Server-side Data Reduction and Analysis with Script Workflow Analysis for MultiProcessing (SWAMP)

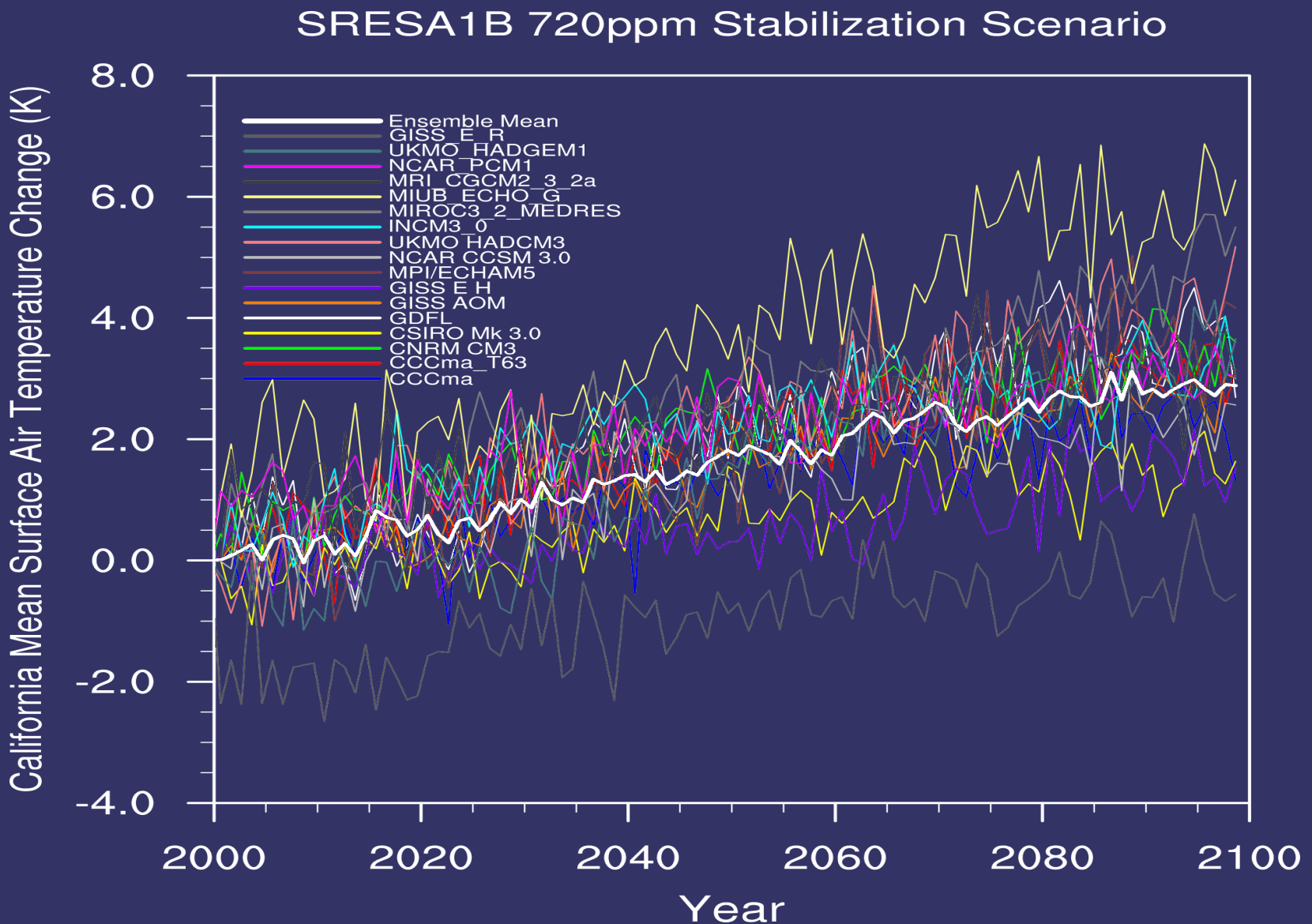
Daniel L. Wang, Charles S. Zender, and Stephen F. Jenks
University of California, Irvine
http://dust.ess.uci.edu/~wangd/pub/sld_WZJ07b.pdf

UCIrvine

{wangd, zender, sjenks}@uci.edu

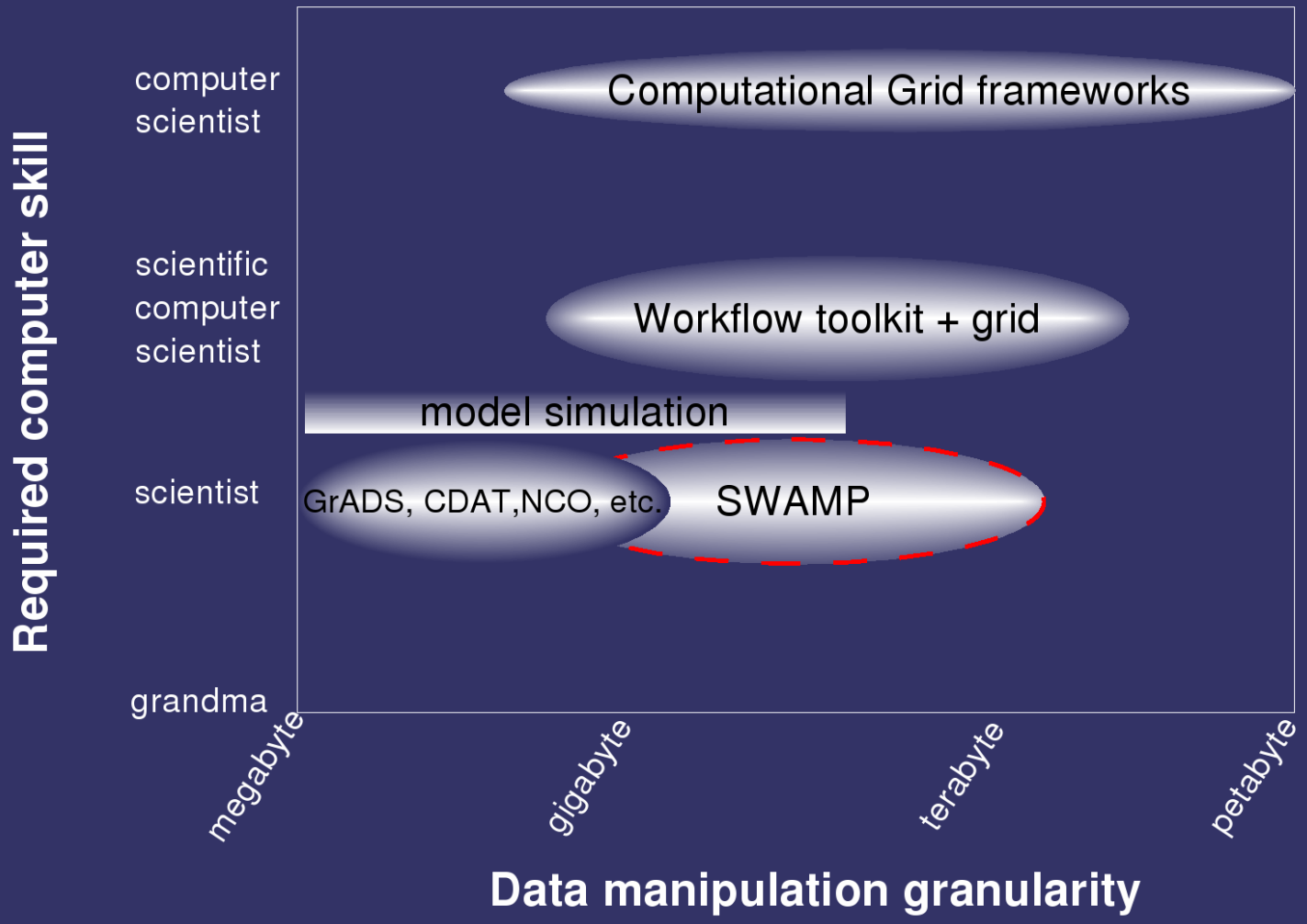
Motivation

- How can we make this task easy?

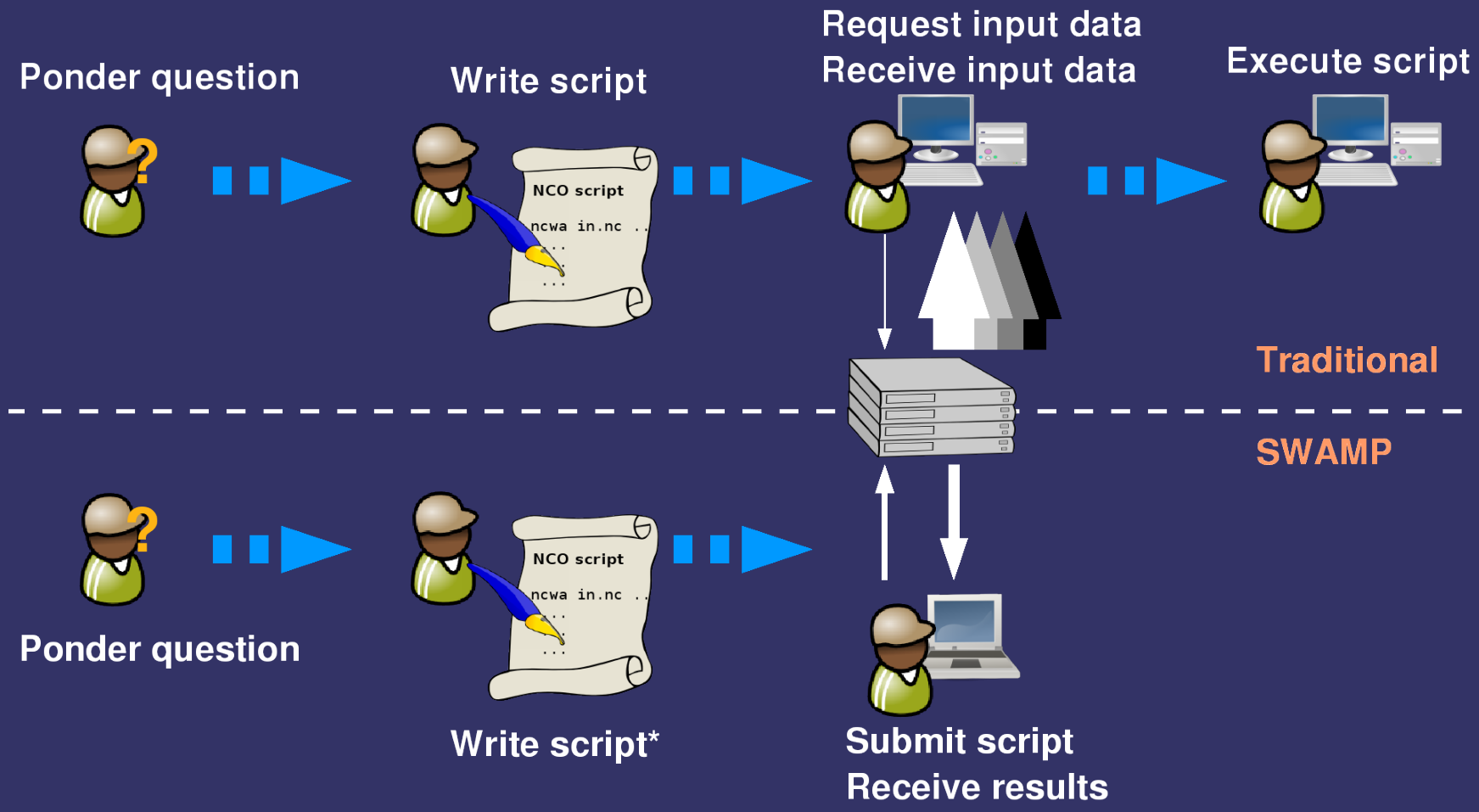


Problem

- Analysis lags simulation/generation
  - Bulk: 1TB of data is non-trivial to manage
  - Locality: Bandwidth ≪ ∞



Approach



- Appropriately distribute computation load
- Leverage existing scripted interfaces
- (Exploit available parallelism)

The "Normal" Way

```
models='cccma_cgcm3_1 cccma_cgcm3_1_t63 cnrm_cm3 csiro_mk3_0 \
       gfdl_cm2_0 gfdl_cm2_1 giss_aom giss_model_e_h giss_model_e_r \
       iap_fgoals1_0_g inmcm3_0 ipsl_cm4 miroc3_2_hires miroc3_2_medres \
       miub_echo_g mpi_echam5 mri_cgcm2_3_2a ncar_ccsm3_0 ncar_pcm1 \
       ukmo_hadcm3 ukmo_hadgem1'
variables='tas pr'
scenarios='sresa1b sresa2 sresb1'

for scn in $scenarios; do

   for mdl in $models; do

     ncwa -O -v $variables -w area -a lat,lon \
       -p http://user:password@climate.llnl.gov/cgi-bin/dap-cgi.py/ipcc4/$scn/$mdl \
       pcmdi.ipcc4.$mdl.$scn.run1.atm.mo.xml $scn_$mdl_200001_209912.nc
     ncwa -F -d time,1,12 $scn_$mdl_200001_209912.nc $scn_$mdl_2000.nc
     ncdiff $scn_$mdl_200001_209912.nc $scn_$mdl_2000.nc $scn_$mdl_anm.nc

   done # end loop over model

   ncea *_200001_209912.nc $scn_avg_200001_209912.nc
   ncwa -F -d time,1,12 $scn_avg_200001_209912.nc $scn_avg_2000.nc
   ncdiff $scn_avg_200001_209912.nc $scn_avg_2000.nc $scn_avg_anm.nc

done # end loop over scenario
```

SWAMP

Script Workflow Analysis for Multiprocessing

- Specialized netCDF data handler for OPeNDAP
  - Use constraint expression to signal computation
- ~sh-script syntax accepts netCDF Operators (NCO) command lines
  - `ncwa -v WIND -d time %tempf_cam1999.nc%\ %outf_winds1999.nc%`
- Server-side parallelizing and optimizing execution engine

Engine implementation

- Replaces the netCDF data handler (dap_nc_handler)
  - Passes non-SWAMP requests to original
- Written in python

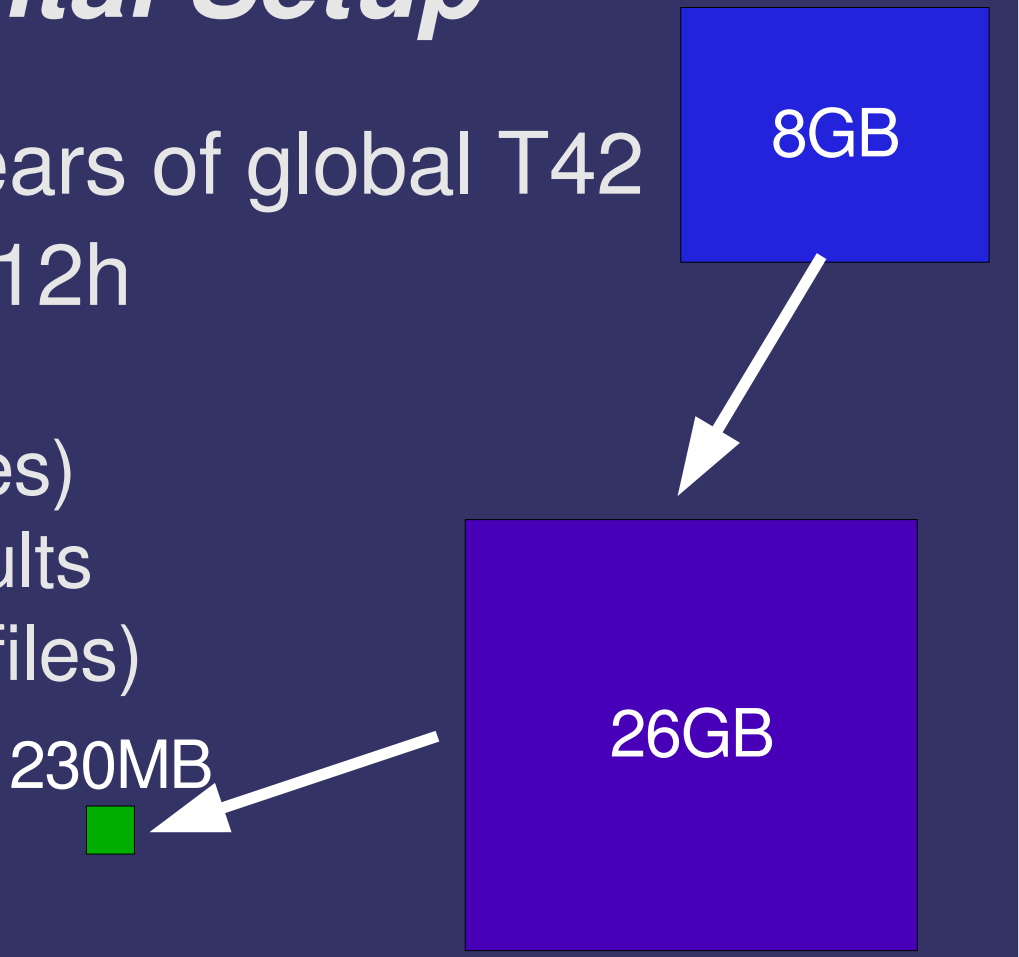
- Parses script to generate workflow
- Locates temporary files in ramdisk, 'live file' analysis to reclaim space
- “Persistent” job tracking via database
  - SQLite for speed and simplicity

Engine Parallelization

- Peer worker parallelization model
  - server-configurable # of slots
  - shared state via DB updates
  - ready list, file status, command state
- Script-line-level parallelization
  - thread-level built-in to NCO (OpenMP)
- I/O concurrency a problem for n>2
  - Use Linux tmpfs ramdisk

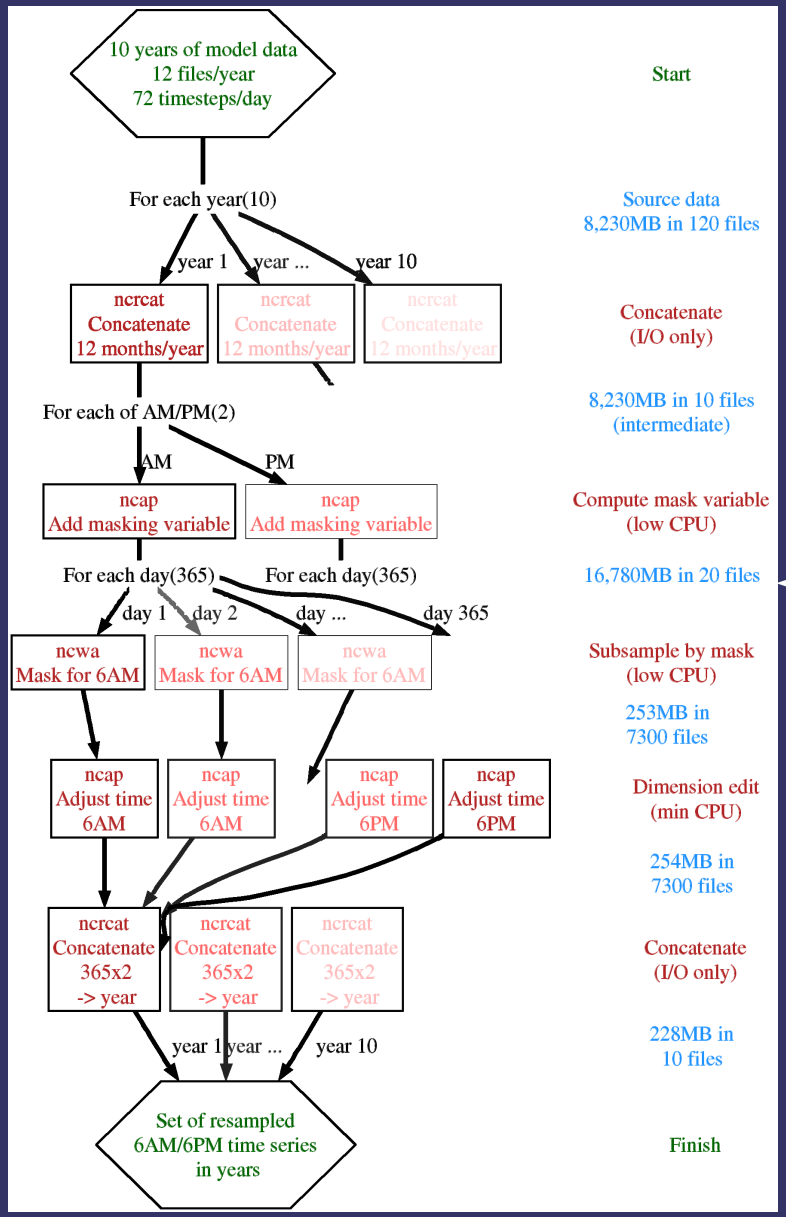
Experimental Setup

- Case: Subsample 10 years of global T42 data at Δt=20m into Δt=12h
  - ~ 14,000 script lines
  - ~ 8GB input data (120 files)
  - ~ 26GB intermediate results
  - ~ 230MB result data (10 files)

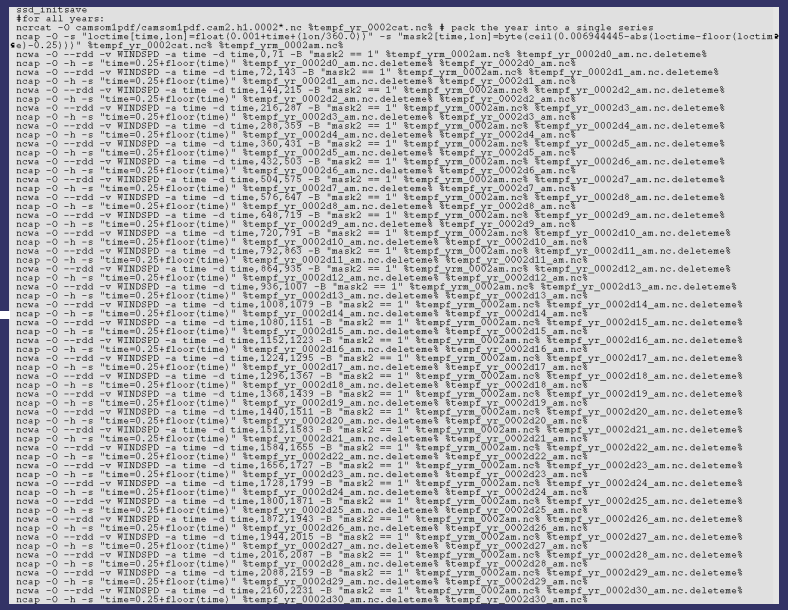


- System:
  - Dual Opteron 270 (4 cores total), 16GB memory
- Compare 9 cases:
  - Traditional vs. SWAMP (1,2,4,8-wide, I/O opt)

SWAMP Parses, Optimizes and Schedules

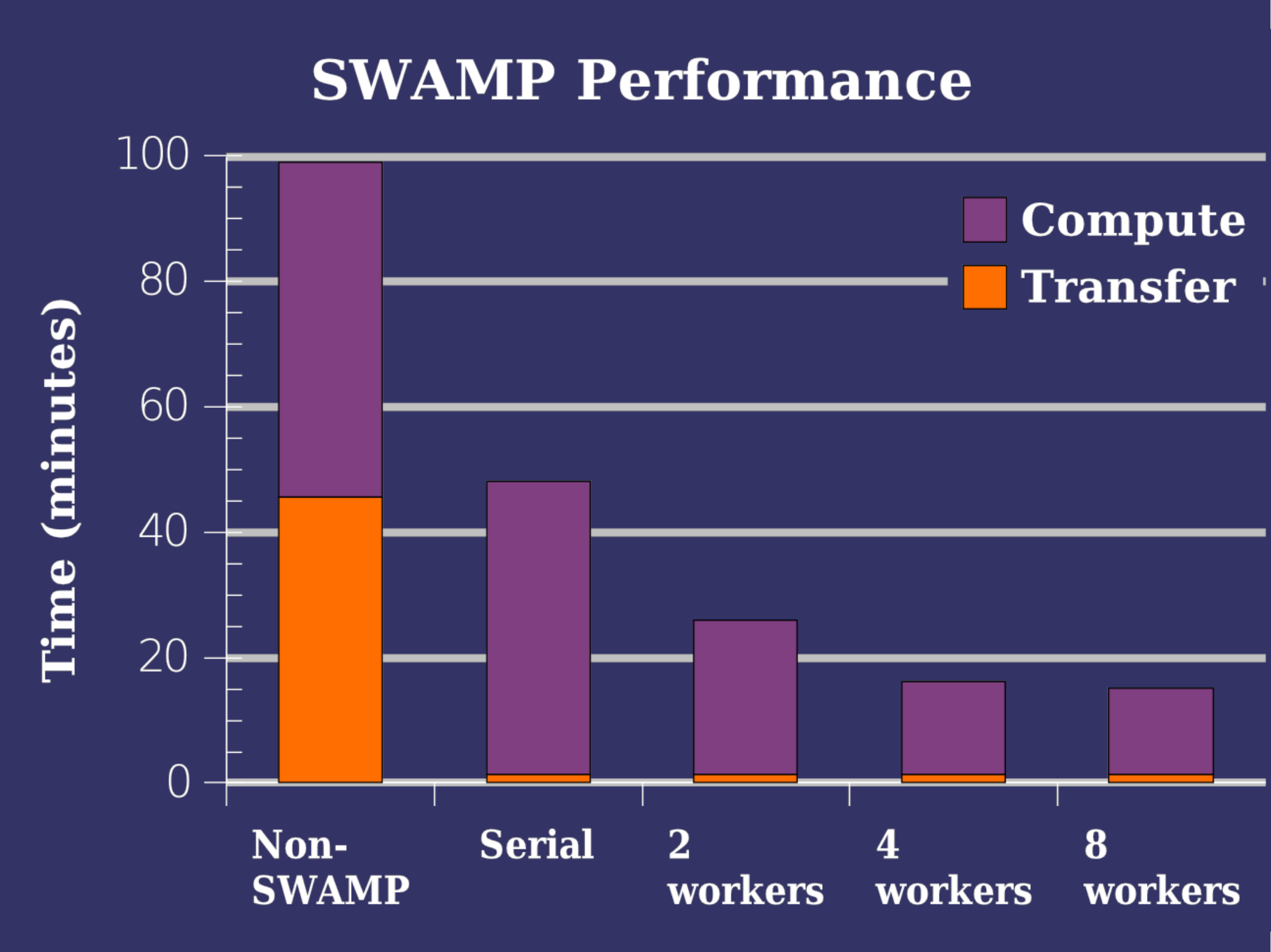


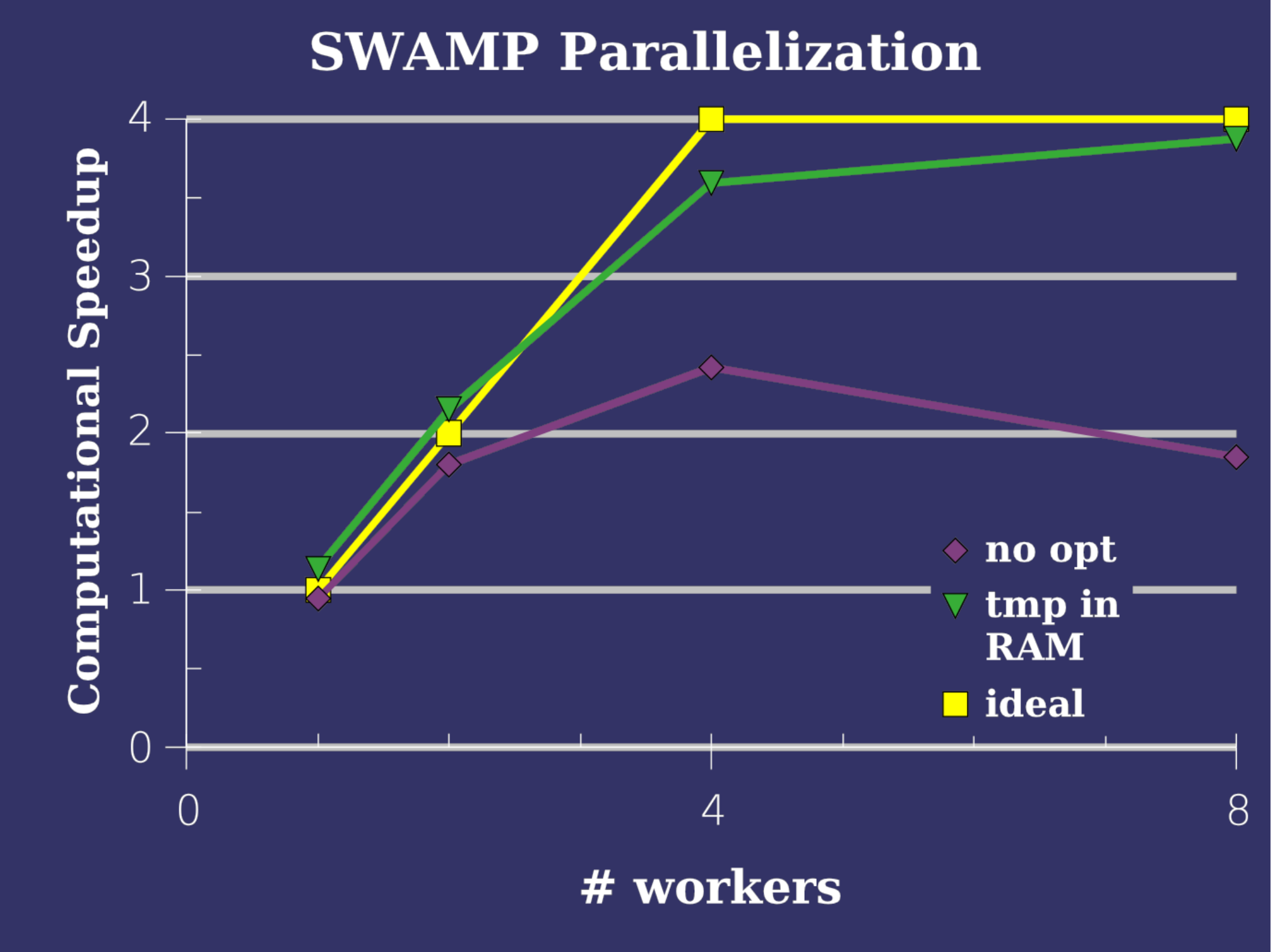
dependency-aware workflow



~14,000 line script

1 line ~ 1 NCO command


SWAMP Performance
Time (minutes)
100
80
60
40
20
0
Compute
Transfer
Non-SWAMP
Serial
2 workers
4 workers
8 workers


SWAMP Parallelization
Computational Speedup
0
1
2
3
4
0
4
8
workers
no opt
tmp in RAM
ideal

Conclusions

- High computational performance potential with tested workflow
  - Parallelism
  - Optimization
- Large speedup through bandwidth optimization
- Viability of shell-script interface
- Room for optimization above existing NCO optimization

Would you like to know more?

Questions?

- offline: wangd@uci.edu

Want to poke at the code?

(please be gentle)

- See NCO CVS source tree on sourceforge: (nco/src/ssdap)

Want to try it?

- Probably need my help to make it work