Scalable Workflow System for Whole Slide Microscopy Analyses Using Neural Networks

Tim Blattner and Michael Majurski

NIST | ITL | SSD | ISG

National Institute of Standards and Technology
U.S. Department of Commerce

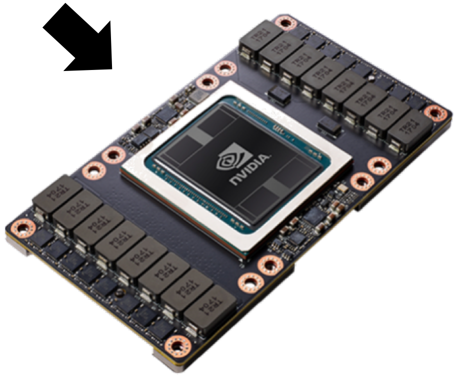
INFORMATION TECHNOLOGY LABORATORY

NIST Disclaimer

- No approval or endorsement of any commercial product by NIST is intended or implied. Certain commercial software, products, and systems are identified in this report to facilitate better understanding. Such identification does not imply recommendations or endorsement by NIST, nor does it imply that the software and products identified are necessarily the best available for the purpose.

Outline

- Part I – Specific Application
  - Microscope Image Acquisition
    - In-Situ Image Based Measurements
  - Data Motion
    - Triggers from Acquisition Software
  - Scalable Data Processing
    - HTGS, Fast Image



- Part II – Generalization
  - How to write software to scale with hardware
  - Problem Decomposition
  - Software developer efficiency

Motivating Problem

- Image-based measurements via Automated Microscopy
  - Automation increases data generation rate
- Microscopes can produce 100's of whole slide images per day
  - 5 gigapixel whole slide images
  - 100000 x 50000 pixels at 10x magnification
  - 5 minutes to acquire each image
- Acquisition rate defines the compute budget for performing image-based measurements.

Zeiss AxioScan Z1 Microscope

DISCLAIMER: Commercial products are identified in this document in order to specify the experimental procedure adequately. Such identification is not intended to imply recommendation or endorsement by NIST, nor is it intended to imply that the products identified are necessarily the best available for the purpose.

Whole Slide Image Analysis Problem

- Find all 100^2 pixel objects of interest within each 100,000 x 50,000 pixel slide

Example Slide

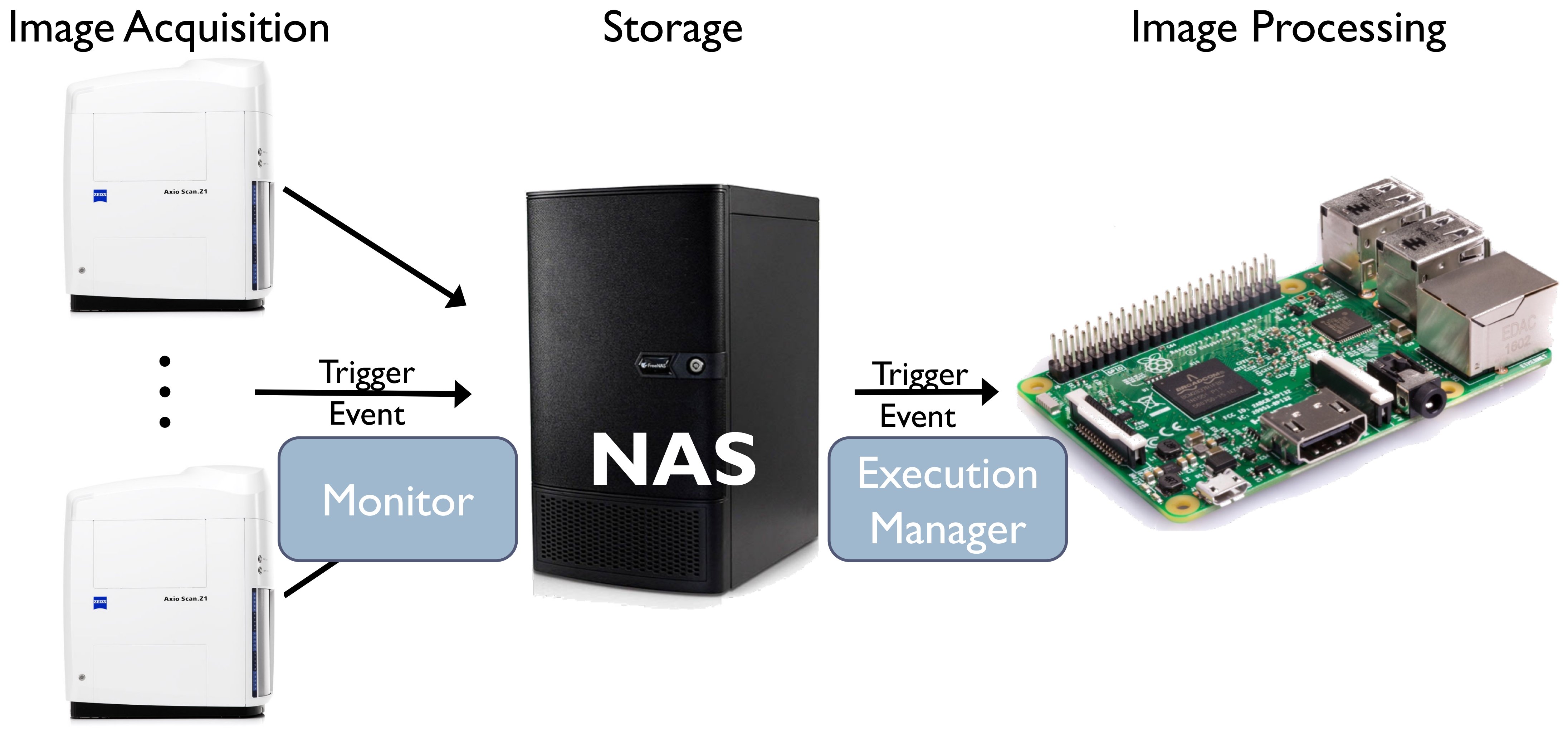
Goal:
Automate tedious,
error prone tasks

In Situ Image Analysis

- Perform analysis in semi-real-time while the next image is being acquired
- Brings image analysis into the wet-lab, providing immediate feedback/results

- Challenges
  - System must be robust and stable
  - Microscope acquisition time defines compute budget
- Benefits
  - End to end solution, user scanned image results are available 90 seconds later
    - Concurrent image acquisition and image processing
  - Data management
    - Integration with Zeiss Microscope to automatically copy image file to storage NAS
    - Processing computation initiated automatically when copy completes

Data Workflow



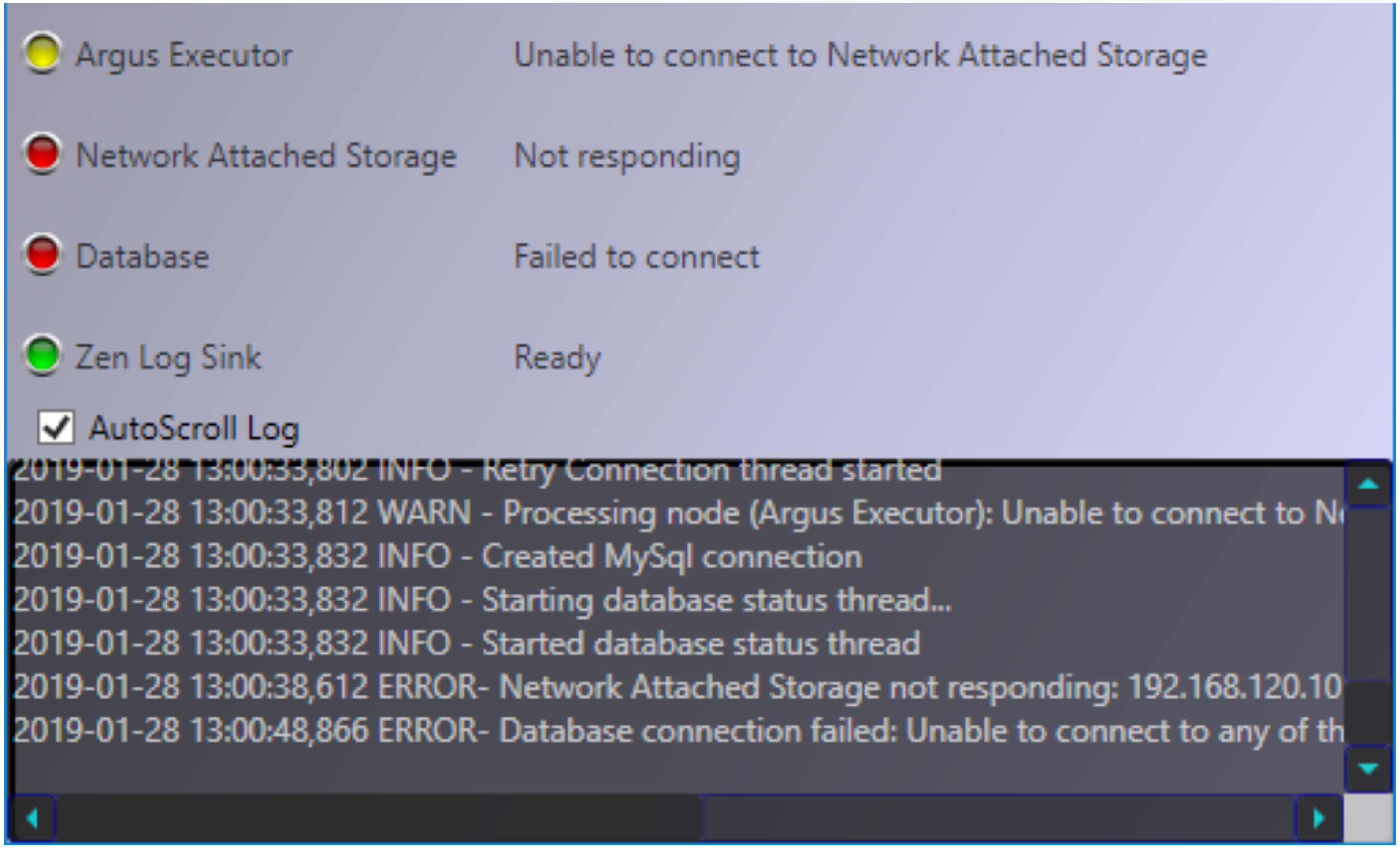
Microscope Acquisition System

- Queue up to 150 slides to be scanned
- AxioScan Z1 is driven from an attached computer
- Zeiss software scans images one at a time
- Images saved locally
- No local compute is allowed
- Lightweight program monitors scan status

Zeiss AxioScan Z1 Microscope

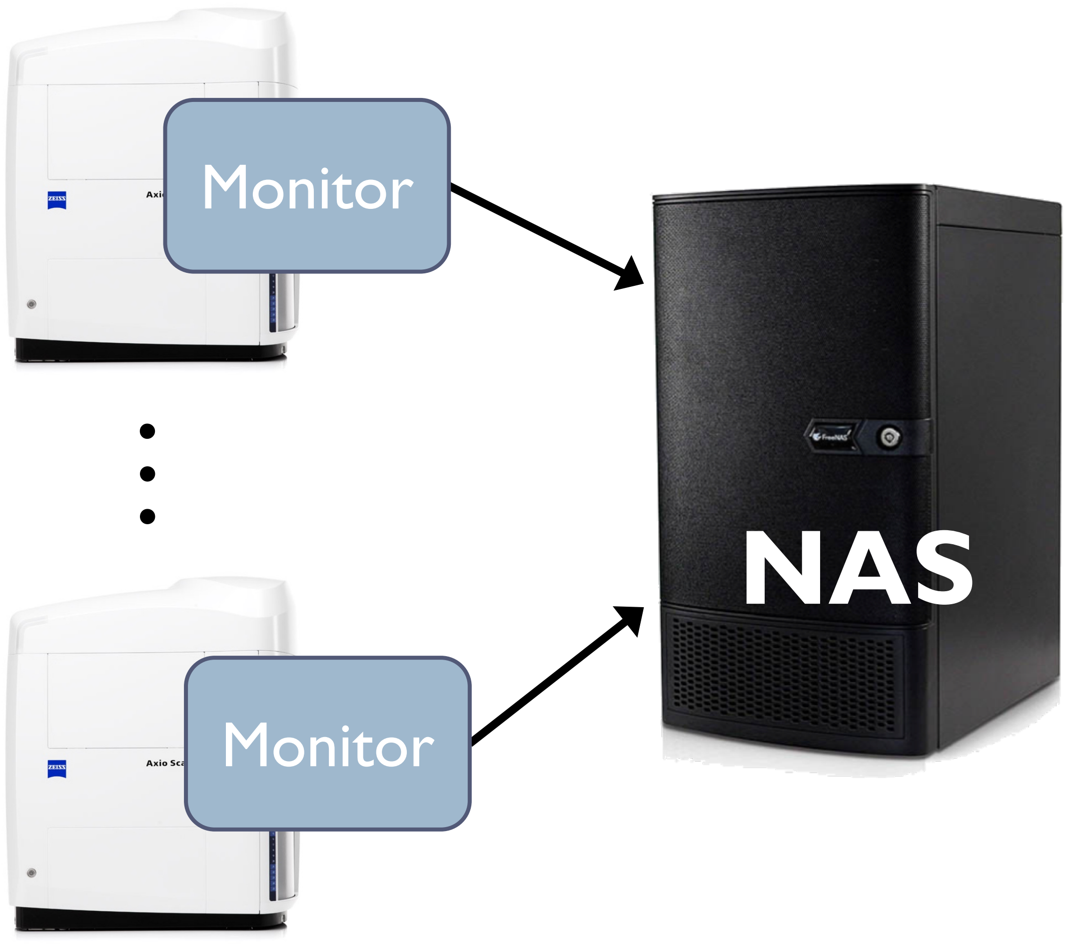
Acquisition Monitor

- Runs on Microscope Machine
- Two responsibilities
  - Showing System Status
  - Monitoring Zeiss log and copying completed scans to the NAS
- Green board means everything is up and running



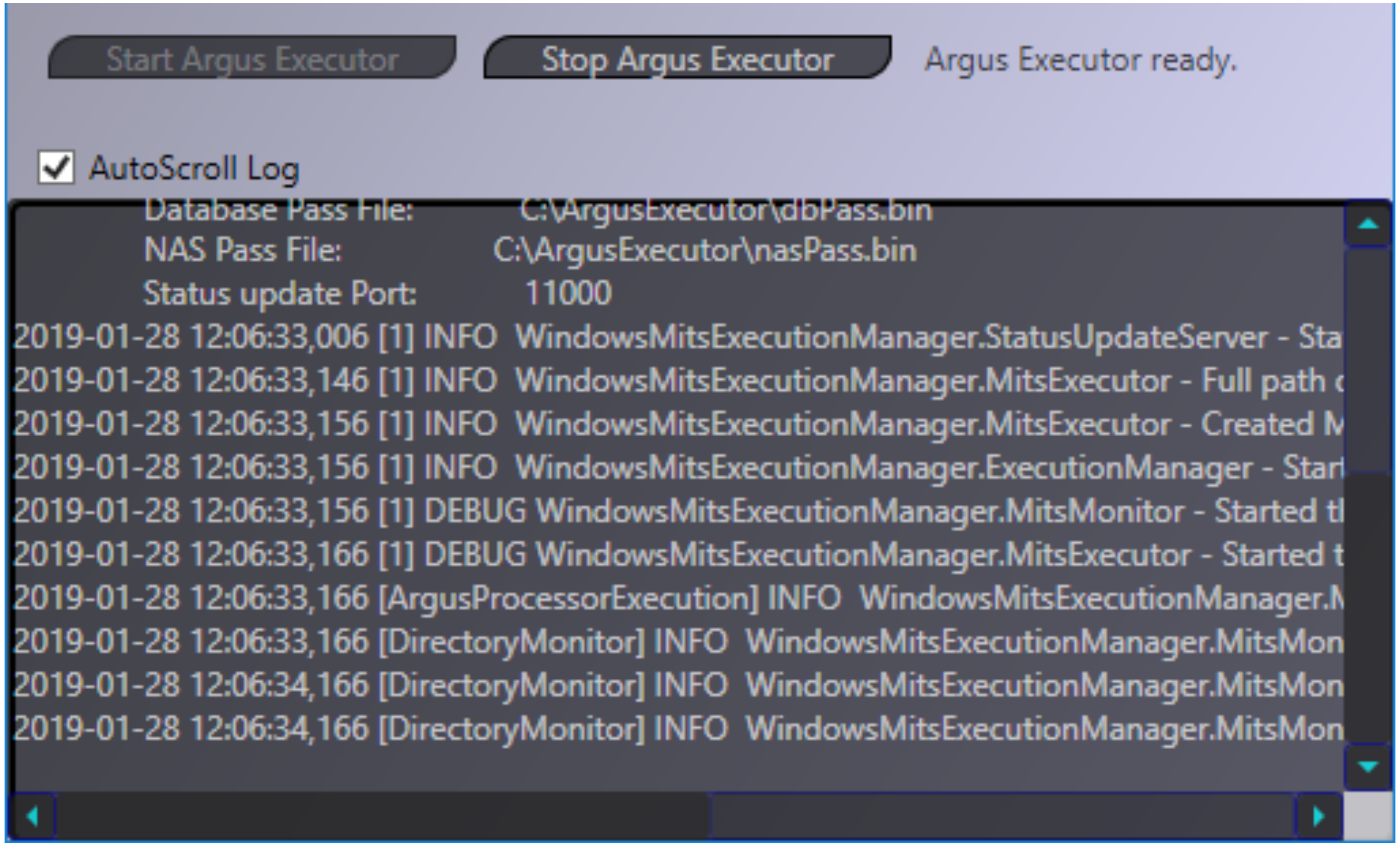
Acquisition Monitor

- Runs on each Microscope Machine
  - System supports N microscopes
  - Current hardware supports 4 microscopes
- Hooks to Zeiss event log
  - Watches for scan completed message
- Launches image copy to NAS
  - Able to recover if the network goes down
- Processing starts upon copy completion
- Copy & Processing decoupled from acquisition
  - Compute is overlapped with Acquisition

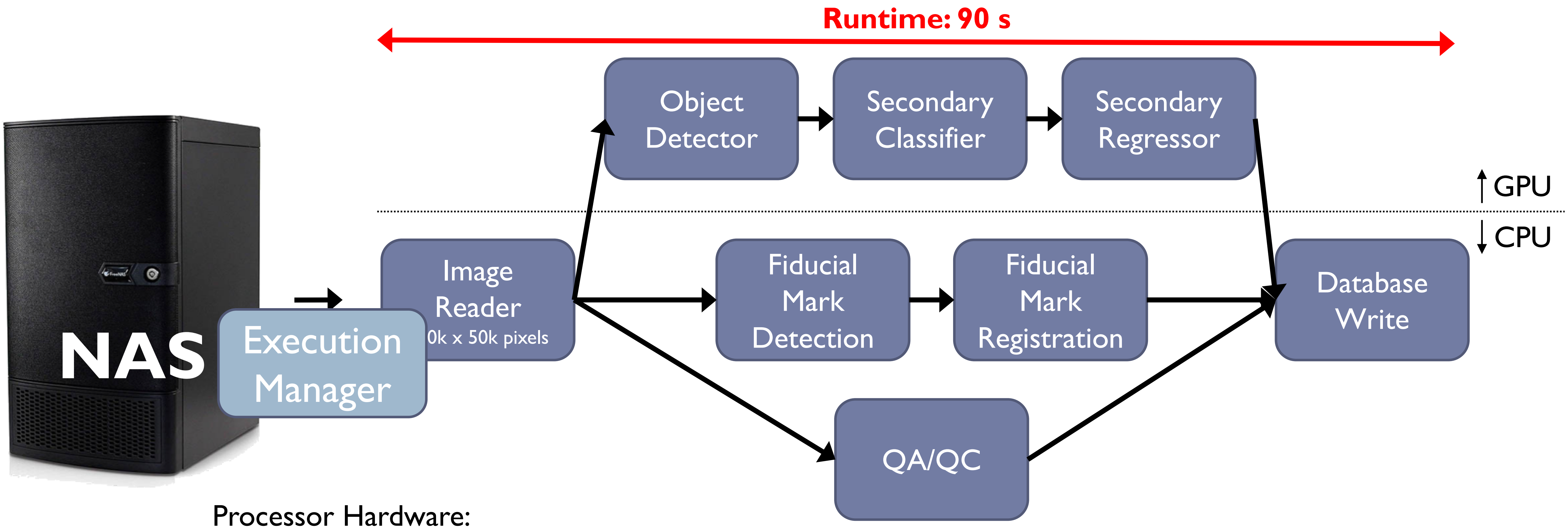


Execution Manager

- Runs on Processing Machine
- Launches Processing Workflow Per Image
  - Has an input queue of images
  - Monitor adds to that queue
- Displays Processing Log



Processing Workflow



Processor Hardware:

2x - Xeon Gold 5120 "Skylake" 14-core CPUs
2x - NVIDIA GTX Titan V graphics cards

Codebase: C++

Libraries: OpenCV, TensorRT, LibCZI, FastImage*, HTGS*

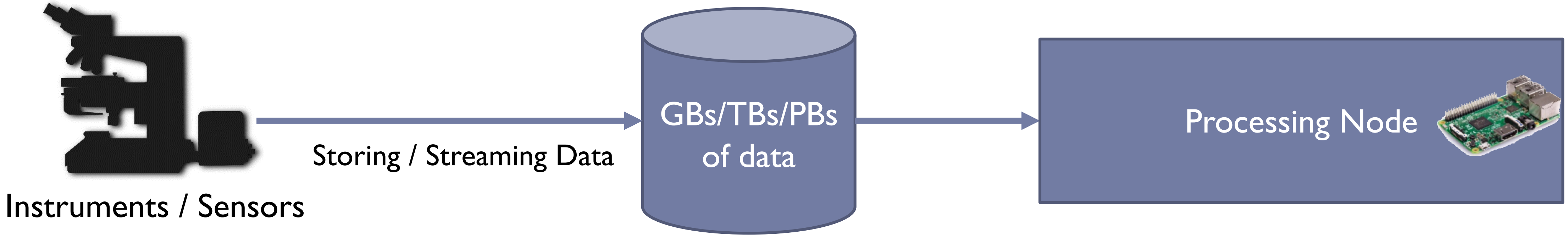
* NIST developed

Scalable Workflows

HTGS - Hybrid Task Graph Scheduler

Processing Workflow

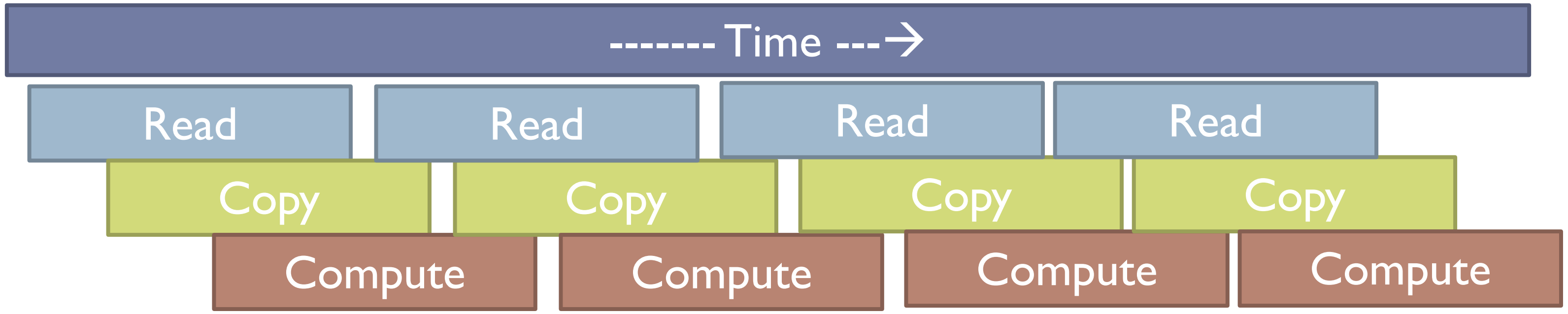
- Requirements
  - Meet time demands
  - Fully utilize processing hardware
  - Scales with increase to data rates
    - Multicore
    - Adding more GPUs to a machine
    - Additional processing nodes
- Approach
  - Asynchronous pipelined workflow
    - Effectively keeps processing resources busy
  - Appropriately size hardware
    - To deliver throughput



What is Pipelining

- Dictionary
  - A form of computer organization in which successive steps of an instruction sequence are executed in turn by a sequence of modules able to operate concurrently, so that another instruction can be begun before the previous one is finished.

- Applicable
  - Instruction pipelining: pre-fetching, branch prediction
  - Task pipelining: reading/writing data, copying, compute



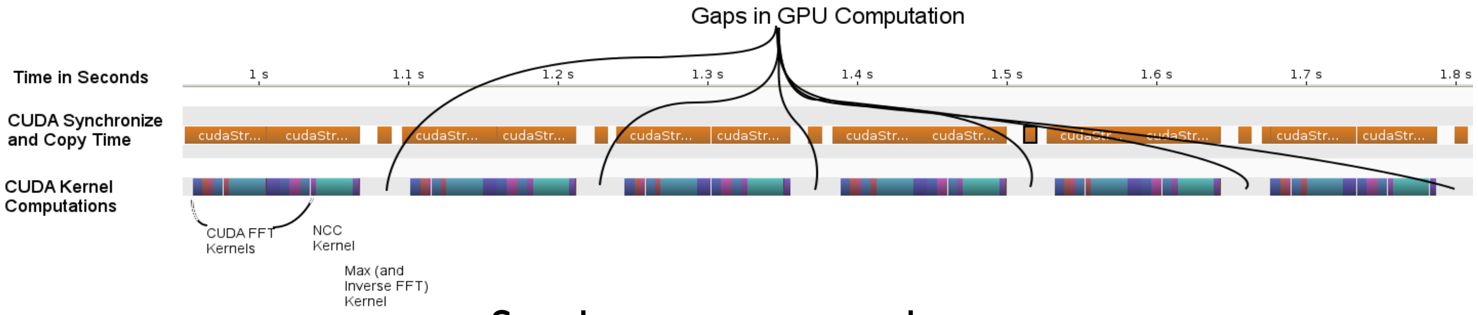
Example pipeline

Why Asynchronous Pipelined Workflow?

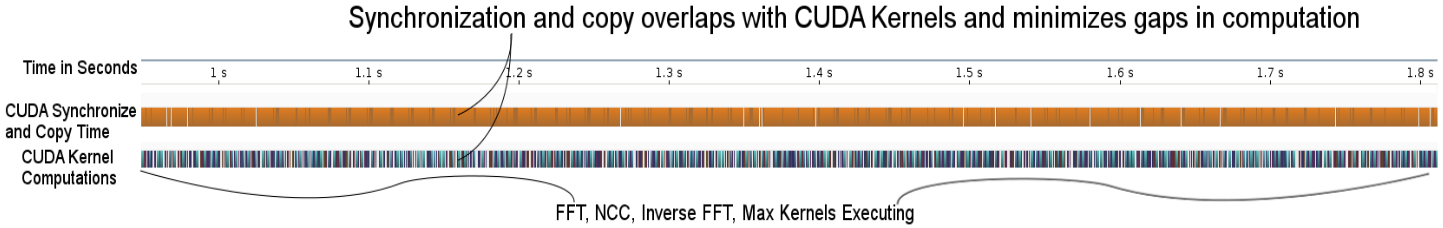
- Impacts on performance
  - Moving data between address spaces (CPU/GPU)
  - Disk I/O
  - Database updates

- Overlap computation with data motion, disk I/O, and database updates

- May require decomposing problem into multiple sub-problems



Synchronous approach



Asynchronous pipelined workflow approach

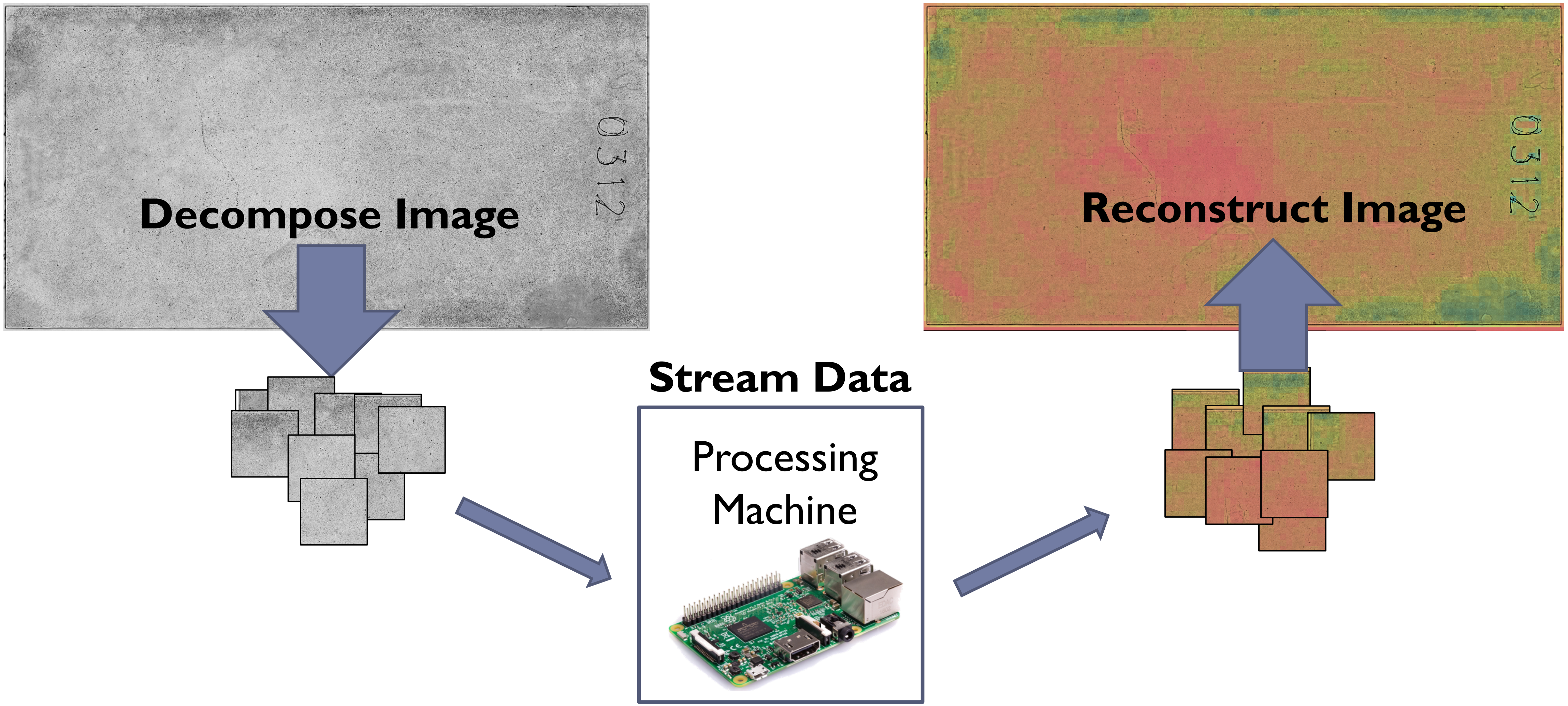
Challenges of Asynchronous Pipelining Workflows

- Legacy code often does not play well with pipelining
  - Restructuring and/or re-writing code may be necessary
    - Possibly algorithmic redesign
- Writing code that pipelines from scratch requires a lot of code structuring
  - Multi-threading
  - Thread safety
- Making sense of asynchronous behavior in an algorithm is not always obvious
- Solution: use existing frameworks … helps
  - Task libraries
    - StarPU, Legion, …
  - Hybrid Task Graph Scheduler (HTGS)

Hybrid Task Graph Scheduler

- Our approach for developing asynchronous pipeline workflows
- Application is a dataflow graph
  - Persists at runtime
  - Experimentation for performance
  - Debug, Profile, Visualize performance using the dataflow representation
- Targets powerful single nodes
  - Dual multicore CPUs & multiple GPUs
- Focus on
  - Separation of concerns
    - State maintenance versus computation
  - Coarse-grain parallelism
  - Hide latency of data motion
  - Memory management
- C++ Header API only

Streaming Tile Based Processing



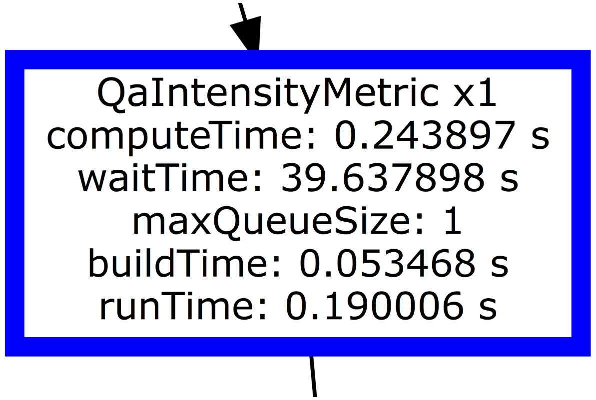
HTGS API

- Task interface
  - Templates define Input and Output
  - Execute(input)
    - Called when data is available for the task
  - addResult(output)
    - Called by user when output is ready to be sent to the next task
- Data is sent to the task
  - Each task holds a pool of threads
  - Extra data + multiple threads = parallelism

- Specialty tasks
  - Bookkeeper task
    - Manages complex data dependencies
    - Maintains state of computation
  - CUDA Task
    - Binds task to NVIDIA CUDA GPU
  - Execution Pipeline Task
    - Creates copies of a task graph
      - Each copy bound to a specified GPU
  - Memory Manager
    - Attaches memory edge to a task
      - getMemory("nameOfEdge")
      - Binds memory allocation to address space
        - CPU, GPU, etc.

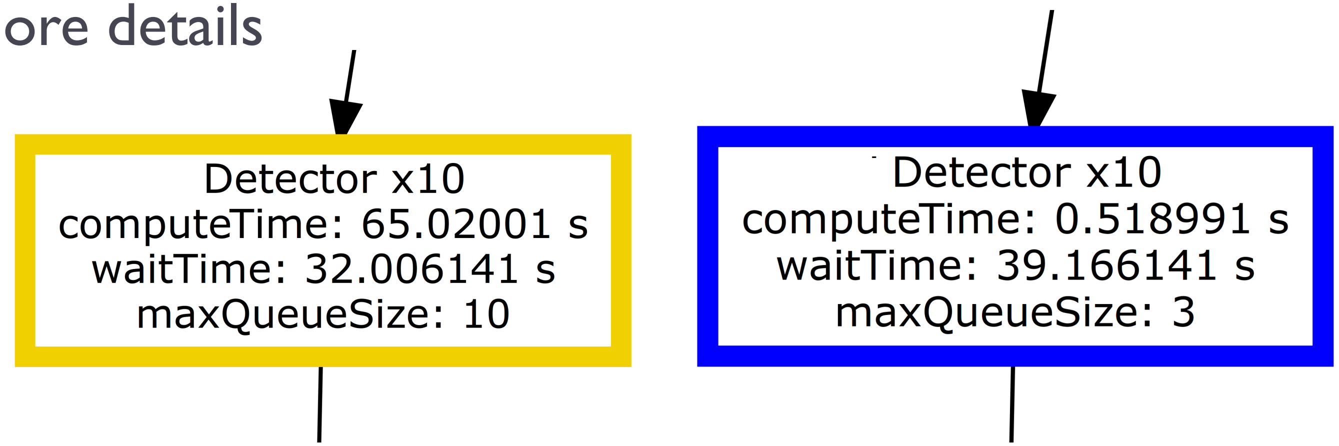
HTGS Profiling

- Zero overhead profiling
  - Profiling is gathered in both Release and Debug
    - Task level



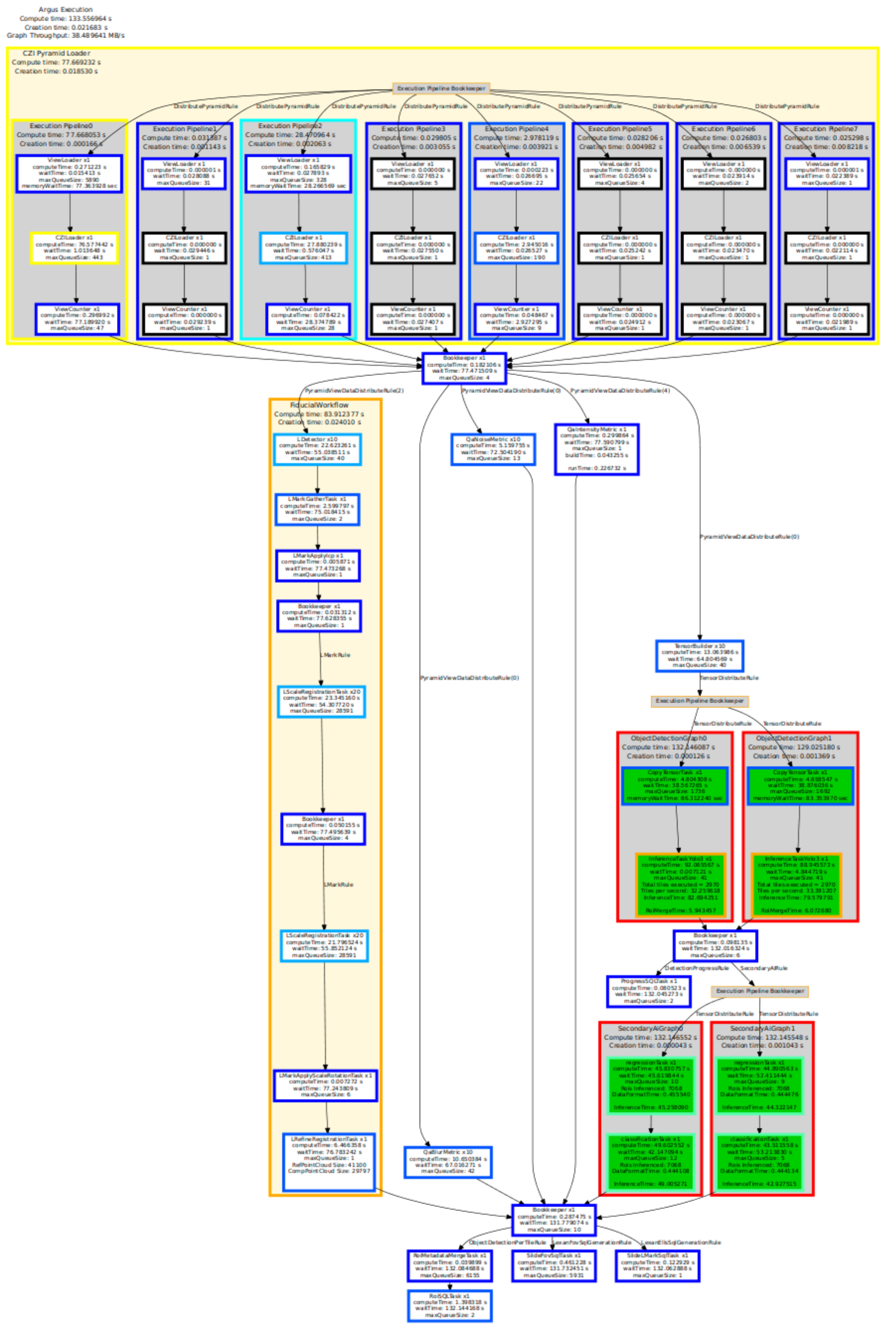
- Graph visualization report generated after every run
  - Immediately identify performance impacts per task
  - Customize task profiling to obtain more details

- Optimize per task
  - Find alternative methods



HTGS Software Engineering

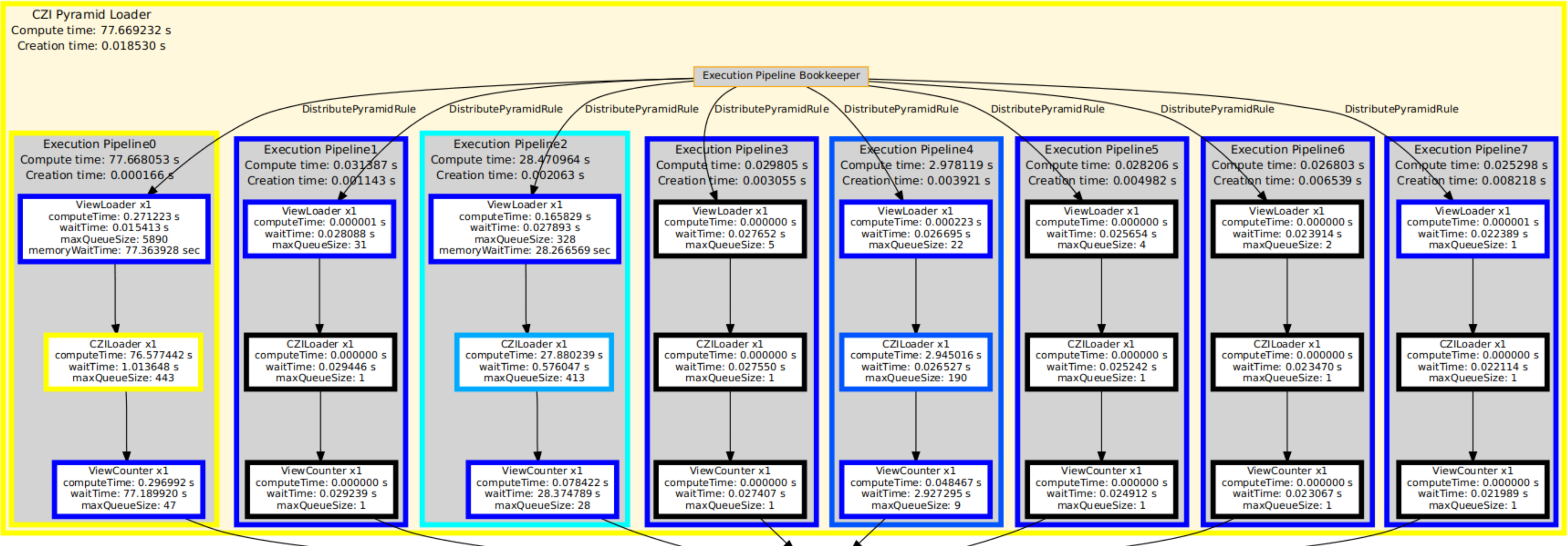
- Distribute tasks to developers
  - Narrow view of the world
    - Operate only on data sent as input and produce output
  - Difficult for developers to have code conflicts
    - Not impossible
- Maintains parallelism of the workflow
- Visual representation of critical path
  - Self-motivating to improve performance



Microscopy Analysis

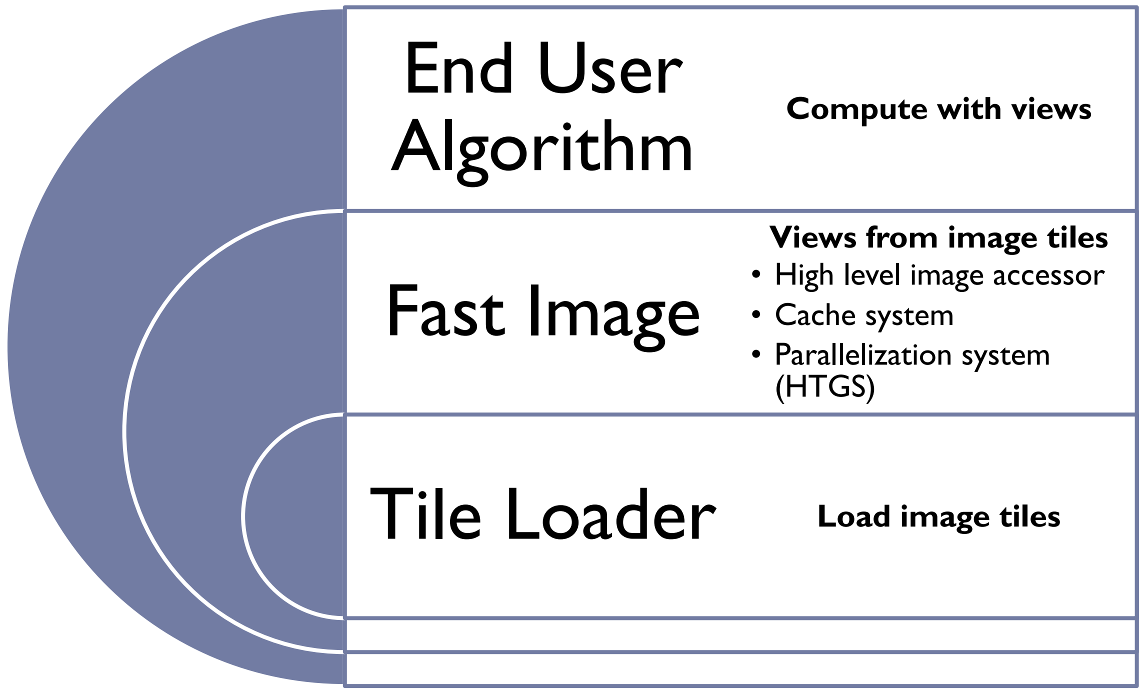
Image Reading

- Zeiss generates CZI images (pyramidal)
  - Tasks can operate at whatever pyramid they need to
  - Fiducial Detection happens at level 2 (25% resolution)
- Read all pyramid levels in parallel using FastImage

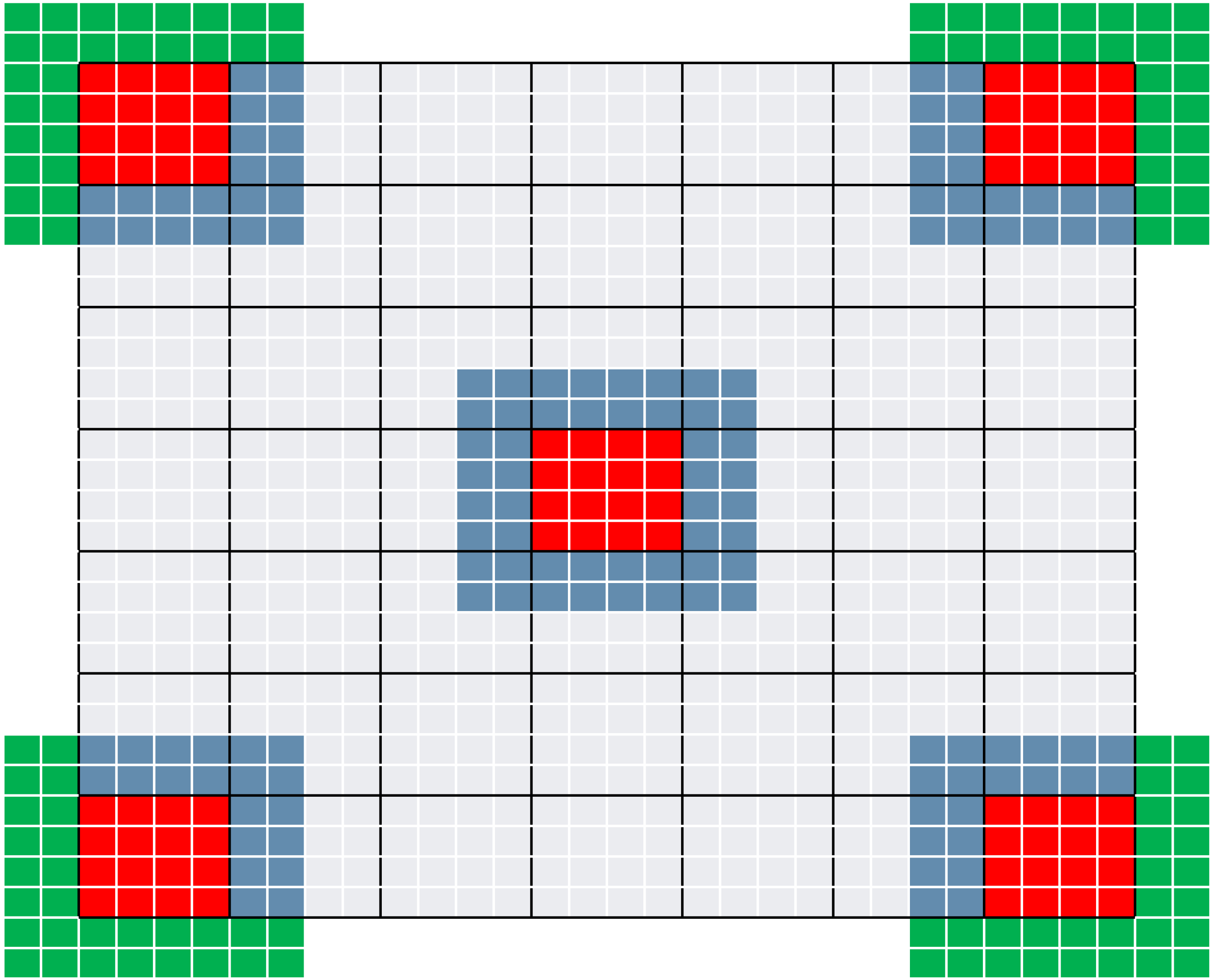


Fast Image

- C++ Library based on HTGS
- High level API to access an image
  - Or part of it
- Only access interesting views in an image
- https://github.com/usnistgov/FastImage



Views and Tiles



Tile

- Part of the image (here 28x28 pixels) given by the image loader

View

- Center tile (here 4x4 pixels)
- Neighboring pixels within a radius (here 2 pixels)
- Ghost (halo) values

Advantages

- Reduce memory footprint
- Enable tile caching & parallelism

Problem Decomposition into Tasks

Task Graph

- Traditional CV Tasks
  - Quality Analysis/Quality Control
  - Fiducial Mark Detection and Registration
    - Pipeline of subtasks
- AI Tasks
  - Object Detection
  - Secondary Classification/Regression

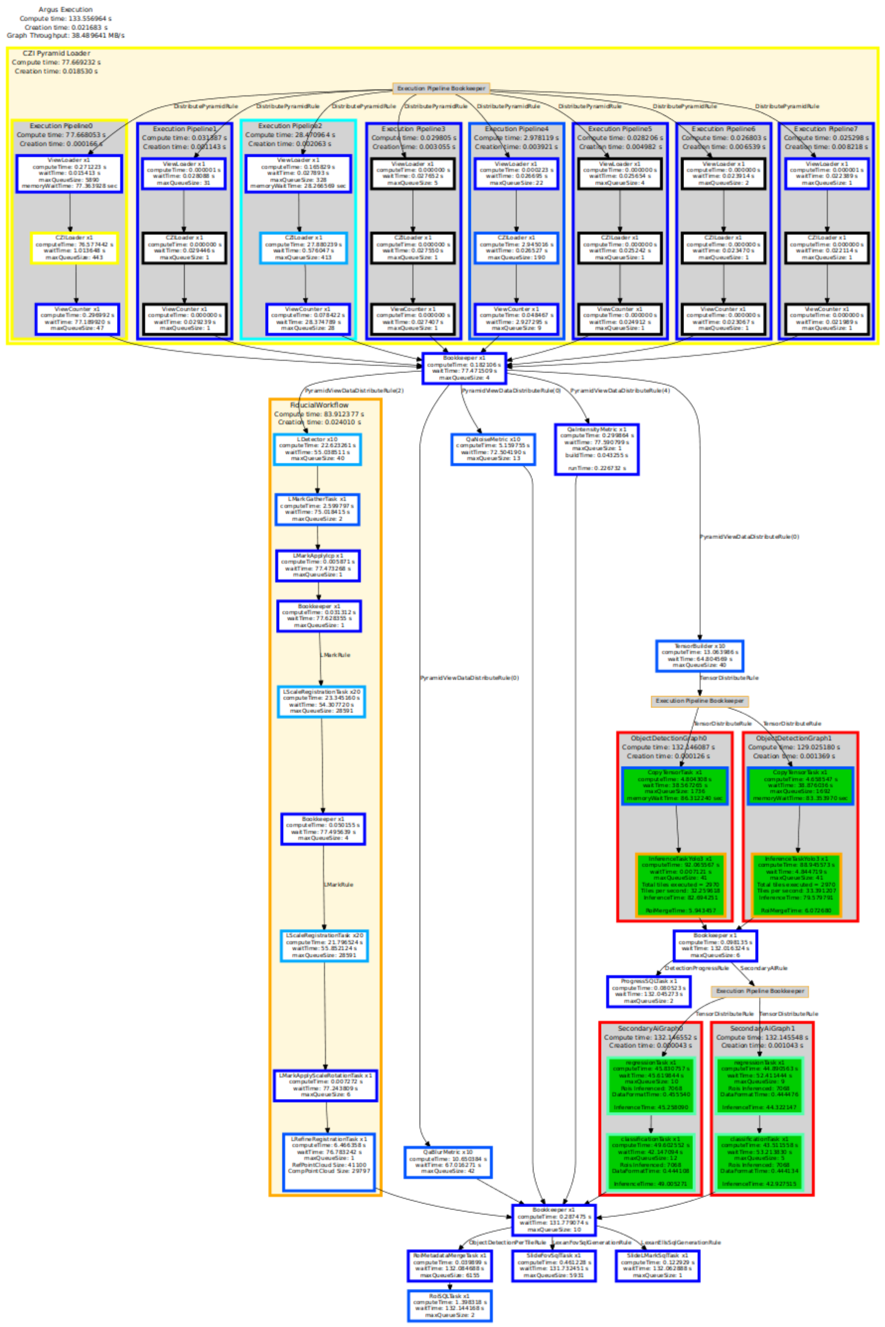
Relative Compute Time

Less More

Image Readers

Processing Workflow

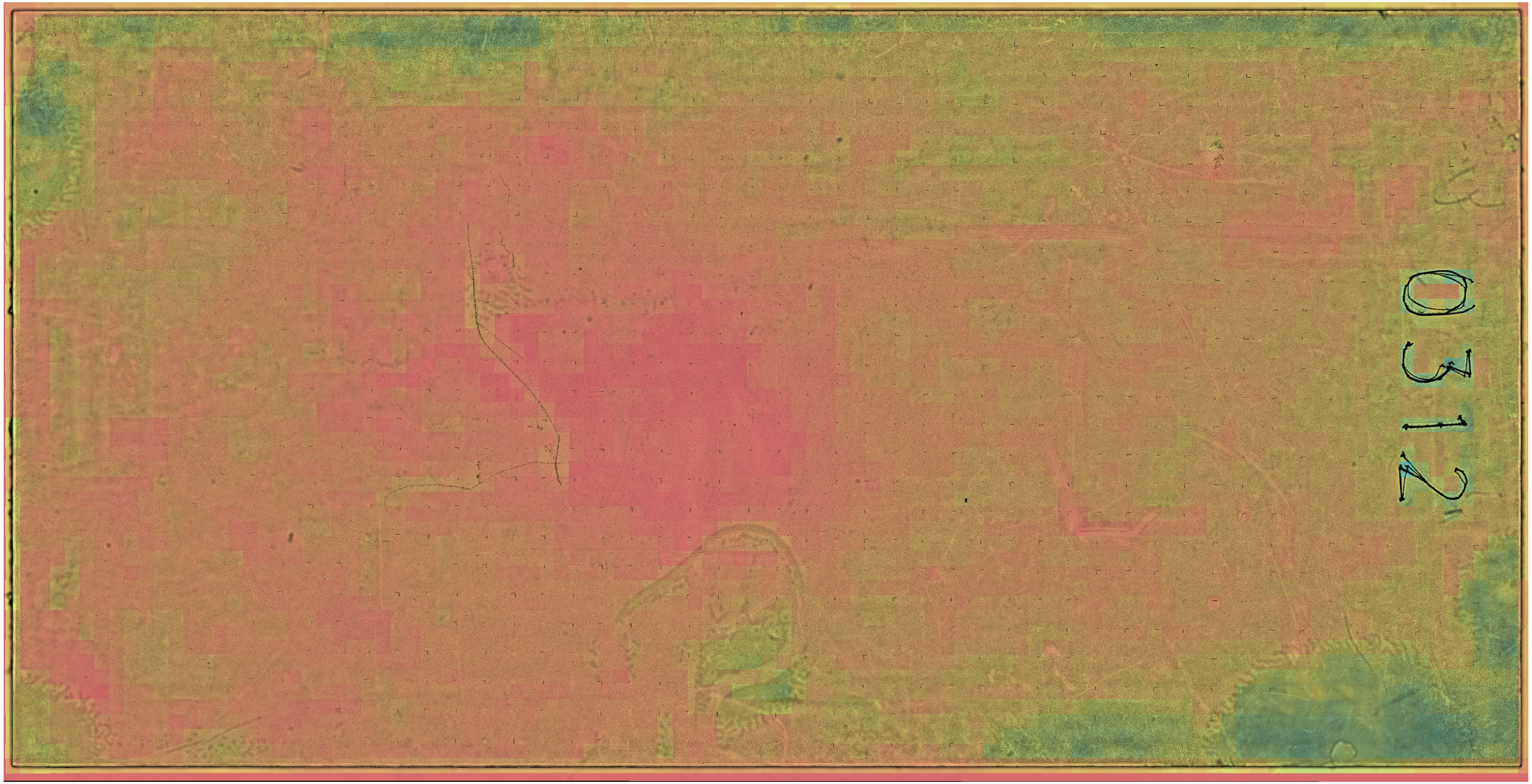
Database Transactions



QA/QC

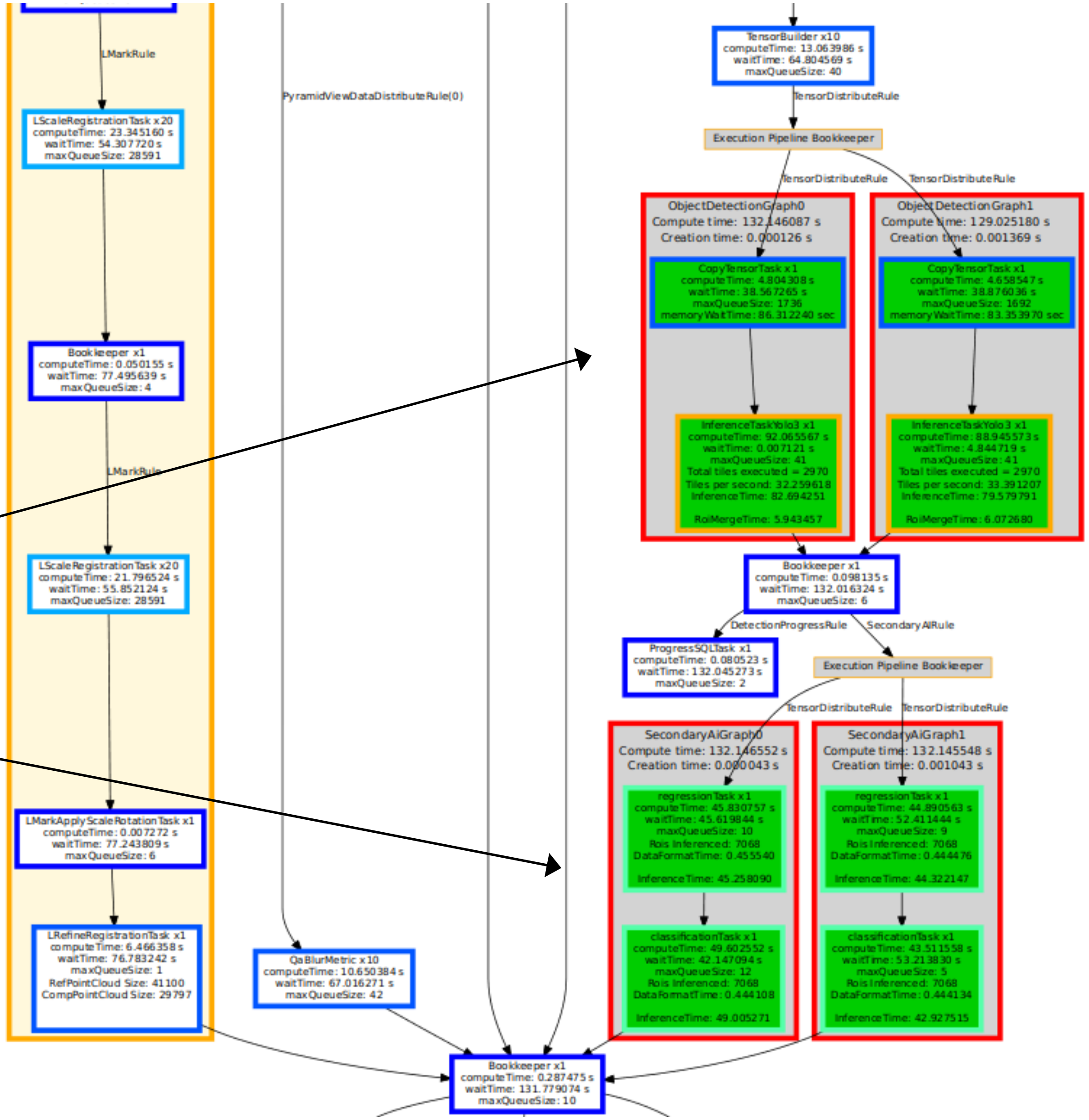
- Input: whole slide image coming directly from microscope
- No guarantees the image was
  - acquired correctly
    - contains the expected content
- Need to verify that the input image is within quality specifications for
  - Focus (image blur)
  - Background noise
  - Image brightness
- Each QA/QC item is a task in the HTGS system
  - Leverages OpenCV

QA/QC – Quality Heatmap



AI Tasks

- Traditional Computer Vision Tasks
  - Quality Analysis/Quality Control
  - Fiducial Mark Detection/Registration
    - Pipeline of subtasks
- AI Tasks
  - Object Detection
  - Secondary Classification/Regression



Relative Compute Time

Less More

AI Tasks

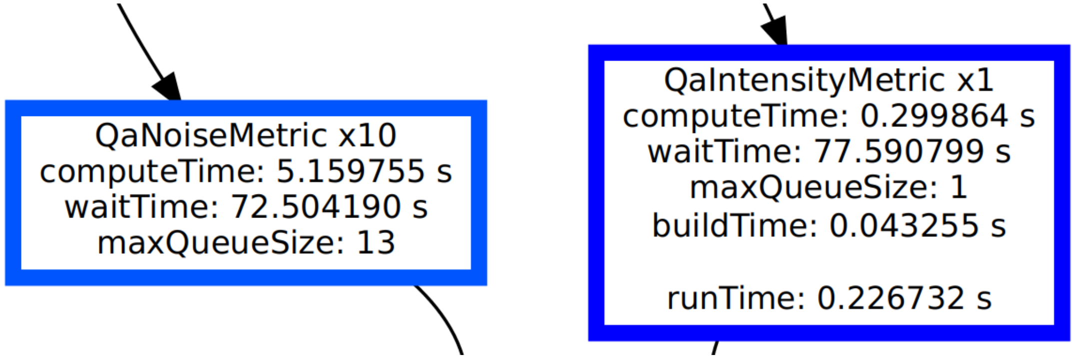
- Object Detection
  - Yolo v3 model
- Secondary Feature Categorization/Regression
  - Pair of ResNet50 models

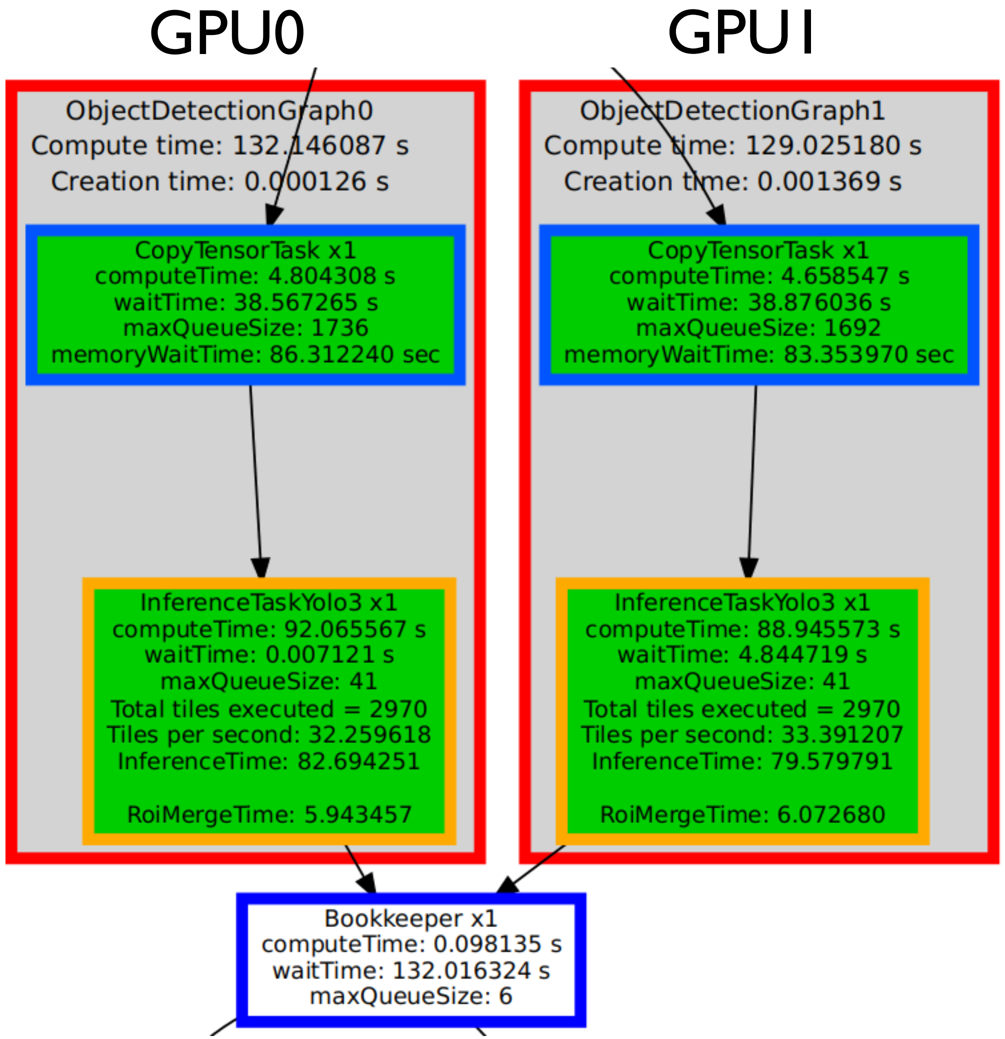
- Object detector extracts Regions of Interest (ROIs)

- ROIs are streamed to secondary AI models
    - As soon as the detector task generates an ROI output
  - Classification and regression tasks are run sequentially per ROI
  - Each task adds a new metadata value to an ROI as it flows through the compute graph

Task Scalability

- Specify thread count per task to control task's processing
  - Fiducial mark detection is expensive
    - 40 threads
  - QA is cheap
    - 1-10 threads
- AI tasks utilize all available GPUs





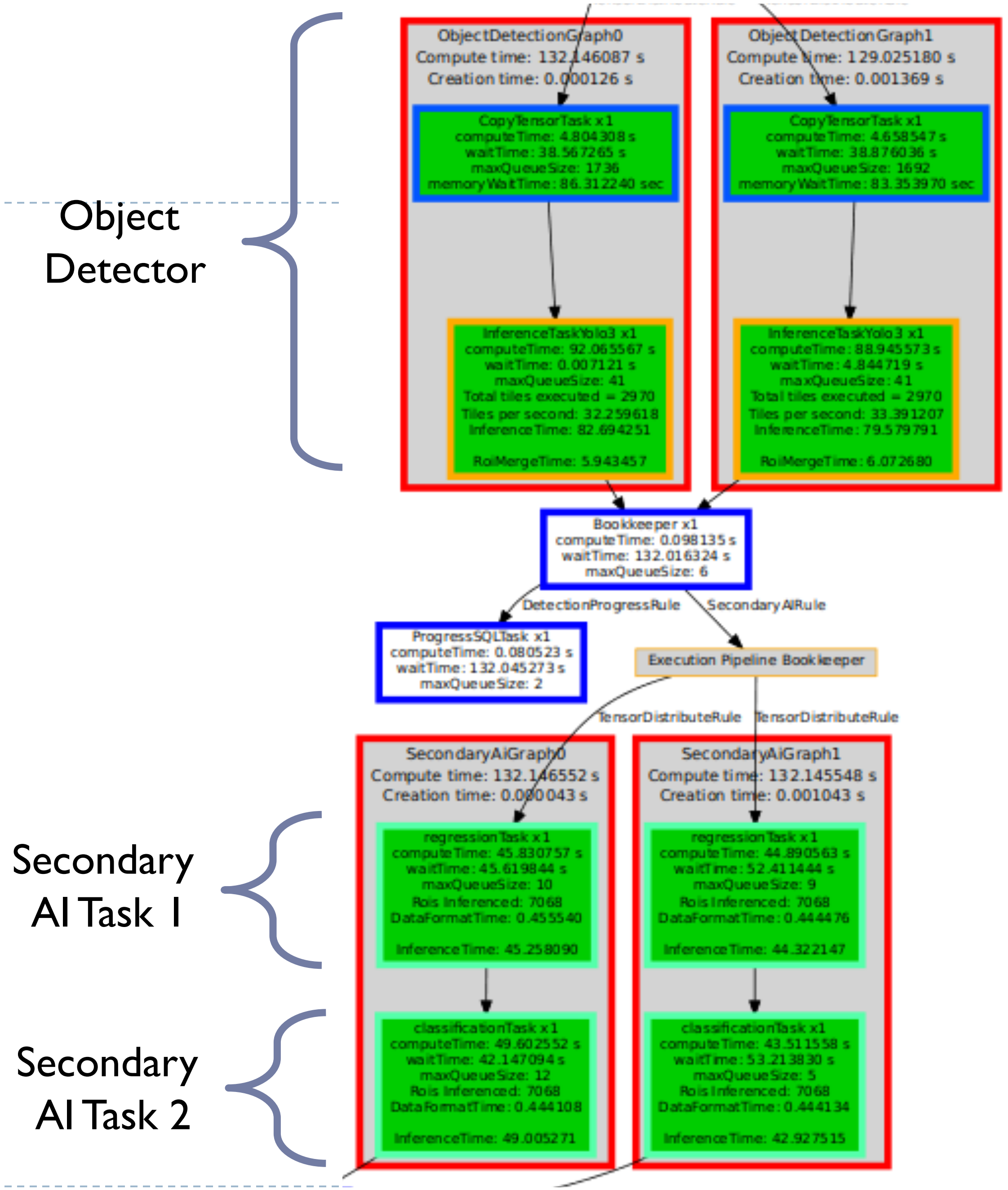
TensorRT

TensorRT

- AI models trained in Tensorflow on Power9 with V100s
  - Trained models saved in UFF format
- Deploy AI in TensorRT
  - Enables FP16 inference using Titan V tensor cores
  - Translate from UFF to TRT
- Processor detects what UFF/TRT models exist
  - Auto-generates new optimized TRT when UFF is newer than existing TRT
  - TRT creation time is amortized over many runs
  - Generate TRT on deployment hardware to have models optimized for platform of interest

TensorRT Engine

- TensorRT engine initialization
  - Loads serialized TRT model
- One TRT Engine per AI task
  - AI tasks duplicate: one per GPU using HTGS execution pipeline
- Example with 2 Titan V GPUs used for inferencing on 3 separate TRT models
  - Each red box is a sub-workflow per GPU
- Progress updates sent to database
  - Real-time visualization of progress



Workflow Scalability

- Codebase is designed for scalability
  - Auto detects the number of GPUs
  - Initializes one AI workflow, per model, per GPU
- Add another GPU if AI tasks are the bottleneck
  - AI tasks will duplicate
- CPU tasks specify the number of threads to use
  - Allocation of threads distinct from what to compute
  - Specifying thread count controls speed of task completion
  - Allows for load balancing between different tasks
    - Fiducial mark registration is CPU-bound and uses 40 threads
    - QA is fairly cheap and gets 1 thread only.

Programming Scalability

- Software developers write HTGS tasks
  - Tasks are constrained so that they can scale arbitrarily if needed
- Once dev decomposes the problem into a data streaming task based processing, tasks become the logical unit of thinking
- For example, whole image is never passed through the GPU for inference.
  - Too large to fit in GPU memory
  - Inference happens per 1024 x 1024 tile managed by Fast Image
- Developers incur upfront cost of learning curve
  - Enables parallel development & testing at task level
  - Simplifies obtaining performance-oriented software

Summary

Requirements for Realtime Microscope Processing

- Hardware:
  - Microscope with event hooks to enable automation
  - Storage for images
  - Processing machine
    - Same as acquisition machine or on LAN

- Software
  - Fixed analysis pipeline which needs to be applied to every image coming off the microscope
  - Task based, data streaming processing model
  - Workflow system to orchestrate compute in a scalable manner

Future Work

- Hedgehog: next generation of HTGS
  - Hedgehog simplifies graph design
    - Each task can have multiple input types and broadcast outputs
  - Hedgehog implementation of FastImage
    - Generalization beyond image data → serialized matrices data
  - Available now
- Shift from UFF AI model format to ONNX

Thank You

- Questions?

- Code
  - HTGS: github.com/usnistgov/HTGS
  - Hedgehog: github.com/usnistgov/hedgehog
  - FastImage: github.com/usnistgov/FastImage
  - HedgehogFastImage: soon

- Email

  - timothy.blattner@nist.gov
  - michael.majurski@nist.gov
