Import Of Large Image Data Sets

Dr. Jürgen Helmers

February 16, 2012

Goals

- Standard compliance
  - Keep meta-data with every single file
  - Use a standard compatible file format
- Security
- Error handling
- Automate the process
  - No user interaction
- Speed

Image Data Source

- ImageXpress Ultra Confocal High Content Screening System
  - Automated true point-scanning confocal microscope
  - MetaXpress High Content Image Acquisition & Analysis Software

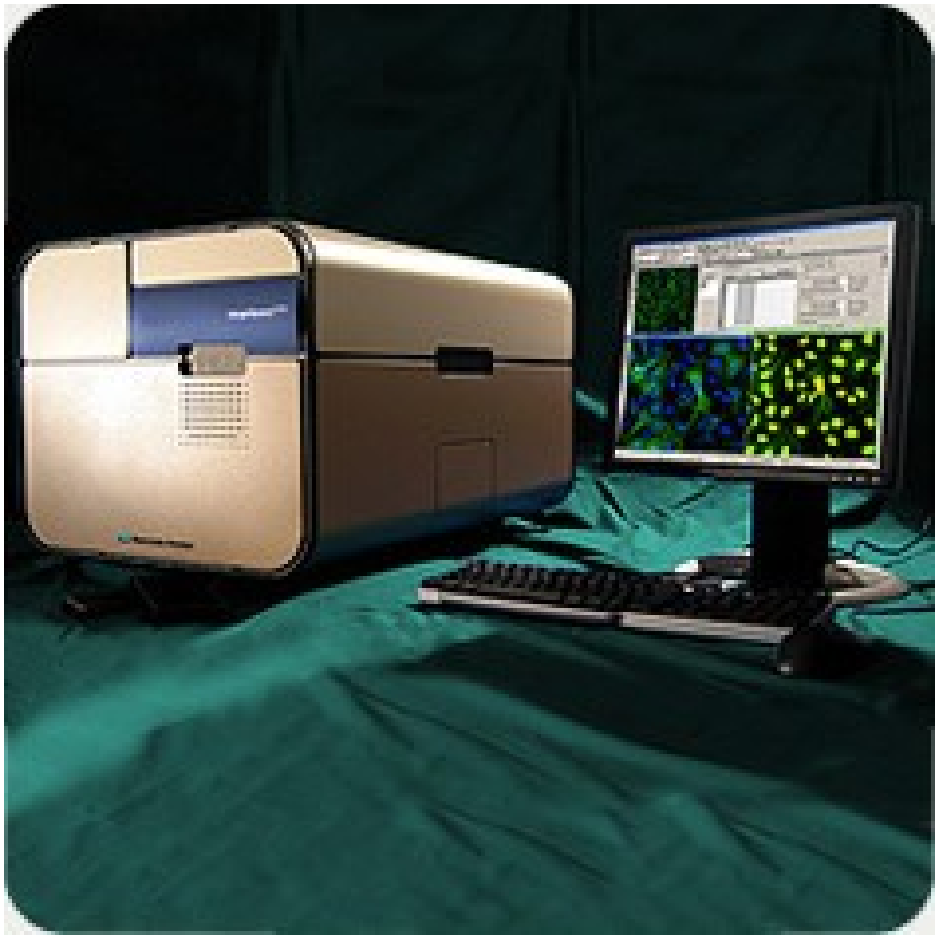
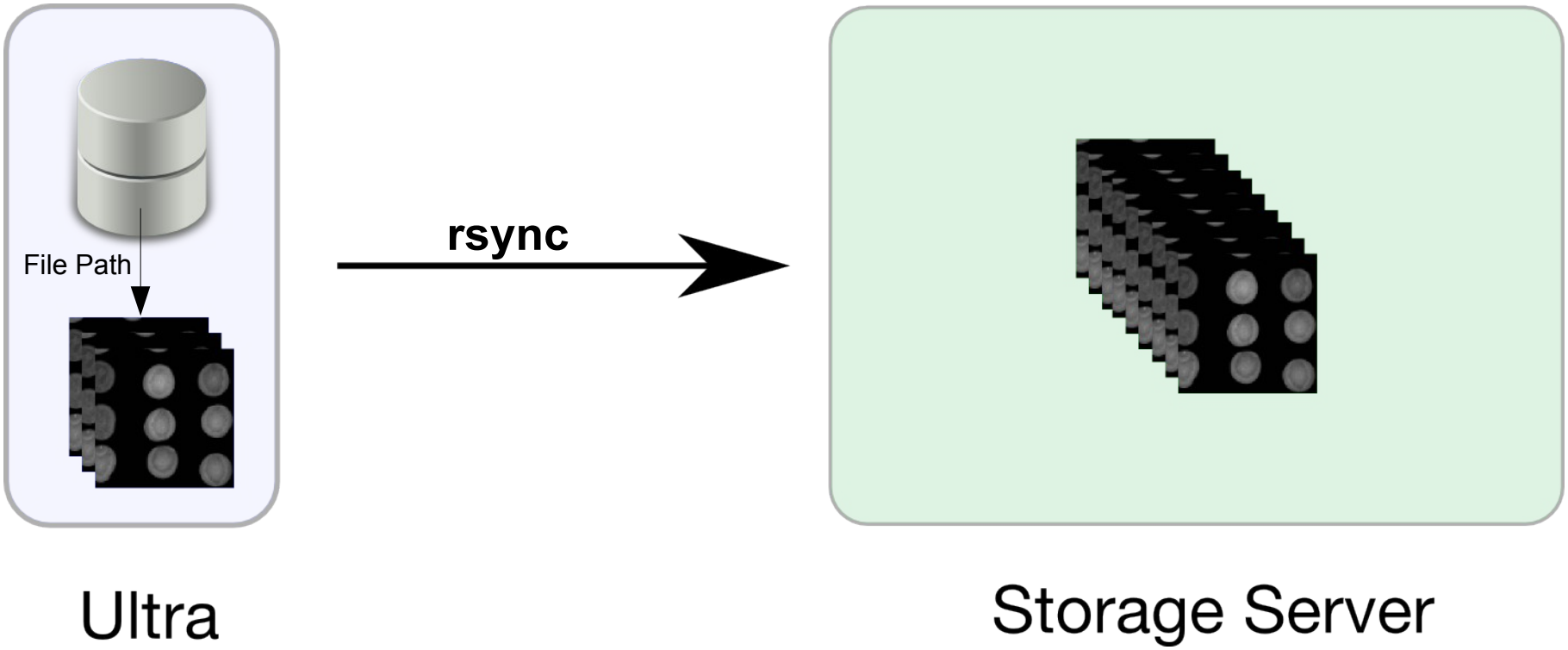


Image Data

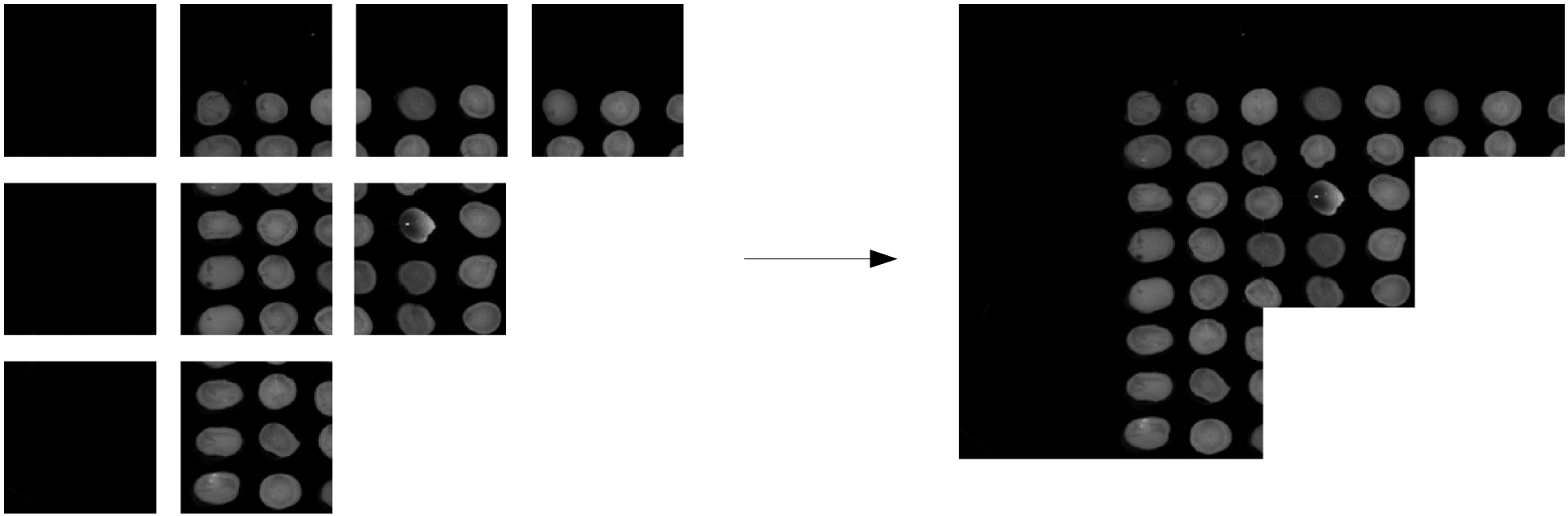
- Printed arrays on glass slide
- max. 2736 spots
- Scanning area
  - Sequentially captured image files
  - 1000 x 1000 Pixel
  - 16-bit uncompressed Tiff files
- Up to 80 000 images per scanning session
- Data size up to 160 GB

Image File Transfer



Montage of Individual Images

- Matlab script
  - create an array montage from individual images



Re-boxed Spot Image Files

- Matlab script
  - Each spot is re-boxed into a single image file
    - 16-bit uncompressed TIFF files
  - All files are save in single sub-directory
  - All EXIF data is preserved

Import Into OMERO

- In a perfect world

```
#omero_importer -u USERNAME \\
 -p PROJECT -d DATASET \\
 REBOXED_FILEPATH/*.TIF

#rm -rf REBOXED_FILEPATH
```

OMERO Importer Problems

- metaXpress files are not compatible
- No user specification w/o password
- No project or dataset generation
- Importer loads all images into memory
  - Out of memory error after 700MB of image files
- Sequential import is very slow

metaXpress Files - OMERO Compatibility

- Import of existing metaXpress TIFF files
  - Import image data only
  - Add meta-data
    - Direct database access
    - Call an OMERO script on dataset
  - Not standard compliant
  - Possible timing and or locking problems
  - Difficult to handle errors

metaXpress Files - OMERO Compatibility

- Conversion of metaXpress TIFF files
  - Conversion of existing image files to an OMERO compatible file format
    - Combine image and meta-data into a single image file
  - Import compatible files using the standard importer application
  - Standard compliant
  - Meta-data is included in every image file
  - Not locking/timing problems

metaXpress Image Data Format

- TIFF files are stored in single directory
  - Image meta-data in image header
  - Experiment meta-data in database

```
Exif.Image.ImageDescription                  Ascii      445  Experiment base
name:20101124  big head LamA 2 bin2x2
Experiment set:Optimization of encapsulation mix
XPO1
overlaid with ATCC Hela
for 48 hours
1 x confluent T25 around three million cells per array
standard fix and label methods
1 to 500 py
1 to 2000 2y
Pinhole diameter = 4.00
Beam splitter position = 405/488/561/635 Quad
Number of averages = 1
Binning = 2x2
561 nm laser at 35% power
PMT 3 at 350 gain
Plate ID:642
```

OME XML Conversion

- Extract and parse meta-data from Image header

```
#extract exif metadata into result hash
command = Escape.shell_command(["exiv2", '-pa', @options[:input_filename]])
result = %x[#{command}]
# check for errors of exiv2 command
# - exiv2 has an strange result code on exit (64768), not null. maybe a bug?
# - but conversion is ok. therefor we check for this exit code also
throw "exiv2 failed for image "+@options[:input_filename]+" Error: " + $?.to_s +
" Command: " + command unless $?.success? or $?.to_i == 64768
@metadata = Parse.exif_tags(result)

# analyze metaexpress flat file data and generate nice hash value => key pairs
@metaexpress_metadata = Parse.unformatted_metaexpress_metadata(@metadata["ImageDescription"])

# merge metadata from exif and metaexpress ImageDescription
@metadata.merge!(@metaexpress_metadata)
@metadata.delete(nil) #cleanup nil error
```

OME XML Conversion

- Extract and parse data from relavent databases

```
begin
  @omero_connection = DBI.connect("dbi:Pg:dbname="+omero_db[:dbname]+";host="+omero_db[:host],
                                  omero_db[:user],
                                  omero_db[:password])
  rescue DBI::DatabaseError => e
    throw "Connection to Omero DB failed | Error code: #{e.err} | Error message: #{e.errstr} Error SQLSTATE: #{e.state}"
end
```

```
def get_importer(experimenter)

  importer = {}

  begin
    importer["id"] = @omero_connection.select_all("SELECT id FROM experimenter WHERE omename='importer';")[0][0]
  rescue DBI::DatabaseError => e
    throw "Importer not found | Error code: #{e.err} | Error message: #{e.errstr} Error SQLSTATE: #{e.state}"
  end
  begin
    importer["group"] = @omero_connection.select_all("SELECT parent FROM groupexperimentermap WHERE child='#{importer["id"]}' limit 1")[0][0]
  rescue DBI::DatabaseError => e
    throw "Importer group not found | Error code: #{e.err} | Error message: #{e.errstr} Error SQLSTATE: #{e.state}"
  end

  importer["permissions"] = experimenter["permissions"]
  importer

end
```

Omero Import Options

- Command-line importer application
  - Can be automated
  - No project and dataset naming
  - No proper user assignment
  - **Slow as images are imported sequentially**

Parallel Import To OMERO

- All OMERO importers work sequentially
  - Import of 80 000 images takes > 24h
  - OMERO server uses
    - Single CPU for importer
    - Single CPU for server process
    - Sinlge CPU for database process

  => Start several import sessions on a multi-core system

Parallel Import To OMERO

- Import using 8 importers at the same time
- Parallel Import results is much faster
- 80 000 images could be imported in under 14 hours
- Problems:
  - Very CPU intensive
    - Single CPU for each importer importing process
    - Multiple database processes using 100% CPU time

Parallel Import To OMERO

- Parallel Import with outsourced OMERO database server is even faster
- 80 000 images could be imported in under 9 hours

- Outsourcing of importer processes should further speed up the data import

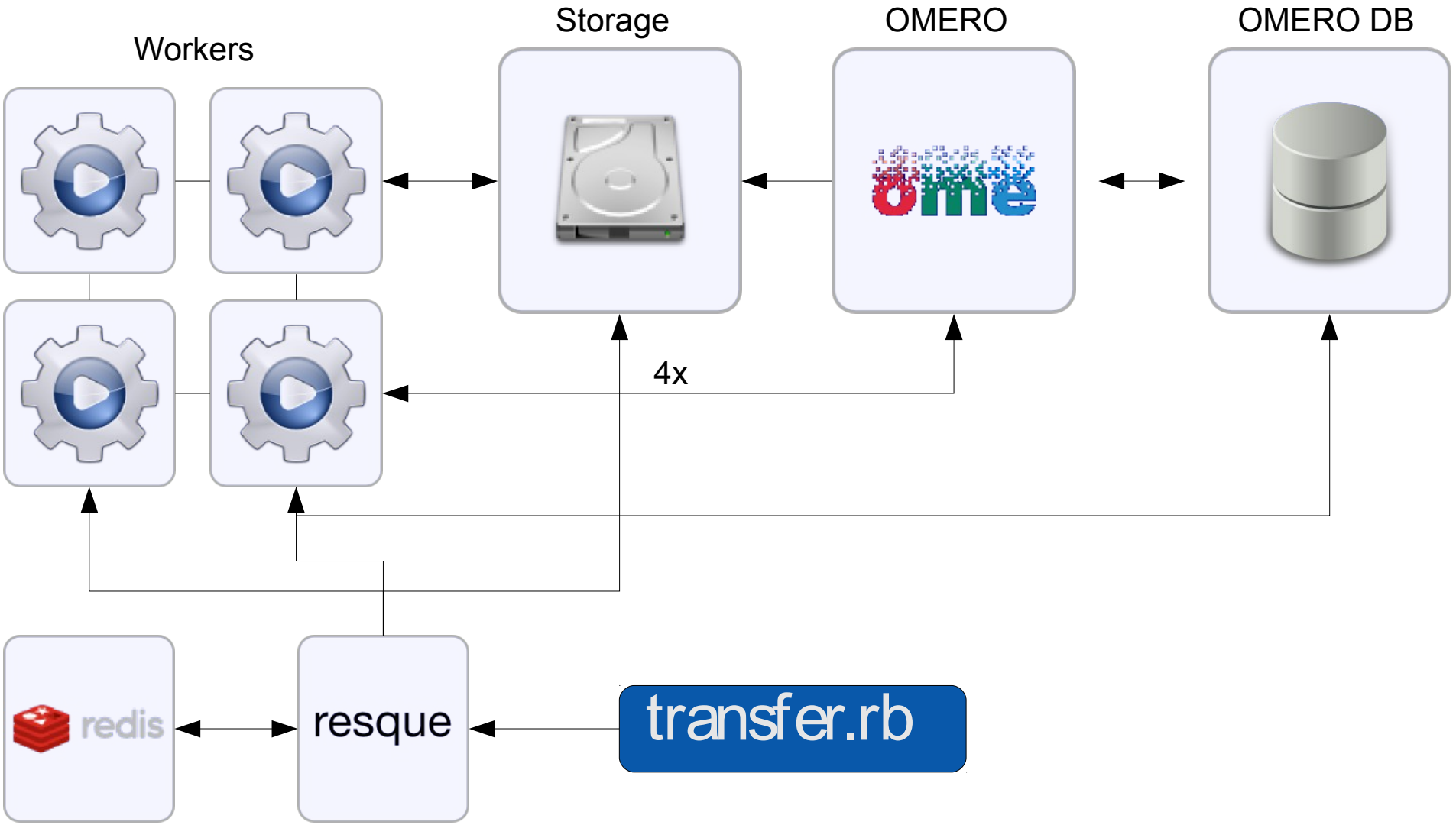
Final Import Work Flow

- Conversion of metaXpress TIFF to OME-TIF
- Import of OME-TIF in packages of 250 files
- Start multiple importers with one image packages each
- Import image files as the OMERO user “import”
- Reassign project, dataset and each image file

Job Scheduling System

- Resque
  - A Ruby library for creating, querying, and processing background jobs
  - Uses Redis as key-value storage backend
  - Starts Queues which span processes
    - Import image files
    - Database reassignment
  - Jobs can be distributed between multiple machines

Import Server Setup



Achieved Goals

- Automated import system
  - Flexible
  - Efficient
  - Scalable
  - Easy to administer
  - Expandable

To Do

- Work-flow adaptation
  - Include all relevant meta-data
- Better handling of image channels
  - integrate into OME-XML
- Performance fine tuning
- OMERO improvements
  - Support for metaXpress image data
  - Importer speed improvements

Plans for Pasteur Satelite

- future general direction of OMERO, streamlined image repository or image analytical platform?
- improvement of import speeds
- multi server setups for large user systems
- setup of complex sharing scenarios with a large number of users
- introduction of a quick sharing mode (no account on server but a disposable sharing link using an authentication token)