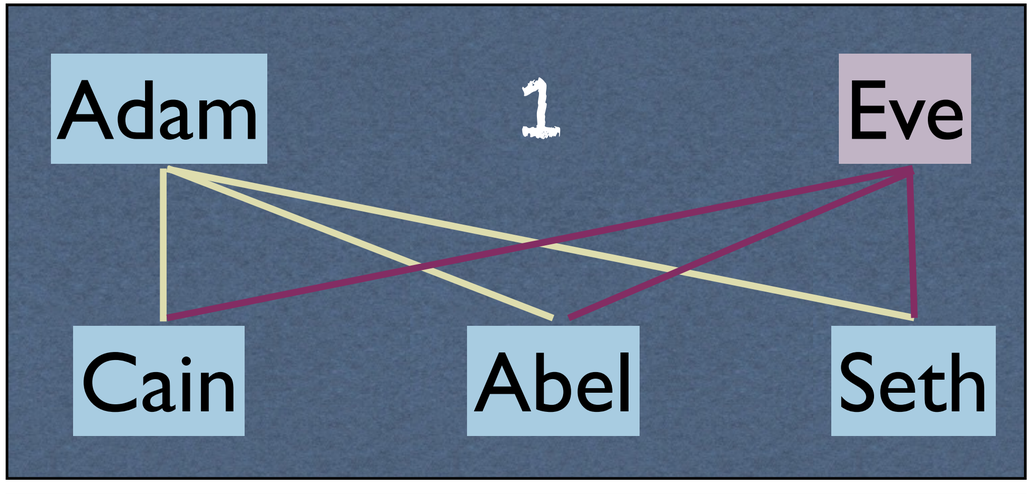
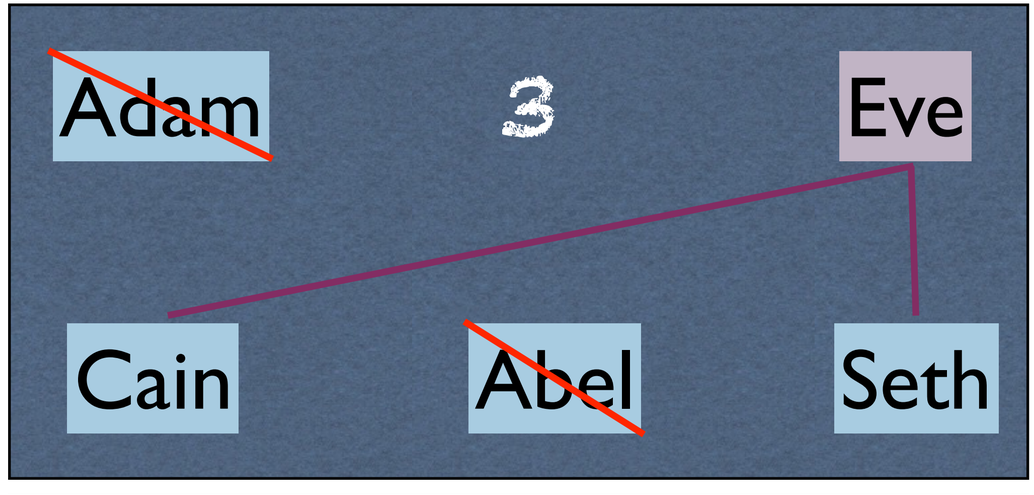
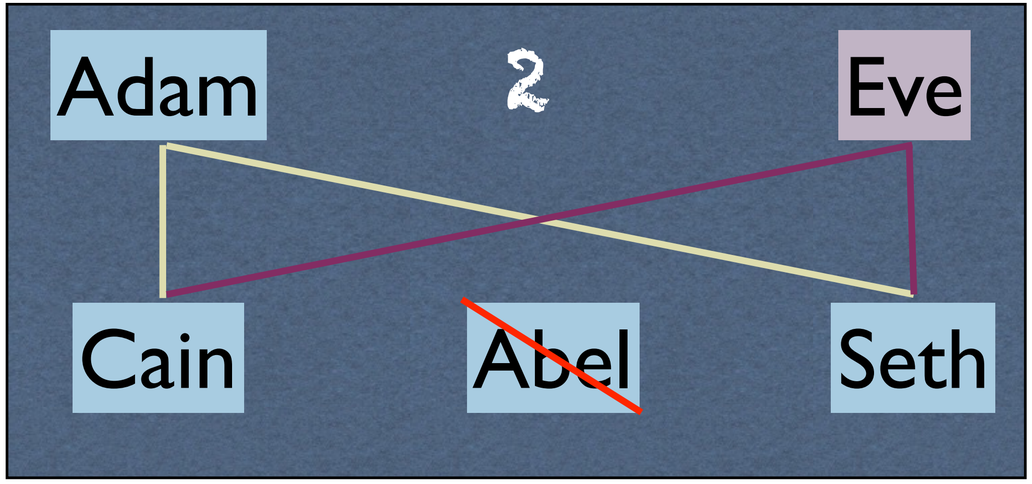
Model (part of) first biblical family

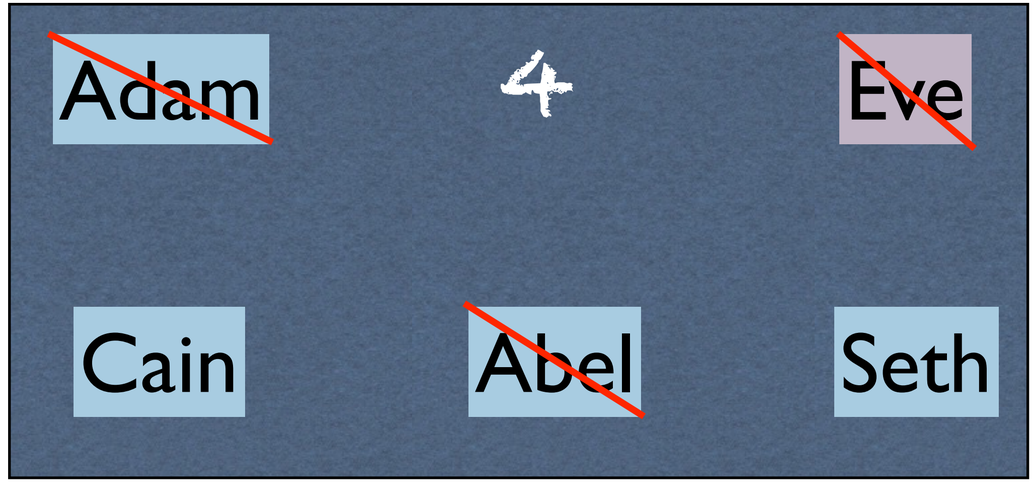
- Adam
- Eve
- sons: Abel, Cain, Seth
- represent Cain's murder of Abel
- represent Adams death
- represent Eve's death
- show a person with its name, parents' names and children's names

Steps and Effects









First cut on shared_ptr

```
#include <memory>
#include <string>
#include <vector>
#include <iosfwd>
using PersonPtr=std::shared_ptr<class Person>;

class Person {
   std::string name;
   std::vector<PersonPtr> children;
public:
   Person(std::string name):name{name}{}
   void addChild(PersonPtr child){
      children.push_back(child);
   }
   void print(std::ostream &) const;
static PersonPtr makePerson(std::string name){
   return std::make_shared<Person>(name);
}
};
```

can define shared_ptr with incomplete type

store shared_ptr in vector

```
#include "Person.h"
#include <ostream>

void Person::print(std::ostream& out)
const {
   out << "Person: "<< name;
   out << "\n    ";
   for(auto child:children){
      out << child->name << ", ";
   }
   out << '\n';
}
```

first main()

```
#include "Person.h"
#include <iostream>

void addson(std::string name,PersonPtr adam, PersonPtr eva) {
   auto son = Person::makePerson(name);
   eva->addChild(son);
   adam->addChild(son);
}
```

must add shared_ptr not object!

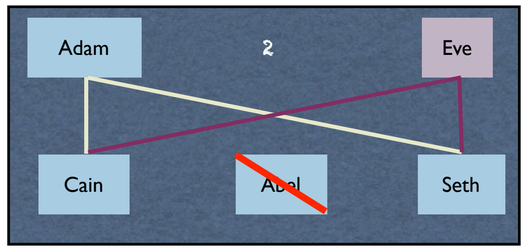
```
int main() {
   auto adam=Person::makePerson("Adam");
   adam->print(std::cout);
   auto eva=Person::makePerson("Eva");
   eva->print(std::cout);
   addson("Cain",adam, eva);
   addson("Abel",adam, eva);
   addson("Seth",adam, eva);
   adam->print(std::cout);
   eva->print(std::cout);
   // how to have Cain kill Abel?
}
```

```
Person: Adam

Person: Eva

Person: Adam
    Cain, Abel, Seth,
Person: Eva
    Cain, Abel, Seth,
```

2nd cut: "killing Abel"

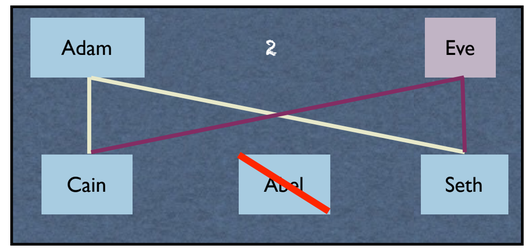


know your parents

```
class Person {
     std::string name;
     PersonPtr father;
     PersonPtr mother;
     std::vector<PersonPtr> children;
public:
     Person(std::string name,
     PersonPtr father,PersonPtr mother)
     :name{name},father{father},mother{mother}{}
     void addChild(PersonPtr child){
          children.push_back(child);
     }
     std::string getName() const { return name; }
     PersonPtr findChild(std::string name) const;
     void killChild(PersonPtr child);
     void print(std::ostream &) const;
static PersonPtr makePerson(std::string name,
                            PersonPtr father={},
                            PersonPtr mother={}){
     auto res =  std::make_shared<Person>(name,father,mother);
     if (father) father->addChild(res);
     if (mother) mother->addChild(res);
     return res;
}
};
```

search and get rid of children

2nd cut: "killing Abel"




```
void Person::print(std::ostream& out) const {
   out << "Person: "<< name ;
   out << "    "<< (father?father->getName():"orphan");
   out << "    "<< (mother?mother->getName():"orphan");
   out << "\n     ";
   for(auto const &child:children){
      out << child->name << ", ";
   }
   out << '\n';
}
PersonPtr Person::findChild(std::string theName) const {
   using namespace std::placeholders;
   auto finder=[theName](PersonPtr const &person){
      return person->getName() == theName;
   };
   auto it=find_if(children.begin(),children.end(),finder);
   if (it != children.end()) return *it;
   return nullptr;
}
void Person::killChild(PersonPtr child) {
   if (child){
      children.erase(find(children.begin(),children.end(),child));
      //if (child->father == ) ?
   }
}
```

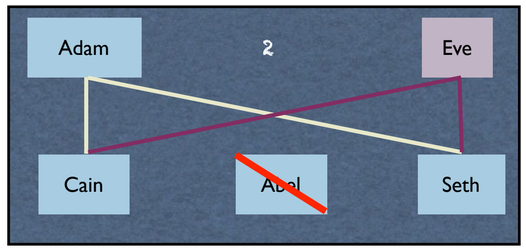
know your parents

search children, predicate as lambda

nullptr means empty shared_ptr

erase found object, shared_ptr == used

2nd cut: "killing Abel" main()




```
#include "Person.h"
#include <iostream>

void addson(std::string name,PersonPtr adam, PersonPtr eva) {
      auto son = Person::makePerson(name);
      eva->addChild(son);
      adam->addChild(son);
}

int main() {
      auto adam=Person::makePerson("Adam");
      adam->print(std::cout);
      auto eva=Person::makePerson("Eva");
      eva->print(std::cout);
      addson("Cain",adam, eva);
      addson("Abel",adam, eva);
      addson("Seth",adam, eva);
      adam->print(std::cout);
      eva->print(std::cout);
    // how to have Cain kill Abel?
    {
        auto abel=eve->findChild("Abel");
        eve->killChild(abel);
        adam->killChild(abel);
        abel->print(std::cout);
    }
    eve->print(std::cout);
    // how can we kill Adam?

}
```

last shared_ptr after killChild
killing means forgetting all references

```
Person: Adam    orphan    orphan

Person: Eve    orphan    orphan

Person: Adam    orphan    orphan
    Cain, Abel, Seth,
Person: Eve    orphan    orphan
    Cain, Abel, Seth,
Person: Abel    Adam    Eve

Person: Eve    orphan    orphan
    Cain, Seth,
```

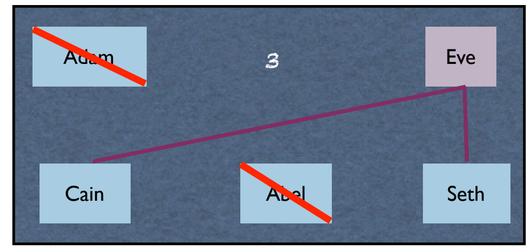
Problem: can not access shared_ptr

```
class Person
: public std::enable_shared_from_this<Person> {
```

- solution: inherit from `enable_shared_from_this`
  - CRTP pattern, pass own class as template argument
- access through `shared_from_this()` function
  - can **not** do that in constructor or destructor!

```
  auto me=shared_from_this();
// or PersonPtr me=shared_from_this();
```

3rd cut: "Adam can die"




```
class Person
: public std::enable_shared_from_this<Person> {
    std::string name;
    PersonPtr father;
    PersonPtr mother;
    std::vector<PersonPtr> children;
public:
    Person(std::string name,PersonPtr father,PersonPtr mother)
    :name{name},father{father},mother{mother}{
        // can not do shared_from_this here!
        // if(father) father->addChild(shared_from_this());
    }
    void addChild(PersonPtr child){
        children.push_back(child);
    }
    std::string getName() const { return name; }
    PersonPtr findChild(std::string name) const;
    void killChild(PersonPtr child);
    void killMe();
    void print(std::ostream &) const;
static PersonPtr makePerson(std::string name,
                            PersonPtr father={},
                            PersonPtr mother={}){
    auto res =  std::make_shared<Person>(name,father,mother);
    if (father) father->addChild(res);
    if (mother) mother->addChild(res);
    return res;
}
};
```

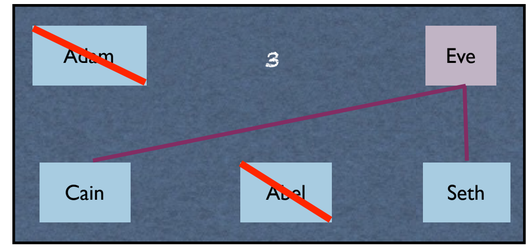
inherit from enable_shared_from_this

must still inform parents in factory function!

disentangling circular object dependency by hand!

```
void Person::killMe() {
    auto me=shared_from_this();
    if (father) father->killChild(me);
    if (mother) mother->killChild(me);
    for(PersonPtr son:children){
        if (me == son->father)
    son->father.reset();
        if (me == son->mother)
    son->mother.reset();
    }
    children.clear();
}
```

3rd part of main()



must disentangle object circular reference before destroying

```
  ...
// how can we kill Adam and Eve?
  {
     adam->killMe();
     adam->print(std::cout);
     adam.reset();
}
eve->print(std::cout);
auto cain=eve->findChild("Cain");
if (cain) cain->print(std::cout);
eve->killMe(); // avoid memory leak
eve->print(std::cout);
eve.reset();
```

last reference to Adam forgotten

last reference to Eve forgotten

Cain doesn't have children, so no need for killMe()

```
Person: Adam   orphan   orphan

Person: Eve   orphan   orphan

Person: Adam   orphan   orphan
    Cain, Abel, Seth,
Person: Eve   orphan   orphan
    Cain, Abel, Seth,
Person: Abel   Adam   Eve


Person: Eve   orphan   orphan
    Cain, Seth,
Person: Adam   orphan   orphan

Person: Eve   orphan   orphan
    Cain, Seth,
Person: Cain   orphan   Eve

Person: Eve   orphan   orphan
```

Problem: circular object reference

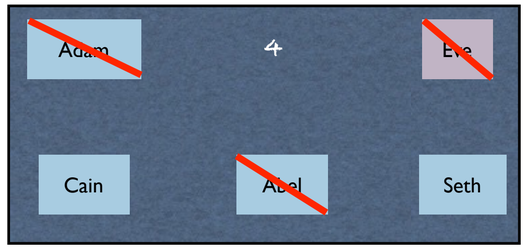
```
using WeakPersonPtr=std::weak_ptr<class Person>;
```

```
WeakPersonPtr father; // don't lock parent objects
WeakPersonPtr mother;
```

- solution: make one direction based on `weak_ptr`
  - internally Observer Pattern
  - recognizes when `shared_ptr` no longer exists
- must call `lock()` to access underlying `shared_ptr`

```
auto realfather=father.lock();
out <<(realfather?realfather->getName():"orphan");
```

4th cut: "All can die"




```
PersonPtr Person::myLock() {
    try {
        auto me=shared_from_this();
        return me;
    }catch(std::bad_weak_ptr const &ex){}
    std::cout << "++++already dead? " << name<< '\n';
    return PersonPtr{}; // already dead
}
void Person::killMe() {
    // here shared_from_this is possible
    auto me=myLock();
    if (!me) return; // already dead
    auto realfather=father.lock();
    if (realfather) realfather->killChild(me);
    auto realmother=mother.lock();
    if (realmother) realmother->killChild(me);
    children.clear();
}

Person::~Person() {
    std::cout << "killing me: "<< name << '\n';
    //killMe(); // can not call shared_from_this() in dtor!
}
```

throws when called from destructor

still need to inform parent, because it keeps a shared_ptr

no more need to inform children, because they keep only weak_ptr

just to show what happens, not needed!

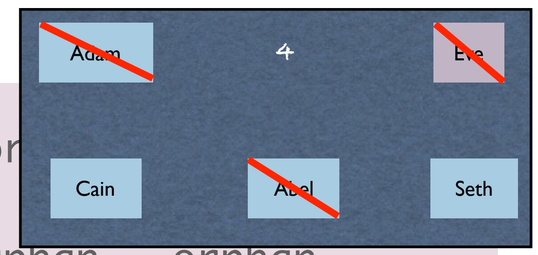
4th cut: main()

```
int main() {
     auto adam=Person::makePerson("Adam");
     adam->print(std::cout);
     auto eve=Person::makePerson("Eve");
     eve->print(std::cout);
     addson("Cain",adam, eve);
     addson("Abel",adam, eve);
     addson("Seth",adam, eve);
     adam->print(std::cout);
     eve->print(std::cout);
     // Cain kills Abel: need to remove from parents
     {
          auto abel=eve->findChild("Abel");
          eve->killChild(abel);
          adam->killChild(abel);
          abel->print(std::cout);
     }
     eve->print(std::cout);
     // kill Adam by forgetting last reference
     {
        std::cout << "killing Adam:\n";
        adam->print(std::cout);
        adam.reset();
     }
     eve->print(std::cout);
     auto cain=eve->findChild("Cain");
     if (cain) cain->print(std::cout);
     eve.reset();
     if (cain) cain->print(std::cout);
}
```

no more need to disentangle

last reference to Adam forgotten

last reference to Eve forgotten




```
Person: Adam   or

Person: Eve   orphan   orphan

Person: Adam   orphan   orphan
    Cain, Abel, Seth,
Person: Eve   orphan   orphan
    Cain, Abel, Seth,
Person: Abel   Adam   Eve

killing me: Abel
Person: Eve   orphan   orphan
    Cain, Seth,
killing Adam:
Person: Adam   orphan   orphan
    Cain, Seth,
killing me: Adam
Person: Eve   orphan   orphan
    Cain, Seth,
Person: Cain   orphan   Eve

killing me: Eve
killing me: Seth
Person: Cain   orphan   orphan

killing me: Cain
```