QDP++ and Chroma

Balint Joo

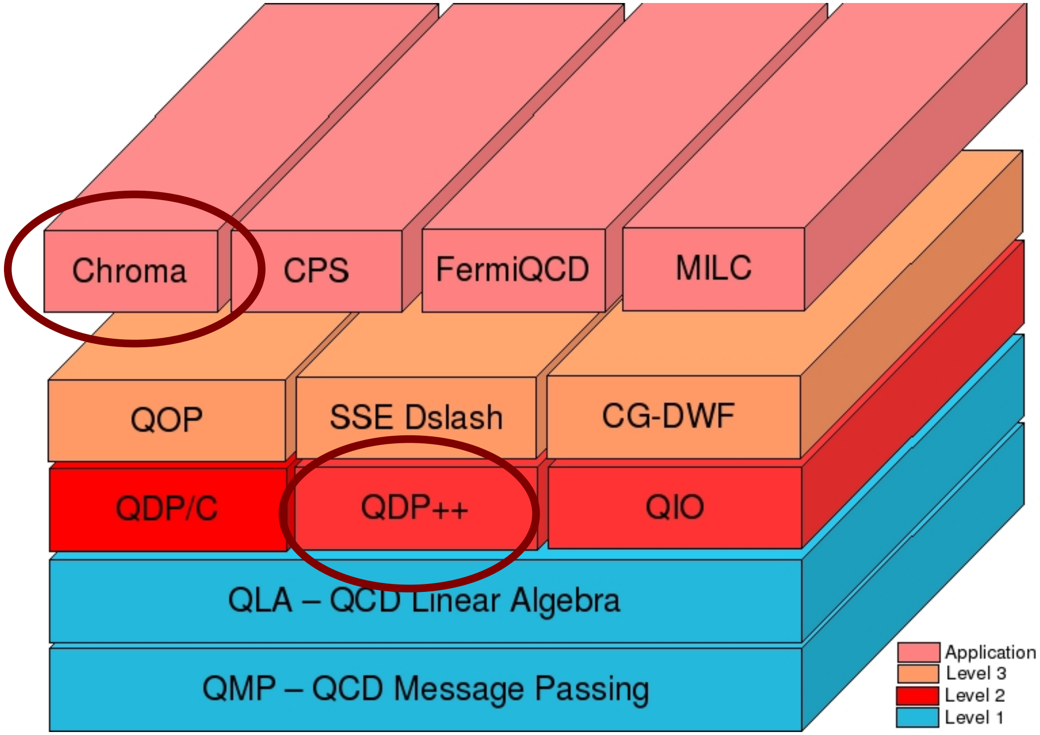
Jefferson Lab

HackLatt '08, Edinburgh

Contents

- High Level View of QDP++ and Chroma
  - What they are, what they do etc
  - Software Maps
  - Getting, Building, Testing(?)
  - Using (?)
- QDP++ Details
  - Data Parallel Primitives
  - Subsets, Checkerboarding
  - I/O with XML, QIO

QDP++ and Chroma What Are They



- Chroma is a lattice QCD framework written in C++
- Provides a library and some applications to do LQCD
- Relies on QDP++ to provide a data parallel platform
- Relies on 3^{rd} party libraries for numerically intensive work

- QDP++ is platform for data parallel QCD computations
- It provides, expressions, shifts, memory management and I/O
- Relies on QIO library for binary I/O, QMP library for parallel communications

Use of Chroma in the US

- USQCD Spectrum Project
  - Anisotropic Wilson/Clover Gauge Generation with (R)HMC algorithm
  - Wilson Clover Spectroscopy, photoproduction etc
- USQCD HASTE Project (now RBC-LHPC-UKQCD megacollaboration ?)
  - Domain Wall Fermion Forward/Sequential Propagator calculations
  - Hadron Structure (GPDA-s)
- USQCD NPLQCD Project
  - Domain Wall Fermion Propagator Calculations
- Algorithmic Studies
  - New inversion algorithms, preconditionings, dilution etc.

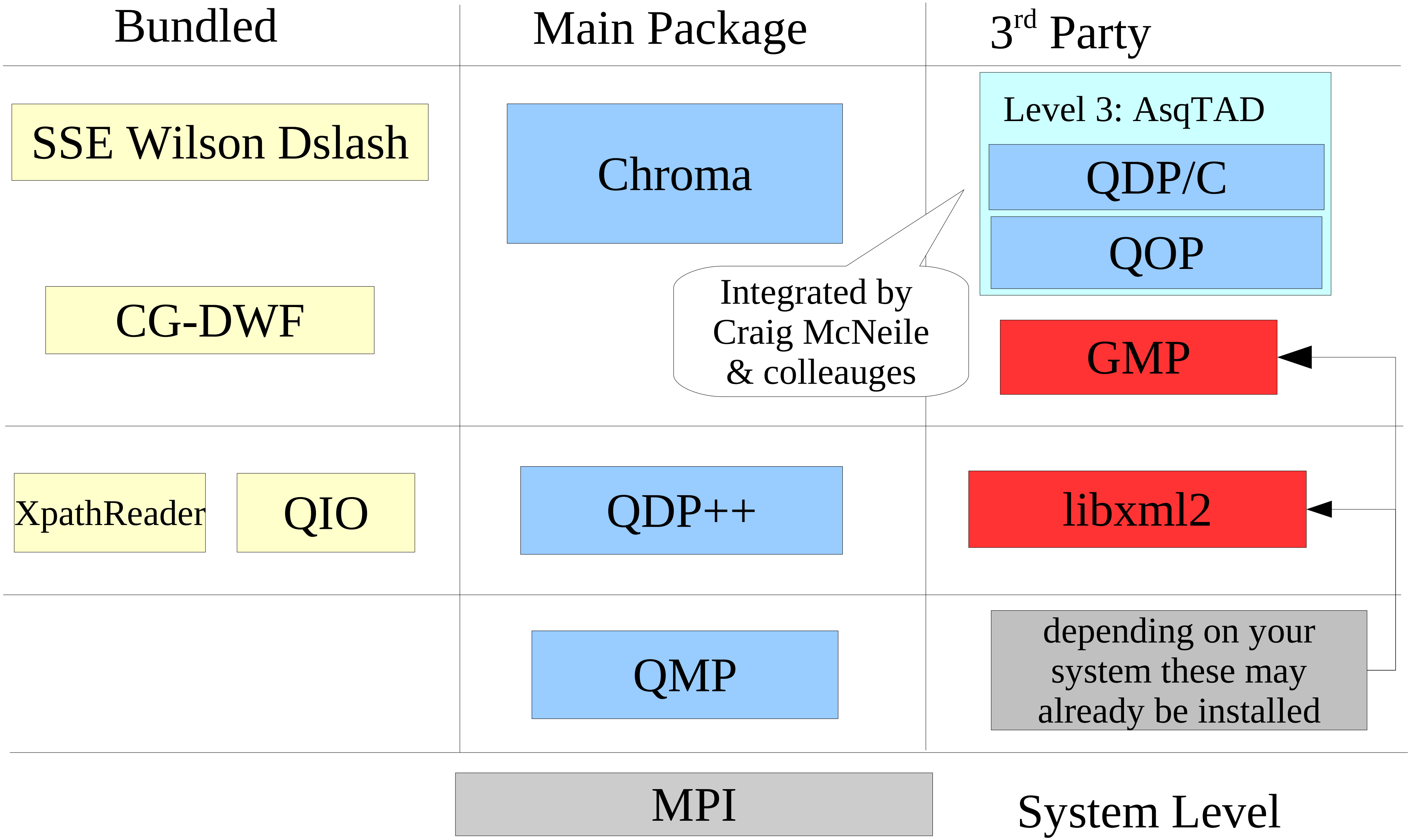
Current List of Target Machines

- SSE2 Based Machines
  - Intel/Opteron workstations
  - Intel/Opteron clusters
  - Cray XT series supercomputers
  - QMP typically over MPI (MVIA-Mesh will deprecate)
  - Fast Dslash and some SSE optimized ops in QDP++
  - Fast DWF CG propagator solver from MIT
- BlueGene/L Machines
  - QMP typically over MPI (native QMP does exist)
  - Fast Bagel based Dslash from Peter Boyle
  - Fast QDP++ operations and Clover term coded using BAGEL Generator from Peter.
- QCDOC Similar setup to BlueGene/L

Future/Development ports

- BlueGene/P
  - Start with BlueGene/L setup
  - Will do our best to integrate new components when/if we have access to them.
- Large Multi-Core Machines of the future
  - Ranger at TACC (port in progress)
  - whatever comes along?

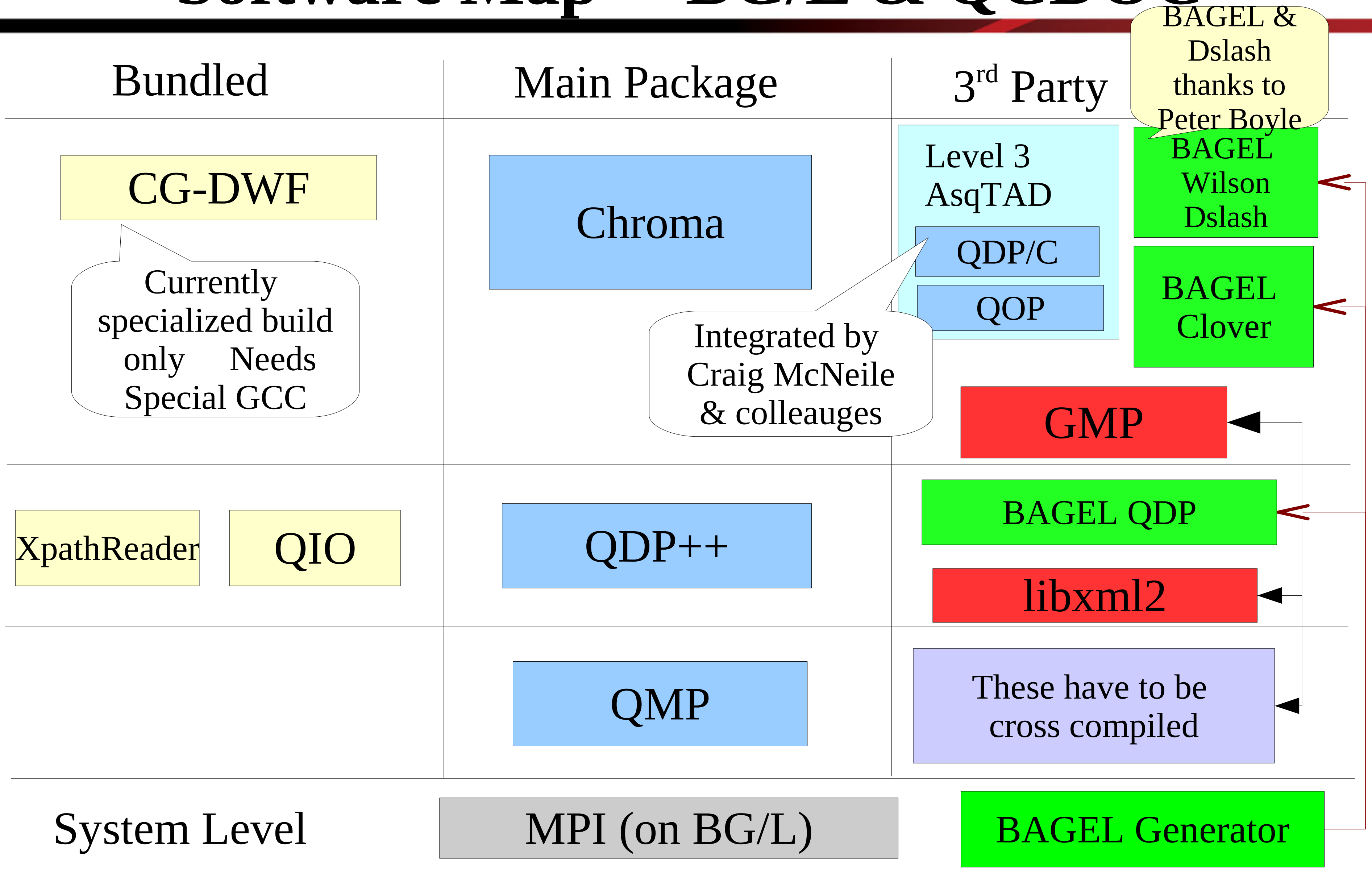
Software Map SSE Based Systems



Bundled	Main Package	3rd Party
SSE Wilson Dslash CG-DWF	Chroma	Level 3: AsqTAD QDP/C QOP GMP
XpathReader QIO	QDP++	libxml2
	QMP	depending on your system these may already be installed
	MPI	System Level

Integrated by Craig McNeile & colleauges

Software Map   BG/L & QCDOC



Bundled

Main Package

3rd Party

BAGEL & Dslash thanks to Peter Boyle

CG-DWF

Currently specialized build only   Needs Special GCC

Chroma

Level 3 AsqTAD

QDP/C

QOP

Integrated by Craig McNeile & colleauges

BAGEL Wilson Dslash

BAGEL Clover

GMP

XpathReader

QIO

QDP++

BAGEL QDP

libxml2

QMP

These have to be cross compiled

System Level

MPI (on BG/L)

BAGEL Generator

Getting the bits and Pieces

- QMP, QDP++ and Chroma (and bundled packages) and BAGEL QDP, BAGEL Clover
  - from anonymous CVS:
    - Root :pserver:anonymous@cvs.jlab.org:/group/lattice/cvsroot
    - Modules: bagel_qdp, bagel_clover, qmp, qdp++, chroma
  - from USQCD Web Page:
    - http://usqcd.jlab.org/usqcd-software
- BAGEL and Wilson Dslash from Peter's web page
    - http://www.ph.ed.ac.uk/~paboyle/bagel/Bagel.html
- GMP from: http://www.swox.com/gmp
- LibXML2 from: http://www.xmlsoft.org

Building

- Packages have a configure ; make ; make install style
  - but configure has many options that need to be set
- Simplest builds
  - use mpicc, mpicxx
  - build qmp, qdp++, chroma in sequence (GMP, libxml are already installed)
- Most complicated builds (BlueGene/L / QCDOC)
  - build libxml2, GMP, bagel, bagel_qdp, etc...
- This is a real pain to manage
- Jlab CVS module: jlab-standard-chroma-build attempts to automate the build workflow (download, configure, etc)
  - most build failures tend to be automake related :(
  - QCDOC poses some special problems...

Compiler Versions etc

- We made the move to gcc-4.x series of compilers on SSE Based Platforms
- QMP and QDP++ should now also compile with Intel/Pathscale/PGI compilers   SSE may not be available with PGI   this needs investigation
- We've switched to automake-1.10 (automake versions tend to move along with our cluster upgrades)

Some Configure Options for QDP++

SZIN Leftovers:
scalar   workstation builds
parscalar -   parallel machine with scalar processing elements   - MPI/QMP

- QDP++
  - --prefix=<install location>
  - --enable-parallel-arch=(scalar|parscalar)
  - --enable-precision=(single|double)
  - --with-libxml2=<location of libxml2 installation>
  - --with-qmp=<location of QMP installation>
- SSE Options
  - --enable-sse2
- BAGEL Related Options
  - --with-bagel-qdp=<location of Bagel QDP installation>
- QCDOC Specific Option
  - --enable-qcdoc (QCDOC Specific memory allocator)
- CXXFLAGS=   -O2 -finline-limit=50000   CFLAGS=   -O2

Some Configure Options for Chroma

- Chroma
  - --prefix=<install location>
  - --with-qdp=<location of QDP++ installation>
  - --with-gmp=<location of GMP library>
- SSE Options
  - --enable-sse-wilson-dslash (SSE Only)
  - --enable-cg-dwf=sse (SSE Domain Wall Solver)
  - --enable-cg-dwf-lowmem (Low memory DWF Solver)
  - --with-qmp=<location of QMP   DWF Solver needs this>

NEW

- BAGEL related options
  - --with-bagel-wilson-dslash=<location of BAGEL dslash>
  - --enable-bagel-wilson-dslash-sloppy <Sloppy Dslash>
  - --with-bagel-clover=<Location of BAGEL Clover>

NEW

- CXXFLAGS=   CFLAGS=   (disables default -g flag)

Hints to Help You Build

- A (slightly out of date) set of installation notes on the USQCD Web Site
- The jlab-standard-chroma-build package in the Chroma CVS server
  - cvs checkout jlab-standard-chroma-build
  - cd jlab-standard-chroma-build
  - Need to configure this with ./configure
  - Useful Options are
    - --enable-install-root=<root of install tree>
    - --enable-qdp-version=<version of QDP++ to use>
    - --enable-chroma-version=<version of Chroma to use>
    - --enable-parallel-make=<no of threads for make>

More on Jlab Standard Chroma Build

- The standard build is organised in directories as:
  - package/machine/build-type
  - eg: Single precision, parscalar build of QDP++ for Jlab 7n machine: qdp++/ib-7n/parscalar-ib-7n
  - eg: Scalar build of Chroma: chroma/scalar/scalar
  - These triplets map directly onto directory names
    - cd qdp++/ib-7n/parscalar-ib-7n
    - There is a build/ subdirectory here.
      - cd ./build/
      - There is   configure.sh   script here.
        - In the configure.sh script are useful flags for this target
- You can use this to get a handle on what flags you want where

More on Jlab Standard Chroma Build

- Can actually build the code with this too
  - look at eg: xt3_build.sh   complete the configure line then run on your favourite Cray XT machine
  - look at eg: bgl_build.sh   complete the configure line then run on your favourite BG/L
- Logs of builds kept in ./logs/package
- Successful builds installed in
  - <install-root>/package/version/build-type
  - eg: /home/bjoo/install/qdp++/qdp1-25-1/parscalar-ib-7n
- ./build.sh package/machine/build-type will attempt to download package source for you
- Package source is downloaded to package/package subdirectory of jlab-standard-chroma-build
  - In case you need the source to run automake etc.

Testing

- Chroma contains many regression tests (over 100) mostly in
  - chroma/tests/chroma (main chroma measurement tests)
  - chroma/tests/t_leapfrog (MD tests)
  - chroma/tests/hmc (HMC tests)
- Test comprised of:
  - executable (chroma, hmc, t_leapfrog etc)
  - input param file (XML)
  - expected output file (XML)
  - output comparison metric file (XML)
    - used to check code output against expected output
- Need xmldiff utility to check output against expected output
- Need some infrastructure to run through all tests

How to run the tests

- Have xmldiff installed and on your PATH before configuring chroma
- After chroma is configured it will produce a script called RUN   you may need to edit this. In the case of a scalar machine it may be as simple as
  - #!/bin/bash
  - $*
- For an MPI machine with 4 CPUs you may want it to look like
  - #!/bin/bash
  - mpirun -np 4 $*
- Then in an interactive session you should run 'make xcheck'
- There are complicated ways to run this through batch queues as well...

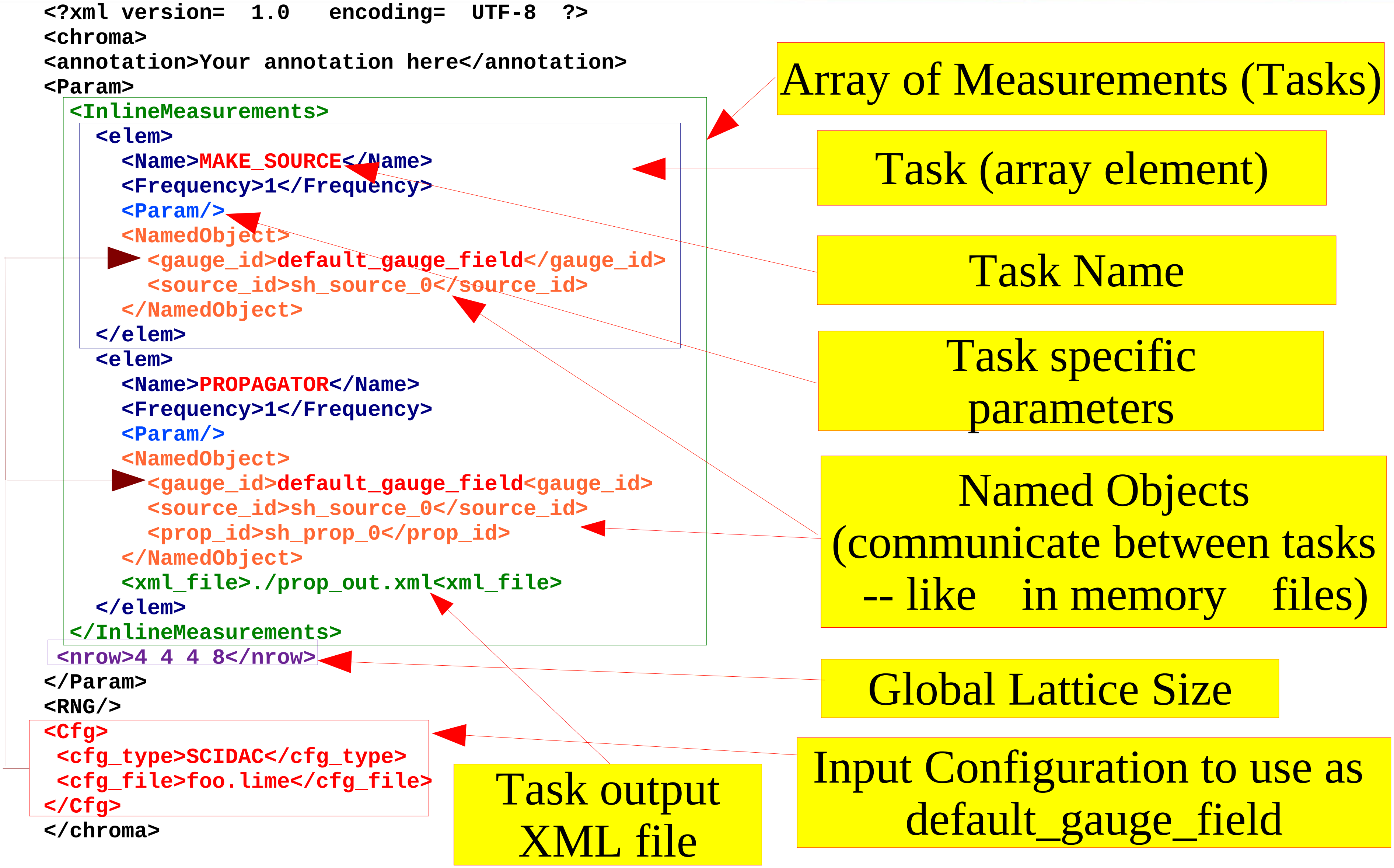
Linking against an installed Chroma

- Suppose chroma is installed in /foo/chroma
- Use script chroma-config in /foo/chroma/bin
  - CXX=`chroma-config --cxx`
  - CXXFLAGS=`chroma-config --cxxflags`
  - LDFLAGS=`chroma-config --ldflags`
  - LIBS=`chroma-config --libs`
- Compile your program (prog.cc) with:
  - $(CXX) $(CXXFLAGS) prog.cc $(LDFLAGS) $(LIBS)
  - NB: Ordering of flags may be important.

Running Chroma Applications

- Measurement Application: chroma
- Gauge Generation Applications: hmc and purgaug
- Installed in same place as chroma-config
  - eg: /foo/chroma/bin
- Typical usage flags (-i, -o, -geom):
  - ./chroma -i in.xml -o out.xml -geom Px Py Pz Pt
  - in.xml   Input Parameter XML File
  - out.xml   Output XML Log File
  - Px Py Pz Pt the (possibly virtual) Processor Geometry (eg -geom   4 4 8 8   for QCDOC Rack)

Chroma XML Input File Structure



```
<?xml version=  1.0   encoding=  UTF-8  ?>
<chroma>
<annotation>Your annotation here</annotation>
<Param>
  <InlineMeasurements>
    <elem>
      <Name>MAKE_SOURCE</Name>
      <Frequency>1</Frequency>
      <Param/>
      <NamedObject>
        <gauge_id>default_gauge_field</gauge_id>
        <source_id>sh_source_0</source_id>
      </NamedObject>
    </elem>
    <elem>
      <Name>PROPAGATOR</Name>
      <Frequency>1</Frequency>
      <Param/>
      <NamedObject>
        <gauge_id>default_gauge_field<gauge_id>
        <source_id>sh_source_0</source_id>
        <prop_id>sh_prop_0</prop_id>
      </NamedObject>
      <xml_file>./prop_out.xml<xml_file>
    </elem>
  </InlineMeasurements>
 <nrow>4 4 4 8</nrow>
</Param>
<RNG/>
<Cfg>
 <cfg_type>SCIDAC</cfg_type>
 <cfg_file>foo.lime</cfg_file>
</Cfg>
</chroma>
```

Array of Measurements (Tasks)

Task (array element)

Task Name

Task specific parameters

Named Objects (communicate between tasks -- like in memory files)

Global Lattice Size

Task output XML file

Input Configuration to use as default_gauge_field

Where to find XML examples

- Measurement Tasks for:
- Numerous Measurement Task Examples in
  - chroma/tests/chroma/hadron/
  - sources, smearings, propagators, spectroscopy, 3pt functions, eigenvalues
  - Reading and Writing Named Objects
- Also MD and HMC input files in
  - chroma/tests/t_leapfrog
  - chroma/tests/hmc
- Input file names usually contain the string ini

QDP++ Details

- QDP++ Is the Foundation of Chroma
- It provides
  - way to write the maths of lattice QCD without looping over indices (expressions)
  - Custom memory allocation possibilities
  - I/O facilities (XML and Binary)
- You can do Lattice QCD just in QDP++ without Chroma
  - See: Lectures from the 2007 INT Lattice Summer School
  - but you'd need to write a whole lot of infrastructure that comes for free with Chroma

QDP++ Templated Types

- In QDP++ tries to capture the index structure of our lattice fields:

	Lattice	Spin	Colour	Reality	BaseType
Real	Scalar	Scalar	Scalar	Real	REAL
LatticeColorMatrix	Lattice	Scalar	Matrix(Nc,Nc)	Complex	REAL
LatticePropagator	Lattice	Matrix(Ns,Ns)	Matrix(Nc,Nc)	Complex	REAL
LatticeFermionF	Lattice	Vector(Ns)	Vector(Nc)	Complex	REAL32
DComplex	Scalar	Scalar	Scalar	Complex	REAL64

- To do this we use C++ templated types:

```
typedef OScalar < PScalar    < PScalar<      RScalar <REAL>    >    > > Real;
typedef OLattice< PScalar    < PColorMatrix< RComplex<REAL>, Nc>    > > LatticeColorMatrix;
typedef OLattice< PSpinMatrix< PColorMatrix< RComplex<REAL>, Nc>, Ns> > LatticePropagator;
```

- QDP++ and Portable Expression Template Engine:
  - Provide expressions and recursion through type structure

QDP++ and Expressions

" Use convenient mathematical expressions:

```
LatticeFermion x,y,z;
Real a = Real(1);
gaussian(x);
gaussian(y);
z = a*x + y;
int mu, nu;
multi1d<LatticeColorMatrix> u(Nd);
Double tmp =  sum( real( trace(  u[mu]
                   *shift(u[nu],FORWARD,mu)
                   * adj( shift(u[mu],FORWARD,nu) )
                   *adj(u[nu])
                     )
                 )
              );
```

Wowee! No indices or for loops!

multi1d<T> =
1 dimensional array
of T

Lattice Wide Shifts
FORWARD =
from x+mu

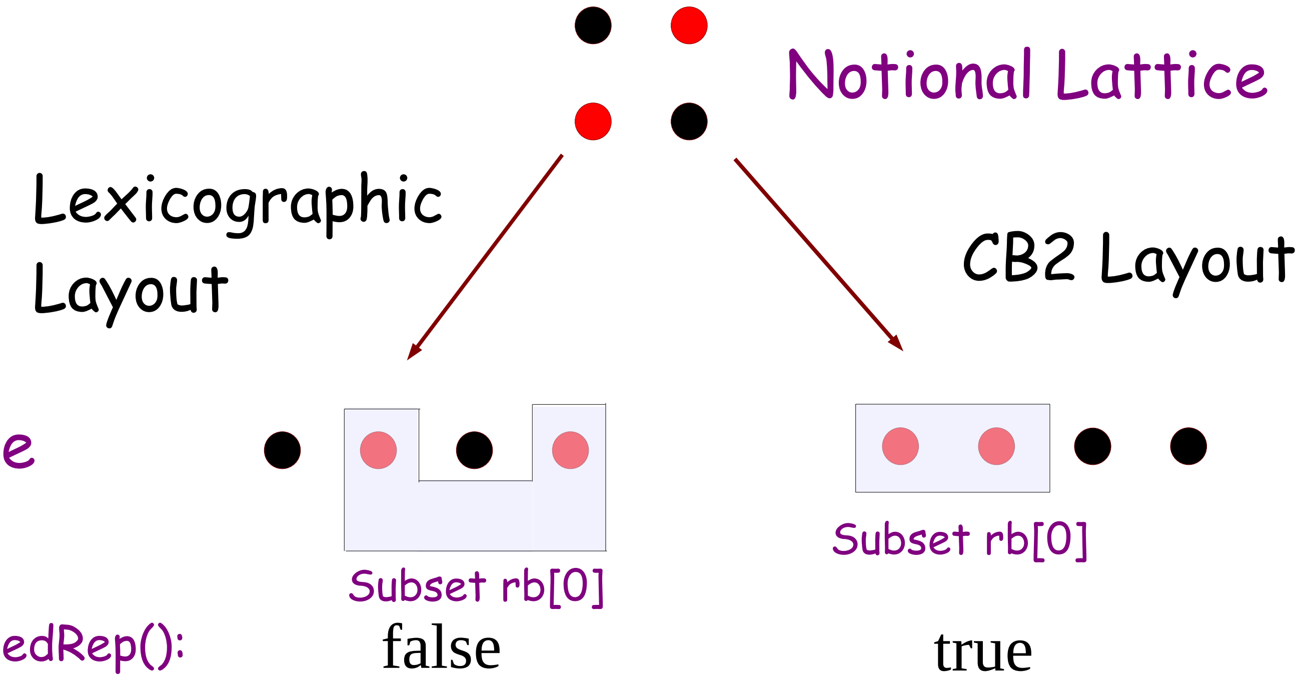
Functions

but array indices are explicit

Subsets and Layouts

- Subset: Object that identifies a subset of sites
- Can be predefined: eg rb is red-black colouring
- Can be contiguous or not (s.hasOrderedRep() == true or not)
- Layout is an ordering of sites in memory (compile time choice)
- Same subset may be contiguous in one layout and not in another

R.I.P.
OrderedSet
OrderedSubset



Using Subsets

In expressions, subset index is on the target

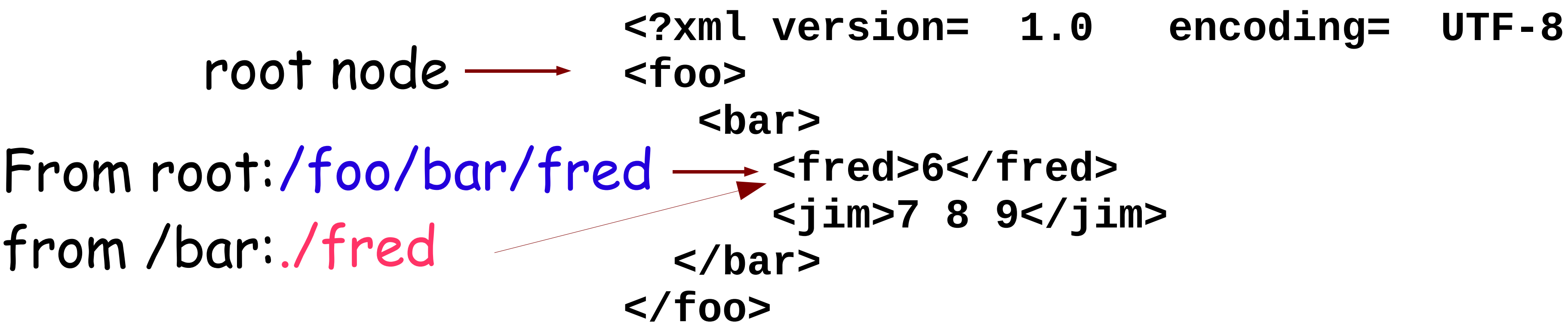
bar[rb[1]] = foo; // (Copy foo's rb[1] subset to bar's)

- Users can define new sets
- Layout chosen at compile time with configure switch
  - Default is CB2 (2 colour checkerboard) layout
- Layout needs to be initialized on entry to QDP++

```
multi1d<int> nrow(4);
nrow[0]=nrow[1]=nrow[2]=4; nrow[3]=8;
Layout::setLattSize(nrow);
Layout::create();
```

QDP++ and XML

- No static data binding in QDP++
- Treat documents as amorphous (contain anything)
- Interrogate documents using XPath.
- Root of Path expression is context node



root node

From root:/foo/bar/fred

from /bar:./fred

```
<?xml version= 1.0  encoding= UTF-8
<foo>
   <bar>
      <fred>6</fred>
      <jim>7 8 9</jim>
  </bar>
</foo>
```

Reading XML

```
XMLReader r(  filename  );

Double y;
multi1d<Int> int_array;
multi1d<Complex> cmp_array;

try {
 read(r,   /foo/cmp_array  , cmp_array);

 XMLReader new_r(r,   /foo/bar  );

 read(new_r,   ./int_array  , int_array);
 read(new_r,   ./double  , y);
}
catch( const std::string& e) {
 QDPIO::cerr <<   Caught exception:
             << e <<endl;
 QDP_abort(1);
}
```

Absolute Path Read

New Context

Relative Paths

QDP++ error "stream"

Array markup of non-simple types

```
<?xml version=  1.0
      encoding=  UTF-8  ?>
<foo>
 <cmp_array>
  <elem>
   <re>1</re>
   <im>-2.0</im>
  </elem>
  <elem>
   <re>2</re>
   <im>3</im>
  </elem>
 </cmp_array>
 <bar>
   <int_array>2 3 4 5</int_array>
   <double>1.0e-7</double>
 </bar>
</foo>
```

Array markup for simple types

Writing XML

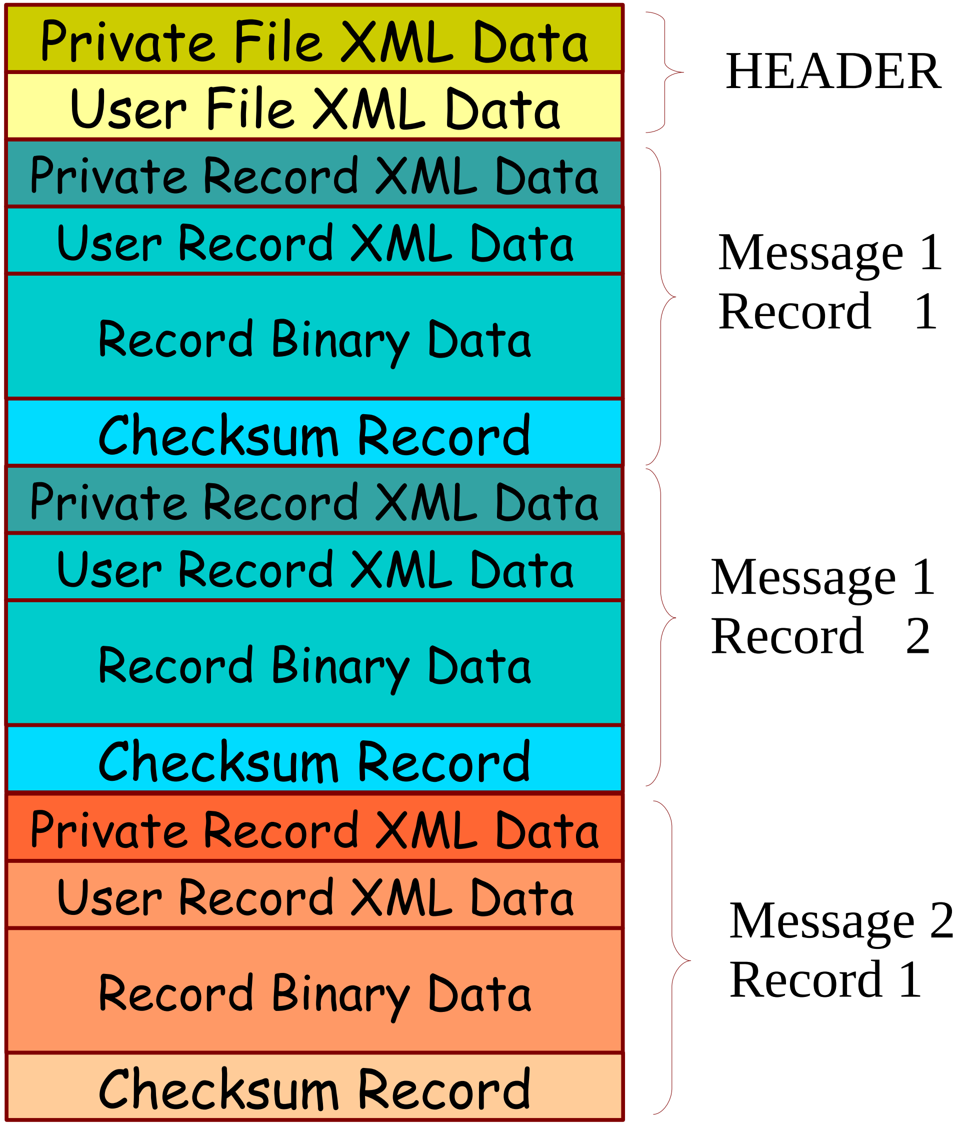
```
// Write to file
XMLFileWriter foo(  ./out.xml  );
push(out,   rootTag  );
int x=5;
Real y=Real(2.0e-7);
write(foo,   xTag  , x);
write(foo,   yTag  , y);
pop(out);
```

```
<?xml version=  1.0  ?>
<rootTag>
  <xTag>5</xTag>
  <yTag>2.0e-7</yTag>
</rootTag>
```

```
// Write to Buffer
XMLBufferWriter foo_buf;
push(foo_buf,   rootTag  );
int x = 5;
Real y = Real(2.0e-7);
write(foo_buf,   xTag  ,  x);
write(foo_buf,   yTag  , y);
pop(foo_buf);
QDPIO::cout <<   Buffer contains   <<foo_buf.str()
            << endl;
```

get buffer content

QIO and LIME Files



Private File XML Data

User File XML Data

HEADER

Private Record XML Data

User Record XML Data

Record Binary Data

Checksum Record

Message 1
Record 1

Private Record XML Data

User Record XML Data

Record Binary Data

Checksum Record

Message 1
Record 2

Private Record XML Data

User Record XML Data

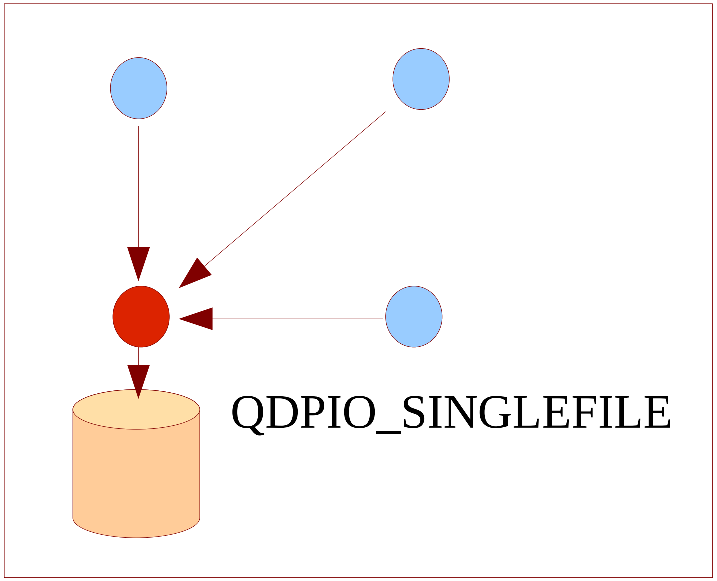
Record Binary Data

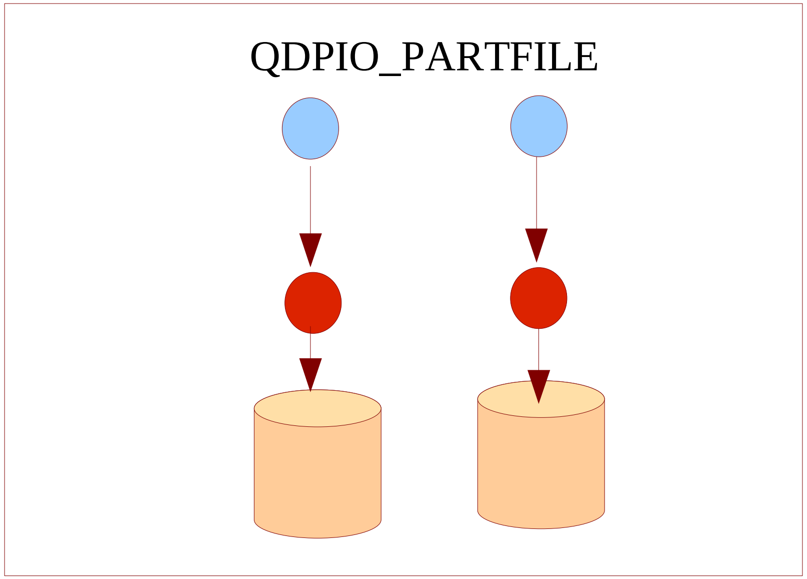
Checksum Record

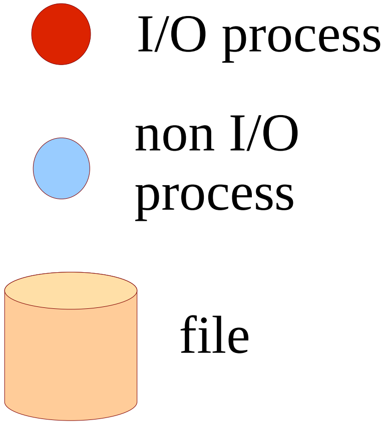
Message 2
Record 1

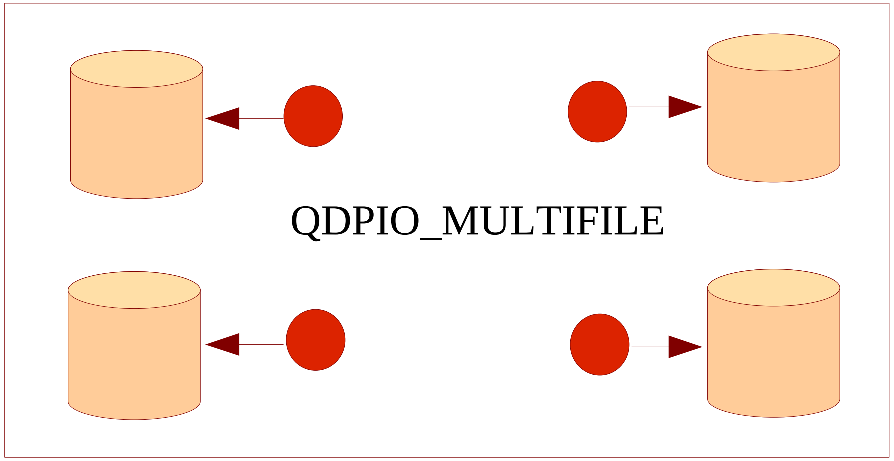
- QIO works with record oriented LIME files
- LIME files made up of messages
- messages are composed of
  - File XML records
  - Record XML records
  - Record Binary data
- SciDAC mandates checksum records
- ILDG mandates certain records

Three modes of writing in QIO







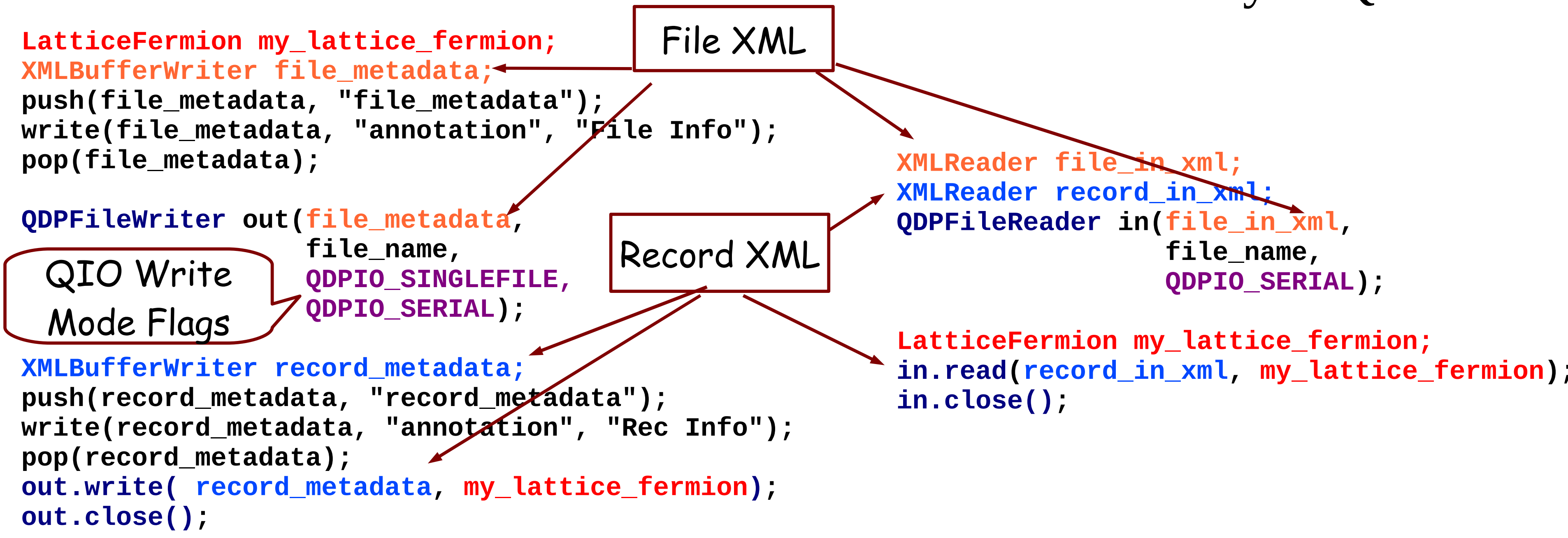


Writing/Reading patterns for QDPIO_SERIAL

QDP++ interface to QIO

- Write with QDPFileWriter
- Must supply user file and user record XML as XMLBufferWriter-s

- Read with QDPFileReader
- User File XML and User Record XML returned in XML Readers
- Checksum/ILDG details checked internally to QIO



```
LatticeFermion my_lattice_fermion;
XMLBufferWriter file_metadata;
push(file_metadata, "file_metadata");
write(file_metadata, "annotation", "File Info");
pop(file_metadata);

QDPFileWriter out(file_metadata,
                  file_name,
                  QDPIO_SINGLEFILE,
                  QDPIO_SERIAL);

XMLBufferWriter record_metadata;
push(record_metadata, "record_metadata");
write(record_metadata, "annotation", "Rec Info");
pop(record_metadata);
out.write( record_metadata, my_lattice_fermion);
out.close();
```

```
XMLReader file_in_xml;
XMLReader record_in_xml;
QDPFileReader in(file_in_xml,
                 file_name,
                 QDPIO_SERIAL);

LatticeFermion my_lattice_fermion;
in.read(record_in_xml, my_lattice_fermion);
in.close();
```

ILDG Support (?)

- The underlying QIO support layer can handle ILDG format.
- Specialization of write() function and type traits:
  - multi1d<LatticeColorMatrix> always written in ILDG format.
  - ILDG dataLFN is optional argument to the QDPFileWriter constructor.
- nersc2ildg program provided in examples/ directory
- lime_contents and lime_extract_record programs from QIO automatically built and installed

Custom Memory Allocation

- Occasionally need to allocate/free memory explicitly e.g. to provide memory to external library.
- Memory may need custom attributes (eg fast/communicable etc)
- Memory may need to be suitably aligned.
- May want to monitor memory usage

Allocate memory from desired pool if possible, with alignment suitable to pool

```
pointer=QDP::Allocator::theQDPAllocator::Instance().allocate( size,
                                                QDP::Allocator::FAST);

QDP::Allocator::theQDPAllocator::Instance()::free(pointer);
```

Namespace

Get reference to allocator

MemoryPoolHint (attribute)

Move To Fast Memory (If you have it)

- QCDOC/SP inspired construct (copy to CRAM?)
- moves/copies data to/from fast memory (eg EDRAM)
- NOP on machines where there is no fast memory

```
LatticeFermion x;

moveToFastMemoryHint(x);

// Do some fast computation here

revertFromFastMemoryHint(x,true);
```

Accelerate x!
Do not copy contents.
Contents of x lost

Bring x back to slow memory. Copy contents

Efficiency Considerations

- PETE eliminates Lattice sized temporaries
- But still there are temporaries at the site level
  - This really hurts performance
- Workaround: Cliché Expressions Optimized
  - eg: a*x + y, a*x + P y, norm2(), innerProduct() etc.
  - Optimizations in C, SSE and bagel_qdp library
- Non optimized expression still slow:
  - eg: a*x+y+z etc.

Summary

- QDP++ and Chroma are quite mature now
- In this presentation I have discussed
  - basic usage (getting, building, testing, using)
  - a reasonably thorough description of QDP++
- Stay tuned for:
  - aspects of the chroma class structure
  - tutorial exercises
  - a presentation on writing XML input files