wendelin.core

effortless out-of-core NumPy

2014-04-03 – Paris

Who am I?

- **Kirill Smelkov**
- **Senior developer at Nexedi**
- **Author of wendelin.core**
- **Contributor to linux, git and scientific libraries from time to time**
- **kirr@nexedi.com**

Agenda

- **Where do we come from**
- **Five problems to solves**
- **The solution**
- **Future Roadmap**

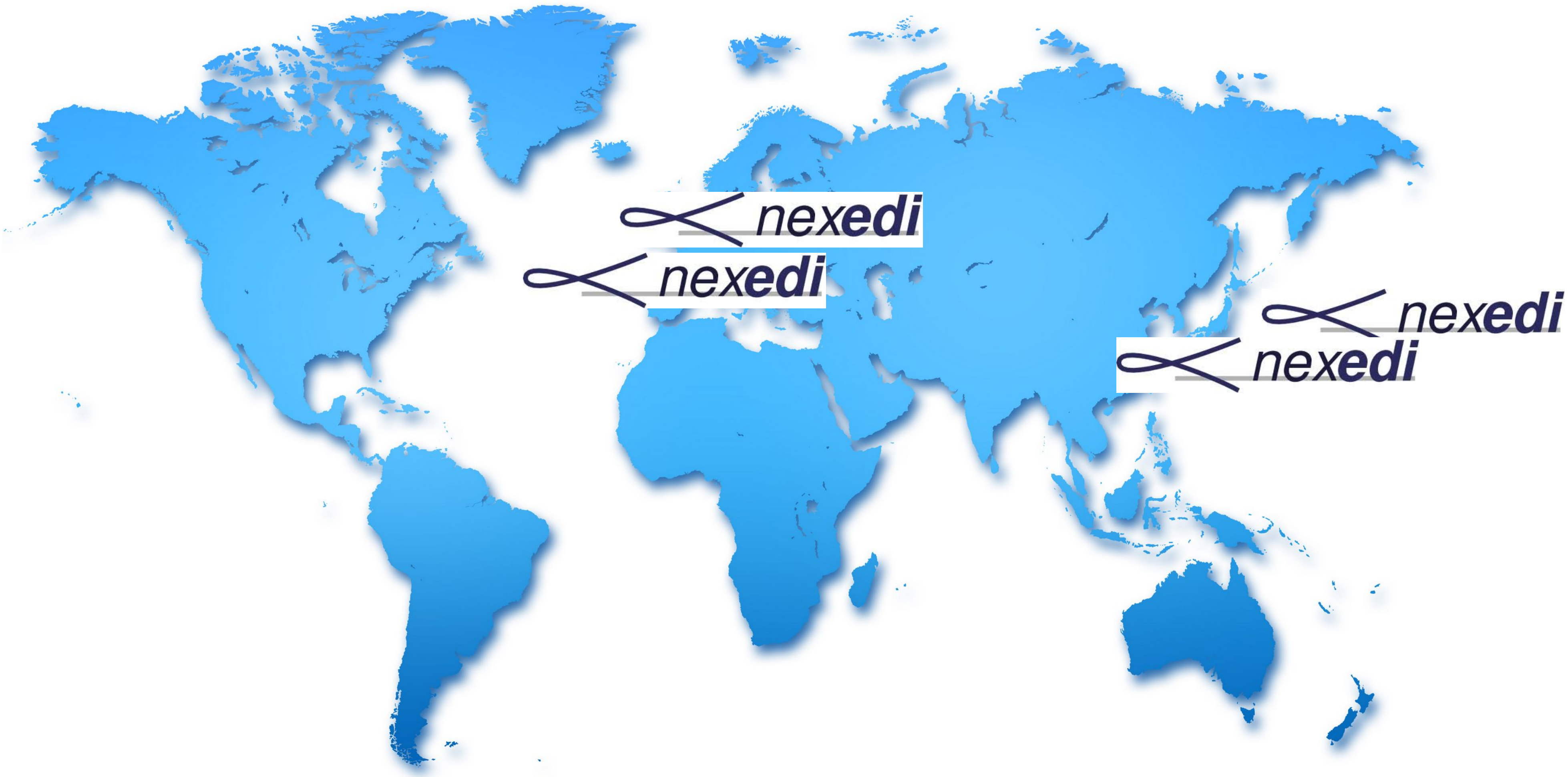
Where do we come from?

WENDELIN

nexedi

Nexedi

- **Possibly Largest OSS Publisher in Europe**
  - **ERP5: ERP, CRM, ECM, e-business framework**
  - **SlapOS: distributed mesh cloud operation system**
  - **NEO: distributed transactional NoSQL database**
  - **Wendelin: out-of-core big data based on NumPy**
  - **re6st: resilient IPv6 mesh overlay network**
  - **RenderJS: javascript component system**
  - **JIO: javascript virtual database and virtual filesystem**
  - **cloudooo: multimedia conversion server**
  - **Web Runner: web based Platform-as-a-Service (PaaS) and IDE**
  - **OfficeJS: web office suite based on RenderJS and JIO**



Aide et Action
L'Education change le monde

WEINPARIS.COM
BY ZAWEE

HANGZHOU DIANZI UNIVERSITY

Liberté • Égalité • Fraternité
RÉPUBLIQUE FRANÇAISE
MINISTÈRE DE L'ÉCONOMIE ET DES FINANCES

MITSUBISHI MOTORS

BCEAO

SOUTH AFRICA
CAPAGO
The Schengen link

AIRBUS
DEFENCE & SPACE

Sanef

Kyorin

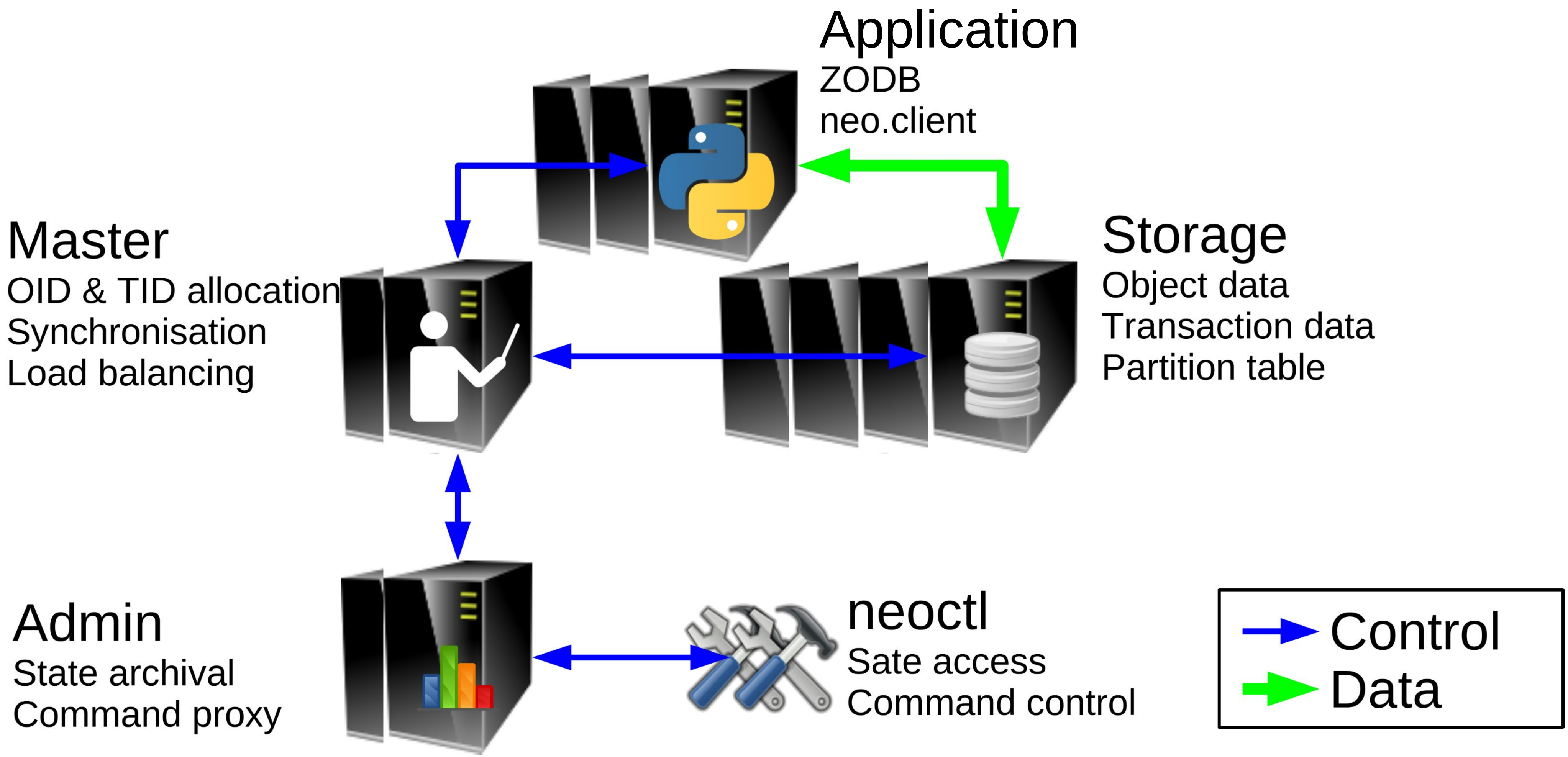
nexedi

Application Convergence

ERP5 + scikit learn

?

ERP5 Storage: NEO



Standard Hardware no router / no SAN

inspur 浪潮 lenovo 联想 CORETO

x 160

- 2 x 10 Gbps
- 2 x 6 core Xeon CPU
- 512 GB RAM
- 4 x 1 TB SSD
- 1 x M2090 GPU

+

x 32

- 10 Gbps
- Unmanaged

+

x 320

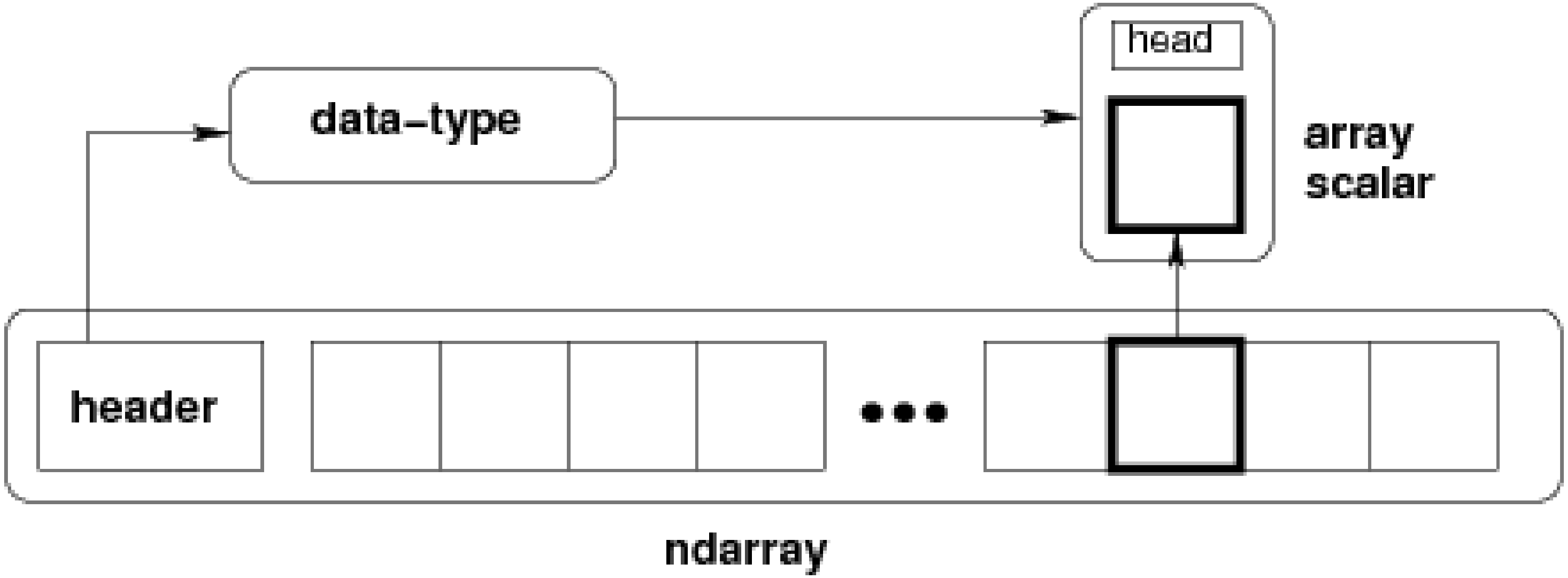
nexedi

Five Problems to Solve

WENDELIN

nexedi

It is All About NumPy



Problem 1: Persistent NumPy

- **How to store NumPy arrays in a database?**
  - **in NEO?**
  - **in NoSQL?**
  - **in SQL?**

Problem 2: Distributed NumPy

- **How to share NumPy arrays in a cluster?**
  - **One PC, many Python processes**
  - **Many PC, many Python processes**

Problem 3: Out-of-core NumPy

- **How to load big NumPy arrays in small RAM?**
  - **ERP5: “it should work even if it does not work”**
  - **Stopping business is not an option (because of not enough RAM)**

Problem 4: Transactional NumPy

- **How to make NumPy arrays transaction safe?**
  - **Exception handling**
  - **Concurrent writes**
  - **Distributed computing**

Problem 5: Compatibility

- **Compatibility with NumPy-based stack is a must**
- **Native BLAS support is a must**
- **Cython/FORTRAN/C/C++ support is a must**
- **Code rewrite is not an option**
  - **Blaze: not NumPy compatible below Python level**
  - **Dato: not NumPy compatible**

The Solution

WENDELIN

nexedi

Unsolutions

- **Update NumPy & libraries with calling notification hooks when memory is changed → not practical**
  - **There is a lot of code in numpy and lot of libraries around numpy**
  - **Catching them all would be a huge task**
- **Compare array data to original array content at commit time and store only found-to-be-changed parts → not good**
  - **At every commit whole array data has to be read/analyzed and array data can be very big**

Remember mmap? READ

- **Region of memory mapped by kernel to a file**
- **Memory pages start with NONE protection**
  → CPU can not read nor write
- **Whenever read request comes from CPU, kernel traps it (thanks to MMU), loads content for that page from file, and resumes original read**

Remember mmap? WRITE

- **Whenever write request comes from CPU, kernel traps it, marks the page as DIRTY, unprotects it and resumes original write**
  → kernel knows which pages were modified
- **Whenever application wants to make sure modified data is stored back to file (msync), kernel goes over list of dirty pages and writes their content back to file**

Partial Conclusion

- **If we manage to represent arrays as files, we'll get "track-changes-to-content" from kernel**

FUSE ?

- **FUSE & virtual filesystem representing "glued" arrays from ZODB BTree & objects**
- **Problem 1: does not work with huge pages**
  - **Performance issues**
  - **Not easy to fix**
- **Problem 2: no support for commit / abort**
  - **Transaction issues**

UVMM: Userspace Virtual Memory Manager

- **Trap write access to memory via installing SIGSEGV signal handler**

UVMM ON CPU WRITE

- **SIGSEGV handler gets notified,**
- **Marks corresponding array block as dirty**
- **Adjust memory protection to be read-write**
- **Resumes write instruction**
- → **we know which array parts were modified**

UVMM ON CPU READ

- **Set pages initial protection to PROT_NONE → no-read and no-write**
- **First load in SIGSEGV handler**
- **When RAM is tight, we can "forget" already loaded (but not-yet modified) memory parts and free RAM for loading new data**

UVMM LIMITS ?

- **Array size is only limited by virtual memory address space size**
  → 127TB on Linux/amd64 (today)
- **Future Linux kernel may support more**

Is it safe to do work in SIGSEGV handler?

- **Short answer: YES**
- **Long answer: www.wendelin.io**

Tutorial: init a BigFile backend

```
from wendelin.bigfile import BigFile

# bigfile with data storage in 'some backend'
class BigFile_SomeBackend(BigFile):
    .blksize = …                    # file is stored in block of size

    def loadblk(self, blk, buf)  # load file block #blk to memory buffer `buf`

    def storeblk(self, blk, buf) # store data from memory buffer `buf` to file
                                 # block blk


f  = BigFile_SomeBackend(...)
```

BigFile Handle: BigFile as Memory

```
# BigFile handle is a representation of file snapshot that could be locally
# modified in-memory. The changes could be later either discarded or stored
# back to file. One file can have many opened handles each with its own
# modifications.
fh = f.fileh_open()

# memory mapping of fh
vma = fh.mmap(pgoffset=0, pglen=N)

# vma exposes memoryview/buffer interfaces
mem = memoryview(vma)

# now we can do with `mem` whatever we like
...

fh.dirty_discard()      # to forget all changes done to `mem` memory
fh.dirty_writeout(...)  # to store changes back to file
```

ZBigFile: ZODB & Transactions

```
from webdelin.bigfile.file_zodb import ZBigFile
import transaction

f  = ZBigFile()                     # create anew
f  = root['...'].some.object        # load saved state from database

# the same as with plain BigFile (previous example)
fh = fileh_open()
vma = fh.mmap(0, N)
mem = memoryview(vma)

# we can also modify other objects living in ZODB

transaction.abort()     # to abort all changes to mem and other objects
transaction.commit()    # to commit all changes to mem and other objects
```

BigArray: “ndarray” on top of BigFile

```
# f - some BigFile
# n - some (large) number
fh = f.fileh_open()     # handle to bigfile (see slide ...)
A  = BigArray(shape=(n,10), dtype=uint32, fh)

a  = A[0:3*(1<<30), :]  # real ndarray viewing first 3 giga-rows (= ~120GB) of
                        # data from f
                        # NOTE 120GB can be significantly > of RAM available

a.mean()                # computes mean of items in above range
                        # this call is just an ndarray.mean() call and code
                        # which works is the code in NumPy.
                        # NOTE data will be loaded and freed by virtual memory
                        # manager transparently to client code which computes
                        # the mean
```

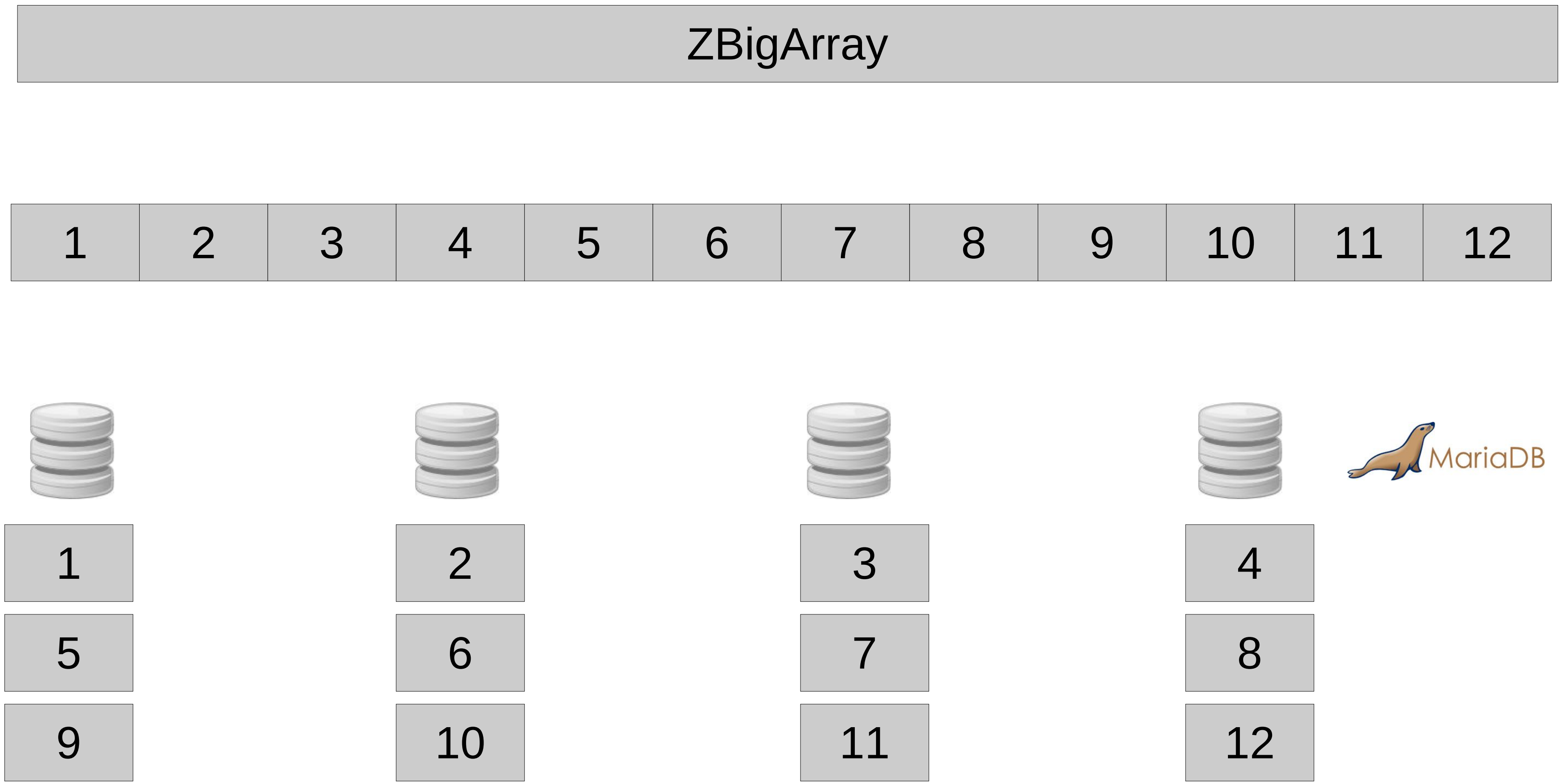
BigArray: Transactions

```
a[2] = ...
...
fh.dirty_discard()       # to discard, or
fh.dirty_writeout()      # to write
```

ZBigArray: ZODB & Transactions

```
from wendelin.bigarra.array_zodb import ZbigArray
import transaction

# root is connection to oped database
root['sensor_data'] = A = ZBigArray(shape=..., dtype=...)

# populate A with data
A[2] = 1

# compute mean
A.mean()

# abort / commit changes
transaction.abort()
transaction.commit()
```

NEO and ZBigArray



Future Improvements

- **Temporary arrays created by NumPy libraries**
- **Performance**
- **Multithreading**

Future Roadmap

WENDELIN

nexedi

Roadmap

www.wendelin.io

- **Make wendelin.core fast**
  - [ ] userfaultfd, filesystem-based approach
  - [ ] remove use of pickles
  - [ ] remove large temporary arrays in NumPy, etc.
- **Yet, you can start using wendelin.core now!**
  - [x] Persistent
  - [x] Distributed
  - [x] Out-of-core
  - [x] Transactional
  - [x] Virtually no change to your code needed
  - [x] Open Source

nexedi

wendelin.core

effortless out-of-core NumPy

2014-04-03 – Paris

WENDELIN

www.wendelin.io

nexedi

wendelin.core
effortless out-of-core NumPy

2014-04-03 – Paris

Who am I?

- **Kirill Smelkov**
- **Senior developer at Nexedi**
- **Author of wendelin.core**
- **Contributor to linux, git and scientific libraries from time to time**
- **kirr@nexedi.com**

Agenda

- **Where do we come from**
- **Five problems to solves**
- **The solution**
- **Future Roadmap**

Where do we come from?

WENDELIN

nexedi

Nexedi

- **Possibly Largest OSS Publisher in Europe**
  - **ERP5: ERP, CRM, ECM, e-business framework**
  - **SlapOS: distributed mesh cloud operation system**
  - **NEO: distributed transactional NoSQL database**
  - **Wendelin: out-of-core big data based on NumPy**
  - **re6st: resilient IPv6 mesh overlay network**
  - **RenderJS: javascript component system**
  - **JIO: javascript virtual database and virtual filesystem**
  - **cloudooo: multimedia conversion server**
  - **Web Runner: web based Platform-as-a-Service (PaaS) and IDE**
  - **OfficeJS: web office suite based on RenderJS and JIO**


nexedi
nexedi
nexedi
nexedi


Aide et Action
L'Education change le monde
WEINPARIS.COM
BY ZAWEE
HANGZHOU DIANZI UNIVERSITY
MITSUBISHI
MOTORS
BCEAO
SOUTH AFRICA
CAPAGO
The Schengen link
AIRBUS
DEFENCE & SPACE
Sanef
Kyorin
nexedi

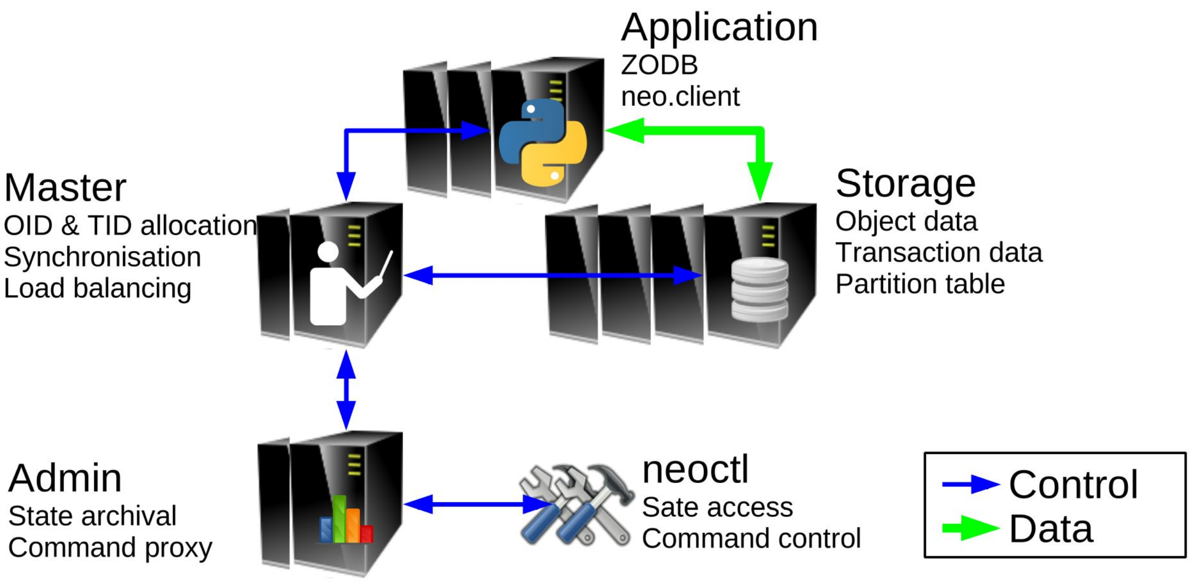
Application Convergence

ERP5 + scikit learn

?

nexedi

ERP5 Storage: NEO



Standard Hardware no router / no SAN

x 160

- 2 x 10 Gbps
- 2 x 6 core Xeon CPU
- 512 GB RAM
- 4 x 1 TB SSD
- 1 x M2090 GPU

+

x 32

- 10 Gbps
- Unmanaged

+

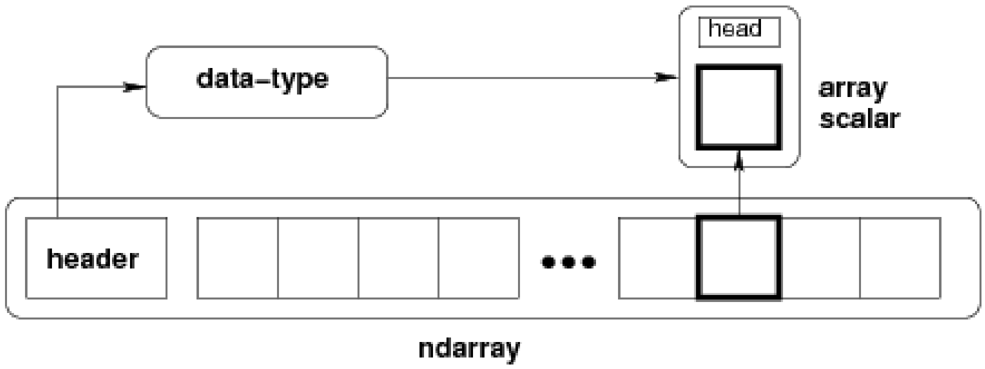
x 320

Five Problems to Solve

WENDELIN

nexedi

It is All About NumPy



Problem 1: Persistent NumPy

- **How to store NumPy arrays in a database?**
  - **in NEO?**
  - **in NoSQL?**
  - **in SQL?**

Problem 2: Distributed NumPy

- **How to share NumPy arrays in a cluster?**
  - **One PC, many Python processes**
  - **Many PC, many Python processes**

Problem 3: Out-of-core NumPy

- **How to load big NumPy arrays in small RAM?**
  - **ERP5: “it should work even if it does not work”**
  - **Stopping business is not an option (because of not enough RAM)**

Problem 4: Transactional NumPy

- **How to make NumPy arrays transaction safe?**
  - **Exception handling**
  - **Concurrent writes**
  - **Distributed computing**

Problem 5: Compatibility

- **Compatibility with NumPy-based stack is a must**
- **Native BLAS support is a must**
- **Cython/FORTRAN/C/C++ support is a must**
- **Code rewrite is not an option**
  - **Blaze: not NumPy compatible below Python level**
  - **Dato: not NumPy compatible**

The Solution

WENDELIN

nexedi

Unsolutions

- **Update NumPy & libraries with calling notification hooks when memory is changed → not practical**
  - **There is a lot of code in numpy and lot of libraries around numpy**
  - **Catching them all would be a huge task**
- **Compare array data to original array content at commit time and store only found-to-be-changed parts → not good**
  - **At every commit whole array data has to be read/analyzed and array data can be very big**

Remember mmap? READ

- **Region of memory mapped by kernel to a file**
- **Memory pages start with NONE protection**
  → CPU can not read nor write
- **Whenever read request comes from CPU, kernel traps it (thanks to MMU), loads content for that page from file, and resumes original read**

Remember mmap? WRITE

- **Whenever write request comes from CPU, kernel traps it, marks the page as DIRTY, unprotects it and resumes original write**
  → kernel knows which pages were modified
- **Whenever application wants to make sure modified data is stored back to file (msync), kernel goes over list of dirty pages and writes their content back to file**

Partial Conclusion

- **If we manage to represent arrays as files, we'll get “track-changes-to-content” from kernel**

FUSE ?

- FUSE & virtual filesystem representing "glued" arrays from ZODB BTree & objects
- Problem 1: does not work with huge pages
  - Performance issues
  - Not easy to fix
- Problem 2: no support for commit / abort
  - Transaction issues

UVMM: Userspace Virtual Memory Manager

- **Trap write access to memory via installing SIGSEGV signal handler**

UVMM ON CPU WRITE

- SIGSEGV handler gets notified,
- Marks corresponding array block as dirty
- Adjust memory protection to be read-write
- Resumes write instruction
- → we know which array parts were modified

UVMM ON CPU READ

- **Set pages initial protection to PROT_NONE**
  → no-read and no-write
- **First load in SIGSEGV handler**
- **When RAM is tight, we can "forget" already loaded (but not-yet modified) memory parts and free RAM for loading new data**

UVMM LIMITS ?

- Array size is only limited by virtual memory address space size
  → 127TB on Linux/amd64 (today)
- Future Linux kernel may support more

Is it safe to do work in SIGSEGV handler?

- **Short answer: YES**
- **Long answer: www.wendelin.io**

Tutorial: init a BigFile backend

```
from wendelin.bigfile import BigFile

# bigfile with data storage in 'some backend'
class BigFile_SomeBackend(BigFile):
    .blksize = …                  # file is stored in block of size

    def loadblk(self, blk, buf)  # load file block #blk to memory buffer `buf`

    def storeblk(self, blk, buf) # store data from memory buffer `buf` to file
                                 # block blk


f  = BigFile_SomeBackend(...)
```

BigFile Handle: BigFile as Memory

```
# BigFile handle is a representation of file snapshot that could be locally
# modified in-memory. The changes could be later either discarded or stored
# back to file. One file can have many opened handles each with its own
# modifications.
fh = f.fileh_open()

# memory mapping of fh
vma = fh.mmap(pgoffset=0, pglen=N)

# vma exposes memoryview/buffer interfaces
mem = memoryview(vma)

# now we can do with `mem` whatever we like
...

fh.dirty_discard()      # to forget all changes done to `mem` memory
fh.dirty_writeout(...)  # to store changes back to file
```

ZBigFile: ZODB & Transactions

```
from webdelin.bigfile.file_zodb import ZBigFile
import transaction

f  = ZBigFile()                   # create anew
f  = root['...'].some.object      # load saved state from database

# the same as with plain BigFile (previous example)
fh = fileh_open()
vma = fh.mmap(0, N)
mem = memoryview(vma)

# we can also modify other objects living in ZODB

transaction.abort()     # to abort all changes to mem and other objects
transaction.commit()    # to commit all changes to mem and other objects
```

BigArray: “ndarray” on top of BigFile

```
# f - some BigFile
# n - some (large) number
fh = f.fileh_open()     # handle to bigfile (see slide ...)
A  = BigArray(shape=(n,10), dtype=uint32, fh)

a  = A[0:3*(1<<30), :]  # real ndarray viewing first 3 giga-rows (= ~120GB) of
                        # data from f
                        # NOTE 120GB can be significantly > of RAM available

a.mean()                # computes mean of items in above range
                        # this call is just an ndarray.mean() call and code
                        # which works is the code in NumPy.
                        # NOTE data will be loaded and freed by virtual memory
                        # manager transparently to client code which computes
                        # the mean
```

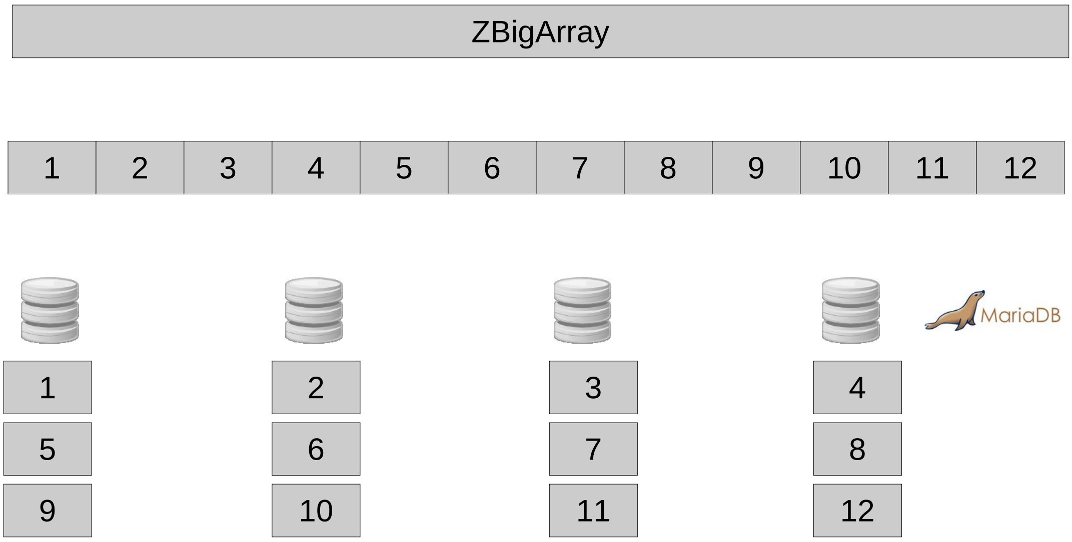
BigArray: Transactions

```
a[2] = ...
...
fh.dirty_discard()      # to discard, or
fh.dirty_writeout()     # to write
```

ZBigArray: ZODB & Transactions

```
from wendelin.bigarra.array_zodb import ZbigArray
import transaction

# root is connection to oped database
root['sensor_data'] = A = ZBigArray(shape=..., dtype=...)

# populate A with data
A[2] = 1

# compute mean
A.mean()

# abort / commit changes
transaction.abort()
transaction.commit()
```

NEO and ZBigArray



Future Improvements

- **Temporary arrays created by NumPy libraries**
- **Performance**
- **Multithreading**

Future Roadmap

WENDELIN

nexedi

Roadmap

www.wendelin.io

- **Make wendelin.core fast**
  - ☐ userfaultfd, filesystem-based approach
  - ☐ remove use of pickles
  - ☐ remove large temporary arrays in NumPy, etc.
- **Yet, you can start using wendelin.core now!**
  - ☑ Persistent
  - ☑ Distributed
  - ☑ Out-of-core
  - ☑ Transactional
  - ☑ Virtually no change to your code needed
  - ☑ Open Source

wendelin.core

effortless out-of-core NumPy

2014-04-03 – Paris

www.wendelin.io