OpenSees

E
E
R

OpenSees Programming Bootcamp

Frank McKenna
NHERI-SimCenter
UC Berkeley

POLL (status)

Outline

- Git 10 min
- OpenSees Software Architecture 5 min
- C++ Programming 30 min
- Adding Element to OpenSees 15 min
- Exercise 60 min

OpenSees

E
E
R

Basic Git Commands

Frank McKenna
NHERI-SimCenter
UC Berkeley

Essential Git Commands

- git init
- git add
- git commit
- git checkout

- git clone
- git pull
- git push

DEMO

OpenSees

E
E
R

OpenSees Architecture

Frank McKenna
NHERI-SimCenter
UC Berkeley

Most Important Concept in all of Computer Science?

- Layers of Abstraction (Donald Knuth)
- How do you take a Complicated Problem or System and chop it up into pieces that you can build relatively independently (John Ousterhaut)

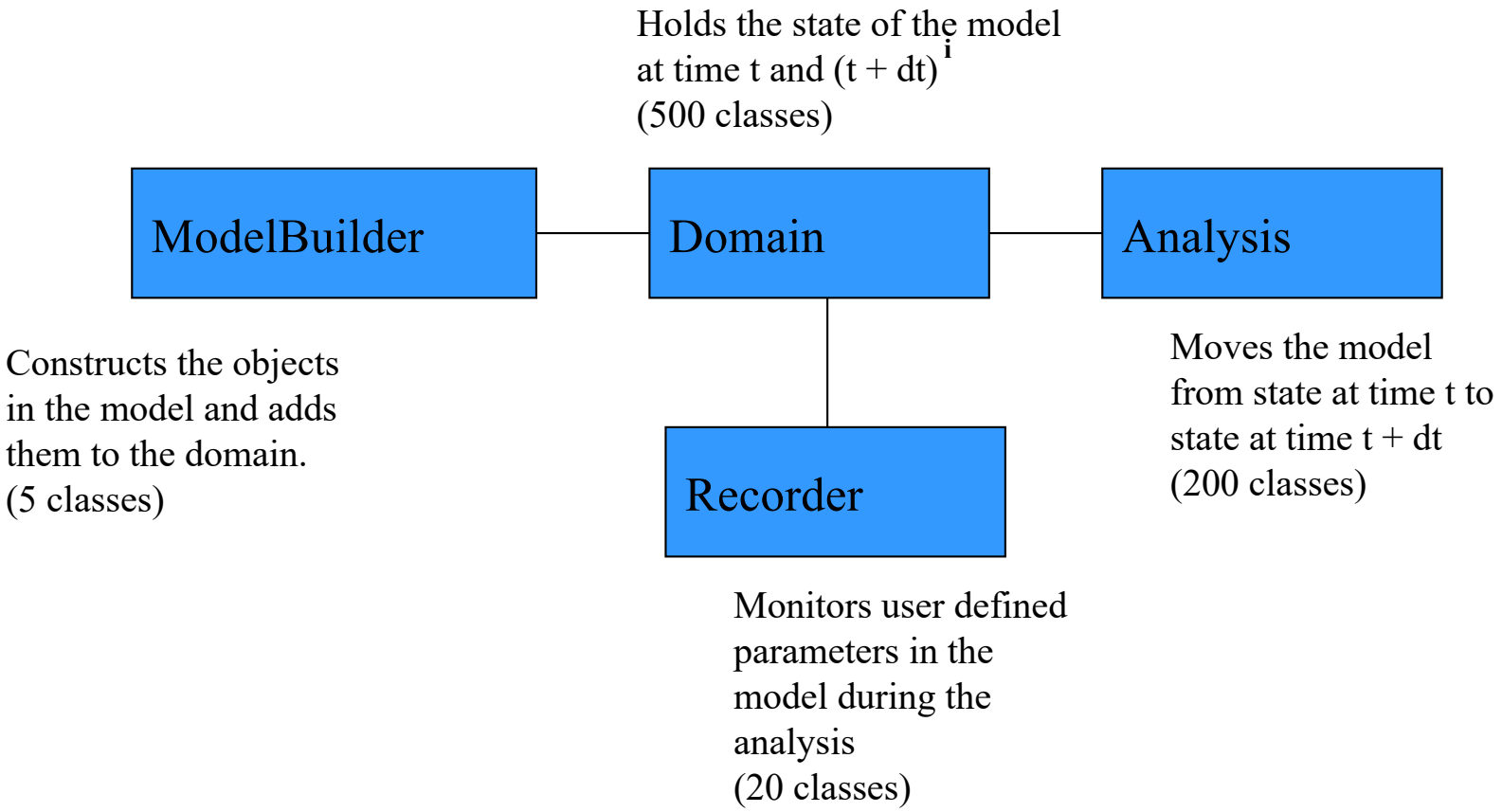
A Philosophy of Software Design | John Ousterhout | Talks at Google

What is OpenSees?

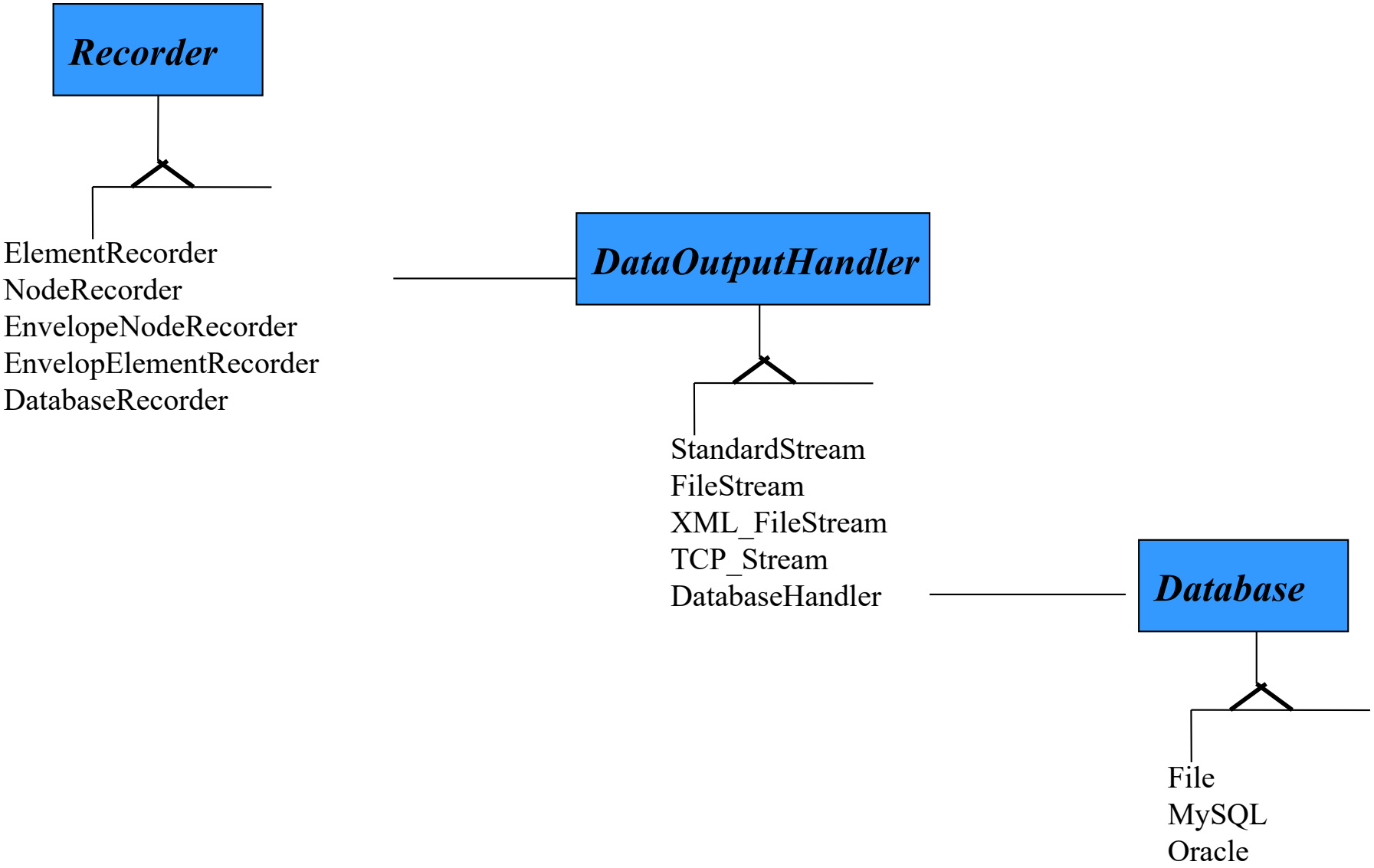
The Open System for Earthquake Engineering Simulation is:

- A software *framework* for developing sequential, parallel and grid-enabled finite element applications.
- A communication mechanism for exchanging and building upon research accomplishments in Structural Engineering.
- As open-source software, it has the potential for being a community code for natural hazards engineering. http://opensees.berkeley.edu/phpBB2/index.cgi

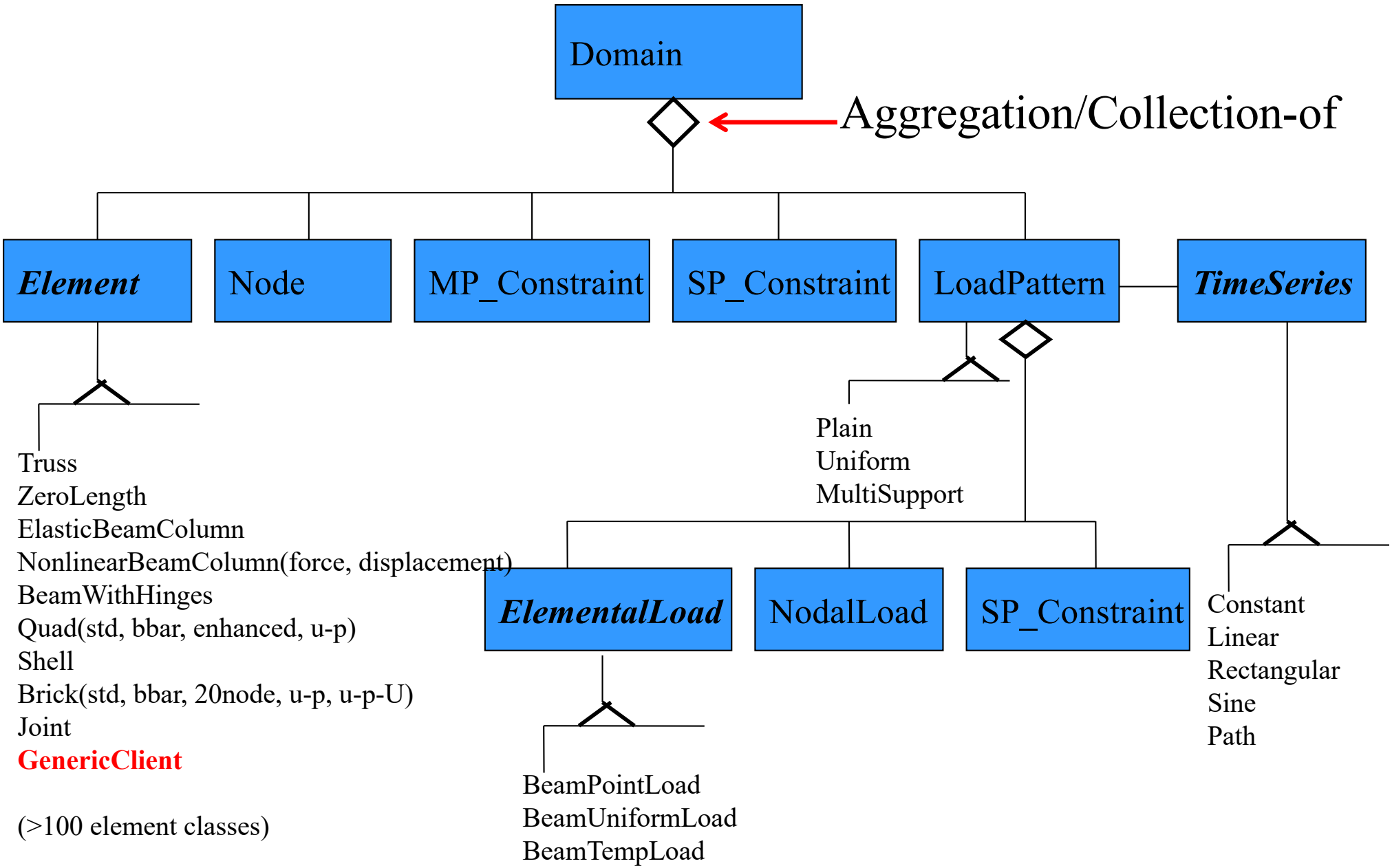
Main Abstractions in OpenSees Framework



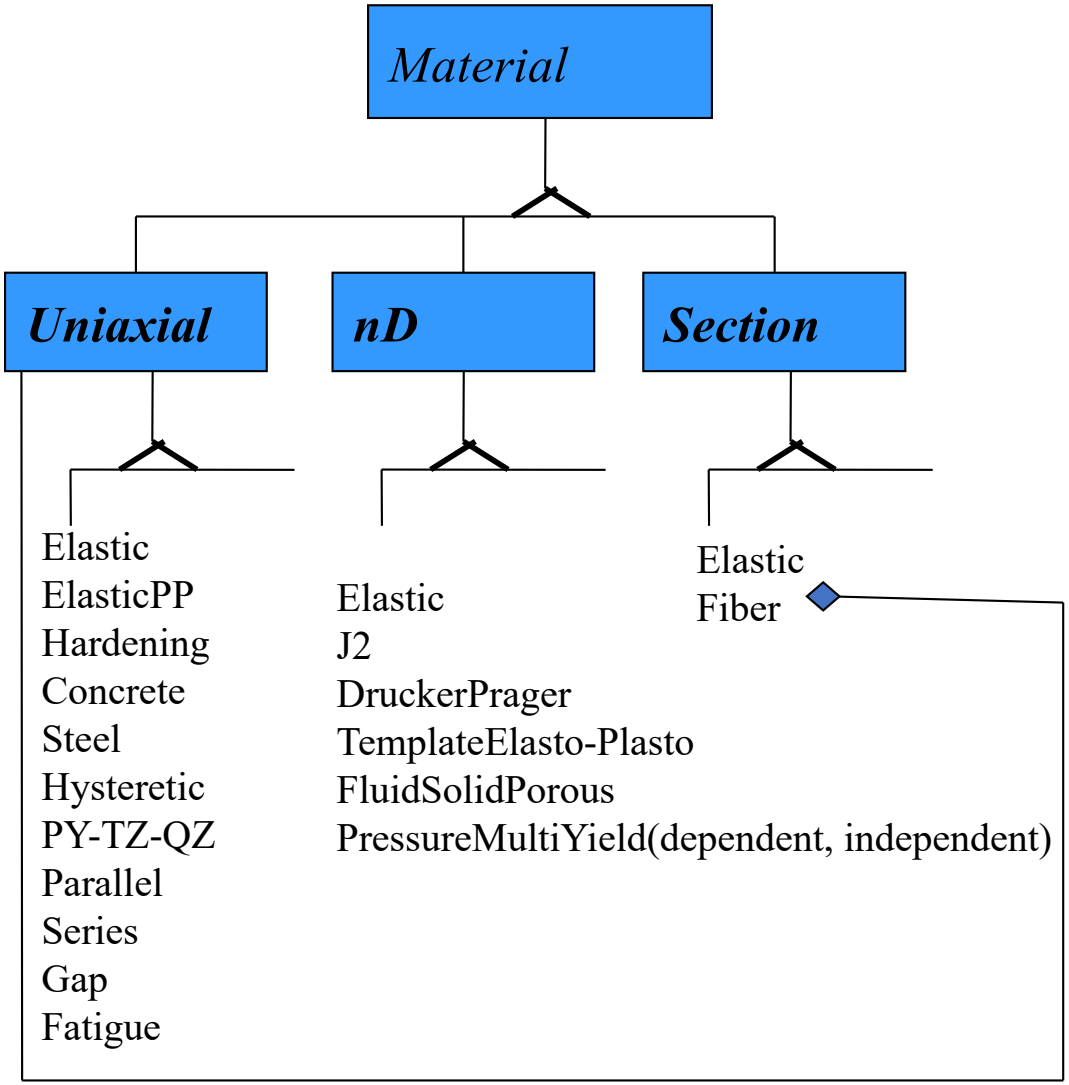
Recorder Abstraction



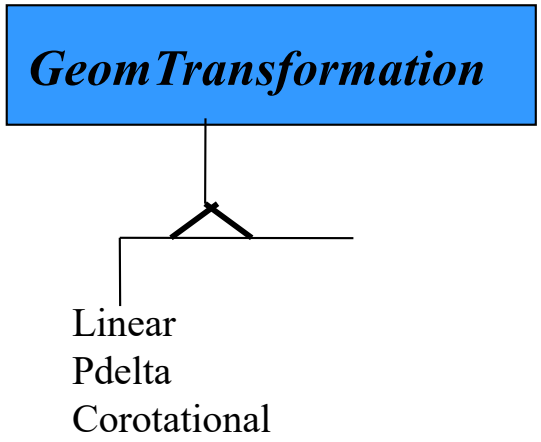
What is in a Domain?



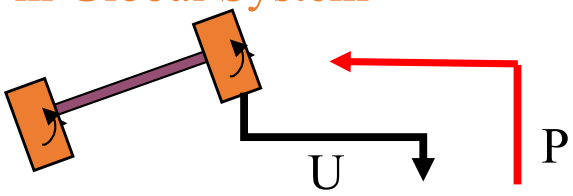
Classes associated with Elements:



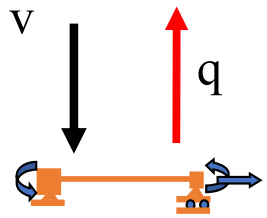
(over 250 material classes)



Element in Global System

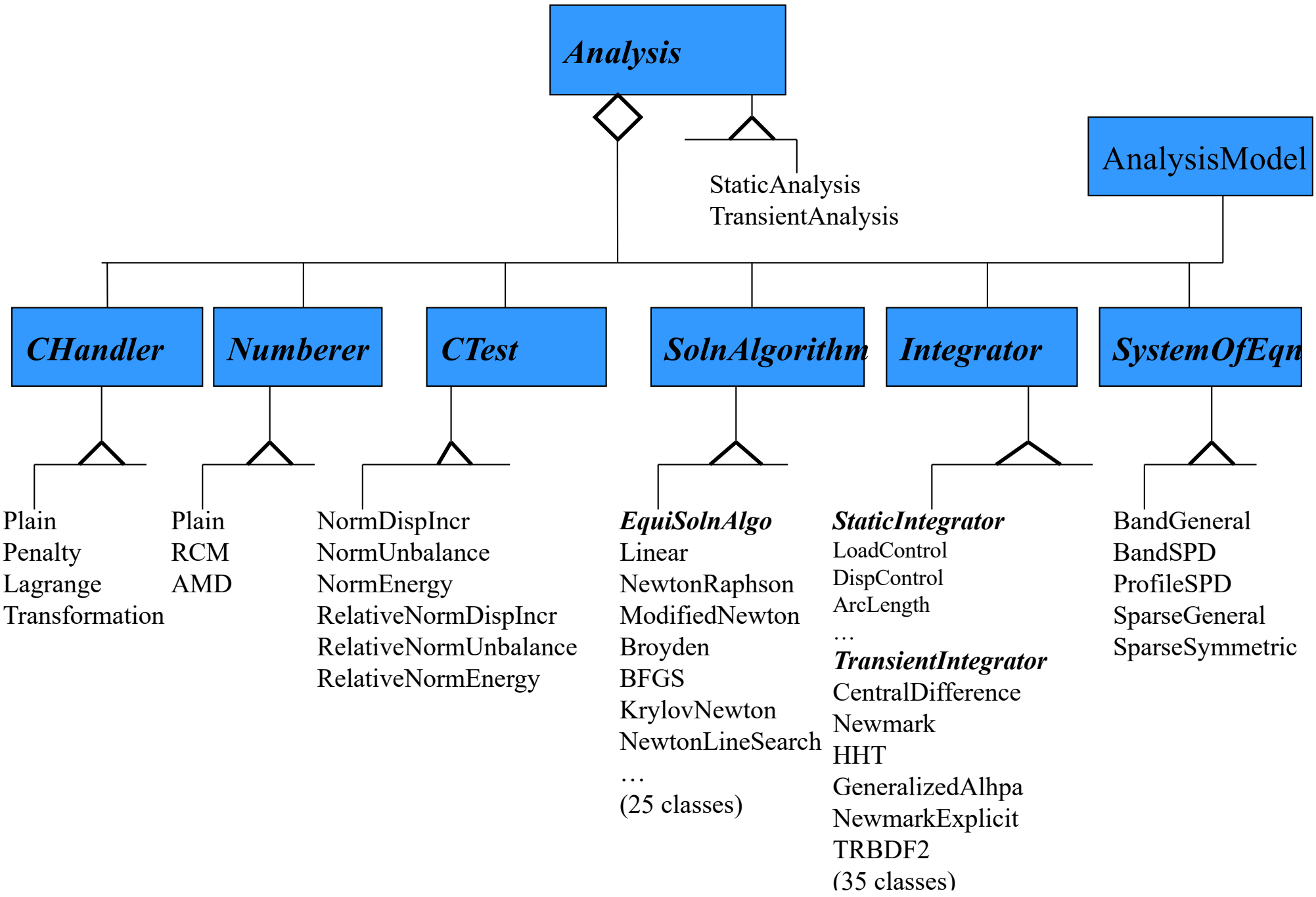


Geometric Transformation



Element in Basic System

What is an Analysis?



What Applications Are Built With the Framework?

1. OpenSees – tcl interpreter

2. OpenSeesPy – Python Module for extending Python

3. SimCenter NHERI Applications

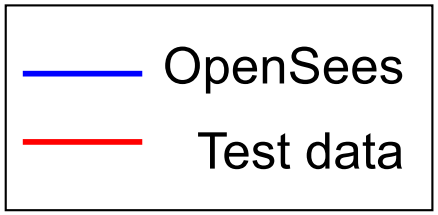
QUOFEM — Sampling, Sensitivity, Reliability, Calibration, Generation of Surrogates

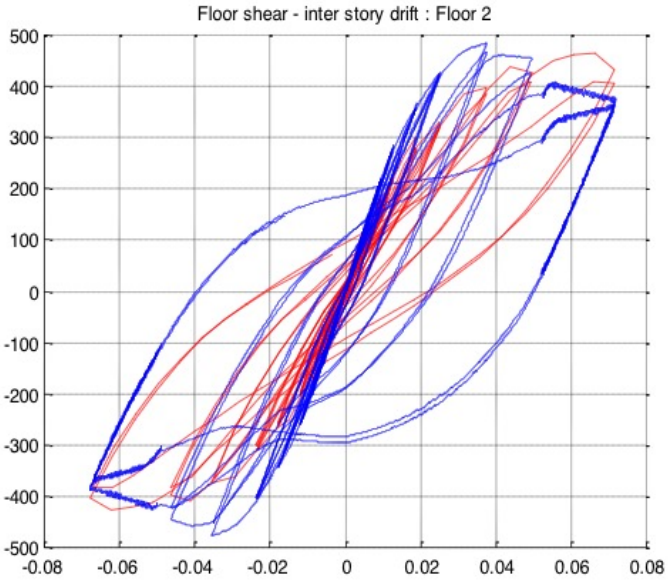
EE-UQ, WE-UQ, Hydro-UQ — Response of structure to natural hazard effects: ground shaking, wind effects, and surge/tsunami flows

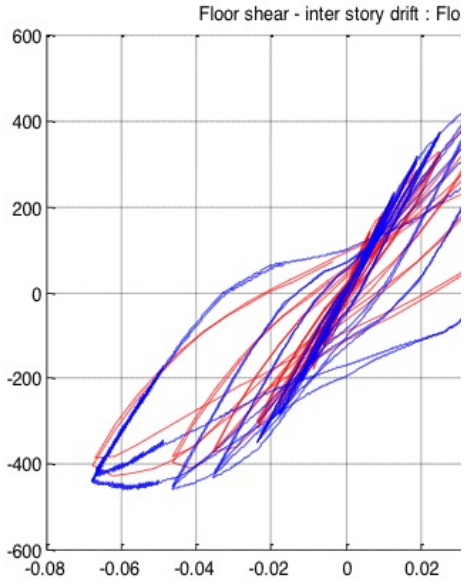
PBE — Performance-based computations of individual facilities to natural hazards

R2D — Regional assessment of facilities and systems to natural hazards to support resilience decision making

Remembering Why We do Finite Element Analysis NCREE frame tested at the Taiwan facility







POLL (homework)

OpenSees

E
E
R

C++ For Structural Engineers

Frank McKenna
NHERI-SimCenter
UC Berkeley

Some Quotes to Start With:

- *The sooner you start to code, the longer the program will take.* — Roy Carlson
- *Simplicity is prerequisite for reliability*.– Edsger Dijkstra
- *Any fool can write code that a computer can understand. Good programmers write code that humans can understand*. – Martin Fowler

C Basics

- Fundamental data types: char, int, float, double
- Derived Types: pointers, arrays, structures
- Variables
- Operators =, +, -, *, /, %, <, <=, >, >=, ==, !=
- Control-flow Constructs: if-else, while, do, for
- Procedures
- Libraries, lots of libraries.
- C Help:
  - The C Programming Language, Brian W. Kernighan and Dennis W. Ritchie, Prentice-Hall.

Hello World!

```
#include <stdio.h>

int main(int argc, char **argv) {
   printf(“Hello World!”);
}
```

Variables and Pointers

- A variable in a program is something with a name, the value of which can vary. In a running program, the compiler/linker assigns a specific block of memory to hold the value of the variable.

```
int k;    /* declaration of variable k to be an int */
k = 2;    /* set the value of k to be 2    */
```

- A pointer variable is a variable designed to hold an address of a block of memory.

```
int *kPtr;      /* declaration of variable kPtr to hold the address of an int*/
kPtr = &k;      /* set the value of kPtr to be address of k */
*kPtr  = 10;    /* set the value of what kPtr is pointing, i.e. k,  to to be 10*/
```

Arrays and Structures

- An array is a contiguous block of memory.

```
int kArray[10];   /* declaration of variable kArray to be an array of 10 integers */
kArray[0] = 2;    /* set the value of the first to be 2 */
kPtr = &kArray[9];  /* set the value of kPtr to be address of last element of array*/
kPtr ++;
*kPtr = 5;   // OOPS! - segmentation fault would be NICE but not guaranteed!
```

- A structure is a user defined collection of data. Unlike arrays, where all members have same data type, structures can group together variables of different data types.

```
typedef struct truss {
    int tag;
    int nodes[2];
    double A;
    double E;
 } Truss;
```

```
Truss t1;   /* struct truss t1
Truss *elePtr = &t1;

t1.nodes[0] = 2;
(*elePtr).nodes[1]=3;
```

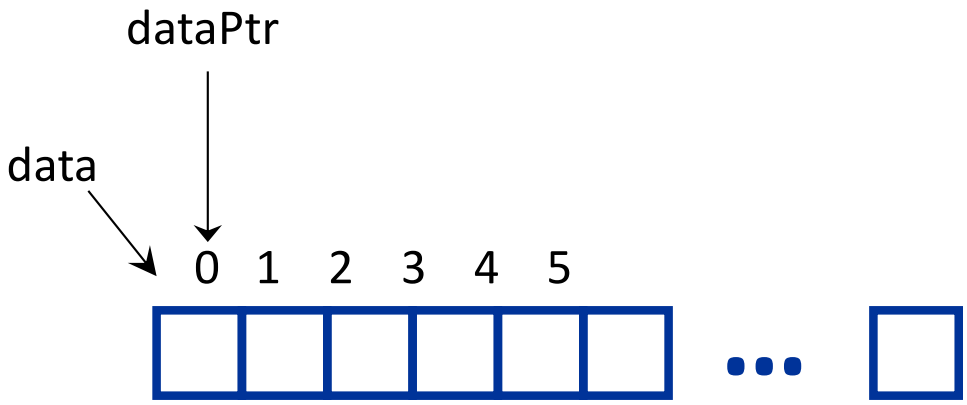
Example – don't do this at home!

```
#include <stdio.h>
#include <stdlib.h>
#define size 10
double sumValues(int, double *);
main () {
  int i;
  double values[size];
  for (i=0; i<size; i++) {
     values[i] = rand();
      printf("random number: %f\n",values[i];
  }
  sum = sumValues(size, values);
  printf("\n sum of numbers: %f\n",sum);
}
```

```
double sumValues(int n, double *data) {
  int i =0;
  double sum =0.0;
  while (i < n) {
     sum = sum+data(i);   /* sum +=data(i) */
     i = i+1;                    /* i++ */
  }
  return sum;
}
```

sumValues() - with a pointer and a do

```
double sumValues(int n, double *data) {
    int i =0;
    double sum =0.0;
    double *dataPtr = data;
    do {
      sum += *dataPtr;
      i++;
      dataPtr++;
  } while (i < n)
   return sum;
}
```



C++ Basics

- C++ is an extension of the C language
  - adds REFERENCES
  - adds CLASSES

C++ Pointers and References

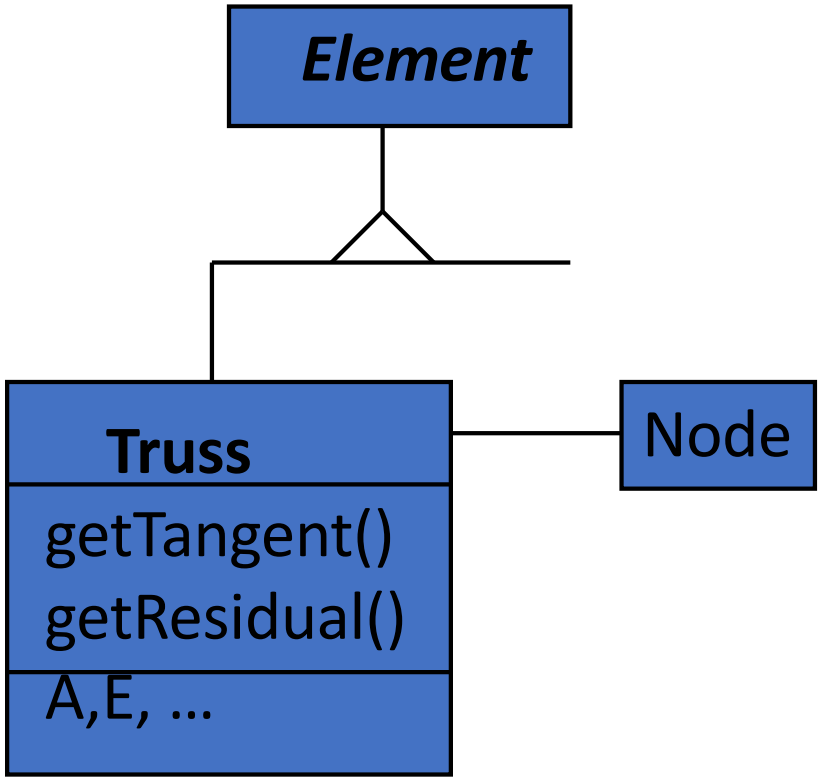
```
void sum(double a, double b, double *c) {
   double result = a + b;
   *c = result;
}
```

```
void sum(double a, double b, double &c) {
   double result = a + b;
   c = result;
}
```

C++ Classes

- A class is a C++ construct to hold both data and functions in the same block of memory.
- Classes typically have a definition which outlines the functions and variables, and their accessibility (public, protected, private). The definition is typically placed in a header file.
- Class also has an implementation. This is where the functions (methods) are defined. This is (typically) placed in a separate file, the implementation file.
- A Class can inherit both variables and implementation from a parent class. This is termed **inheritance**.
- A Class can override (redefine) the methods of the parent class. This is termed **polymorphism**.

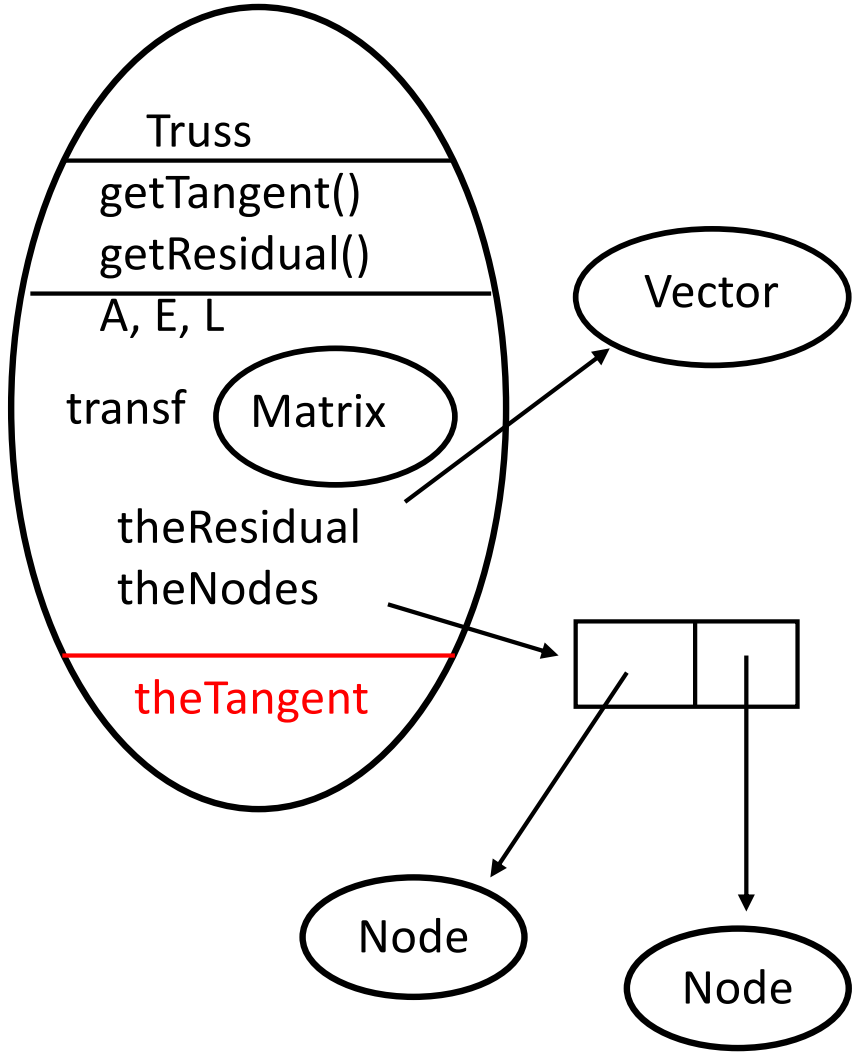
Simple Truss Example




```
class Truss : public Element {
    public:
            Truss(double A, double E,
                Node *node1, Node *node2);
            ~Truss();
        const Matrix &getTangent();
        const Vector &getResidual();
    private:
            double computeTrialStrain();
            double A, E, L;
            Matrix transf;
            Vector *theResidual;
            Node **theNodes;
            static Matrix theTangent;
};
```

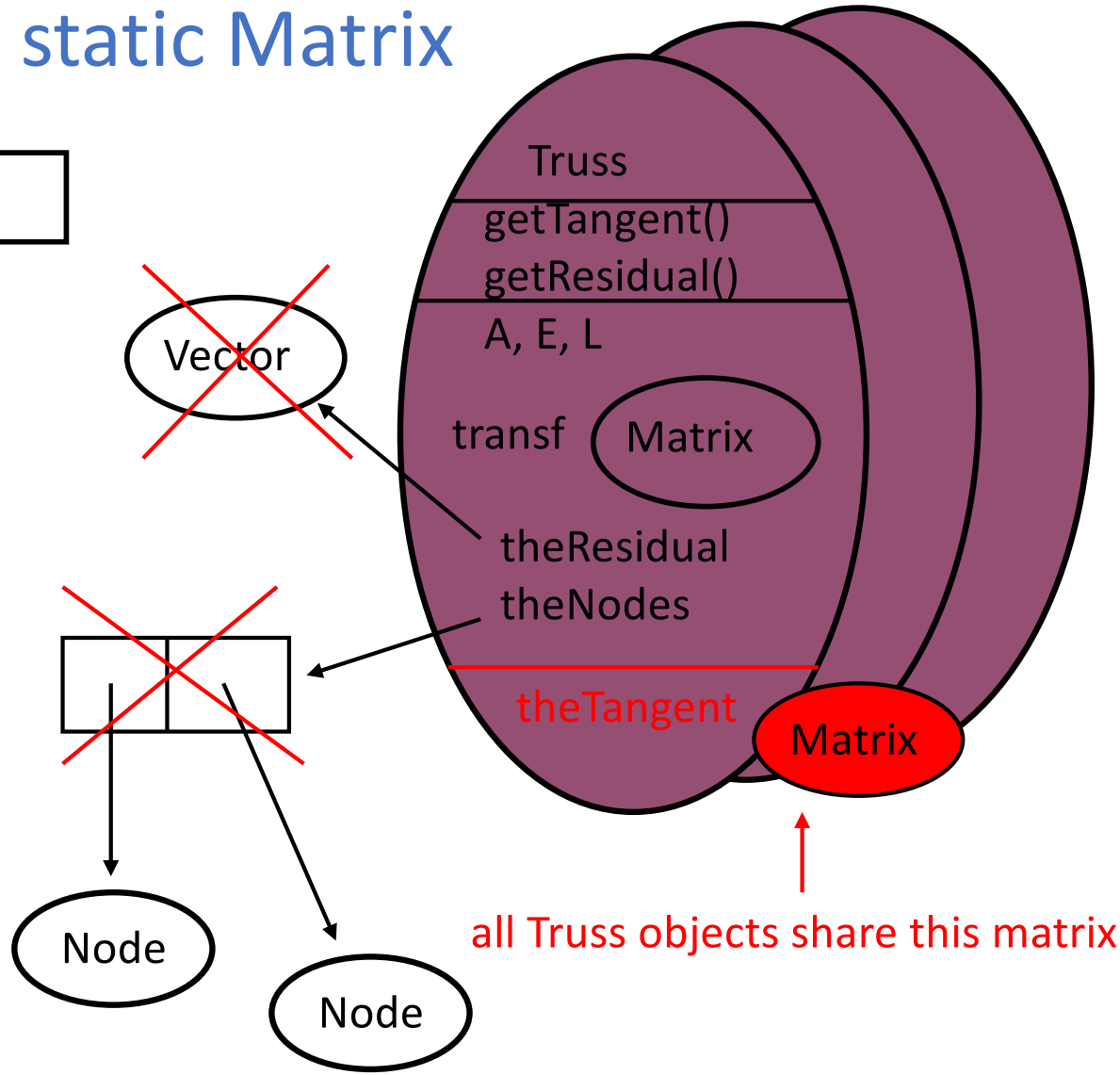
Constructor

```
Truss::Truss(double a, double e,
                Node *node1, Node *node2)
:Element(), A(a), E(e), transf(1,4)
{       theResidual = new Vector(4);
        theNodes = new Node*[2];
        theNodes[0] = node1;
            theNodes[1] = node2;
        Vector &crd1 = node1->getCrds();
        Vector &crd2=  node2->getCrds();
        double dx = crd2(0) - crd1(0);
        double dy = crd2(1) – crd1(1);
        L = sqrt(dx * dx + dy * dy);
        double cs = dx/L; double sn = dy/L;
        trans(0,0) = -cs;  trans(0,1) = -sn;
             trans(0,2) =  cs; trans(0,3) =  sn;
}
```



And remember that static Matrix

```
Matrix Truss::theTangent(4,4);
```



Destructor

```
Truss::~Truss()
{
    delete theResidual;
    delete [ ] theNodes;
}
```

Typically only delete objects you constructed

Public Methods

```
 const Matrix &getTangent(void) {
      theMatrix = transf ^ transf;
      theMatrix *= A * E / L;
      return theMatrix;
}

const Vector &getResidual() {
      double strain = this->computeStrain();
      double force = A * E / L * strain;
      Vector &resid = *theResidual;
      for (int i=0; i<4; i++)
            resid(i) = transf(0,i) * force;
      return resid;
}
```

Private Method

```
double Truss::computeTrialStrain() {
   Vector &disp1 = theNodes[0]->getTrialDisp();
   Vector &disp2 = theNodes[1]->getTrialDisp();
   double dLength = 0.0;
   for (int i=0; i<2; i++)
       dLength -= (disp2(i)-disp1(i)) * trans (0,i);
  double strain = dLength / L;
  return strain;
}
```

POLL (still with me)

45 min IN? (5 Min BREAK)

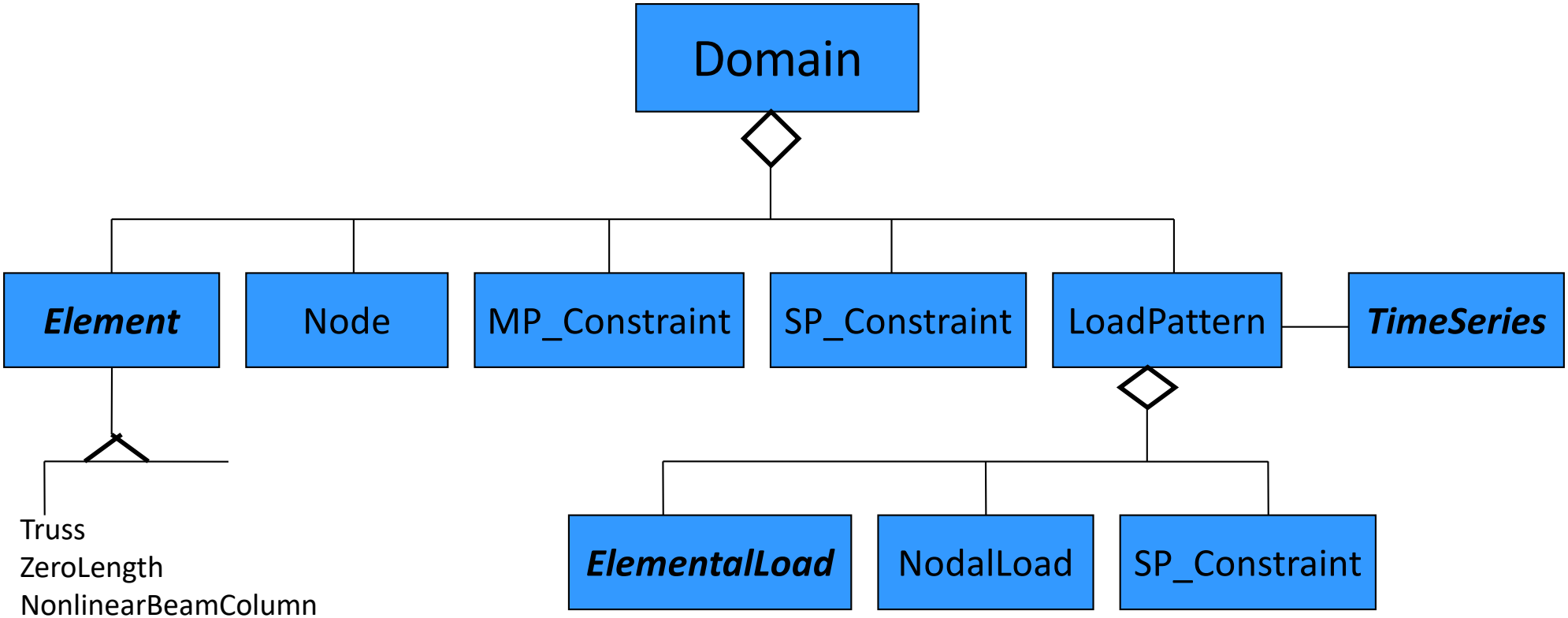
OpenSees

E
E
R

Adding a New Element to OpenSees

Frank McKenna
NHERI-SimCenter
UC Berkeley

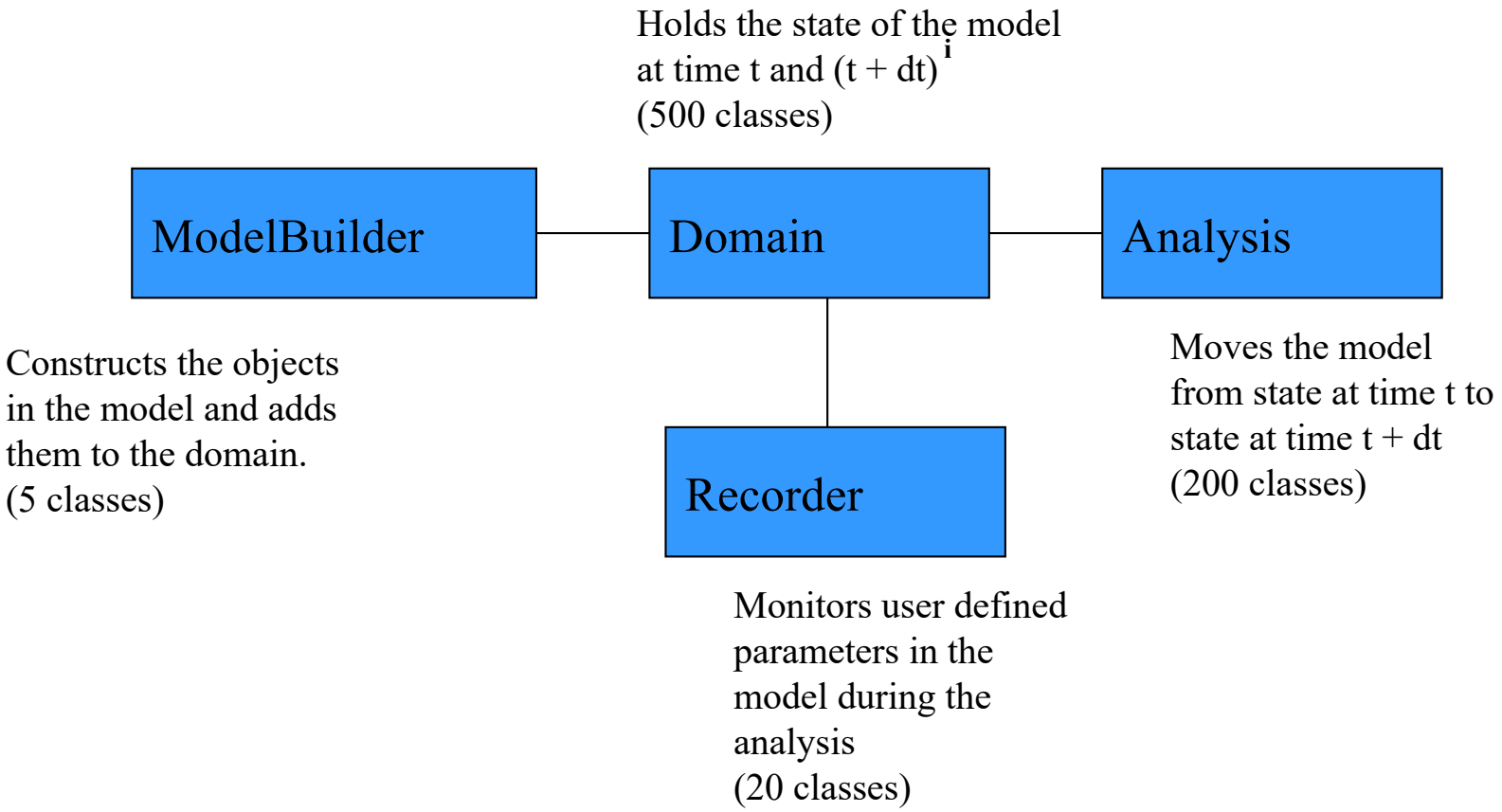
Todays Exercise:



Truss
ZeroLength
NonlinearBeamColumn
BeamWithHinges
Quad
Shell
Brick
PlanarTruss

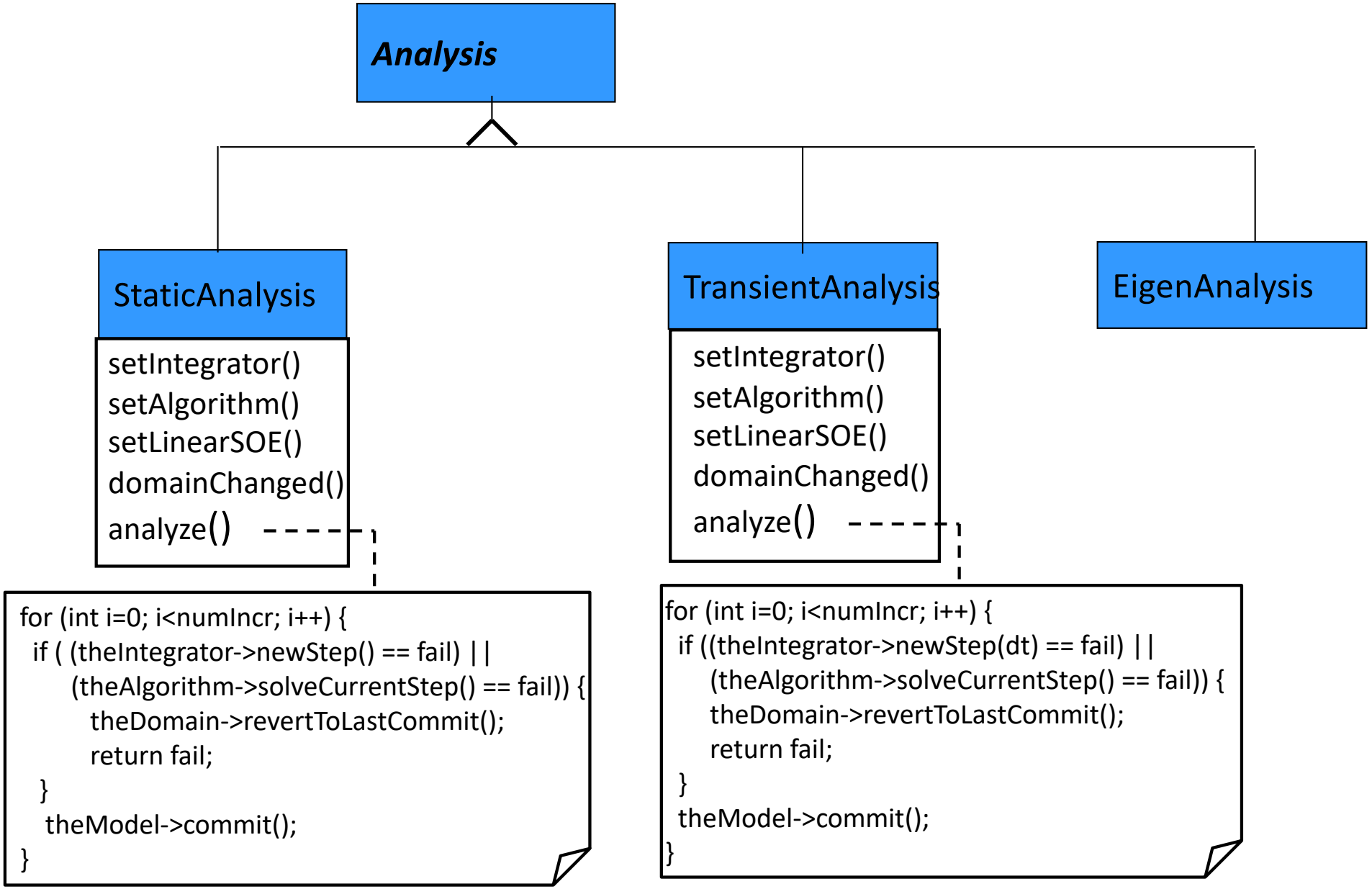
We are going to add a 2d truss to OpenSees!

Main Abstractions in OpenSees Framework



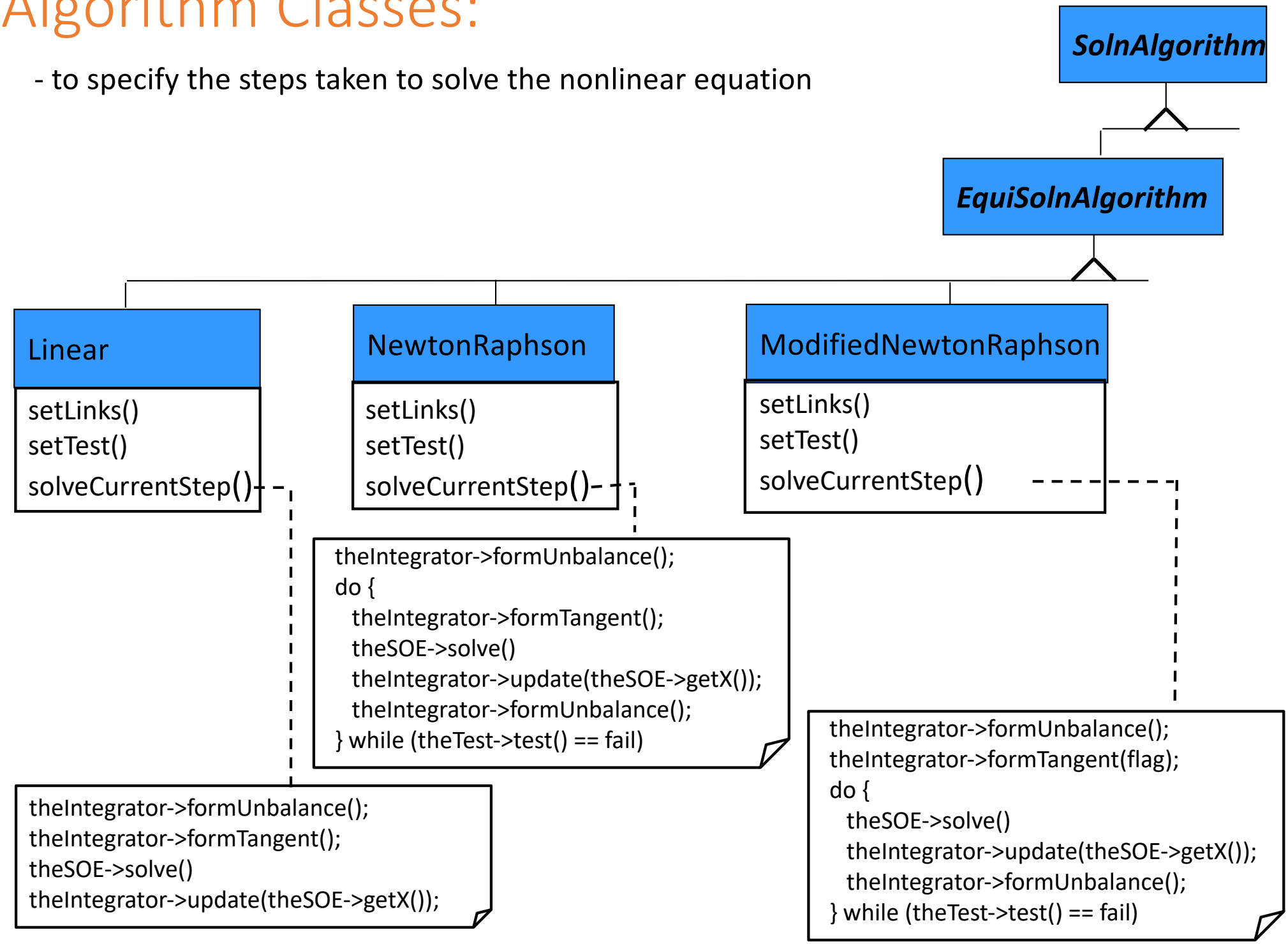
Analysis Classes:

- to update the state of the Domain



Algorithm Classes:

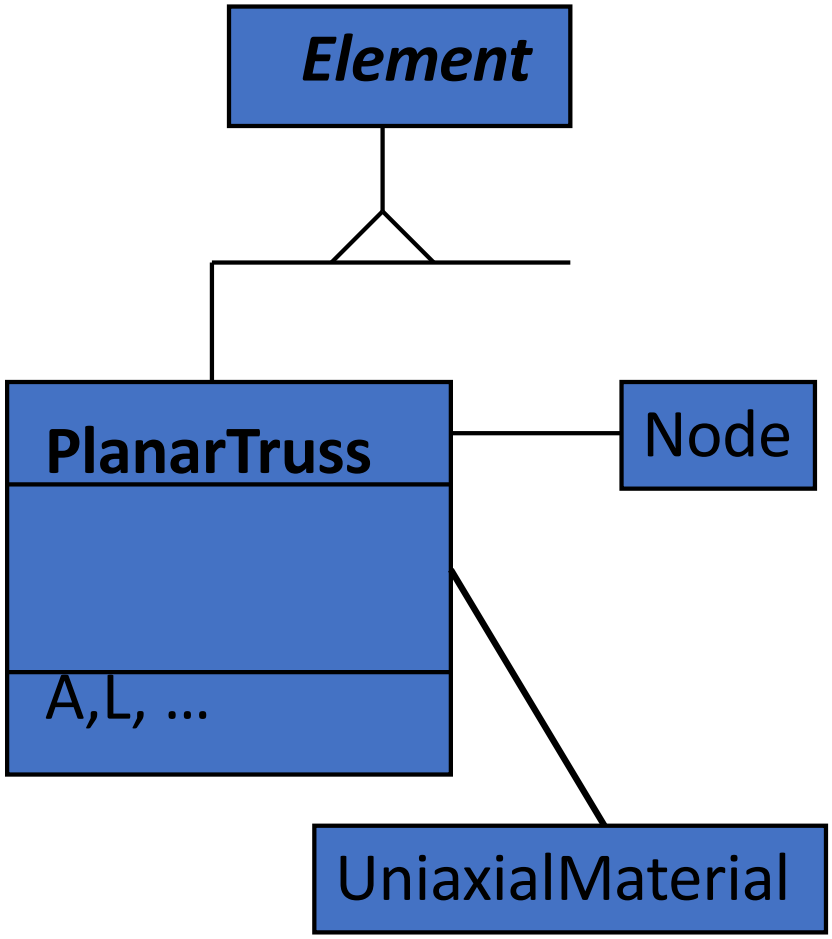
- to specify the steps taken to solve the nonlinear equation



Element Interface

```
class Element : public DomainComponent {
    public:
        Element(int tag, int classTag);
          virtual ~Element();
          virtual int getNumExternalNodes();
          virtual const ID &getExternalNodes()
          virtual Node **getNodePtrs;
          virtual int getNumDOF(void);
          virtual setDomain(Domain *theDomain);
          virtual int commitState(void);
          virtual int revertToStart();
          virtual int revertToLastCommit(void);
          virtual int update(void);
          virtual const Matrix &getTangentStiff(void);
          virtual const Matrix &getInitialStiff(void);
          virtual const Matrix &getDamp(void);
           virtual const Matrix &getMass(void);
          virtual void zeroLoad(void);
          virtual int addLoad(ElementLoad *theLoad, double loadFactor);
          virtual int addInertiaLoadToUnbalance(const Vector &accel);
          virtual const Vector &getResistingForce(void);
          virtual const Vector &getResistingForceIncInertia(void);
          virtual Response *setResponse(const char *argv, int argc, Information &info);
          virtual int getResponse(int responseID, Information &info);
          void Print(OPS_Stream &ops, int flag=0);
         // ++++++++++++ SOME Others
 };
```

New PlanarTruss Element



PlanarTruss.h

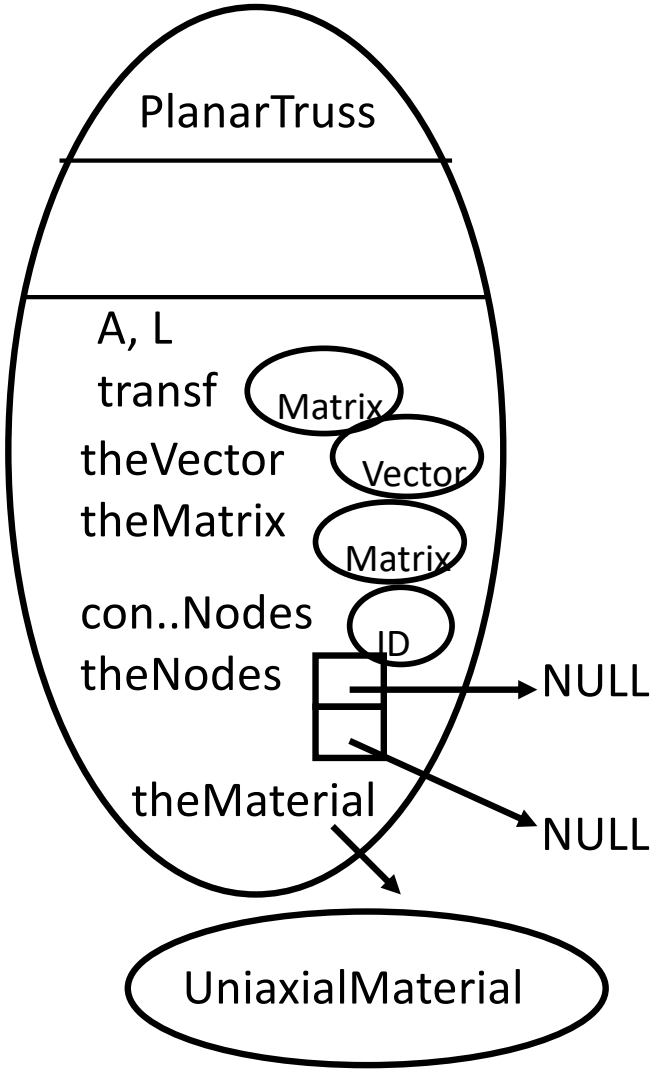
```
class PlanarTruss : public Element {
    public:
            PlanarTruss(int tag, int node1, int node2,
                    UniaxialMaterial &theMat,
                double A);
            ~PlanarTruss();

                ...
    private:
            double A, L;
             Matrix transf;
             Node *theNodes[2];
             UniaxialMaterial *theMaterial;
             ID connectedExternalNodes;
            Vector theVector;
            Matrix theMatrix;
};
```

Constructor

PlanarTruss.cpp

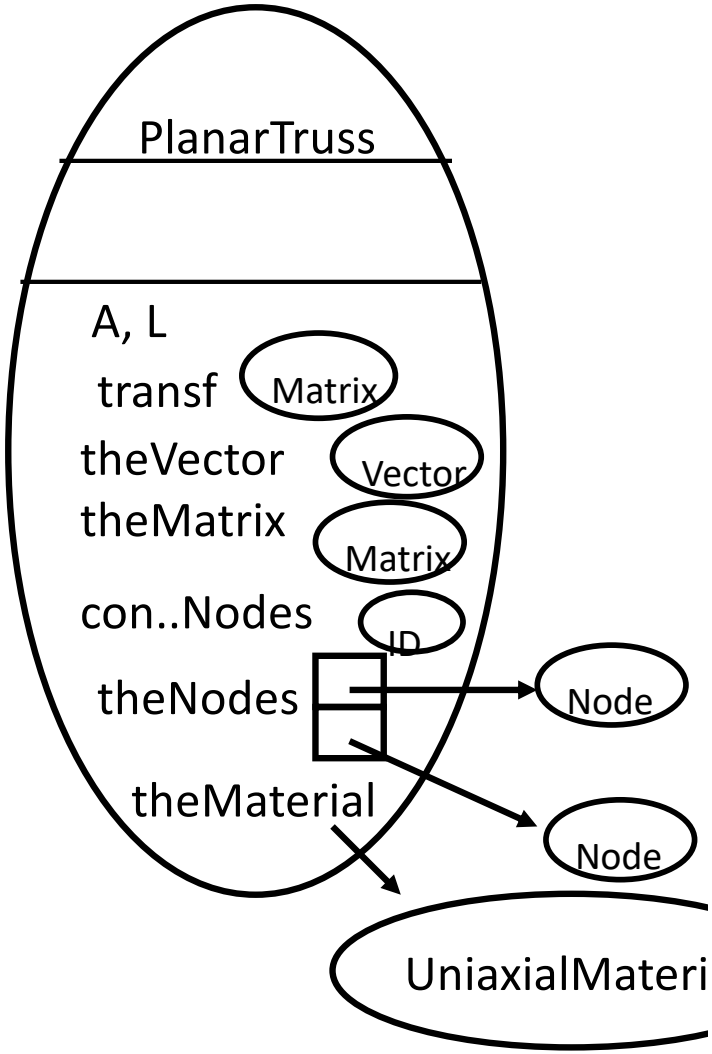
```
#define numNode 2
#define numDOF 4
PlanarTruss::PlanarTruss(int tag, int node1, int node2,
              UniaxialMaterial &theMat,
              double a)
:Element(tag, ELE_TAG_PlanarTruss),
 A(a), L(0), transf(1,4),
 connectedExternalNodes(numNode),
 theMatrix(numDOF,  numDOF),
theVector(numDOF)
{
        connectedExternalNodes(0) = node1;
        connectedExternalNodes(1) = node2;
        theMaterial = theMat.getCopy();
        theNodes[0] = 0;
        theNodes[1] = 0;
}
```



Destructor & setDomain

```
PlanarTruss::setDomain(Doman *theDomain)
{
  int node1 = connectedExternalNodes(0);
  int node2 = connectedExternalNodes(1);
  theNodes[0] = theDomain->getNode(node1);
  theNodes[1] = theDomain->getNode(node2);
  this->DomainComponent::setDomain(theDomain);
  const Vector &crd1 = end1Ptr->getCrds();
  const Vector &crd2 = end2Ptr->getCrds();
  double dx = crd2(0)-crd1(0);
  double dy = crd2(1)-crd1(1);
  L = sqrt(dx*dx + dy * dy);
  double cs = dx/L; double sn = dy/L;
  trans(0,0)=-cs; trans(0,1)=-sn;
  trans(0,2) = cs; trans(0,3)=sn;
  this->update();
}
```

```
PlanarTruss::~PlanarTruss()
{
    delete theMaterial;
}
```



Public Methods - some easy ones

```
int PlanarTruss::getNumNodes(void)
{
   return numNode;
}
```

```
int PlanarTruss::commitState(void)
{
    return theMaterial->commitState()
}
```

```
Node **PlanarTruss::getNodes(void)
{
    return theNodes;
}
```

Public Methods - more difficult!

```
const Matrix &PlanarTruss::getTangentStiff(void) {
    double E = theMaterial->getTangent();
    theMatrix = transf ^ transf;
    theMatrix *= A*E/L;
    return theMatrix;
}
```

```
const Matrix &PlanarTruss::getInitialStiff(void) {
    // one line needs to be changed from above
    // which one??? SRC/material/uniaxialMaterial/UniaxialMaterial.h
}
```

```
const Vector &PlanarTruss::getResistingForce(){
   double force = A*theMaterial->getStress();
   for (int I=0; I<4; I++)
       theVector(I) = transf(0,I)*force;
    return theVector;
}
```

Public Methods - most difficult!

```
int PlanarTruss::update(void)
{
   const Vector &disp1 = theNodes[0]->getTrialDisp();
   const Vector &disp2 = theNodes[1]->getTrialDisp();
   double dLength = 0.0;
   for (int i=0; i<2; i++)
       dLength -= (disp2(i)-disp1(i)) * trans (0,i);
  double strain = dLength / L;
  return theMaterial->setTrialStrain(strain);
}
```

```
void PlanarTruss::Print(OPS_Stream &out, int flag)
{
   out << “PlanarTruss  tag: “ << this->getTag() << endln;
   out << “resisting Force: “ << this->getResistingForce();
   theMaterial->Print(out, flag);
}
```

POLL (installed software)

HANDS ON EXERCISE

A Quote Before You Start:

- *The only way to learn a new programming language is by writing programs in it.* – Dennis Ritchie

Hands on Exercise – adding a PlanarTruss:

1. In OpenSees/SRC/element folder create a new folder planarTruss
2. Into this folder **COPY** the files NewElement.h, NewElement.cpp, CMakeListsNewElement.txt and OPS_NewElement.cpp from the OpenSees/SRC/element folder
3. In the **NEW** directory, Rename the files:
    1. NewElement.h to be PlanarTruss.h
    2. NewElement.cpp to be PlanarTruss.cpp
    3. OPS_NewElement.cpp to be OPS_PlanarTruss.cpp
    4. CMakeListsNewElement.txt to be CMakeLists.txt
4. Open CMakeLists.txt and do a global replace to change NewElement to be PlanarTruss
5. Open PlanarTruss.h and .cpp and do the same global replace for NewElement.
6. Open OPS_PlanarTruss and again start by doing the global replace. You will need to modify the code for your input line.
7. Open the CMakeLists.txt file in OpenSees/SRC/element and add a line to add the planarTruss directory
8. Open The file SRC/element/TclElementCommands.cpp and add the OPS_PlanarTruss function. Hint look at the OPS_Truss and repeat.
9. Compile OpenSees from the build directory. It should compile and OpenSees exe will be in build/bin folder
10. Make the necessary changes to PlanarTruss.h and .cpp files as indicated in slides. Fill in of course were needed.
11. Does it Work? .. You need to compile OpenSees again and test it (hint: use Example1.1.tcl in the OpenSees/EXAMPLES/ExampleScripts folder)