Architecture of PFS

- Page file is a sequence of pages
  - A *page* is a basic unit of processing
  - All pages have the same length
  - A *page identifier* (abbr. `pgid`) points to the beginning of the given page (offset of a page in page file)
  - The page FS does not know anything about the contents of pages
- Architecture of Page file server
  - Page server is an Erlang process
    - Page file server accepts requests from possibly many (ISAM) processes
    - Process defines callback routines that define the messages (protocol of process)
    - Process is connected by using data streams implemented in query_node.erl (maybe it should be renamed to streams.erl)
  - Binary file storage of pages
    - N-th page is accessed by reading|writting from|to the position n*page-size in db file
    - Pages are stored in binaries (unused fragments at the end of block)
    - The question is weather the stream data pages are of the same size as file pages
    - Read operation reads N pages from the given starting position in file
    - Write operation writes N pages from the given starting position
    - Append operation appends N pages to the end of data file
    - Data is needed for read and write operation is transfered via data streams
    - Protocol thus require the completition message for read and write (in opposite directions)
  - PFS is linked to a client vie I/O data streams
    - Input/output data of write/read operations is obtained via data streams
    - Data streams are composed of data messages that contain up to TRIPLES_IN_PAGE triples (or less)
    - Reading/writing data messages/triples from/to a stream
      - Processing unit is either a data message or a triple
      - Stream type is defined on initialization of a named queue

  - Requests are placed in a queue and served one by one
    - Pid of the client process is stored for each request
    - Request to read N pages is completed after all the pages are read and sent to client
    - Request to write N pages starts after complete data has been transfered
      - Data can be stored in a map that maps Pids to lists of collected data pages
    - Each request can process (read or write) a chunk of data
      - A chunk of data is defined by the number of pages
      - After a chunk is processed the state is stored in request and it is put back at the end of queue
      - This implements a kind of round-robin algorithm
      - All other request do not freeze if a large request is being processed
    - (to-do) Does it make sense to have sessions (with a given process `pid`)?
  - (expand) A cache is part of PFS
    - Page are read into buffer pool
    - LRU page replacement strategy is used

- Page file server interface
  - { data_read, pgid, N }
    - Reads a sequence of N pages starting at the page pgid
    - Read data pages are sent to the client process via data streams
  - { data_read_end }
    - Signals the completition of data_read operation
    - Number of pages sent to client is N
  - { data_write, pgid, N }
    - Writes a sequence of N pages to the db file
    - Data pages to write are received from a client process via data streams
  - { data_write_end }
    - Signals the completition of data_write or data_append operation
    - Number of pages received from client has to be equal N
  - { data_append, N } }
    - Writes a sequence of N pages to the end of data file
    - Data pages to write are received from a client process via data streams