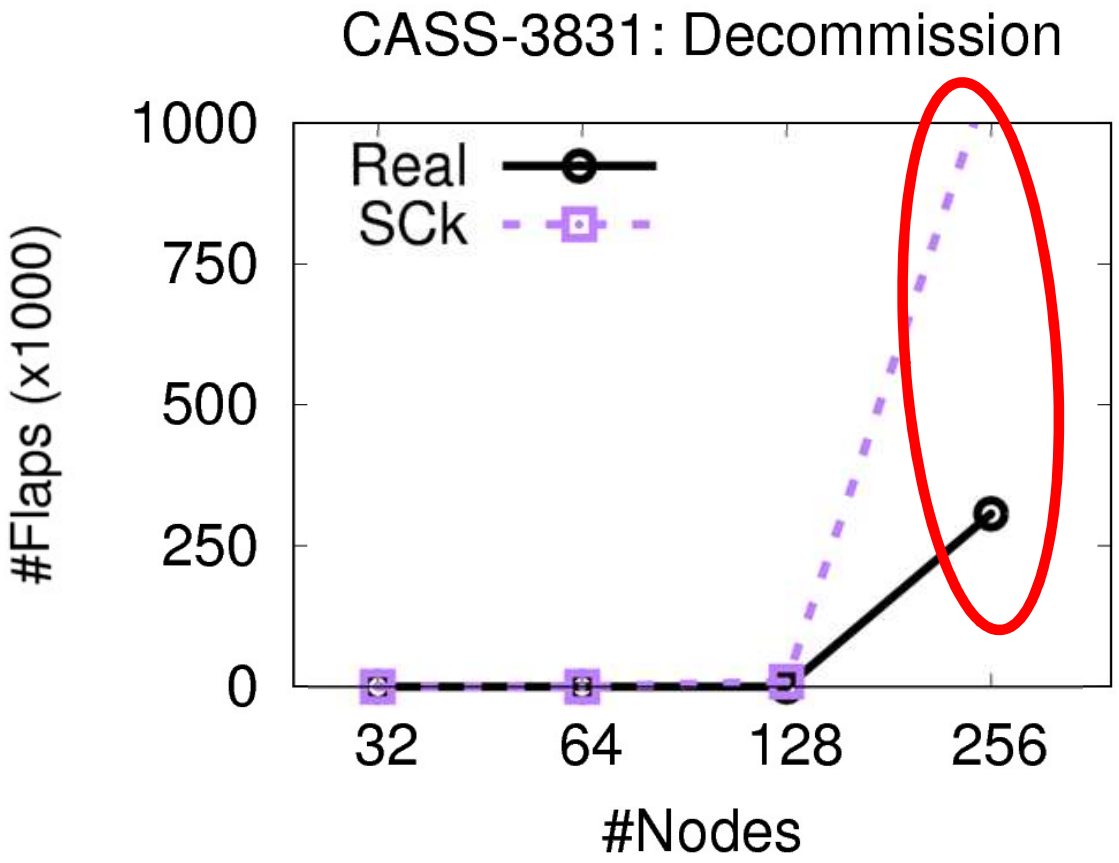
$O(N^3)$

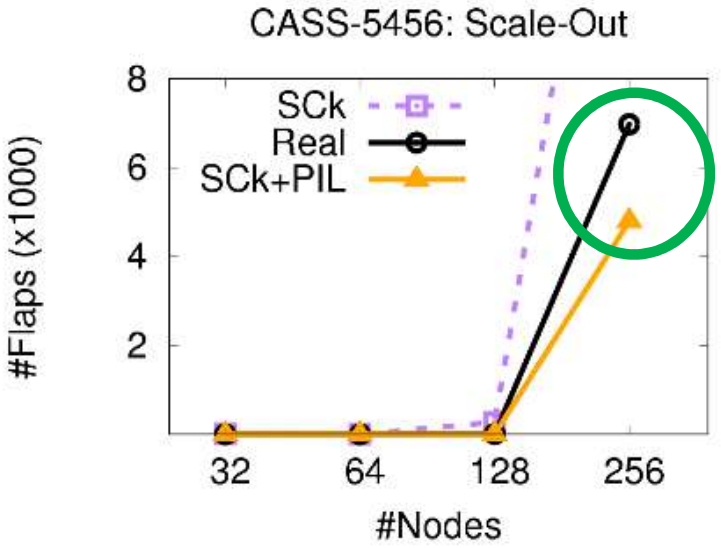
Can be PILed…

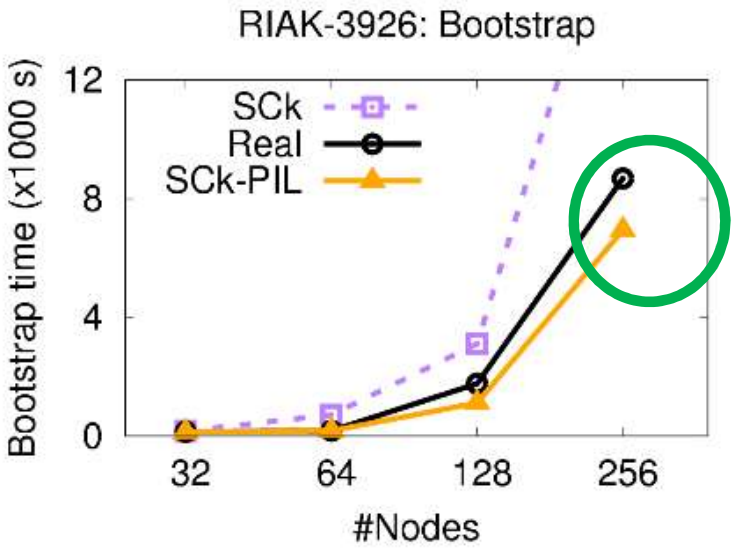
```
function relevantAndRelevant(input)
  if(!ScaleCheck){
    for(…){
      for(…){
        for(…){
          modifyClusterState(…)
        }
      }
    }
  }
  else{
    t, output = getNextState(input)
    sleep(t)
    clusterState = output
  }
```

Offline/Online memoization

How about now?





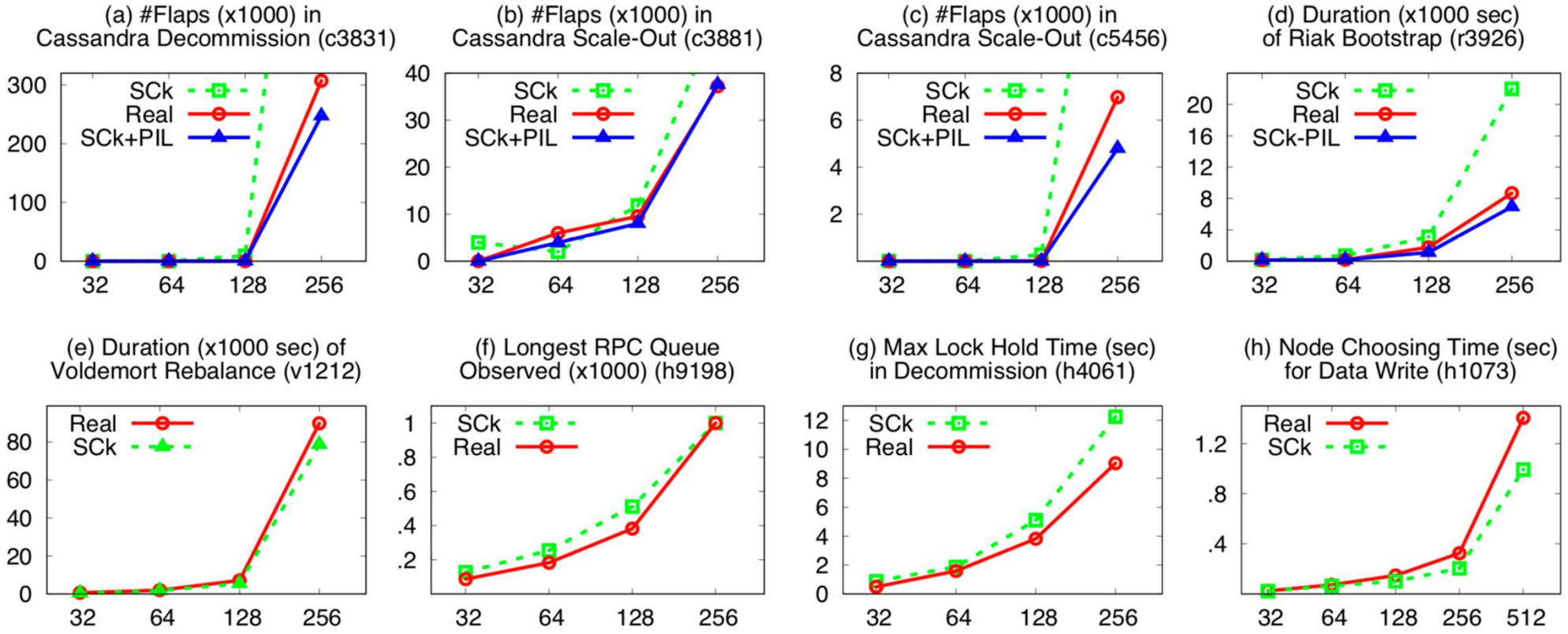


Outline

- Introduction
- SFind
- STest
- Evaluations and Conclusions

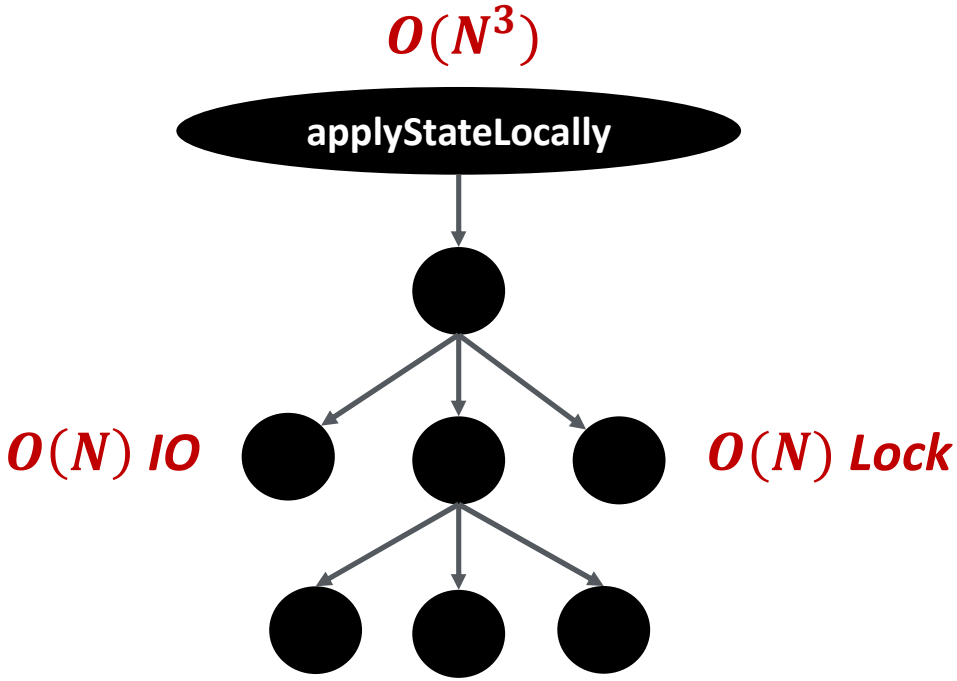
Reproducing 8 known bugs accurately

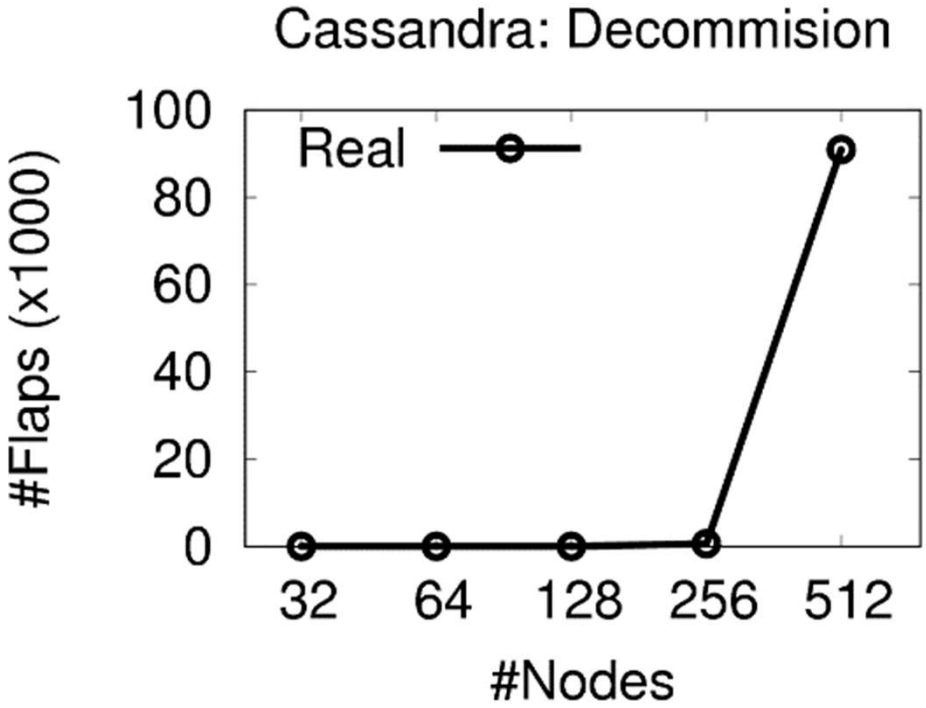
- [see paper for details]



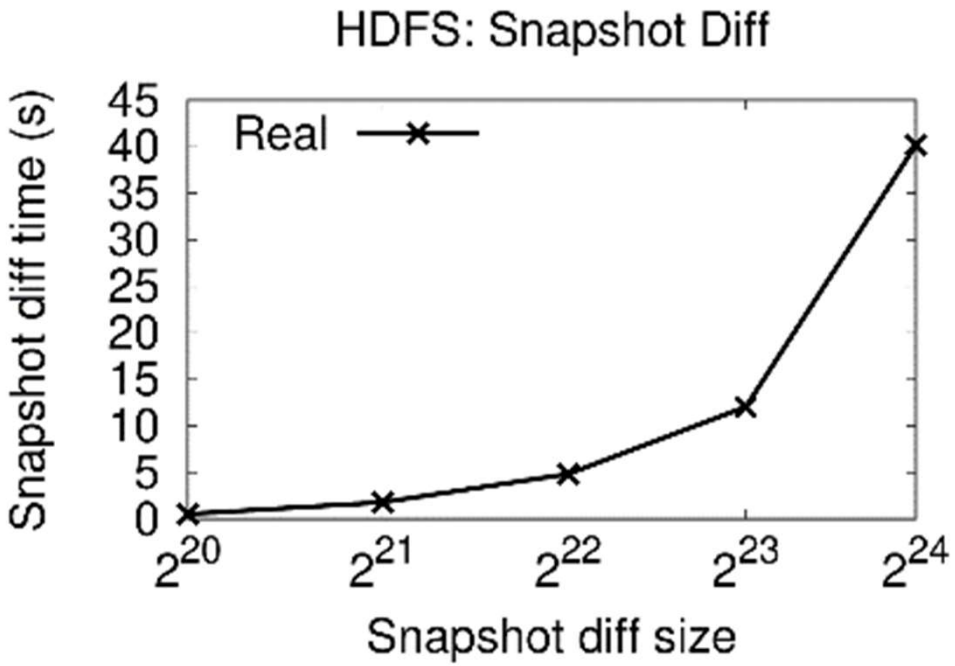
Finding New Bugs

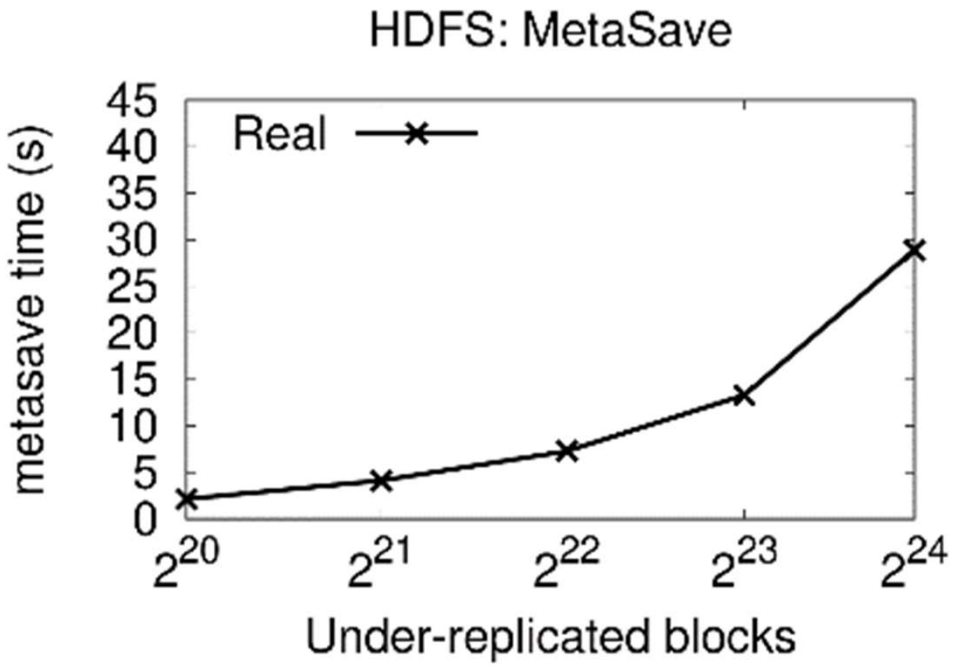
Meanwhile in latest Cassandra…





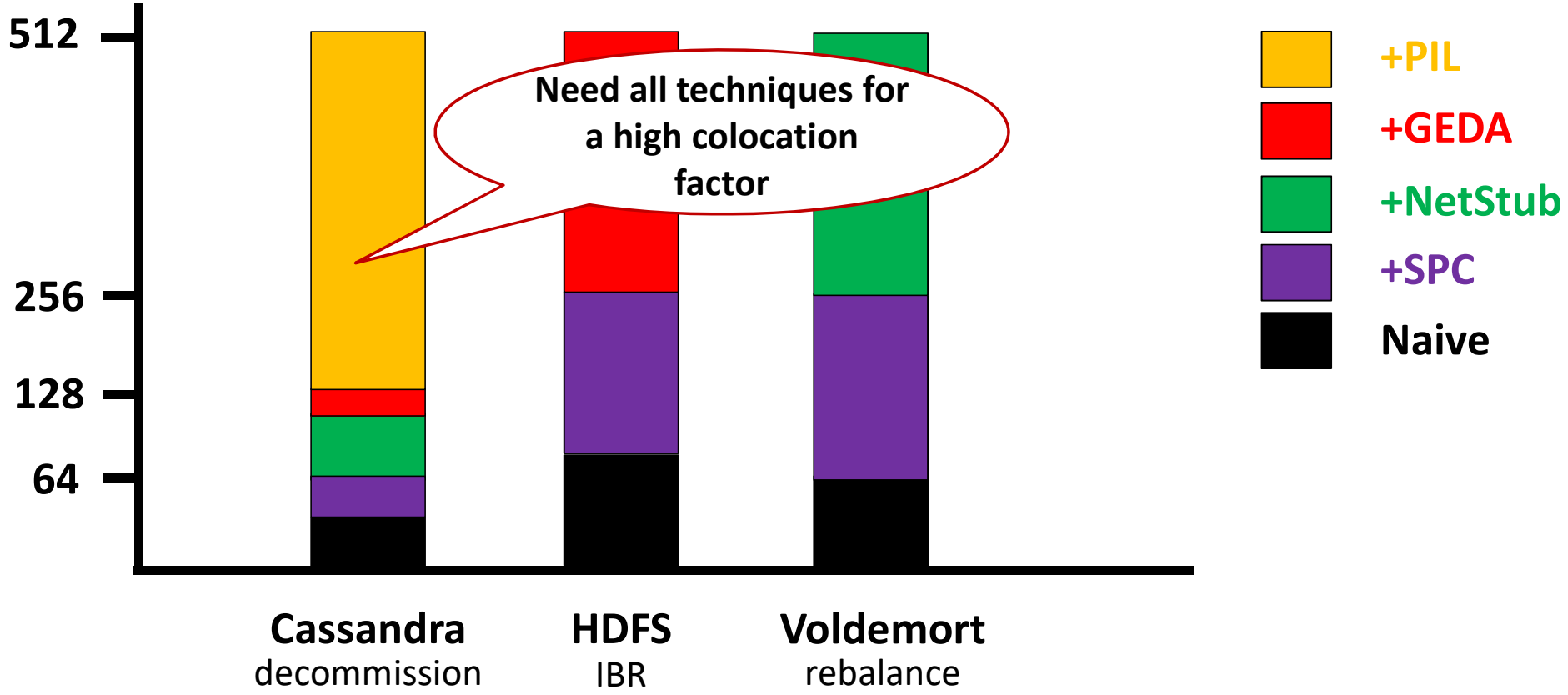
Finding New Bugs





Colocation Factor

How many nodes can we colocate in a single machine **maintaining accuracy?**

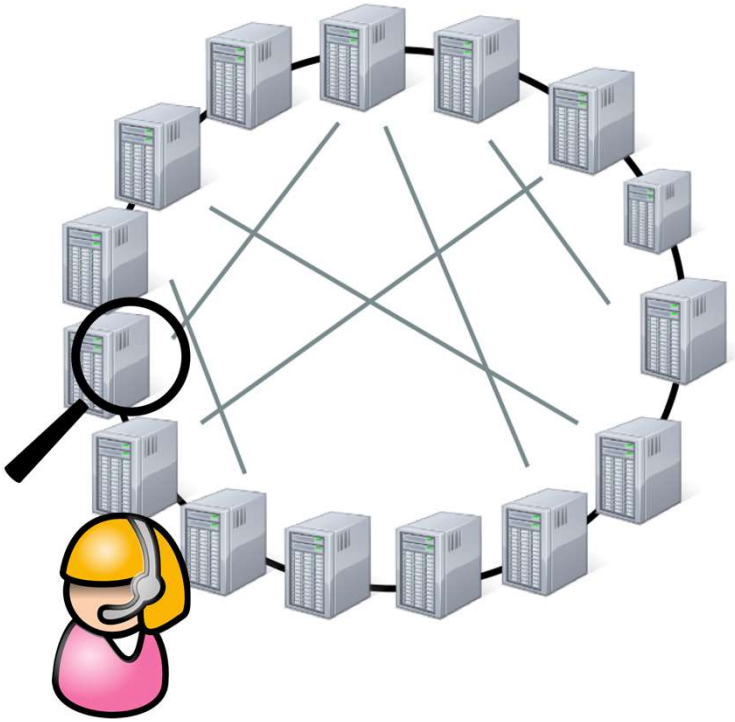


Limitations and Future Work

- Focus on scale dependent CPU/Processing time
  - E.g. cluster size, number of files, etc…
  - Scale of **load**, Scale of **failure**
- Colocation factor limited by a single machine
  - Akin to unit testing
  - **Multiple machines**
- PIL might require real scale memoization
  - Offline memoization takes too long (days)
  - Explore **other CPU contention reduction techniques**
- New code = New maintenance costs
  - Low cost (about 1% of code base)
  - Going for **zero-effort emulation**

Conclusions

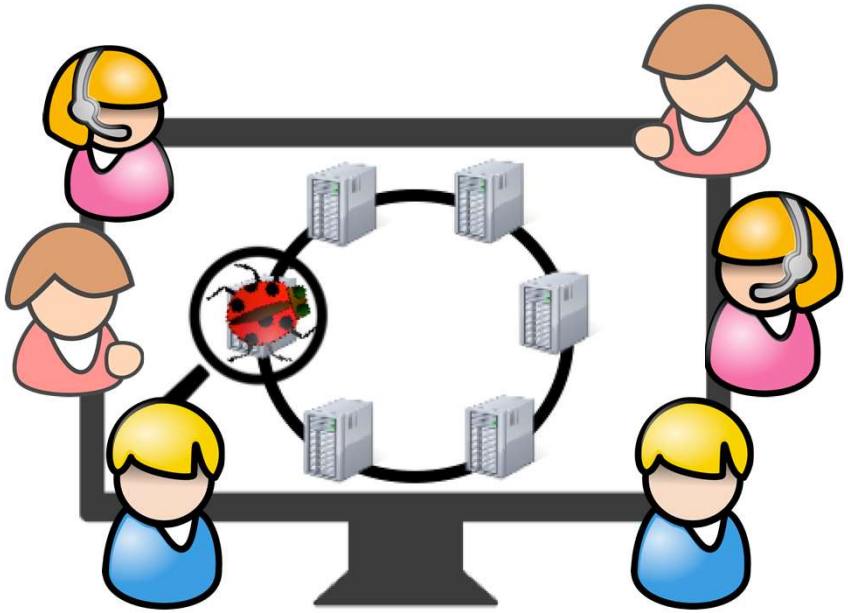
Real-scale testing



ScaleCheck:

Discovering scalability bugs

Democratizing large scale testing



Thank you!
Questions?



http://ucare.cs.uchicago.edu



http://cs.uchicago.edu



https://ceres.uchicago.edu