POLYMORPHIC INPUT ITERATORS

C++ USERGROUP KARLSRUHE
JANUAR 9TH 2019
DAVID FARAGÓ

ITERATOR DESIGN PATTERN

Gang of Four

Intent

... access the elements ...without exposing its underlying representation.

Motivation

...the client commits to a particular collection. It would be better if we could change the collection without changing client code. We can do this by generalizing the iterator concept to support polymorphic iteration.

Bjarne Stroustrup

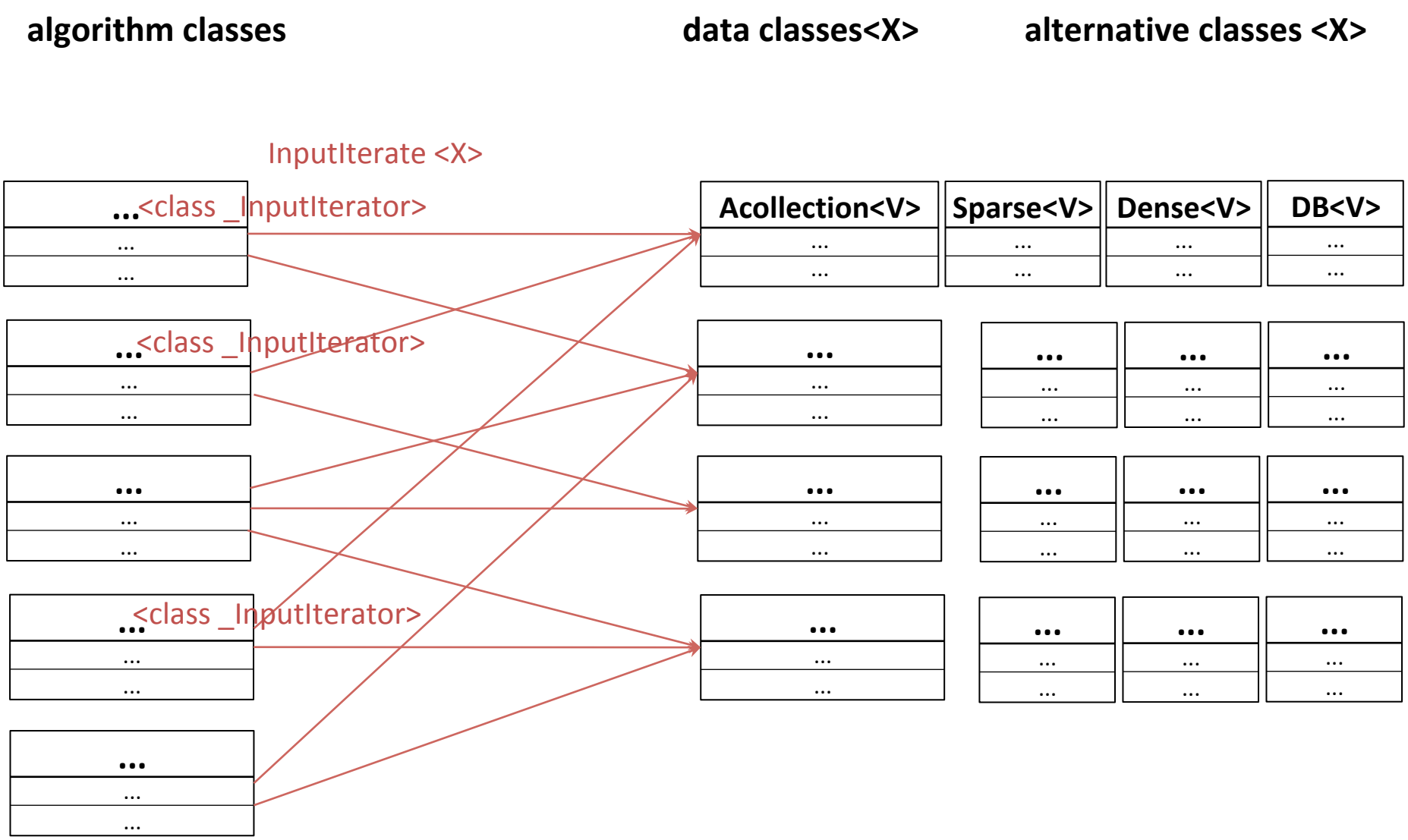
Polymorphism

Providing a single interface to entities of different types.

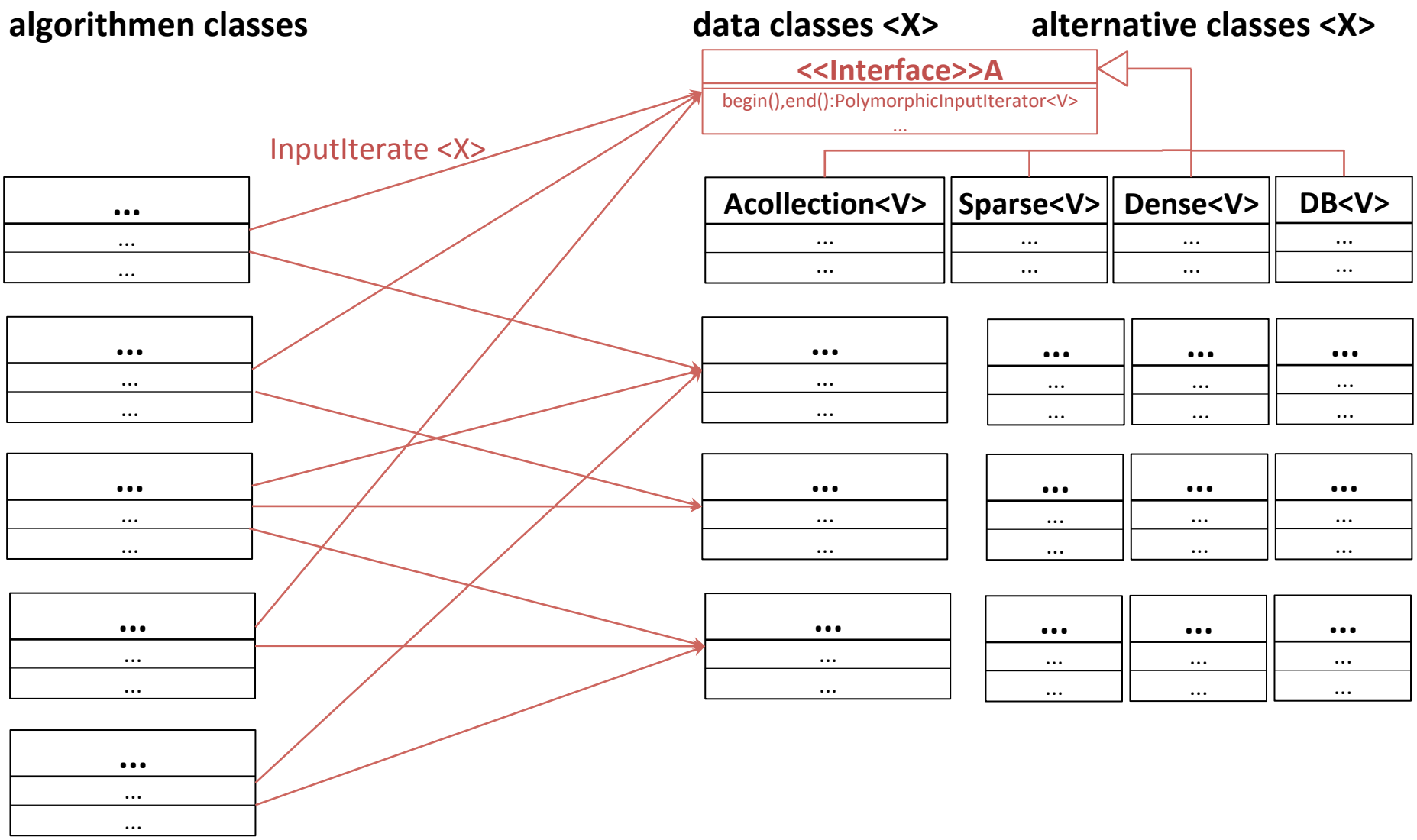
INPUT ITERATOR CONCEPTS

Legacy	std::experimental::ranges
	WeaklyInc->Semiregular->DefaultConstructible
CopyConstructible, ConversionConstructible to const CopyAssignable Destructible	WeaklyInc->Semiregular->Copyable->CopyConstructible WeaklyInc->Semiregular->Copyable->Assignable WeaklyInc->Semiregular->Copyable->Movable WeaklyInc->Semiregular->Copyable->Assignable
Swappable	WeaklyInc->Semiregular->Copyable->Movable->Swappable
Dereferenceable, Dereference ->	Dereferenceable, Readable
pre-incrementable, pre-increment post-increment	WeaklyInc WeaklyInc
equality-comparable inequality-comparable	
std::iterator_traits<II> has the 5 member typedefs	std::iterator_traits<II> has the 5 member typedefs

MOTIVATION AT QPR



MOTIVATION AT QPR



IMPLEMENTATION

Polymorphism in C++ (but not macros, overloading, coersion)

	templates (e.g. CRTP)	std::variant (std::visit)	OOP (virtual)
ad-hoc vs. parametric	☺	☺ with lambda type deduction	☺
semantics	☺ value sematics, copying	☺ value sematics, copying	☹ reference sem., dyn. alloc.
change to existing code	☹	☺	☺
optimizations, runtime	☺ iff no code bloat	☺ but less, see next slides	☹ dynamic polymorphism
code complexity (readabilty, error messages, compile time)	☹	☺	☺

DATA MEMBER

```
// Copyright QPR Technologies 2018
template <class V>
class PII final
{
private:
    /*
     * Extension block: Add further iterator types here.
     *
     * For many iterators (e.g. llvm::SmallVector's),
     * no extension is necessary because they typedef down to a plain pointer.
     */
    using CvVectorIter = typename std::conditional_t<
        std::is_const_v<V>, typename std::vector<std::remove_cv_t<value_type>>::const_iterator,
        typename std::vector<std::remove_cv_t<value_type>>::iterator>;
    using CvListIter = typename std::conditional_t<
        std::is_const_v<V>, typename std::list<std::remove_cv_t<value_type>>::const_iterator,
        typename std::list<std::remove_cv_t<value_type>>::iterator>;
    using CvDboIter = typename std::conditional_t<
        std::is_const_v<V>, typename Wt::Dbo::collection<std::remove_cv_t<value_type>>::const_iterator,
        typename Wt::Dbo::collection<std::remove_cv_t<value_type>>::iterator>;

    using WrappedIterVarType = std::variant<pointer, CvVectorIter, CvListIter, CvDboIter>;
    /*
     * End of extension block
     */

    WrappedIterVarType m_wrappedIterVar;
```

SIGNIFICANT METHODS

```
template <typename V>
using ToggleConst =
    typename std::conditional_t<std::is_const_v<V>, std::remove_const_t<V>, const V>;
```

```
/// Generalized conversion ctor (not part of IIC) for the iterators that can be wrapped
template <typename BaseIter,
          typename = std::enable_if_t<std::is_convertible_v<BaseIter, WrappedIterVarType>>>
PII(BaseIter baseIter) : m_wrappedIterVar(std::move(baseIter))
{
}

/// IC ConversionConstructible from nonconst to const (but compiler error from const to nonconst).
///
/// This is called a generalized copy ctor, and requires declaring a normal copy ctor (see EC++ Item 45).
PII(const PII<ToggleConst<V>> &other)
    // unfortunately, we cannot delegate to generalized (conversion) ctor.
    : m_wrappedIterVar{
        std::visit([](auto wrappedIter) -> WrappedIterVarType { return WrappedIterVarType{wrappedIter}; },
                   other.m_wrappedIterVar)}
{
    static_assert(std::is_const_v<V>);
}

/// IC pre-incrementable and IIC pre-increment
PII &operator++()
{
    std::visit([](auto &&wrappedIter) { ++wrappedIter; }, m_wrappedIterVar);
    return *this;
}
```

PERFORMANCE

measured runtime of accumulating a vector with 3e+7 doubles (into double or ULL):

dynamically in ms (wall clock time),

statically in k cpu cycles (LLVM Machine Code Analyzer, see godbolt.org/z/H9bXPs)

clang (trunk)
-march=broadwell −O3

with –ffast-math

	0.0	0ULL
wrapped	212 20.5	277 22.4
direct	17 5.7	217 14.4

w/o –ffast-math

	0.0	0ULL
wrapped	196 20.5	279 22.4
direct	32 4.3	218 25.4

gcc (trunk)

	0.0	ULL
wrapped	166 2.2	186 4.2
direct	14 2.9	117 2.5

	0.0	ULL
wrapped	166 2.2	204 4.0
direct	44 2.4	119 2.7

FEEDBACK

LLVM-MCA VON VECTOR-TESTS

double with gcc (trunk) with -std=c++17 -O3 -march=broadwell

Wrapped:
Runtime for vector size 30.000.000: TODO
MCA: [0] Code Region - stopwatchedAccumulate
Iterations: 100
Instructions: 4200
Total Cycles: 2196
Total uOps: 5000
Dispatch Width: 6
uOps Per Cycle: 2.28
IPC: 1.91
Block RThroughput: 8.3
Resource pressure per iteration:

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
-	-	7.62	7.95	5.70	5.72	7.00	7.81	7.62	4.58

Wrapped for accumulationition into ULL:
Runtime for vector size 30.000.000: TODO
MCA: [0] Code Region - stopwatchedAccumulate
Iterations: 100
Instructions: 8200
Total Cycles: 4020
Total uOps: 9300
Dispatch Width: 6
uOps Per Cycle: 2.31
IPC: 2.04
Block RThroughput: 15.5
Resource pressure per iteration:

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
-	-	18.38	19.54	6.02	6.32	6.00	17.76	18.32	4.66