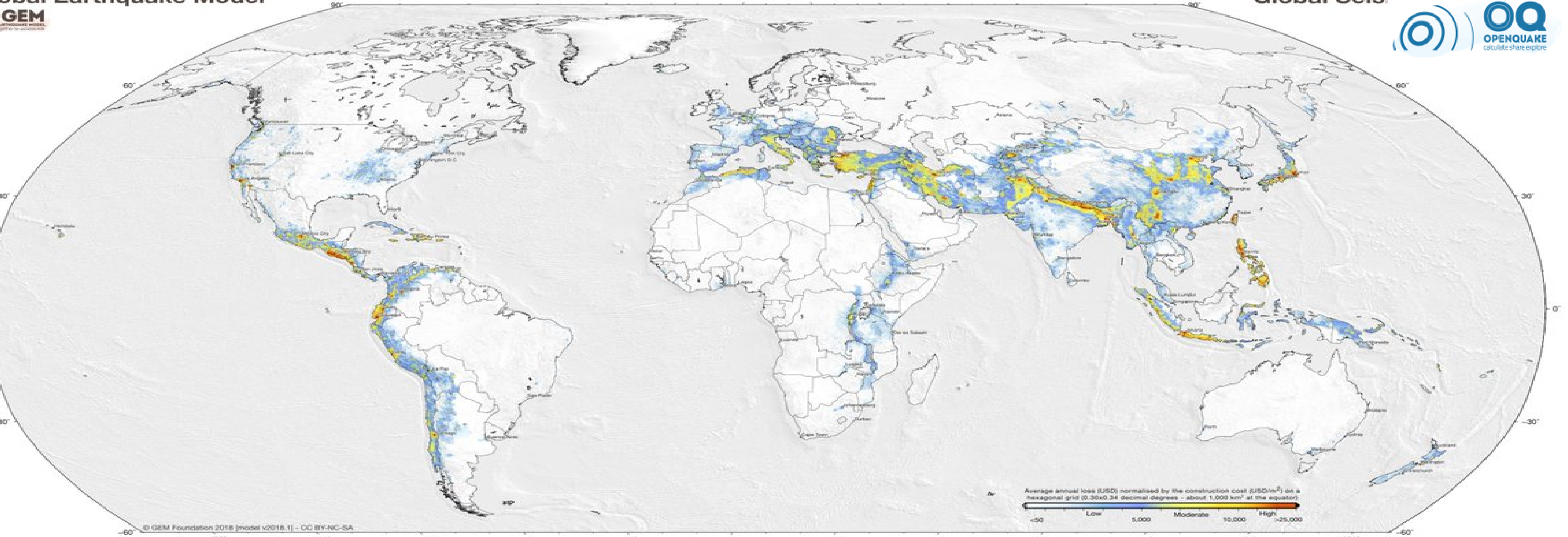
Tips for the Scientific Programmer

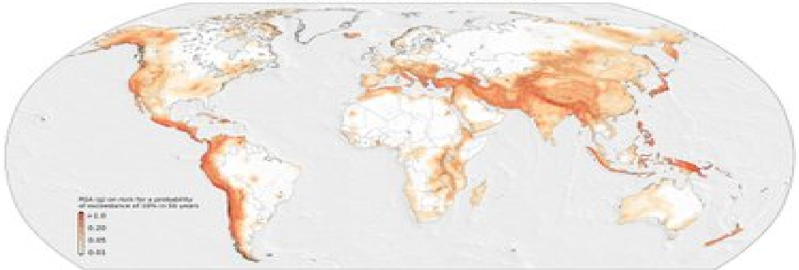
Michele Simionato@GEM Foundation

Global Earthquake Model

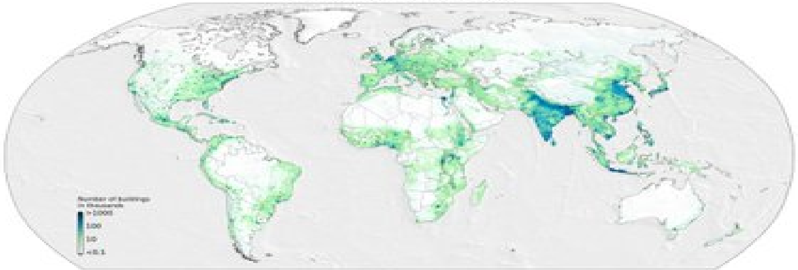
Global Seis



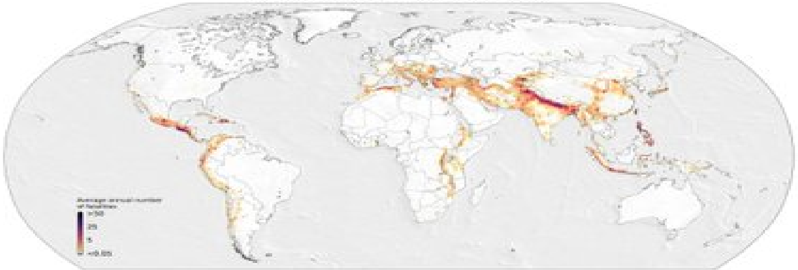
Global Seismic Hazard Map



Global Exposure Map



Global Seismic Fatalities Map



Global Earthquake Model (GEM) Global Seismic Risk Map

The Global Seismic Risk Map (v2018.1) comprises four global maps. The main map presents the geographic distribution of average annual loss (USD) normalised by the average construction costs of the respective country (USD/m²) due to ground shaking in the residential, commercial and industrial building stock, considering contents, structural and non-structural components. The normalised metric allows a direct comparison of the risk between countries with widely different construction costs. It does not consider the effects of tsunamis, liquefaction, landslides, and fires following earthquakes. The loss estimates are from direct physical damage to buildings due to shaking, and thus damage to infrastructure or indirect losses due to business interruption are not included. The Global Seismic Hazard Map depicts the geographic distribution of the Peak Ground Acceleration (PGA) with a 10% probability of being exceeded in 50 years, computed for reference rock conditions (shear wave velocity of 760-800 m/s). The Global Exposure Map depicts the geographic distribution of residential, commercial and industrial buildings. The Global Seismic Fatalities Map depicts an estimate of average annual human losses due to earthquake-induced structural collapse of buildings. The results for human losses do not consider indirect fatalities such as those from post-earthquake epidemics. The average annual losses and number of buildings are presented on a hexagonal grid, with a spacing of 0.30 x 0.34 decimal degrees (approximately 1,000 km² at the equator). The average annual losses were computed using the event-based calculator of the OpenQuake engine, an open-source software for seismic hazard and risk analysis developed by the GEM Foundation. The seismic hazard, exposure and vulnerability models employed in these calculations were provided by national institutions, or developed within the scope of regional programs or bilateral collaborations. These global maps and the underlying databases are based on best available and publicly accessible datasets and models. Due to possible model limitations, regions portrayed with low risk may still experience potentially damaging earthquakes. The GEM Risk Map is intended to be a dynamic product, such that it may be updated when new datasets and models become available. Releases of updated versions of the seismic risk map are anticipated on a regular basis. Additional hazard and risk metrics for each country can be explored at globalquakemodel.org/gem.

The Global Earthquake Model (GEM) Foundation

The Earthquake Risk Map 2018 is a product of the GEM Foundation. Initiated by the Organisation for Economic Co-operation and Development (OECD) Global Science Forum in 2006, GEM was formed in 2009 as a non-profit foundation in Pavia (Italy), funded through a public-private sponsorship with the vision to create a world that is resilient to earthquakes. Participants represent national research or disaster management institutions, the private sector and international organisations. GEM expands the assessment of seismic hazard at the global scale initially started by the Global Seismic Hazard Assessment Program (GSHAP) in support of the UN International Decade of Natural Disaster Reduction in 1999 to the consideration of direct economic and human losses. Observing its core values of collaboration, transparency, openness, credibility, and serving the public good, GEM goes beyond GSHAP by extending the scope of work to the risk domain, providing an institutional framework for continuous updates, and fostering direct applications to risk reduction and prevention projects. GEM's collaborative network comprises more than 70 public and private institutions organised under more than 25 regional, national and multilateral projects.

GEM's OpenQuake platform (platform.openquake.org) provides access to all data, models, tools and software behind the maps. GEM's open-source OpenQuake engine enables probabilistic hazard and risk calculations worldwide and at all scales, from global down to regional, national, local, and site-specific in a single software package. The Sendai Framework for Disaster Risk Reduction (SFDRR) calls for "decision-making on disaster risk reduction to be based on solid and openly accessible scientific work". GEM supports the SFDRR goals by contributing openly accessible products for hazard and risk assessment and capacity development in risk reduction projects. GEM also serves as a baseline or exemplar for the development of a broader multi-hazard framework for risk assessment in support of a holistic and comprehensive approach to disaster risk reduction. Technical details on the development and compilation of the hazard and risk maps, underlying models and the list of contributors can be found at globalquakemodel.org/gem.

How to use and cite this work

Please cite this work as: V. Silva, D. Amo-Oduro, A. Calderon, J. Dabbeek, V. Despotaki, L. Martins, A. Rao, M. Simionato, D. Viganò, C. Yepes-Estrada, A. Acevedo, H. Crowley, N. Horspool, K. Jaiswal, M. Journeay, M. Pittore (2018). Global Earthquake Model (GEM) Seismic Risk Map (version 2018.1). DOI: 10.13117/GEM-GLOBAL-SEISMIC-RISK-MAP-2018.

Acknowledgements

This map is the result of a collaborative effort and extensively relies on the enthusiasm and commitment of various organisations to openly share and collaborate. The creation of this map would not have been possible without the support provided by several public and private organisations during GEM's second working programme (2014-2018). None of this would have been possible without the extensive support of all GEM Secretariat staff. These key contributions are profoundly acknowledged. A complete list of the contributors can be found at globalquakemodel.org/gem.

Legal statements

Contacts

GEM (Global Earthquake Model) Foundation
Via Ferrata, 1 – 27100, Pavia, Italy
info@globalquakemodel.org

More information available at globalquakemodel.org/gem

Sponsors and major contributors

This talk is about "Middle Performance Computing"

- profiling is invaluable for finding bottlenecks like slow operations in inner loops, but I do that 1-2 times per year
- what it is really essential is instrumenting your code
- what makes the difference is using the right library and the right architecture / data structure

Input/output formats

- I learned the hard way a very essential lesson: *never, EVER change the input formats*
- You cannot. Really, you can not.
- Even if it is impossible to get right the input format at the beginning ☹
- There is more freedom with the output formats
- Where you can really work is on the internal formats

Inputs formats we are using

- INI (good, but TOML would have been better)
- XML/NRML/XSD (could have been simpler)
- CSV (should have been used more)
- HDF5 (in rare cases: UCERF3, GMPE tables)
- ZIP (okay)

Output formats we are using

- XML / NRML: we are removing it
- CSV with pre-header: we are using it more and more
- HDF5: used sometimes
- NPZ: by necessity

Outputs from calculation 24329

ID	Name	Type	Action
32393	Full Report	fullreport	Download rst
32394	Hazard Curves	hcurves	Download csv Download xml Download npz
32395	Hazard Maps	hmaps	Download csv Download xml Download npz
32396	Input Files	input	Download zip
32397	Realizations	realizations	Download csv
32398	Seismic Source Groups	sourcegroups	Download csv
32399	Uniform Hazard Spectra	uhs	Download csv Download xml Download npz

Download hdf5 datastore

Internal formats we are using

- .hdf5
- .toml
- .sqlite

They are good 👍

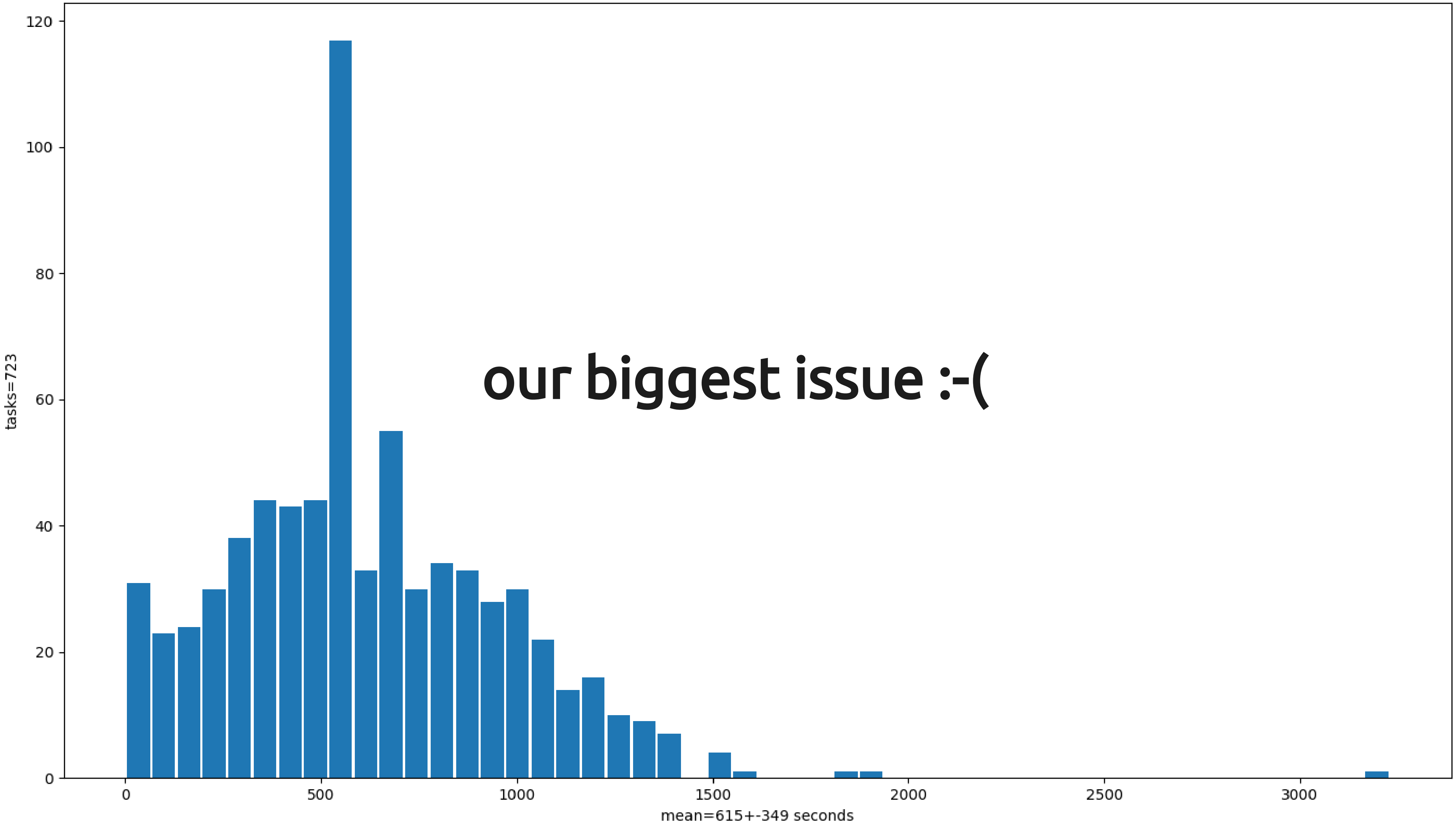
The choice of the data format has a big performance impact

- XML/CSV exporters
- XML/CSV importers
- clearly the choice of the internal formats is even more important: HDF5 is the way to go

Task distribution

- we are using *multiprocessing/zmq* on a single machine
- and *celery/rabbitmq/zmq* on a cluster

- celery/rabbitmq is not ideal for our use case but it works enough, including the REVOKE functionality



our biggest issue :-(

Slow tasks

- slow tasks have been a PITA for years ☹
- a few months ago we had a breakthrough: subtasks
- we made the output receiver able to recognize tuples of the form `(callable, arg1, arg2, ...)` and to send them as tasks

- task producing subtasks:

```
def task_splitter(sources, arg1, arg2, ...):
  blocks = split_in_blocks(sources, maxweight)
  for block in blocks[:-1]:
       yield (task_func, block, arg1, arg2, ...)
  yield task_func(block[-1], arg1, arg2, ...)
```

- heavy tasks can be split in many light tasks
- the weight of a seismic source is the number of earthquakes it can produce
- it can be *very* different from the duration of the calculation

Calibrating the computation

- we introduced a task splitter able to perform a subset of the calculation and to estimate the expected task duration depending on the weight
- it can split the calculation in subtasks with estimated runtime smaller that an user-given `task_duration` parameter

Automatic task splitting

- successively, we made the engine smart enough to determine a sensible default for the `task_duration`, depending on the number of ruptures, sites and levels
- => slow tasks are greatly reduced
- except for non-splittable sources

Solving the data transfer issue

- we switched to using zmq to return the outputs 👍
- we switched to NFS to read the inputs (and it is also useful for sharing the code)
- **important**: do not produce too many tasks, the data transfer will kill you, or the output queue will run out of memory, or both

Memory occupation

- a big problem we had to fight constantly is running out of memory (even with 1280 GB split on 10 machines)
- notice that running out of memory *early* can be a good thing
- it is all about the tradeoff memory/speed
- NB: memory allocation can be the *dominating* factor for performance

How to reduce the required memory

- use as much as possible numpy arrays instead of Python objects
- use a site-by-site algorithm if you really must
- remember that big tasks are still better, if you have enough memory
- we measure the memory with `psutil.Process(pid).memory_info()`

Saving memory by yielding partial results

```
def big_task(sources, arg1, arg2, ...):
    accum = []
    for src in sources:
        accum.append(process(src, arg1, arg2, ...)
        if len(accum) > max_size:
            yield accum
            accum.clear()  # save memory
    if accum:
        yield accum
```

Lesson: a nice parallelization framework really helps


Questions?