# A Case for Optimistic Coordination in HPC Storage Systems

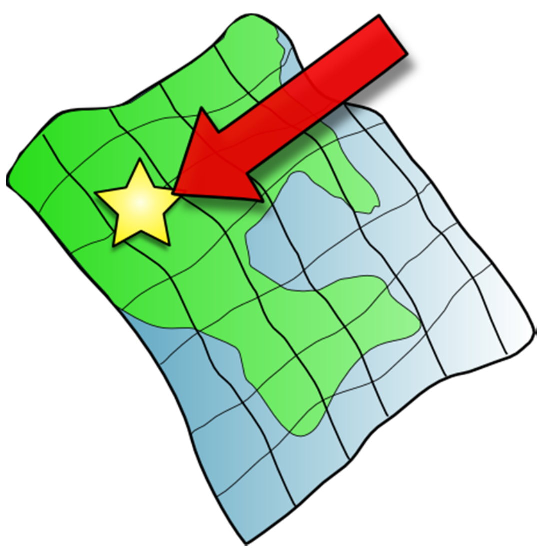
Philip Carns, Kevin Harms, **Dries Kimpe**, Justin M. Wozniak,
Robert Ross, Lee Ward, Matthew Curry, Ruth Klundt,
Geoff Danielson, Cengiz Karakoyunlu, John Chandy,
Bradley Settlemeyer, William Gropp

# Overview

- Introduction
  - Situation
- Problem Description
  - Driver Application
  - Existing approach
- Proposed solution
  - Optimistic Coordination
  - The A-B-A Problem
- Evaluation
- Conclusions & Future Work

# Situation of this work

## Techniques for Application Coordination while Accessing Data

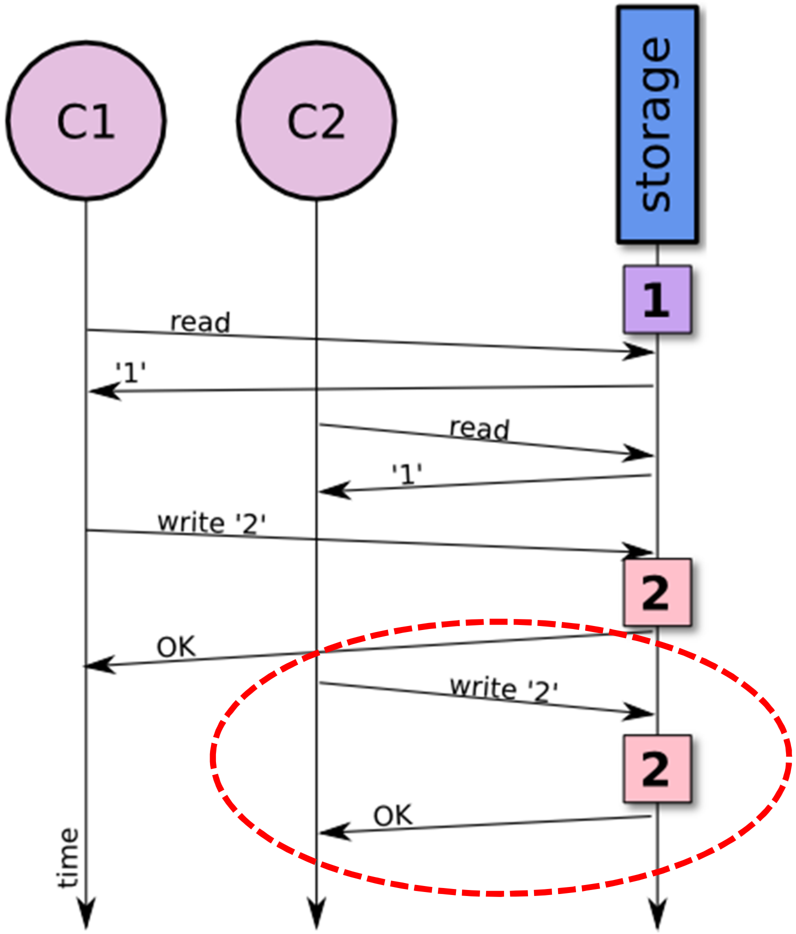


Where:

- **Data** is data stored on a **storage system**
  - **Typically shared storage**
  - **Mainly targeting High-Performance Computing**
- **Applications cannot** easily **coordinate** among themselves
- Access can be **reading** or **writing (update)**

**Examples**: Loosely coupled calculations, GUPS-workloads, parallel histogram , unaligned access in block based systems

# The Issue:
# Concurrent Updates to Shared Storage




- Client wants to **increment** counter.
  - No "increment" in storage, so performs **read** followed by **write**.
- **Multiple** clients execute **concurrently**
  - Certain execution schedules will lead to lost updates or incorrect results.

Update from C1 lost!

# Motivating Example: Parallel Histogram Calculation

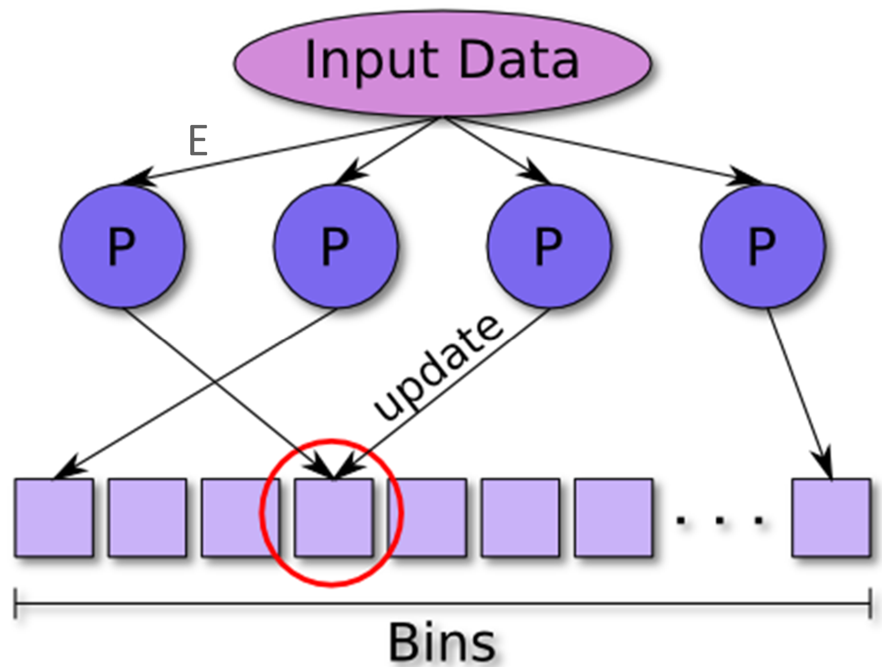
We want to learn more about the distribution of a certain property in a data set.

- For each entry (E) in the data set, **classify** to a **bin**:

  binnum = classify (E)

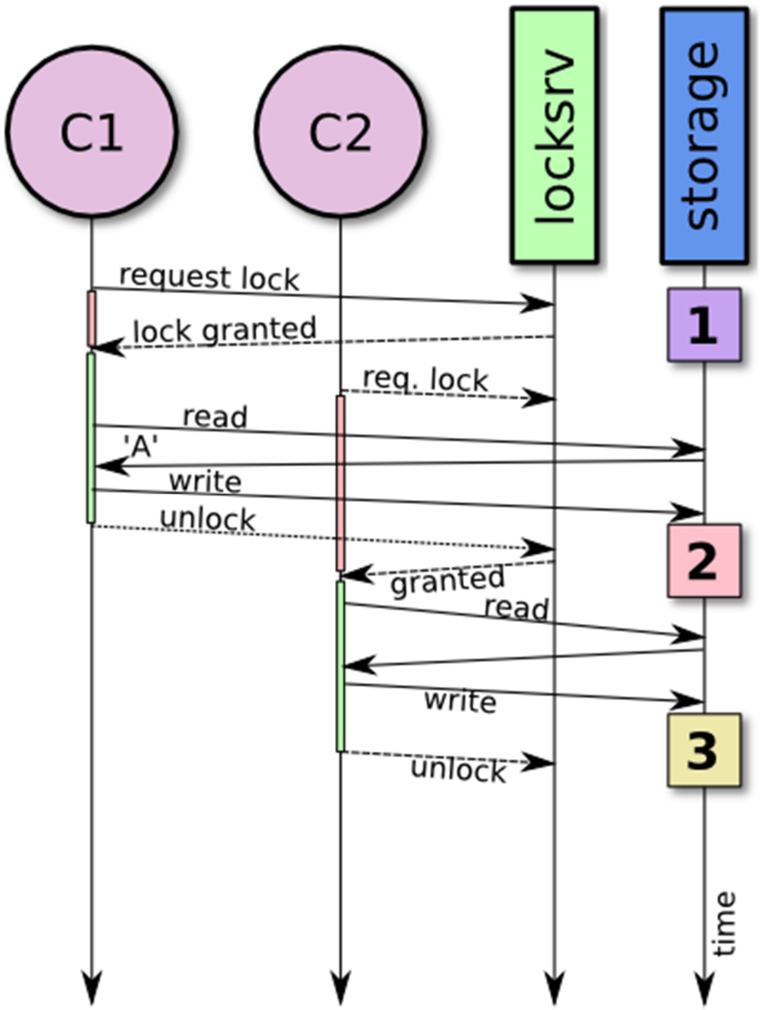
- **Update** bin *binnum*:

  1. oldcontent = **read** (bin)
  2. newcontent = update (content, E)
  3. **write** (bin, newcontent)




**Updates** to the **same bin** need to be synchronized!

# A Solution: Distributed Locking



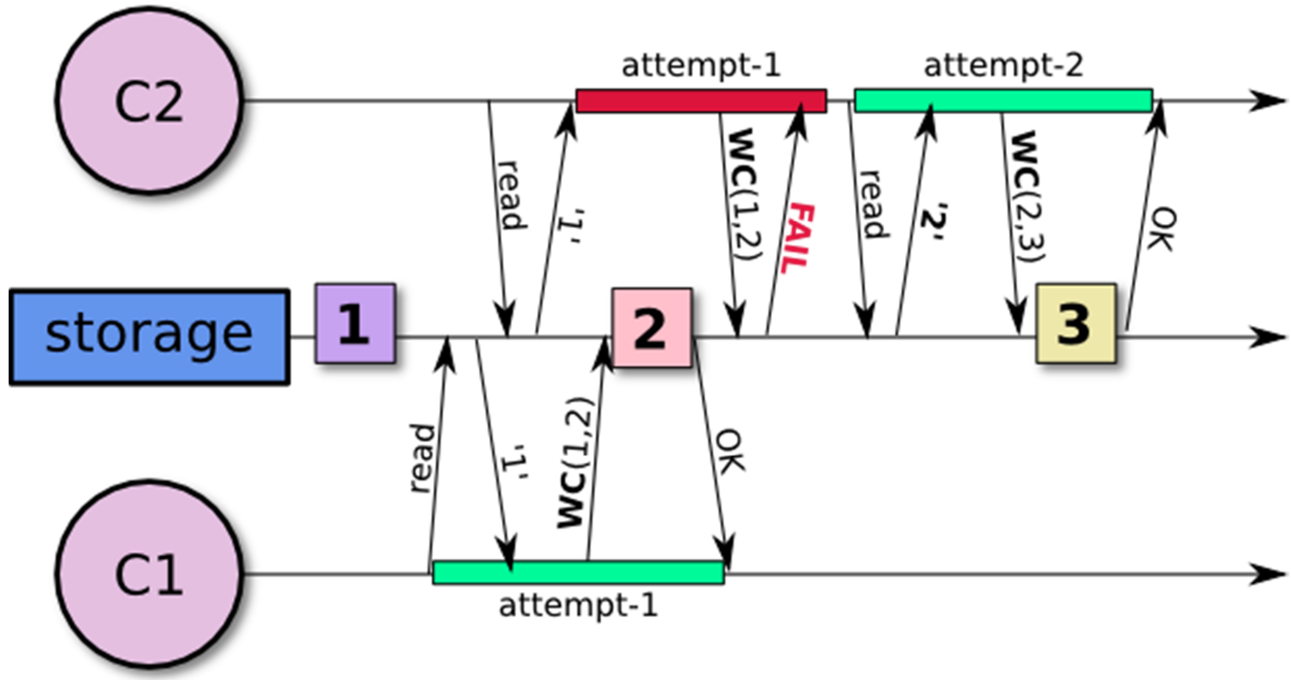

**Algorithm:**

- A **lock server** holds lock for each shared entity.
- Before accessing data, the lock needs to be **obtained**.
- Other lock requests will be **delayed** until the lock is released.
- Lock is **released** after update.

**Disadvantages:**

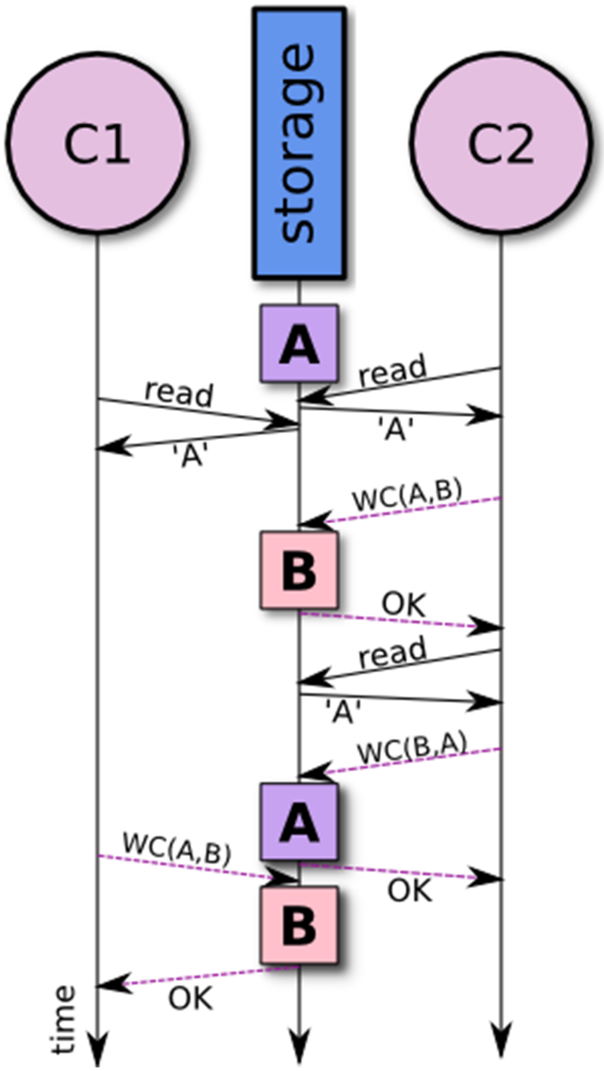
- Lock server holds **state**;
  What if client fails while holding lock?
- Extra lock servers/**infrastructure**
- **Pessimistic**: Always extra cost, even if no locking would have been needed.
- Lock **granularity?**

# A (Better?) Solution: Optimistic Coordination




- Instead of **write**: *Write-Conditional (expvalue, newvalue)*
  - Read current value, and only write newvalue if current value equal to expvalue
  - **Atomic operation**; Write and write-conditional to same location are serialized
- **Optimistic:** Expect no problems (conflicting access)
  - Repeat algorithm if assumption was incorrect
- **No state** on server; OK if client disappears or otherwise misbehaves

# The A-B-A Issue: “value is the same” vs “update”




The comparison **cannot detect if the data was updated**; it can only detect if it is **different** from what the client is **expecting (previous value read).**

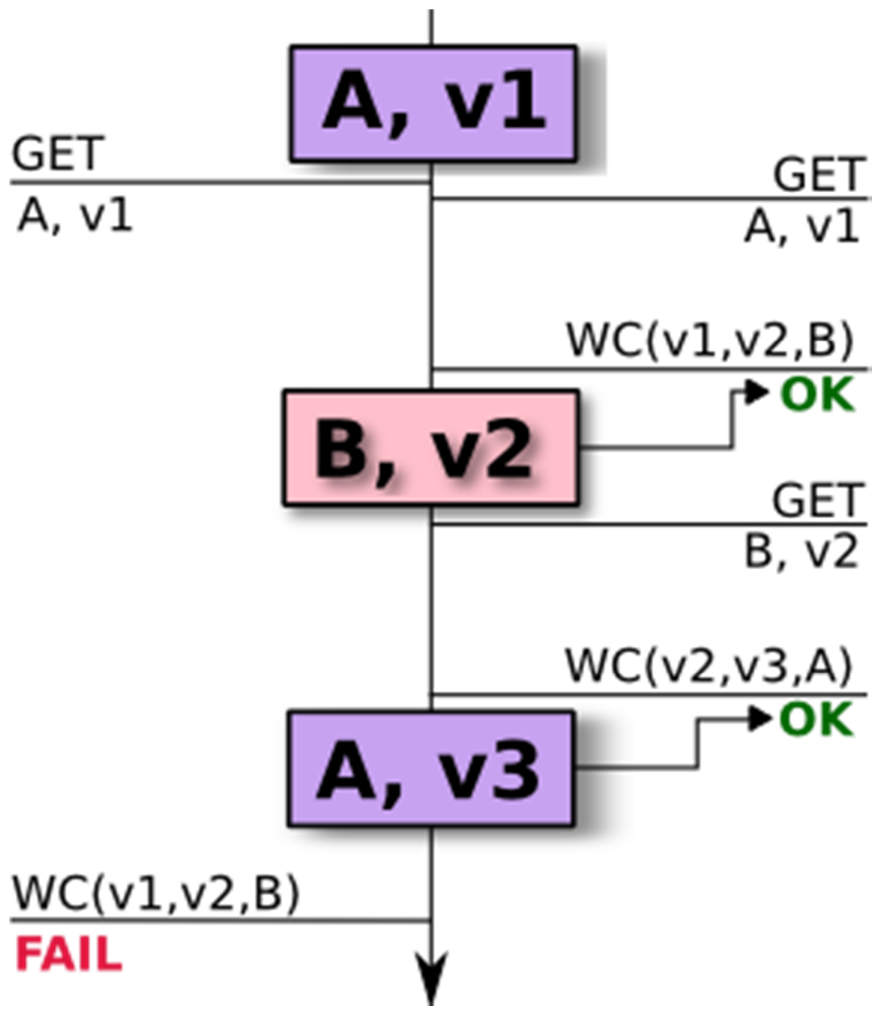
**Example:**

- C1 reads the current value and finds **‘A’**.
- C2 reads the current value and finds **‘A’**.
- They both want to update to **‘B’** and proceed.
  - C2 goes first, succeeds in updating to **‘B’** but performs another update reverting back to **‘A’**.
  - C1 performs update changing to **‘B’**.

C1 **cannot assume** that the storage **did not change**!

- Only that **contents** are the same as when it last checked.
- Problematic: for example if storing **references**

# A Solution for the A-B-A Problem



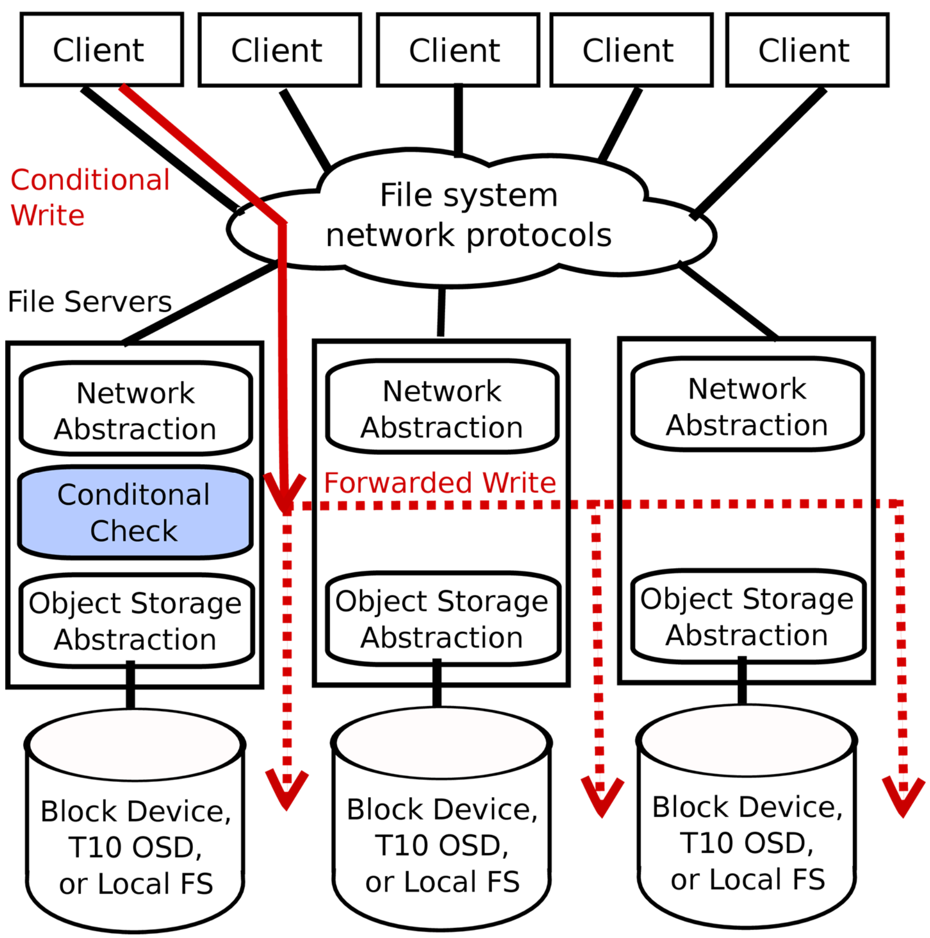

Solution is to track **change** independent from **content**.

- **Add version field** which will be used for comparison.
  (No longer comparing data!)
- **Read** and **write** include version in addition to data.
- Advantages:
  - Solves A-B-A issue
  - Comparing version can be quicker, (comparison and transfer) especially for large accesses
- Disadvantages:
  - Multiple attempts needed; Fairness

Implementation details:

- Byte? Block? Extent?
- Multiple versions?

# Updates in Replicated Storage




### How to handle replicated storage?

- Conditional write could succeed multiple times with different outcome and lead to non-deterministic result!

### Solution:

- Force all updates to go through a single 'master' server
  - Condition check only performed once
  - Natural serialization point
  - Master server can be different for different objects.

# Evaluation: Experiment Setup

- Implementation builds on earlier work: Transactional Object Storage Device [1]
  - Added version-based conditional operators
  - Byte-granularity atomicity
  - Extent based version tracking.

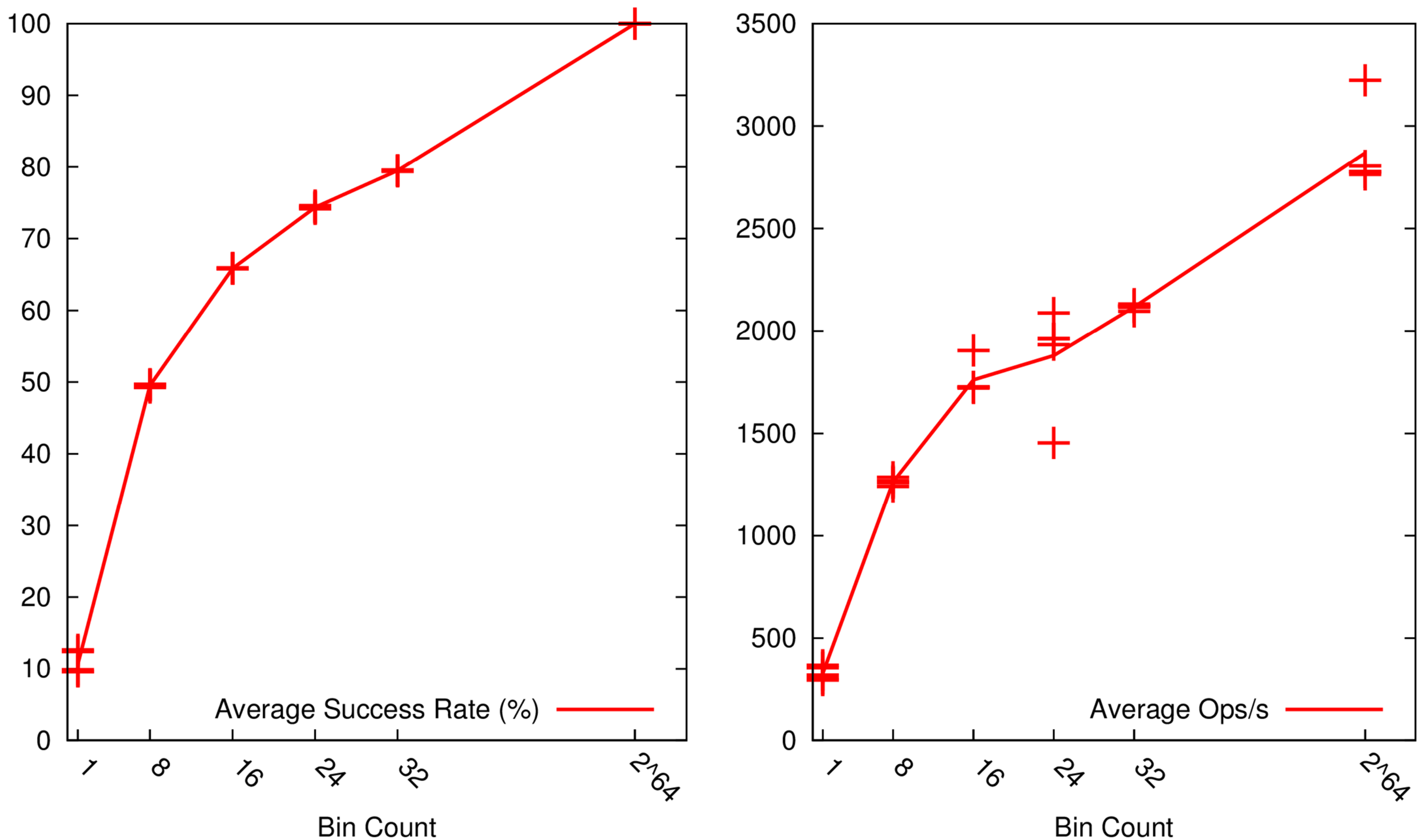
- Using Fusion cluster at Argonne National Laboratory
  - Used ramdisk for storage
  - Node: 2x Intel Nehalem 2.6Ghz, 36GB ram, 16 cores total
  - Communication: mpich 1.2.1

- Comparing lock-based coordination against version-based conditionals in performing **histogram update workload**
  - **ZooKeeper** is used for locking; **Single** lock server, **unique lock for each bin**.
  - Run experiment for at least 60 seconds, at least 5 runs.
  - Each **bin** is **4K** in size, variable number of bins, variable number of clients.
  - Input data considered random, so simply **picking random bin number**.

[1] P. Carns, R. Ross and S. Lang "Object Storage Semantics for Replicated Concurrent-Writer File Systems" (IASDS 2010).

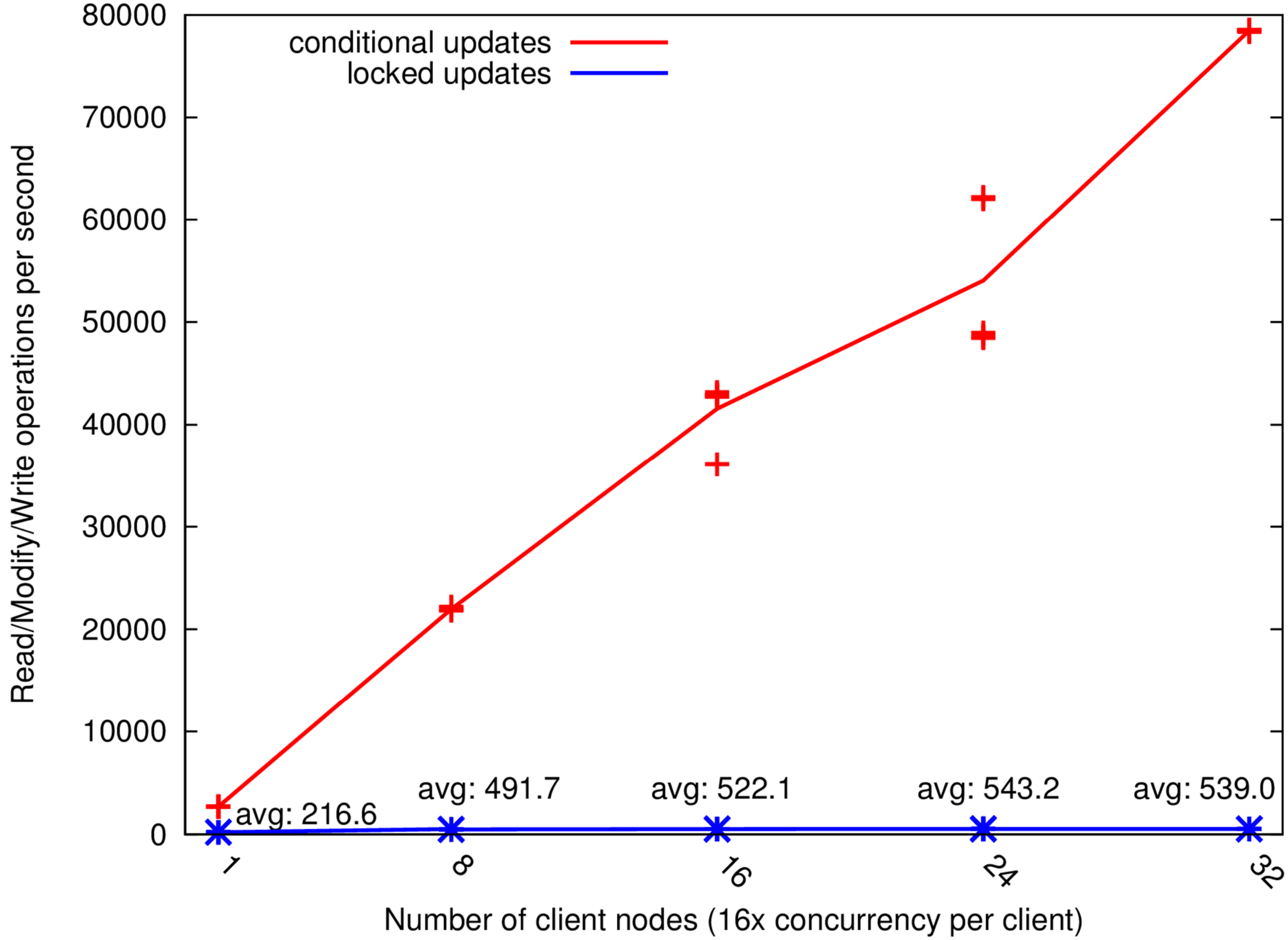
# Effect of Conflicts: Sensitivity Analysis

## 16 Concurrent Operations, Single Client, Variable number of bins

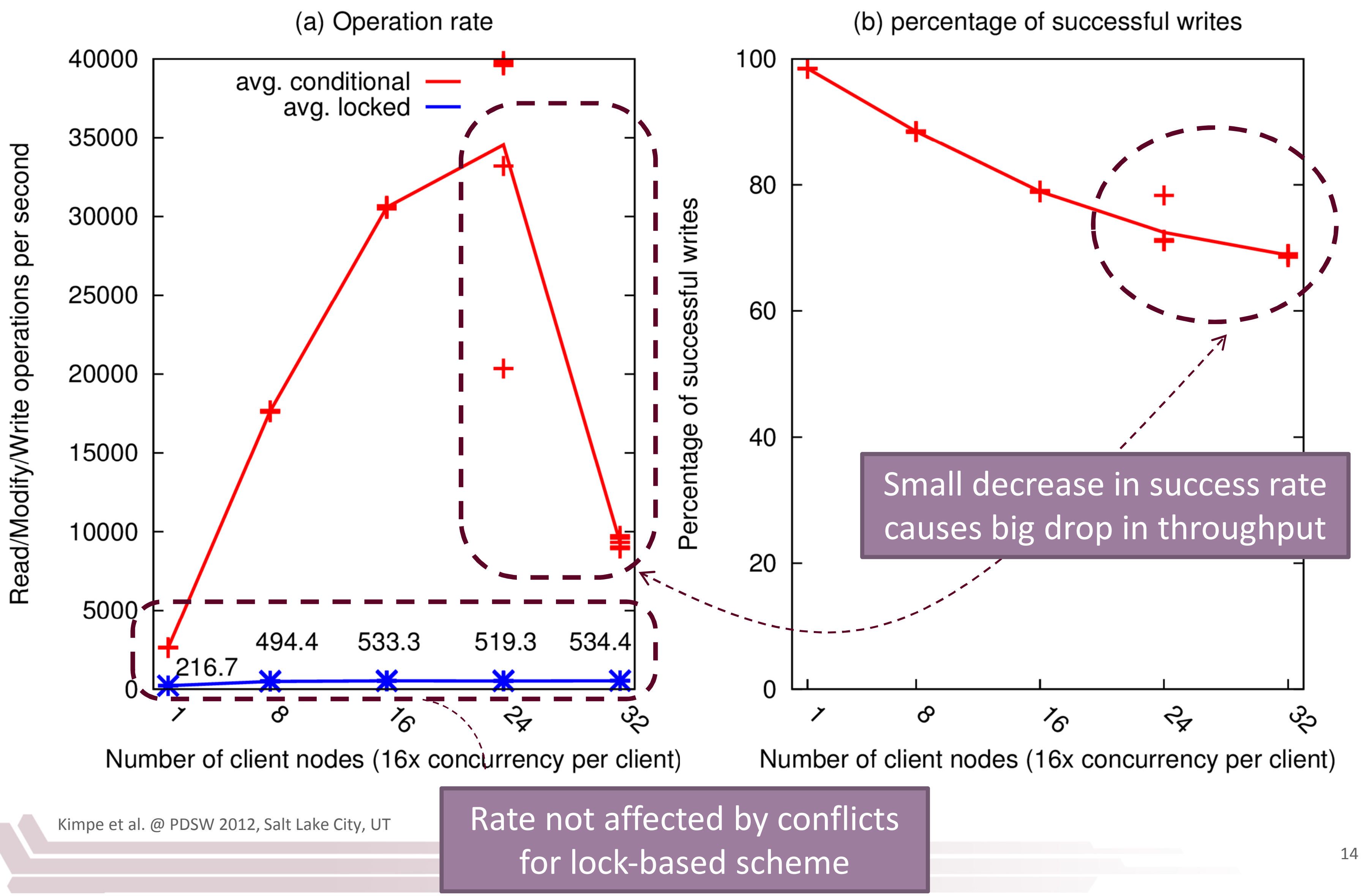


# Scalability

## $2^{64}$ bins, (almost) no conflicts

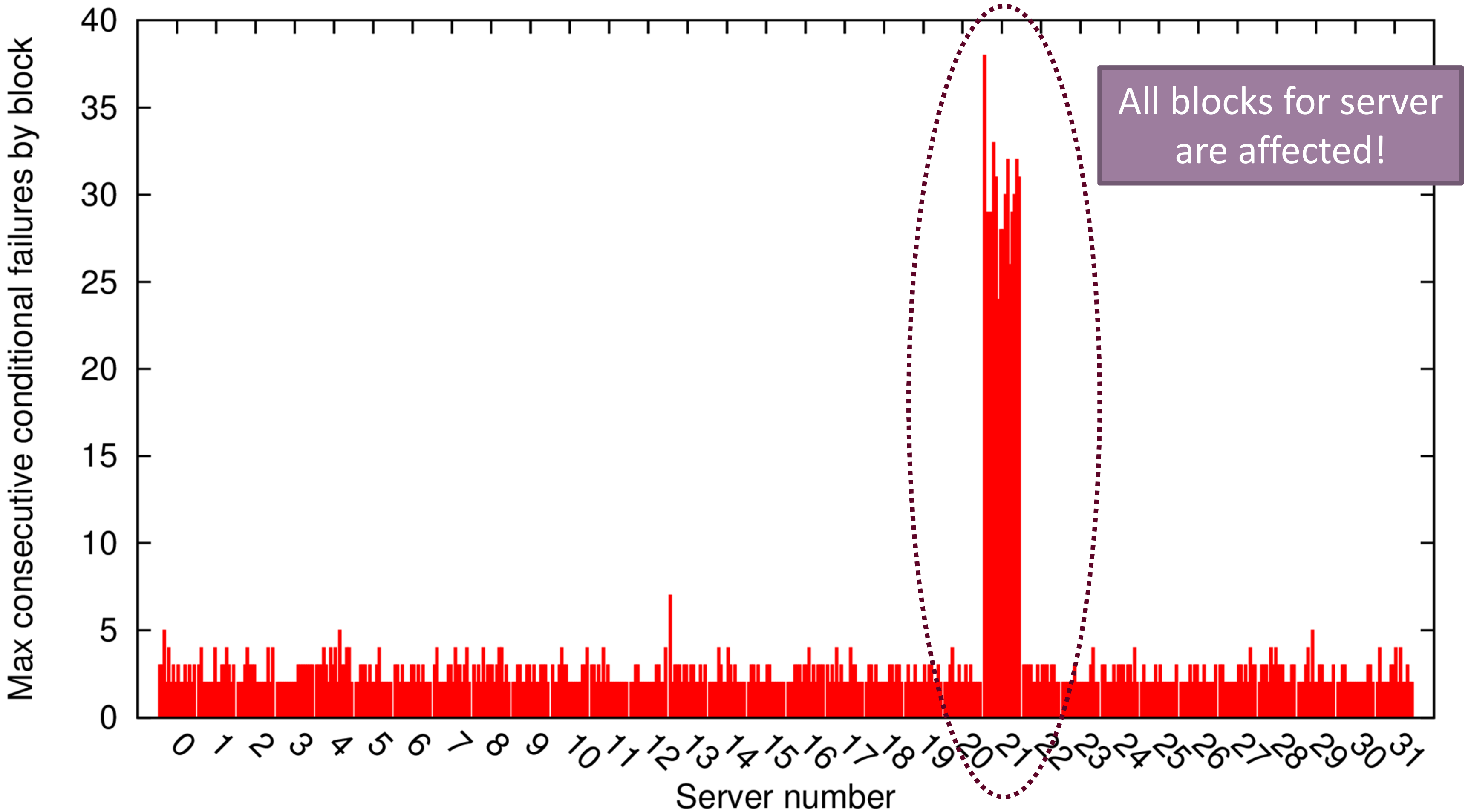


# Scalability
## 512 bins, Conflicts likely



# Conflict Rate Analysis

## Self-Reinforcing Effect: 512 bins, 32 servers



# Conclusions

- Studied Optimistic Coordination in the context of High-Performance Computing Storage Systems.

- Evaluated by comparing to traditional, distributed locking (pessimistic) approach.
  - Found nearly linear scaling up to 512 concurrent operations **provided there is little contention.**
    - For high-contention scenario's, some form of throttling is needed. (topic of **future work**)
  - Optimistic locking outperformed traditional locking by a wide margin.

- **Future work:**
  - Explore back-off algorithms to reduce contention: both server and client initiated.
  - Investigate the use of optimistic locking in other common file system workloads
    - Distributed data structures in scientific data analysis
    - Consistency in namespaces (for example directories)

# Acknowledgements


- **Co-authors & Collaborators**

  **Philip Carns, Kevin Harms, Justin M. Wozniak, Robert Ross**: Argonne National Laboratory; **Lee Ward, Matthew Curry, Ruth Klundt, Geoff Danielson**: Sandia National Laboratories; **Cengiz Karakoyunlu, John Chandy**: University of Connecticut**; Bradley Settlemeyer:** Oak Ridge National Laboratory; **William Gropp:** University of Illinois at Urbana-Champaign

- **Sponsor**

  This material is based on work supported by, or in part by U.S. Department of Energy's Oak Ridge National Laboratory and included the Extreme Scale Systems Center, located at ORNL and funded by the DoD in part by the "Novel Software Storage Architectures" contract. This work also was supported by U.S. Department of Energy, under contracts DE-AC02-06CH11357 and DE-FG02-08ER25835.

- The use of the computing resources provided on "Fusion", operated by the Laboratory Computing Resource Center at Argonne National Laboratory.


Thank you for your attention!

Questions?