# **orbwfproc** and **dbwfproc** - Swiss army knives for continuous waveform processing

March, 2011

Antelope User Group Meeting

Bucharest, Romania

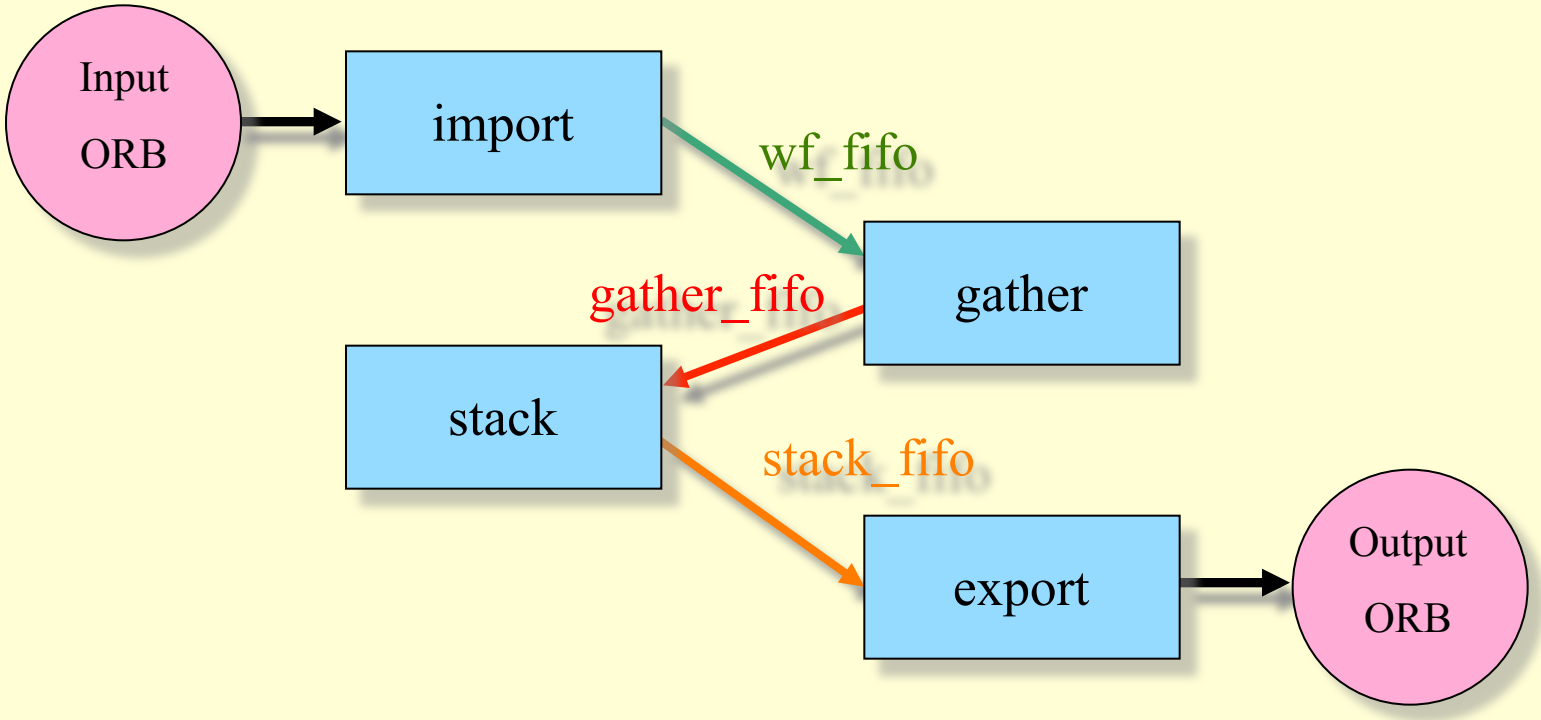
BRTT

BOULDER REAL TIME TECHNOLOGIES, INC.

# `orbwfproc` - a generalized waveform processor

- The continuous waveform processing analog to **`orbevproc`**
- **`orbwfproc`** runs a set of "tasks" continuously, each task in its own thread
- Tasks communicate with each other via a set of managed FIFO queues (described in **`pmtmanagedfifo(3)`**)
- Data objects passed between tasks are either task defined or are objects that can be represented by ORB packets, such as `PktChannel` waveform packets (see **`newPktChannel(3)`**)
- All tasks are currently written in **`c`**
- New tasks can be added by users as long as they conform to the task definition described in **`orbwfproc(5)`**
- Tasks distributed in 5.1 release are `import`, `export`, `inspector`, `wftest`, `wffilter`, `wfgather`, `wfstack`, `wfstats`
- Note that the internal representation of data flowing through the **`orbwfproc`** tasks is packet oriented instead of database oriented, as in **`orbevproc`**

# `orbwfproc` - array processing example



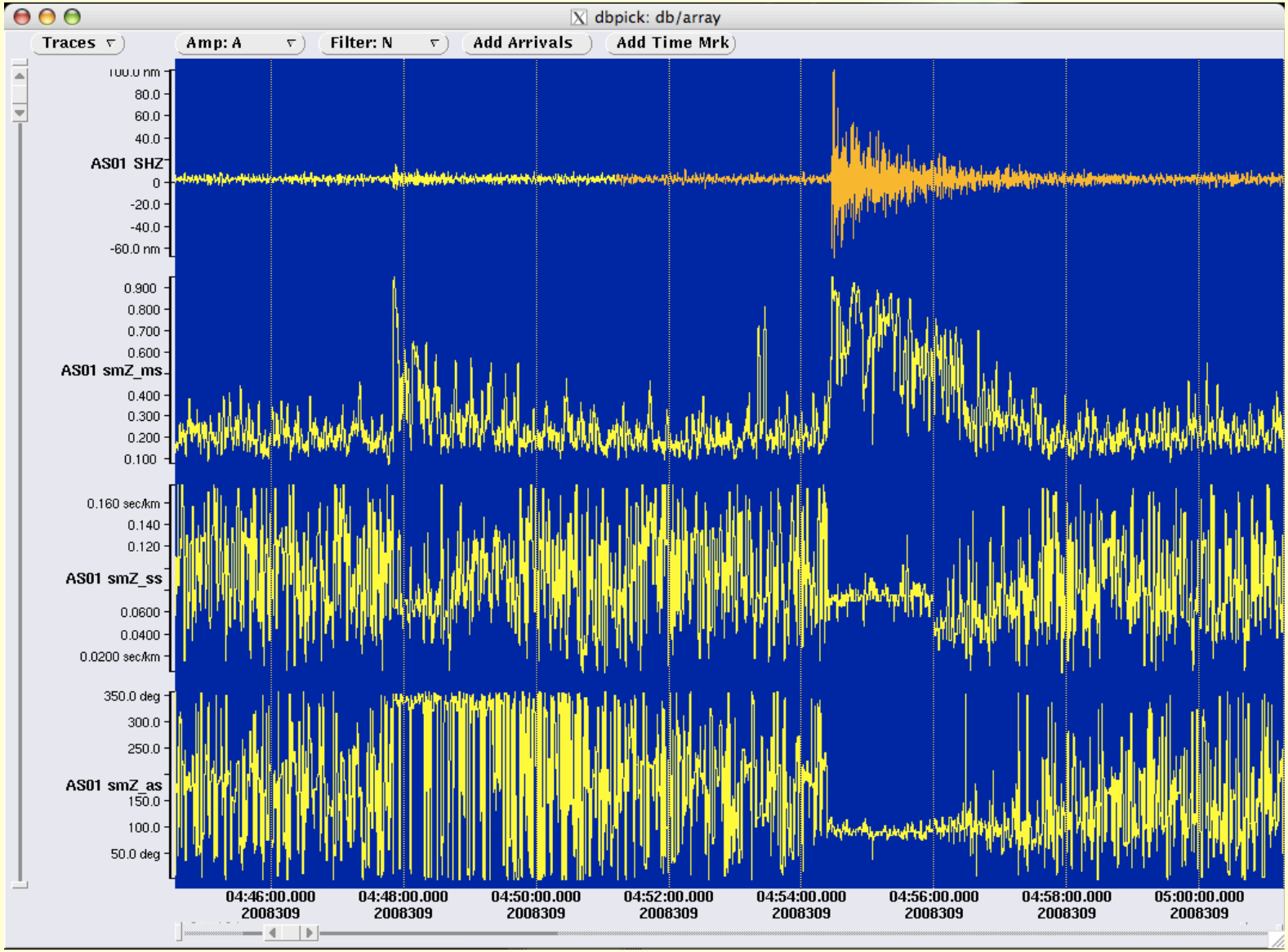
# orbwfproc - array processing example

```
#    This is the orbwfproc parameter file

#    This is the list of processing tasks to be run

wf_tasks &Tbl{
    #task_name  class_name  parameters          input       output
    import      import     import_params        -           wf_fifo     # import raw data from an ORB
    gather      wfgather   gather_params        wf_fifo     gather_fifo # form a gather
    stack       wfstack    stack_params         gather_fifo stack_fifo  # stack over a grid
    export      export     export_params        stack_fifo  -           # export the stack data to an ORB
}

#    These are parameters for each of the processing tasks

import_params &Arr{                 # parameters for import task
            input       orbdata                 # input from orbdata tag in command line
}

export_params &Arr{                 # parameters for export task
            output      orbout                  # output to orbout tag in command line
            subcode     STACK                   # specify a subcode for the output ORB packet srcnames
}

gather_params &Arr{                 # parameters for gather task
            . . .
}

stack_params &Arr{                  # parameters for stack task
            . . .
}
```

BRTT

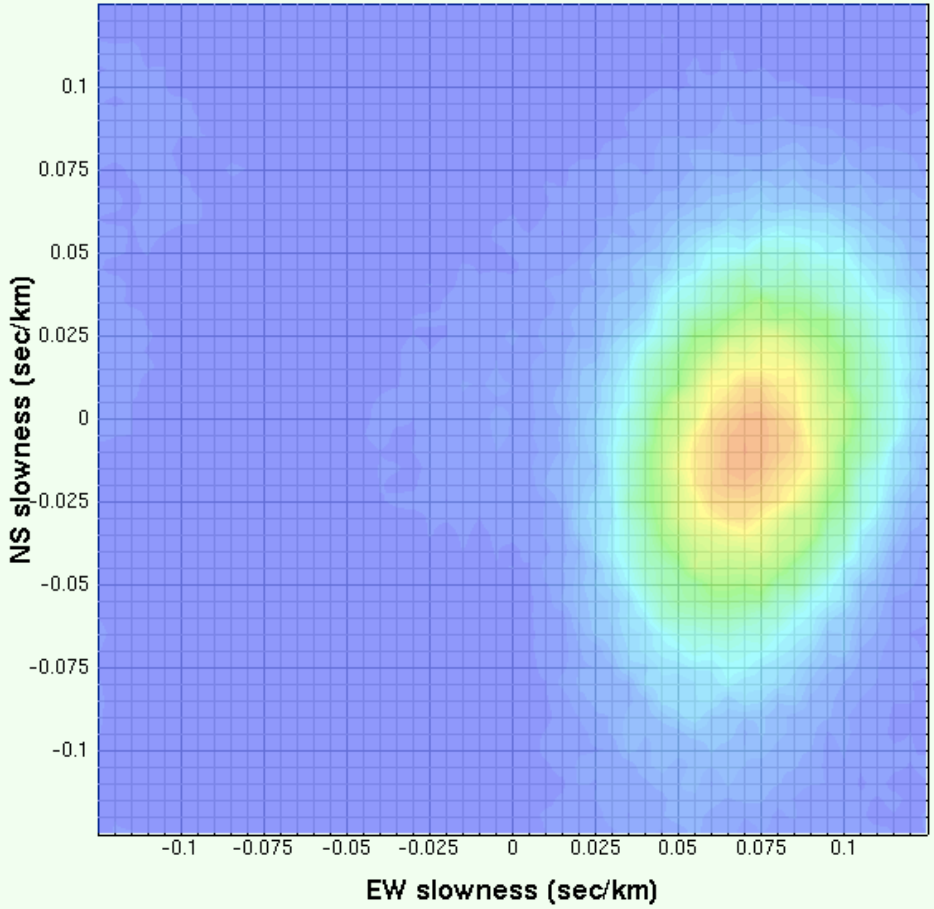
# What does this `orbwfproc` do?

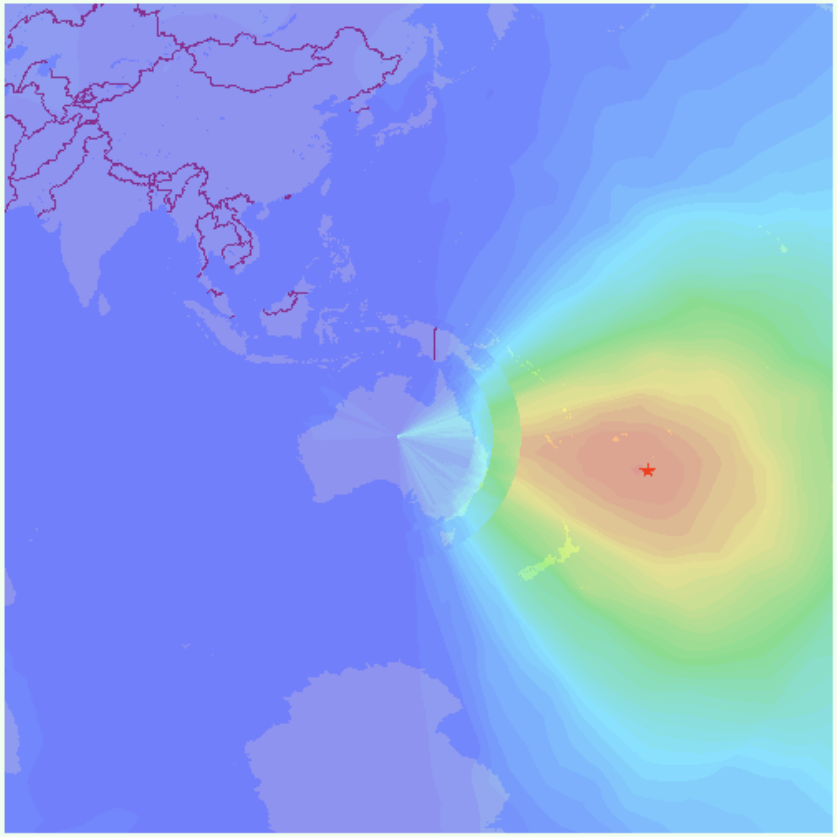
1. Import raw waveform data from real-time ORB (`import` task)
2. Form pre-filtered "gathers" of waveform channels for each array into regularized channel-sample grids (`gather` task)
3. Compute beams (stacks) over grids of horizontal vector slowness values assuming planar wavefield characteristics; this is done for each time sample over a grid of slowness values (`stack` task)
4. Also compute power averages for each beam at each time sample (power of the stack) plus power averages for each of the individual array channels plus an average of the individual array channel powers (stack of the powers) (`stack` task)
5. Also compute "semblance" by dividing the beam power averages by the stack of the individual channel power averages (`stack` task)
6. Scan the semblance grids for the slowness vector that corresponds to maximum semblance for each time sample (`stack` task)
7. Compute azimuth and scalar slowness corresponding to maximum semblance (`stack` task)
8. Export beams as waveforms plus maximum semblance, azimuth and scalar slowness as waveforms plus the semblance grids themselves into ORB output packets (`export` task)




BRTT

March 2011



BRTT

March 2011

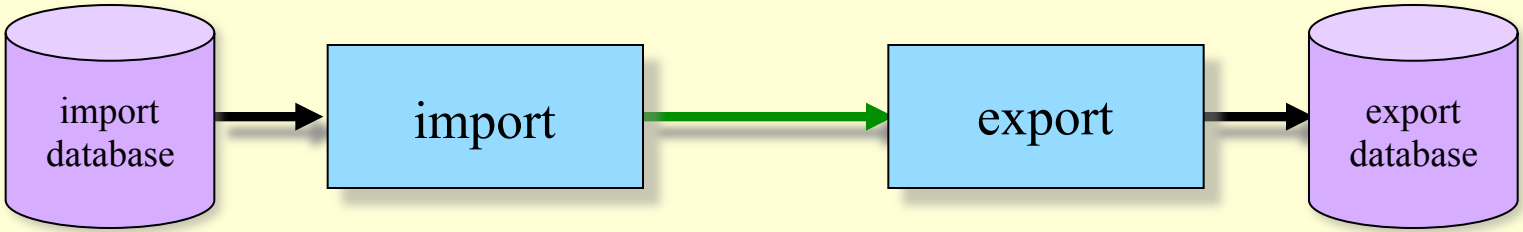
# Database import

- The `import` task can be configured to read waveform data from a Datascope database
- The Antelope **`pktchannel2trace(3)`** utility is used to read all waveform data for all channels over fixed time durations specified by the *`time_slice_duration`* parameter in the import task parameter file
- All of the channels for each time slice are then converted to `PktChannel` structures and put into the output FIFO
- All waveform sample data is represented in the `PktChannel` structures in floating point
- **`dbwfproc`** will automatically configure `import` tasks to read from databases
- **`orbwfproc`** can configure `import` tasks to read from databases by setting parameters in the import task parameter files

# Database export

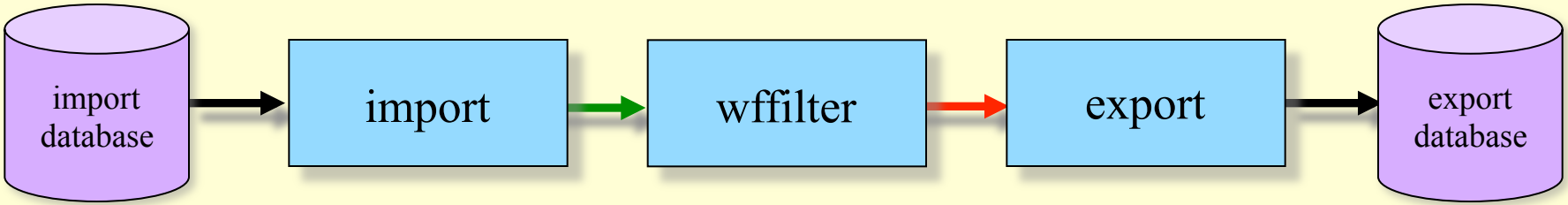
- The `export` task can be configured to write waveform data to a Datascope database
- The Antelope **`pktchannel2db(3)`** utility is used to write all waveform data for all channels into output databases
- The **`pktchannel2db(3)`** utility operates in a manner similar to **`cdorb2db(1)`** - waveform data is written into regular timing grids in fixed byte-per-sample formats and gaps are represented by special gap sample values
- The **`pktchannel2db(3)`** utility supports waveform output in both integer and floating formats. The utility also provides adaptive output buffering so that very latent data is heavily buffered to maximize performance whereas low latency data is not buffered at all to preserve minimum processing latency
- Note that `export` tasks do not support the output of waveform data in miniseed format
- **`dbwfproc`** will automatically configure `export` tasks to write to databases
- **`orbwfproc`** can configure `export` tasks to write to databases by setting parameters in the export task parameter files

# dbwfproc - waveform healing example




- This configuration will heal a highly fragmented import waveform database, with lots of wfdisc rows, data out of time order, overlapping data, gaps, etc., into an export database with one wfdisc row per channel per day.
- This is equivalent to running **dbreplay** into an **orbserver** and **cdorb2db** to produce the output database
- Conversion to miniseed would require running **db2mseed** as a post process, similar to **cdorb2db**

# `dbwfproc` - waveform resampling example




- This configuration can be used to resample waveform data or to filter waveform data in some other way and save the filtered data into an export database.
- The `export` task can be configured to morph the SEED codes so that the data written to the export database has different SEED codes than the import data

# Future developments for `orbwfproc`

- Systematic way of dealing with restarts from a state file
- Development of a `wfdetect` task class - this will require exporting of non-waveform data
- Development of `importsegment` task class used to import waveform segments based upon detections or arrivals
- Continued development of array processing tasks