Overcoming the Data-flow limit with **Structural Approximation**

Vignesh Balaji Brandon Lucia Radu Marculescu

Carnegie Mellon University

Electrical & Computer ENGINEERING

Carnegie Mellon University

Outline

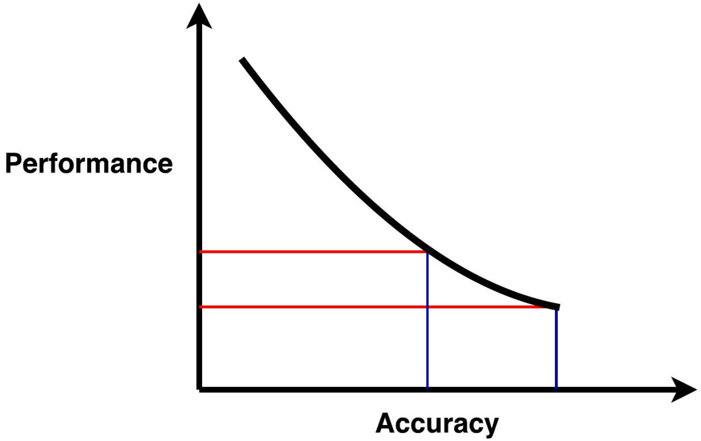
- Trends in Approximate Computing
- Structural Approximation
- Making Structural Approximation viable

Approximate Computing

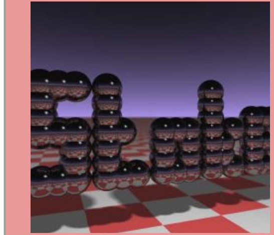
Correctness is *costly*

AND

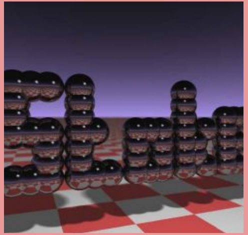
Precision is not always *required*



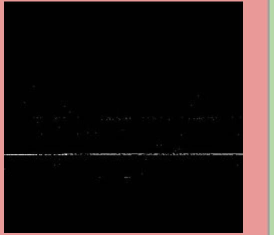
Successful Domains for Approximation



Precise Output

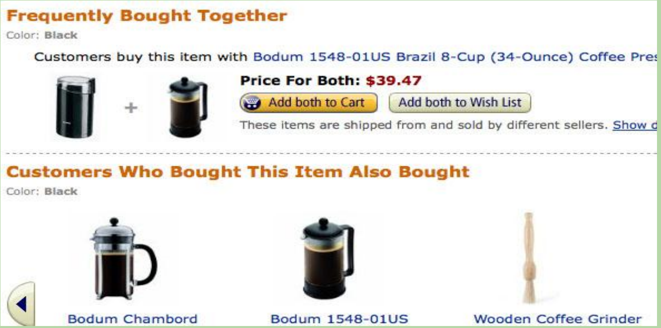


Approx. Output



Pixel error values

Performance improves by 6.4% *



Frequently Bought Together

Color: **Black**

Customers buy this item with Bodum 1548-01US Brazil 8-Cup (34-Ounce) Coffee Pres

Price For Both: $39.47

These items are shipped from and sold by different sellers. Show d

Customers Who Bought This Item Also Bought

Color: **Black**

Bodum Chambord

Bodum 1548-01US

Wooden Coffee Grinder

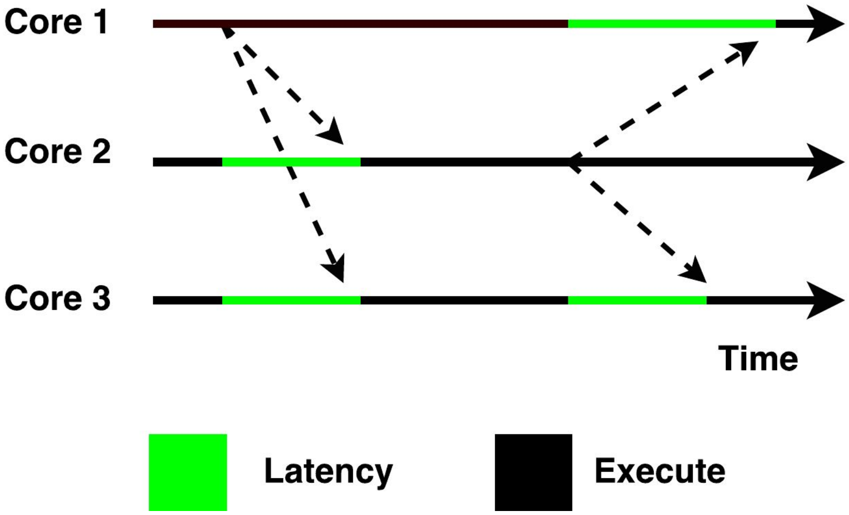
Multiple *correct* answers possible.

Most existing approximations typically target only *numeric values*

* Eric Schkufza, Rahul Sharma, and Alex Aiken. 2014. Stochastic optimization of floating-point programs with tunable precision. In *Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation* (PLDI '14)

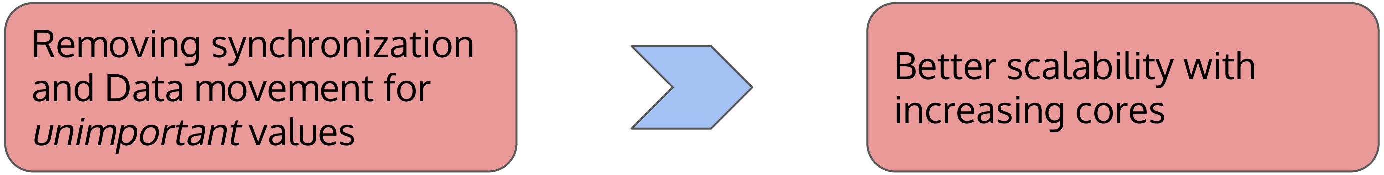
Another Interesting Domain - Parallel Computing

- Parallel programs communicate to enforce *correctness*:
  - Synchronization primitives (locks, barriers, mutexes, etc.)
  - Cache Coherence
- Communication limits *scalability*



Should we still communicate for values that are not *critical* for the app's correctness?

Removing Communication via Approximation



Removing synchronization and Data movement for *unimportant* values

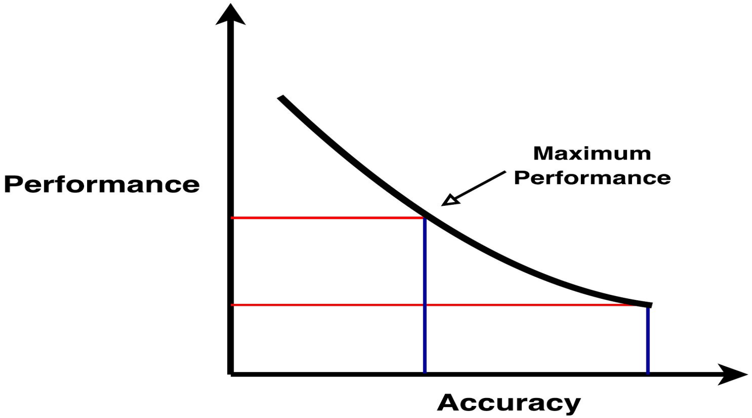
Better scalability with increasing cores

Prior work has successfully removed synchronization for such *values*

Rely, Relaxing Synchronization for Performance and Insight, Dancing With Uncertainty, Unsynchronized Techniques for Approximate Parallel Computing, ...

Are such *value* approximations enough? Or is there a need for something more?

Quantifying the Performance Limit



Definition of correctness for the limit study -

The application should not *crash*

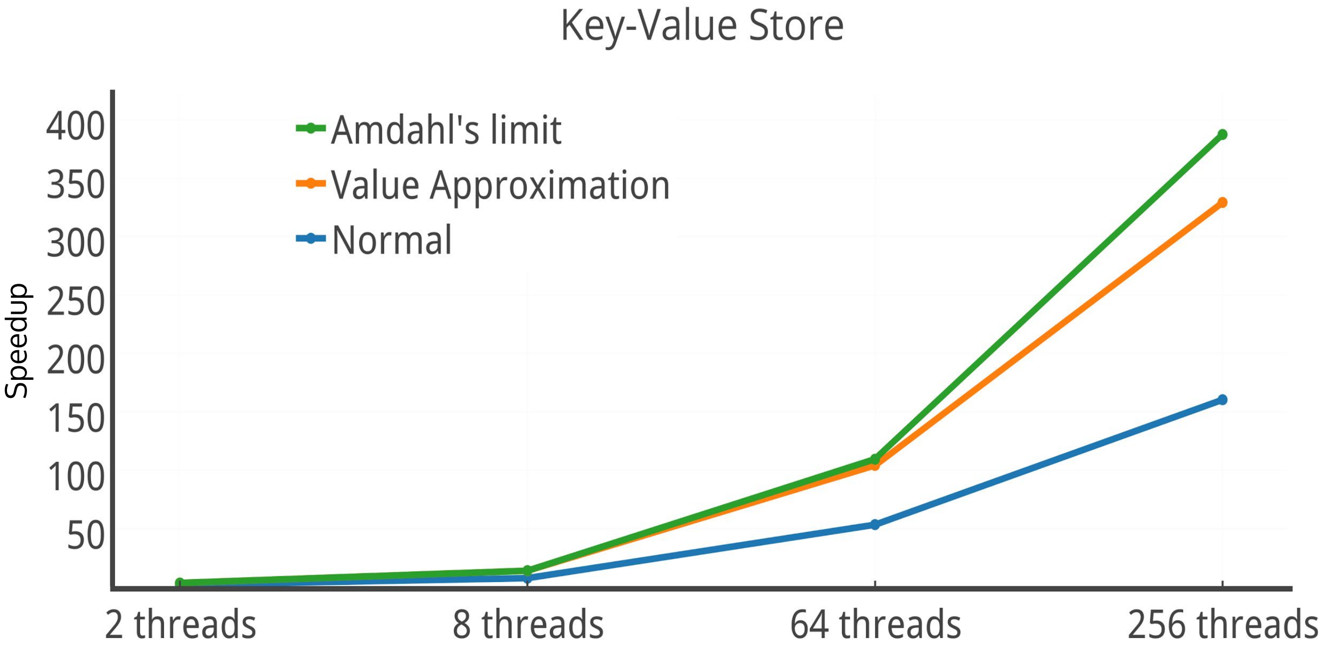
Simulated the maximum performance possible with Value Approximation :

➢ Removed *all* locks guarding values

➢ Removed all *effects* of cache coherence

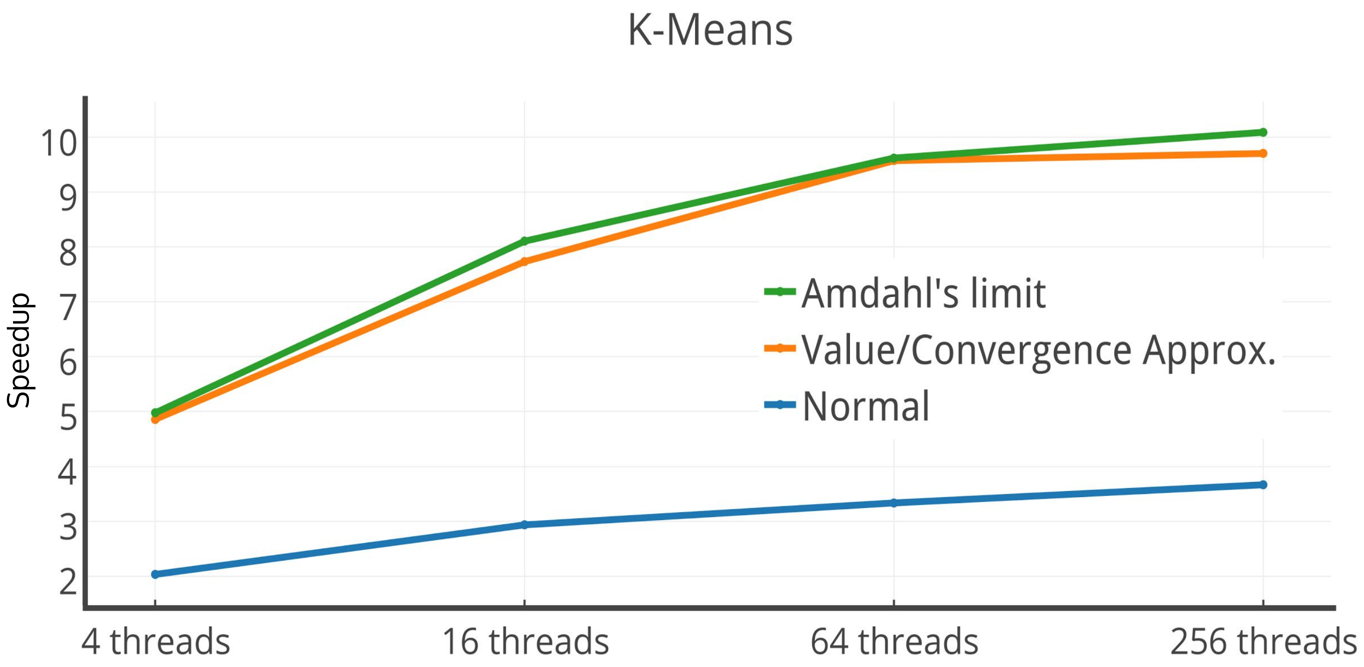
➢ Removed the *cost* of performing reduction operations

Approximating Key-Value Store



```
Lock(table[x].mutex);
table[x].data++;
Unlock(table[x].mutex);
```

Approximating K-Means



Step 1: Assign points to the closest cluster

Step 2: Compute means of the new clusters

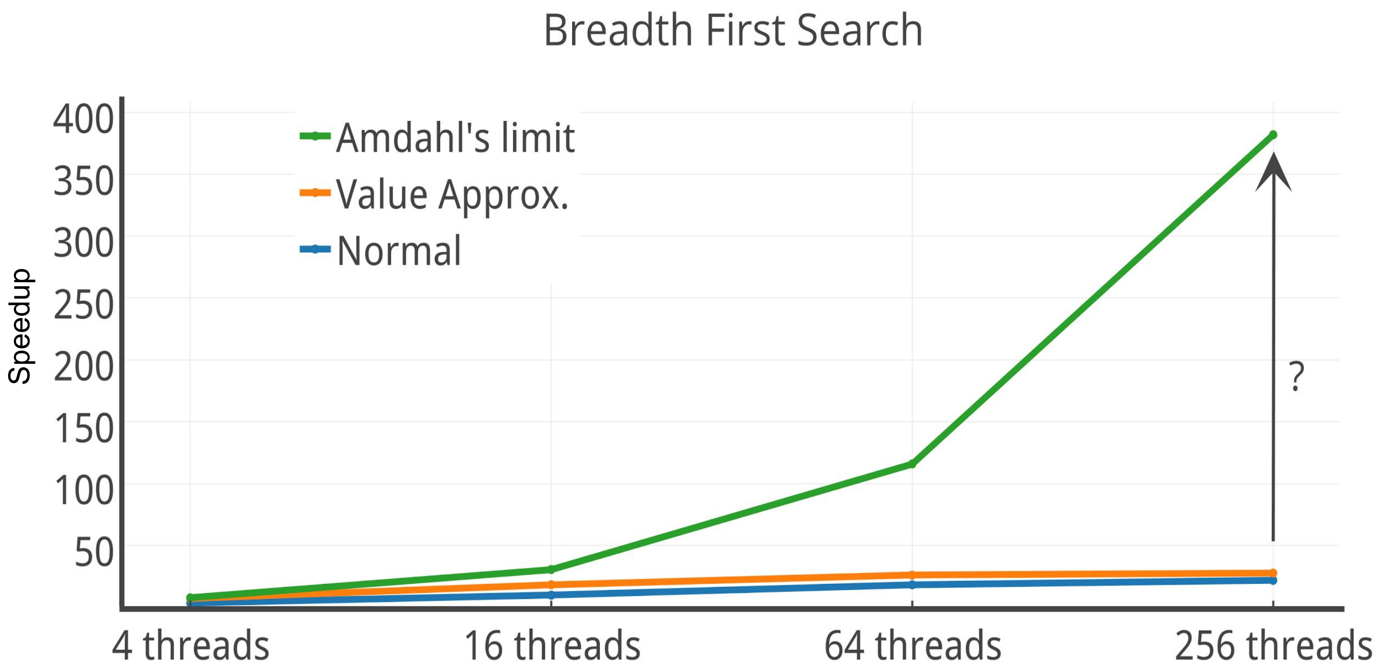
Value Approximation Works Well

- Approximating data *values* took performance close to Amdahl’s limit

- Previous two applications only shared *numerical values*

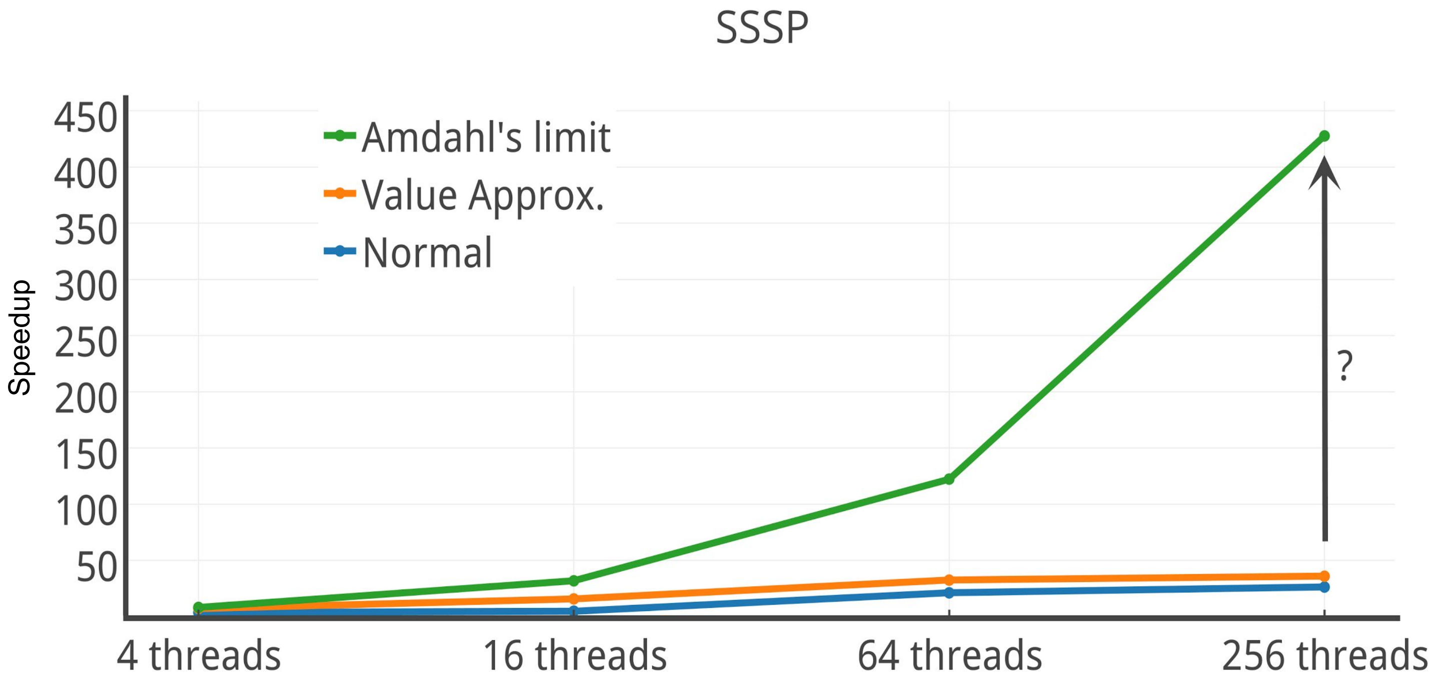
What if we have applications that share *more* than just numerical values?

Breadth First Search



Data structure that holds the children nodes of a frontier is shared

Single Source Shortest Path



Data structure maintaining nodes closest to the source is shared

End of the Road for Value Approximation

What’s different in these applications?

These application *share data structures* in addition to values

What if we remove synchronization for operations guarding data structures?

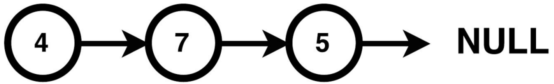
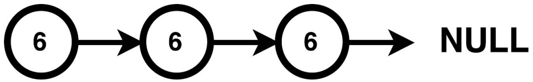
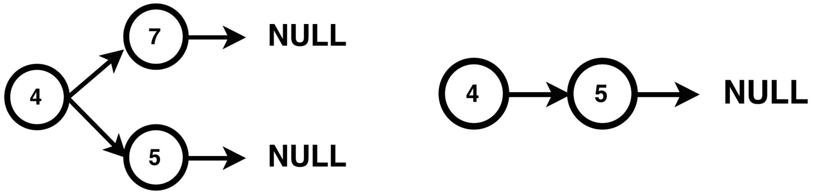
Removing synchronization for data structures leaves them *inconsistent*

How do we reason about output quality now?

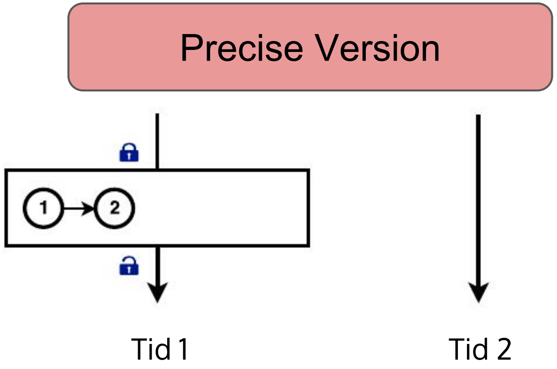
Outline

- Trends in Approximate Computing
- Structural Approximation
- Making Structural Approximation viable

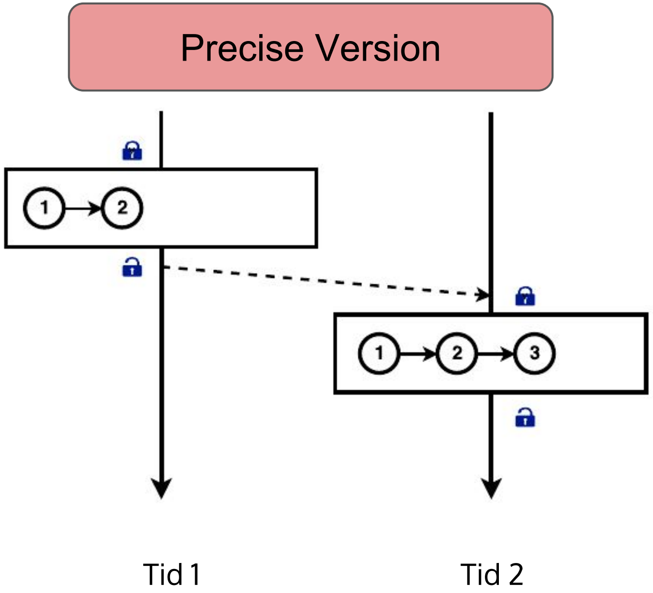
Overview of Structural Approximation

Precise Output	
Output after Value Approximation	
Output after Structural Approximation	

Mechanism of Structural Approximation

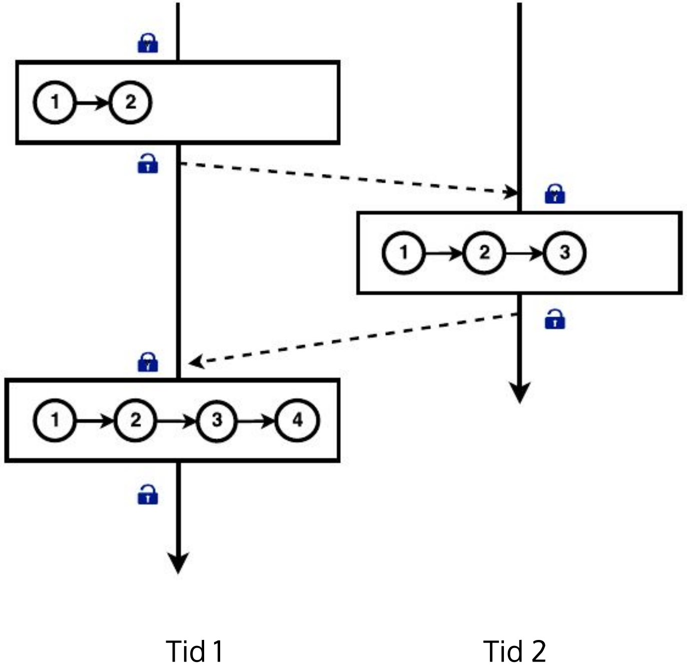


Mechanism of Structural Approximation



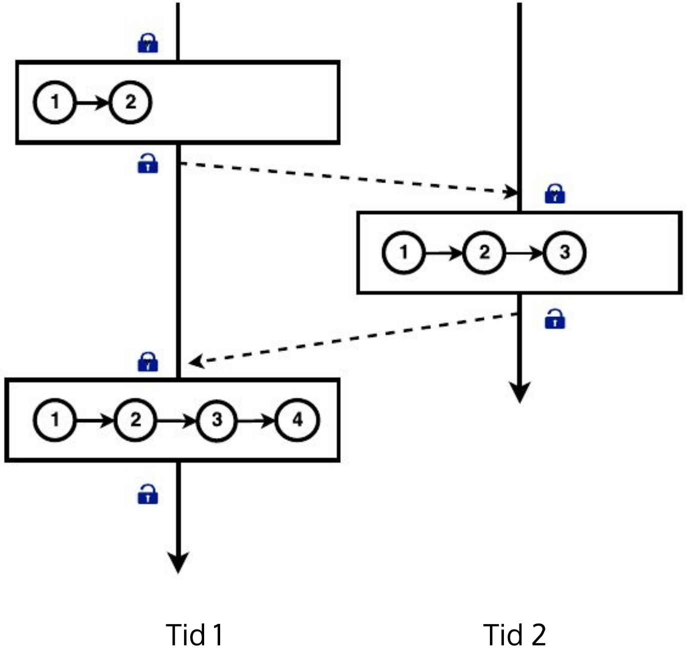
Mechanism of Structural Approximation

Precise Version

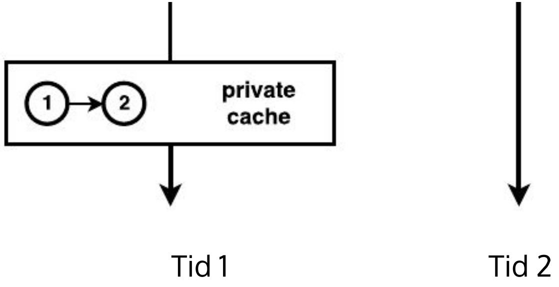


Mechanism of Structural Approximation

Precise Version

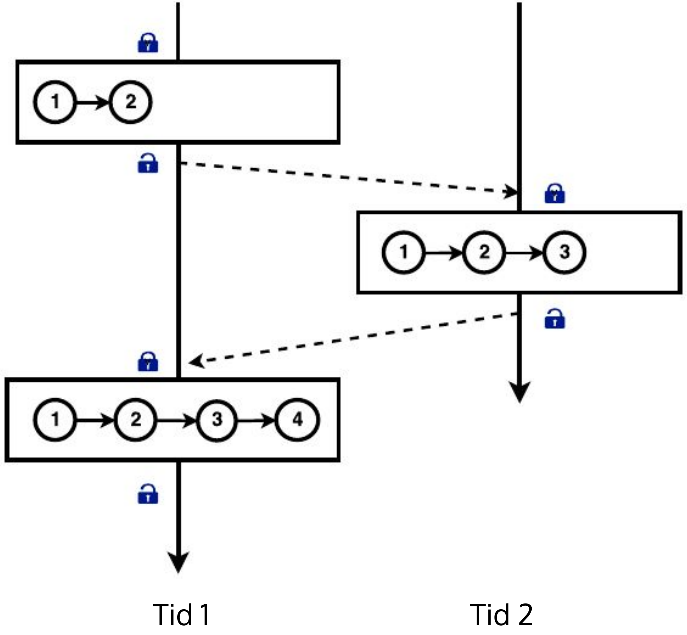


Approximate Version

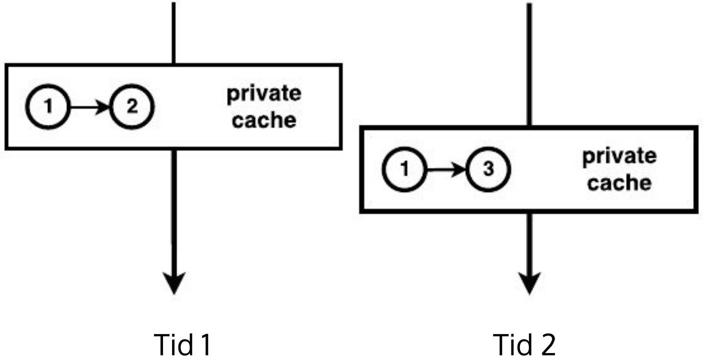


Mechanism of Structural Approximation

Precise Version

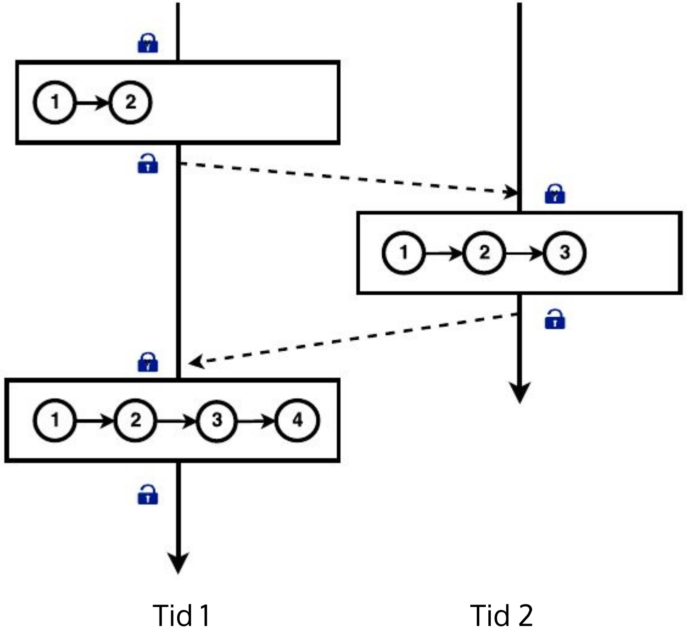


Approximate Version

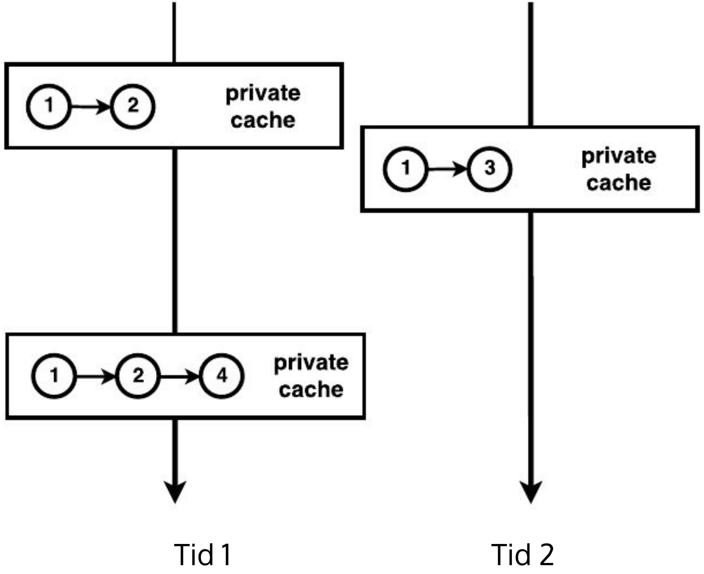


Mechanism of Structural Approximation

Precise Version



Approximate Version



How can we ensure *usable* results from Structural Approximation?

Outline

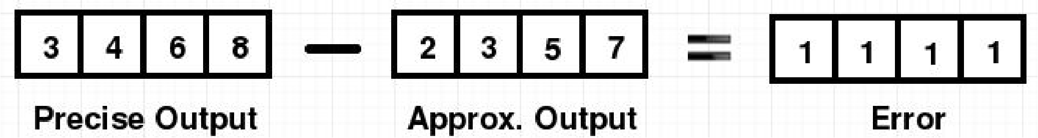
- Trends in Approximate Computing
- Structural Approximation
- Making Structural Approximation viable

Existing Definition of Correctness

- Value based approximations use *numerical distance* to measure error

$$Metric = (Output_{precise} - Output_{approximate})$$

- Makes sense for value approximation



3	4	6	8	−	2	3	5	7	=	1	1	1	1

We cannot use this error definition for Structural Approximation!

New Definition of Correctness

The new definition should:

- ➢ Accommodate *temporary* inconsistency in data structures
- ➢ *Tolerate* the loss of a few elements in data structures

Need mechanisms to *enforce* this correctness definition

Resilient Operators

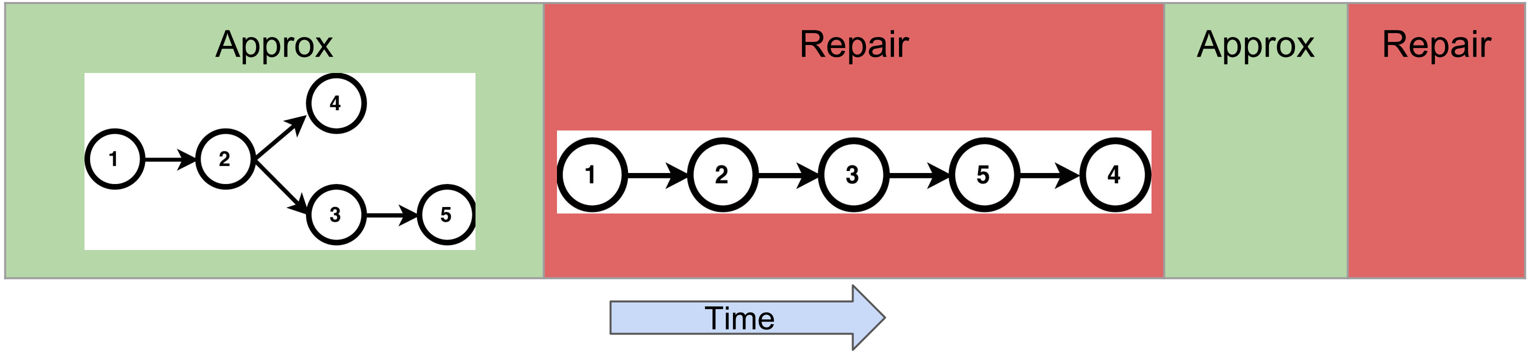
- Operators that *successfully* service queries even on inconsistent data structures
- Similar to the concept of Defensive Programming

```
int risky_visitor(int size){
  while(i < size){
    do_work(node->value); //Fails if elements
                          //dropped from list
    node = node->next;
    i++;
}}
```

```
int secure_visitor(int size){
  while(i < size){
    if (node != NULL){
       do_work(node->value); //saved from
                             //NULL ptr exception
       node = node->next;
    i++;
    }
    else
      break;
}}
```

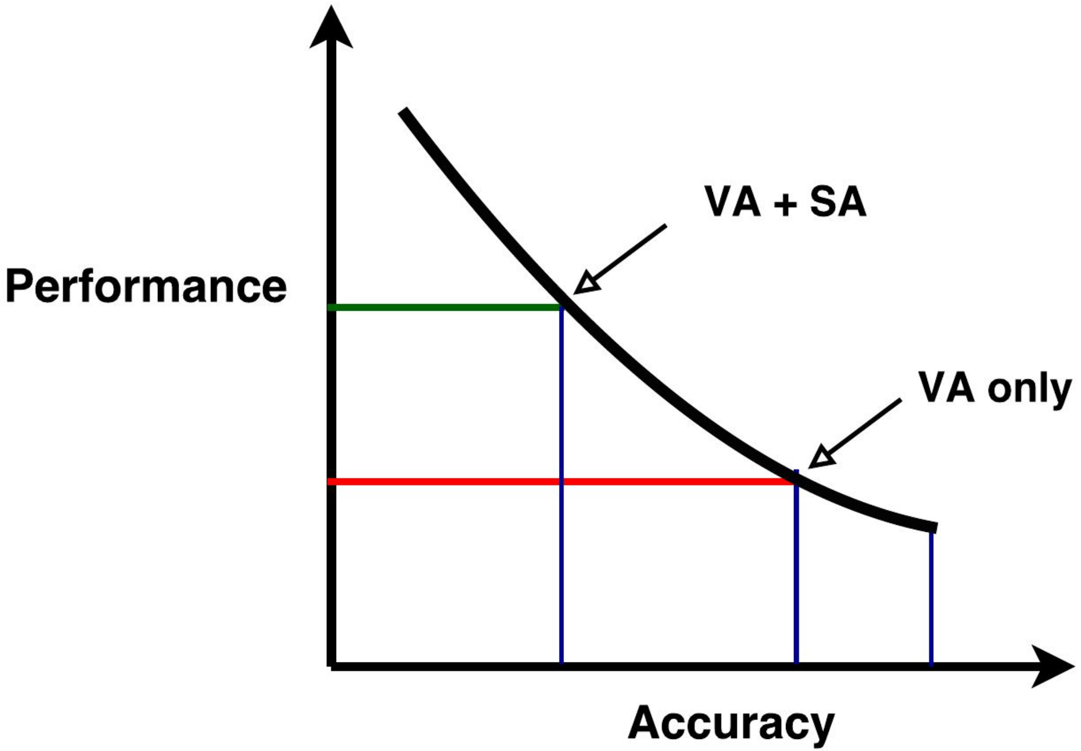
Handle all failures that are possible due to Structural Approximation

Data Structure Repair

- 

- Major challenge - How to implement *low-cost* repairs?
  - ➢ Offload repair to cloud?

Role of SA in the Future of Approximate Computing



Thank You