

Hybrid Checking with Data Structure Invariants

Moss Prescott

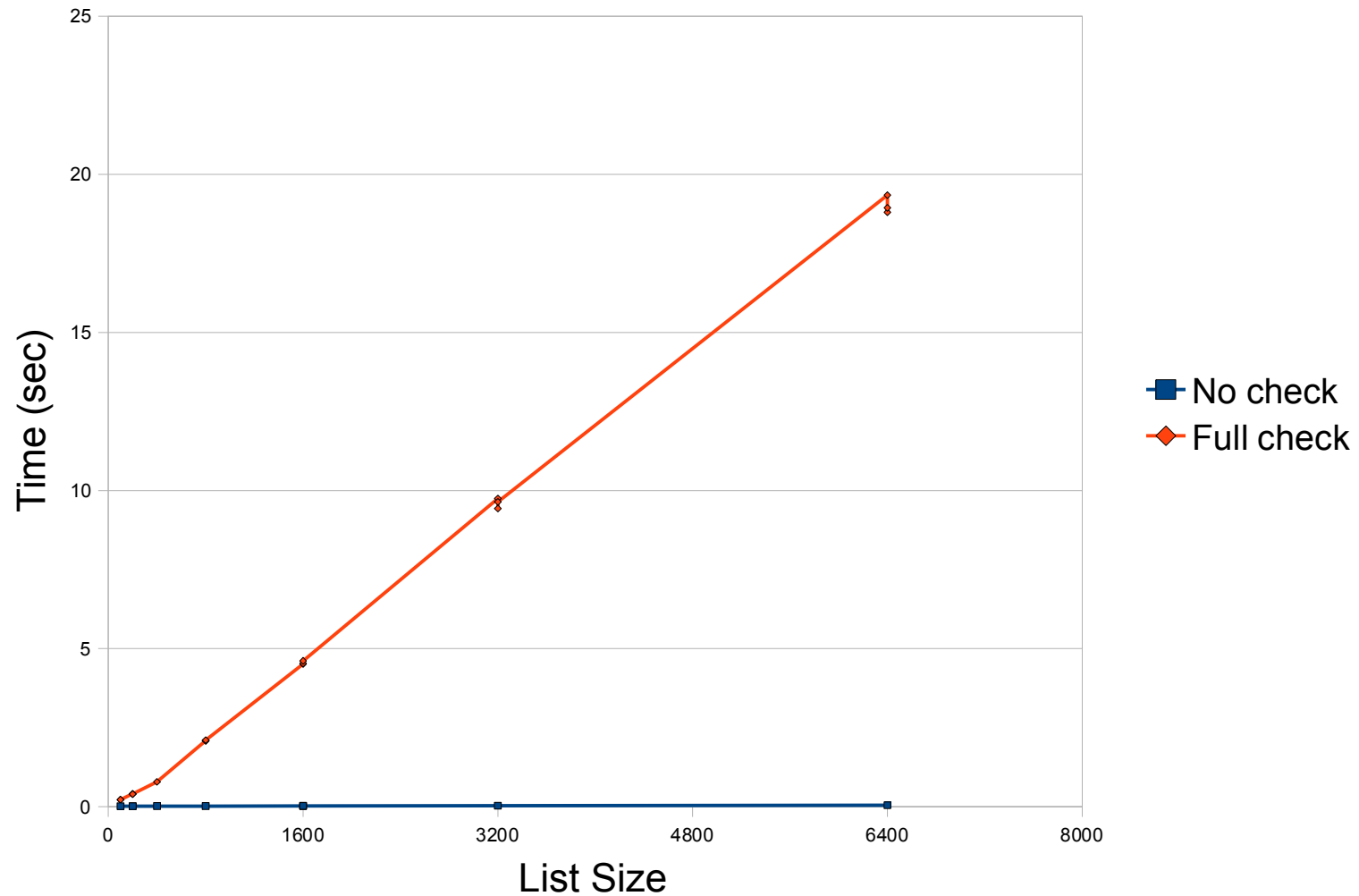
University of Colorado, Boulder
CSCI 5535, Spring 2009

Invariant Checks

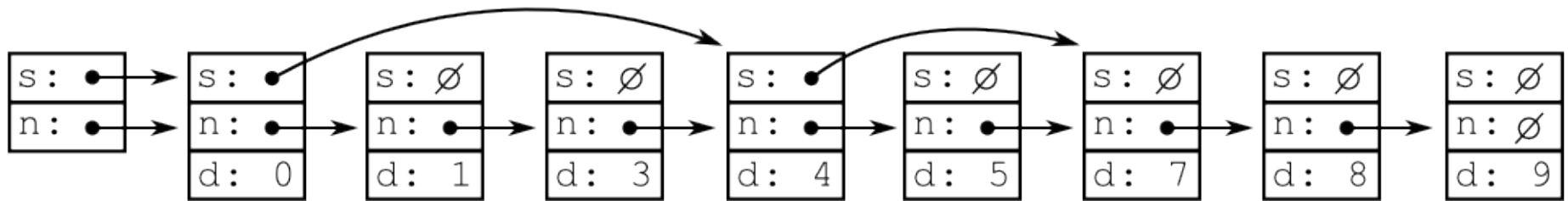
- An easy way for developers to specify/verify a data structure and its implementation
- One common approach
 - simple, recursive boolean functions
 - assert the result is `true` before and after mutating the data structure
- No sweat, and yet invariant checking isn't commonly used, even during development

Invariant-checking Overhead

Time for 100,000 Operations



Example: Skip List (Ordered) Set



- A simple data structure with some nice properties
- Still, easy to get it wrong
- How about some invariants?

Skip List Set Invariants

```
boolean ordered(Node n) {  
    return n.next == null ||  
        (n.data < n.next.data && ordered(n.next));  
}  
  
boolean skip(Node n) {  
    return n == null ||  
        (skip(n.skip) && contains(n.next, n.skip));  
}  
  
boolean contains(Node n, Node e) {  
    return n == e ||  
        (n != null && n.skip == null && contains(n.next, e));  
}
```

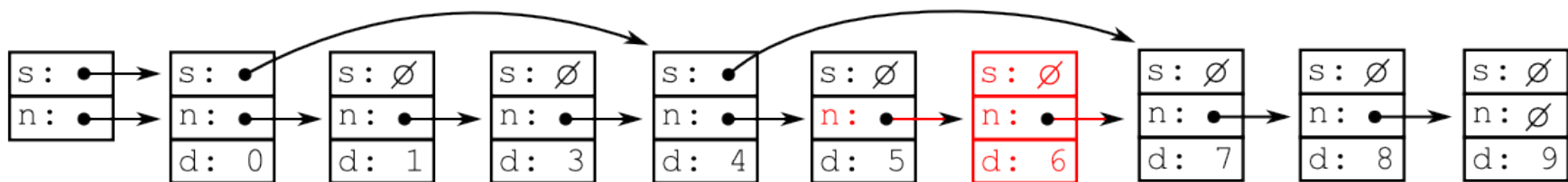
Skip List Set Mutator

```
void insert(int value) {  
    assert ordered(head) && skip(head);  
  
    Node cur = head;  
    while (cur != null && cur.data < value) ...  
  
    assert ordered(head) && skip(head);  
}
```

Simple, but checks walk the entire list (multiple times).

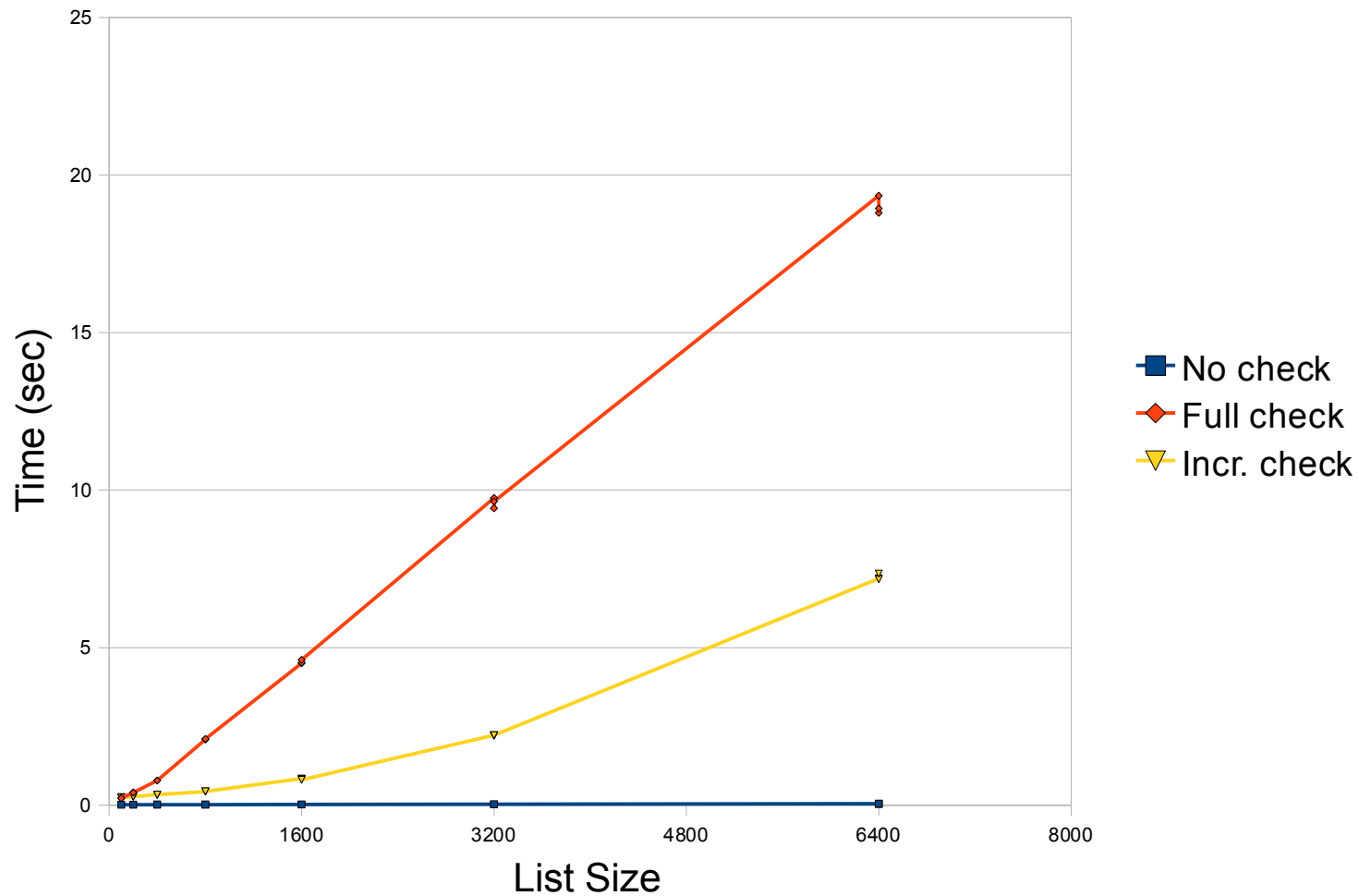
Reducing Runtime Cost

- Invariant-checking is typically $O(n)$
- Number of actually changed nodes is $O(1)$
- How can we check *only* the changed nodes?
- Use an incrementalizing transformation
 - for example, Ditto



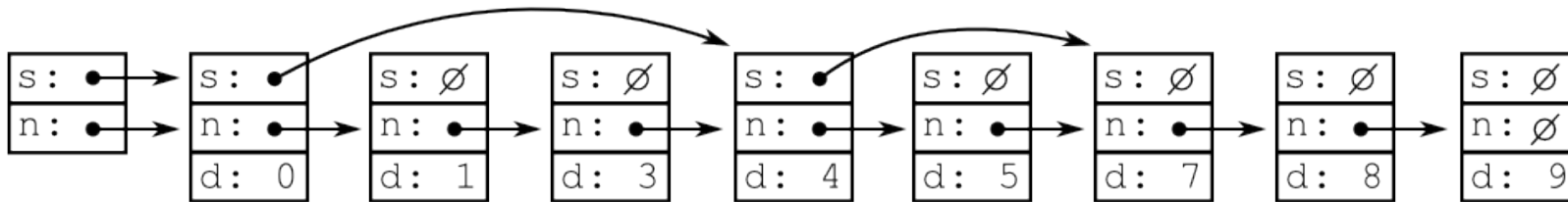
Incrementalized Checking Cost

Time for 100,000 Operations



What Went Wrong?

- insert()/delete() touches only one or two nodes
- rebalance() touches *many*
- The price of making a probabilistic algorithm partially deterministic



Solution

- 2 kinds of invariants
- We can treat them differently

```
boolean ordered(Node n) {  
    return n.next == null ||  
        (n.data < n.next.data && ordered(n.next));  
}
```

Completely unaffected
by `rebalance()`




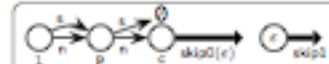

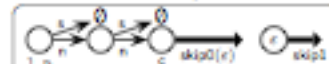
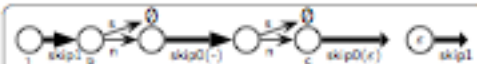
```
boolean skip(Node n) {  
    return n == null ||  
        (skip(n.skip) && contains(n, n.skip));  
}
```

Concerned only with
shape, not data.

A Static Check

- The skip() property can be statically verified
- Use Xisa:

```

void rebalance(SkipNode* l) {
  SkipNode *p, *c;
  assert (l != null && skip1(l));
1  
  p = l; // previous level 1 node
  c = l->n; // cursor
  l->s = null;
  while (c != null) {
2  
3  
4  if (c should be a level 1 node) {
5  p->s = c; // set the skip pointer of the previous level 1 node
6  p = p->s;
7  c->s = null; c = c->n;
8  
9  
10 } else {
11 c->s = null; c = c->n;
12 
13 
14 }
15 }
16 assert (l != null && skip1(l));
}

```

First Iteration
At Fixed Point

Combining Two Approaches

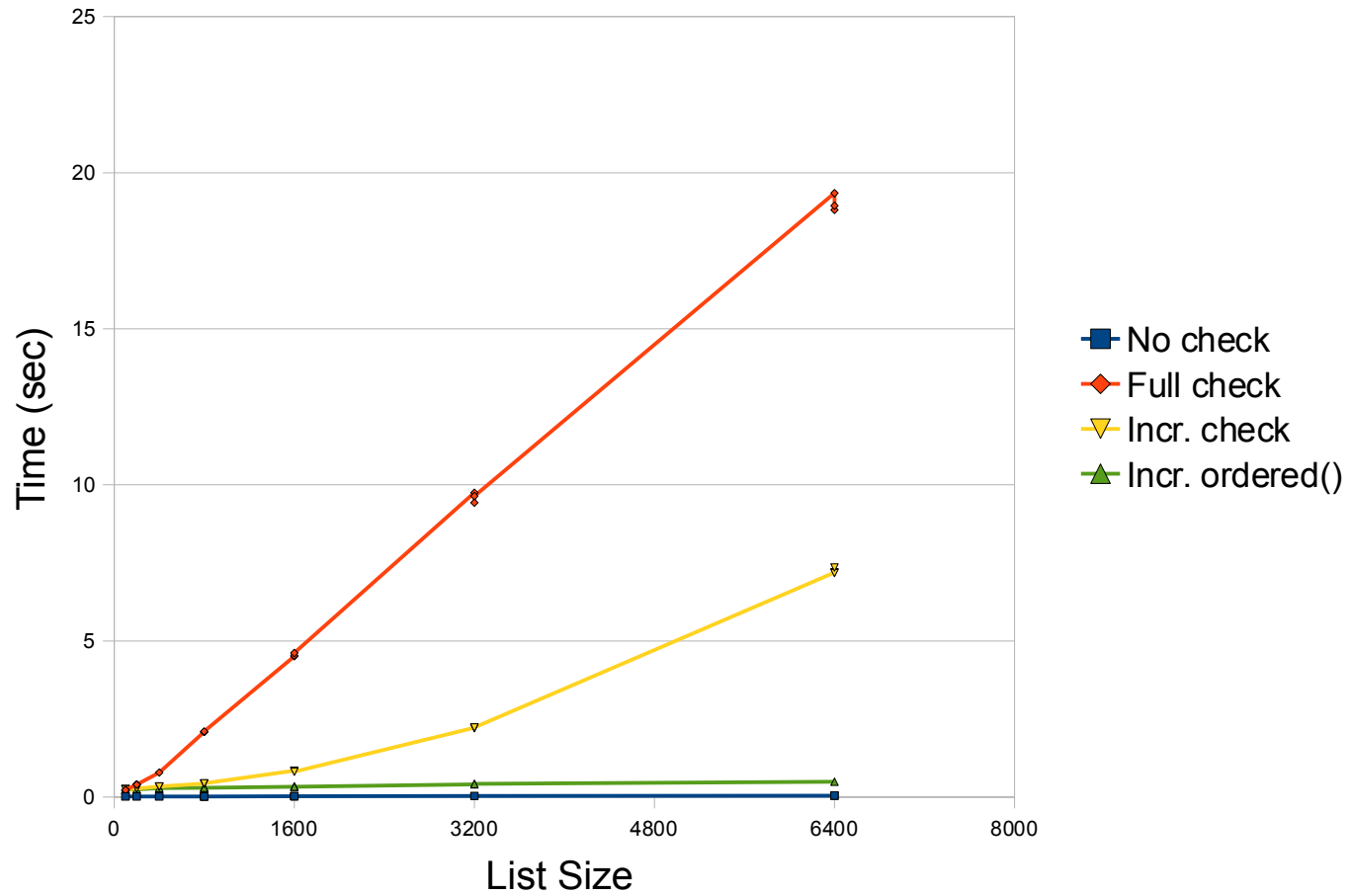
- Run a static analysis
 - a pre-condition “assert” is an *assumption*
 - a post-condition “assert” is a *goal*
- Strip out assertions that can be verified:

```
//assert ordered(head) && skip(head);  
asset ordered(head) && true;
```

- What's left over is checked (incrementally) at runtime
- This is a non-trivial programming challenge

“Results”

Time for 100,000 Operations



References

B. Chang, X. Rival, and G. Necula. Shape analysis with structural invariant checkers. In SAS, 2007

Ajeet Shankar and Rastislav Bodik. DITTO: Automatic Incrementalization of Data Structure Invariant Checks (in Java). In PLDI, 2007.

Implications

- What semantics for checker calls?

```
void insert(int value) {  
    assert ordered(head) && skip(head);  
  
    Node cur = head;  
    while (cur != null && cur.data < value) ...  
  
    assert ordered(head) && skip(head);  
}
```

What About the Programmer?

- Checkers look simple, but now fraught with (multiple) meanings
- Is performance predictable?
- Is this really Java anymore?