Tutorial on
The Open Graph Drawing Framework (OGDF)

Carsten Gutwenger

TU Dortmund

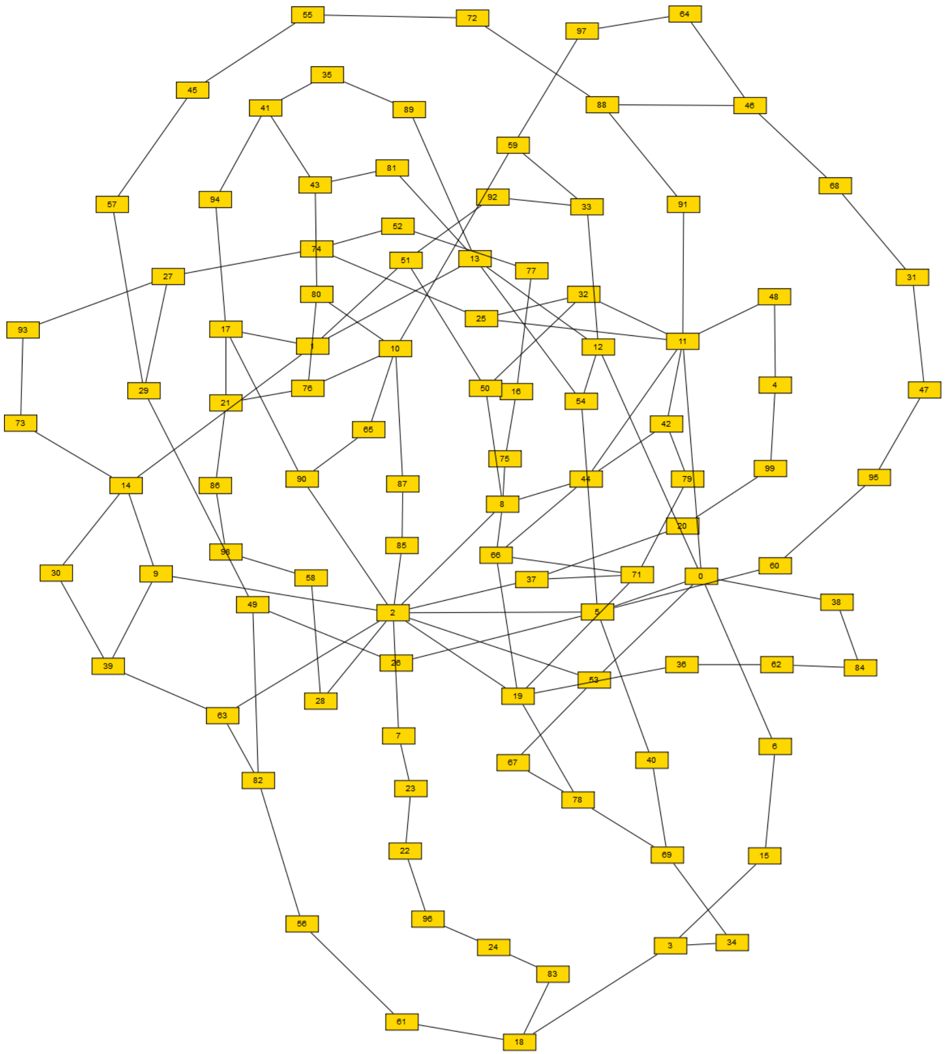
Minisymposium on Publicly Available Geometric / Topological Software (GTS)

June 17th, 2012 • Chapel Hill, NC, USA

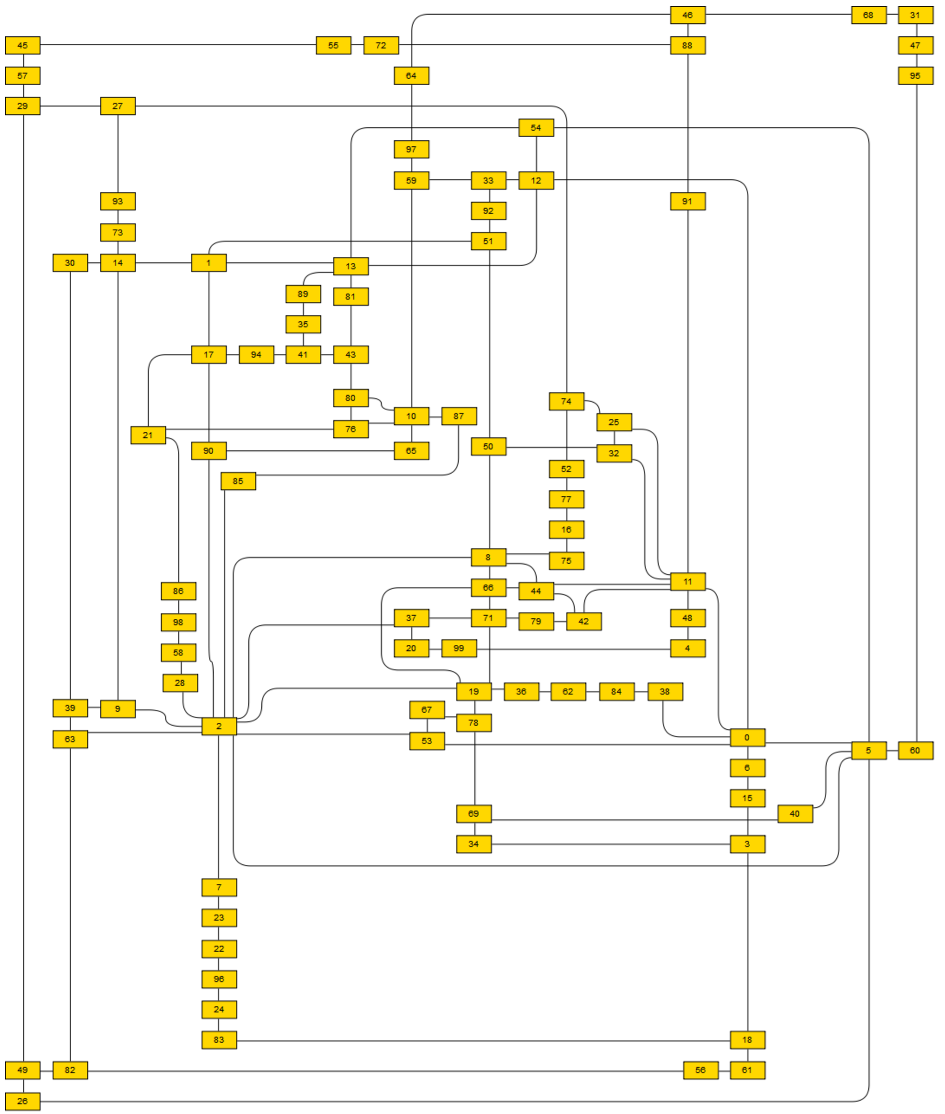
technische universität dortmund

department of computer science

That’s a mess!



Much better!



Overview

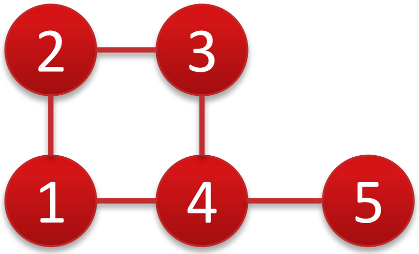
- Graph Drawing
- The OGDF
- Modularization and algorithm frameworks
- Graph Drawing algorithms in OGDF
  - planar layout algorithms and planarization
  - hierarchical layout
  - energy-based layout
  - cluster graph layout
- Tutorial
  - Sugiyama Layout
  - Planarization Layout
  - Drawing large graphs

Graph Drawing

- Graph Drawing considers a simple problem:
  - **Given** a graph G=(V,E)
  - **Create** a *drawing* of G which maps
    - vertices to points (or rectangles,…) and
    - edges to curves (straight-lines, polylines,…)

  such that edges connect their endpoints.

V = { 1, 2, 3, 4, 5}
E = { (1,2), (2,3), (3,4), (4,1), (4,5)

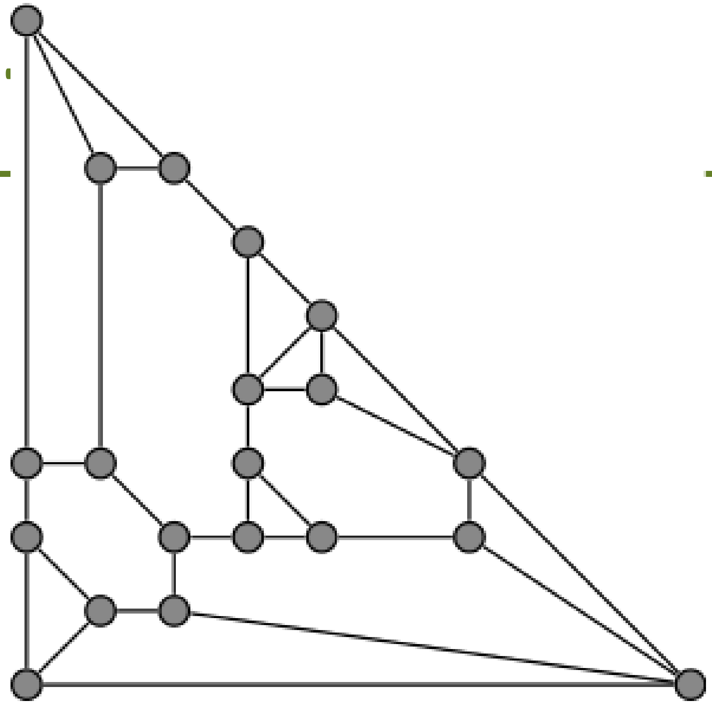


Aesthetic Criteria

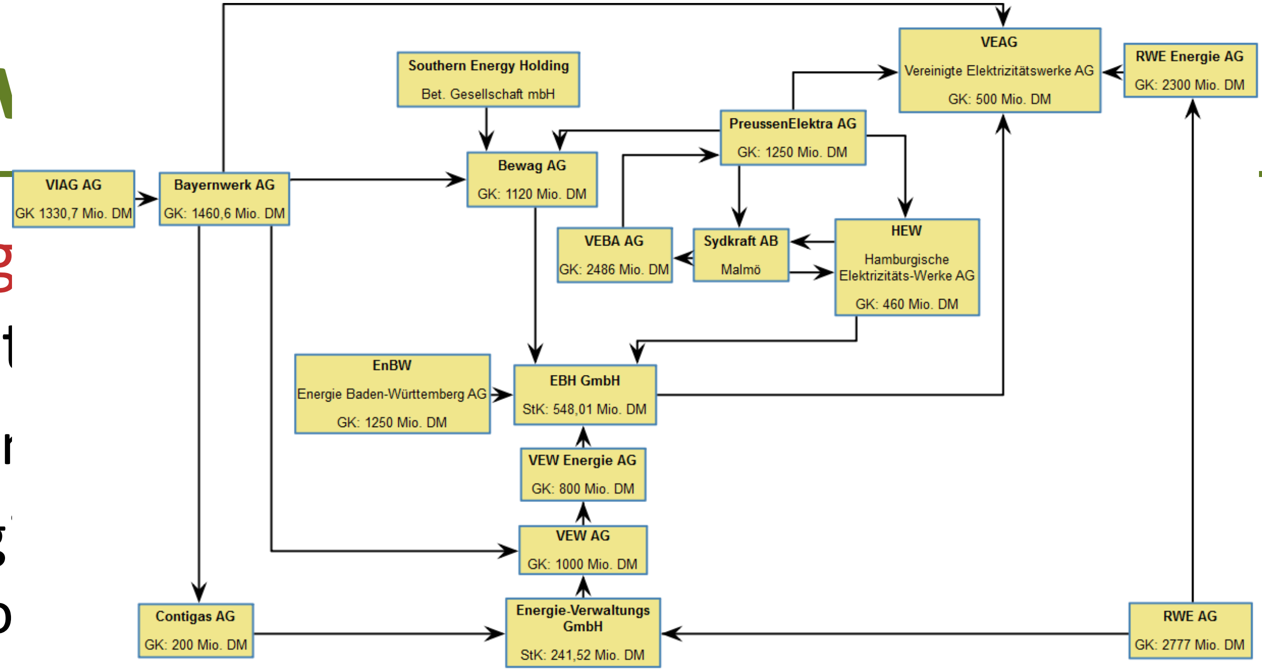
- What is a "good" drawing?
  - Should be *clear*, *nice*, display the *graph structure*,…
- Widely accepted aesthetic criteria
  - few crossings
  - few bends (e.g. polyline drawings)
  - small area (e.g. grid drawings)
  - short edges (total / maximum edge length)
  - uniform edge lengths
  - large angles between adjacent edges
  - good aspect ratio (e.g. given aspect ratio of screen or paper)
  - …

Drawing Conven

- Additional drawing conventions the layout must satisfy
  - straight-line drawings
  - planar drawings (given a planar graph, no edge crossings allowed)



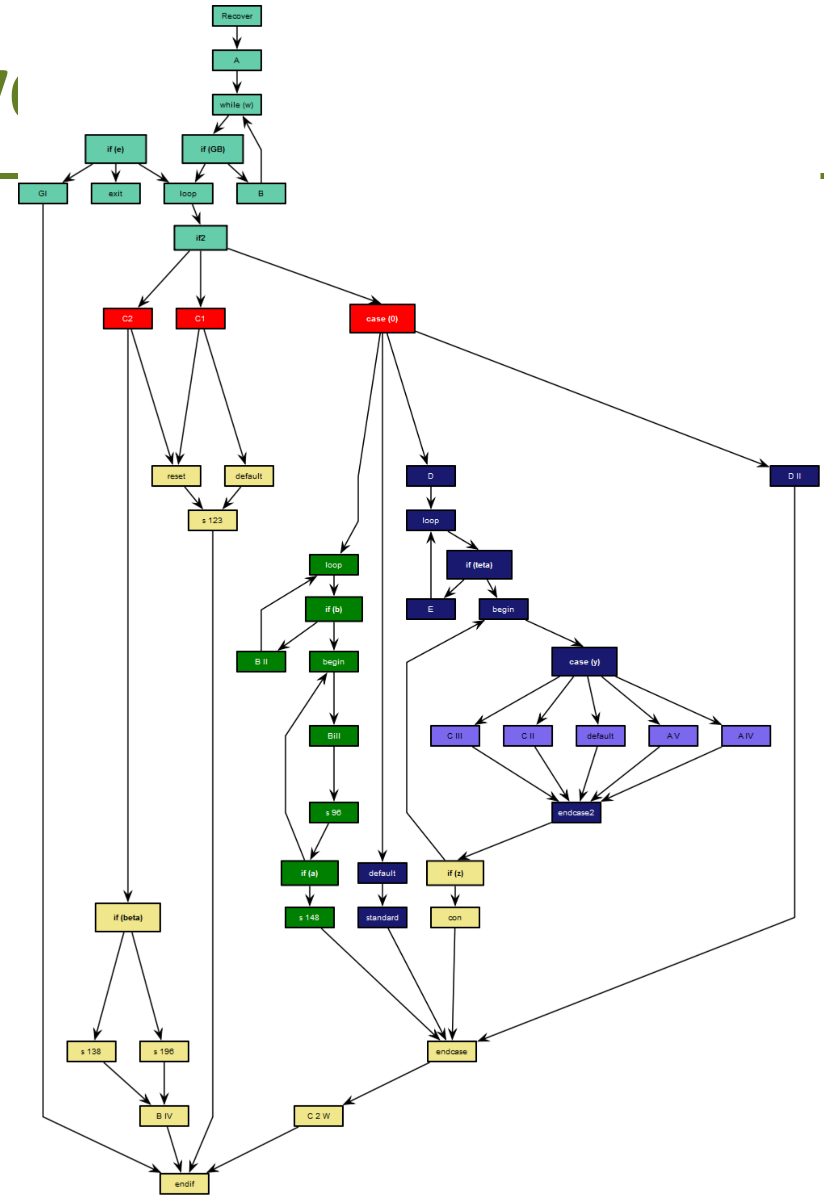
Dra



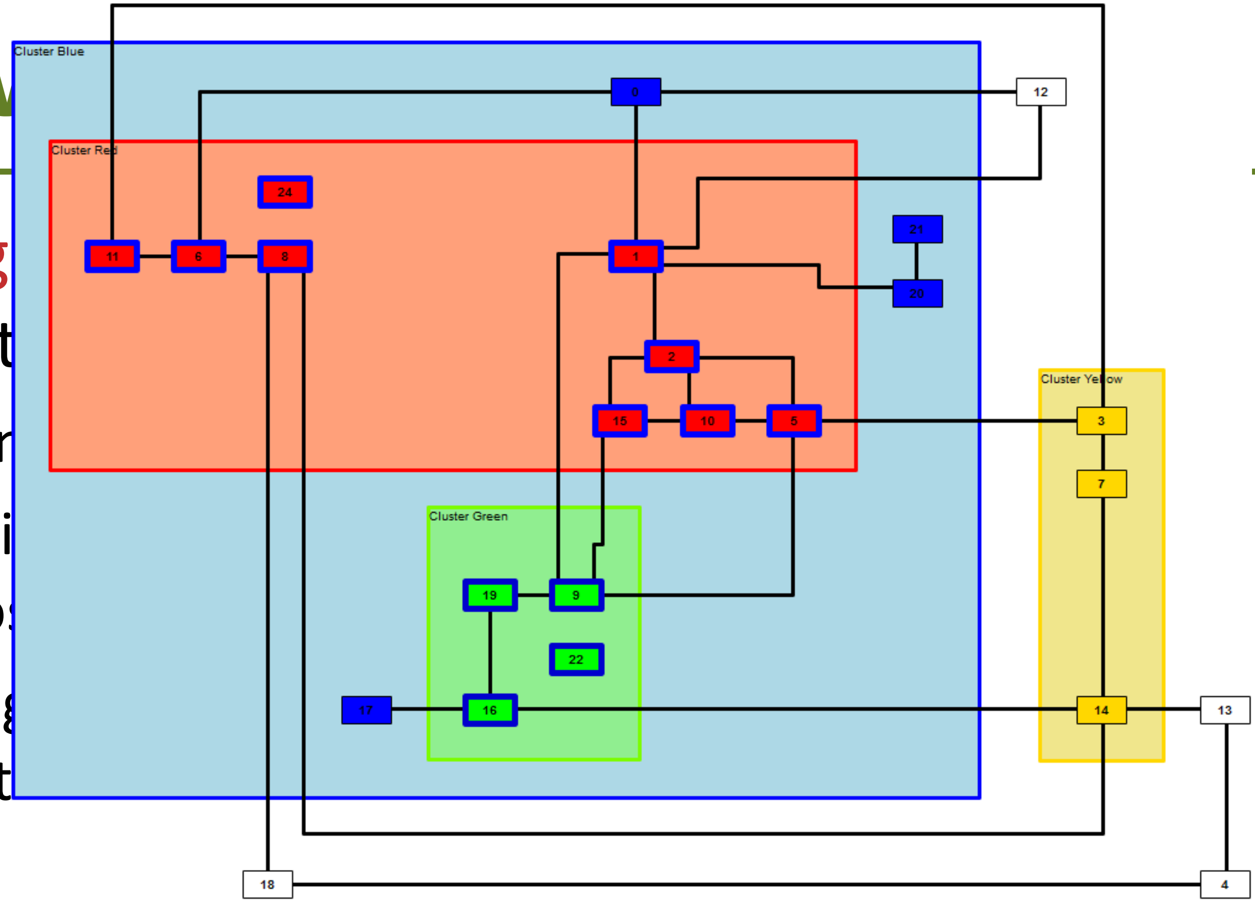
- Additional drawing
  the layout must sat
  - straight-line drawi
  - planar drawings (g
    graph, no edge cro
  - orthogonal drawings (only horizontal and vertical edge segments)

Drawing Conv

- Additional drawing conventions the layout must satisfy
  - straight-line drawings
  - planar drawings (given a planar graph, no edge crossings allowed)
  - orthogonal drawings (only horizontal and vertical edge segments)
  - hierarchical drawings (given a DAG, all edges must point upwards)



Draw



- Additional drawing
  the layout must sat
  - straight-line drawi
  - planar drawings (gi
    graph, no edge cro
  - orthogonal drawing
    horizontal and vert
    segments)
  - hierarchical drawings (given a DAG, all edges must point upwards)
  - clustered drawings (given a (hierarchical) cluster structure on the vertices)

The OGDF

- C++-library of data structures and algorithms
- Contains
  - various graph drawing algorithms,
  - but also many other (complex) data structures and algorithms (e.g. planarity testing and graph decomposition)
- Open Source (GPL v2 and v3)
- History
  - successor of AGD (developed since 1996)
  - development started in 1999 (as an internal project)
  - Open Source since 2007

http://www.ogdf.net

The OGDF

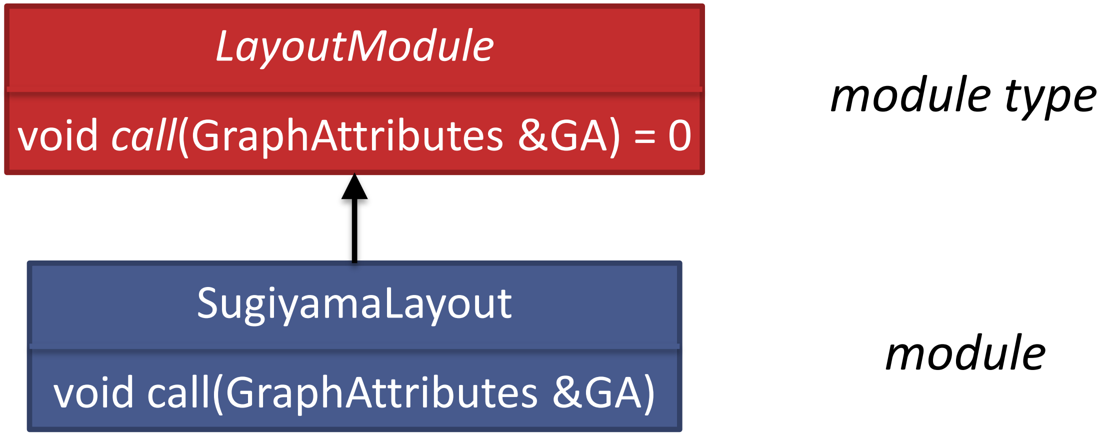
- Main contributing parties
  - TU Dortmund (Petra Mutzel, Carsten Gutwenger)
  - University of Jena (Markus Chimani)
  - University of Cologne (Michael Jünger)
  - University of Sydney (Karsten Klein)
  - oreas GmbH
- Used in current research and teaching

Major Design Concepts

- Provide a wide range of layout algorithms
- Allow to reuse and replace certain algorithm phases
  $\rightarrow$ Algorithm frameworks
- Provide sophisticated data structures commonly used in graph drawing algorithms
  $\rightarrow$ Makes it easier to implement new algorithms
- Self-contained code (except for some LP-/ILP-based algorithms)
- Portable C++-Code (Windows/Linux/MacOS)
  $\rightarrow$ no GUI or graphical display

Modularization

- Algorithms (e.g. layout algorithms)
  - are derived from a common base class (e.g. `LayoutModule`) specifying their interface.
  - such algorithms are called modules
  - the base class is the type of the module



Module Options

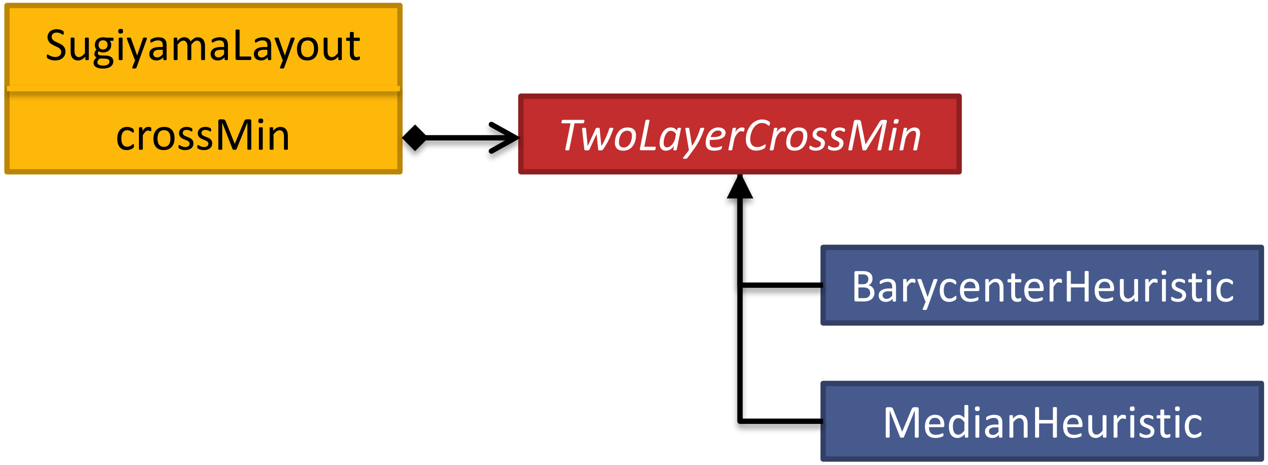
- Algorithms are represented as objects
  $\rightarrow$ can be exchanged (at runtime) by another object of the same module type
- Algorithm Frameworks
  - some sub-procedures of the algorithm are realized as module options
  - user can set this option to another module already defined in OGDF
  - or: define and implement a new module
  - or: reuse these sub-procedures in new algorithms

Example for Module Options

```
class SugiyamaLayout : public LayoutModule {
  ModuleOption<TwoLayerCrossMin> m_crossMin;
public:
  SugiyamaLayout() {
    m_crossMin.set(new BarycenterHeuristic);
  }
  void setCrossMin(TwoLayerCrossMin *pCrossMin)
    { m_crossMin.set(pCrossMin); }
};
```

Main program:

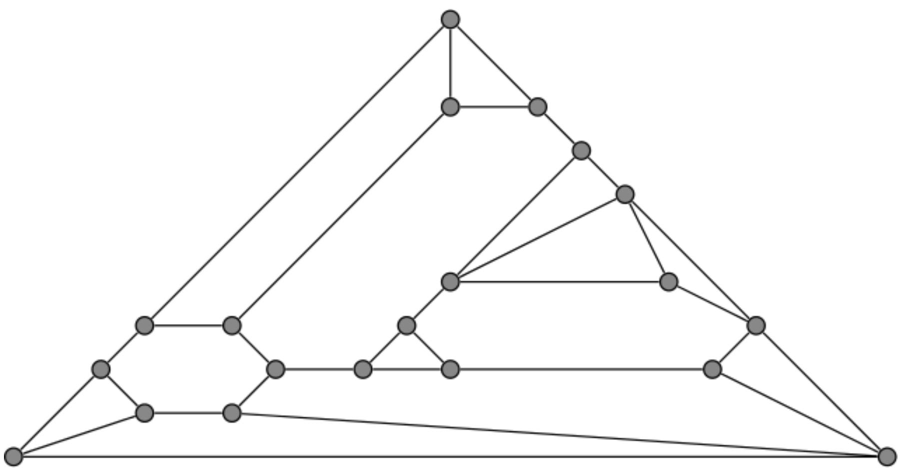
```
SugiyamaLayout sugi;
sugi.setCrossMin(new MedianHeuristic);
```

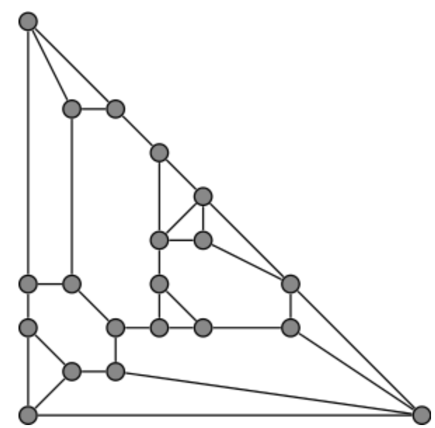


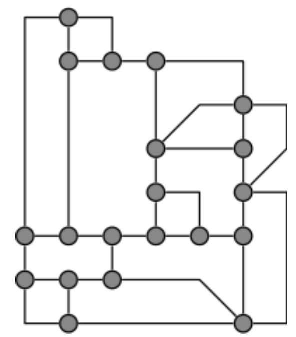
Graph Drawing Algorithms in OGDF

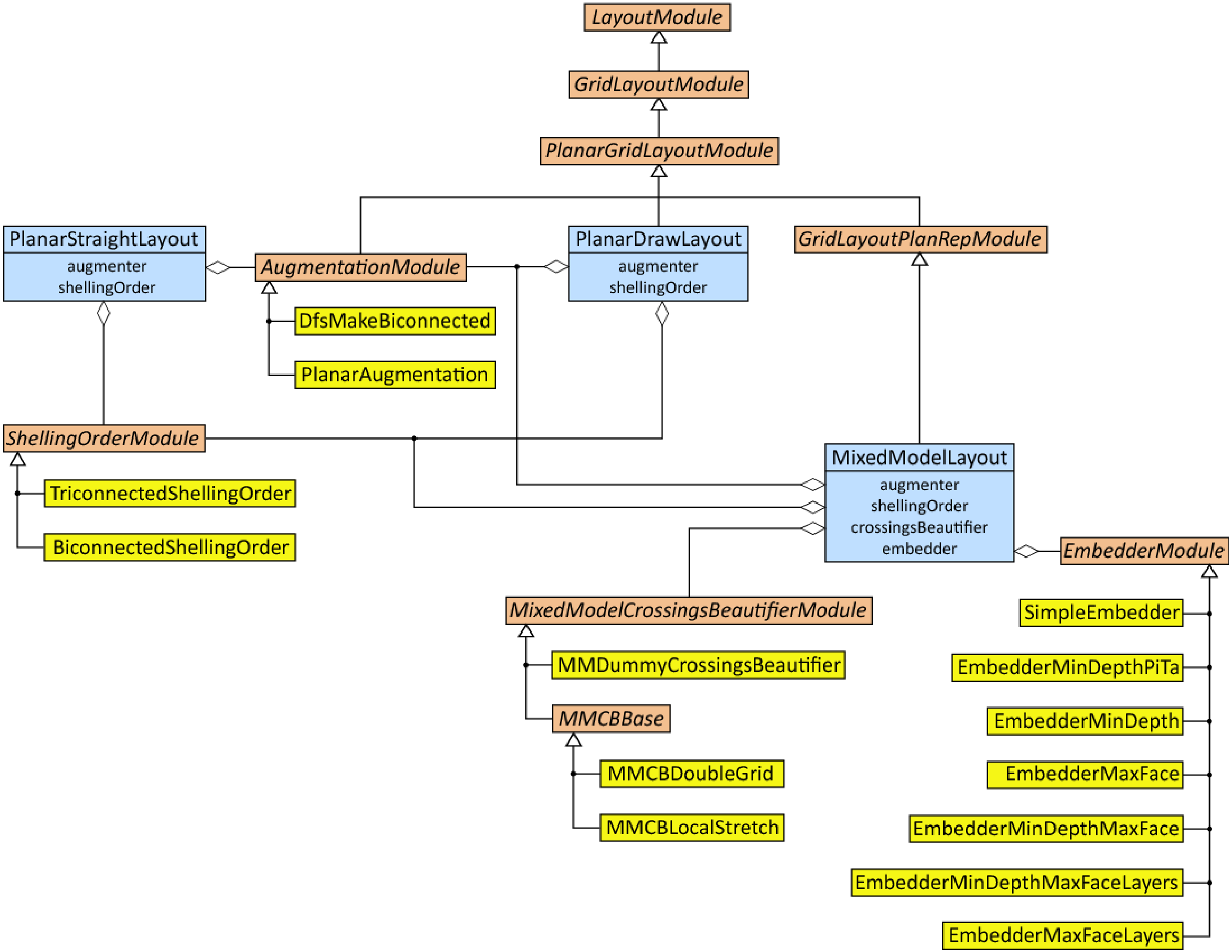
Planar Drawing Algorithms

- Planarity testing and planar embedding
- Straight-line
  - PlanarStraightLayout
  - PlanarDrawLayout
- Polyline
  - MixedModelLayout
  - Orthogonal layout (embeded in PlanarizationLayout)



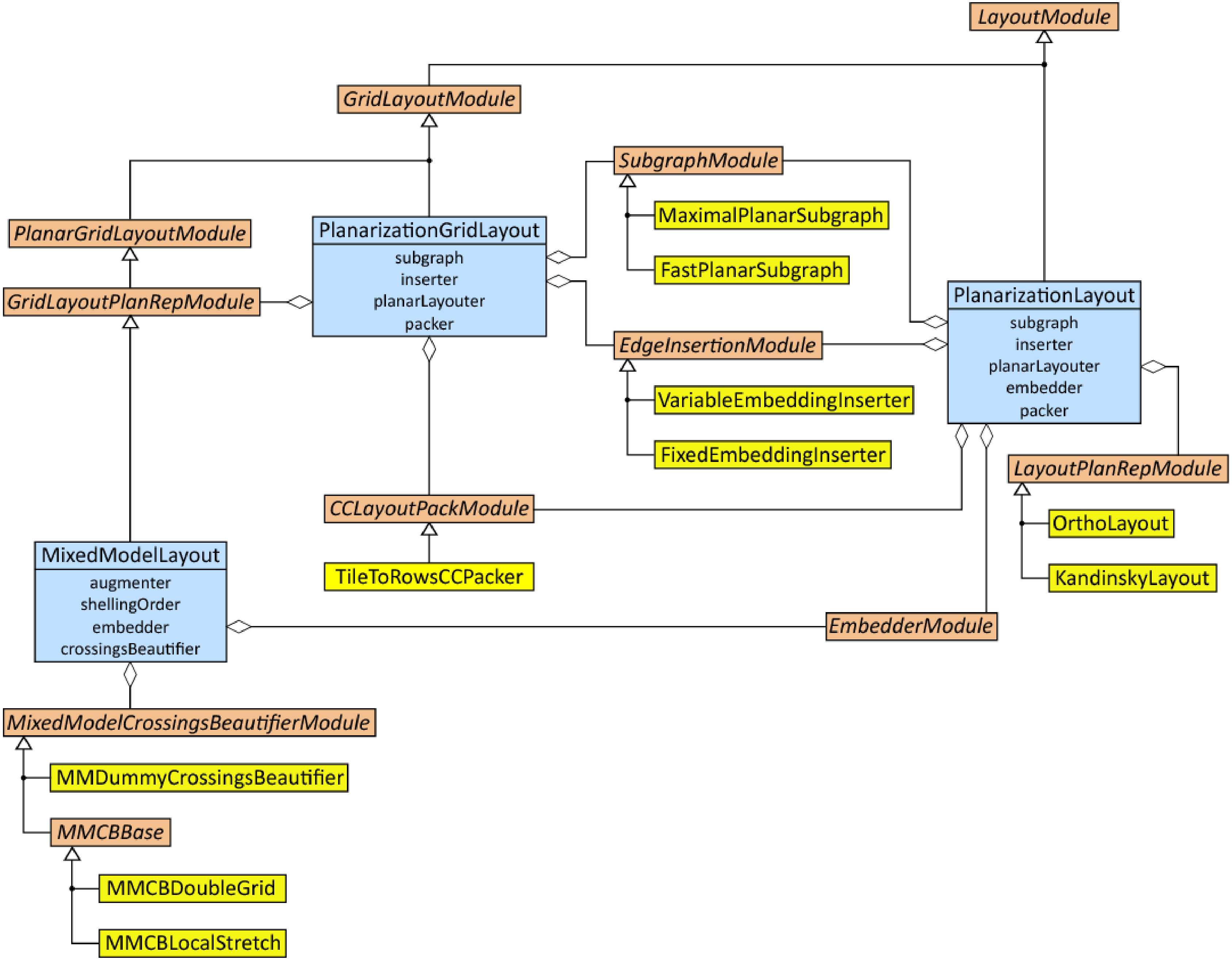




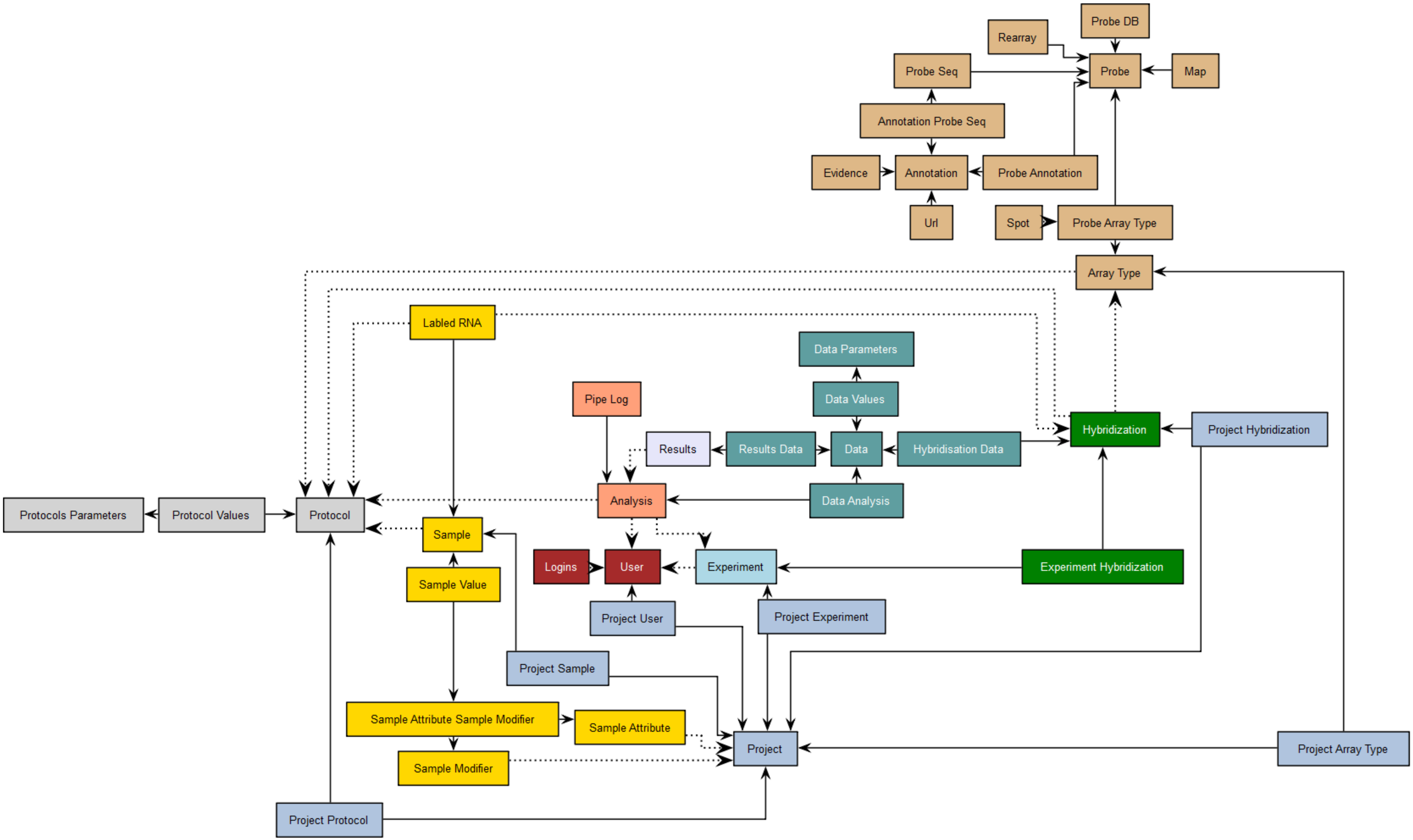

LayoutModule
GridLayoutModule
PlanarGridLayoutModule
PlanarStraightLayout
augmenter
shellingOrder
AugmentationModule
PlanarDrawLayout
augmenter
shellingOrder
GridLayoutPlanRepModule
DfsMakeBiconnected
PlanarAugmentation
ShellingOrderModule
MixedModelLayout
augmenter
shellingOrder
crossingsBeautifier
embedder
TriconnectedShellingOrder
BiconnectedShellingOrder
EmbedderModule
MixedModelCrossingsBeautifierModule
SimpleEmbedder
MMDummyCrossingsBeautifier
EmbedderMinDepthPiTa
MMCBBase
EmbedderMinDepth
MMCBDoubleGrid
EmbedderMaxFace
MMCBLocalStretch
EmbedderMinDepthMaxFace
EmbedderMinDepthMaxFaceLayers
EmbedderMaxFaceLayers

Planarization

- Extends planar layouts to non-planar graphs
- Crossing minimization replaces crossing by degree-4 dummy nodes
- Extensive framework for high-quality crossing minimization
- Two variants
  - PlanarizationGridLayout
  - PlanarizationLayout


LayoutModule
GridLayoutModule
SubgraphModule
PlanarGridLayoutModule
PlanarizationGridLayout
subgraph
inserter
planarLayouter
packer
MaximalPlanarSubgraph
FastPlanarSubgraph
GridLayoutPlanRepModule
PlanarizationLayout
subgraph
inserter
planarLayouter
embedder
packer
EdgeInsertionModule
VariableEmbeddingInserter
FixedEmbeddingInserter
LayoutPlanRepModule
CCLayoutPackModule
OrthoLayout
TileToRowsCCPacker
KandinskyLayout
MixedModelLayout
augmenter
shellingOrder
embedder
crossingsBeautifier
EmbedderModule
MixedModelCrossingsBeautifierModule
MMDummyCrossingsBeautifier
MMCBBase
MMCBDoubleGrid
MMCBLocalStretch

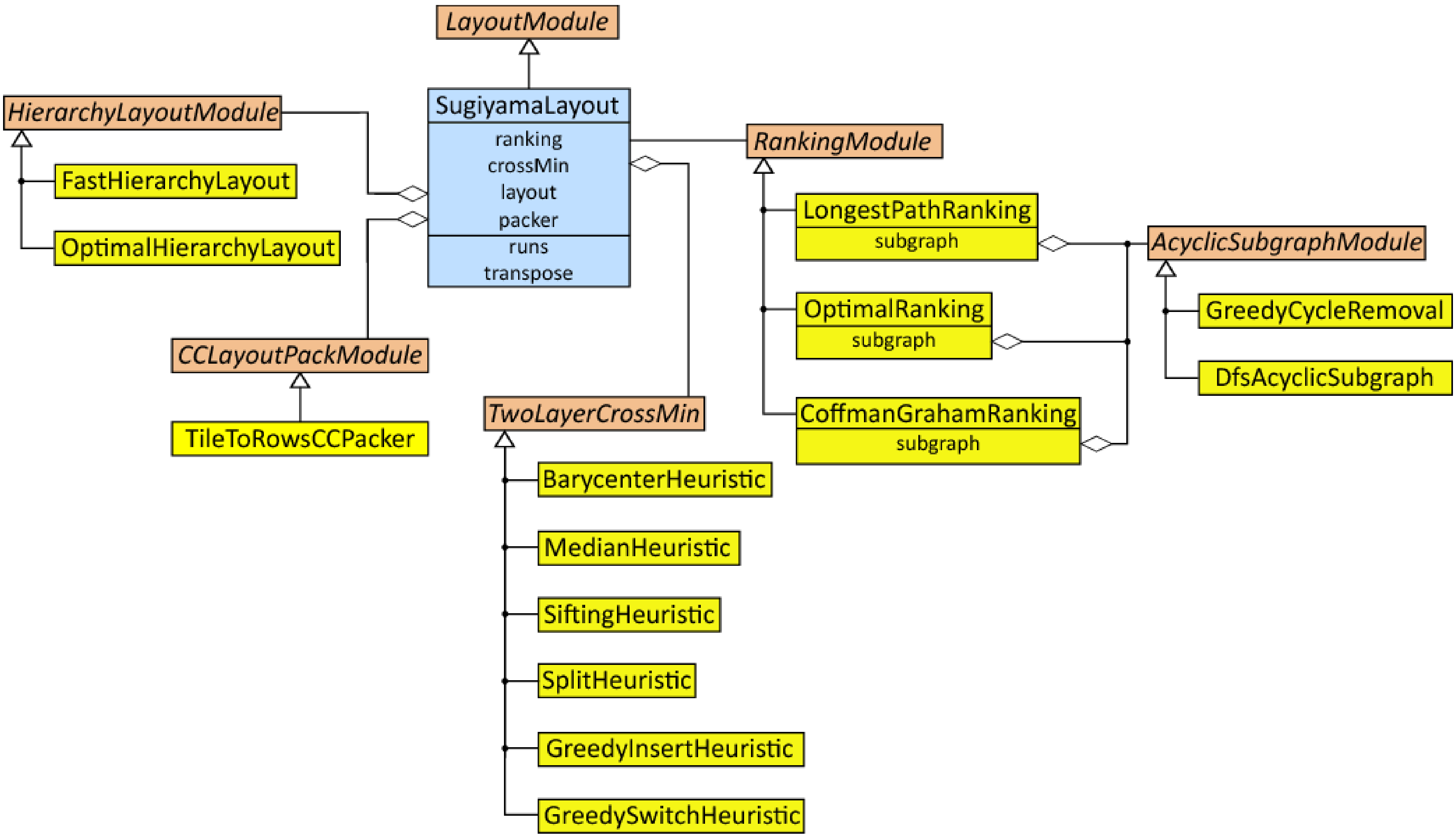
Example: PlanarizationLayout

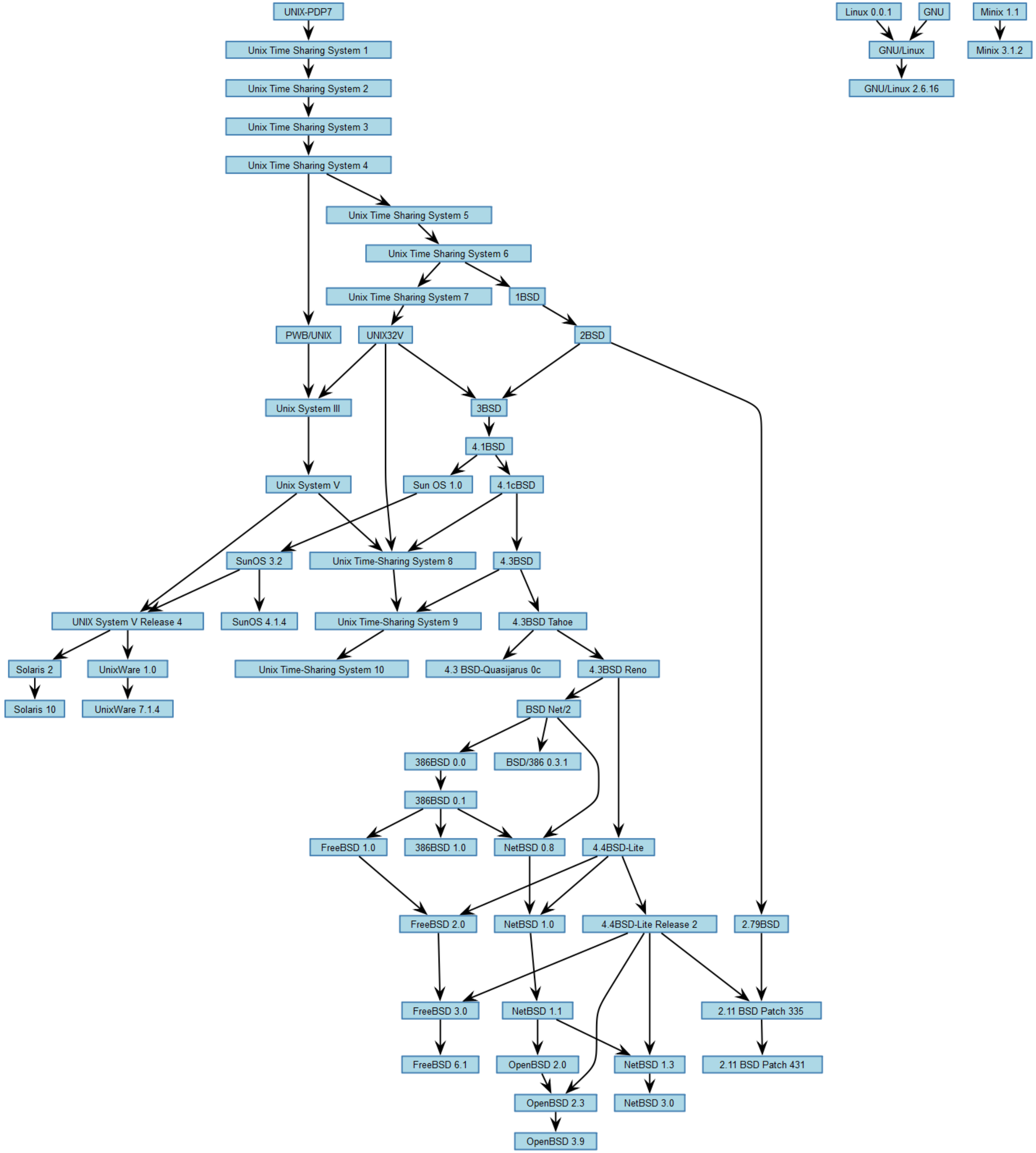


Hierarchical Layout

- SugiyamaLayout
  1. Compute a Layering
     (long edges are split so that edges connect nodes on adjacent layers)
  2. Minimize crossings by permuting nodes on layers
     (layer-by-layer-sweep $\rightarrow$ two-layer crossing minimization)
  3. Final layout: Assign y-coordinates to layers and x-coordinates to nodes
  4. Packing of connected components (if graph is not connected)
- Alternative: UpwardPlanarizationLayout

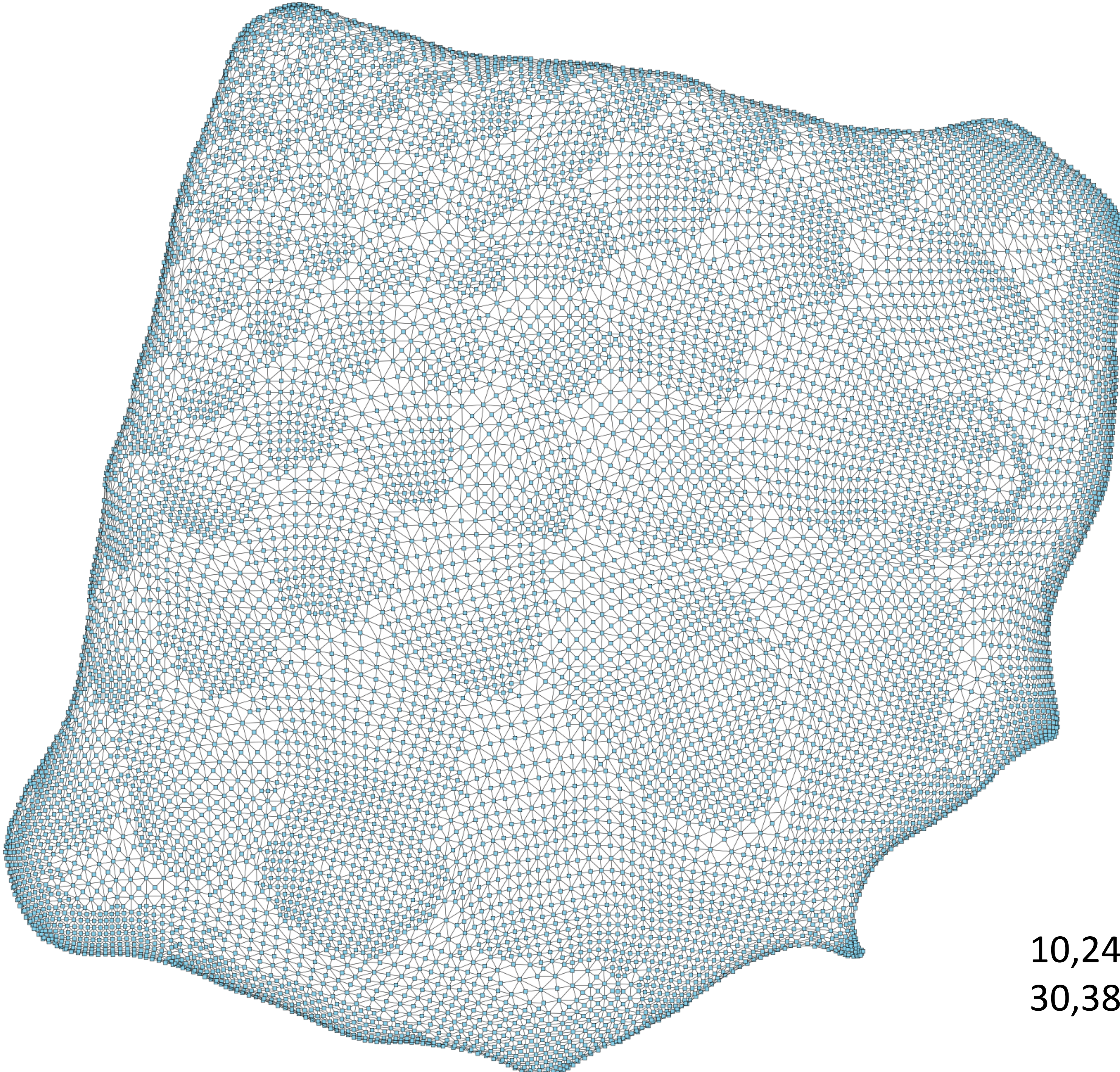
Framework for Sugiyama-Layout




UNIX-PDP7
Unix Time Sharing System 1
Unix Time Sharing System 2
Unix Time Sharing System 3
Unix Time Sharing System 4
Unix Time Sharing System 5
Unix Time Sharing System 6
Unix Time Sharing System 7
1BSD
PWB/UNIX
UNIX32V
2BSD
Unix System III
3BSD
4.1BSD
Unix System V
Sun OS 1.0
4.1cBSD
SunOS 3.2
Unix Time-Sharing System 8
4.3BSD
UNIX System V Release 4
SunOS 4.1.4
Unix Time-Sharing System 9
4.3BSD Tahoe
Solaris 2
UnixWare 1.0
Unix Time-Sharing System 10
4.3 BSD-Quasijarus 0c
4.3BSD Reno
Solaris 10
UnixWare 7.1.4
BSD Net/2
386BSD 0.0
BSD/386 0.3.1
386BSD 0.1
FreeBSD 1.0
386BSD 1.0
NetBSD 0.8
4.4BSD-Lite
FreeBSD 2.0
NetBSD 1.0
4.4BSD-Lite Release 2
2.79BSD
FreeBSD 3.0
NetBSD 1.1
2.11 BSD Patch 335
FreeBSD 6.1
OpenBSD 2.0
NetBSD 1.3
2.11 BSD Patch 431
OpenBSD 2.3
NetBSD 3.0
OpenBSD 3.9
Linux 0.0.1
GNU
Minix 1.1
GNU/Linux
Minix 3.1.2
GNU/Linux 2.6.16

Energy-based Layout

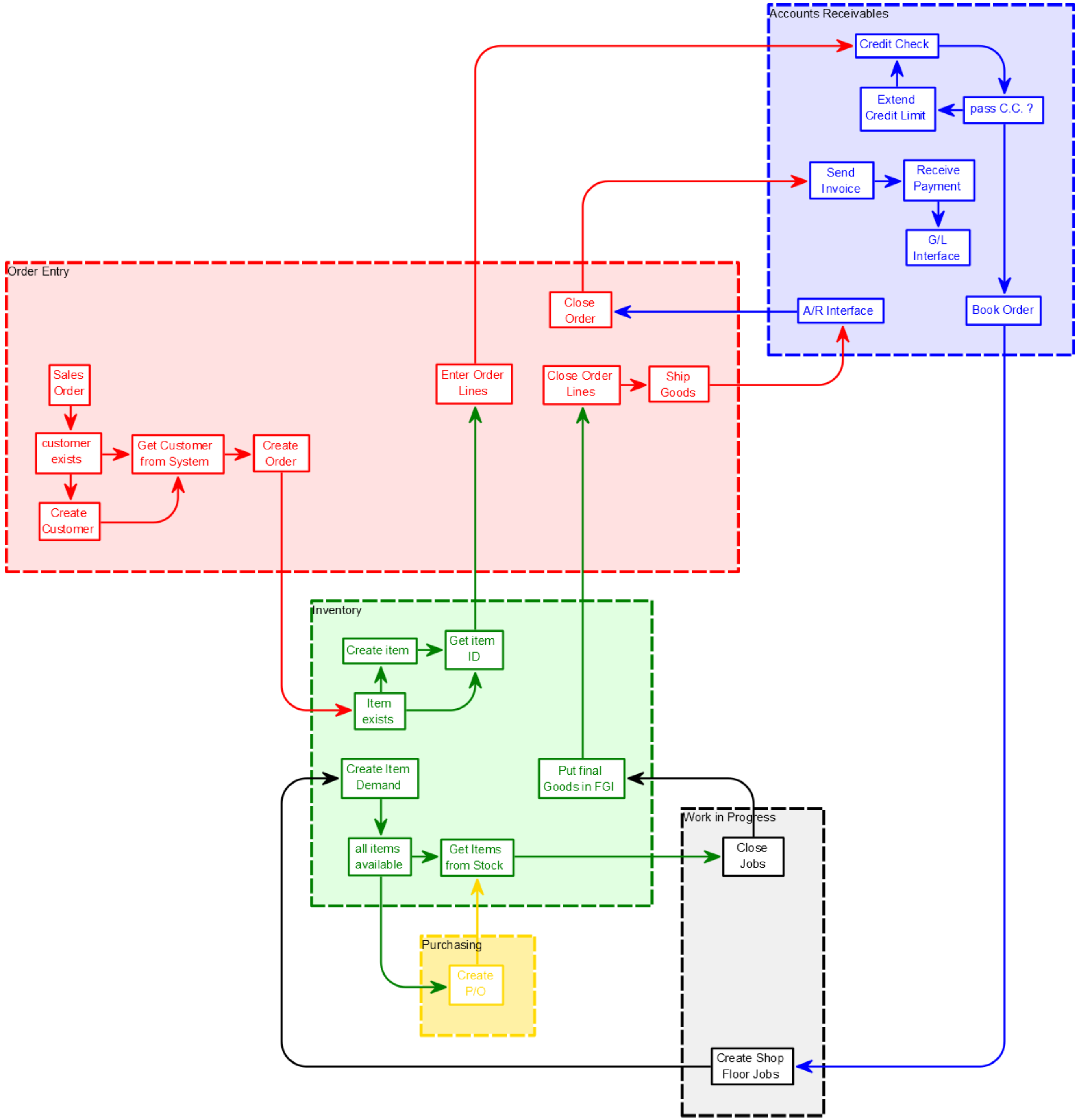
- Mostly model a “physical system” and try to minimize an energy function
- Many standard algorithms
  - spring embedder, Kamada-Kawai, GEM, Davidson-Harel, …
- Multilevel Algorithms for large graphs
  - FMMMLayout
  - FastMultipoleMultilevelEmbedder
  - MultilevelMixer



10,240 nodes
30,380 edges

Cluster Graph Layout

- ClusterPlanarizationLayout
  - Extension of orthogonal layout (planarization)
  - Supports hierarchically clustered graphs
- Also: SugiyamaLayout variant for clustered graphs


Accounts Receivables
Credit Check
Extend Credit Limit
pass C.C. ?
Send Invoice
Receive Payment
G/L Interface
A/R Interface
Book Order
Order Entry
Close Order
Sales Order
Enter Order Lines
Close Order Lines
Ship Goods
customer exists
Get Customer from System
Create Order
Create Customer
Inventory
Create item
Get item ID
Item exists
Create Item Demand
Put final Goods in FGI
all items available
Get Items from Stock
Work in Progress
Close Jobs
Purchasing
Create P/O
Create Shop Floor Jobs

Tutorial

Prerequisites

- C++-Compiler
  - Windows: Visual Studio 2008, 2010
  - Linux / Mac: g++ 4.x
- OGDF v2012.05 installed
- How do we visualize the layouts?
  - gml2pic utility
    see http://www.ogdf.net/doku.php/project:gml2pic
  - compiling requires Qt library (4.7 or 4.8)
  - Windows installer self-contained

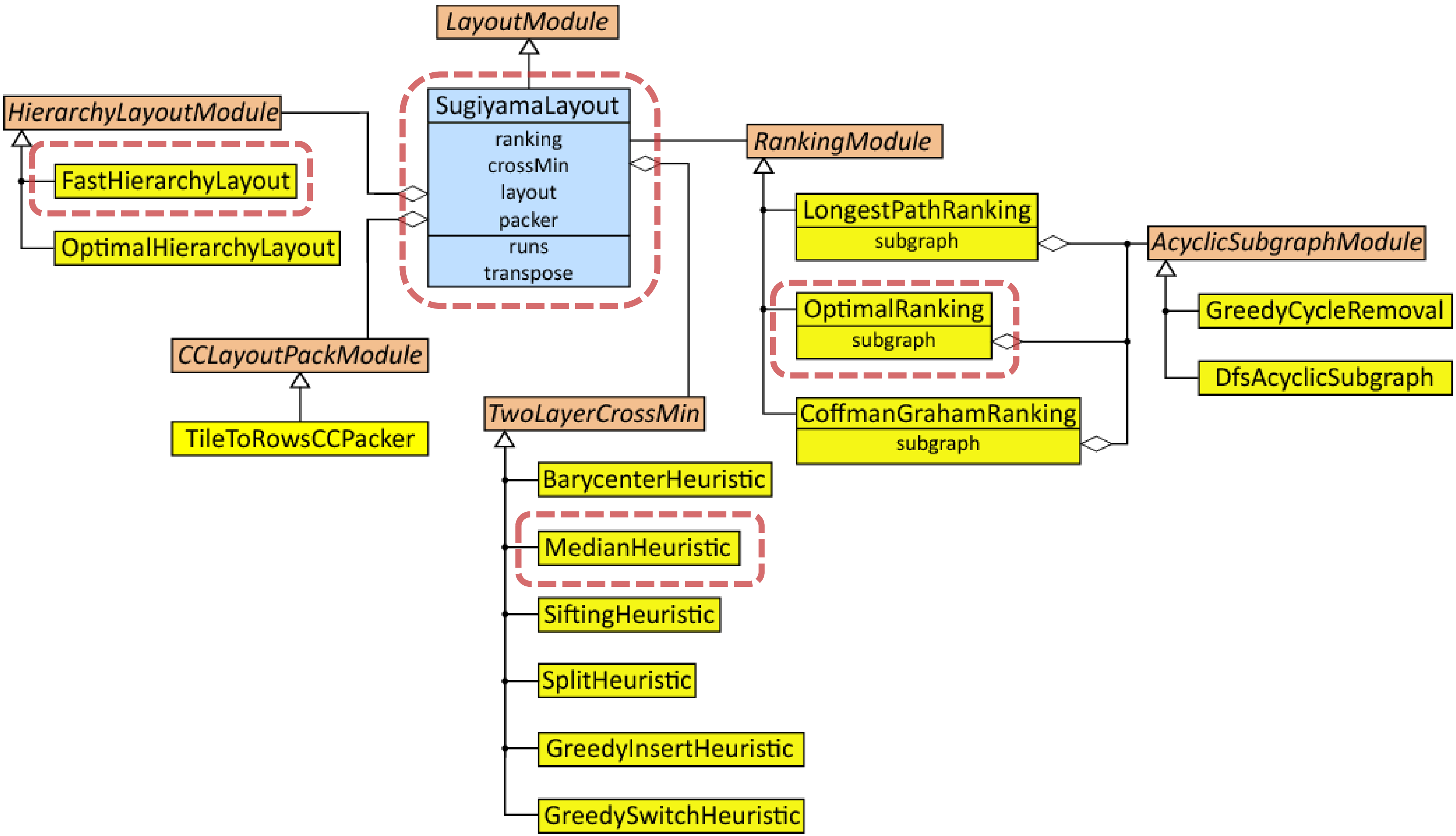
Setting-up a Project

- Assume OGDF is installed in *OGDF-INSTALL-DIR* (build as static library)
- Add *OGDF-INSTALL-DIR* to the compiler include path
- Add the path to OGDF library to the library search path, e.g.
  - Windows, 32-bit, Release: *OGDF-INSTALL-DIR*/Win32/Release
  - Linux, release: *OGDF-INSTALL-DIR*/_release
- Link against OGDF
  - Windows: Add ogdf.lib and psapi.lib to the linker input
  - Linux: Link with `-lOGDF -pthread`
- When linking against debug versions
  - Define OGDF_DEBUG when compiling

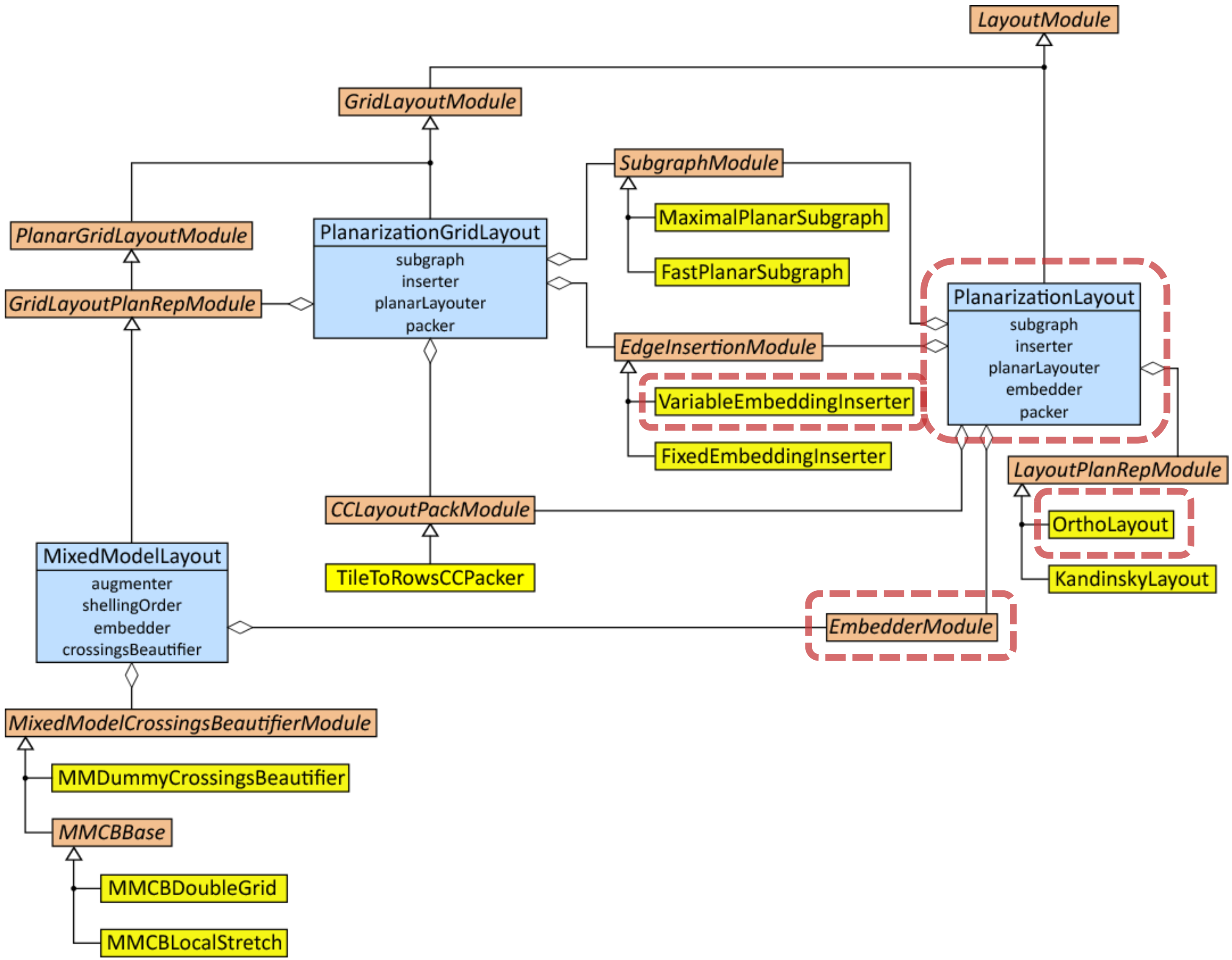
Compiling with g++

- Assume
  - our source-code is just tutorial1.cpp
  - OGDF is installed in the directory above
- Compile and link with
  - `g++ -I../OGDF tutorial1.cpp -o main -L../OGDF/_release -lOGDF -pthread`

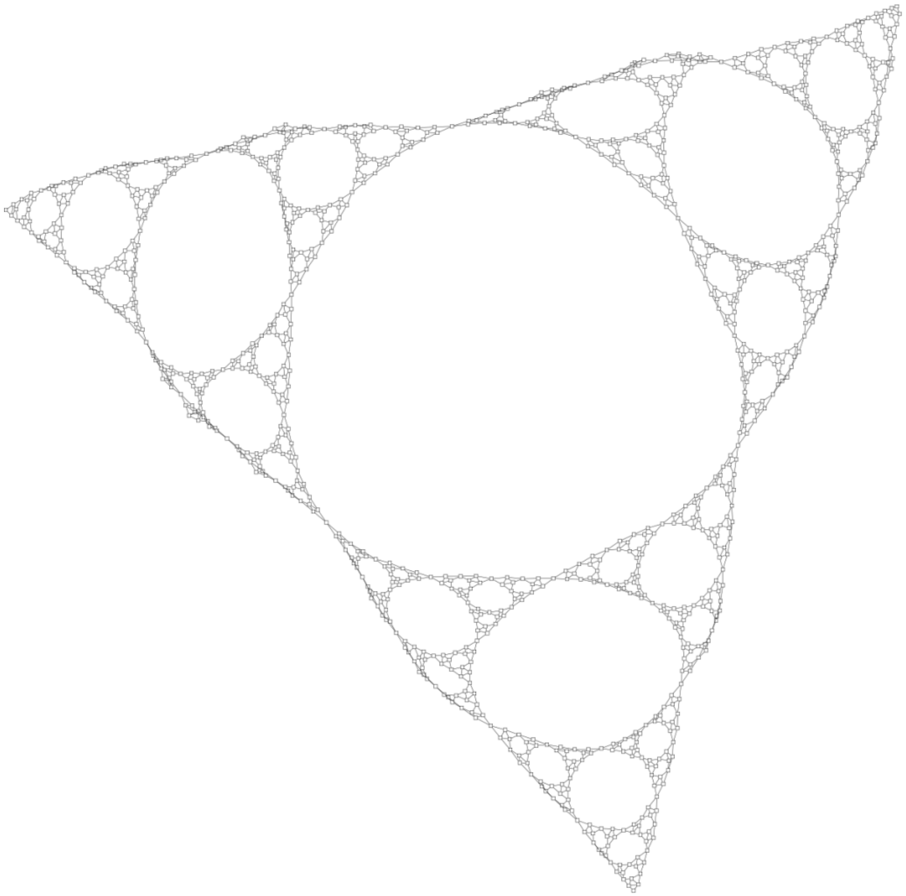
Example 1: Sugiyama-Layout



Example 2: Orthogonal Layout



Example 3: Large Graphs



sierpinski_06
1095 nodes, 2187 edges

crack
10240 nodes, 30380 edges