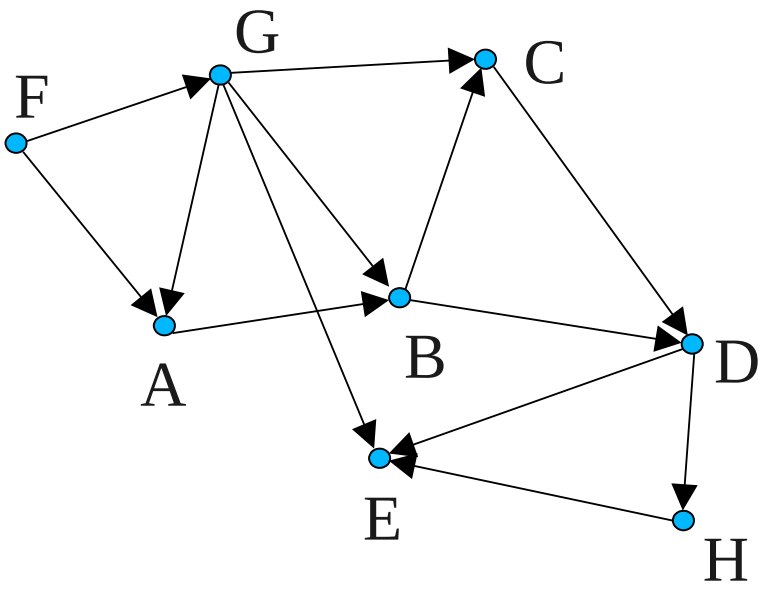
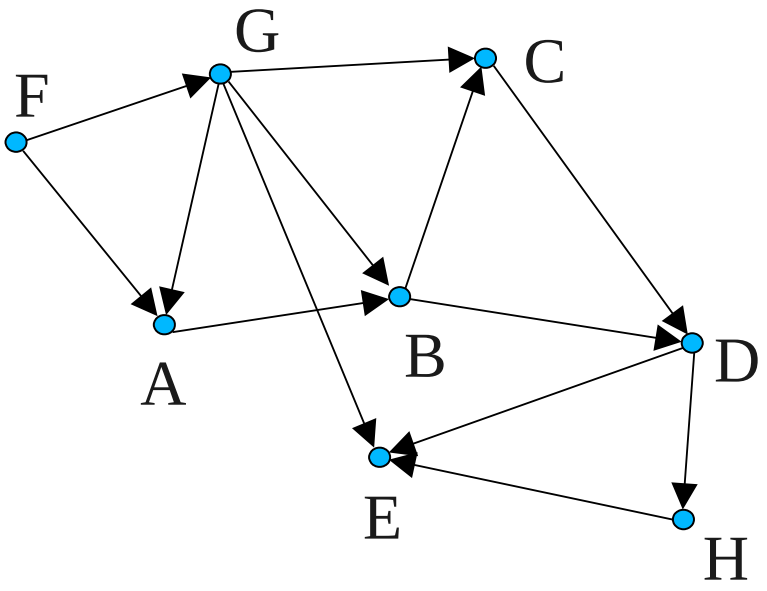
How to build a solution to topological sort

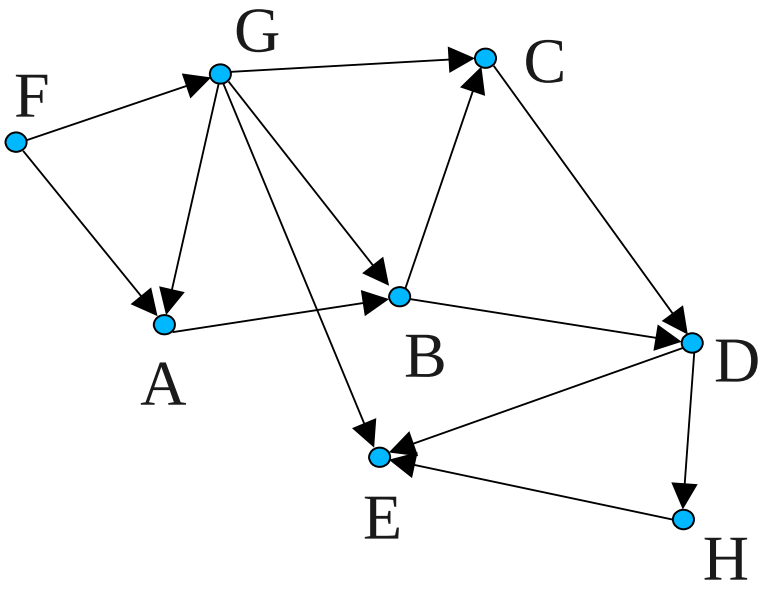


How to build a solution to topological sort



1. Identify objects

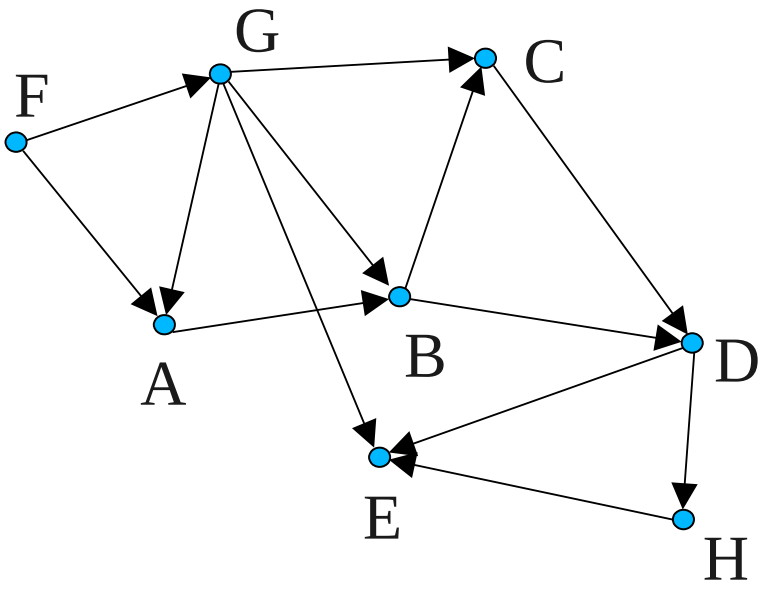
How to build a solution to topological sort



1. Identify objects

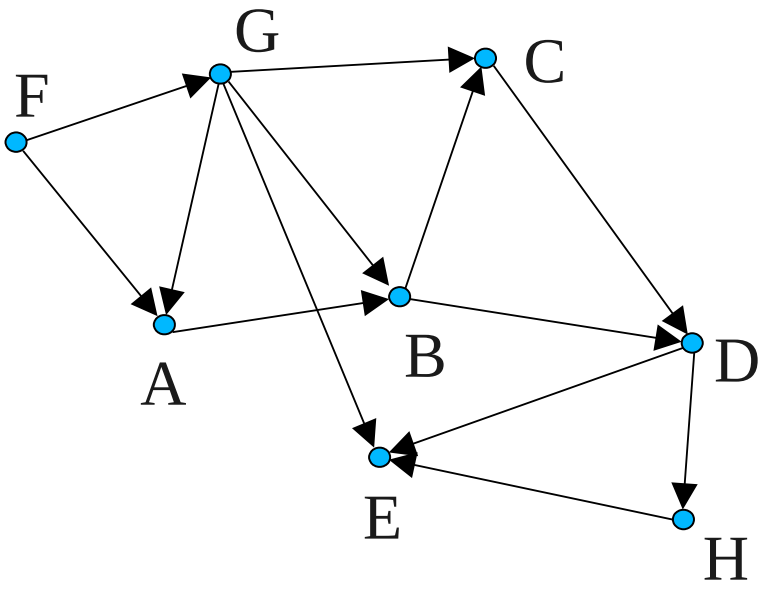
   The ● - call them Nodes

How to build a solution to topological sort



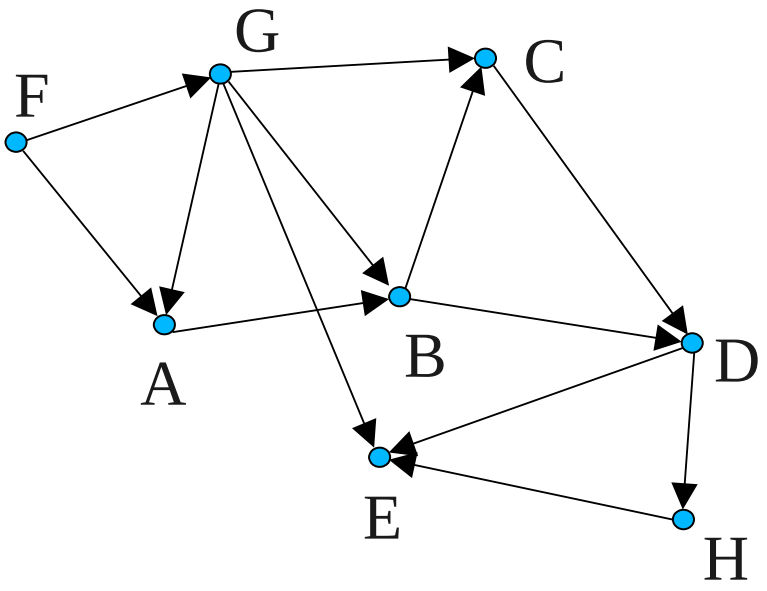
1. Identify objects
   - The ● - call them Nodes
   - The → - call then arcs

How to build a solution to topological sort



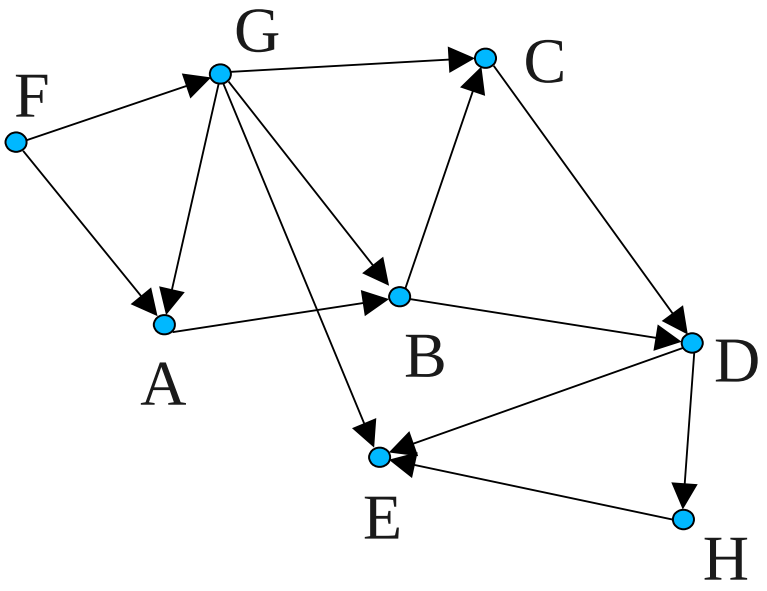
1. Identify objects
   - The ● - call them Nodes
   - The → - call then arcs
2. Identify object properties

How to build a solution to topological sort



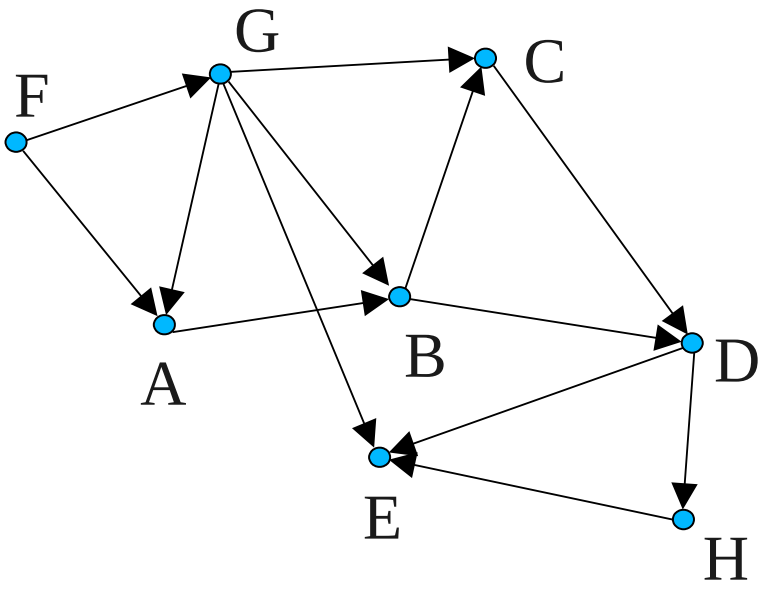
1. Identify objects
   - The ● - call them Nodes
   - The → - call then arcs
2. Identify object properties
   For Nodes:

How to build a solution to topological sort



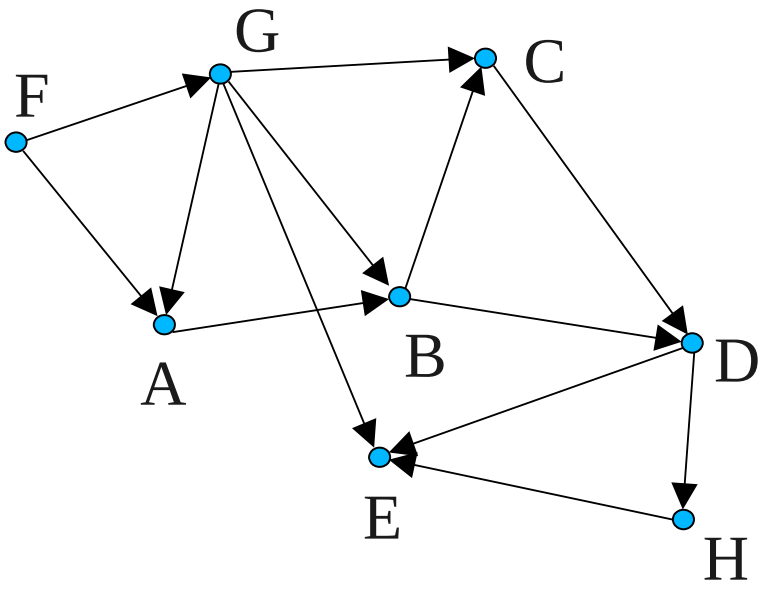
1. Identify objects
   - The ● - call them Nodes
   - The → - call then arcs
2. Identify object properties
   - For Nodes: list of dependent Nodes

How to build a solution to topological sort



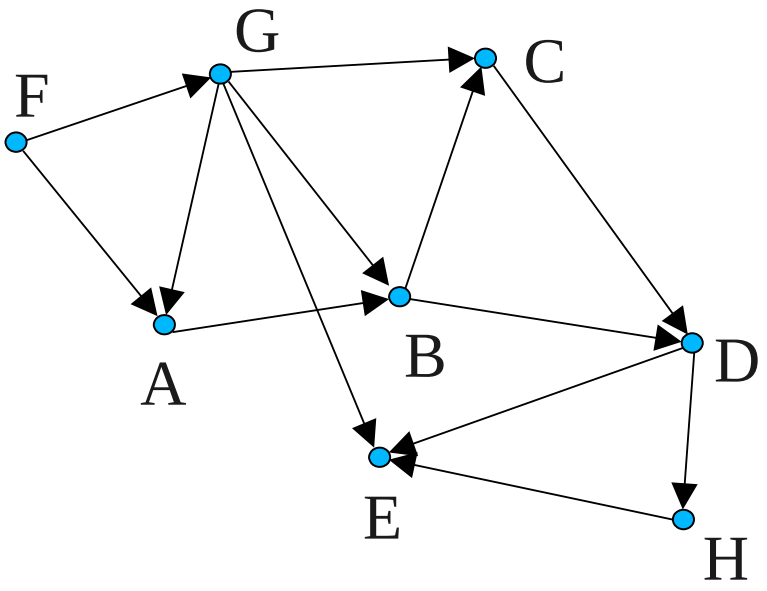
1. Identify objects
   - The ● - call them Nodes
   - The → - call then arcs
2. Identify object properties
   - For Nodes: list of dependent Nodes
     an identity

How to build a solution to topological sort



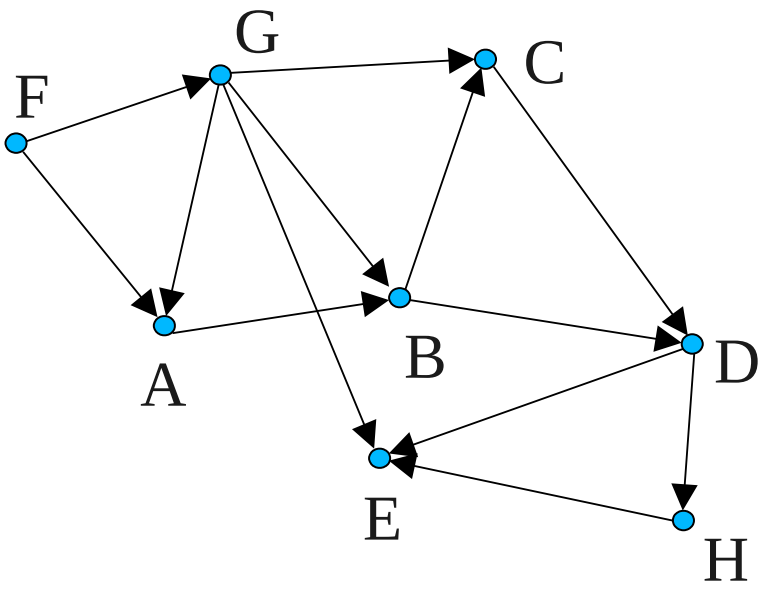
1. Identify objects
   - The ● - call them Nodes
   - The → - call then arcs
2. Identify object properties
   - For Nodes: list of dependent Nodes
     - an identity
     - a color ● ● ●

How to build a solution to topological sort



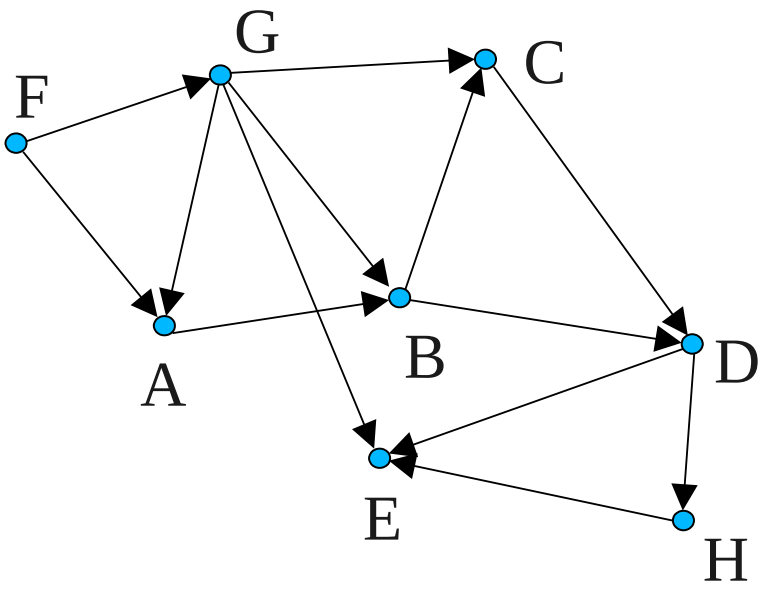
1. Identify objects
   - The ● - call them Nodes
   - The → - call then arcs
2. Identify object properties
   - For Nodes: list of dependent Nodes
     - an identity
     - a color ● ● ●
   - For arcs:

How to build a solution to topological sort



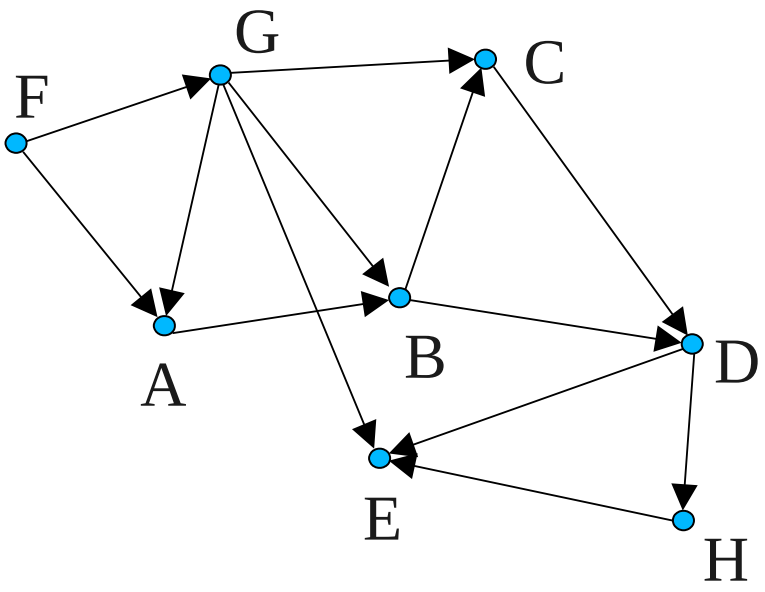
1. Identify objects
   - The ● - call them Nodes
   - The → - call then arcs
2. Identify object properties
   - For Nodes: list of dependent Nodes
     - an identity
     - a color ● ● ●
   - For arcs: two nodes, one depends on another

How to build a solution to topological sort



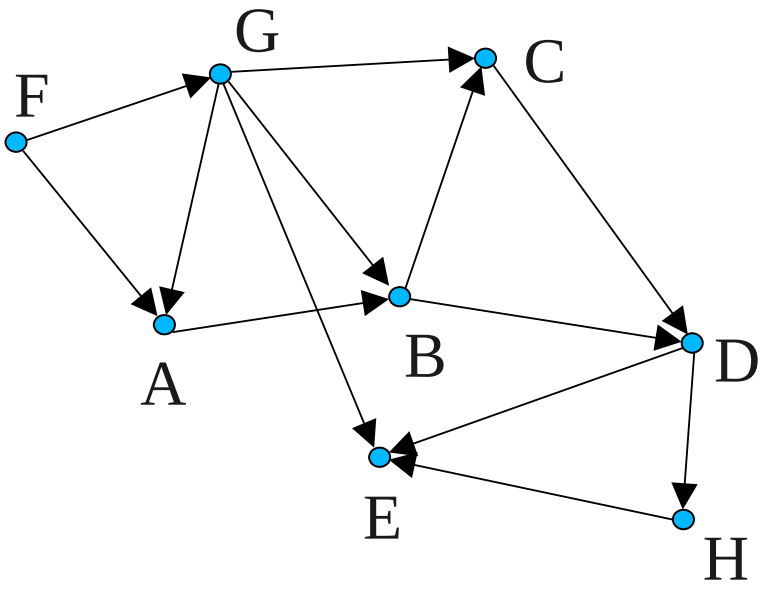
1. Identify objects
   - The ● - call them Nodes
   - The ➤ - call then arcs
2. Identify object properties
   - For Nodes: list of dependent Nodes
     - an identity
     - a color ● ● ●
   - For arcs: two nodes, one depends on another
3. Identify operations on objects

How to build a solution to topological sort



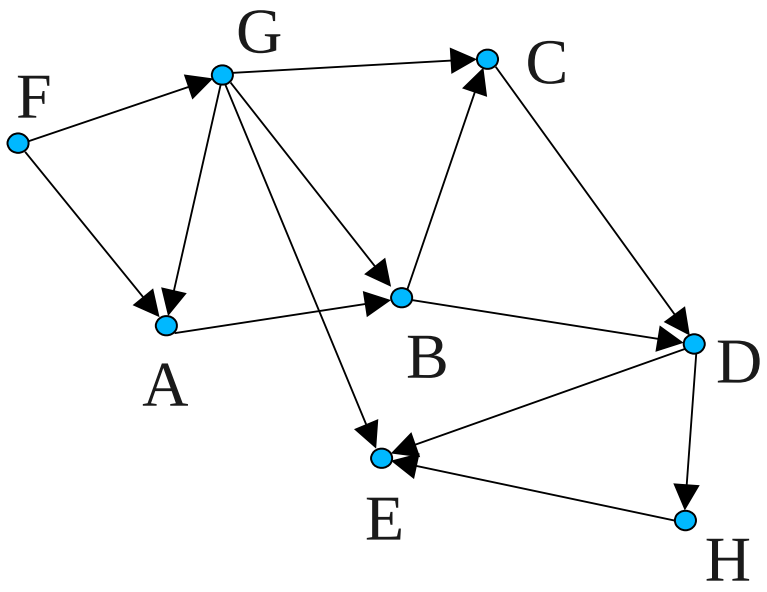
1. Identify objects
   - The ● - call them Nodes
   - The ➤ - call then arcs
2. Identify object properties
   - For Nodes: list of dependent Nodes
     - an identity
     - a color ● ● ●
   - For arcs: two nodes, one depends on another
3. Identify operations on objects
   - For Nodes:

How to build a solution to topological sort



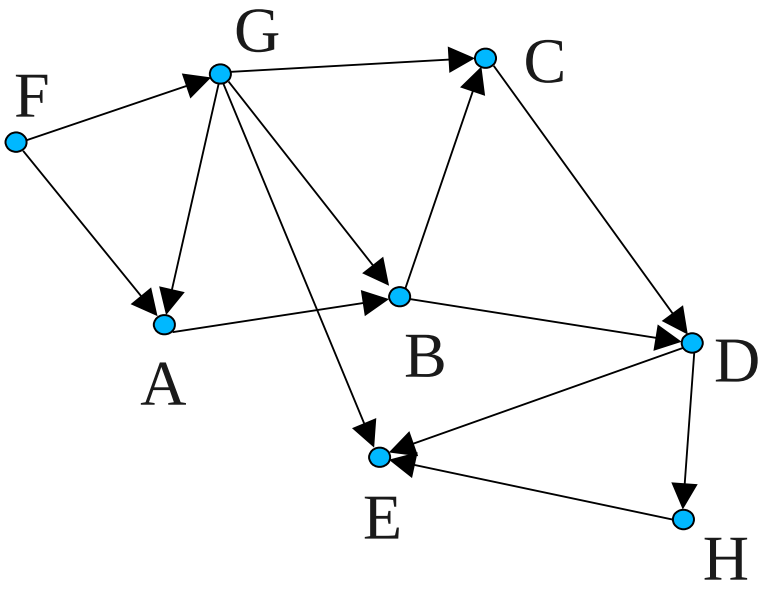
1. Identify objects
   - The ● - call them Nodes
   - The ➔ - call then arcs
2. Identify object properties
   - For Nodes: list of dependent Nodes
     - an identity
     - a color ● ● ●
   - For arcs: two nodes, one depends on another
3. Identify operations on objects
   - For Nodes: Output ID of all dependent Nodes then output ID of “this” Node

How to build a solution to topological sort



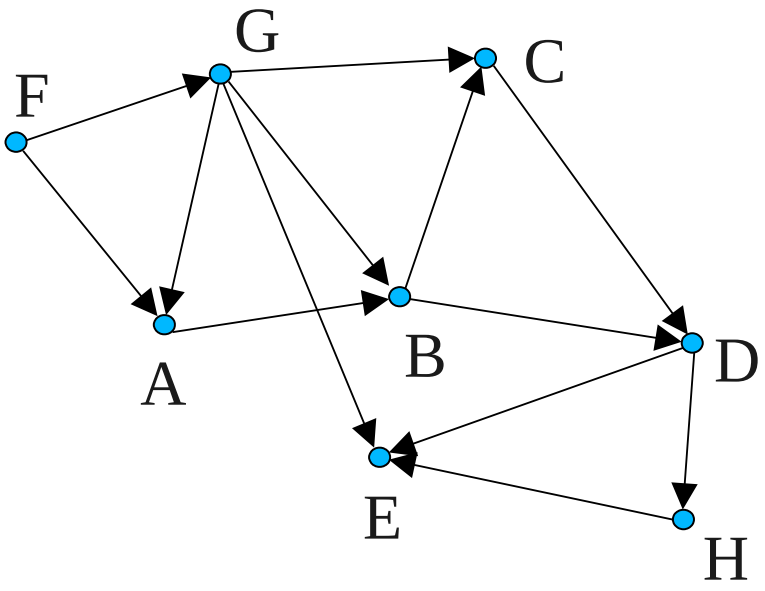
1. Identify objects
   - The ● - call them Nodes
   - The → - call then arcs
2. Identify object properties
   - For Nodes: list of dependent Nodes
     - an identity
     - a color ● ● ●
   - For arcs: two nodes, one depends on another
3. Identify operations on objects
   - For Nodes: Output ID of all dependent Nodes then output ID of “this” Node
   - For arcs:

How to build a solution to topological sort



1. Identify objects
   - The ● - call them Nodes
   - The → - call then arcs
2. Identify object properties
   - For Nodes: list of dependent Nodes
     - an identity
     - a color ● ● ●
   - For arcs: two nodes, one depends on another
3. Identify operations on objects
   - For Nodes: Output ID of all dependent Nodes then output ID of “this” Node
   - For arcs: ???

How to build a solution to topological sort



1. Identify objects
   - The ● - call them Nodes
   - The ➔ - call then arcs
2. Identify object properties
   - For Nodes: list of dependent Nodes
     - an identity
     - a color ● ● ●
   - For arcs: two nodes, one depends on another
3. Identify operations on objects
   - For Nodes: Output ID of all dependent Nodes then output ID of “this” Node
   - For arcs: ???
4. Consider a supervisor for reading data from file and running the sort

How to build a solution to topological sort

How do we want the `main` procedure to look?

```
int main (int argc, char **argv) {
   if (argc != 2) {
      cerr << “Usage: “ << argv[0] << “ <file>\n”;
      exit(0);
   }

   Supervisor *supervisor = new Supervisor(argv[1]);
   supervisor->getNodes();
   supervisor->setDependencies();
   supervisor->sort();
   supervisor->printSolution();
}
```

How to build a solution to topological sort

Build the class for making Node objects

How to build a solution to topological sort

Build the class for making Node objects

```
class Node {
```

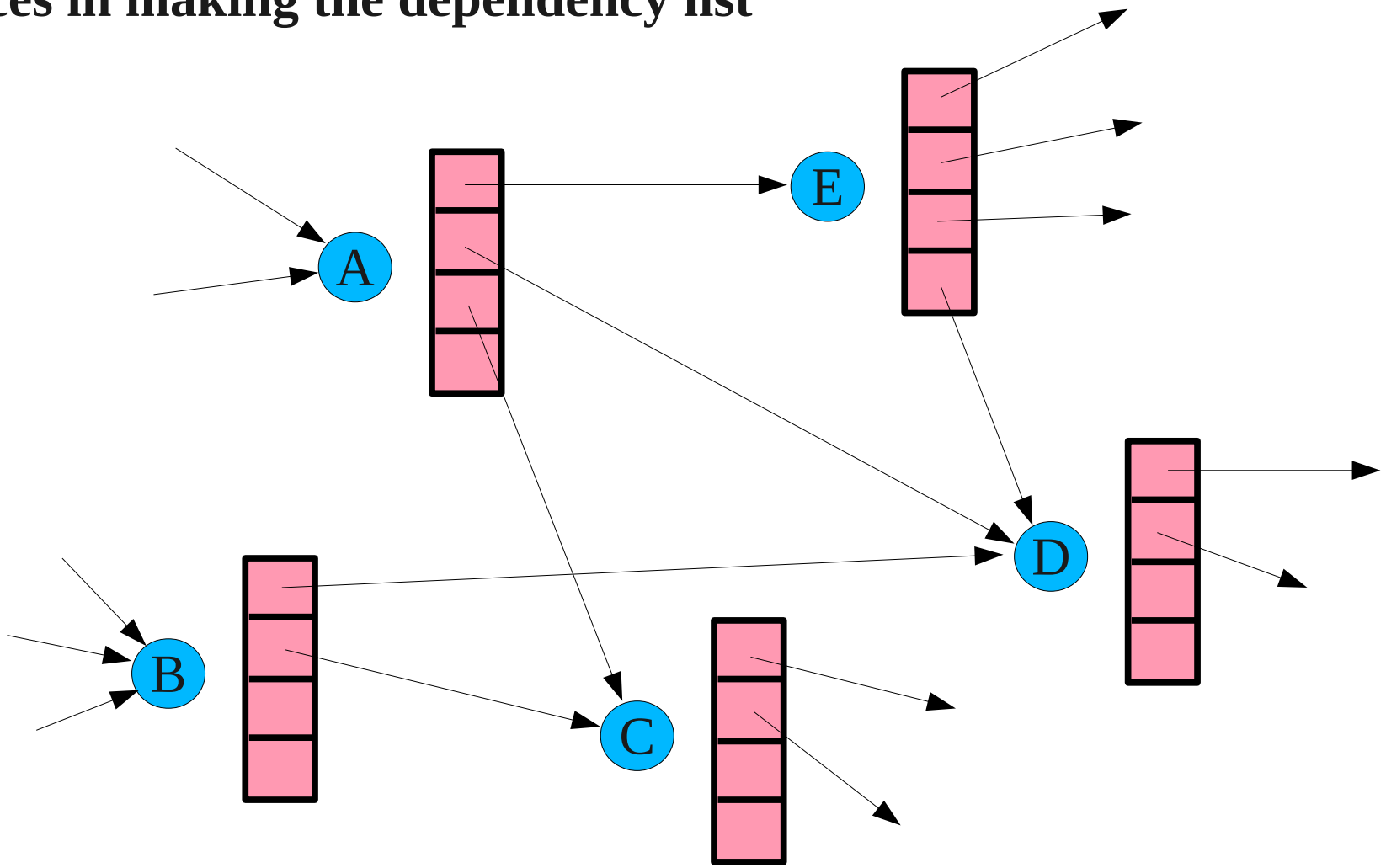
How to build a solution to topological sort

Build the class for making Node objects

```
class Node {
   char *id;            // Identity of the Node
```

How to build a solution to topological sort

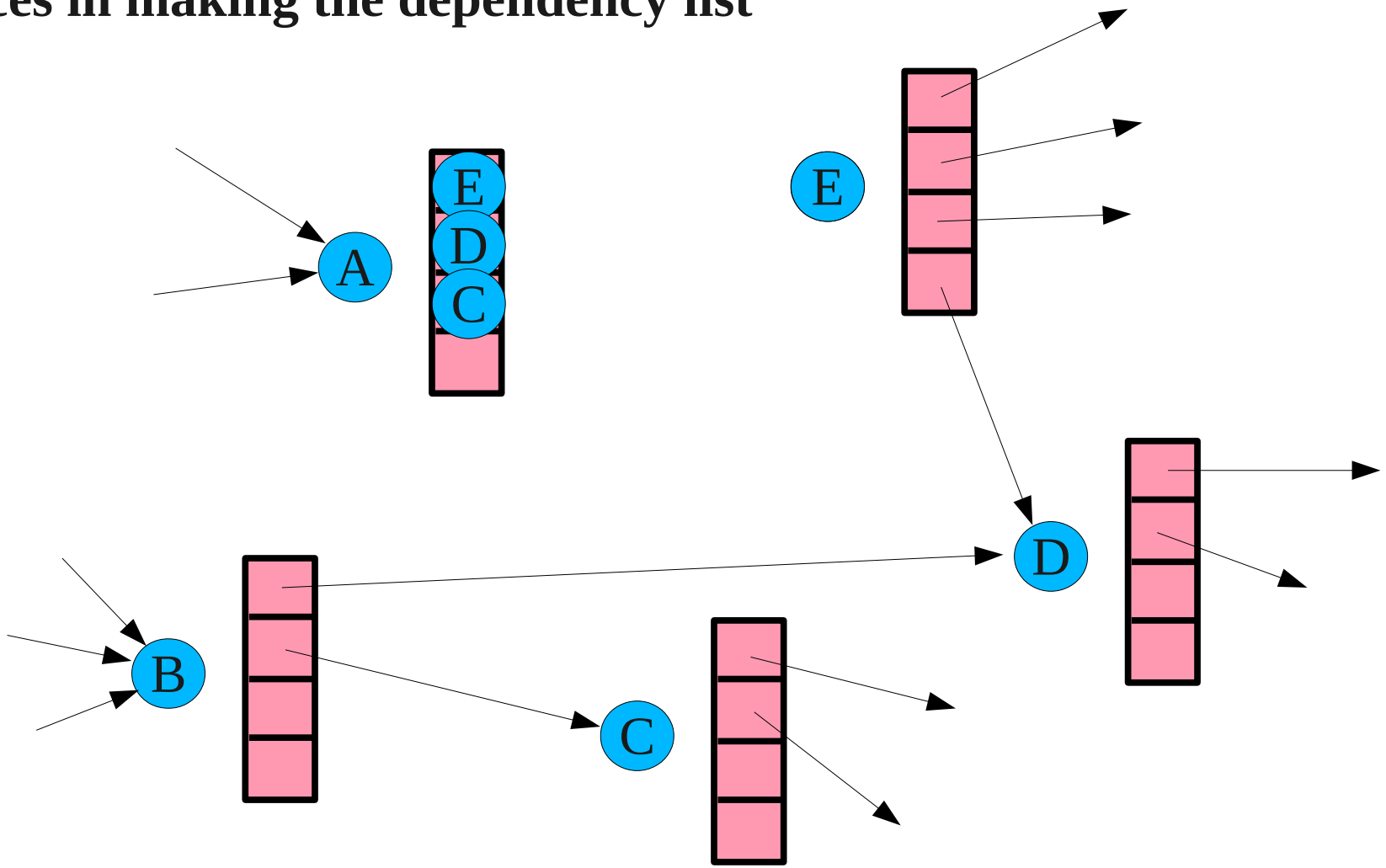
Choices in making the dependency list



Each object is unique, dependency list is a list of pointers to objects

How to build a solution to topological sort

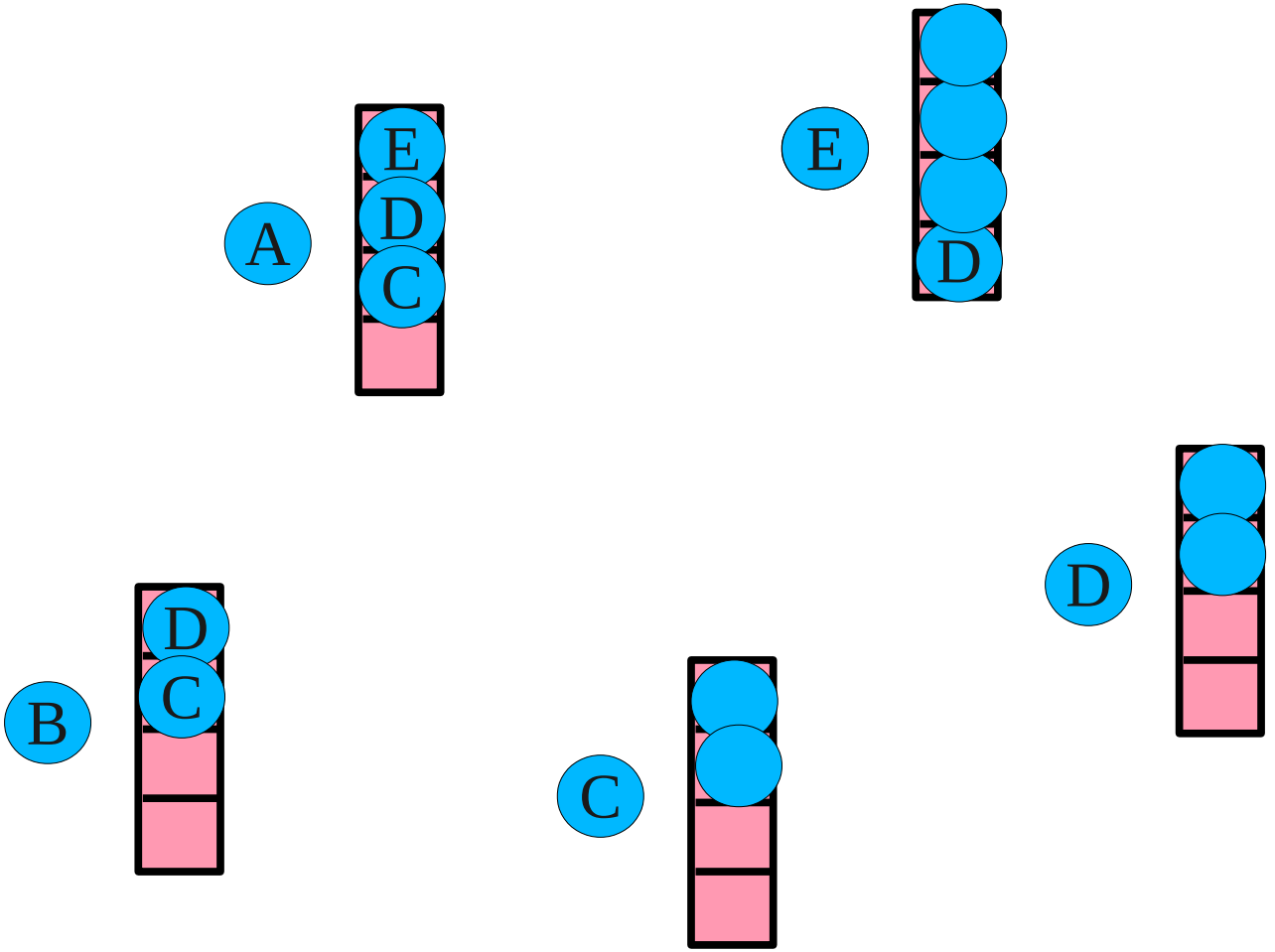
Choices in making the dependency list



Many copies of each object. Dependency list has actual objects.

How to build a solution to topological sort

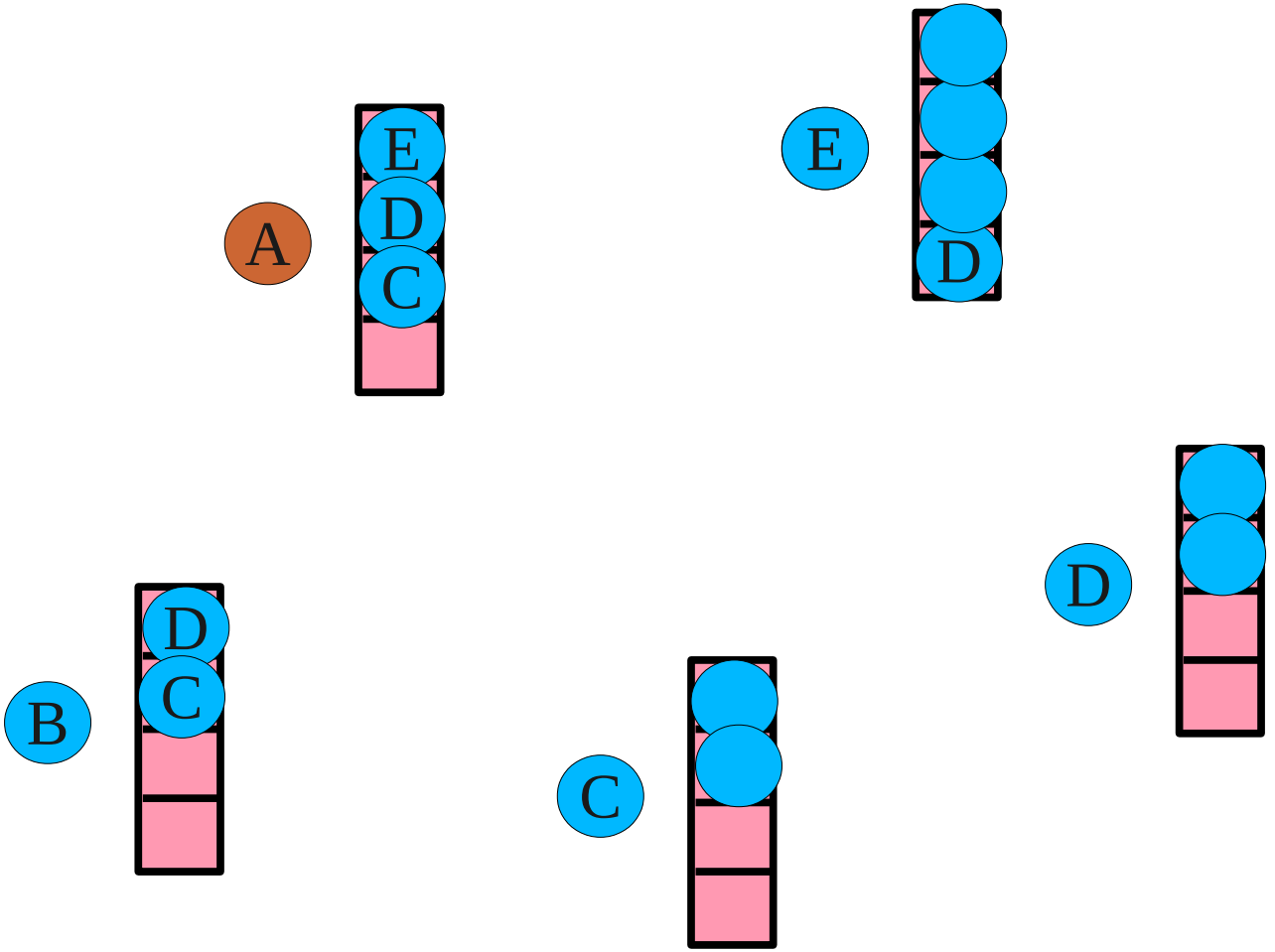
Choices in making the dependency list



Many copies of each object. Dependency list has actual objects.

How to build a solution to topological sort

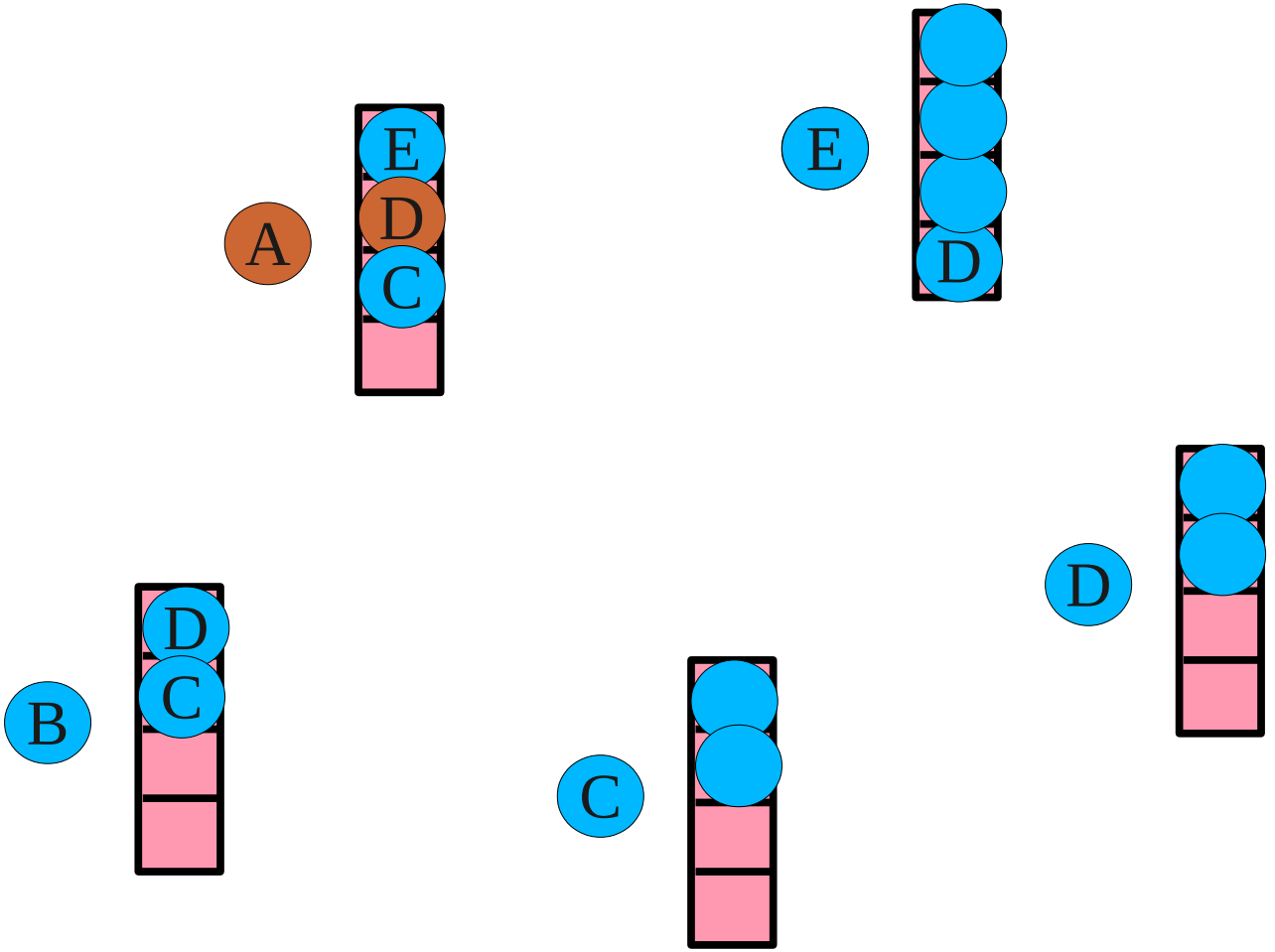
Choices in making the dependency list



How to keep track of visits to neighboring nodes?

How to build a solution to topological sort

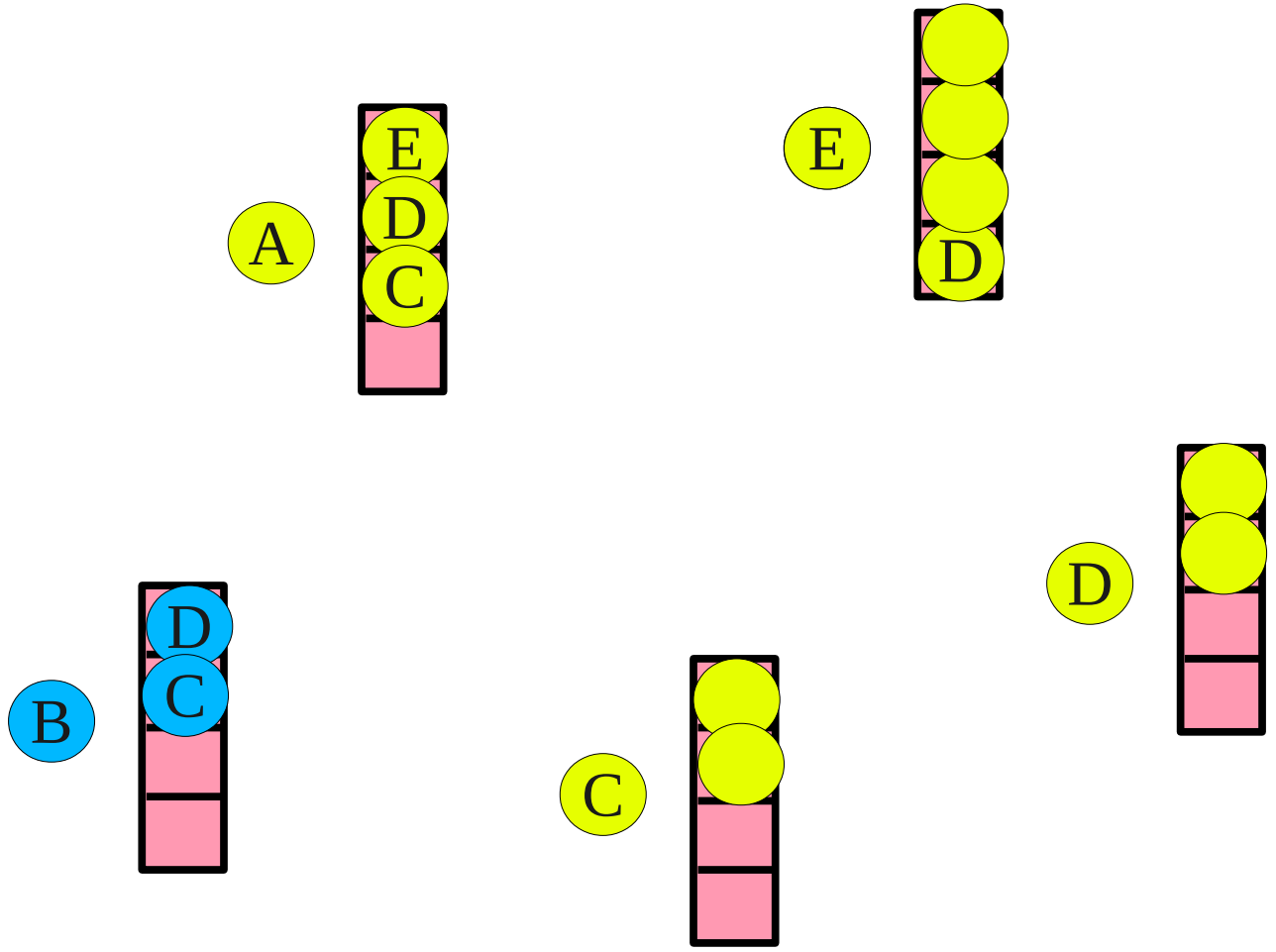
Choices in making the dependency list



How to keep track of visits to neighboring nodes?
Suppose A visits D

How to build a solution to topological sort

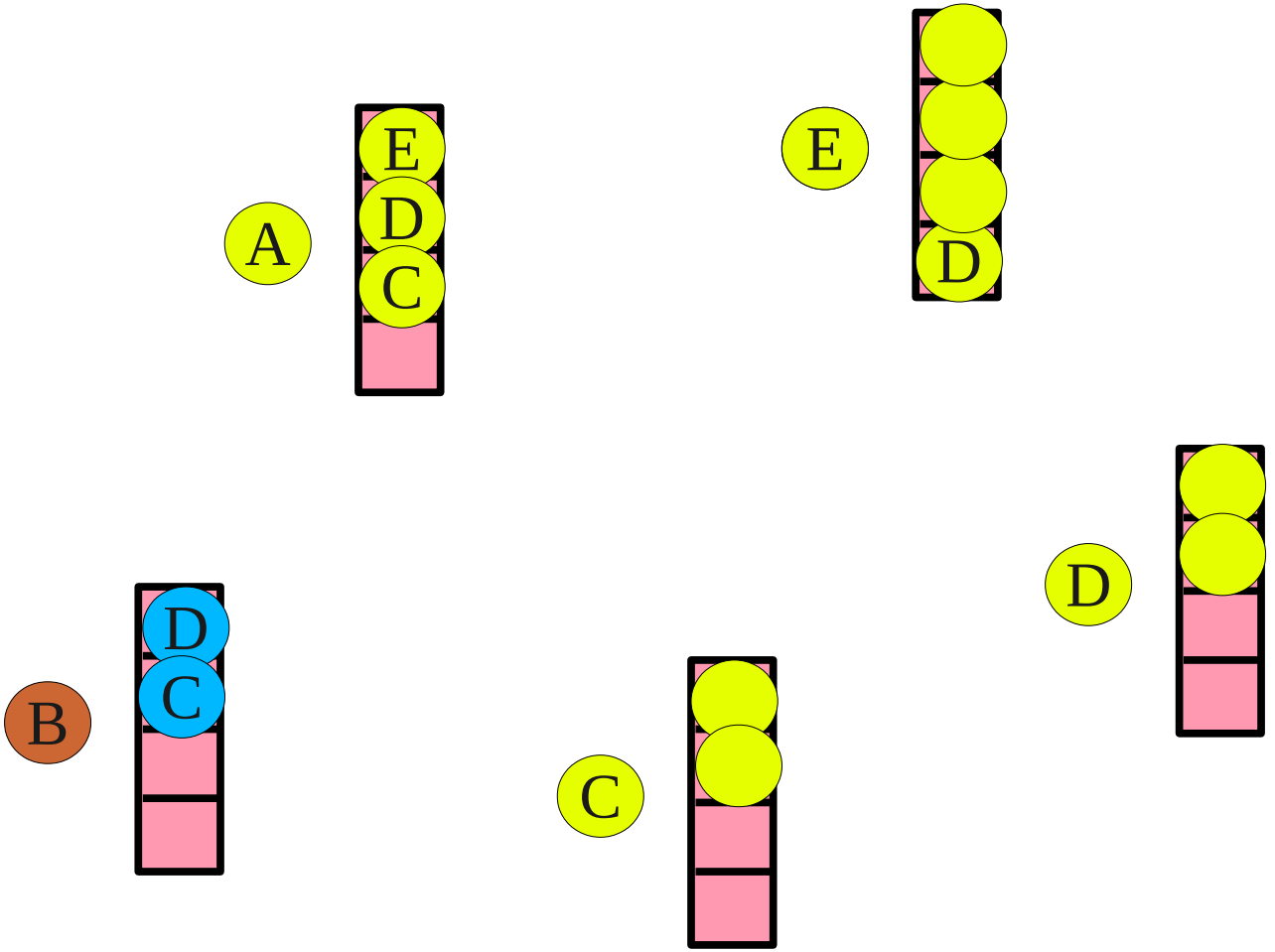
Choices in making the dependency list



How to keep track of visits to neighboring nodes?
Suppose A visits D ... later A is output after all its dependencies

How to build a solution to topological sort

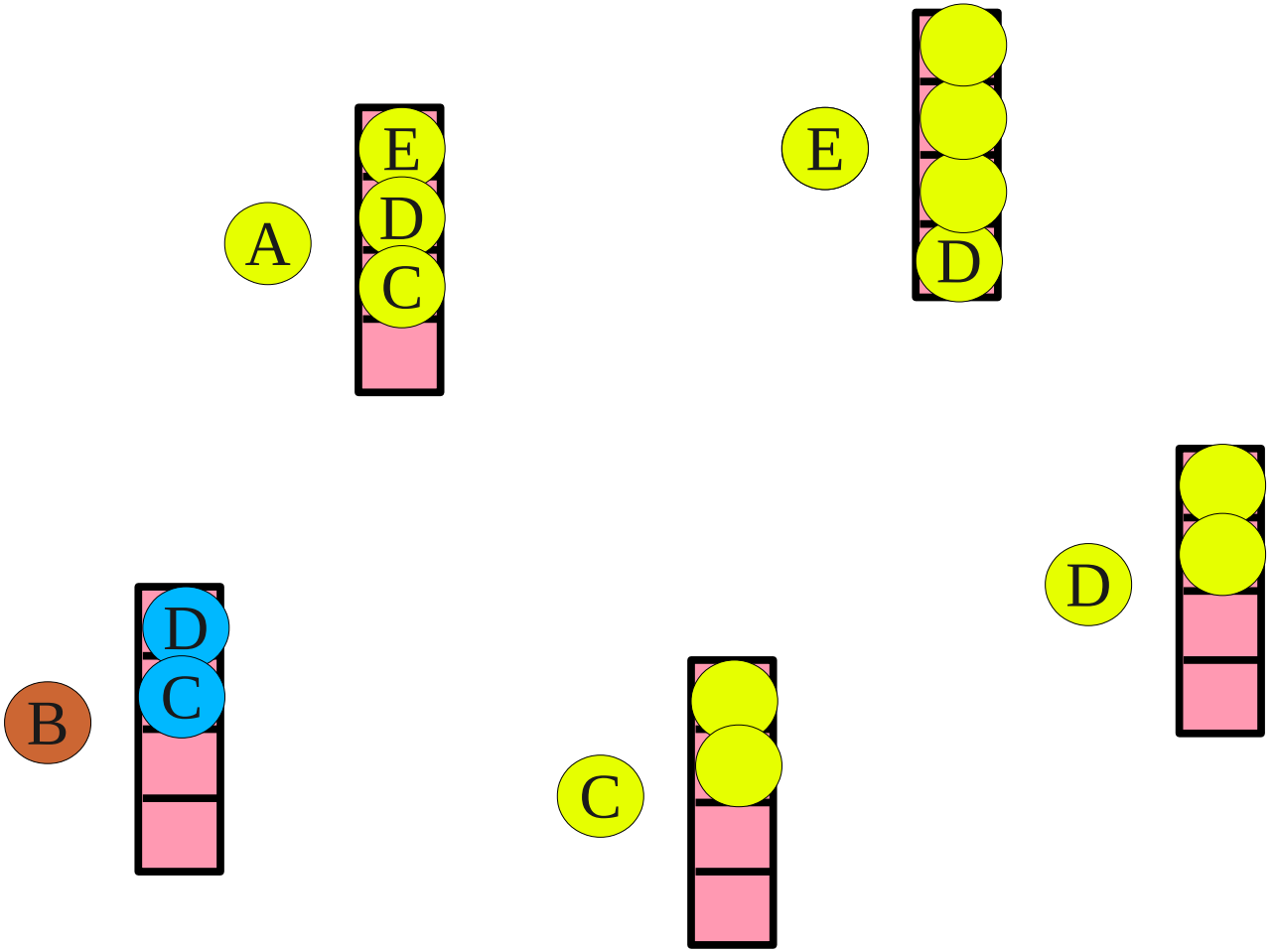
Choices in making the dependency list



How to keep track of visits to neighboring nodes?
Then B is visited for the first time

How to build a solution to topological sort

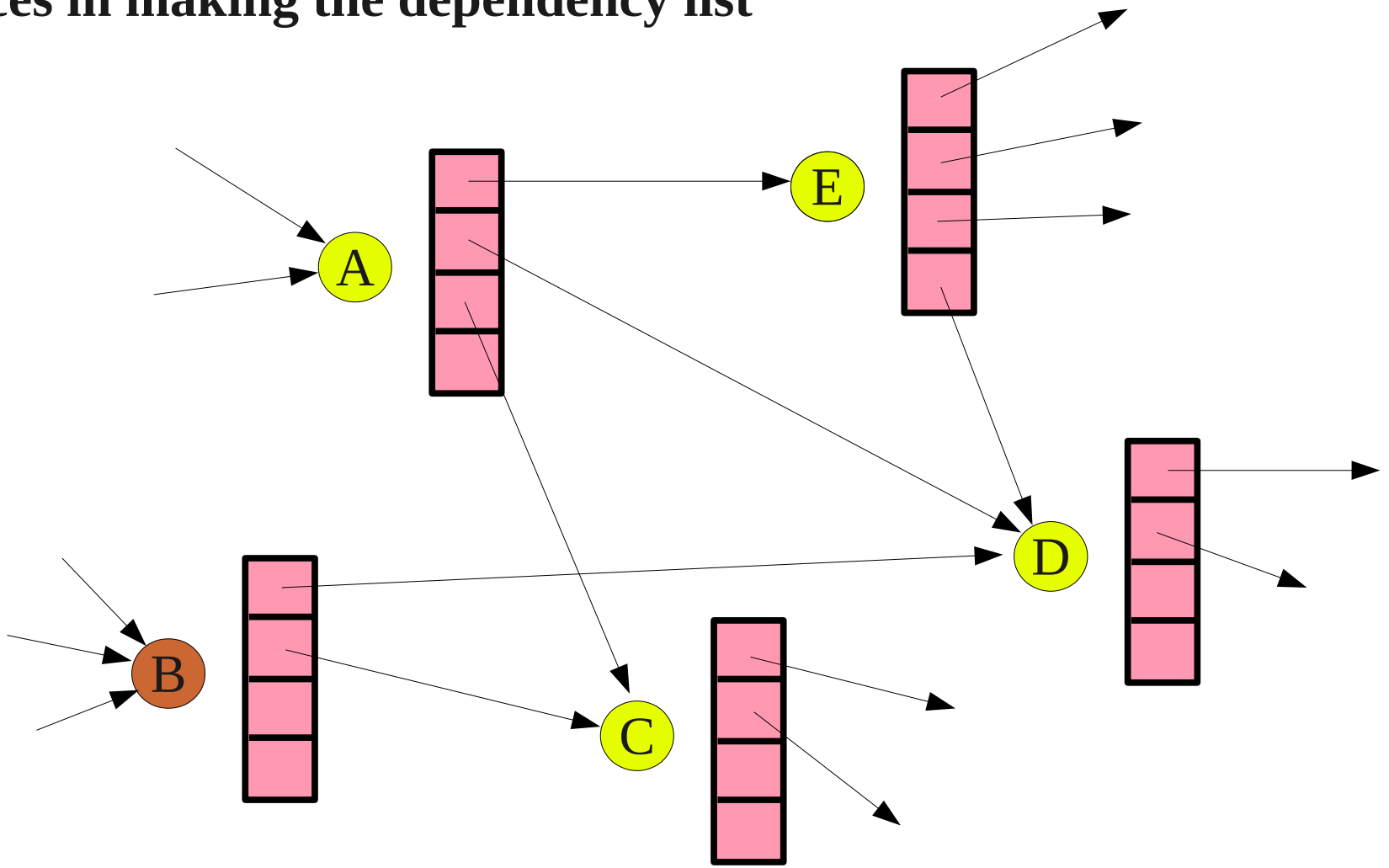
Choices in making the dependency list



How to keep track of visits to neighboring nodes?
Then B is visited for the first time – should avoid D, but D is blue

How to build a solution to topological sort

Choices in making the dependency list



How to keep track of visits to neighboring nodes?
If pointers are used, this is not a problem!

How to build a solution to topological sort

Build the class for making Node objects

```
class Node {
    char *id;           // Identity of the Node
    Node **depends;     // Array of pointers to Nodes
```

How to build a solution to topological sort

Build the class for making Node objects

```
class Node {
    char *id;           // Identity of the Node
    Node **depends;     // Array of pointers to Nodes
    int ndepends;       // Number of dependencies
```

How to build a solution to topological sort

Build the class for making Node objects

```
class Node {
    char *id;           // Identity of the Node
    Node **depends;     // Array of pointers to Nodes
    int ndepends;       // Number of dependencies
    Color color;        // Values=BLUE, BROWN, YELLOW
```

How to build a solution to topological sort

Build the class for making Node objects

```
class Node {
    char *id;              // Identity of the Node
    Node **depends;        // Array of pointers to Nodes
    int ndepends;          // Number of dependencies
    Color color;           // Values=BLUE, BROWN, YELLOW

    void init (char *id) {
        ndepends = 0;      // Initialize state
        this->id = new char[strlen(id)+1];
        strncpy(this->id, id, strlen(id));
        depends = NULL;
        color = BLUE;
    }
```

How to build a solution to topological sort

Build the class for making Node objects

```
class Node {
   char *id;            // Identity of the Node
   Node **depends;      // Array of pointers to Nodes
   int ndepends;        // Number of dependencies
   Color color;         // Values=BLUE, BROWN, YELLOW

   void init (char *id) {
      ndepends = 0;     // Initialize state
      this->id = new char[strlen(id)+1];
      strncpy(this->id, id, strlen(id));
      depends = NULL;
      color = BLUE;
   }

   ~Node () {           // free memory to be
      delete depends;   // grabbed later
   }
```

How to build a solution to topological sort

Build the class for making Node objects

```
class Node {
   char *id;            // Identity of the Node
   Node **depends;      // Array of pointers to Nodes
   int ndepends;        // Number of dependencies
   Color color;         // Values=BLUE, BROWN, YELLOW

   void topo () {
     // Outputs all dependents then itself
```

How to build a solution to topological sort

Build the class for making Node objects

```
class Node {
   char *id;            // Identity of the Node
   Node **depends;      // Array of pointers to Nodes
   int ndepends;        // Number of dependencies
   Color color;         // Values=BLUE, BROWN, YELLOW

   void topo () {
     // Outputs all dependents then itself
     // Hard to conceive of how to do this
     // so we call on our best friend, recursion,
     // to help out.
```

How to build a solution to topological sort

Build the class for making Node objects

```
class Node {
   char *id;            // Identity of the Node
   Node **depends;      // Array of pointers to Nodes
   int ndepends;        // Number of dependencies
   Color color;         // Values=BLUE, BROWN, YELLOW

   void topo () {
     // Outputs all dependents then itself
     // Hard to conceive of how to do this
     // so we call on our best friend, recursion,
     // to help out.
     // Assume everyone else has such a topo -
     // we can invoke the topos of all our direct
     // dependents, that insures they are all
     // output ahead of us (let them figure out how
     // to output themselves properly).  Then we
     // output ourselves.  What does this look like?
```

How to build a solution to topological sort

Build the class for making Node objects

```
class Node {
   char *id;            // Identity of the Node
   Node **depends;      // Array of pointers to Nodes
   int ndepends;        // Number of dependencies
   Color color;         // Values=BLUE, BROWN, YELLOW

   void topo () {
     // Outputs all dependents first
     for (int i=0 ; i < ndepends ; i++)
        depends[i]->topo();
     // Output this Node's identity
     cout << " " << id;
   }
```

How to build a solution to topological sort

Build the class for making Node objects

```
class Node {
   char *id;            // Identity of the Node
   Node **depends;      // Array of pointers to Nodes
   int ndepends;        // Number of dependencies
   Color color;         // Values=BLUE, BROWN, YELLOW

   void topo () {
     // Outputs all dependents first
     for (int i=0 ; i < ndepends ; i++)
        depends[i]->topo();
     // Output this Node's identity
     cout << " " << id;
   }
```

But what happens if someone calls our `topo()` when we are YELLOW?

How to build a solution to topological sort

Build the class for making Node objects

```
class Node {
   char *id;            // Identity of the Node
   Node **depends;      // Array of pointers to Nodes
   int ndepends;        // Number of dependencies
   Color color;         // Values=BLUE, BROWN, YELLOW

   void topo () {
     if (color == YELLOW) return;  // finished
     // Outputs all dependents first
     for (int i=0 ; i < ndepends ; i++)
        depends[i]->topo();
     // Output this Node's identity
     cout << " " << id;
     color = YELLOW;     // Now finished
   }
```

Set color variable to YELLOW after done, test upon entry for quick return

How to build a solution to topological sort

Build the class for making Node objects

```
class Node {
   char *id;            // Identity of the Node
   Node **depends;      // Array of pointers to Nodes
   int ndepends;        // Number of dependencies
   Color color;         // Values=BLUE, BROWN, YELLOW

   void topo () {
     if (color == YELLOW) return;  // finished
     // Outputs all dependents first
     for (int i=0 ; i < ndepends ; i++)
        depends[i]->topo();
     // Output this Node's identity
     cout << " " << id;
     color = YELLOW;     // Now finished
   }
```

But what if someone calls our topo() after we start and before we finish?

How to build a solution to topological sort

Build the class for making Node objects

```
class Node {
   char *id;            // Identity of the Node
   Node **depends;      // Array of pointers to Nodes
   int ndepends;        // Number of dependencies
   Color color;         // Values=BLUE, BROWN, YELLOW

   void topo () {
     if (color == YELLOW) return;  // finished
     if (color == BROWN) exit(0);
     color = BROWN;
     for (int i=0 ; i < ndepends ; i++)
        depends[i]->topo();
     cout << " " << id;
     color = YELLOW;     // Now finished
   }
```

Set `color` to BROWN just before loop and test for `color` being BROWN

How to build a solution to topological sort

Build the class for making Node objects

```
class Node {
    char *id;            // Identity of the Node
    Node **depends;      // Array of pointers to Nodes
    int ndepends;        // Number of dependencies
    Color color;         // Values=BLUE, BROWN, YELLOW

    void topo () {
      if (color == YELLOW) return;  // finished
      if (color == BROWN) exit(0);
      color = BROWN;
      for (int i=0 ; i < ndepends ; i++)
         depends[i]->topo();
      cout << " " << id;
      color = YELLOW;     // Now finished
    }
};
```

How to build a solution to topological sort

Build the class for making a supervisor (reads data from file and sorts)

```
class Supervisor {
```

How to build a solution to topological sort

Build the class for making a supervisor (reads data from file and sorts)

```
class Supervisor {
    Node *nodes;      // Will be an array of nodes
    int nnodes;       // Will contain the array size
```

How to build a solution to topological sort

Build the class for making a supervisor (reads data from file and sorts)

```
class Supervisor {
    Node *nodes;      // Will be an array of nodes
    int nnodes;       // Will contain the array size
    fstream fin;      // File handle
```

How to build a solution to topological sort

Build the class for making a supervisor (reads data from file and sorts)

```
class Supervisor {
   Node *nodes;     // Will be an array of nodes
   int nnodes;      // Will contain the array size
   fstream fin;     // File handle

   Supervisor (char *filename) {
      fin.open(filename, ios::in);
      if (fin.fail()) {
         cerr << "Cannot open " << filename <<"\n";
         exit(0);
      }
   }
```

How to build a solution to topological sort

Build the class for making a supervisor (reads data from file and sorts)

```
class Supervisor {
   Node *nodes;     // Will be an array of nodes
   int nnodes;      // Will contain the array size
   fstream fin;     // File handle

   Supervisor (char *filename) {
      fin.open(filename, ios::in);
      if (fin.fail()) {
         cerr << "Cannot open " << filename <<"\n";
         exit(0);
      }
   }

   ~Supervisor () {
      delete nodes;
   }
```

How to build a solution to topological sort

Build the class for making a supervisor (reads data from file and sorts)

```
class Supervisor {
    Node *nodes;      // Will be an array of nodes
    int nnodes;       // Will contain the array size
    fstream fin;      // File handle

    void getNodes () {  // Get space for Nodes
```

Should we use an array of `Nodes`?

How to build a solution to topological sort

Build the class for making a supervisor (reads data from file and sorts)

```
class Supervisor {
    Node *nodes;      // Will be an array of nodes
    int nnodes;       // Will contain the array size
    fstream fin;      // File handle

    void getNodes () {  // Get space for Nodes
```

Should we use an array of `Nodes`? If so, we will be unable to use any constructor other than the default (no argument) one.

How to build a solution to topological sort

Build the class for making a supervisor (reads data from file and sorts)

```
class Supervisor {
    Node *nodes;      // Will be an array of nodes
    int nnodes;       // Will contain the array size
    fstream fin;      // File handle

    void getNodes () {  // Get space for Nodes
```

Should we use an array of `Nodes`? If so, we will be unable to use any constructor other than the default (no argument) one. Instead, we use the method

```
void init (char *);
```

that was defined in class `Node` earlier.

How to build a solution to topological sort

Build the class for making a supervisor (reads data from file and sorts)

```
class Supervisor {
   Node *nodes;     // Will be an array of nodes
   int nnodes;      // Will contain the array size
   fstream fin;     // File handle

   void getNodes () {  // Get space for Nodes
      // Count number of Node objects
      // Create the array of Nodes
      // Set the Node identities
```

How to build a solution to topological sort

Build the class for making a supervisor (reads data from file and sorts)

```
class Supervisor {
   Node *nodes;     // Will be an array of nodes
   int nnodes;      // Will contain the array size
   fstream fin;     // File handle

   void getNodes () {  // Get space for Nodes
      // Count number of Node objects
      char tok[1024];
      for (nnodes=0 ; fin >> tok ; nnodes++)
         if (tok[0] == '-') break;
      // Create the array of Nodes
      // Set the Node identities
```

How to build a solution to topological sort

Build the class for making a supervisor (reads data from file and sorts)

```
class Supervisor {
   Node *nodes;     // Will be an array of nodes
   int nnodes;      // Will contain the array size
   fstream fin;     // File handle

   void getNodes () {  // Get space for Nodes
      // Count number of Node objects
      char tok[1024];
      for (nnodes=0 ; fin >> tok ; nnodes++)
         if (tok[0] == '-') break;
      // Create the array of Nodes
      Nodes = new Node[nnodes];
      // Set the Node identities
```

How to build a solution to topological sort

Build the class for making a supervisor (reads data from file and sorts)

```
class Supervisor {
   Node *nodes;     // Will be an array of nodes
   int nnodes;      // Will contain the array size
   fstream fin;     // File handle

   void getNodes () {  // Get space for Nodes
      // Count number of Node objects
      char tok[1024];
      for (nnodes=0 ; fin >> tok ; nnodes++)
         if (tok[0] == '-') break;
      // Create the array of Nodes
      Nodes = new Node[nnodes];
      // Set the Node identities
      fin.seekg(0, ios::beg);
      for (int i=0; fin>>tok && tok[0] != '-'; i++)
         nodes[i].init(tok);
   }
```

How to build a solution to topological sort

Build the class for making a supervisor (reads data from file and sorts)

```
class Supervisor {
    Node *nodes;      // Will be an array of nodes
    int nnodes;       // Will contain the array size
    fstream fin;      // File handle

    void setDependency (int i) { // set depends list
                                 // for Node i
```

How to build a solution to topological sort

Build the class for making a supervisor (reads data from file and sorts)

```
class Supervisor {
   Node *nodes;      // Will be an array of nodes
   int nnodes;       // Will contain the array size
   fstream fin;      // File handle

   void setDependency (int i) {
```

Observe we have to create the dependency list for the i^{th} `Node` here in the supervisor. This means there will have to be a method added to class `Node` which allows the `Supervisor` to "hand off" the newly created dependency list. In class `Node` add:

```
void setdeps(Node **deps, int ndeps) {
   ndepends = ndeps;
   depends = deps;
}
```

How to build a solution to topological sort

Build the class for making a supervisor (reads data from file and sorts)

```
class Supervisor {
   Node *nodes;      // Will be an array of nodes
   int nnodes;       // Will contain the array size
   fstream fin;      // File handle

   void setDependency (int i) {
```

Other considerations: Use a variable

```
char tok[1024];
```

to receive tokens from the file.

Use

```
size_t mark = fin.tellg();
fin.seekg(mark, ios::beg);
```

to remember the file cursor at the beginning of a line and then to return to that point later.

How to build a solution to topological sort

Build the class for making a supervisor (reads data from file and sorts)

```
class Supervisor {
   Node *nodes;     // Will be an array of nodes
   int nnodes;      // Will contain the array size
   fstream fin;     // File handle

   void setDependency (int i) {
     // Save the file cursor (at beginning of line)
     // Count the number of dependencies on the line
     // Make dependency list: array of Node pointers
     // Cursor set to beginning of line
     // Reread line – set links – skip bogus links
     // Hand depends array off to the ith Node
```

How to build a solution to topological sort

Build the class for making a supervisor (reads data from file and sorts)

```
class Supervisor {
   Node *nodes;     // Will be an array of nodes
   int nnodes;      // Will contain the array size
   fstream fin;     // File handle

   void setDependency (int i) {
     // Save the file cursor (at beginning of line)
     size_t mark = fin.tellg();
     // Count the number of dependencies on the line
     // Make dependency list: array of Node pointers
     // Cursor set to beginning of line
     // Reread line – set links – skip bogus links
     // Hand depends array off to the ith Node
```

How to build a solution to topological sort

Build the class for making a supervisor (reads data from file and sorts)

```
class Supervisor {
   Node *nodes;     // Will be an array of nodes
   int nnodes;      // Will contain the array size
   fstream fin;     // File handle

   void setDependency (int i) {
     // Save the file cursor (at beginning of line)
     size_t mark = fin.tellg();
     // Count the number of dependencies on the line
     for (ndeps=0; fin>>tok && tok[0]!='-'; ndeps++);
     // Make dependency list: array of Node pointers
     // Cursor set to beginning of line
     // Reread line – set links – skip bogus links
     // Hand depends array off to the ith Node
```

How to build a solution to topological sort

Build the class for making a supervisor (reads data from file and sorts)

```
class Supervisor {
   Node *nodes;     // Will be an array of nodes
   int nnodes;      // Will contain the array size
   fstream fin;     // File handle

   void setDependency (int i) {
     // Save the file cursor (at beginning of line)
     size_t mark = fin.tellg();
     // Count the number of dependencies on the line
     for (ndeps=0; fin>>tok && tok[0]!='-'; ndeps++);
     // Make dependency list: array of Node pointers
     Node **deps = new Node*[ndeps];
     // Cursor set to beginning of line
     // Reread line – set links – skip bogus links
     // Hand depends array off to the ith Node
```

How to build a solution to topological sort

Build the class for making a supervisor (reads data from file and sorts)

```
class Supervisor {
    Node *nodes;      // Will be an array of nodes
    int nnodes;       // Will contain the array size
    fstream fin;      // File handle

    void setDependency (int i) {
      // Save the file cursor (at beginning of line)
      size_t mark = fin.tellg();
      // Count the number of dependencies on the line
      for (ndeps=0; fin>>tok && tok[0]!='-'; ndeps++);
      // Make dependency list: array of Node pointers
      Node **deps = new Node*[ndeps];
      // Cursor set to beginning of line
      fin.seekg(mark, ios::beg);
      // Reread line – set links – skip bogus links
      // Hand depends array off to the ith Node
```

How to build a solution to topological sort

Build the class for making a supervisor (reads data from file and sorts)

```
class Supervisor {
   Node *nodes;     // Will be an array of nodes
   int nnodes;      // Will contain the array size
   fstream fin;     // File handle

   void setDependency (int i) {
     size_t mark = fin.tellg();
     for (ndeps=0; fin>>tok && tok[0]!='-'; ndeps++);
     Node **deps = new Node*[ndeps];
     fin.seekg(mark, ios::beg);
     // Reread line – set links – skip bogus
     for (ndeps=0 ; fin>>tok && tok[0] != '-' ; ) {
       int n=atoi(tok);
       if (0<=n && n<nnodes) deps[ndeps++]=&nodes[n];
     }
     // Hand depends array off to the ith Node
```

How to build a solution to topological sort

Build the class for making a supervisor (reads data from file and sorts)

```
class Supervisor {
   Node *nodes;     // Will be an array of nodes
   int nnodes;      // Will contain the array size
   fstream fin;     // File handle

   void setDependency (int i) {
     size_t mark = fin.tellg();
     for (ndeps=0; fin>>tok && tok[0]!='-'; ndeps++);
     Node **deps = new Node*[ndeps];
     fin.seekg(mark, ios::beg);
     for (ndeps=0 ; fin>>tok && tok[0] != '-' ; ) {
       int n=atoi(tok);
       if (0<=n && n<nnodes) deps[ndeps++]=&nodes[n];
     }
     // Hand depends array off to the ith Node
     nodes[i].setdeps(deps, ndeps);
   }
```

How to build a solution to topological sort

Build the class for making a supervisor (reads data from file and sorts)

```
class Supervisor {
   Node *nodes;     // Will be an array of nodes
   int nnodes;      // Will contain the array size
   fstream fin;     // File handle

   void setDependency (int i) {
     size_t mark = fin.tellg();
     for (ndeps=0; fin>>tok && tok[0]!='-'; ndeps++);
     Node **deps = new Node*[ndeps];
     fin.seekg(mark, ios::beg);
     for (ndeps=0 ; fin>>tok && tok[0] != '-' ; ) {
       int n=atoi(tok);
       if (0<=n && n<nnodes) deps[ndeps++]=&nodes[n];
     }
     nodes[i].setdeps(deps, ndeps);
   }
```

How to build a solution to topological sort

Build the class for making a supervisor (reads data from file and sorts)

```
class Supervisor {
    Node *nodes;      // Will be an array of nodes
    int nnodes;       // Will contain the array size
    fstream fin;      // File handle

    void setDependencies () {
        for (int i=0 ; i < nnodes ; i++)
        setDependency(i);
    }

    void sort () {
       for (int i=0 ; i < nnodes ; i++)
          nodes[i].topo();
       cout << "\n";
    }
};
```