BEADD

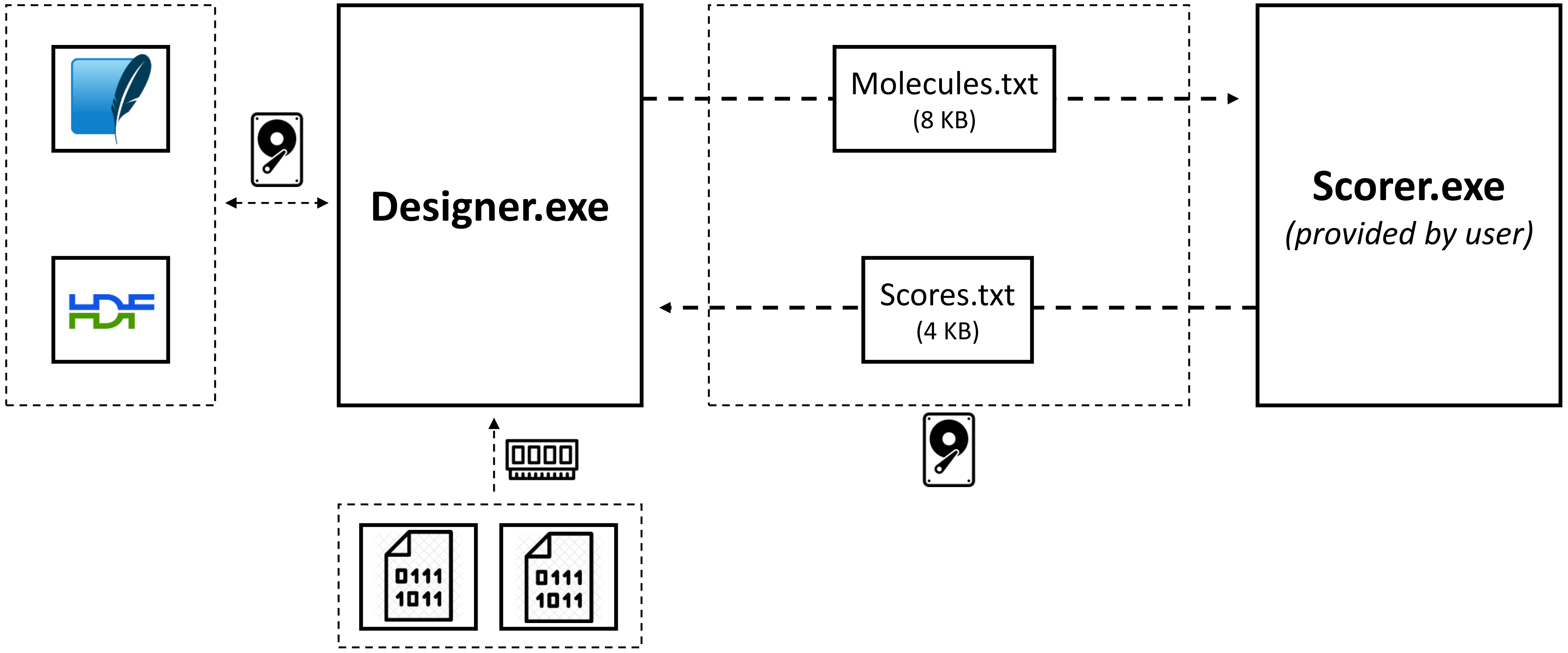
Alan Kerstjens

What is it?

- Evolutionary algorithm to design/optimize molecules
- Written in C++
- Optimality measured by a user-specified objective function

$$\begin{tikzcd}
\textbf{Design} \arrow[r, bend left] & \textbf{Score} \arrow[l, bend left]
\end{tikzcd}$$

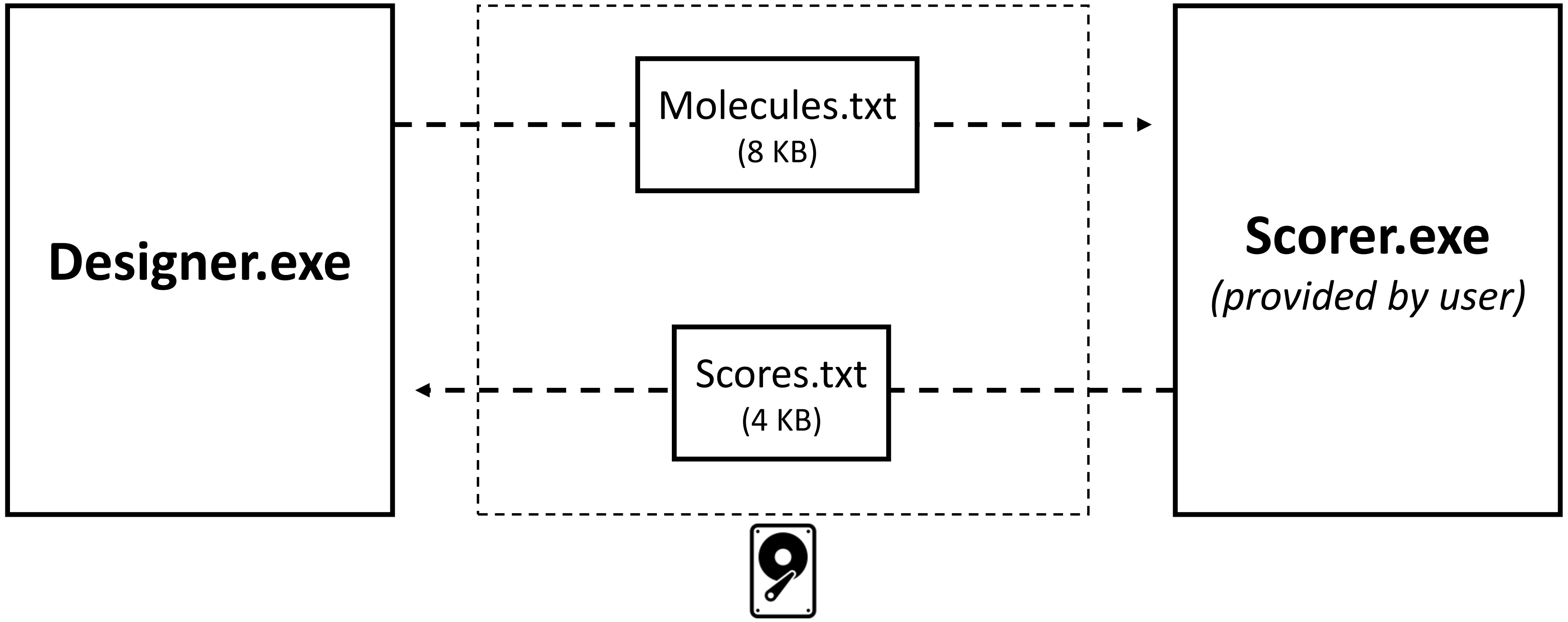
Program structure: production



Program structure: production

When scoring is necessary:

1. Designer.exe writes molecules to Molecules.txt
2. Designer.exe spawns Scorer.exe and goes to sleep
3. Scorer.exe reads Molecules.txt and scores them
4. Scorer.exe writes scores to Scores.txt
5. Scorer.exe terminates and Designer.exe resumes

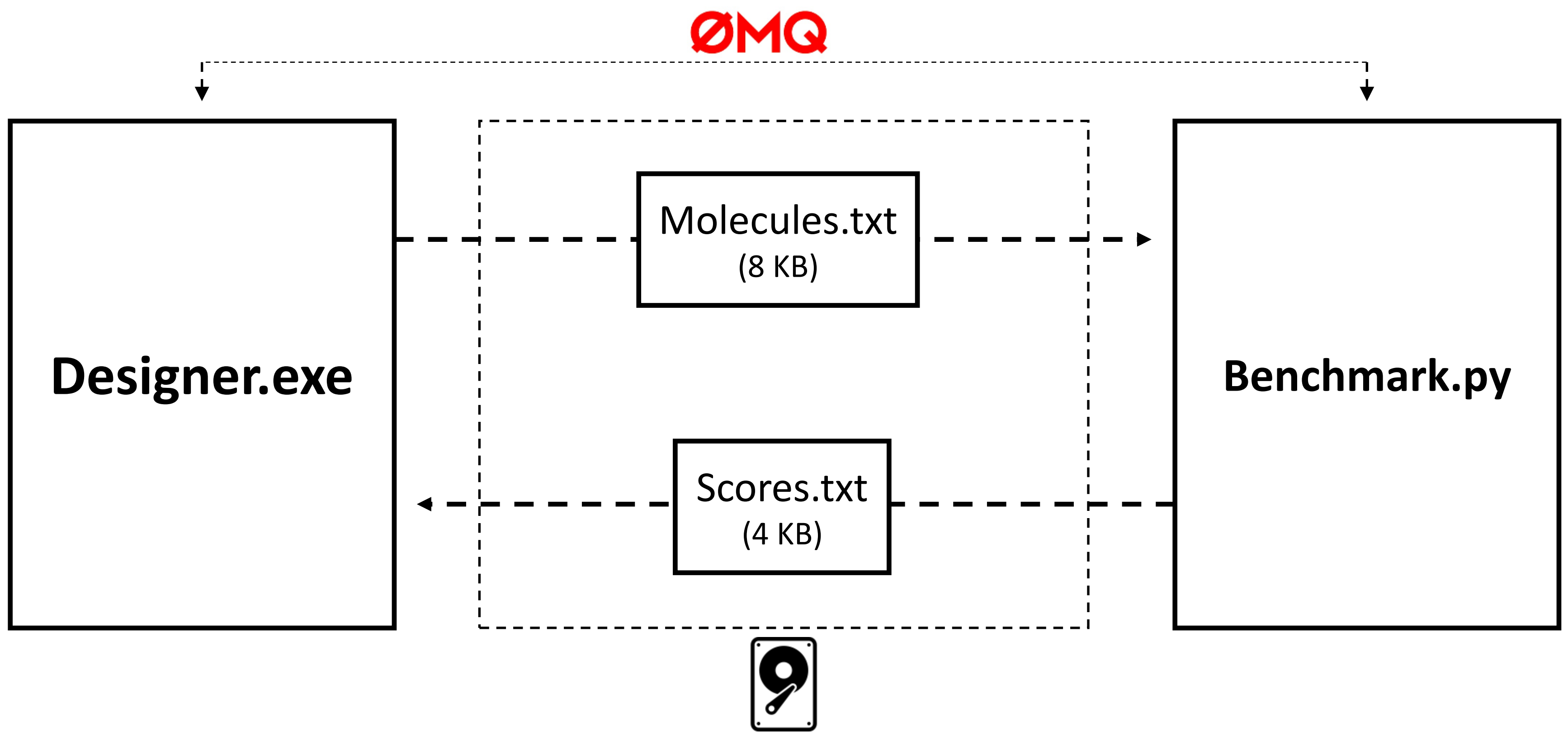


Benchmark

- Third-party benchmark to evaluate molecular design algorithms
- Includes standardized scoring functions
- 1 benchmark suite → 20 individual tests
- Multiple replicas
- Multiple settings

Program structure: benchmark

- Benchmark.py is the driver script (Python script)
- Benchmark.py is the scoring function
- For each test, Benchmark.py invokes Designer.exe
- Each test involves multiple generations (multiple exchanges of data)
- Message passing library used to notify executables of when they can proceed



How to reduce load on file system?

- /dev/shm?
- For the benchmark: serialized data transfer between C++ and Python?
- Is it worthwhile?

How to best parallelize it?

Current approach:

- 1 job per benchmark, 1+1 processes per job, 1+1 processes per core
- Since while one process sleeps the other works I can have 2 processes per core?

Problem:

- With some settings the application is memory bound (~15GB/process) $\rightarrow$ Hopper nodes?
- Running the whole benchmark single-threaded on a Intel i9-9900KF (3.60GHz) takes 70h $\rightarrow$ longer on Hopper?
- Will it hit the job limit of 7 days?

Alternative approaches

- 1 job per benchmark, 1 designer process, many (19) scoring threads → good for some tests, bad for others
- 1 job per test → Shorter, buy many jobs. Tricky data analysis