Design patterns for physics

A brief introduction to patterns and an example of pattern use in Fastjet

Henry Day-Hall

Overview

- What is a pattern?
  - Scales in software and Conway's law
  - Gamma patterns by intent
- Developing fastjet plugins; an example of good pattern use in physics.
  - Template method pattern for clean code reuse
  - Strategy pattern for altering behaviour at run time
- Fastjet plugins; what other patterns would have worked here?
  - Alternatives to template method pattern
  - Alternatives to strategy pattern
  - Strong v.s. weakly typed languges
- Conclusions.

What is a pattern?

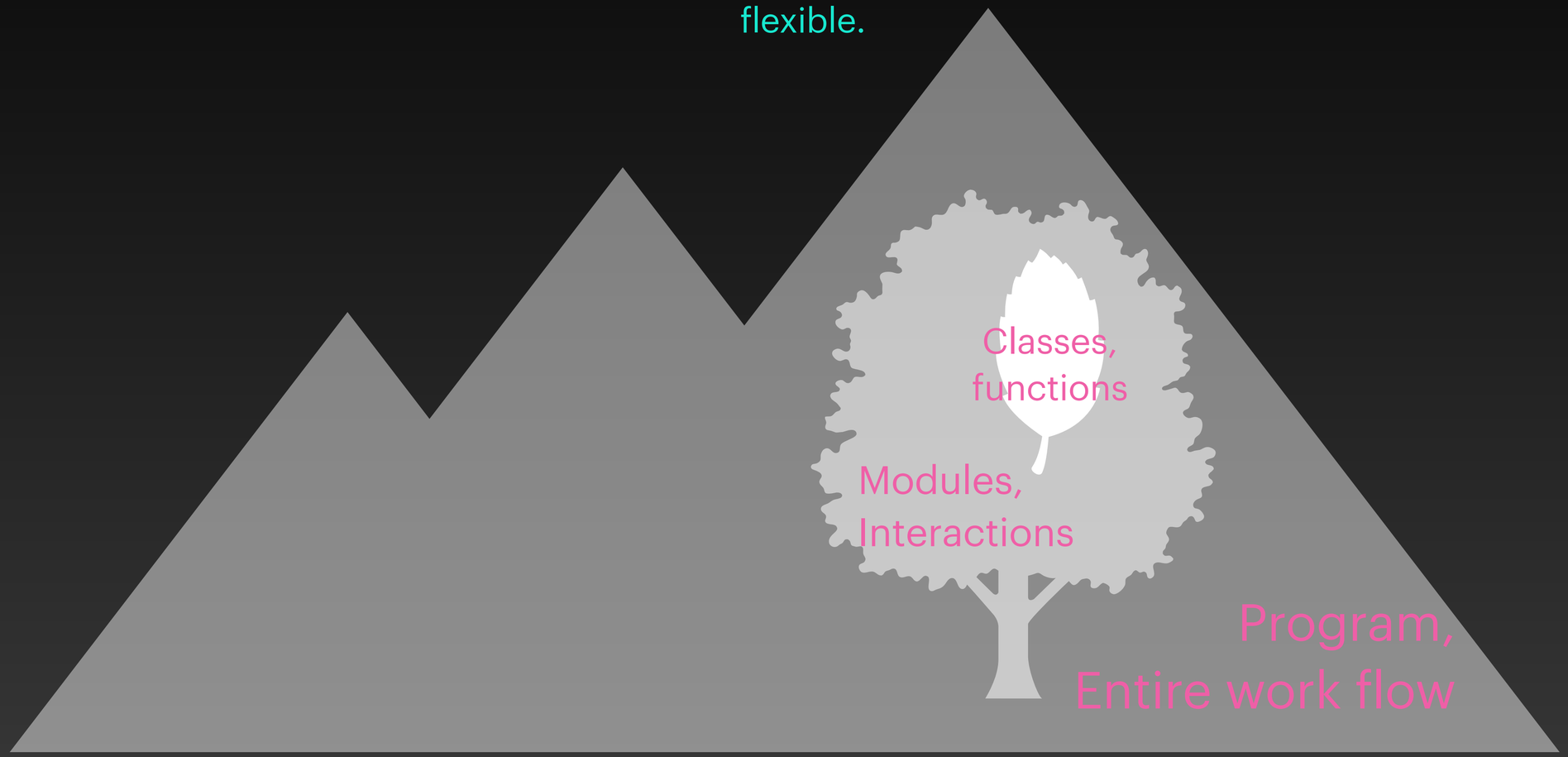
- Scales in software and Conway's law
- Gamma patterns by intent

- Developing fastjet plugins; an example of good pattern use in physics.
  - Template method pattern for clean code reuse
  - Strategy pattern for altering behaviour at run time
- Fastjet plugins; what other patterns would have worked here?
  - Alternatives to template method pattern
  - Alternatives to strategy pattern
  - Strong v.s. weakly typed languges
- Conclusions.

What is a pattern?

A pattern is a good idea for designing software

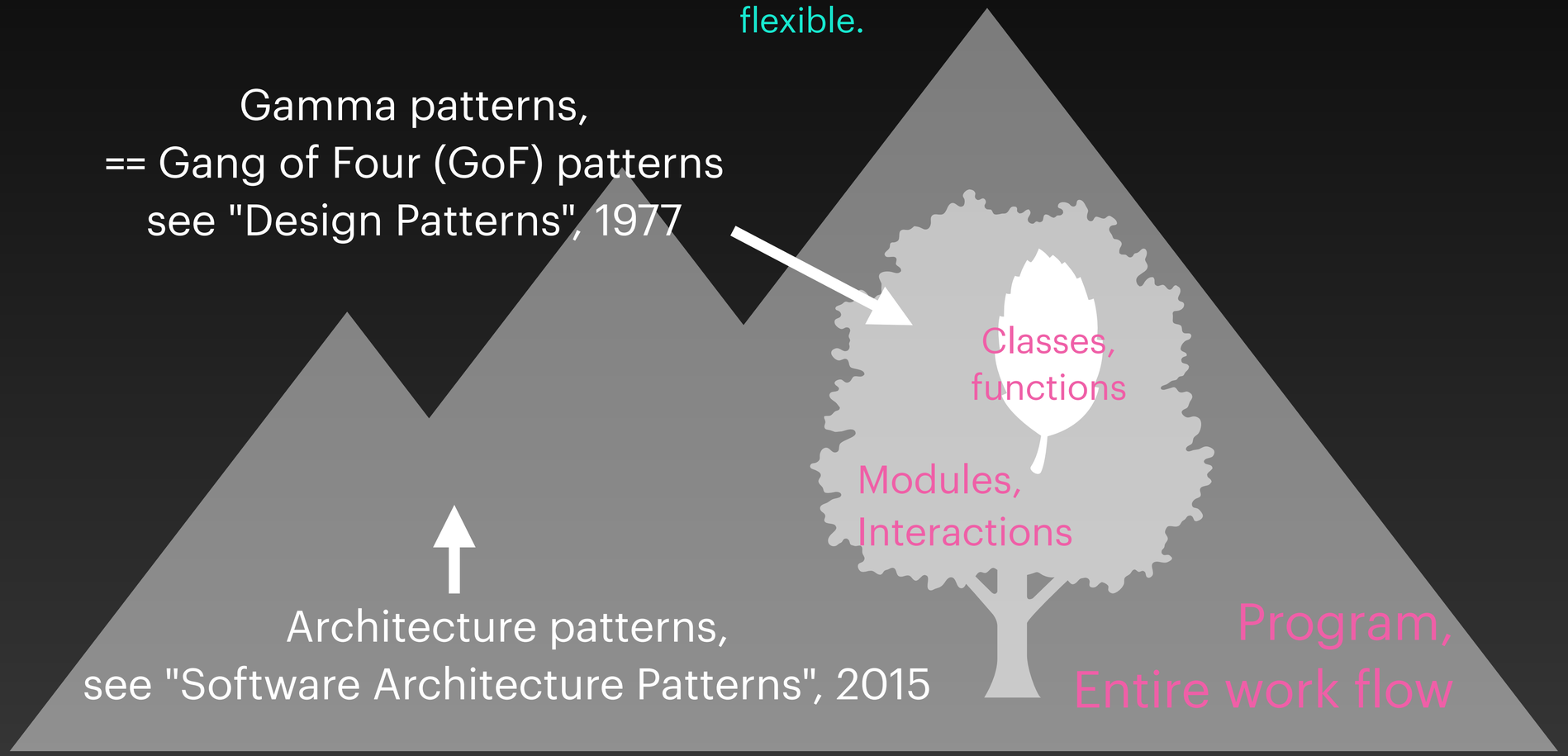
It should be easy to conceptualise, it should be easy to identify in code, and it should keep code flexible.



What is a pattern?

A pattern is a good idea for designing software

It should be easy to conceptualise, it should be easy to identify in code, and it should keep code flexible.



What is a pattern?

A pattern is a good idea for designing software

It should be easy to conceptualise, it should be easy to identify in code, and it should keep code flexible.

- Make variations on a theme without repetition.
- Communicate flexibly between varying interfaces.
- Communicate effectively in complex structures.
- Allow for changes at runtime, without excessive branching.
- Preempt problems with intentional limits.
- Memory management for objects.

What is a pattern?

A pattern is a good idea for designing software

It should be easy to conceptualise, it should be easy to identify in code, and it should keep code flexible.

- Make variations on a theme without repetition.
- Communicate flexibly between varying interfaces.
- Communicate effectively in complex structures.
- Allow for changes at runtime, without excessive branching.
- Preempt problems with intentional limits.
- Memory management for objects.

State
Template method
Builder
Decorator

What is a pattern?

A pattern is a good idea for designing software

It should be easy to conceptualise, it should be easy to identify in code, and it should keep code flexible.

- Make variations on a theme without repetition.
- Communicate flexibly between varying interfaces.
- Communicate effectively in complex structures.
- Allow for changes at runtime, without excessive branching.
- Preempt problems with intentional limits.
- Memory management for objects.

Adaptor
Visitor
Mediator
Command
Composite

What is a pattern?

A pattern is a good idea for designing software

It should be easy to conceptualise, it should be easy to identify in code, and it should keep code flexible.

- Make variations on a theme without repetition.
- Communicate flexibly between varying interfaces.
- Communicate effectively in complex structures.
- Allow for changes at runtime, without excessive branching.
- Preempt problems with intentional limits.
- Memory management for objects.

Bridge
Observer
Chain of responsibility
Command
Facade

What is a pattern?

A pattern is a good idea for designing software

It should be easy to conceptualise, it should be easy to identify in code, and it should keep code flexible.

- Make variations on a theme without repetition.
- Communicate flexibly between varying interfaces.
- Communicate effectively in complex structures.
- Allow for changes at runtime, without excessive branching.
- Preempt problems with intentional limits.
- Memory management for objects.

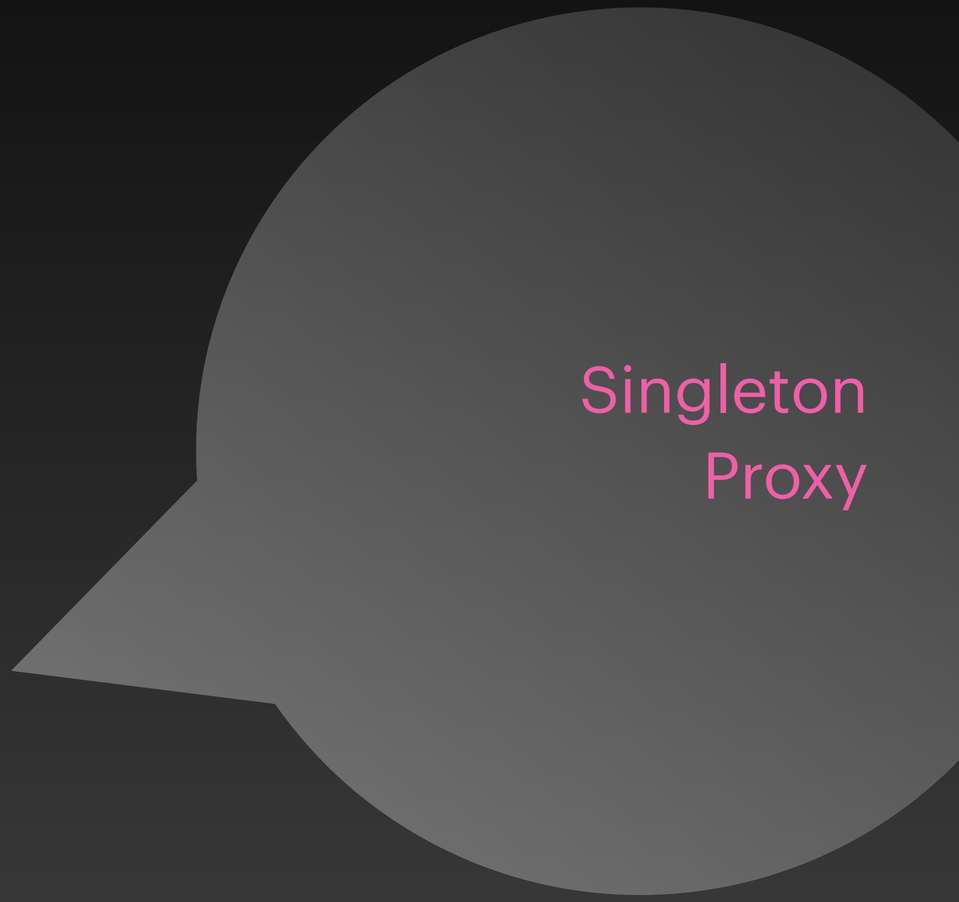
Abstract factory
Factory method
Prototype
Strategy

What is a pattern?

A pattern is a good idea for designing software

It should be easy to conceptualise, it should be easy to identify in code, and it should keep code flexible.

- Make variations on a theme without repetition.
- Communicate flexibly between varying interfaces.
- Communicate effectively in complex structures.
- Allow for changes at runtime, without excessive branching.
- Preempt problems with intentional limits.
- Memory management for objects.



What is a pattern?

A pattern is a good idea for designing software

It should be easy to conceptualise, it should be easy to identify in code, and it should keep code flexible.

- Make variations on a theme without repetition.
- Communicate flexibly between varying interfaces.
- Communicate effectively in complex structures.
- Allow for changes at runtime, without excessive branching.
- Preempt problems with intentional limits.
- Memory management for objects.

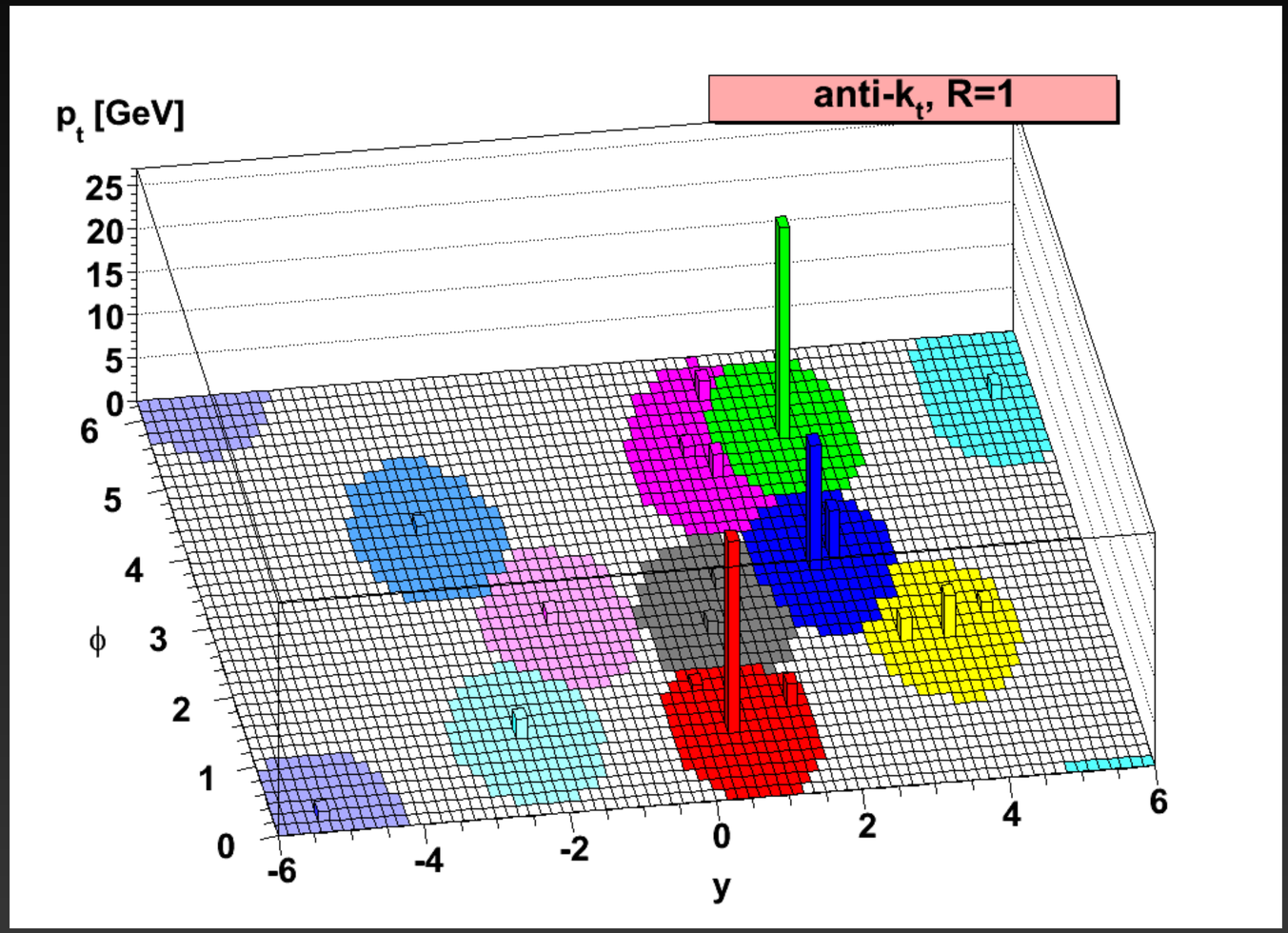
- What is a pattern?
  - Scales in software and Conway's law
  - Gamma patterns by intent

Developing fastjet plugins; an example of good pattern use in physics

  - Template method pattern for clean code reuse
  - Strategy pattern for altering behaviour at run time
- Fastjet plugins; what other patterns would have worked here?
  - Alternatives to template method pattern
  - Alternatives to strategy pattern
  - Strong v.s. weakly typed languges
- Conclusions.

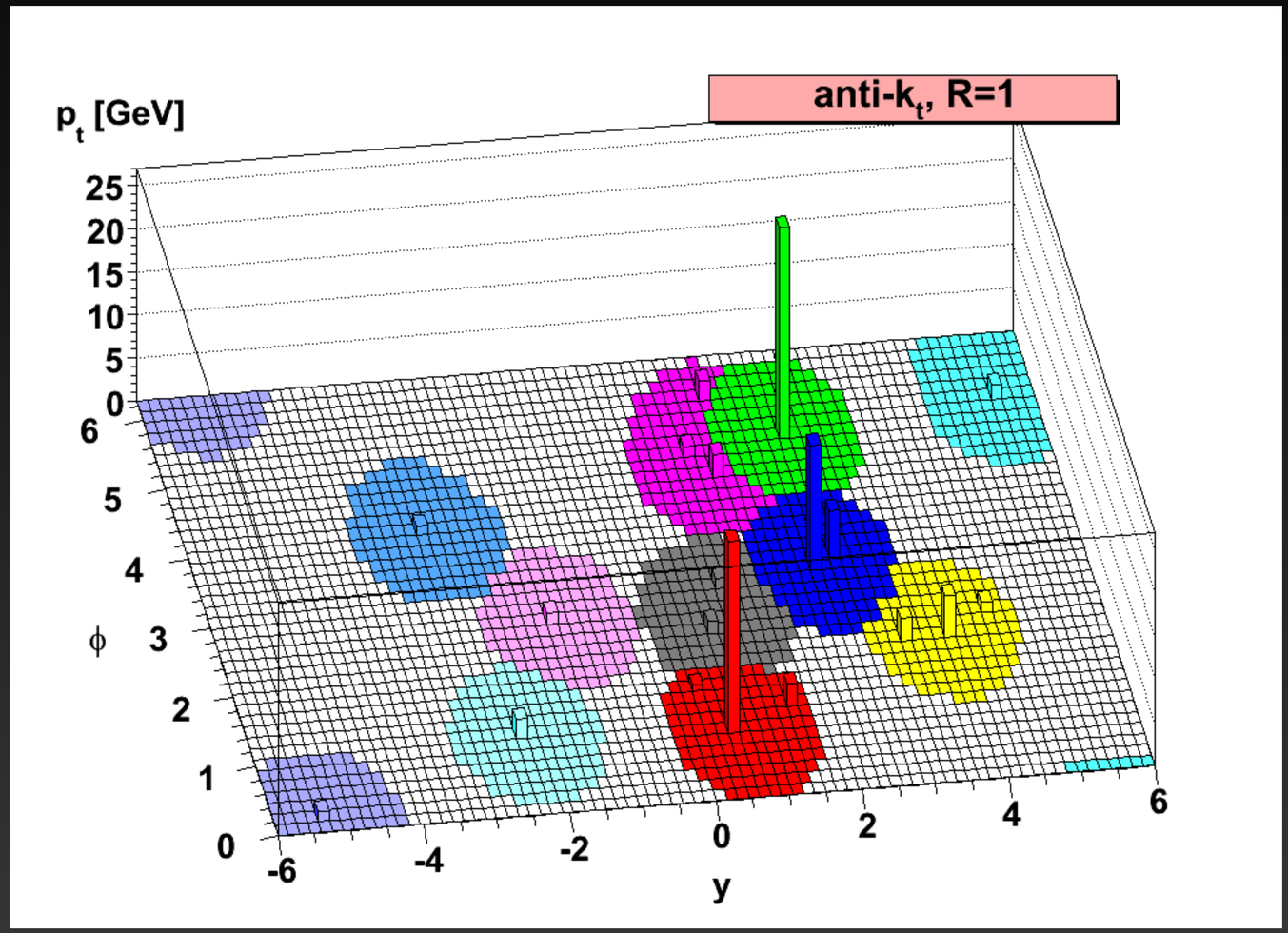
Fastjet - an algorithm for forming jets (possibly quite fast)

- Originally written for the gen-kt algorithms.
- Very well written, so it's the default framework for jet formation.
- Provides tools for defining new clustering algorithms, with minimal code repetition.
- Allows us to switch algorithm without recompiling.



Fastjet - an algorithm for forming jets (possibly quite fast)

- Originally written for the gen-kt algorithms.
- Very well written, so it's the default framework for jet formation.
- Provides tools for defining new clustering algorithms, with minimal code repetition.
- Allows us to switch algorithm without recompiling.



Defining new clustering algorithms

- What we want; "Make variations on a theme without repetition."
- Theme = clustering particles (with all their particle-like properties) into jets (with all their jet-like properties).
- Variation = deciding which particles go in which jets.

Defining new clustering algorithms

- What we want; "Make variations on a theme without repetition."
- Theme = clustering particles (with all their particle-like properties) into jets (with all their jet-like properties).
- Variation = deciding which particles go in which jets.

Defining new clustering algorithms

- What is used; Template method pattern
- *"Define the skeleton of an algorithm in an operation, deferring some steps to subclasses. Template Method lets subclasses redefine certain steps of an algorithm without changing the algorithm's structure."* - Gamma 1977

```
//======================================================================
/// @ingroup advanced_usage
/// \class Plugin
/// a class that allows a user to introduce their own "plugin" jet
/// finde
///
/// Note that all the plugins provided with FastJet are derived from
/// this class
class Plugin{
public:
  /// return a textual description of the jet-definition implemented
  /// in this plugin
  virtual std::string description() const = 0;

  /// given a ClusterSequence that has been filled up with initial
  /// particles, the following function should fill up the rest of the
  /// ClusterSequence, using the following member functions of
  /// ClusterSequence:
  ///   - plugin_do_ij_recombination(...)
  ///   - plugin_do_iB_recombination(...)
  virtual void run_clustering(ClusterSequence &) const = 0;

  virtual double R() const = 0;

  /// return true if there is specific support for the measurement
  /// of passive areas, in the sense that areas determined from all
  /// particles below the ghost separation scale will be a passive
  /// area. [If you don't understand this, ignore it!]
  virtual bool supports_ghosted_passive_areas() const {return false;}

  /// set the ghost separation scale for passive area determinations
  /// in future runs (strictly speaking that makes the routine
  /// a non const, so related internal info must be stored as a mutable)
  virtual void set_ghost_separation_scale(double scale) const;
  virtual double ghost_separation_scale() const {return 0.0;}

  /// if this returns false then a warning will be given
  /// whenever the user requests "exclusive" jets from the
  /// cluster sequence
  virtual bool exclusive_sequence_meaningful() const {return false;}

  /// returns true if the plugin implements an algorithm intended
  /// for use on a spherical geometry (e.g. e+e- algorithms, as
  /// opposed to most pp algorithms, which use a cylindrical,
  /// rapidity-phi geometry).
  virtual bool is_spherical() const {return false;}

  /// a destructor to be replaced if necessary in derived classes...
  virtual ~Plugin() {};
};
```

Defining new clustering algorithms

- Use existing tools for calculating rapidity and phi.
- Use existing framework for tracking available particles.
- Use existing framework for recording jet history tree.
- May use existing code to determine jet area.
- May use existing recombination scheme.

```
//----------------------------------------------------------------------
/// @ingroup advanced_usage
/// \class Plugin
/// a class that allows a user to introduce their own "plugin" jet
/// finde
///
/// Note that all the plugins provided with FastJet are derived from
/// this class
class Plugin{
public:
  /// return a textual description of the jet-definition implemented
  /// in this plugin
  virtual std::string description() const = 0;

  /// given a ClusterSequence that has been filled up with initial
  /// particles, the following function should fill up the rest of the
  /// ClusterSequence, using the following member functions of
  /// ClusterSequence:
  ///   - plugin_do_ij_recombination(...)
  ///   - plugin_do_iB_recombination(...)
  virtual void run_clustering(ClusterSequence &) const = 0;

  virtual double R() const = 0;

  /// return true if there is specific support for the measurement
  /// of passive areas, in the sense that areas determined from all
  /// particles below the ghost separation scale will be a passive
  /// area. [If you don't understand this, ignore it!]
  virtual bool supports_ghosted_passive_areas() const {return false;}

  /// set the ghost separation scale for passive area determinations
  /// in future runs (strictly speaking that makes the routine
  /// a non const, so related internal info must be stored as a mutable)
  virtual void set_ghost_separation_scale(double scale) const;
  virtual double ghost_separation_scale() const {return 0.0;}

  /// if this returns false then a warning will be given
  /// whenever the user requests "exclusive" jets from the
  /// cluster sequence
  virtual bool exclusive_sequence_meaningful() const {return false;}

  /// returns true if the plugin implements an algorithm intended
  /// for use on a spherical geometry (e.g. e+e- algorithms, as
  /// opposed to most pp algorithms, which use a cylindrical,
  /// rapidity-phi geometry).
  virtual bool is_spherical() const {return false;}

  /// a destructor to be replaced if necessary in derived classes...
  virtual ~Plugin() {};
};
```

Defining new clustering algorithms

- Use existing tools for calculating rapidity and phi.
- Use existing framework for tracking available particles.
- Use existing framework for recording jet history tree.
- May use existing code to determine jet area.
- May use existing recombination scheme.

```
//----------------------------------------------------------------------
/// @ingroup advanced_usage
/// \class Plugin
/// a class that allows a user to introduce their own "plugin" jet
/// finde
///
/// Note that all the plugins provided with FastJet are derived from
/// this class
class Plugin{
public:
  /// return a textual description of the jet-definition implemented
  /// in this plugin
  virtual std::string description() const = 0;

  /// given a ClusterSequence that has been filled up with initial
  /// particles, the following function should fill up the rest of the
  /// ClusterSequence, using the following member functions of
  /// ClusterSequence:
  ///   - plugin_do_ij_recombination(...)
  ///   - plugin_do_iB_recombination(...)
  virtual void run_clustering(ClusterSequence &) const = 0;

  virtual double R() const = 0;

  /// return true if there is specific support for the measurement
```

```
52  // This acts like any fastjet plugin since it implements run_clustering
53  class VariableRPlugin : public JetDefinition::Plugin {
54
55  public:
56    /// Type of clustering
57    ///
58    /// Since version 1.2.0 of VariableR, the clustering is treated as
59    /// a generalised-kt algorithm and the previous "ClusterType"
```

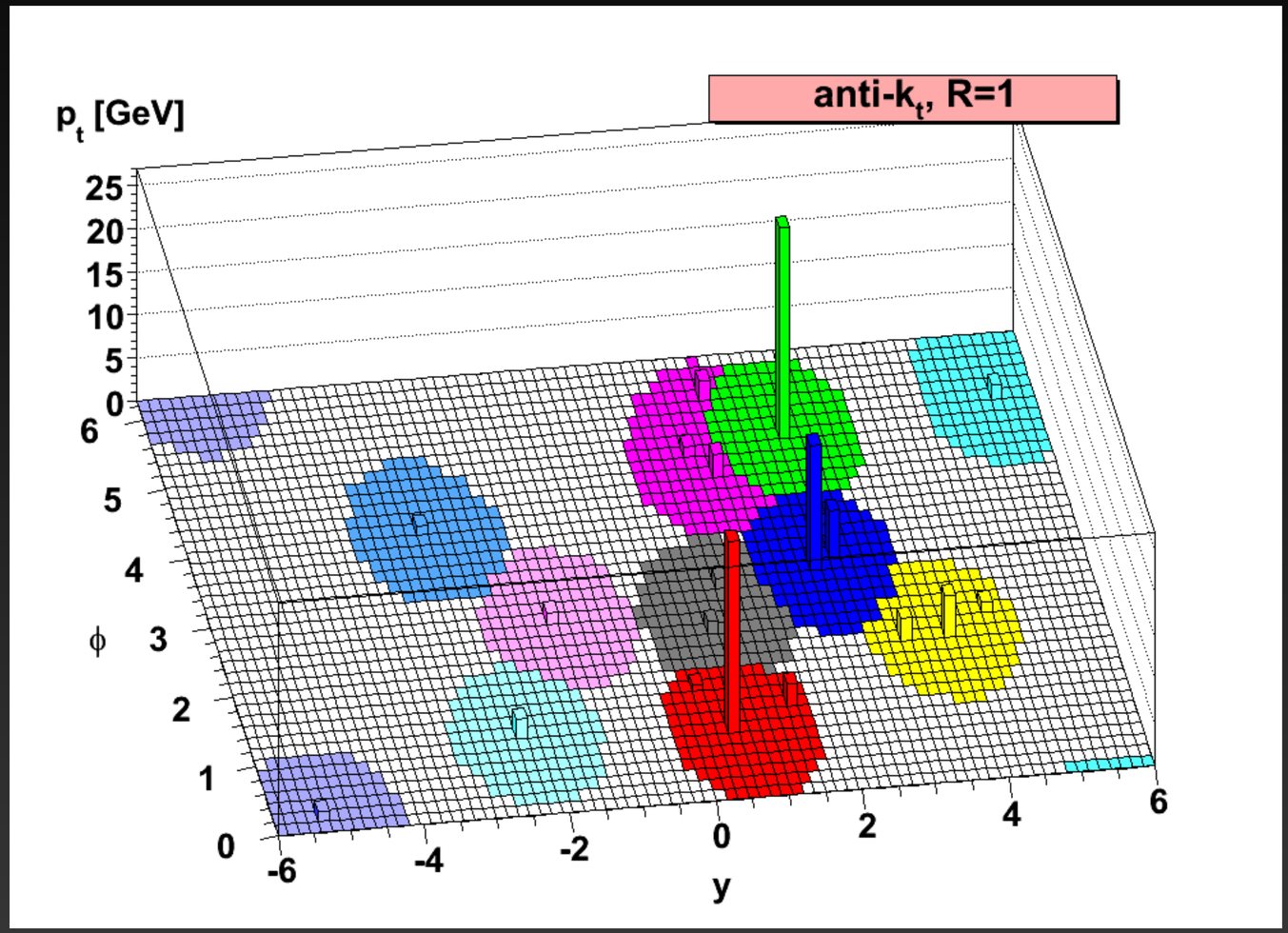
```
  virtual bool exclusive_sequence_meaningful() const {return false;}

  /// returns true if the plugin implements an algorithm intended
  /// for use on a spherical geometry (e.g. e+e- algorithms, as
  /// opposed to most pp algorithms, which use a cylindrical,
  /// rapidity-phi geometry).
  virtual bool is_spherical() const {return false;}

  /// a destructor to be replaced if necessary in derived classes...
  virtual ~Plugin() {};
};
```

Switch algorithms without recompiling

- What we want; "Allow for changes at runtime, without excessive branching."
- Do want to be able to specify algorithm when I launch madanalysis, without having to recompile anything.
- Don't want lots of `if` statements, because branching is error prone and slow.

Fastjet - an algorithm for forming jets (possibly quite fast)

- Originally written for the gen-kt algorithms.
- Very well written, so it's the default framework for jet formation.
- Provides tools for defining new clustering algorithms, with minimal code repetition.
- Allows us to switch algorithm without recompiling.



Switch algorithms without recompiling

- What is used; Strategy pattern
- *"Define a family of algorithms, encapsulate each one, and make them interchangeable. Strategy lets the algorithm vary independently from clients that use it."* - Gamma 1977

```
56
57  //------------------------------------------------------------------------
58  /// \class HOTVR
59  ///
60  class HOTVR : public JetDefinition::Plugin {
```

```
41  /// "Semi-classical approach to sequential recombination algorithms
42  /// for jet clustering", arXiv:1304.1025 (2013).
43  class ScJet : public JetDefinition::Plugin {
```

```
52    // This acts like any fastjet plugin since it implements run_clustering
53    class VariableRPlugin : public JetDefinition::Plugin {
54
55    public:
56      /// Type of clustering
57      ///
58      /// Since version 1.2.0 of VariableR, the clustering is treated as
59      /// a generalised-kt algorithm and the previous "ClusterType"
```

```
60  // defining parameters
61  double rho = 2000.0;
62  double min_r = 0.0;
63  double max_r = 2.0;
64  double ptmin = 5.0;
65
66  VariableRPlugin lvjet_pluginAKT(rho, min_r, max_r, VariableRPlugin::AKTLIKE);
67  fastjet::JetDefinition jet_defAKT(&lvjet_pluginAKT);
```

```
45 JetDefinition::JetDefinition(JetAlgorithm jet_algorithm_in,
46             double R_in,
47             RecombinationScheme recomb_scheme_in,
48             Strategy strategy_in,
49                                 int nparameters) :
50   _jet_algorithm(jet_algorithm_in), _Rparam(R_in), _strategy(strategy_in) {
51
```

Switch algorithms without recompiling

- The family of algorithms includes inbuilt gen-kt algorithms, and plugin algorithms defined by third parties.
- These are passed to `JetDefinition`.
- `JetDefinition` encapsulates the algorithm, providing a standard external interface.

```
56
57  //----------------------------------------------------------------------
58  /// \class HOTVR
59  ///
60  class HOTVR : public JetDefinition::Plugin {
```

```
41  /// "Semi-classical approach to sequential recombination algorithms
42  /// for jet clustering", arXiv:1304.1025 (2013).
43  class ScJet : public JetDefinition::Plugin {
```

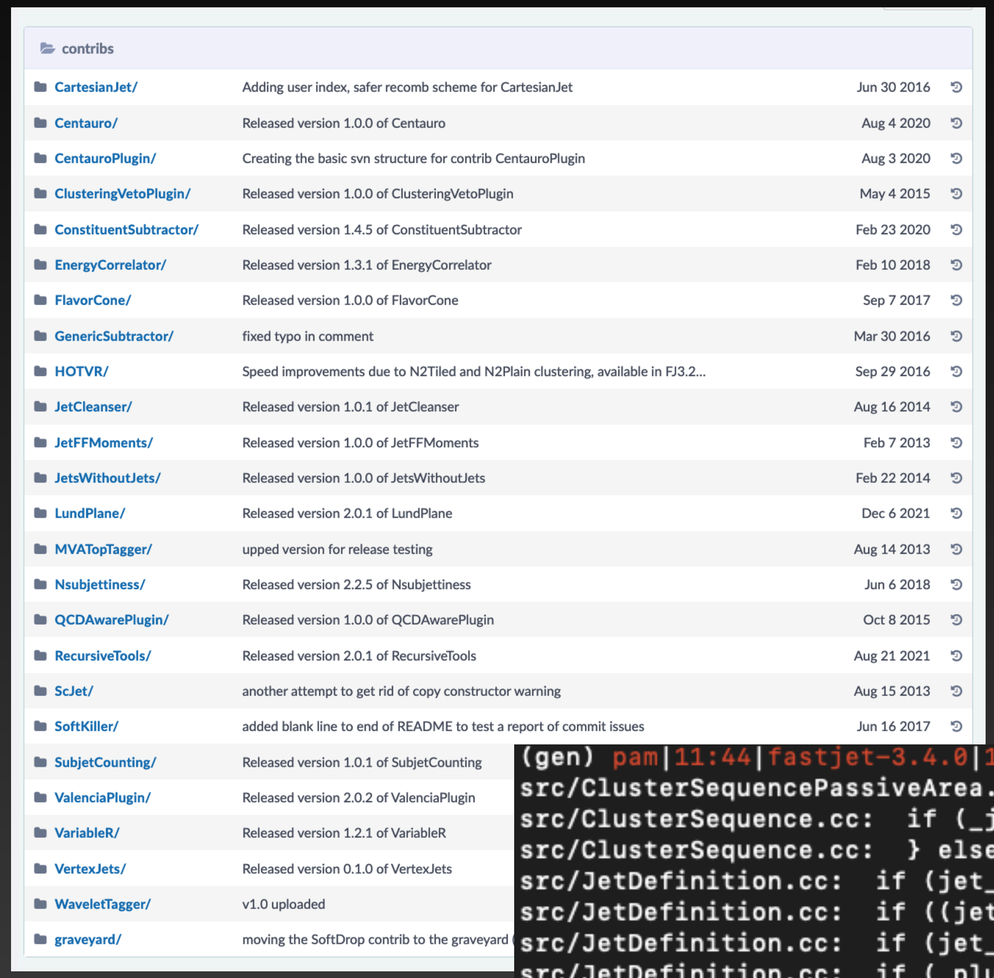
```
52    // This acts like any fastjet plugin since it implements run_clustering
53    class VariableRPlugin : public JetDefinition::Plugin {
54
55    public:
56      /// Type of clustering
57      ///
58      /// Since version 1.2.0 of VariableR, the clustering is treated as
59      /// a generalised-kt algorithm and the previous "ClusterType"
```

```
60  // defining parameters
61  double rho = 2000.0;
62  double min_r = 0.0;
63  double max_r = 2.0;
64  double ptmin = 5.0;
65
66  VariableRPlugin lvjet_pluginAKT(rho, min_r, max_r, VariableRPlugin::AKTLIKE);
67  fastjet::JetDefinition jet_defAKT(&lvjet_pluginAKT);
```

```
45 JetDefinition::JetDefinition(JetAlgorithm jet_algorithm_in,
46             double R_in,
47             RecombinationScheme recomb_scheme_in,
48             Strategy strategy_in,
49                                 int nparameters) :
50   _jet_algorithm(jet_algorithm_in), _Rparam(R_in), _strategy(strategy_in) {
51
```

Success?

Do the Template method and strategy patterns work well here?



contribs

CartesianJet/	Adding user index, safer recomb scheme for CartesianJet	Jun 30 2016
Centauro/	Released version 1.0.0 of Centauro	Aug 4 2020
CentauroPlugin/	Creating the basic svn structure for contrib CentauroPlugin	Aug 3 2020
ClusteringVetoPlugin/	Released version 1.0.0 of ClusteringVetoPlugin	May 4 2015
ConstituentSubtractor/	Released version 1.4.5 of ConstituentSubtractor	Feb 23 2020
EnergyCorrelator/	Released version 1.3.1 of EnergyCorrelator	Feb 10 2018
FlavorCone/	Released version 1.0.0 of FlavorCone	Sep 7 2017
GenericSubtractor/	fixed typo in comment	Mar 30 2016
HOTVR/	Speed improvements due to N2Tiled and N2Plain clustering, available in FJ3.2...	Sep 29 2016
JetCleanser/	Released version 1.0.1 of JetCleanser	Aug 16 2014
JetFFMoments/	Released version 1.0.0 of JetFFMoments	Feb 7 2013
JetsWithoutJets/	Released version 1.0.0 of JetsWithoutJets	Feb 22 2014
LundPlane/	Released version 2.0.1 of LundPlane	Dec 6 2021
MVATopTagger/	upped version for release testing	Aug 14 2013
Nsubjettiness/	Released version 2.2.5 of Nsubjettiness	Jun 6 2018
QCDAwarePlugin/	Released version 1.0.0 of QCDAwarePlugin	Oct 8 2015
RecursiveTools/	Released version 2.0.1 of RecursiveTools	Aug 21 2021
ScJet/	another attempt to get rid of copy constructor warning	Aug 15 2013
SoftKiller/	added blank line to end of README to test a report of commit issues	Jun 16 2017
SubjetCounting/	Released version 1.0.1 of SubjetCounting	
ValenciaPlugin/	Released version 2.0.2 of ValenciaPlugin	
VariableR/	Released version 1.2.1 of VariableR	
VertexJets/	Released version 0.1.0 of VertexJets	
WaveletTagger/	v1.0 uploaded	
graveyard/	moving the SoftDrop contrib to the graveyard	

- Lots of varied plugins have been written; the Template must be easy to understand.
- There are minimal branches in the code relating to plugins; the Strategy is encapsulating the variation.

```
(gen) pam|11:44|fastjet-3.4.0|1013$ grep --include \*.cc "if.*plugin" src -R -I
src/ClusterSequencePassiveArea.cc:  } else if (jet_def_in.jet_algorithm() == plugin_algorithm &&
src/ClusterSequence.cc:  if (_jet_algorithm == plugin_algorithm) {
src/ClusterSequence.cc:  } else if (_jet_algorithm == plugin_algorithm
src/JetDefinition.cc:  if (jet_algorithm() == plugin_algorithm) {
src/JetDefinition.cc:  if ((jet_algorithm() == plugin_algorithm) || (jet_algorithm() == undefined_jet_algorithm)){
src/JetDefinition.cc:  if (jet_algorithm() == plugin_algorithm) {
src/JetDefinition.cc:  if (_plugin == 0){
(gen) pam|11:45|fastjet-3.4.0|1014$
```

- What is a pattern?
  - Scales in software and Conway's law
  - Gamma patterns by intent
- Developing fastjet plugins; an example of good pattern use in physics.
  - Template method pattern for clean code reuse
  - Strategy pattern for altering behaviour at run time

Fastjet plugins; what other patterns would have worked here?

  - Alternatives to template method pattern
  - Alternatives to strategy pattern
  - Strong v.s. weakly typed languges
- Conclusions.

Alternatives to Template Method pattern

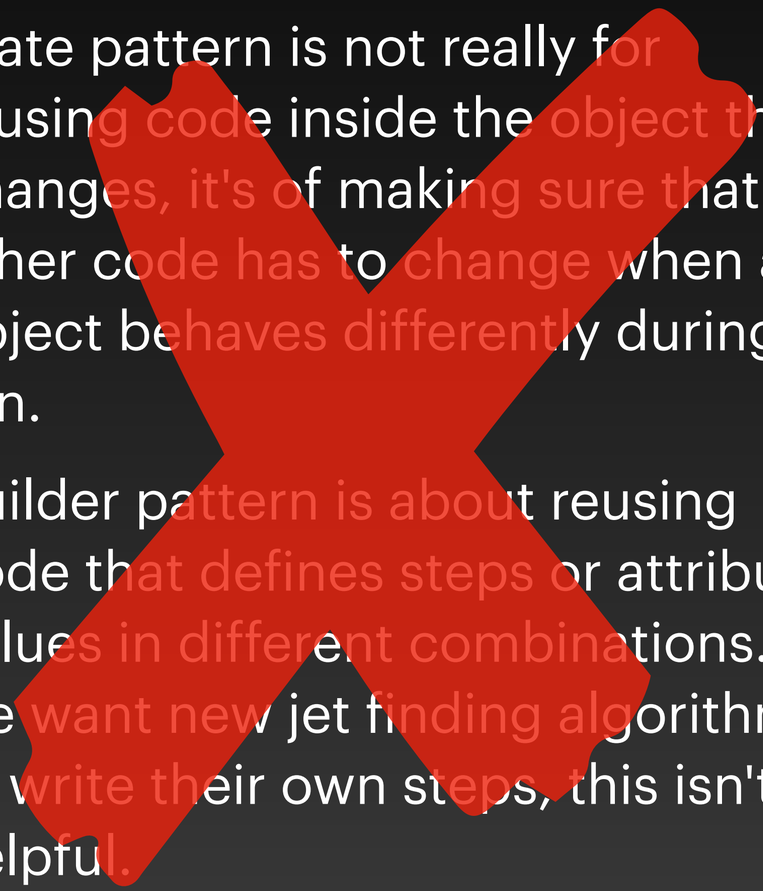
"Make variations on a theme without repetition."

- State pattern is not really for reusing code inside the object that changes, it's of making sure that no other code has to change when an object behaves differently during a run.
- Builder pattern is about reusing code that defines steps or attribute values in different combinations. As we want new jet finding algorithms to write their own steps, this isn't so helpful.

Alternatives to Template Method pattern

"Make variations on a theme without repetition."

- State pattern is not really for reusing code inside the object that changes, it's of making sure that no other code has to change when an object behaves differently during a run.
- Builder pattern is about reusing code that defines steps or attribute values in different combinations. As we want new jet finding algorithms to write their own steps, this isn't so helpful.



- Decorator pattern is for augmenting or overriding the behaviour of an object. It is a separate object, which carries a reference to the object it is decorating. Multiple decorators can sit in layers inside each other. Decorators can be applied at run time.

Alternatives to Strategy pattern

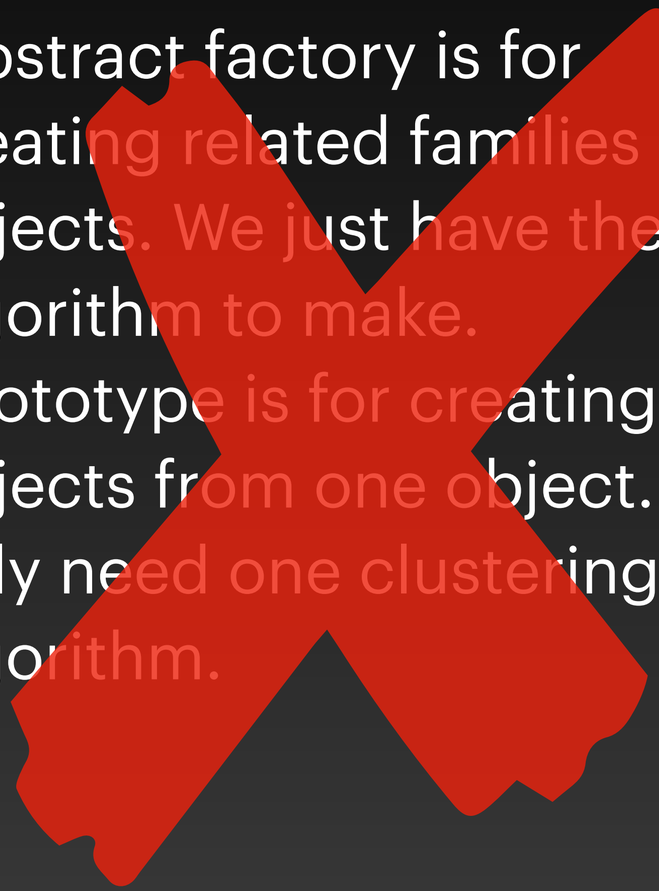
"Allow for changes at runtime, without excessive branching."

- Abstract factory is for creating related families of objects. We just have the one algorithm to make.
- Prototype is for creating many objects from one object. We only need one clustering algorithm.

Alternatives to Strategy pattern

"Allow for changes at runtime, without excessive branching."

- Abstract factory is for creating related families of objects. We just have the one algorithm to make.
- Prototype is for creating many objects from one object. We only need one clustering algorithm.



- Arguably, with duck typing, or a cast to a common base class, we could have had an object version of a strategy pattern. In a nominally (strongly) typed language, that would have meant giving Plugin and the default algorithms a common base class.
- Provided the language permits inferred types, we could also use a Factory method.

Alternatives to Strategy pattern

"Allow for changes at runtime, without excessive branching."

Strong v.s Weakly typed

Static v.s. Dynamic

- When does type checking happen?
- Dynamically typed = you can have a type error at run time.
- Statically typed = classes are not objects at run time.

Manifest v.s. Inferred

- Do you have to state the type of each variable?
- Even in languages that are mostly strongly typed, exceptions can be made, see `auto` in c++.

Nominal v.s. Structural

- What determines if two objects are compatible?
- Structural type conversions can be creative.

Alternatives to Strategy pattern

"Allow for changes at runtime, without excessive branching."

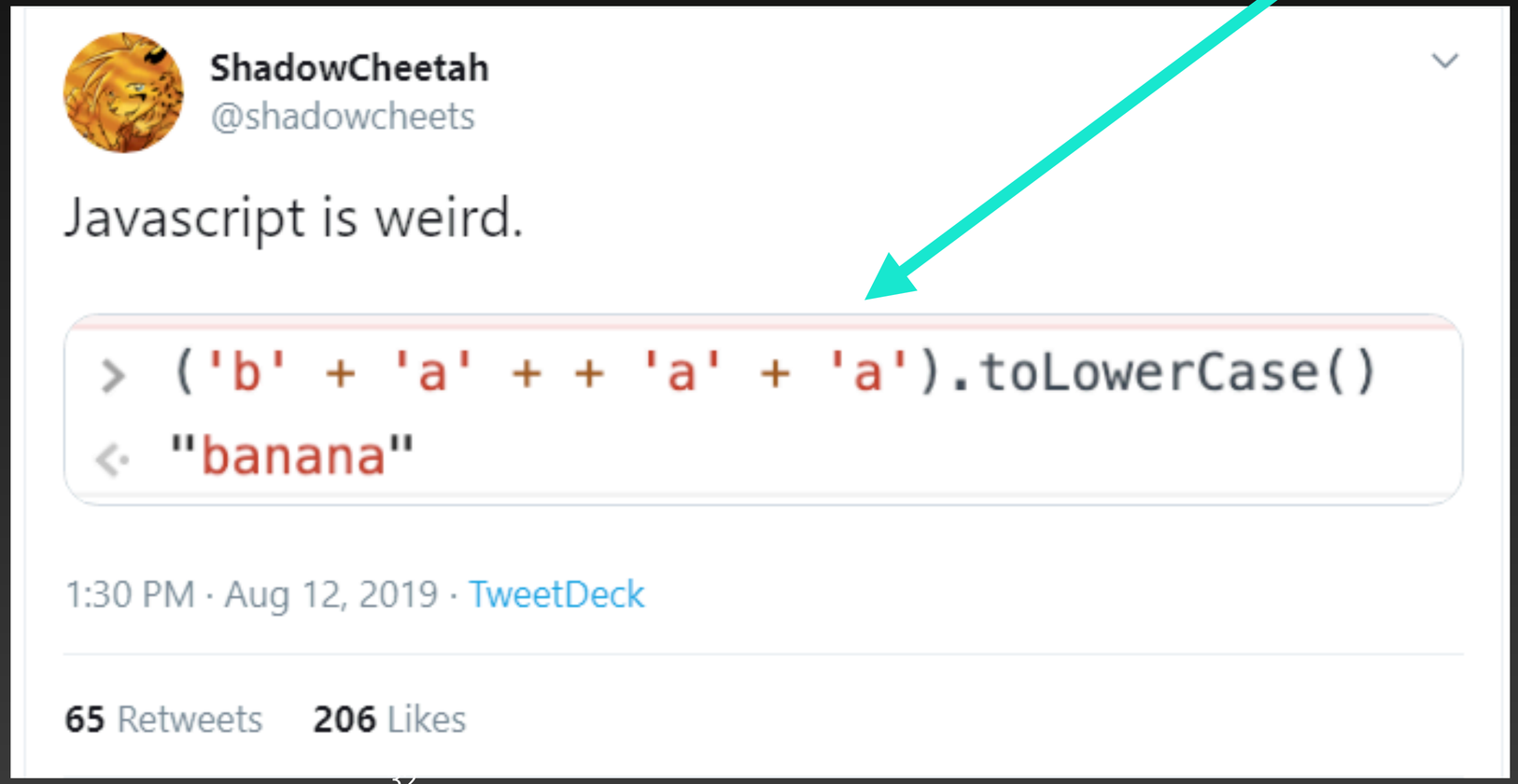
Strong v.s Weakly typed

Static v.s. Dynamic

Manifest v.s. Inferred

Nominal v.s. Structural

- When does type checking happen?
- Dynamically typed = you can have a type error at run time.
- Statically typed = classes are not objects at run time.



Alternatives to Strategy pattern

"Allow for changes at runtime, without excessive branching."

Strong v.s Weakly typed

Static v.s. Dynamic

Manifest v.s. Inferred

Nominal v.s. Structural

- When does type checking happen?
- Dynamically typed = you can have a type error at run time.
- Statically typed = classes are not objects at run time.

Also see

https://www.destroyallsoftware.com/talks/wat

Conclusions

- Design patterns are abstract methods for achieving common aims in code.
- They offer the benefit of existing experience.
- Patterns are a common language that can help you understand other code and write more understandable code.
- Discussing the pros and cons of different design choices is easier when we have labels for the options.

- "Design Patterns: Elements of Reusable Object-Oriented Software" by E Gamma, R Helm, J Vlissides, R Johnson. (http://www.javier8a.com/itc/bd1/articulo.pdf)
- "Software Architecture Patterns" - M Richards. (https://get.oreilly.com/rs/107-FMS-070/images/Software-Architecture-Patterns.pdf)