Incidents in Multi-threaded environment

Sami Kama

Southern Methodist University

2016-06-06 Mon

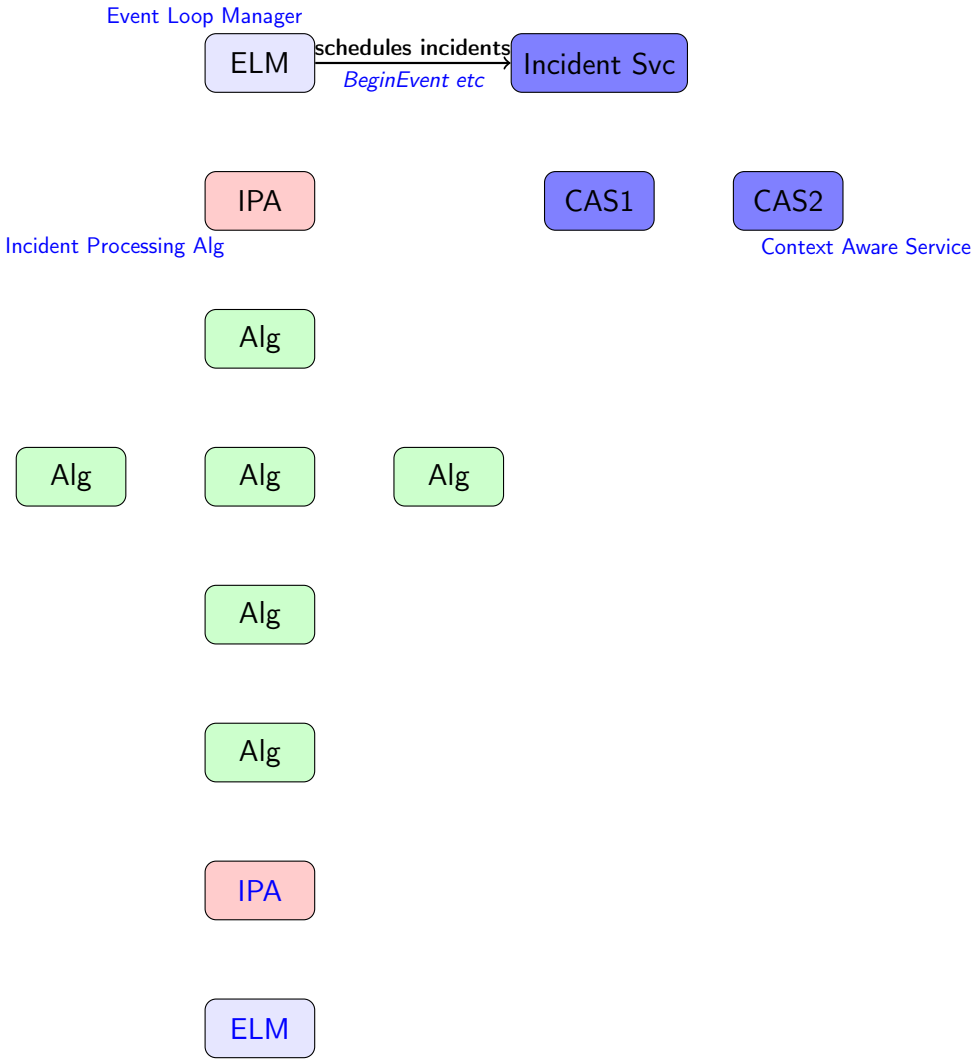
Existing Incidents

- We have simple incidents that are mostly fired outside eventloop
  - see `https://indico.cern.ch/event/472619/#day-2016-02-26`
- Usually used to setup or clean containers or variables
- In multi threaded environment incident listeners have to be context aware and thread safe
- Decision was to move context (incident) sensitive data to services and let users of them to poll the services with the context

Incidents in R2E

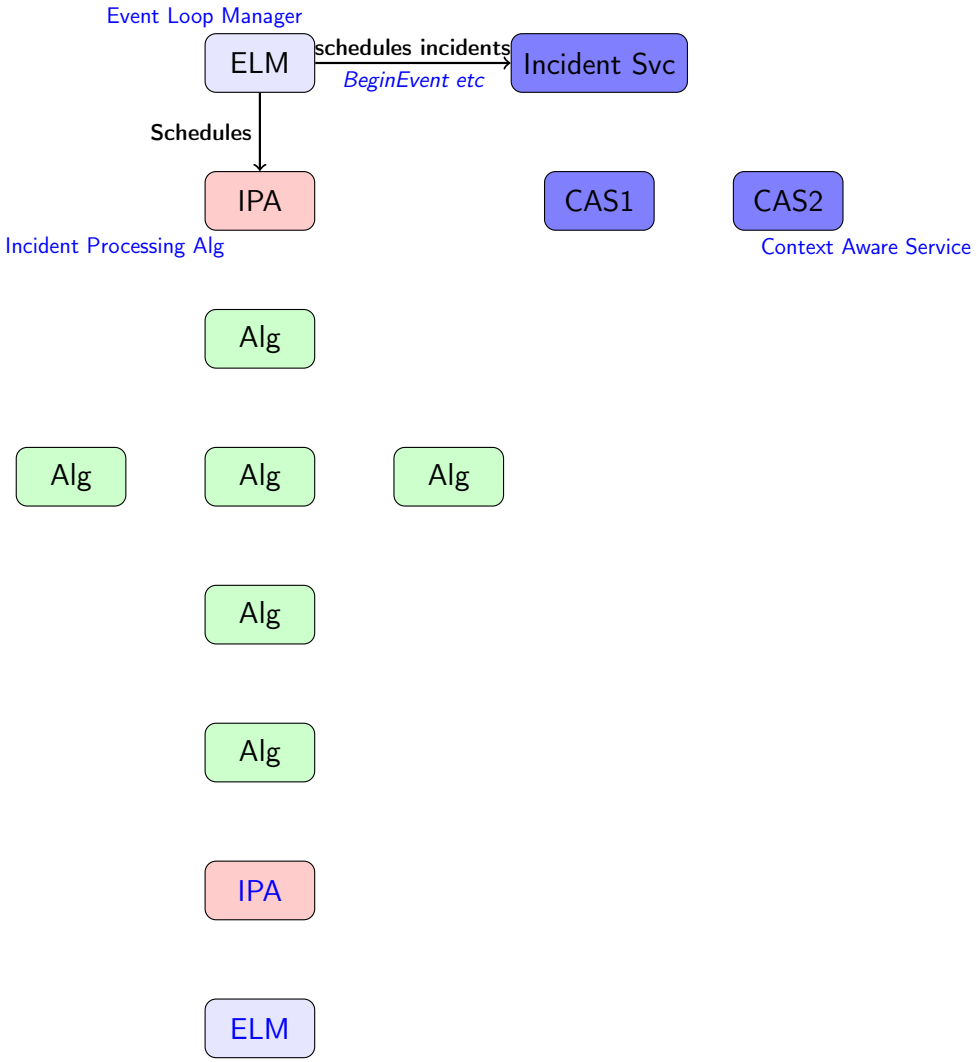
- BeginEvent
- BeginInputFile
- BeginOutputFile
- BeginRun
- EndEvent
- EndInputFile
- EndRun
- FirstInputFile
- LastInputFile
- MetaDataStop
- StoreCleared
- TrigConf

New Incidents



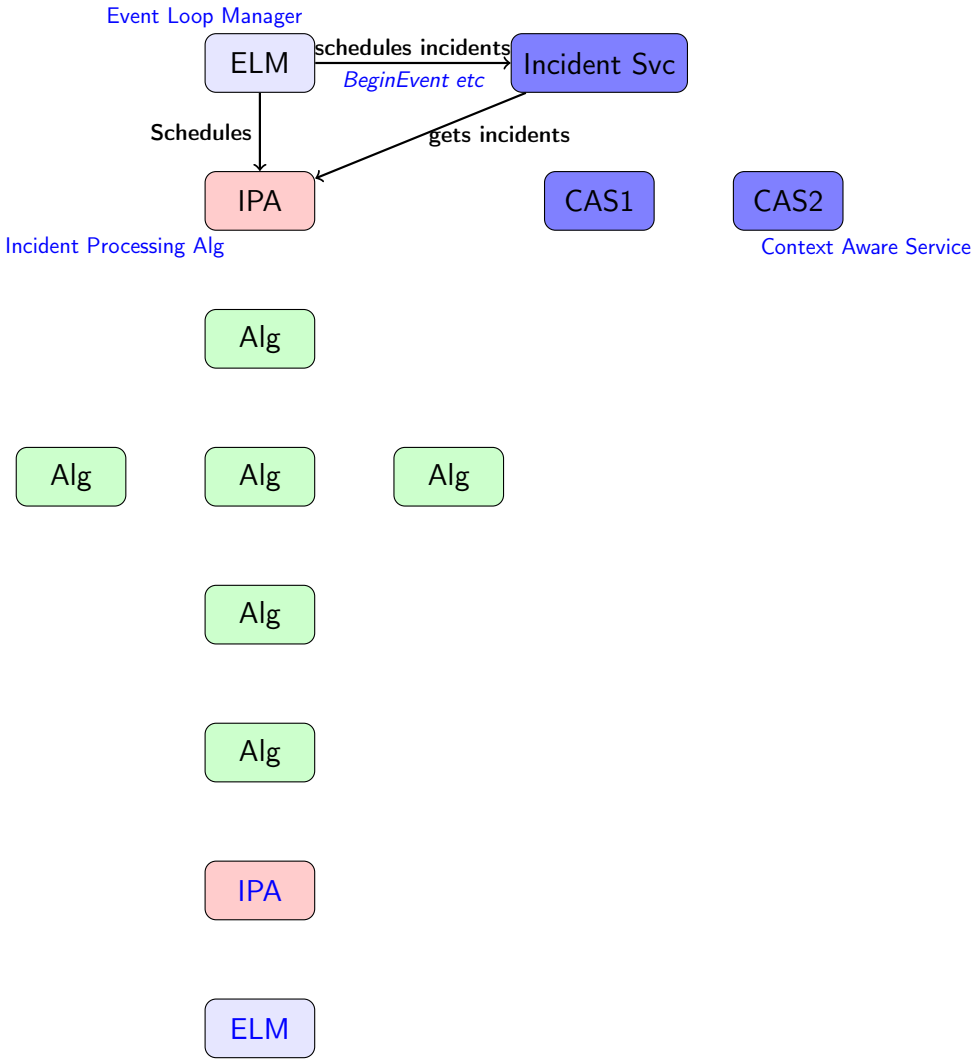
- Event Loop Manager schedules incidents

New Incidents



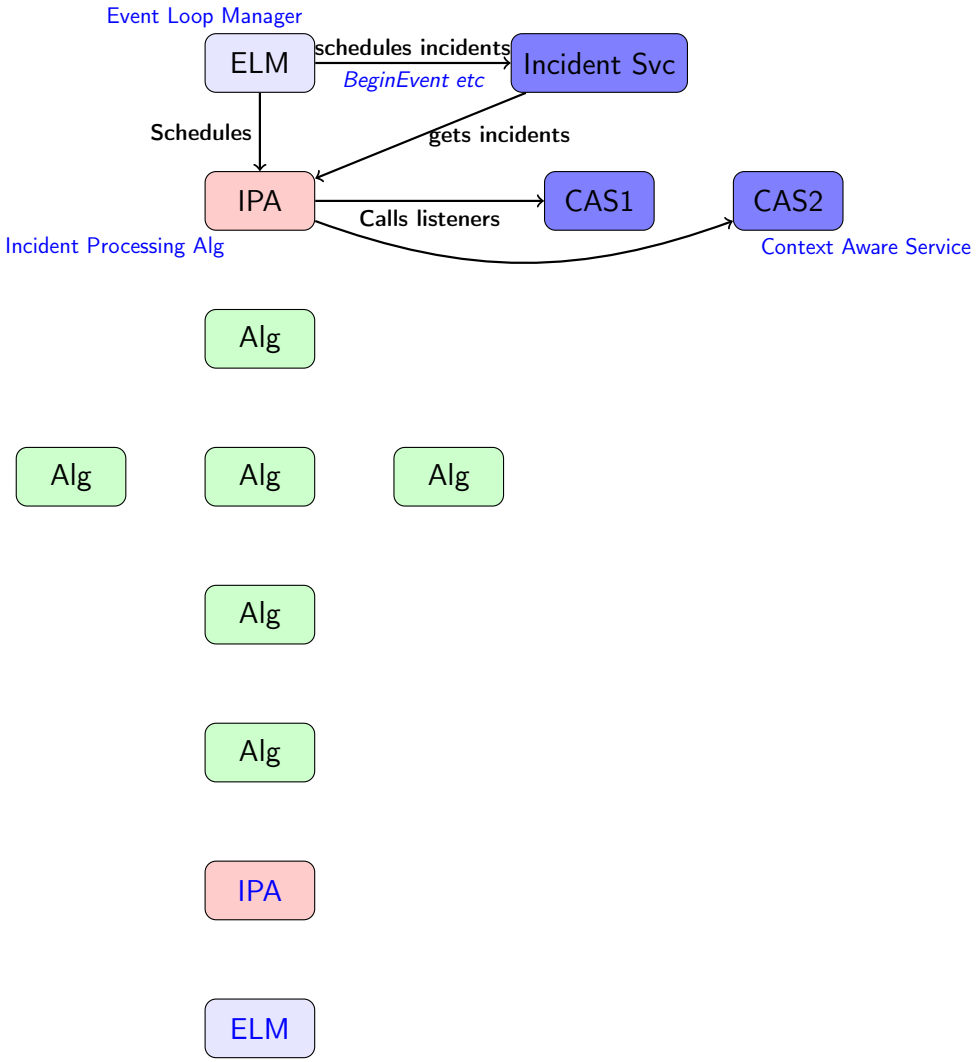
- Event Loop Manager schedules incidents
- Incident Processor Algorithm is scheduled

New Incidents



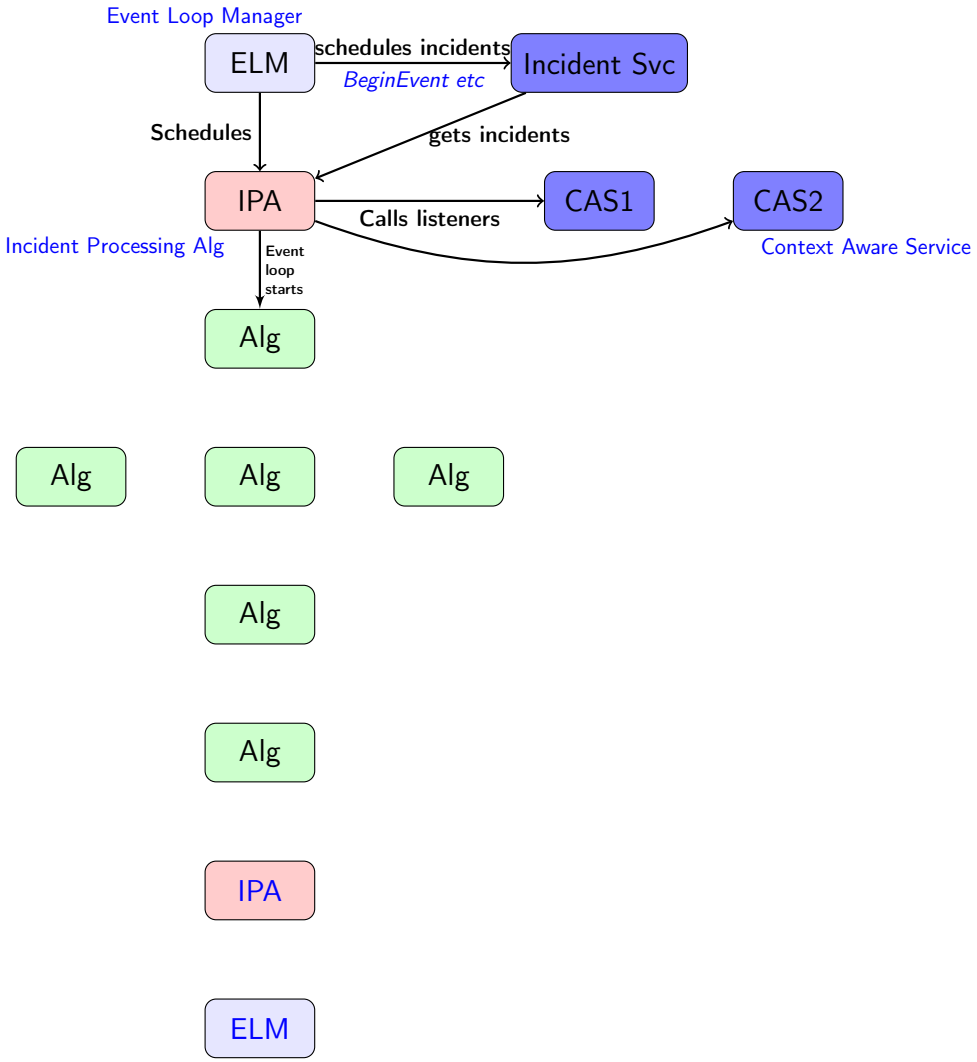
- Event Loop Manager schedules incidents
- Incident Processor Algorithm is scheduled
- IPA gets incidents

New Incidents



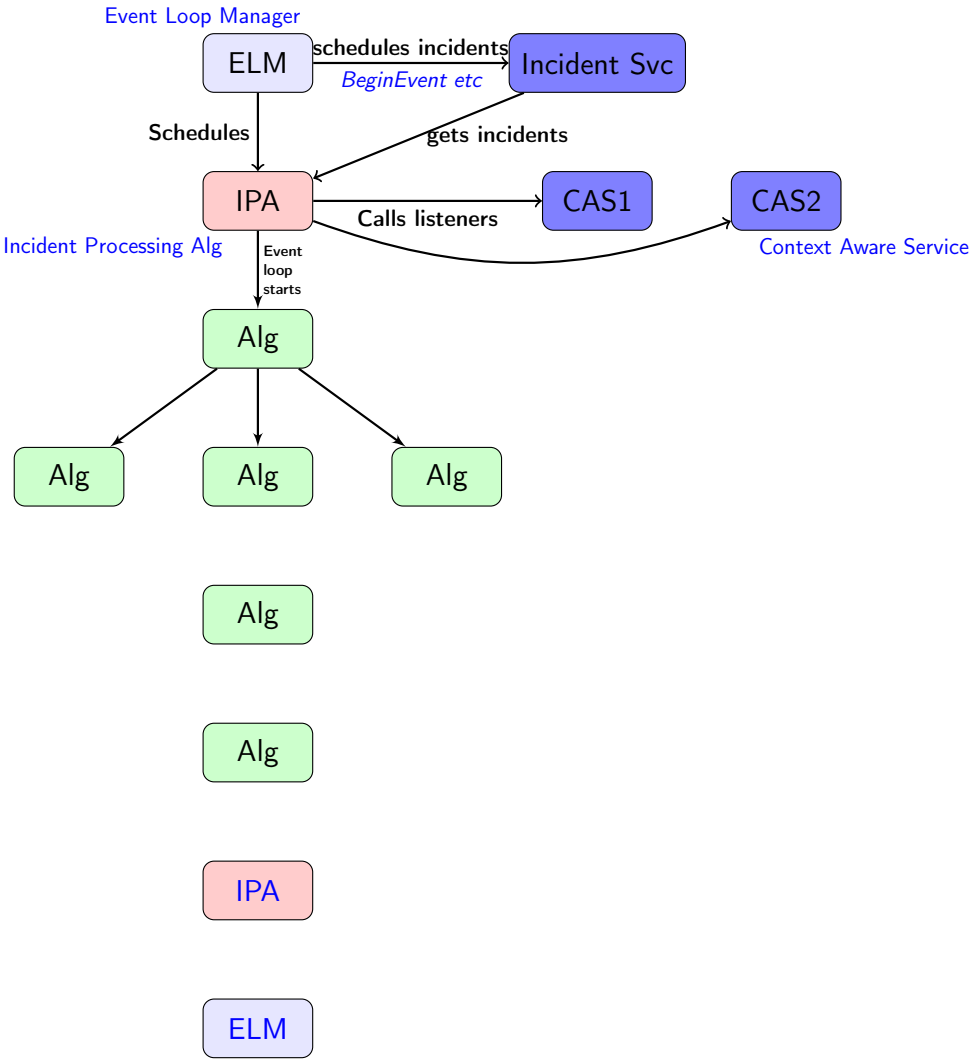
- Event Loop Manager schedules incidents
- Incident Processor Algorithm is scheduled
- IPA gets incidents
- and calls handlers

New Incidents



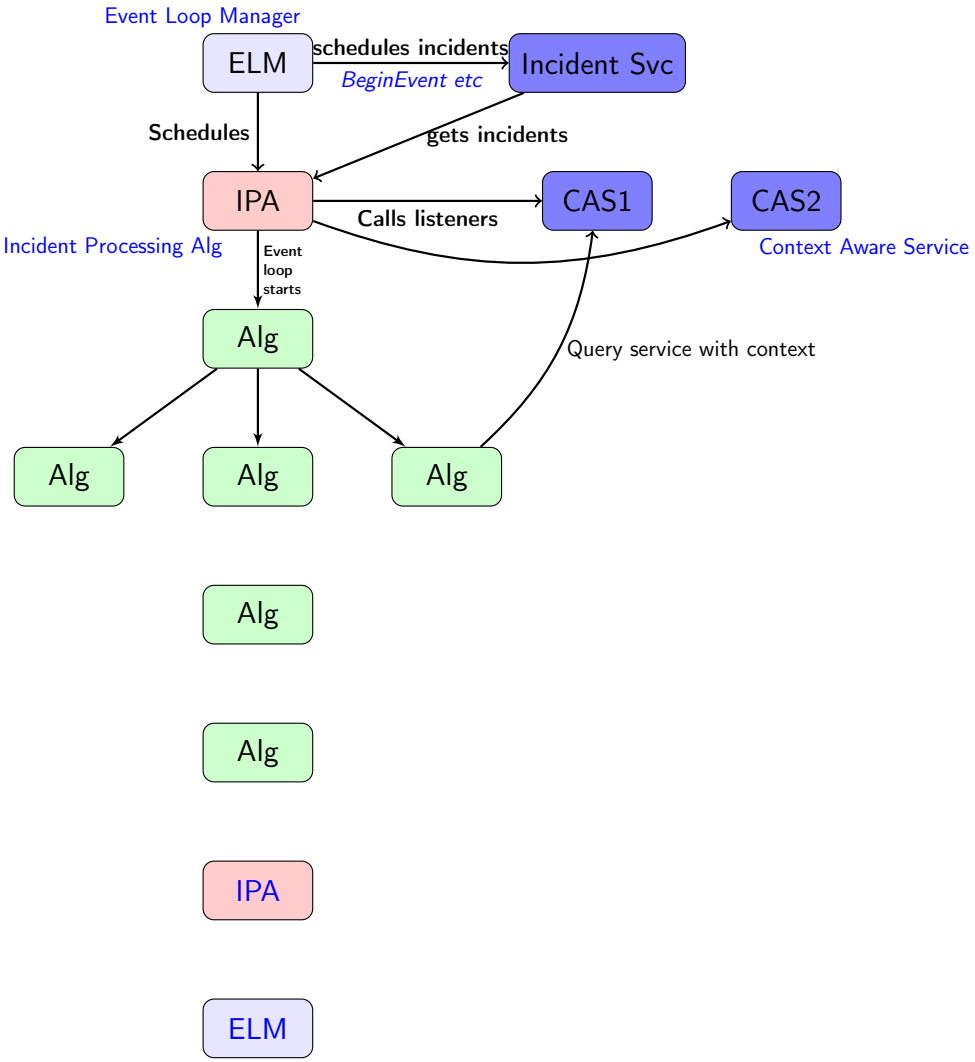
- Event Loop Manager schedules incidents
- Incident Processor Algorithm is scheduled
- IPA gets incidents
- and calls handlers
- Event loop continues

New Incidents



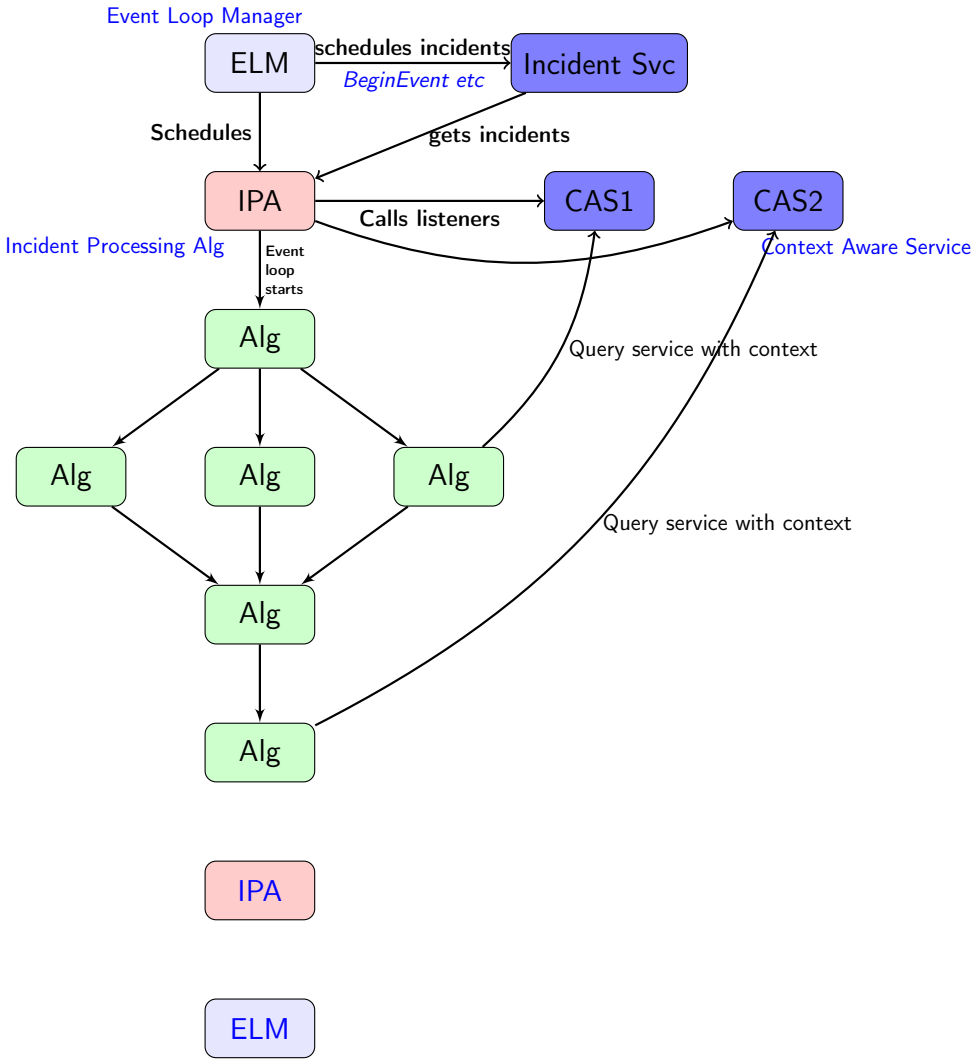
- Event Loop Manager schedules incidents
- Incident Processor Algorithm is scheduled
- IPA gets incidents
- and calls handlers
- Event loop continues
- Algorithms/Tools query the services with context

New Incidents



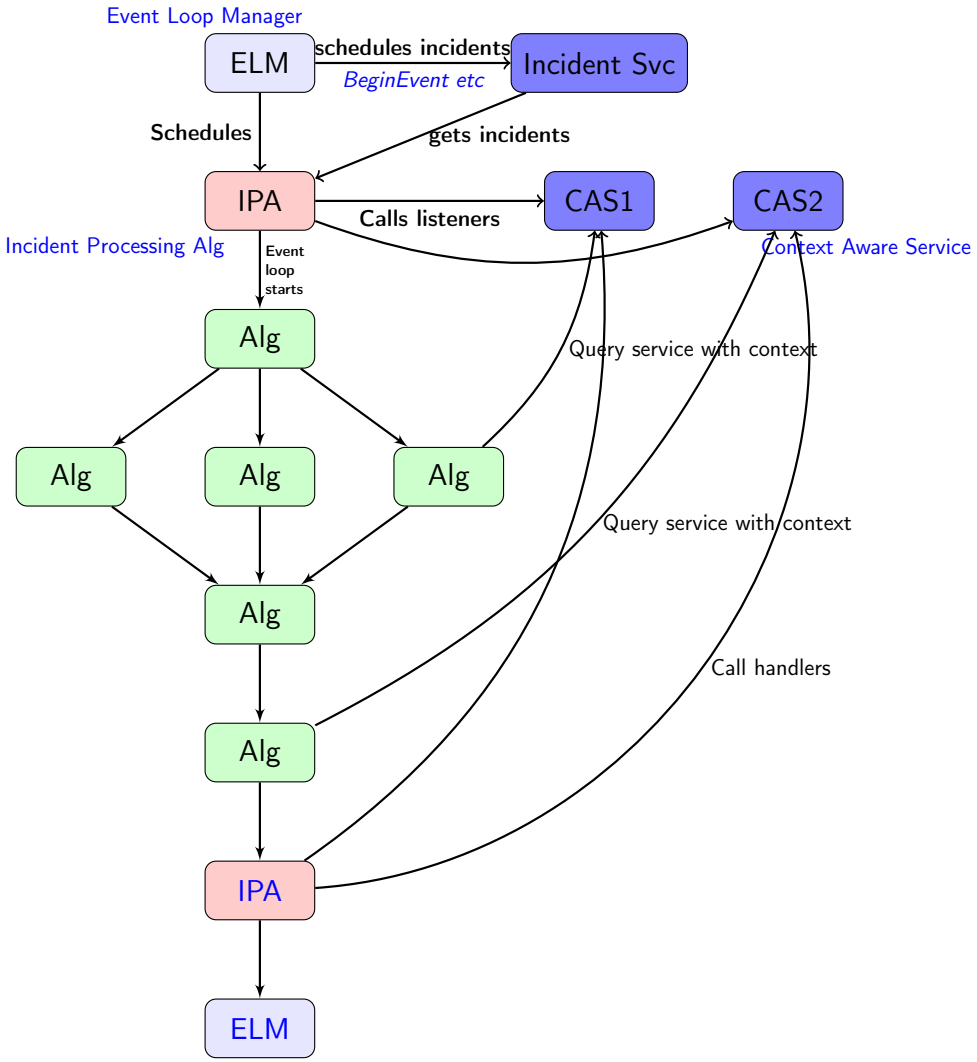
- Event Loop Manager schedules incidents
- Incident Processor Algorithm is scheduled
- IPA gets incidents
- and calls handlers
- Event loop continues
- Algorithms/Tools query the services with context

New Incidents



- Event Loop Manager schedules incidents
- Incident Processor Algorithm is scheduled
- IPA gets incidents
- and calls handlers
- Event loop continues
- Algorithms/Tools query the services with context

New Incidents



- Event Loop Manager schedules incidents
- Incident Processor Algorithm is scheduled
- IPA gets incidents
- and calls handlers
- Event loop continues
- Algorithms/Tools query the services with context
- Loop continues until the end of the event

Changes to IncidentSvc

Incidents are asynchronous so incidents need to be stored until consumed.

```
fireIncident(std::unique_ptr<Incident> inc);
```

New fireIncident method causes IncidentSvc to schedule incident, otherwise old serial behavior is kept

Example Service

Incident listeners become services, or at least reentrant.

Example

```
class  IncidentAsyncTestSvc: public extends<Service,
    IIncidentListener,
    IIncidentAsyncTestSvc> {
public:
  IncidentAsyncTestSvc( const std::string& name, ISvcLocator* svcloc);
     virtual ~IncidentAsyncTestSvc();
  StatusCode initialize() override;
  StatusCode finalize() override;
  // Handle callback
  virtual void handle(const Incident& incident) final;
  //real users query service to get the data
  virtual void getData(uint64_t* data,EventContext* ctx=0) const final override;
private:
  std::string m_name;
  uint64_t m_fileOffset;
  uint64_t m_eventMultiplier;
  long m_prio;
  StringArrayProperty m_incidentNames;
  SmartIF<IMessageSvc> m_msgSvc;
  SmartIF<IIncidentSvc> m_incSvc;
  tbb::concurrent_unordered_map<EventContext,uint64_t,
     EventContextHash,EventContextHash> m_ctxData;
  std::mutex m_eraseMutex;
}
```

Example Service 2

Handler implementation

```
void IncidentAsyncTestSvc::handle(const Incident &incident) {
   MsgStream log( m_msgSvc, m_name );
   if(incident.type()==IncidentType::BeginEvent){
     //consume incident
     auto res=m_ctxData.insert(std::make_pair(incident.context(),
      incident.context().evt()*m_eventMultiplier+m_fileOffset));
     if(!res.second){
       log << MSG::WARNING << m_name<<" Context already exists for '" << incident.type()
   << "' event="<<incident.context().evt() << endmsg;
     }
   }else if(incident.type()==IncidentType::EndEvent){
     {
       //release resources
       std::unique_lock<decltype(m_eraseMutex)>(m_eraseMutex);
       auto res=m_ctxData.unsafe_erase(incident.context());
       if(res==0){
 log << MSG::WARNING << m_name<<" Context is missing for '" << incident.type()
     << "' event="<<incident.context().evt() << endmsg;
       }
     }
     log << MSG::INFO <<m_name<< " Cleaned up context store for event =" <<incident.context().evt()
 << " for incident='"<<incident.type() <<"'"<<endmsg;
   }
   log << MSG::INFO << m_name<<" Handling incident '" << incident.type()
       << "' at ctx="<<incident.context() << endmsg;
 }
```

Example Service 3

Service implementation

```
void IncidentAsyncTestSvc::getData(uint64_t* data,EventContext* ctx)const {
  MsgStream log( m_msgSvc, m_name );
  log<<MSG::DEBUG<<"Asked for data with context "<<*ctx<<endmsg;
  if(ctx){
    auto cit=m_ctxData.find(*ctx);
    if(cit==m_ctxData.end()){
      log<<MSG::FATAL<<" data for event "<<ctx->evt()
<<" is not initialized yet!. This shouldn't happen!"<<endmsg;
      return;
    }
    *data=cit->second;
  }else{
    const auto& ct=Gaudi::Hive::currentContext();
    auto cit=m_ctxData.find(ct);
    if(cit==m_ctxData.end()){
      log<<MSG::FATAL<<" data for event "<<ct.evt()
       <<" is not initialized yet!. This shouldn't happen!"<<endmsg;
      return;
    }
    *data=cit->second;
  }
}
```

Consumer algorithm

Simple Example

```
StatusCode IncidentAsyncTestAlg::execute() {
  uint64_t data=0;
  MsgStream logstream(msgSvc(), name());
  for (auto & inputHandle: m_inputObjHandles){
    if(!inputHandle->isValid())
      continue;

    DataObject* obj = nullptr;
    obj = inputHandle->get();
    if (obj == nullptr)
      logstream << MSG::ERROR << "A read object was a null pointer." << endmsg;
  }
  m_service->getData(&data);
  for (auto & outputHandle: m_outputObjHandles){
    if(!outputHandle->isValid())
      continue;
    outputHandle->put(new DataObject());
  }
  info() << "Read data "<<data << endmsg;
  return StatusCode::SUCCESS;
}
```

Migration Steps

- We should convert listeners to services
- Heavyweight incidents should be converted to algorithms
- Consumers of incident information should be made contex aware