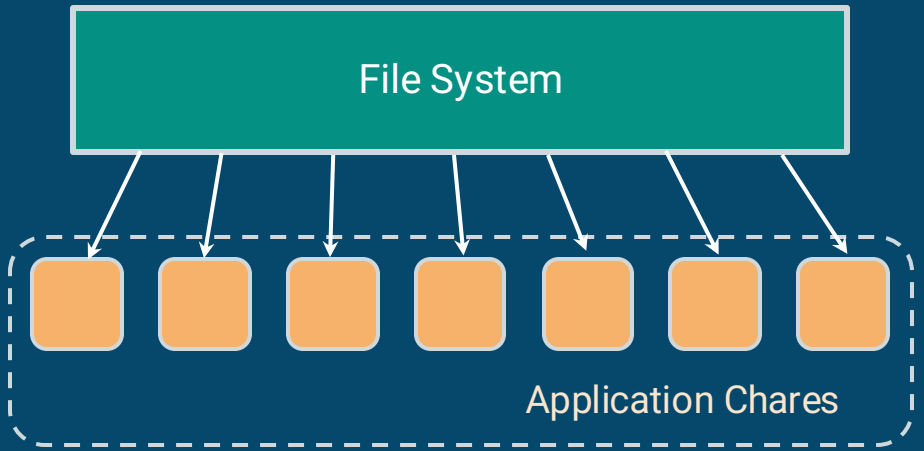
CkIO: Parallel File Input for Over-Decomposed Task-Based Systems

Mathew Jacob & Maya Taylor
University of Illinois Urbana Champaign

Overview

Charm++ programming model:

- task-based (chares)
- supports over-decomposition
- provides dynamic load balancing
- asynchronous



Challenges to parallel file input:

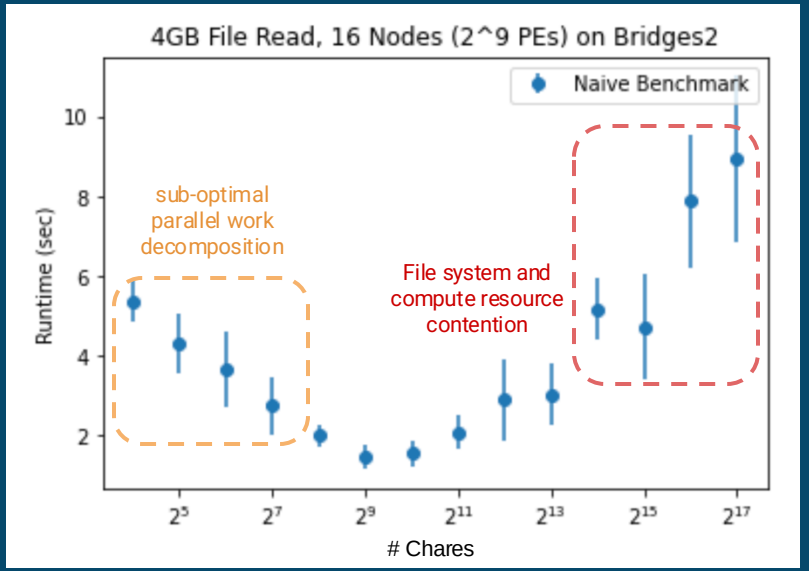
- numerous chares contending for compute and file system resources
- want to support concurrent computational work

Naive Parallel Input

How does parallel input perform under this model?

Naive input in Charm++:

- Fix file size and compute resources
- Chares read disjoint sections of a single file
- Vary the number of chares

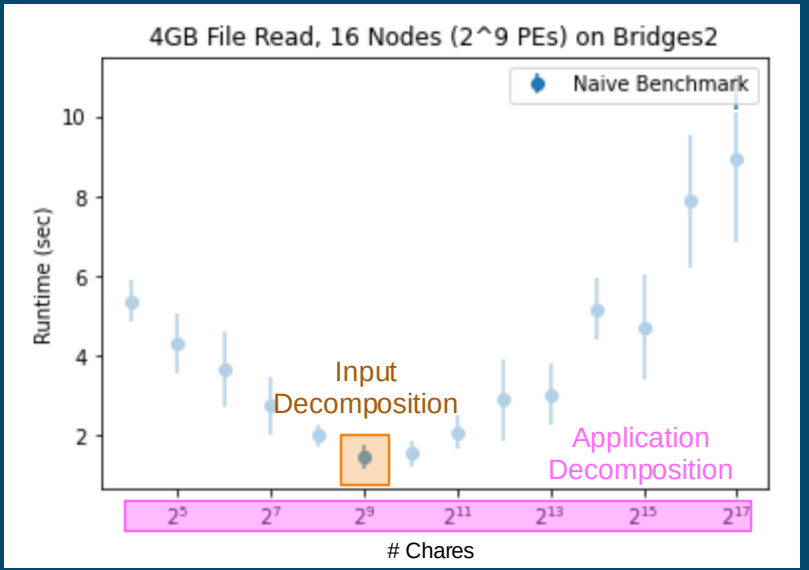


provide user freedom over the number of application tasks

CkIO Input Library

separation of concerns:

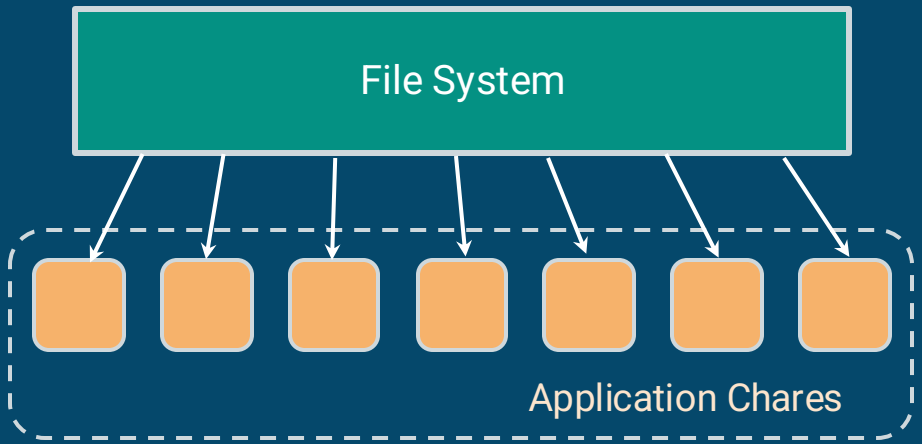
- input decomposition vs
- application decomposition



CkIO Input Library

separation of concerns:

- input decomposition vs
- application decomposition

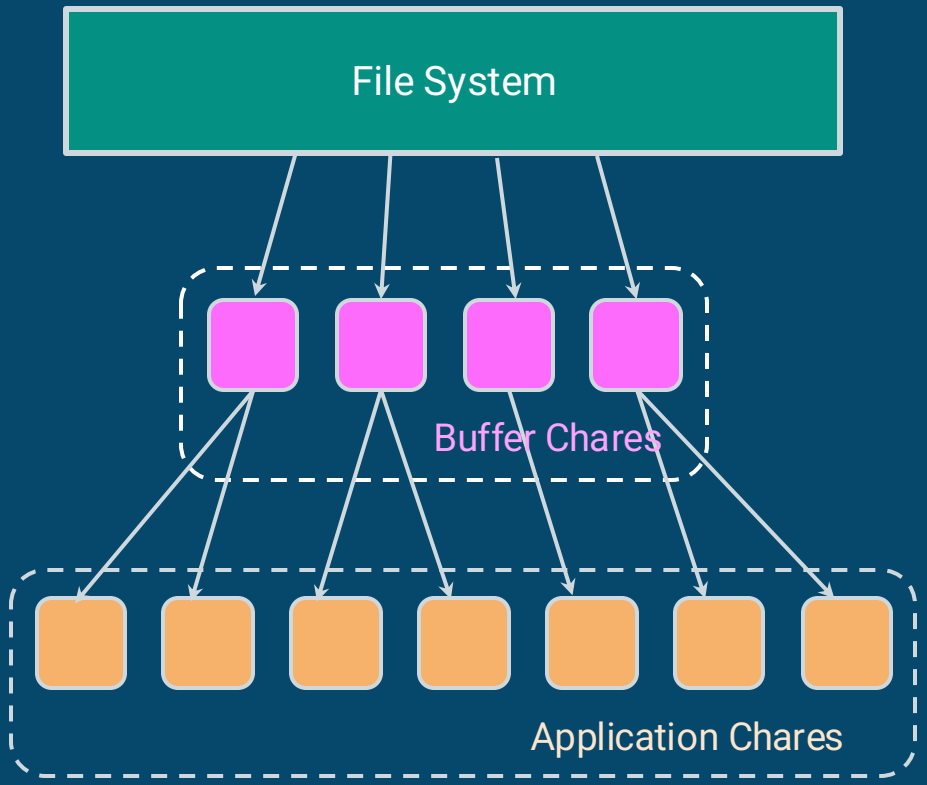


CkIO Input Library

separation of concerns:

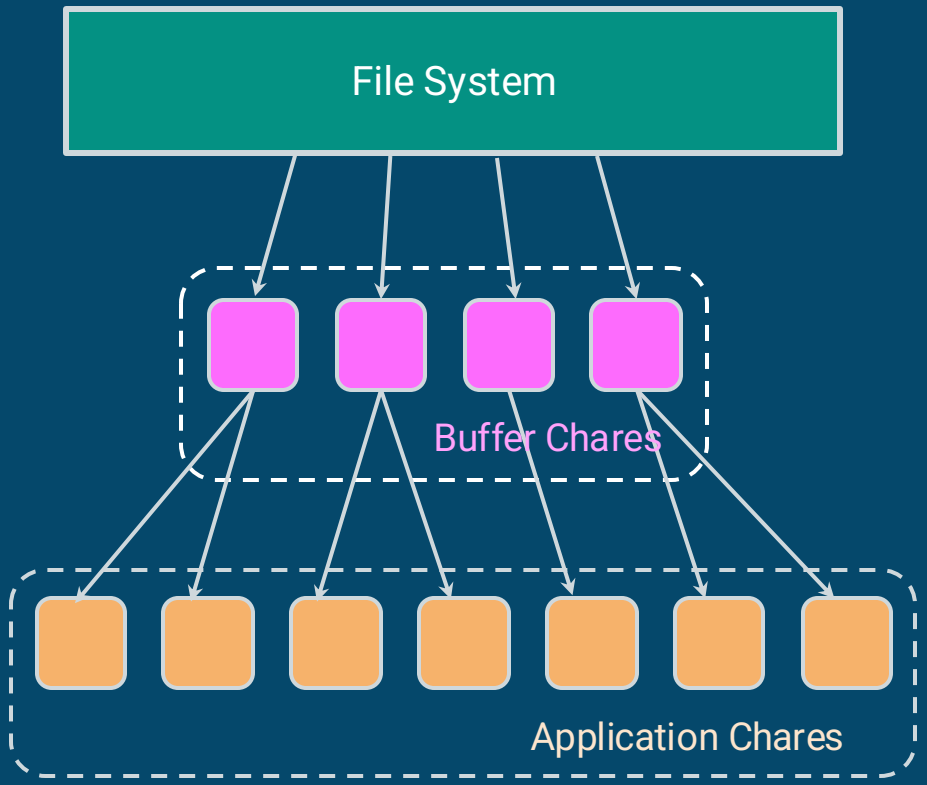
- input decomposition vs
- application decomposition

- abstraction via intermediary layer of buffer chares



CkIO Input Library

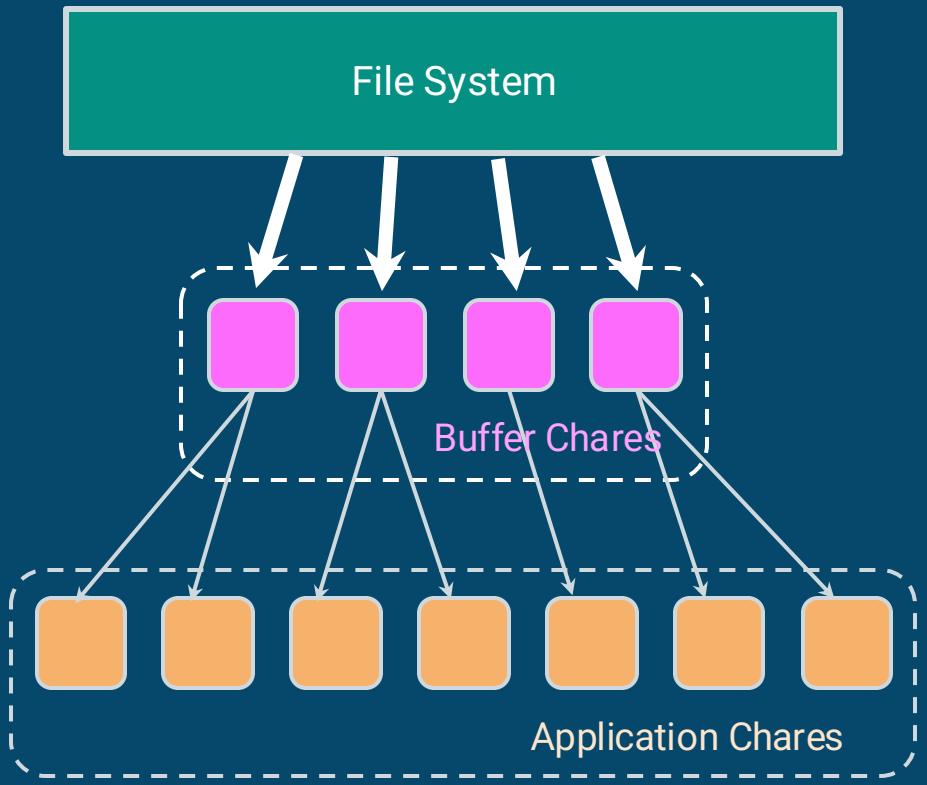
- Network is fast
- I/O is slow



CkIO Input Library

- Network is fast
- I/O is slow

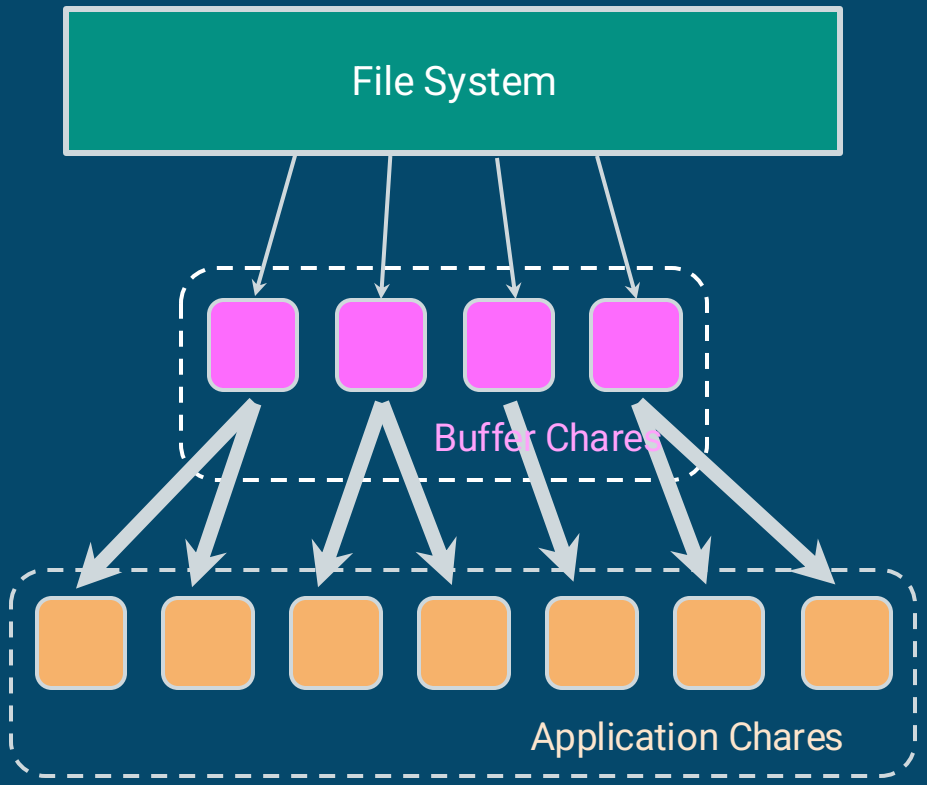
- Optimize input performance



CkIO Input Library

- Network is fast
- I/O is slow

- Optimize input performance
- Overhead of network distribution

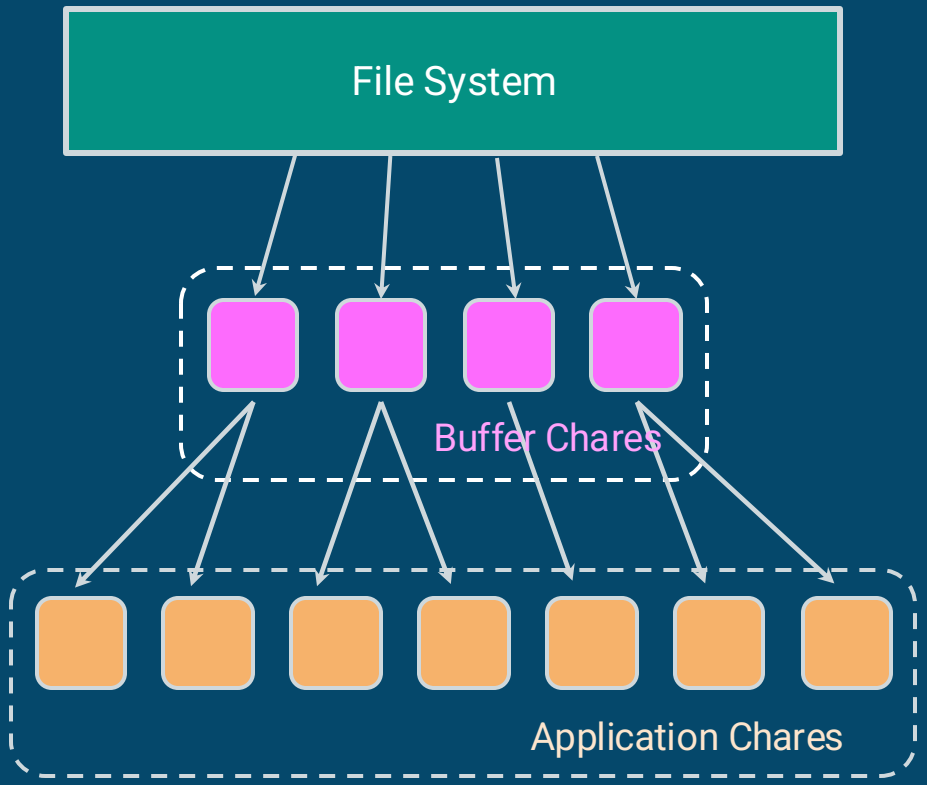


CkIO Input Library

- Network is fast
- I/O is slow

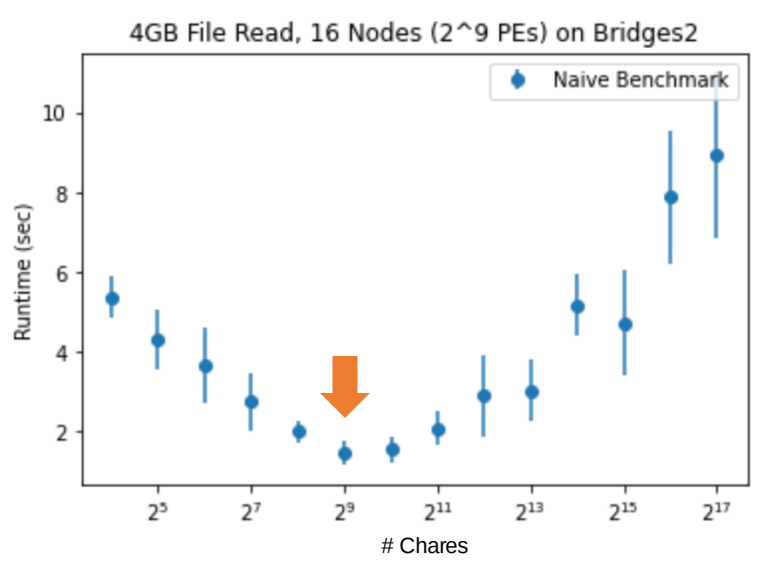
- Optimize input performance
- Overhead of network distribution

- Improve overall performance



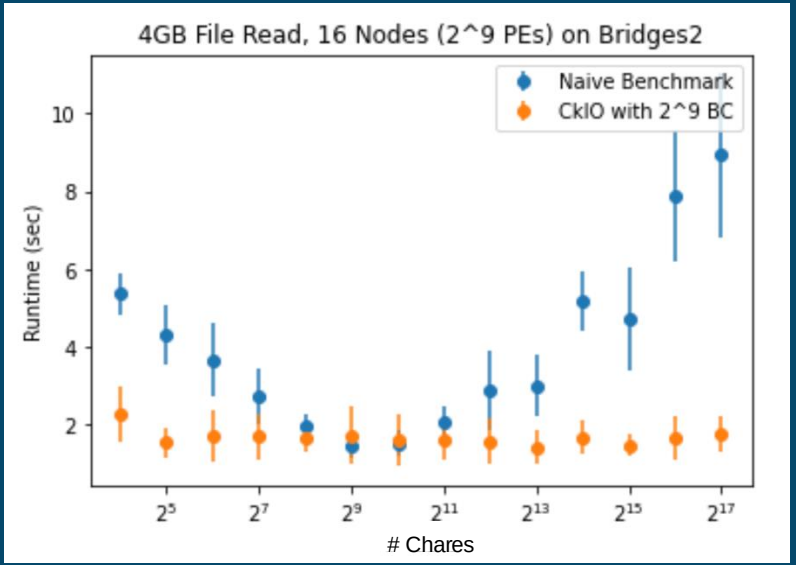
CkIO Input vs Naive Parallel Input

- choose number of buffer chares to match ideal input decomposition



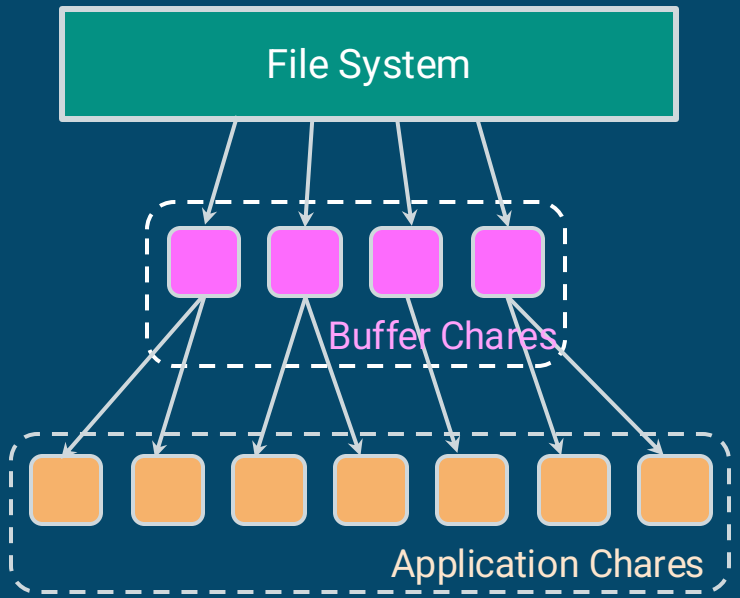
CkIO Input vs Naive Parallel Input

- choose number of buffer chares to match ideal input decomposition
- consistent input performance regardless of application decomposition



Supporting Concurrent Background Work

- when waiting on a read, buffer chares release control back to the Charm++ runtime system
- allows concurrent work to proceed during read calls to the file system

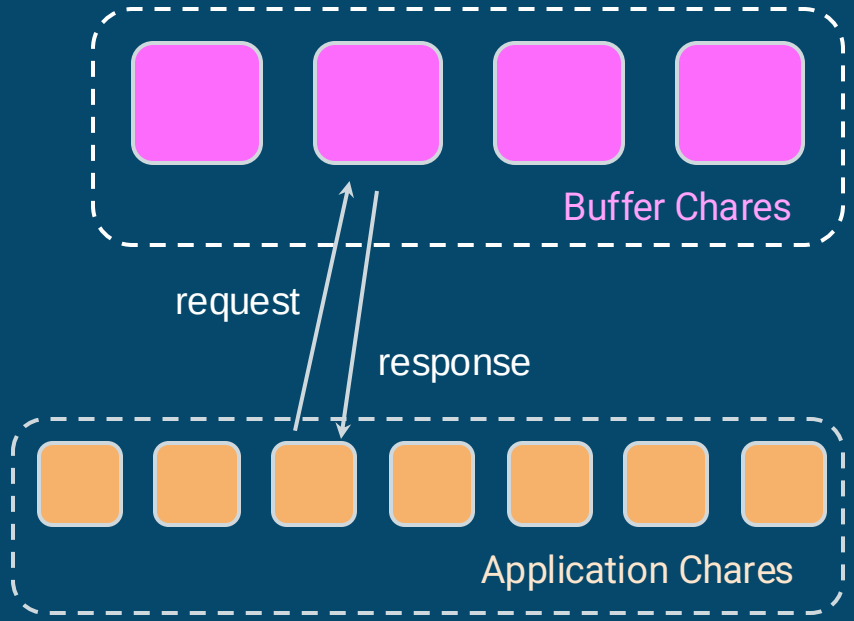


Supporting Concurrent Background Work

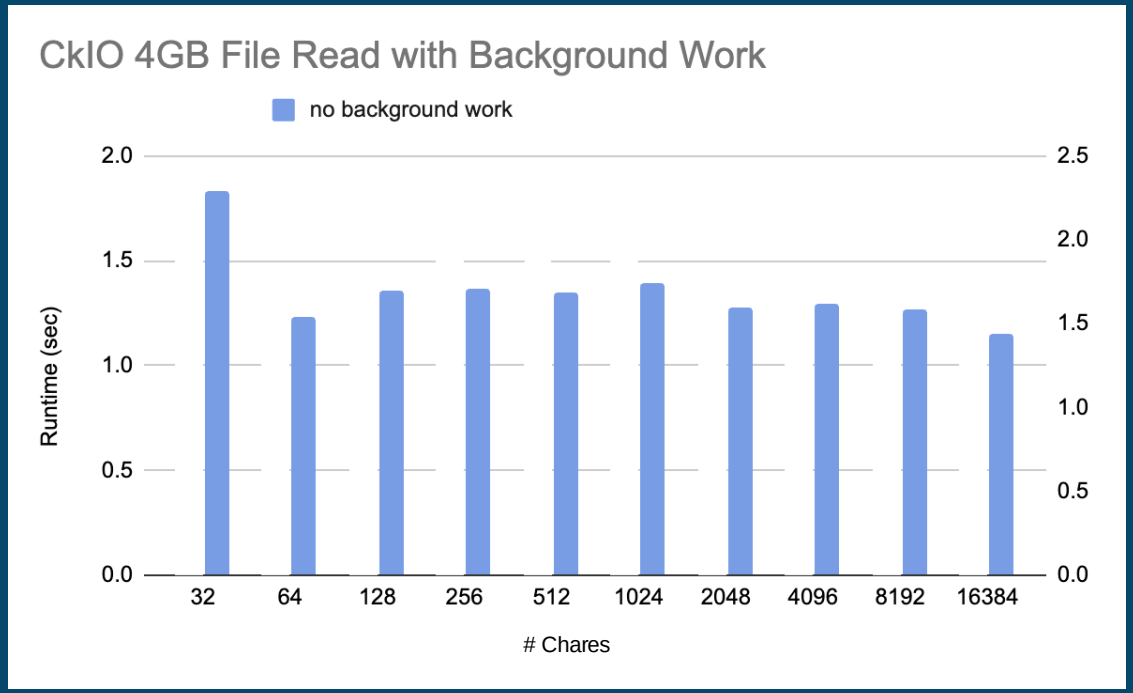
Create a pthread to read designated section into buffer.

Create a user-level Charm++ thread to intermittently check the status of the read.

Requests are buffered until read is complete and the data can be sent.



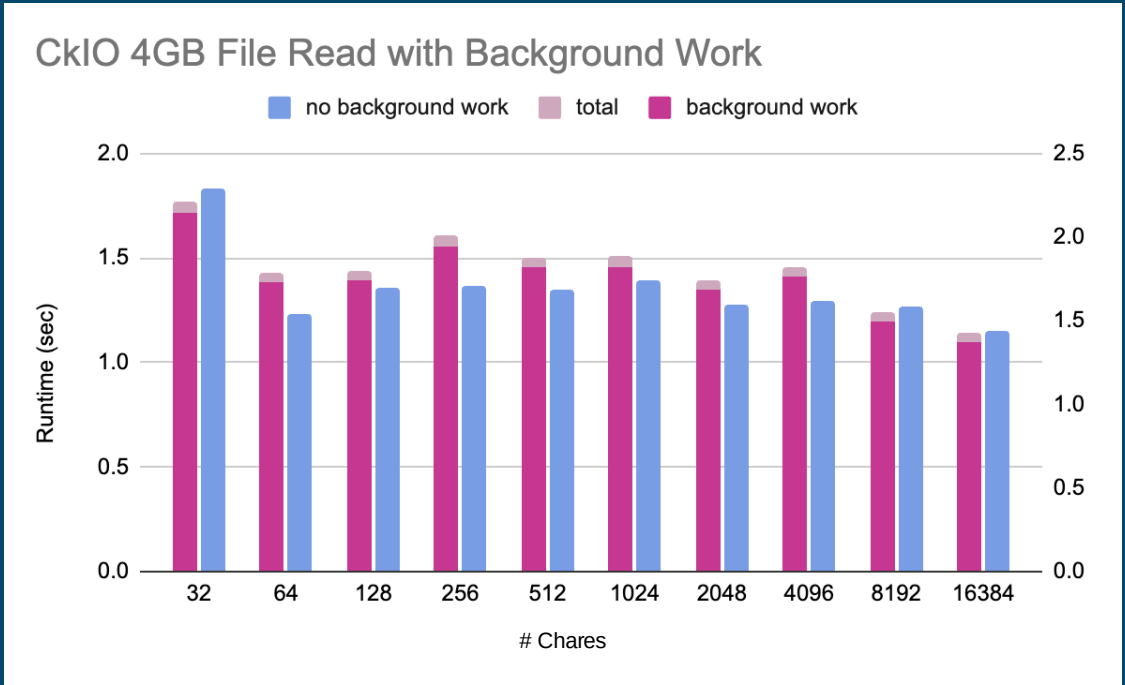
Background Work Performance



Baseline:

- CkIO with 2^9 buffer chares
- No concurrent work

Background Work Performance



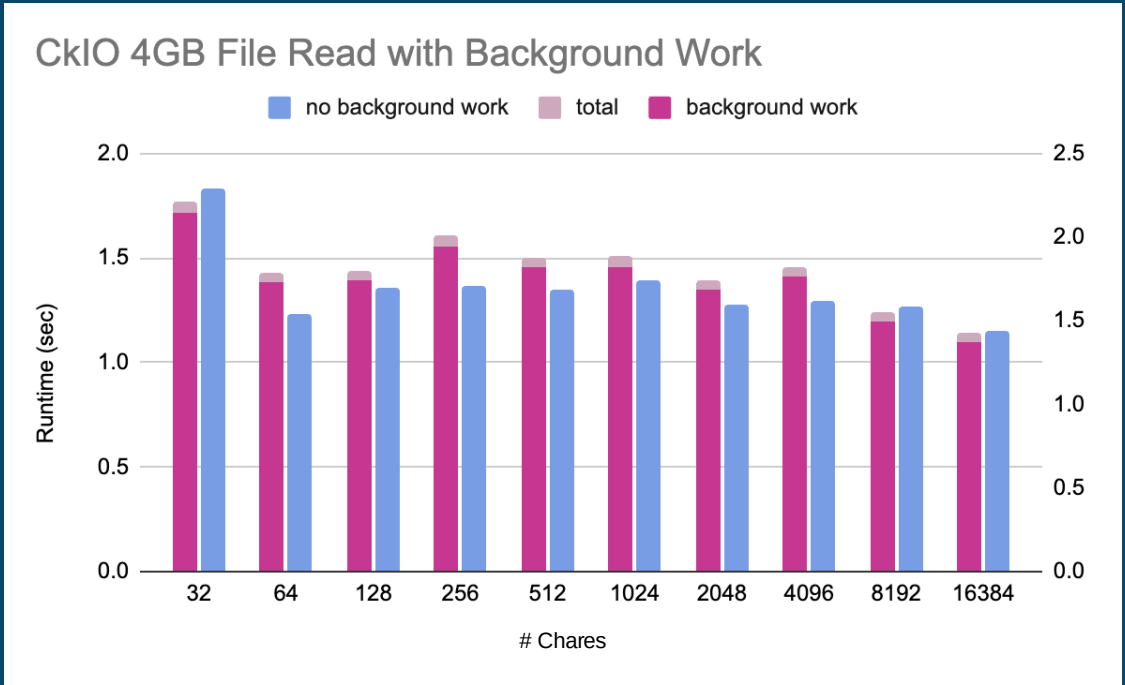
Baseline:

- CkIO with 2^9 buffer chares
- No concurrent work

Background work:

- Concurrent dummy computation
- Releases control intermittently

Background Work Performance



Takeaways:

- Almost perfect overlap
- Roughly maintain overall runtime

CkIO Input API

```
void Ck::IO::open(string name, CkCallback opened, Options opts);
void Ck::IO::startReadSession(File file, size_t bytes, size_t
                              offset, CkCallback ready);
void Ck::IO::read(Session session, size_t bytes, size_t offset,
                  char* data, CkCallback after_read);
void Ck::IO::closeReadSession(Session read_session, CkCallback
                              after_end);
void Ck::IO::close(File file, CkCallback closed);
```

CkIO Input API

```
void Ck::IO::open(string name, CkCallback opened, Options opts);
void Ck::IO::startReadSession(File file, size_t bytes, size_t
                              offset, CkCallback ready);
void Ck::IO::read(Session session, size_t bytes, size_t offset,
                  char* data, CkCallback after_read);
void Ck::IO::closeReadSession(Session read_session, CkCallback
                              after_end);
void Ck::IO::close(File file, CkCallback closed);
```

CkIO Input API

```
void Ck::IO::open(string name, CkCallback opened, Options opts);
void Ck::IO::startReadSession(File file, size_t bytes, size_t
                              offset, CkCallback ready);
void Ck::IO::read(Session session, size_t bytes, size_t offset,
                  char* data, CkCallback after_read);
void Ck::IO::closeReadSession(Session read_session, CkCallback
                              after_end);
void Ck::IO::close(File file, CkCallback closed);
```

CkIO Input API

```
void Ck::IO::open(string name, CkCallback opened, Options opts);
void Ck::IO::startReadSession(File file, size_t bytes, size_t
                              offset, CkCallback ready);
void Ck::IO::read(Session session, size_t bytes, size_t offset,
                  char* data, CkCallback after_read);
void Ck::IO::closeReadSession(Session read_session, CkCallback
                              after_end);
void Ck::IO::close(File file, CkCallback closed);
```

CkIO Input API

```
void Ck::IO::open(string name, CkCallback opened, Options opts);
void Ck::IO::startReadSession(File file, size_t bytes, size_t
                              offset, CkCallback ready);
void Ck::IO::read(Session session, size_t bytes, size_t offset,
                  char* data, CkCallback after_read);
void Ck::IO::closeReadSession(Session read_session, CkCallback
                              after_end);
void Ck::IO::close(File file, CkCallback closed);
```

CkIO Input API

```
void Ck::IO::open(string name, CkCallback opened, Options opts);
void Ck::IO::startReadSession(File file, size_t bytes, size_t
                              offset, CkCallback ready);
void Ck::IO::read(Session session, size_t bytes, size_t offset,
                  char* data, CkCallback after_read);
void Ck::IO::closeReadSession(Session read_session, CkCallback
                              after_end);
void Ck::IO::close(File file, CkCallback closed);
```

CkIO FileReader

- Abstraction built on top of raw Ck::IO API
- C++ object (not a chare)
- Designed to match the std::ifstream API

CkIO FileReader API

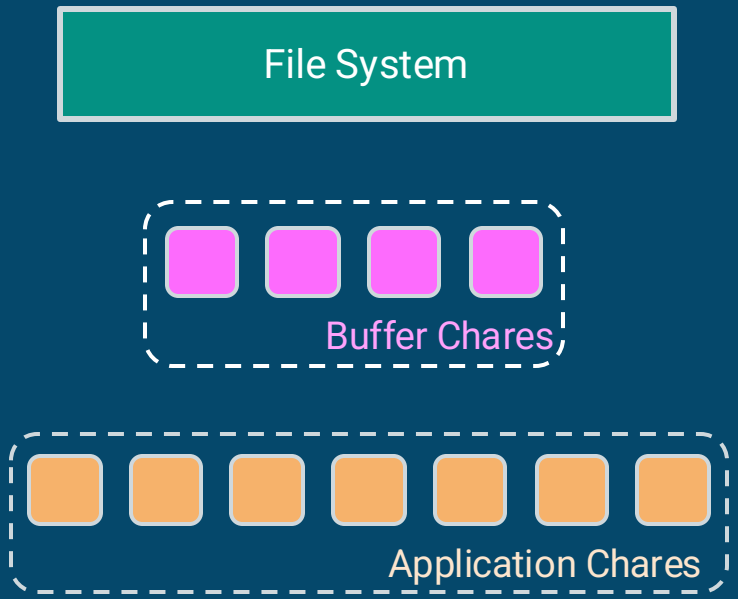
```
Ck::IO::FileReader::read(char* buffer, size_t num_bytes_to_read)
Ck::IO::FileReader::seekg(size_t pos)
Ck::IO::FileReader::seekg(size_t pos, std::ios_base::seekdir dir)
Ck::IO::FileReader::tellg()
Ck::IO::FileReader::gcount()
Ck::IO::FileReader::eof()
```

Must be used within a threaded entry method

CkIO FileReader

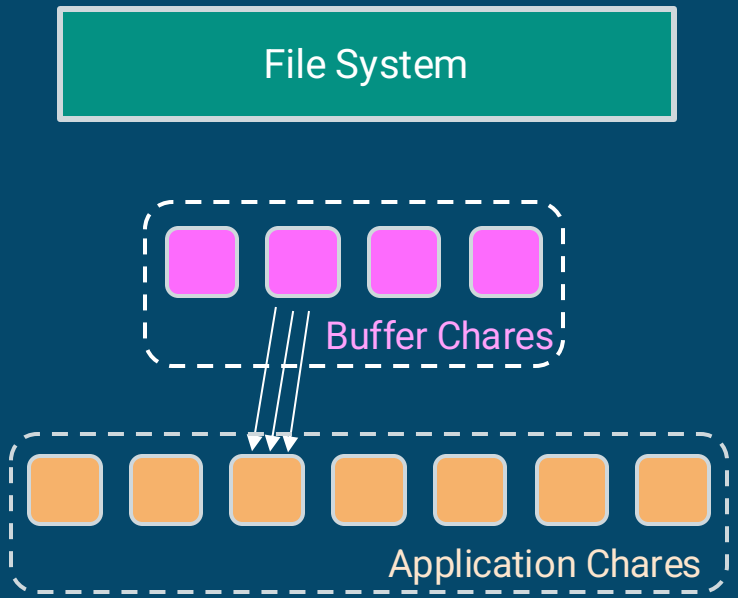
- Useful for when doing many sequential, small reads
- Optimizations: buffering to improve performance

CkIO FileReader



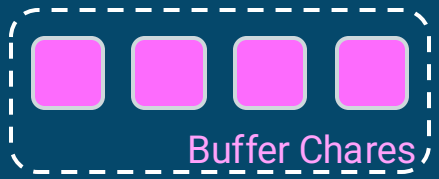
- Useful for when doing many sequential, small reads
- Optimizations: buffering to improve performance

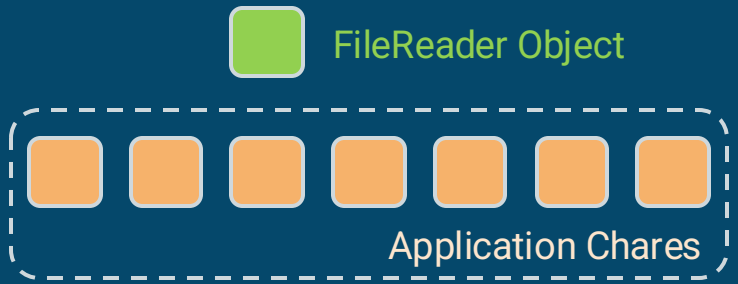
CkIO FileReader



- Useful for when doing many sequential, small reads
- Optimizations: buffering to improve performance

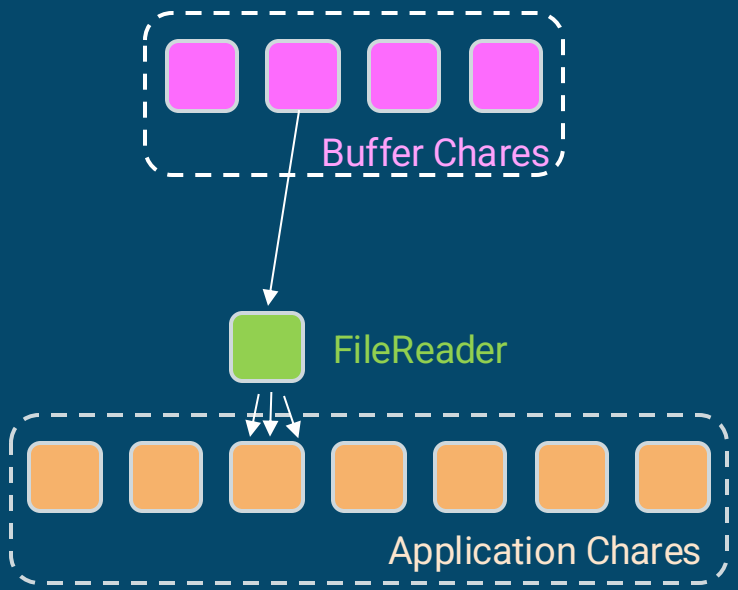
CkIO FileReader





- Useful for when doing many sequential, small reads
- Optimizations: buffering to improve performance

CkIO FileReader



- Useful for when doing many sequential, small reads
- Optimizations: buffering to improve performance

Applying CkIO: ChaNGa

Very much a work in progress!

Applying CkIO: ChaNGa

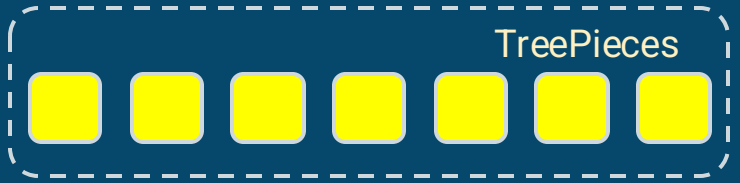
Existing input structure in ChaNGa:

- All input happens before computation

Applying CkIO: ChaNGa

Existing input structure in ChaNGa:

- All input happens before computation
- TreePieces collectively load input file



Applying CkIO: ChaNGa

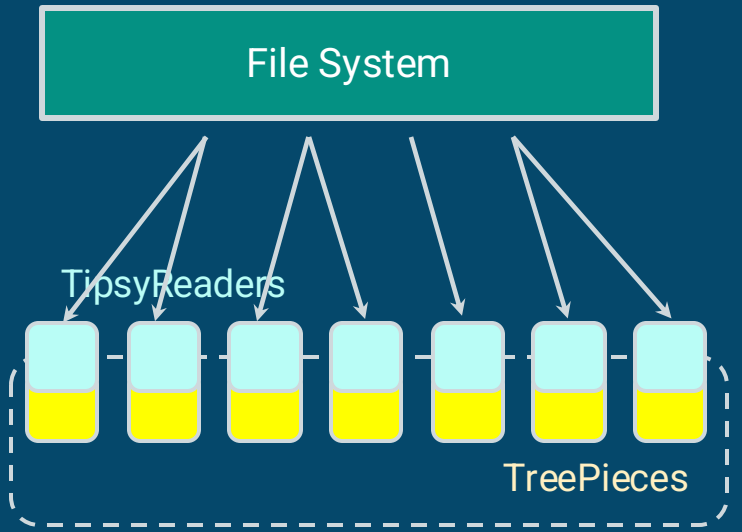
Existing input structure in ChaNGa:

- All input happens before computation
- TreePieces collectively load input file
- Each TreePiece creates a TipsyReader (plain c++ objects)

Applying CkIO: ChaNGa

Existing input structure in ChaNGa:

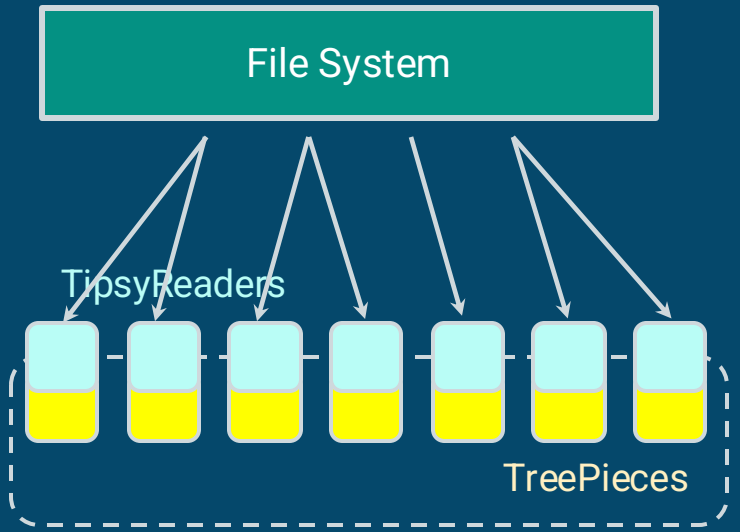
- All input happens before computation
- TreePieces collectively load input file
- Each TreePiece creates a TipsyReader (plain c++ objects)
- TipsyReader reads via std::ifstream



Applying CkIO: ChaNGa

Existing input structure in ChaNGa:

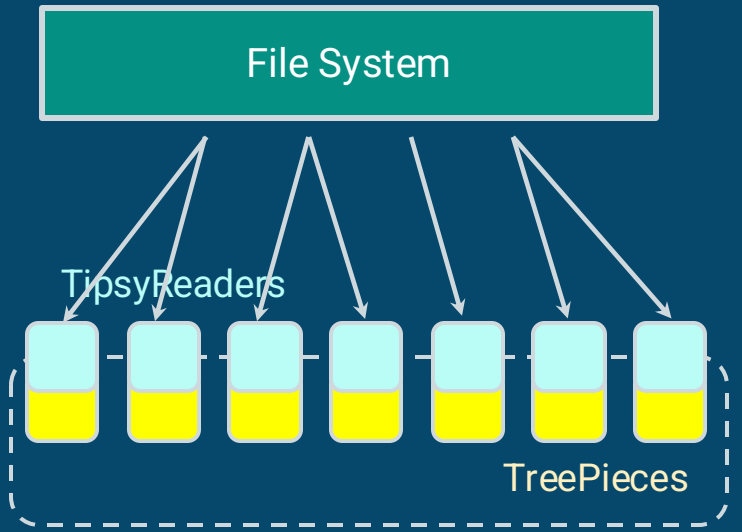
- All input happens before computation
- TreePieces collectively load input file
- Each TreePiece creates a TipsyReader (plain c++ objects)
- TipsyReader reads via std::ifstream
- Then TreePieces do other work



Applying CkIO: ChaNGa

Opportunities for optimization:

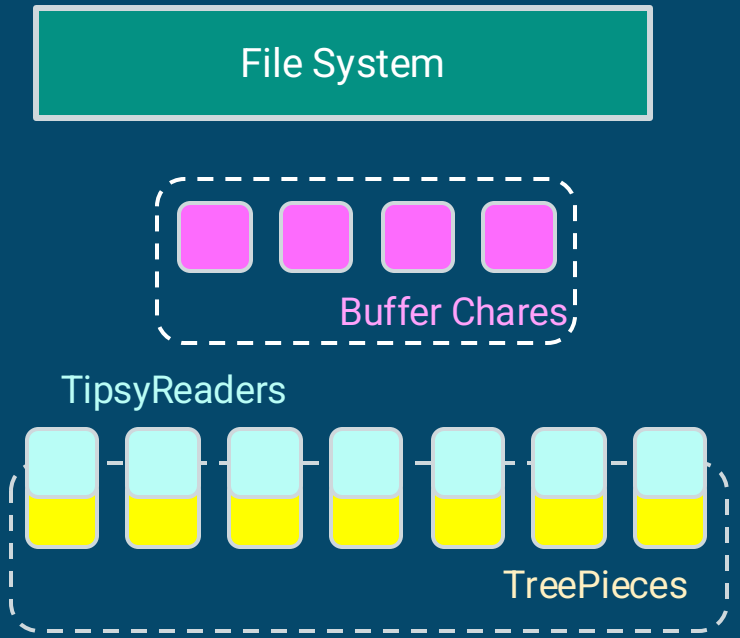
- No overlap



Applying CkIO: ChaNGa

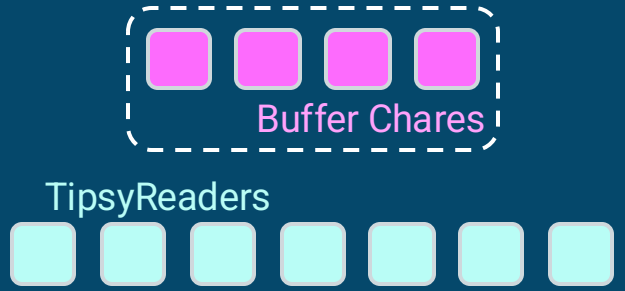
Opportunities for optimization:

- No overlap
- But separation of concerns can still provide benefit



Integrating FileReader into TipsyReader

```
1. class AbstractReader {
2. public:
3.     …
4.     virtual AbstractReader &read(char *s, size_t size);
5.     virtual bool operator!() const;
6.     virtual size_t tellg();
7.     virtual AbstractReader &seekg(size_t pos);
8.     virtual AbstractReader &seekg(size_t pos, std::ios_base::seekdir dir);
9. };
10.
11. class CkIOReader : public AbstractReader {…}    // wraps a FileReader
12. class StreamWrapper : public AbstractReader {…}   // wraps an ifstream
```




```
fileReader->read(reinterpret_cast<char *>(&gp), gas_particle::sizeBytes);
```

```
abstractReader->read(reinterpret_cast<char *>(&gp), gas_particle::sizeBytes);
```

Integrating FileReader into ChaNGa

```
treeProxy.loadTipsy(basefilename, dTuFac, bInDPos, bInDVel, CkCallbackResumeThread());
```

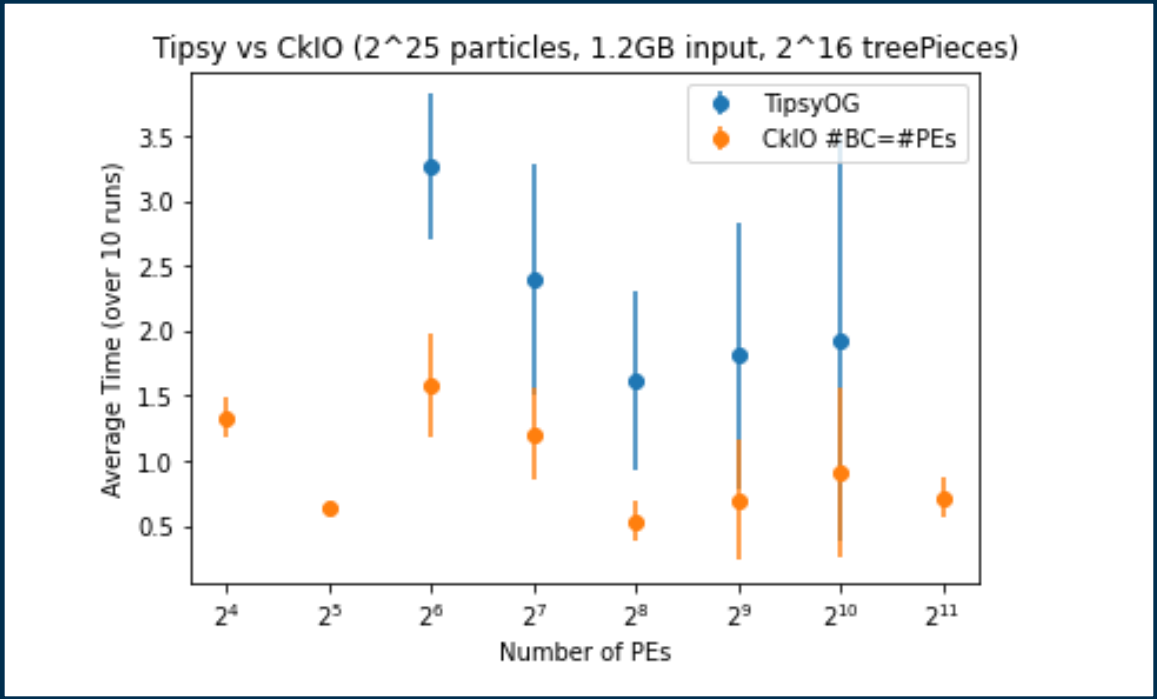
```
Ck::IO::Options opts;
opts.numReaders = ❓;

Ck::IO::open(basefilename.c_str(), CkCallbackResumeThread((void *&)fmsg), opts);
Ck::IO::startReadSession(fmsg->file, s.st_size, 0, CkCallbackResumeThread((void *&)smsg));
```

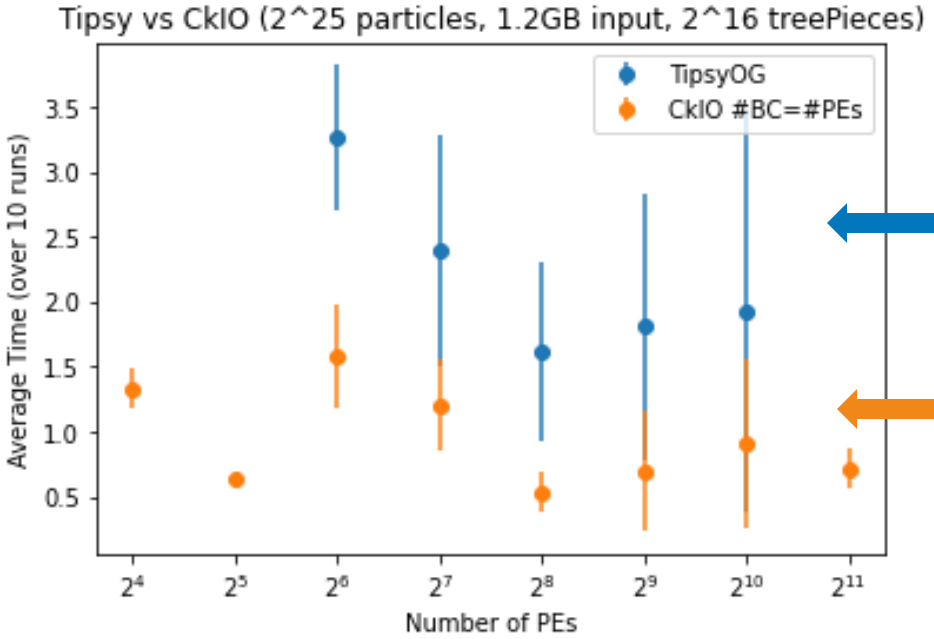
```
treeProxy.loadTipsy(basefilename, dTuFac, bInDPos, bInDVel, smsg->session,
                    CkCallbackResumeThread());
```

```
entry[threaded] void loadTipsy(const std::string &filename,
                               const double dTuFac,
                               const bool bDoublePos,
                               const bool bDoubleVel,
                               const CkCallback &cb);
```

ChaNGa + CkIO: Preliminary Results



ChaNGa + CkIO: Preliminary Results



naive Tipsy performs unexpectedly well (compared to our naive benchmarks)

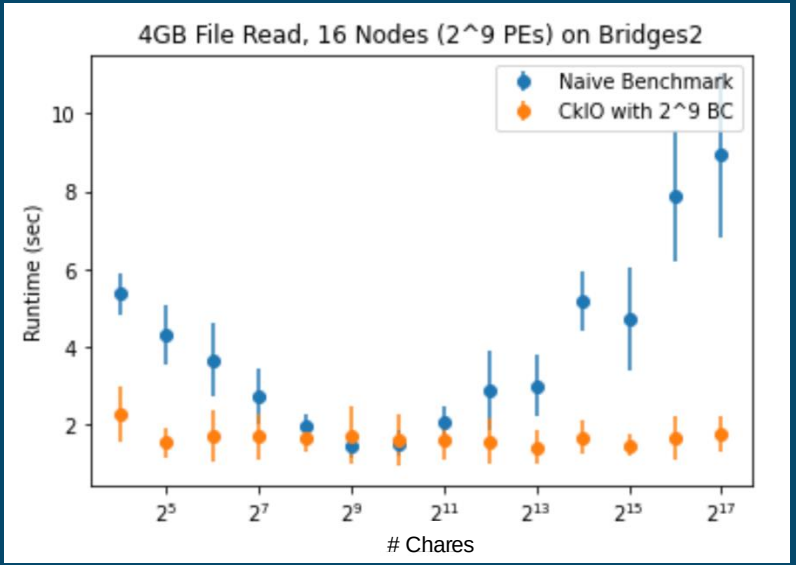
CkIO scaling not super smooth

Conclusion

- CkIO basics: separation of concerns
- Concurrent background work
- CkIO FileReader: supporting streaming reads
- CkIO Core and FileReader API
- ChaNGa WIP
- Future work: automation of **buffer chare selection** considering...
  - #PEs
  - #nodes
  - file size

Benchmark Details
Naive Input vs Core CkIO

- Bridges2 (Pittsburgh Supercomputing Center)
- 4GB File on Lustre Filesystem
- 16 CPU Nodes
- Utilizing 32 cores on each node (out of 64 total)



Benchmark Details Background Work

- Background work spins for approx. 10 µsec before releasing control
  - via Charm++ threads and runtime system
- CkIO monitor thread implemented via future::wait_for() and CthYield()