Astronomical Image Processing with Hadoop

Keith Wiley*

Survey Science Group

Dept. of Astronomy, Univ. of Washington

* Keith Wiley, Andrew Connolly, YongChul Kwon, Magdalena Balazinska, Bill Howe, Jeffrey Garder, Simon Krughoff, Yingyi Bu, Sarah Loebman and Matthew Kraus

Acknowledgments

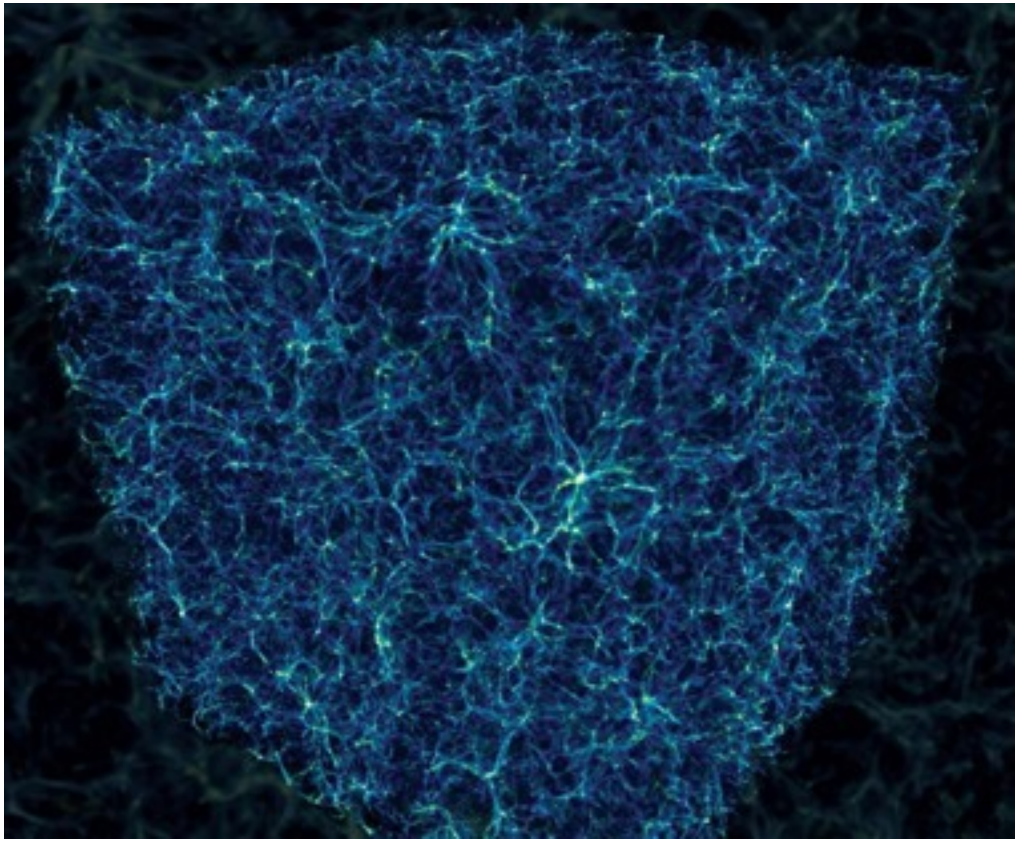
- This work is supported by the:
  - NSF Cluster Exploratory (CluE) grant IIS-0844580.
  - NASA grant 08-AISR08-0081.

Session Agenda

- Astronomical Survey Science
- Image Coaddition
- Implementing Coaddition within MapReduce
- Optimizing the Coaddition Process
- Conclusions
- Future Work

Astronomical Topics of Study

- Dark energy
- Large scale structure of universe
- Gravitational lensing
- Asteroid detection/tracking



What is Astronomical Survey Science?

- Dedicated sky surveys,
  usually from a single calibrated telescope/camera pair.

- Run for years at a time.

- Gather millions of images and TBs of storage*.

- Require high-throughput data reduction pipelines.

- Require sophisticated off-line data analysis tools.

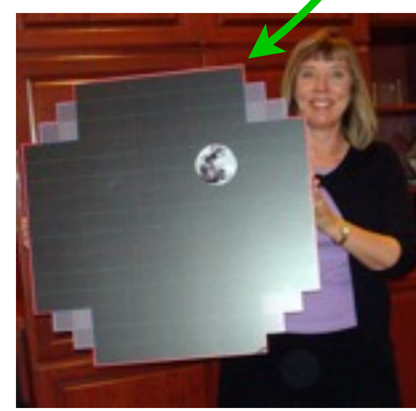
* Next generation surveys will gather PBs of image data.

Sky Surveys: Today and Tomorrow

- ***SDSS**** (1999-2005)
- Founded in part by UW
- 1/4 of the sky
- 80TBs total data

- ***LSST***† (2015-2025)
- 8.4m mirror, 3.2 gigapixel camera
- Half sky every three nights
- 30TB per night...
  ...one *SDSS* every three nights
- 60PBs total (nonstop ten years)
- 1000s of exposures of each location

That's a person!

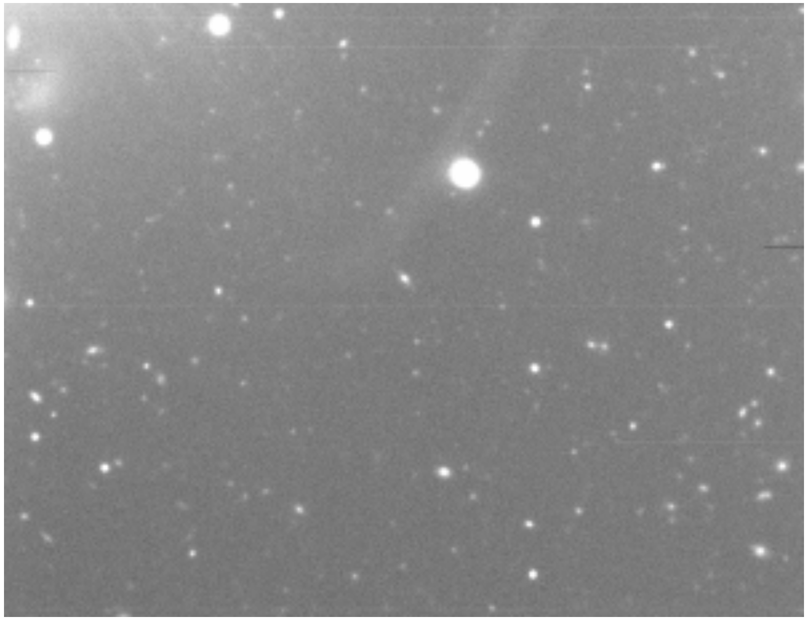


* Sloan Digital Sky Survey
† Large Synoptic Survey Telescope

FITS (Flexible Image Transport System)

- Common astronomical image representation file format
- Metadata tags (like *EXIF*):
  - **Most importantly:** Precise astrometry*
  - Other:
    - Geolocation (telescope location)
    - Sky conditions, image quality, etc.

- ***Bottom line***:
  - **An image format that knows where it is looking.**



* Position on sky

Image Coaddition

- Give multiple partially overlapping images and a ***query*** (color and sky bounds):
  - › Find images' intersections with the query bounds.
  - › **Project** bitmaps to the bounds.
  - › **Stack** and **mosaic** into a final product.

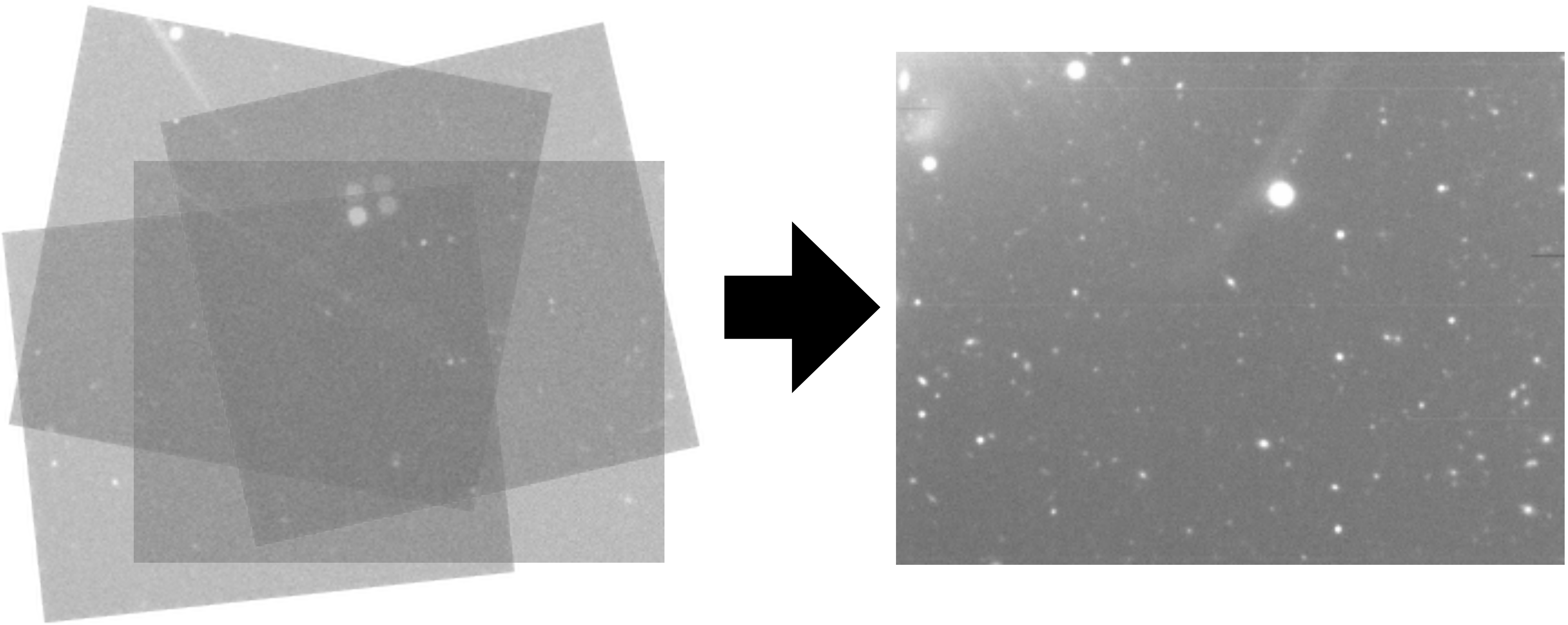
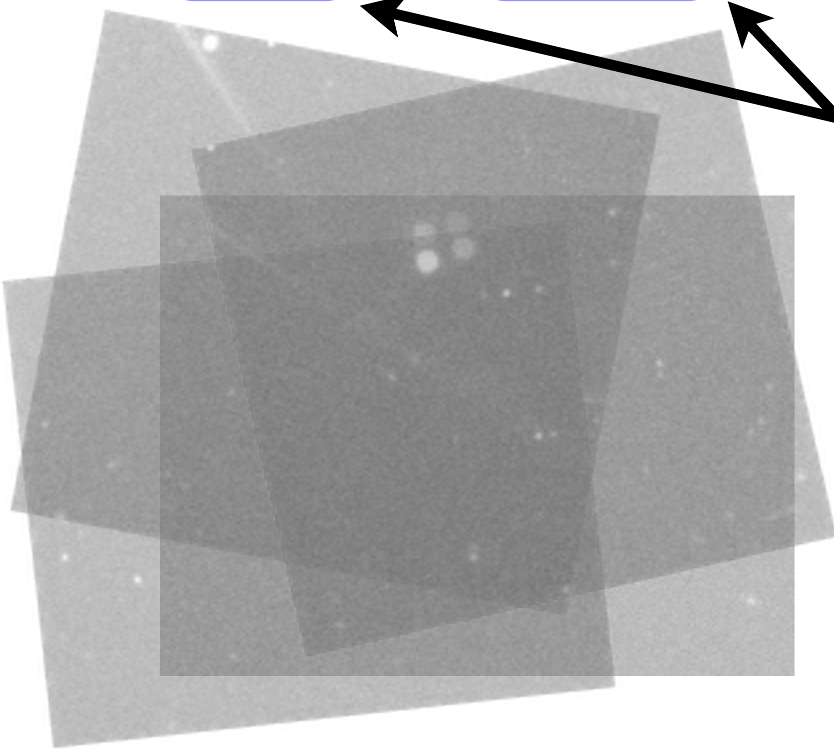


Image Coaddition

Expensive

- Give multiple partially overlapping images and a ***query*** (color and sky bounds):
  - › Find images' intersections with the query bounds.
  - › **Project** bitmaps to the bounds.
  - › **Stack** and **mosaic** into a final product.

Cheap



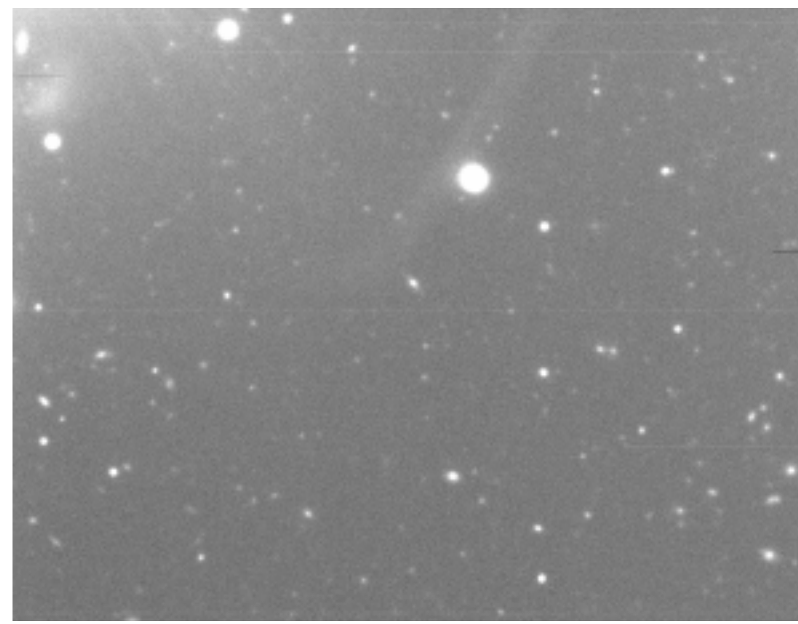
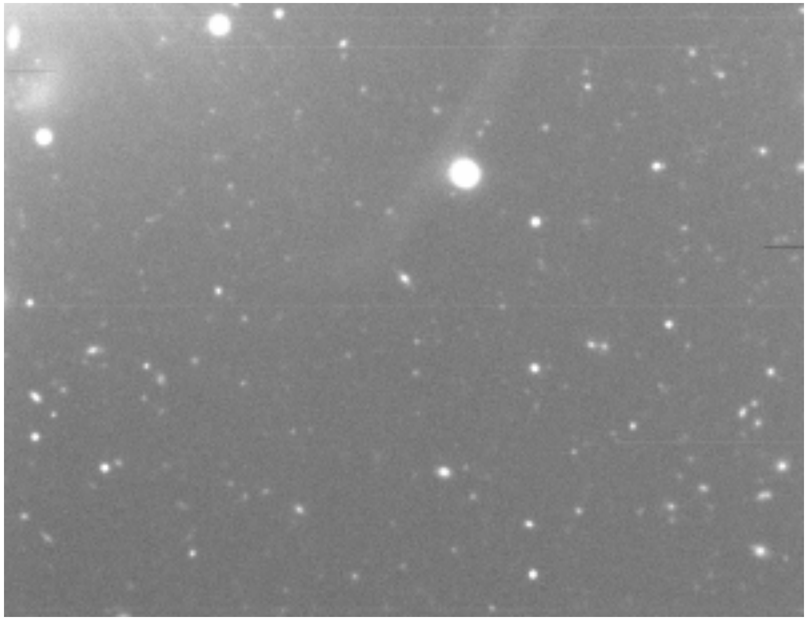


Image Stacking (Signal Averaging)



- **Stacking** improves SNR:
  - Makes **fainter objects visible.**
- Example (*SDSS*, Stripe 82):
  - **Top**: Single image, R-band
  - **Bottom**: 79-deep stack
    - (~9x SNR improvement)
    - Numerous additional detections
- Variable conditions (*e.g.*, atmosphere, PSF, haze) mean stacking algorithm complexity can exceed a mere sum.

Existing Image Coaddition Systems

- *SWarp*
  - Multi-threaded parallelism (single machine only).
- SDSS coadds of Stripe 82 (Fermilab)
  - Same dataset used in our work.
  - One-off project – not a general-purpose tool.
- *Montage* (run on *TeraGrid*)
  - Most similar to our work.
  - MPI (complicated), TeraGrid (dedicated, expensive).

- *MapReduce* (our work, this talk)
  - High-level, potentially simpler to program.
  - Scalable on cheap commodity hardware (the *cloud*).

Advantages of MapReduce (Hadoop)

- High-level problem description. No effort spent on inter-node communication, message-passing, etc.

- Programmed in Java (accessible to most science researchers, not just computer scientists and engineers).

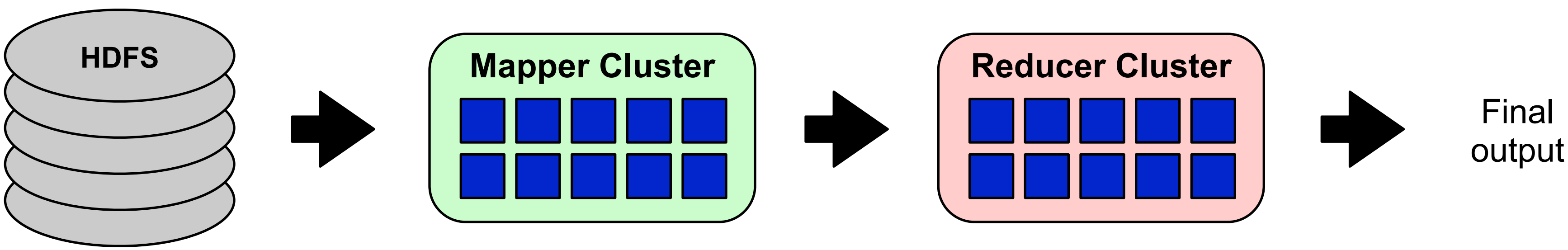
- Runs on cheap commodity hardware, potentially in the *cloud*, *e.g.*, Amazon's EC2.

- Scalable: 1000s of nodes can be added to the cluster with no modification to the researcher's software.

- Large community of users/support.

Hadoop

- A massively parallel database-processing system:
  - In one sense: a parallel computing system (a cluster)
  - In another sense: a parallel database
  - It’s both!



Data stored on HDFS (*Hadoop Distributed File System*)

100s of computers in a cluster

Data processed by numerous parallel Mappers

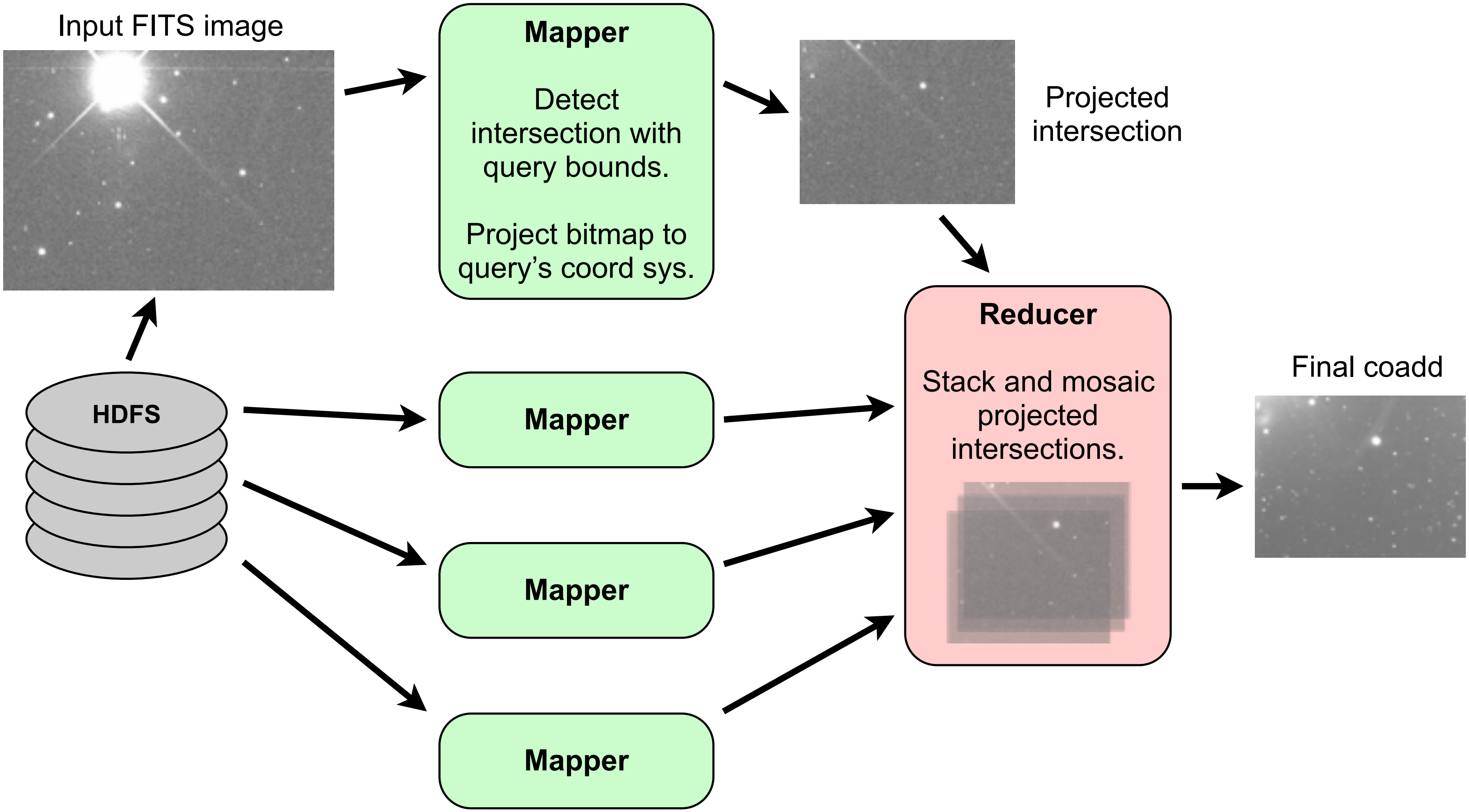
(programs on the HDFS computers)

Data further processed by numerous parallel Reducers

(programs on the HDFS computers)

Hadoop job done, data fully processed

Coaddition in Hadoop



Parallel by image

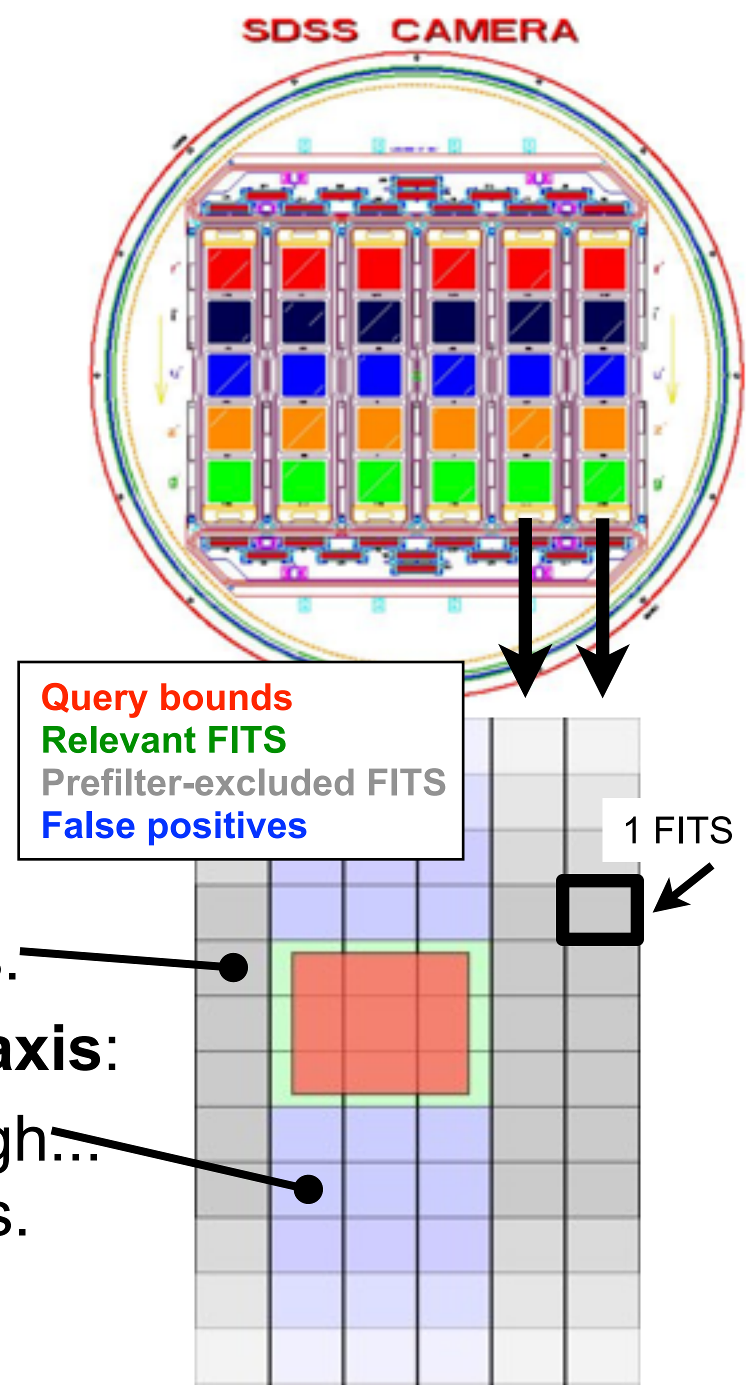
Parallel by query

Driver Prefiltering

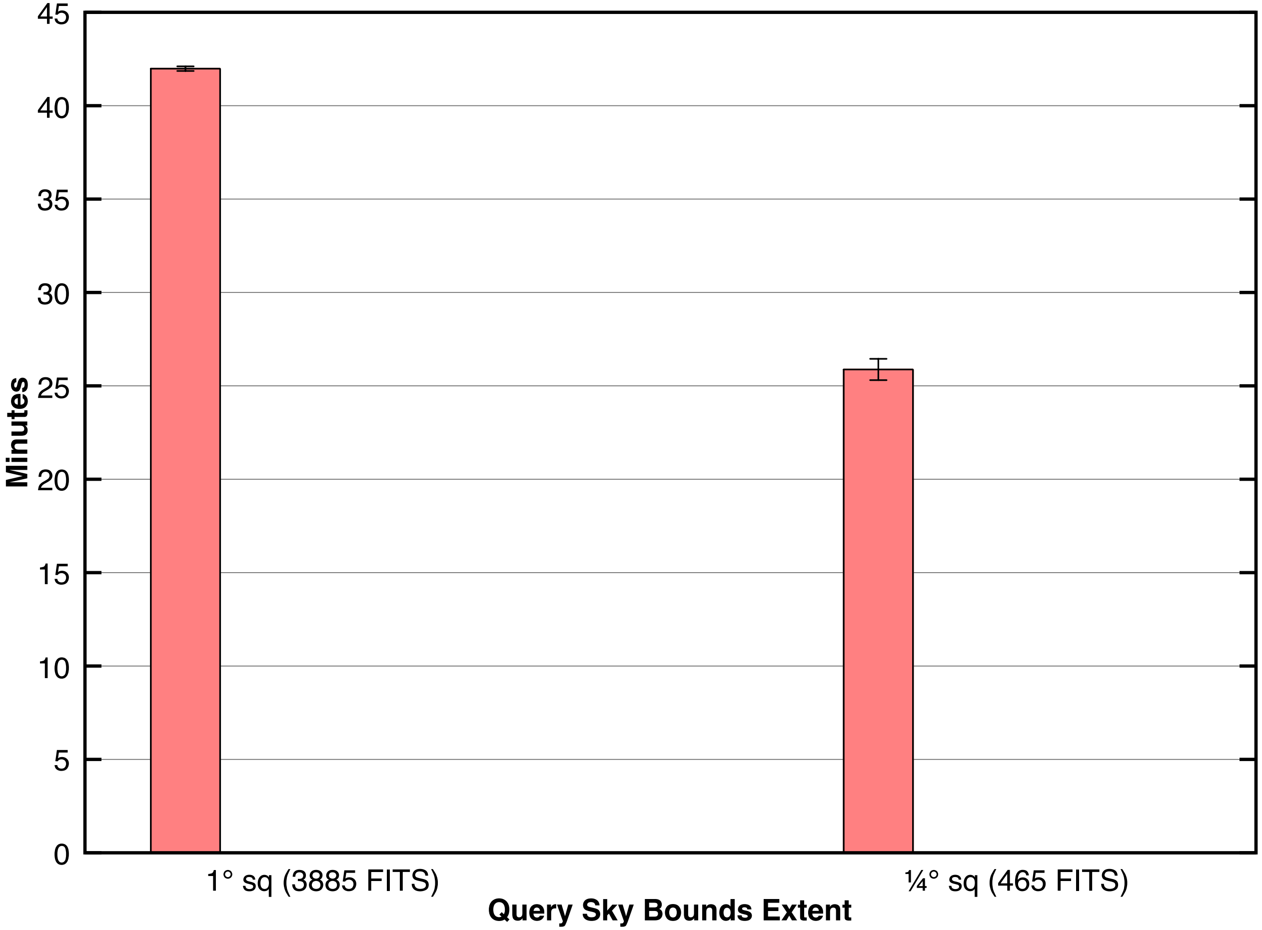
- To assist the process we prefilter the FITS files in the driver.

- *SDSS* camera has 30 CCDs:
  - 5 colors
  - 6 abutting strips of sky

- **Prefilter (path glob) by color and sky coverage** (single axis):
  - Exclude many irrelevant FITS files.
  - Sky coverage filter is only **single axis**:
    - Thus, **false positives** slip through...
      ...to be discarded in the mappers.



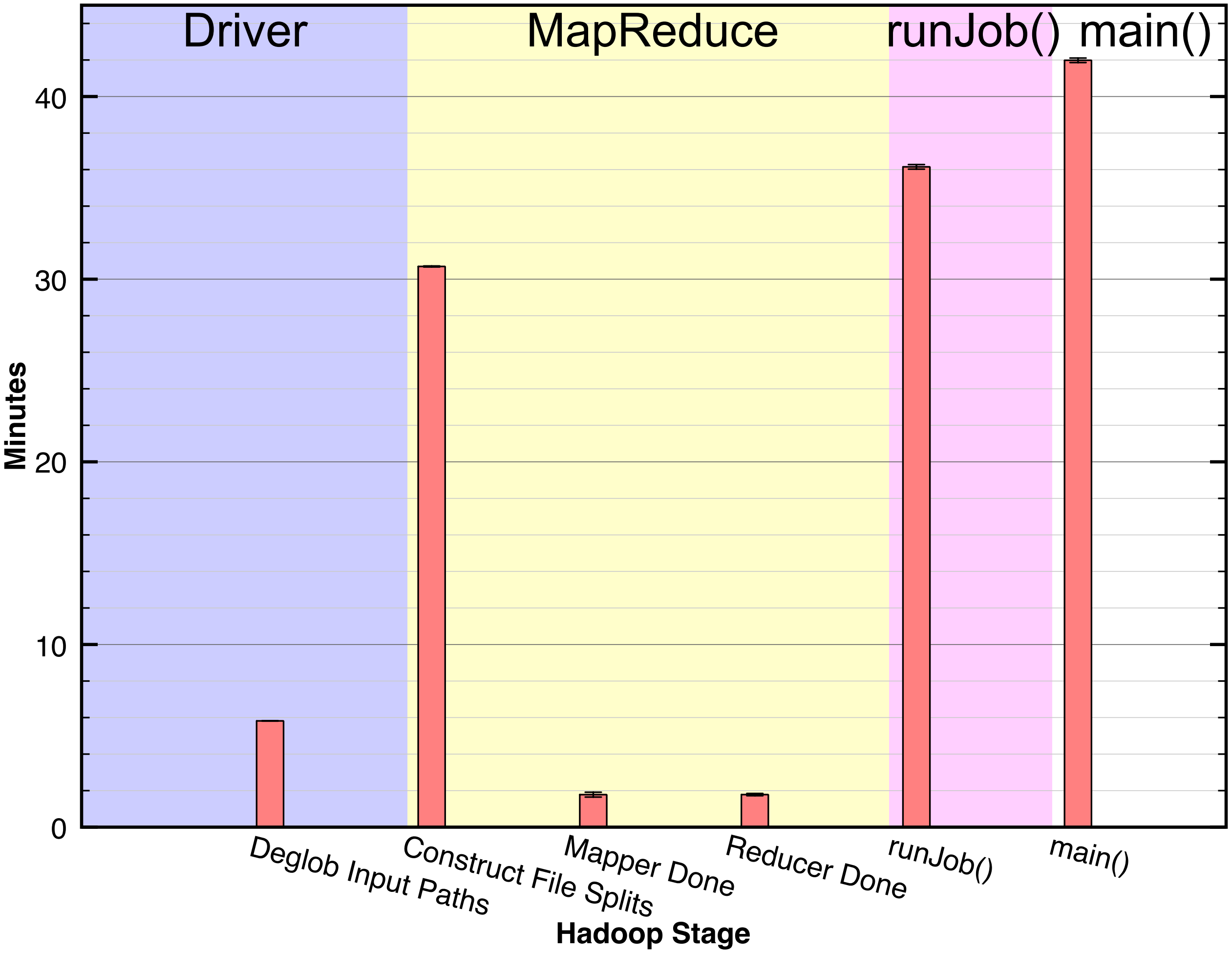
Performance Analysis



Error bars show 95% confidence intervals — Outliers removed via Chauvenet

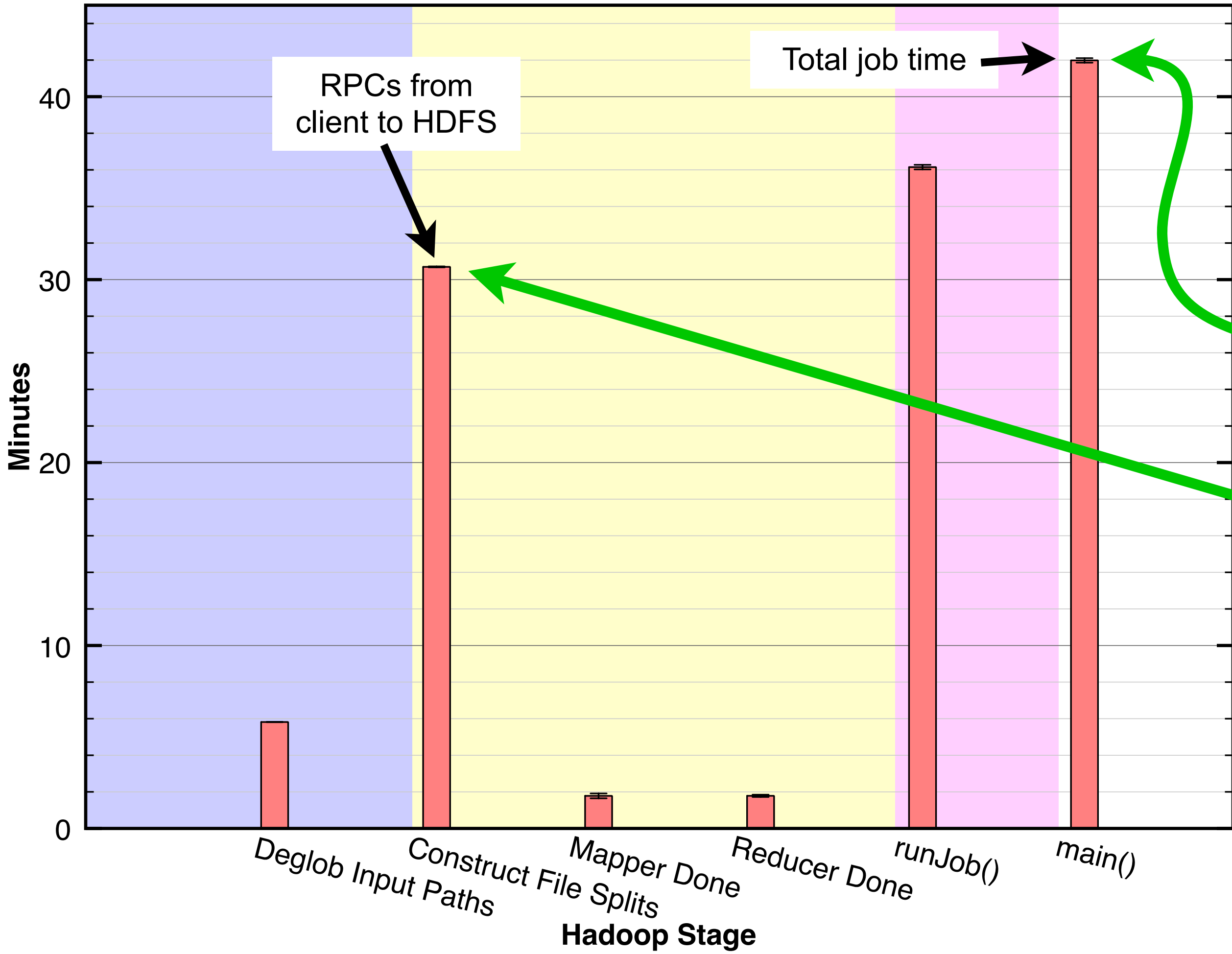
- Running time:
  - 2 query sizes
  - Run against 1/10th of *SDSS* (100,058 FITS)
- **Conclusion:**
  - Considering the small dataset, this is too slow!
  - Remember 42 minutes for the next slide.

Performance Analysis



- Breakdown of large query running time
- **main()** is sum of:
  - **Driver**
  - **runJob()**
- **runJob()** is sum of **MapReduce** parts.

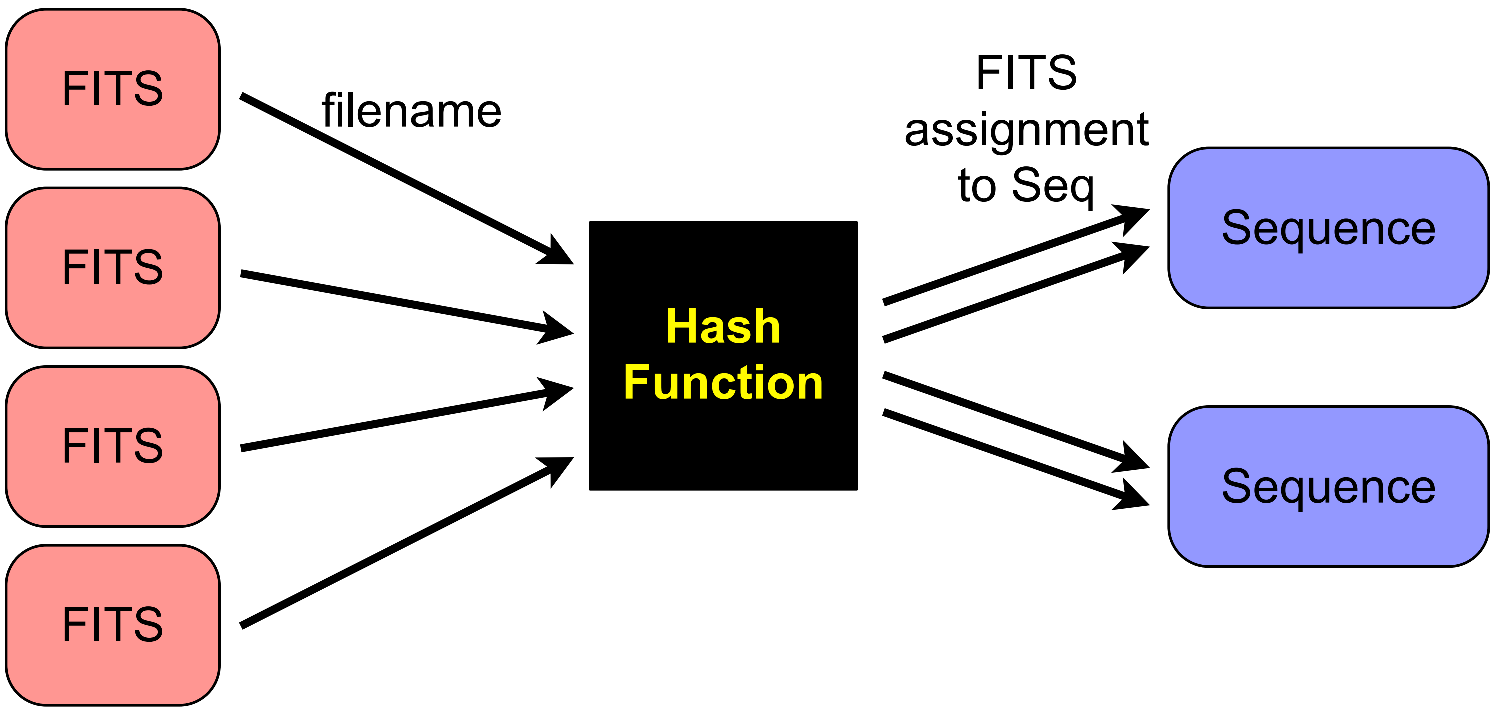
Performance Analysis



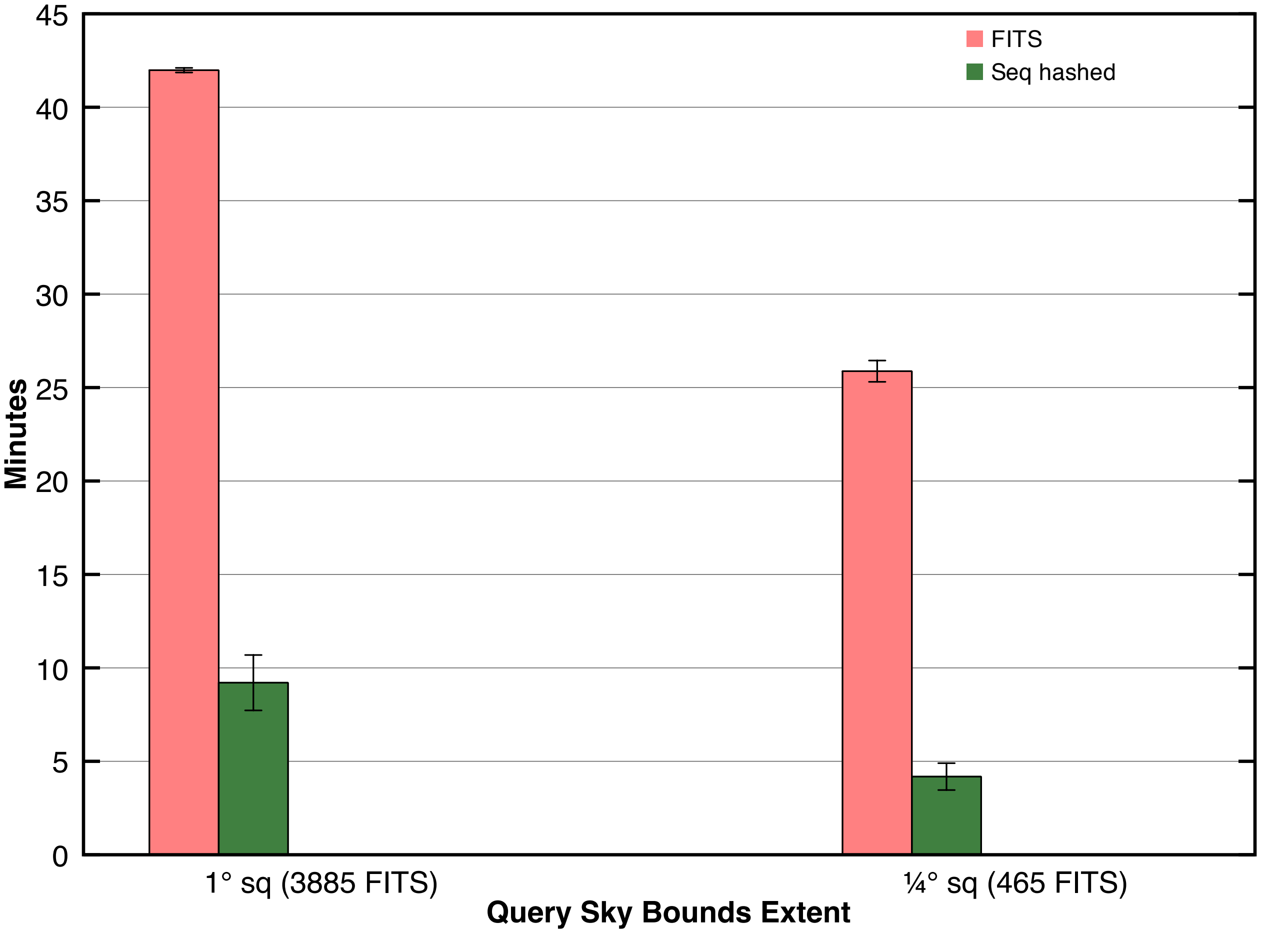
- Breakdown of large query running time
- **Observation:**
  - **Running time** dominated by **RPCs** from client to HDFS to process 1000s of FITS file paths.
- **Conclusion:**
  - **Need to reduce number of files.**

Sequence Files

- Sequence files group many small files into a few large files.
- ***Just what we need!***
- Real-time images may not be amenable to logical grouping.
  - Therefore, sequence files filled in an arbitrary manner:



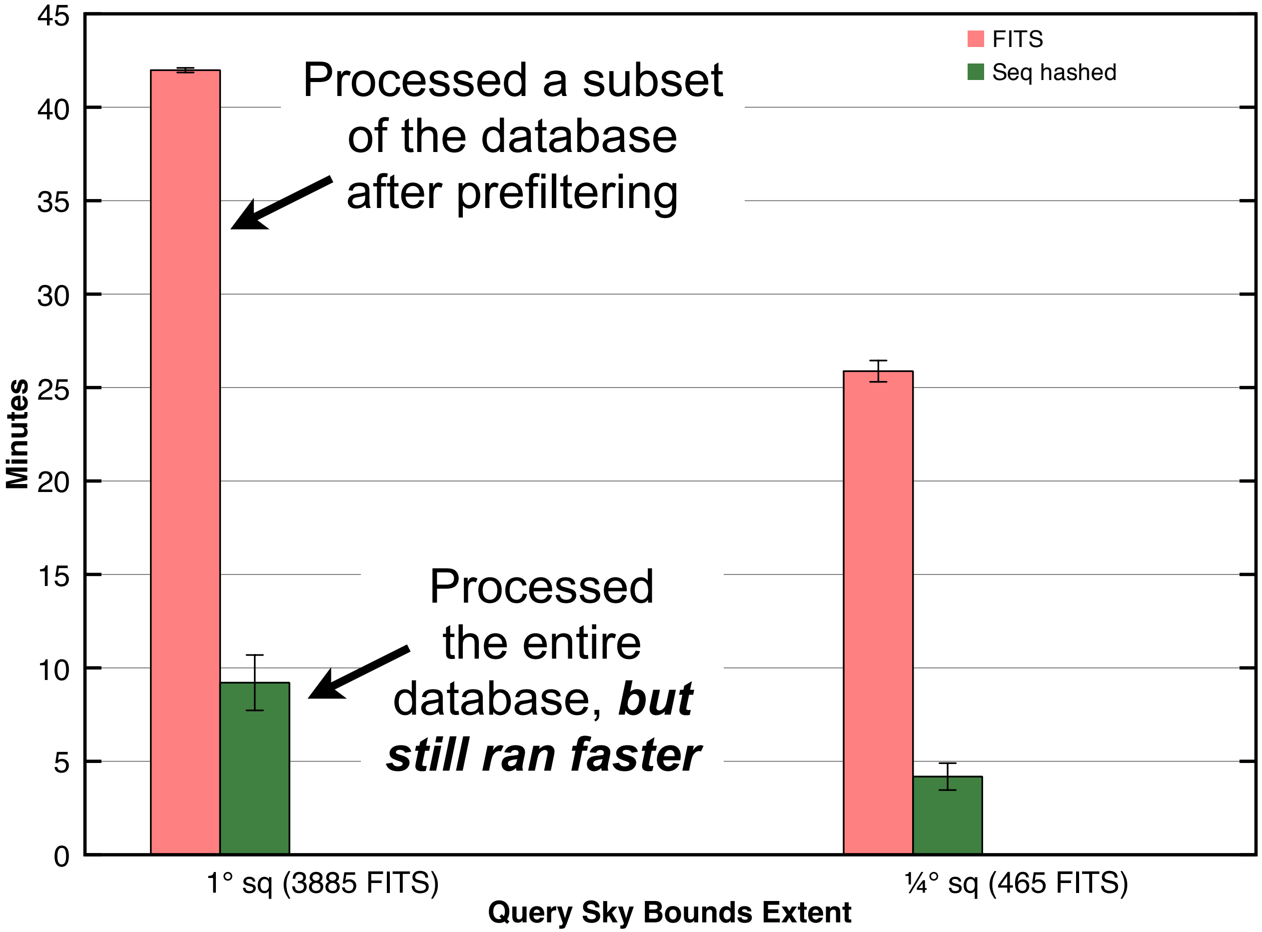
Performance Analysis



- **Comparison:**
  - FITS input vs. unstructured sequence file input*
- **Conclusion:**
  - *5x speedup!*
- Hmmm...
  Can we do better?

* 360 seq files in hashed seq DB.

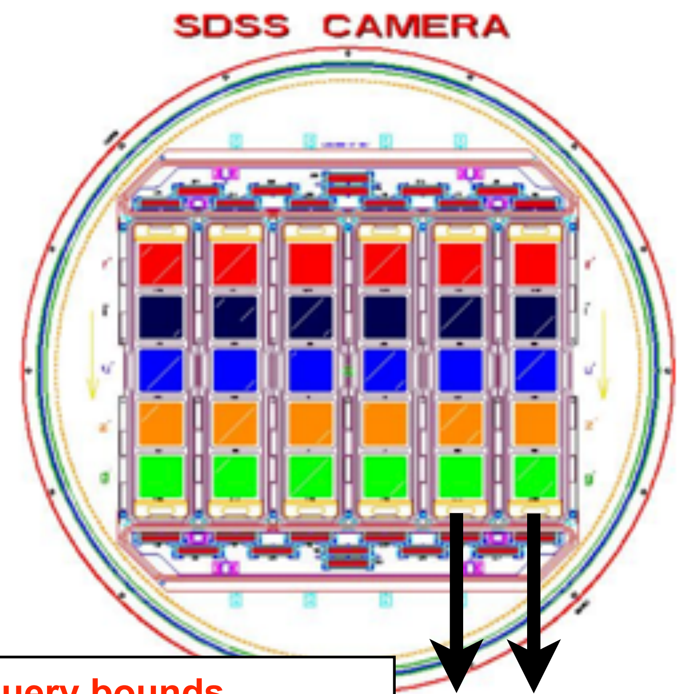
Performance Analysis



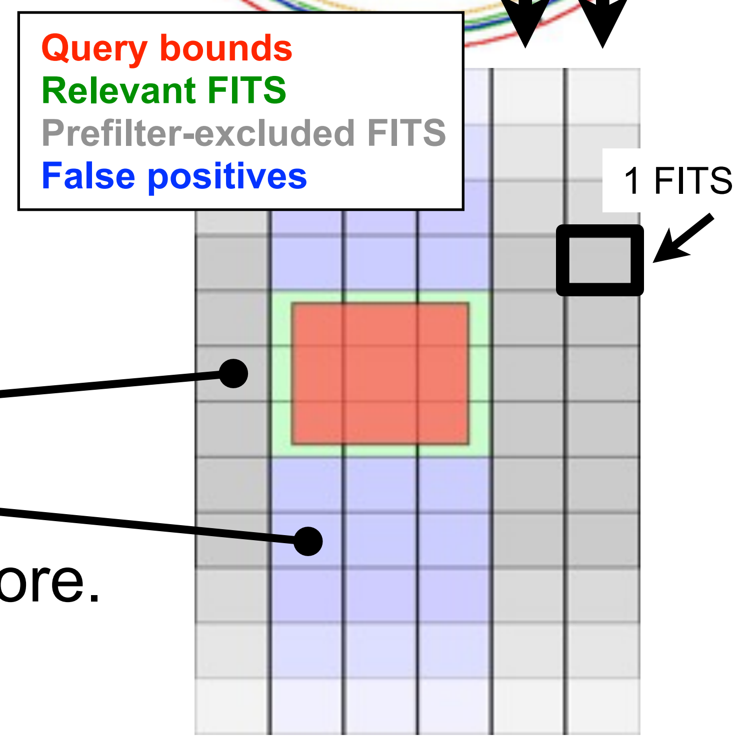
- **Comparison:**
  - FITS input vs. unstructured sequence file input*
- **Conclusion:**
  - *5x speedup!*
- Hmmm...
  Can we do better?

* 360 seq files in hashed seq DB.

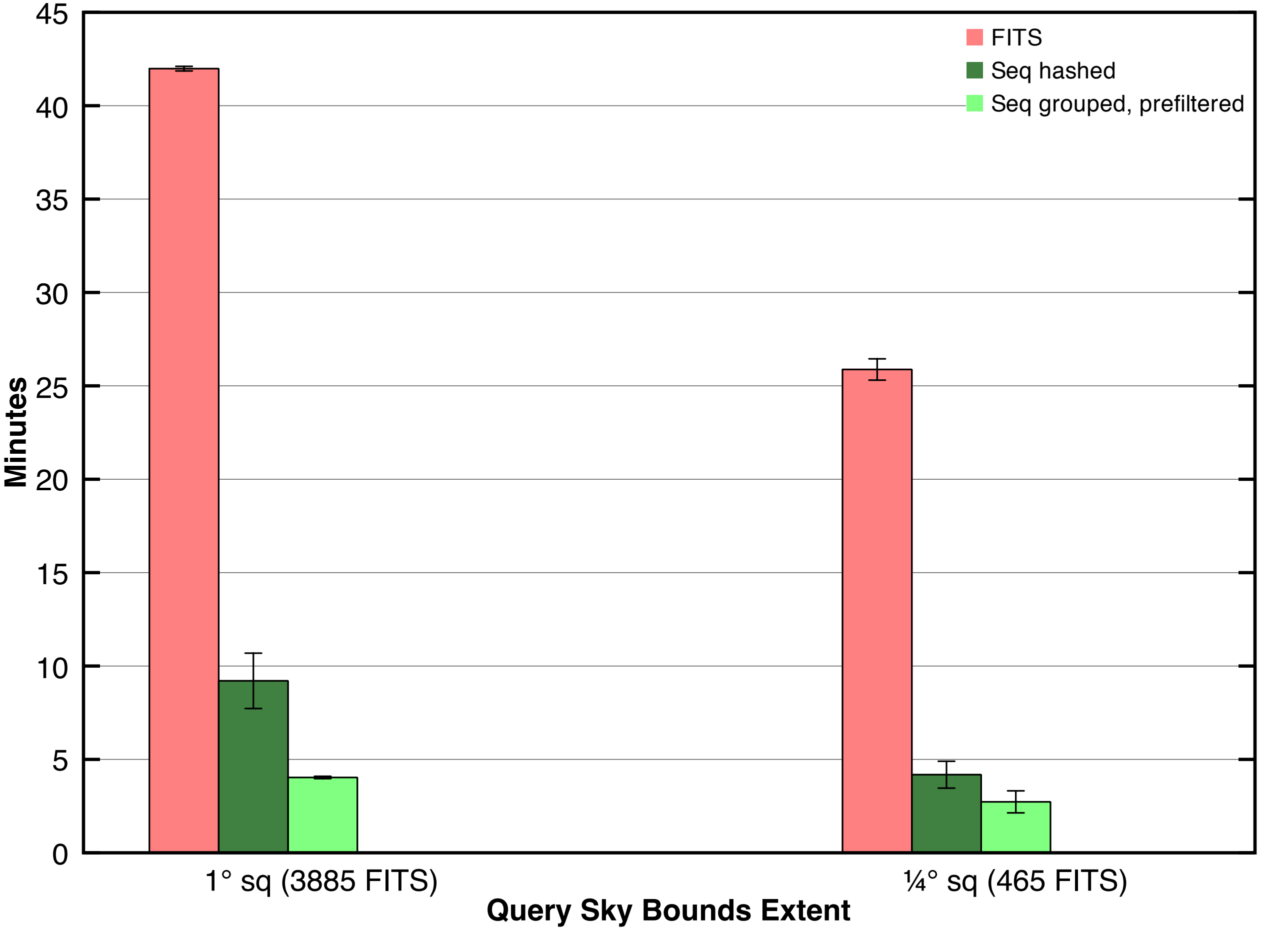
Structured Sequence Files



- Similar to the way we prefiltered FITS files...
- *SDSS* camera has 30 CCDs:
  - 5 colors
  - 6 abutting strips of sky
  - **Thus, 30 sequence file types**
- Prefilter by color and sky coverage (single axis):
  - Exclude irrelevant sequence files.
  - Still have **false positives**.
  - Catch them in the mappers as before.



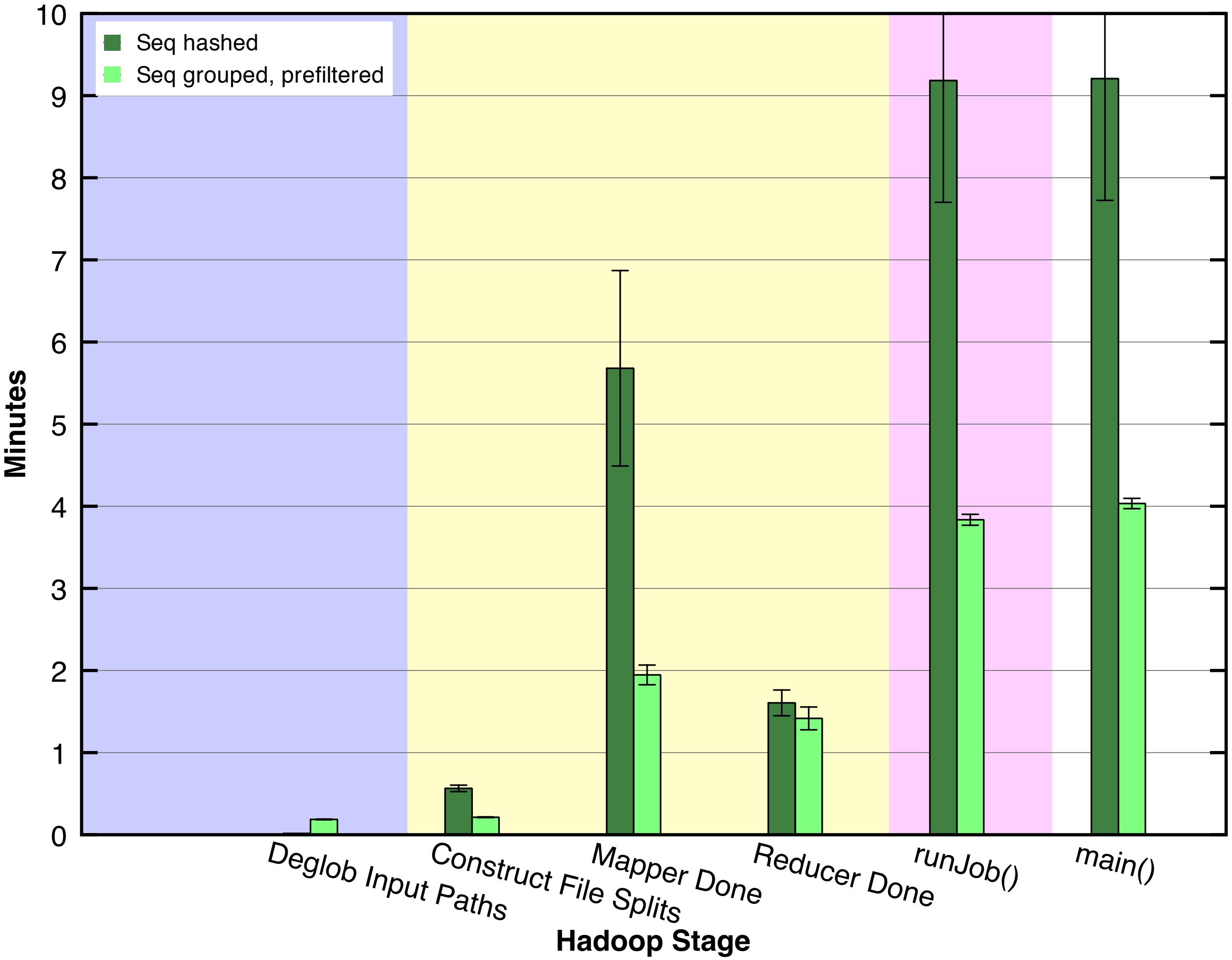
Performance Analysis



- **Comparison:**
  - FITS vs. unstructured sequence* vs. structured sequence files†
- **Conclusion:**
  - Another 2x speedup for the large query, 1.5x speedup for the small query.

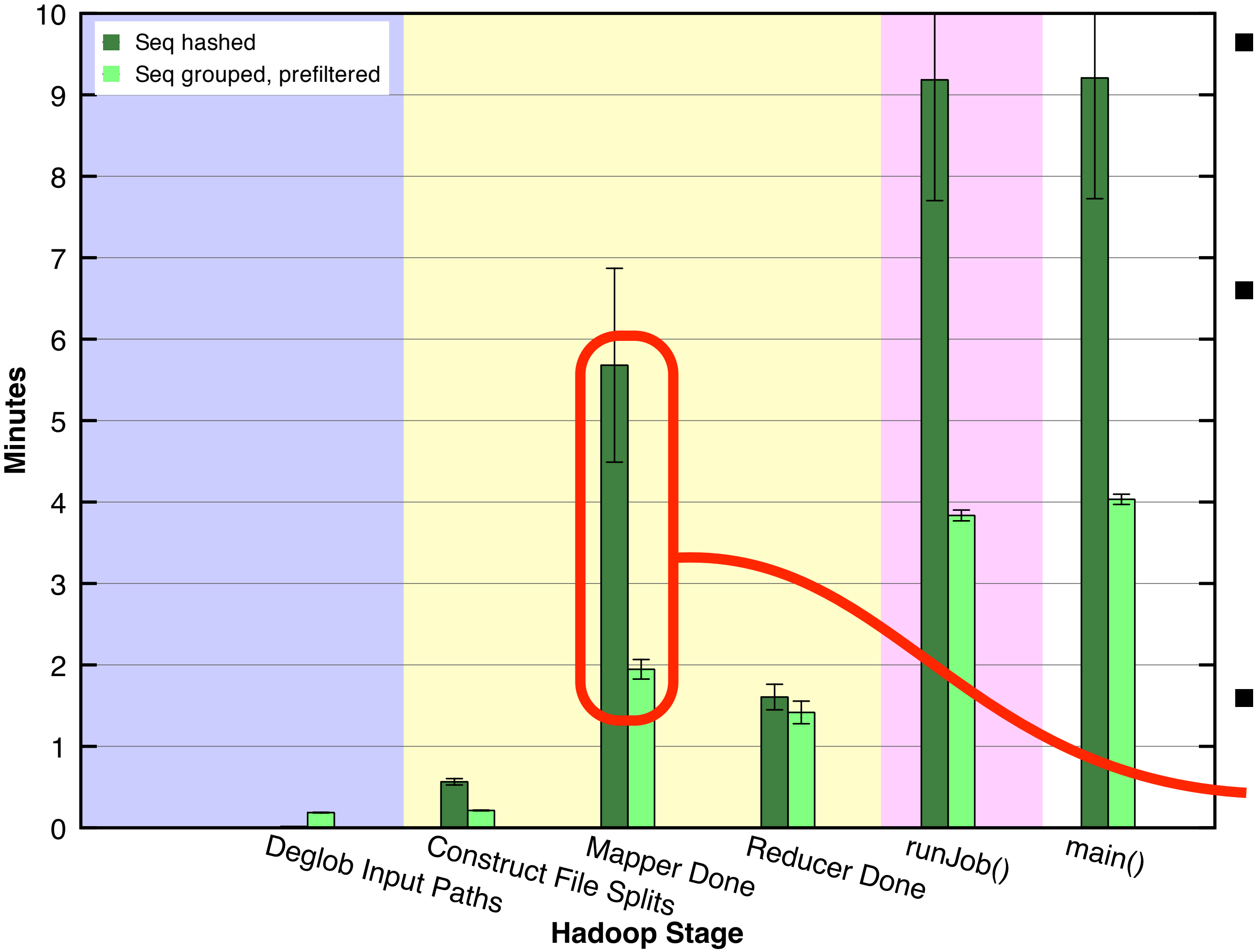
* 360 seq files in hashed seq DB.
† 1080 seq files in structured DB.

Performance Analysis



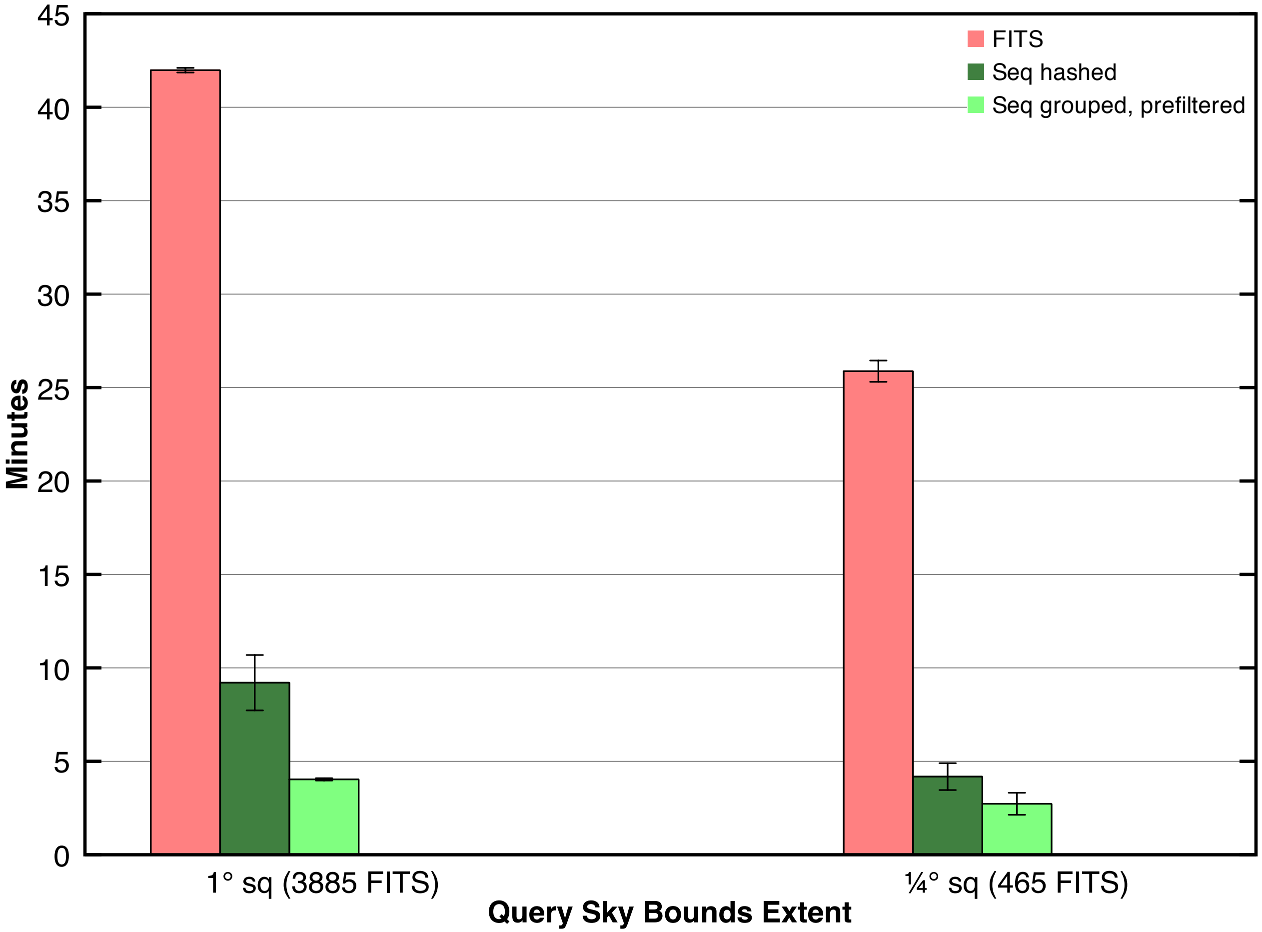
- Breakdown of large query running time
- **Prediction:**
  - Prefiltering should gain performance in the mapper.
- Does it?

Performance Analysis



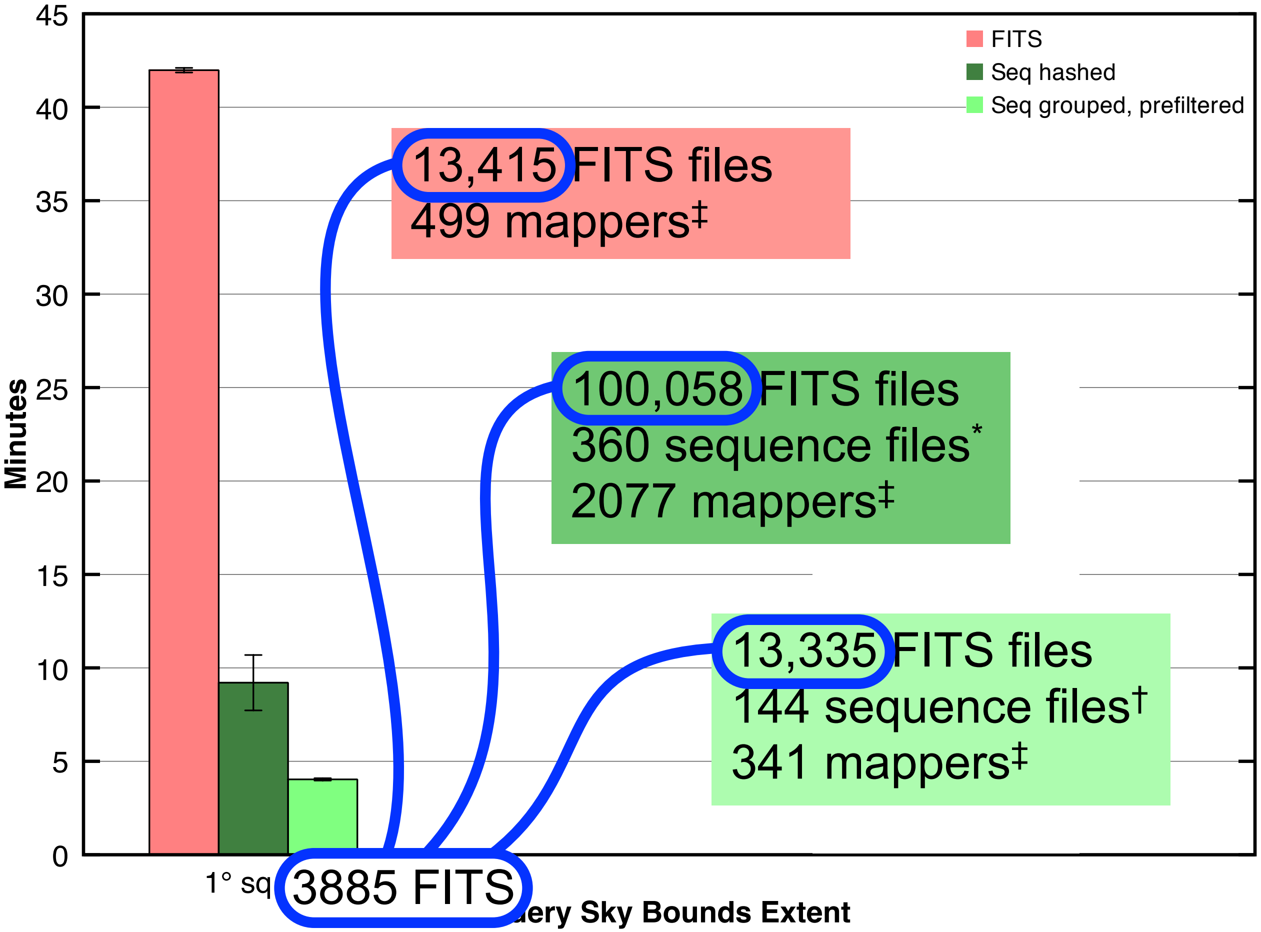
- Breakdown of large query running time
- **Prediction:**
  - Prefiltering should gain performance in the mapper.
- **Conclusion:**
  - Yep, just as expected.

Performance Analysis



- Experiments were performed on a 100,058 FITS database (1/10th *SDSS*).
- How much of this database is Hadoop churning through?

Performance Analysis

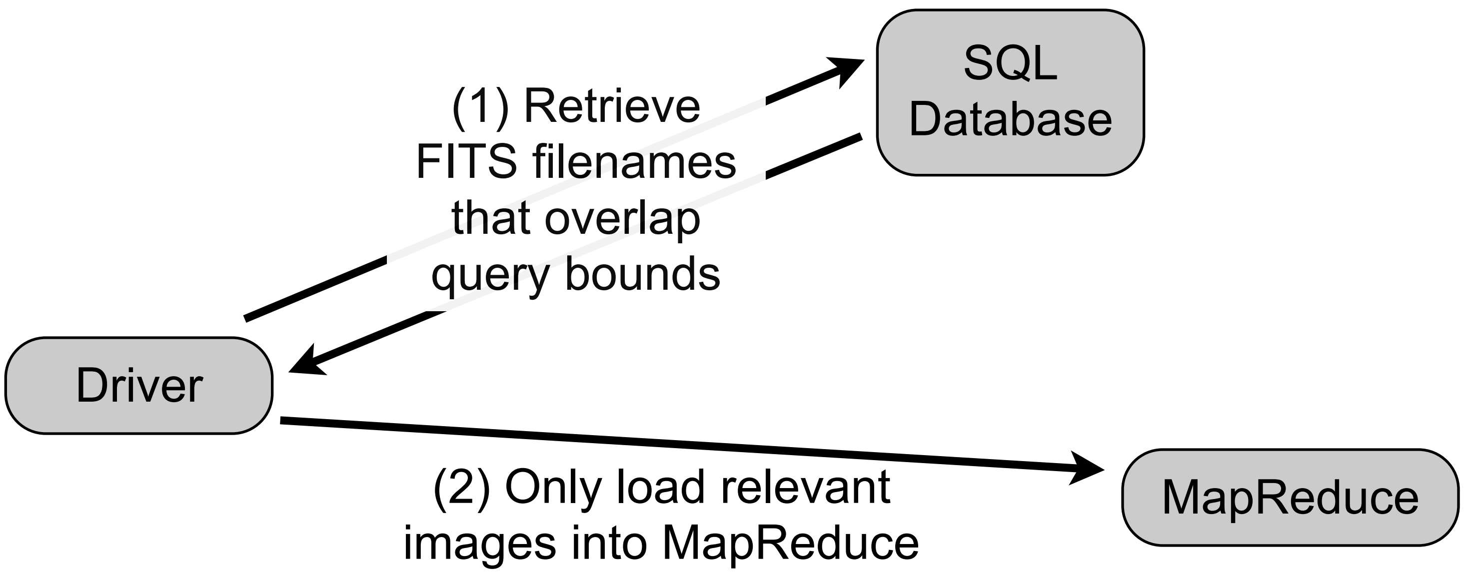


- **Comparison:**
  - Number of FITS considered in mappers vs. number contributing to coadd
- **Conclusion:**
  - Mappers must discard many FITS files due to nonoverlap of query bounds.

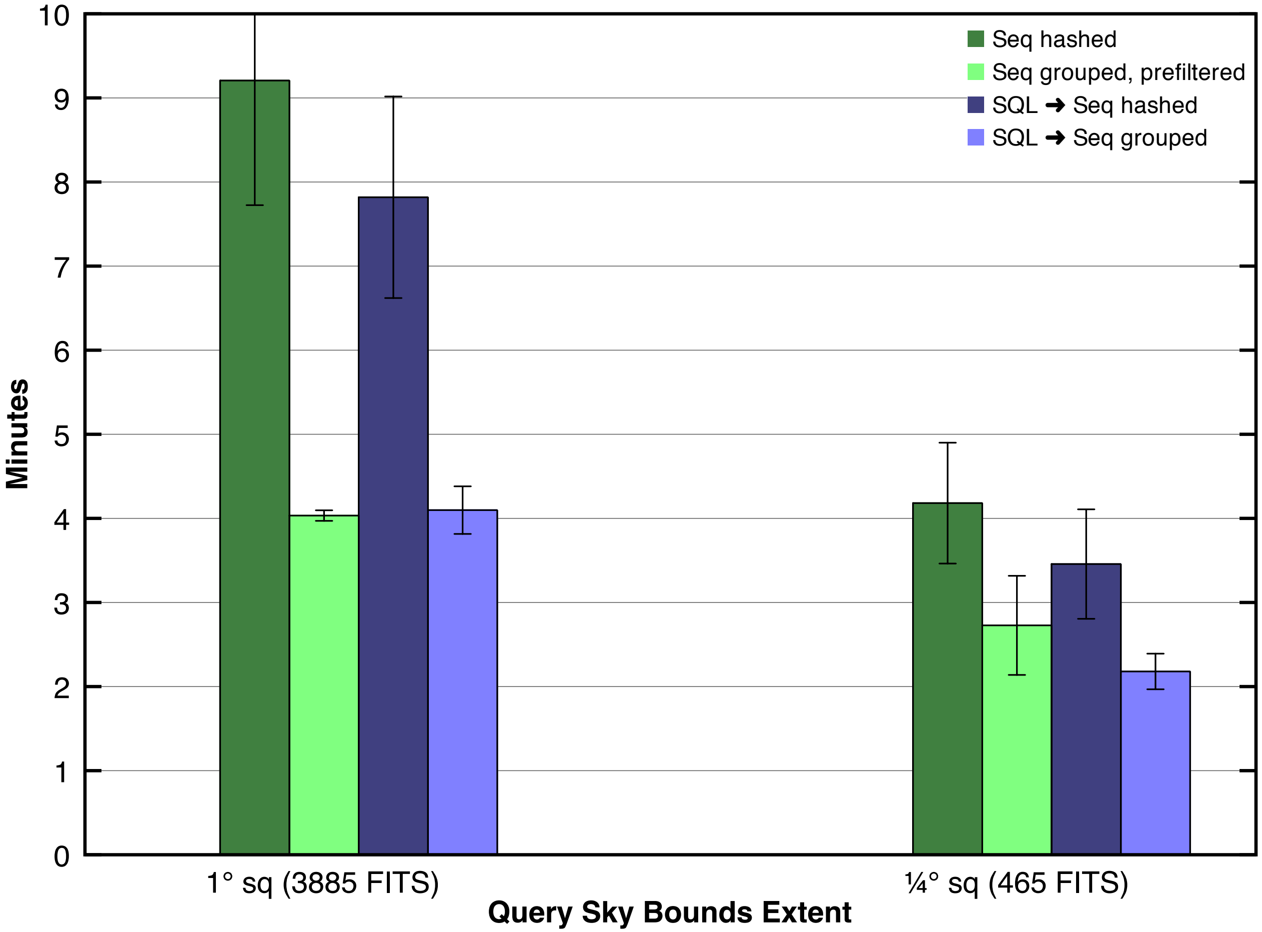
* 360 seq files in hashed seq DB.
† 1080 seq files in structured DB.
‡ 800 mapper slots on cluster.

Using SQL to Find Intersections

- Store all image colors and sky bounds in a database:
  - **First**, query color and intersections via SQL.
  - **Second**, send only ***relevant*** images to MapReduce.
- ***Consequence***:
  All images processed by mappers contribute to coadd.
  No time wasted considering irrelevant images.

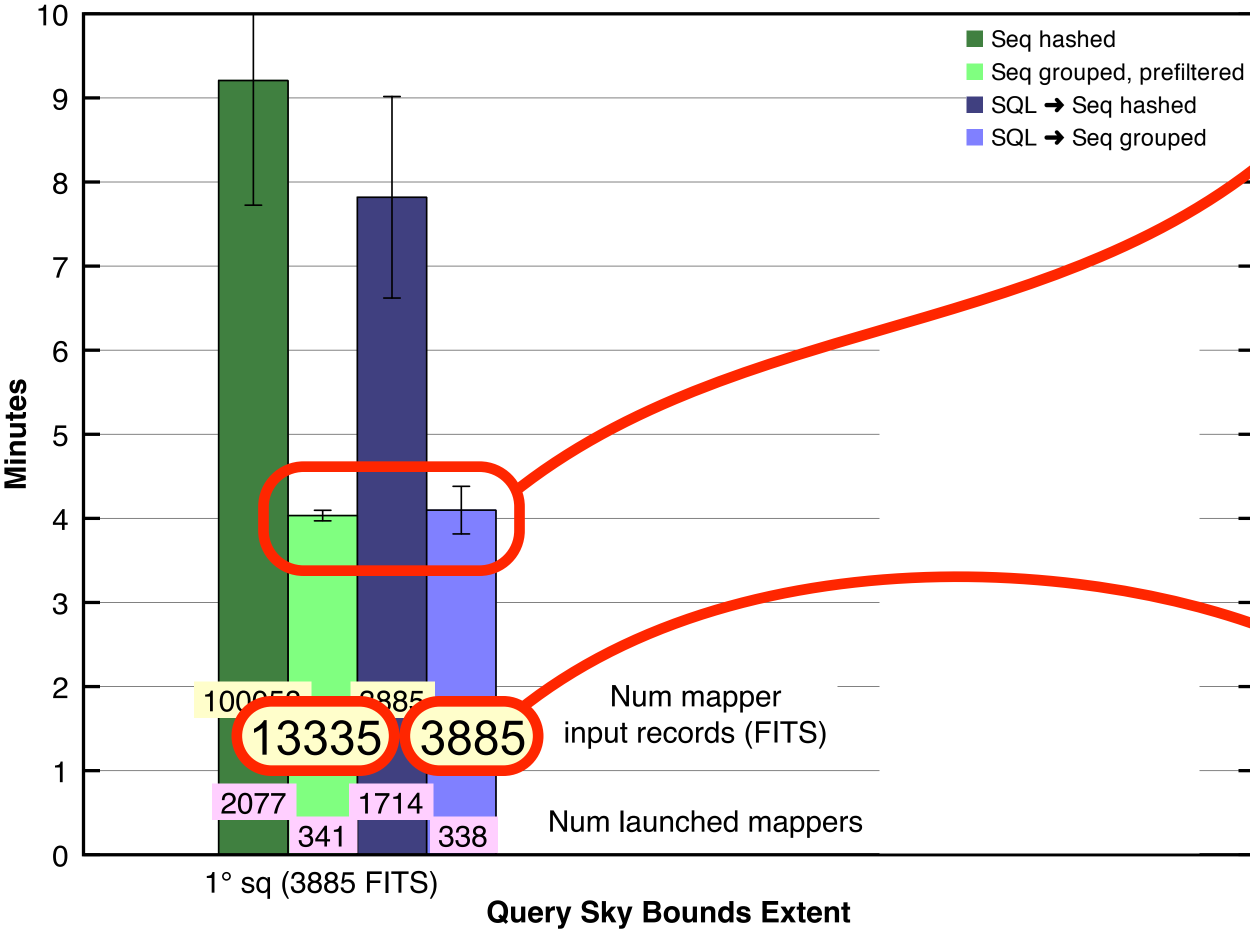


Performance Analysis



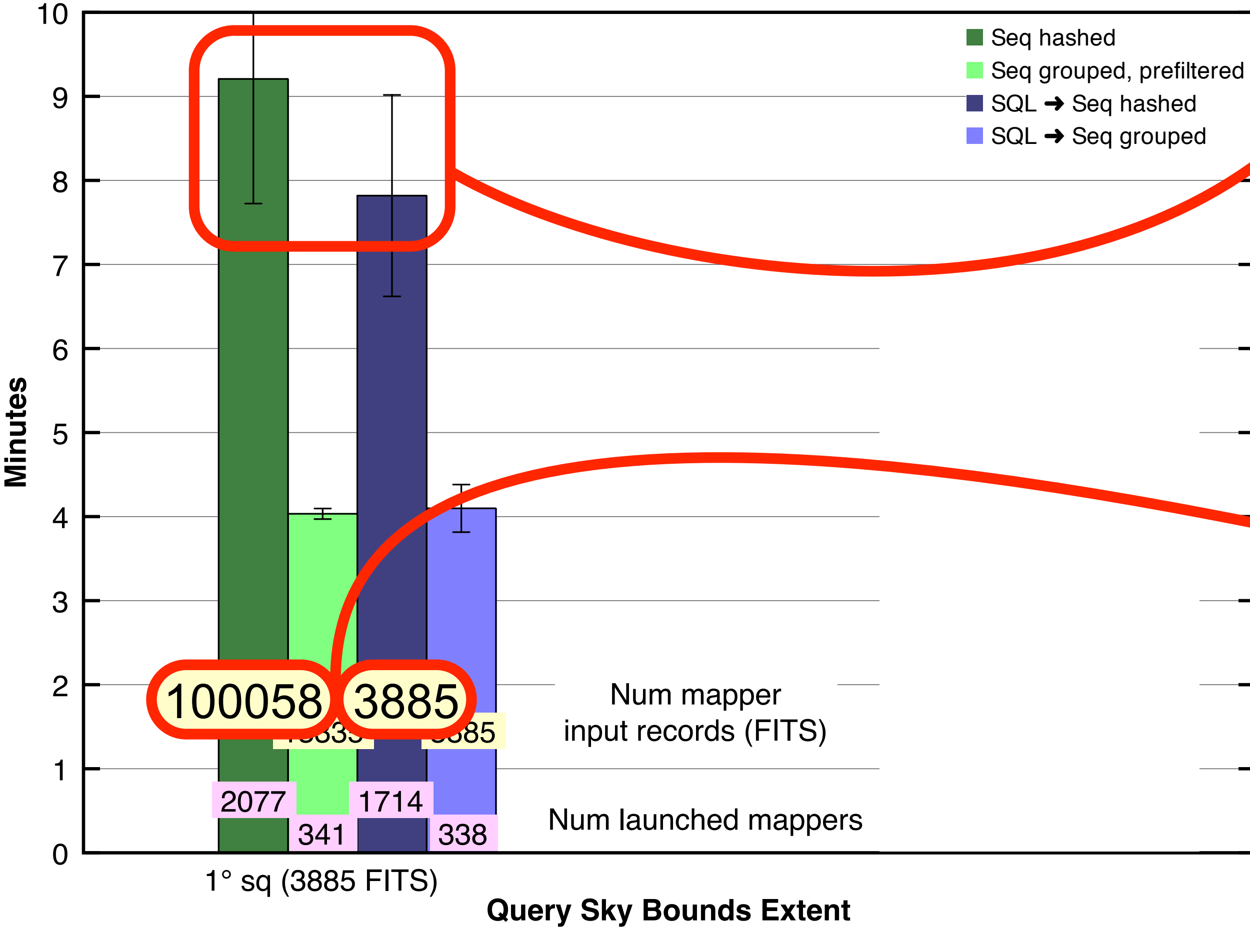
- **Comparison:**
  - **nonSQL** vs. **SQL**
- **Conclusion:**
  - *Sigh*, no major improvement (**SQL** is not remarkably superior to **nonSQL** for given pairs of bars).

Performance Analysis



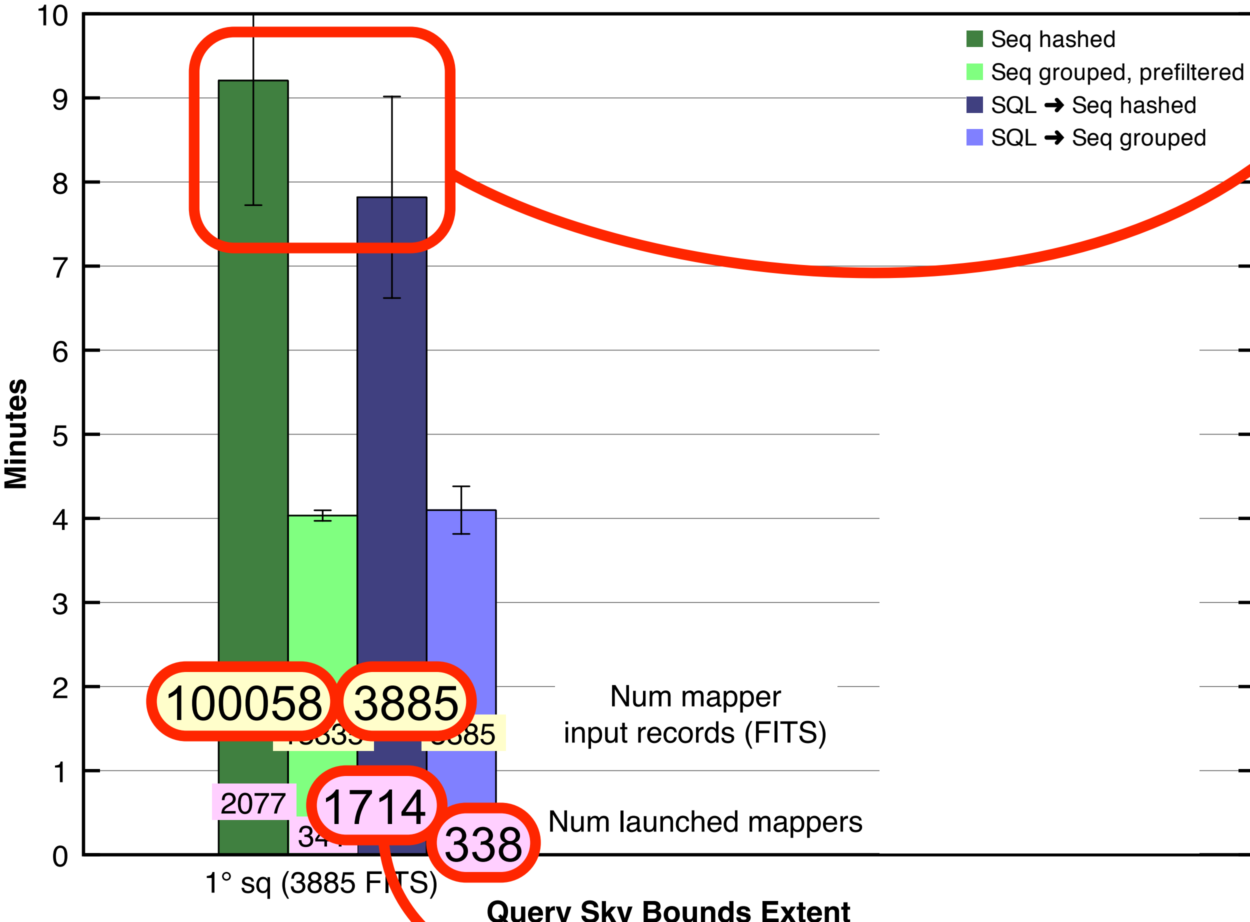
- Comparable performance here makes sense:
  - In essence, **prefiltering** and **SQL** performed similar tasks, albeit with 3.5x different mapper inputs (FITS).
- **Conclusion:**
  - Cost of discarding many images in the nonSQL case was negligible.

Performance Analysis



- Low improvement for SQL in the hashed case is surprising at first
  - ...especially considering 26x different mapper inputs (FITS)!

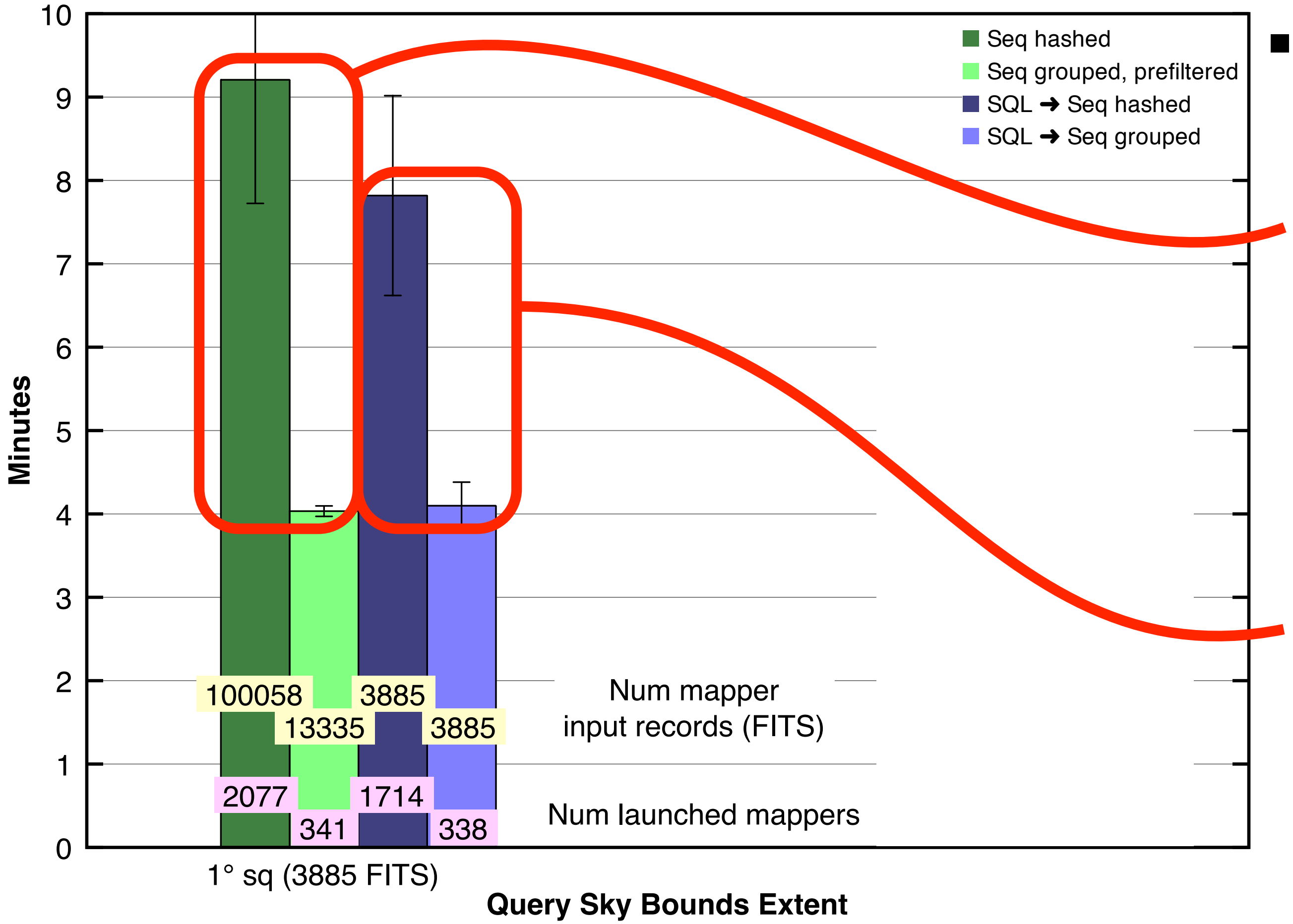
Performance Analysis



- Low improvement for SQL in the hashed case is surprising at first
- **Theory:**
  - Scattered distribution of relevant FITS prevented efficient mapper reuse.

Starting each mapper is expensive. This overhead hurt overall performance.

Results



- **Just to be clear:**
  - **Prefiltering** improved due to reduction of mapper load.
  - **SQL** improved due to data locality and more efficient mapper allocation – ***the required work was unaffected (3885 FITS)***.

Utility of SQL Method

- Despite our results
  (which show SQL to be equivalent to prefiltering)...

- ...we predict that SQL should outperform prefiltering on larger databases.

- Why?
  - Prefiltering would contend with an increasing number of false positives in the mappers*.
  - SQL would incur little additional overhead.

- No experiments on this yet.

* A spacing-filling curve for grouping the data might help.

Conclusions

- Packing many small files into a few large files is essential.
- Structured packing and associated prefiltering offers significant gains (reduces mapper load).
- SQL prefiltering of ***unstructured*** sequence files yields little improvement (failure to combine scattered HDFS file-splits leads to mapper bloat).
- SQL prefiltering of structured sequence files performs comparably to driver prefiltering, but we anticipate superior performance on larger databases.
- On a shared cluster (*e.g.* the cloud), performance variance is high – doesn't bode well for online applications. Also makes precise performance profiling difficult.

Future Work

- Parallelize the reducer.
- Less conservative CombineFileSplit builder.
- Conversion to C++, usage of existing C++ libraries.
- Query by time-range.
- Increase complexity of projection/interpolation:
  - PSF matching
- Increase complexity of stacking algorithm:
  - Convert straight sum to weighted sum by image quality.
- Work toward the larger science goals:
  - Study the evolution of galaxies.
  - Look for moving objects (asteroids, comets).
  - Implement fast parallel machine learning algorithms for detection/classification of anomalies.

Questions?

kbwiley@astro.washington.edu