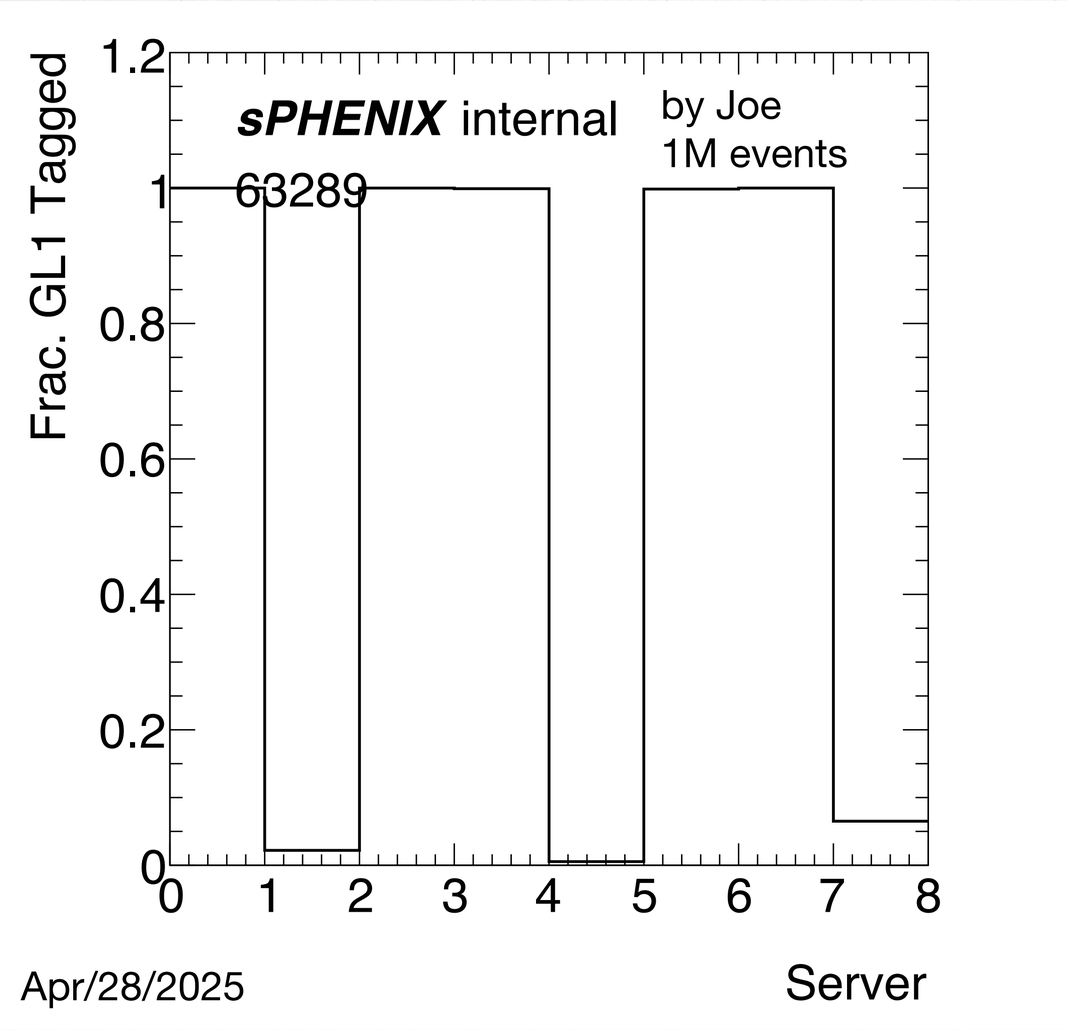
GL1-INTT matching

J. Bertaux (Purdue), J. Hwang(Korea),
S. Liechty (Colorado Boulder), G. Nukazuka (RIKEN),
J. Osborn (BNL), J. Park (Colorado Boulder)

GL1-INTT matching at packet level



Posted on Mattermost

BCO of each INTT half-ladder (fee packet BCO) is taken by intt_pool::lValue(fee, k, “BCOVAL”).
If the difference b/w Fee packet BCOs and the reference BCO, which is InttRawHit[0] BCO), is less than 3, it’s counted.

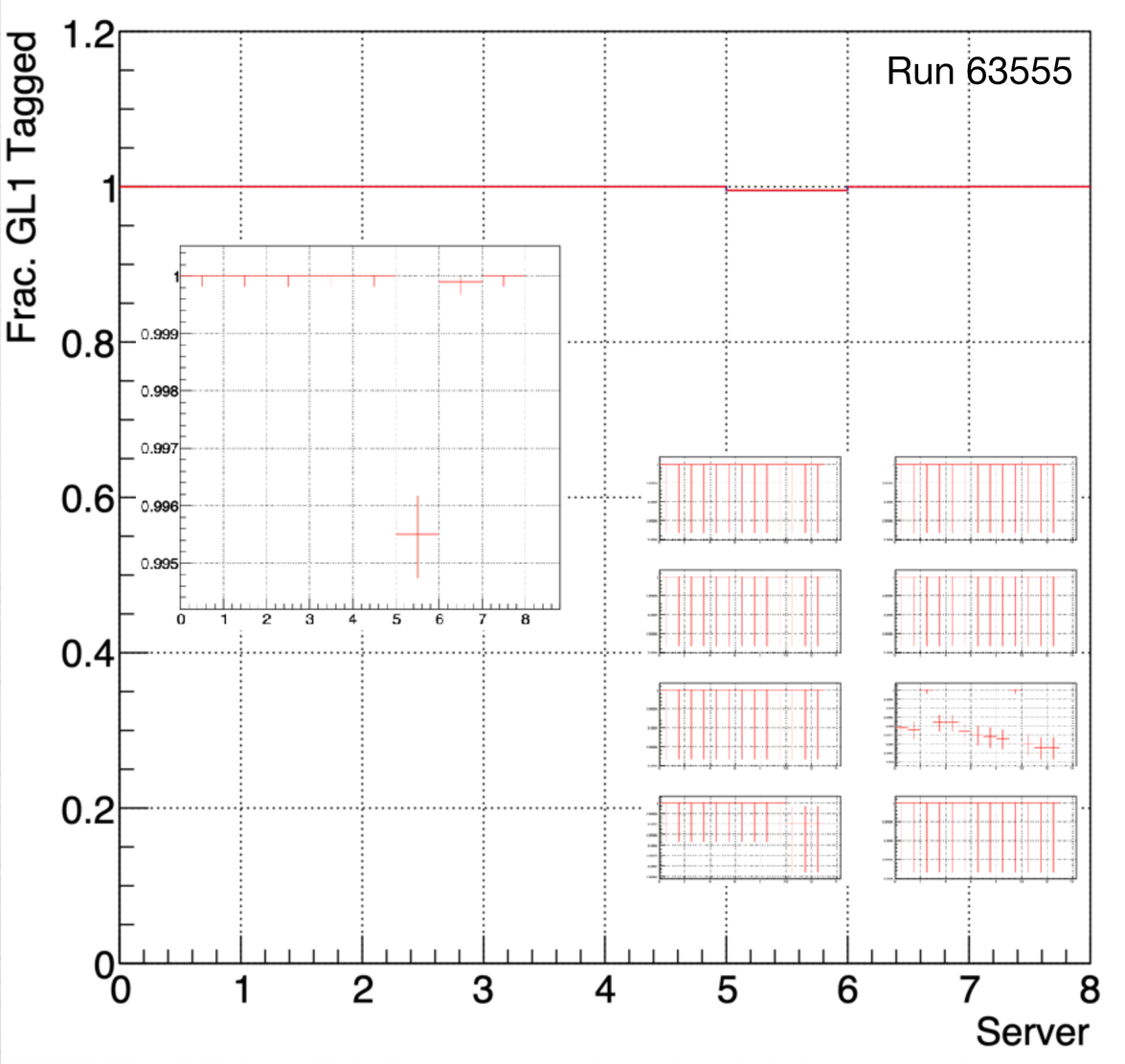
$$\text{Frac. GL1 Tagged} \equiv \frac{\text{N(All INTT ladder BCO aligned around reference BCO)}}{\text{N(InttRawHit[0] BCO (=\#GL1 BCO))}}$$

It’s true if INTT pool is perfect.

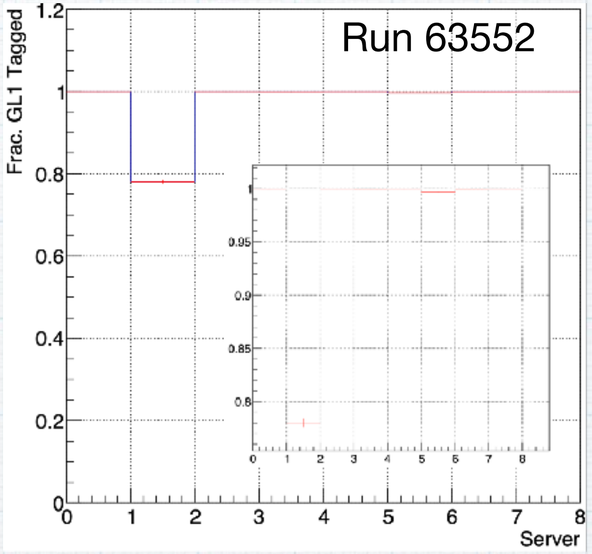
If all fee in all INTT FELIX have the same BCO as GL1’s, the fraction is 1.

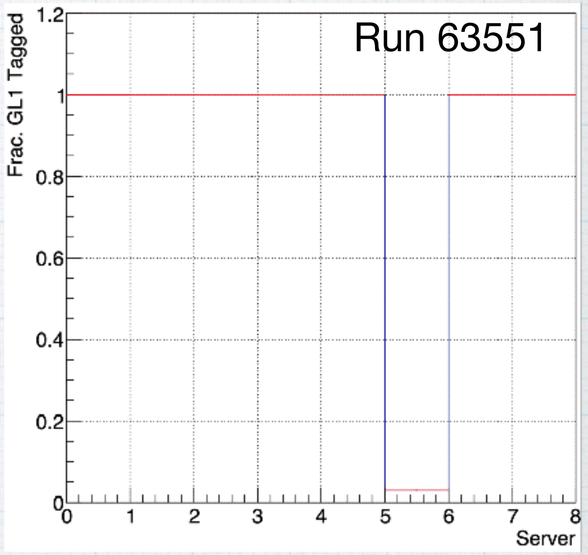
GL1-INTT matching at packet level

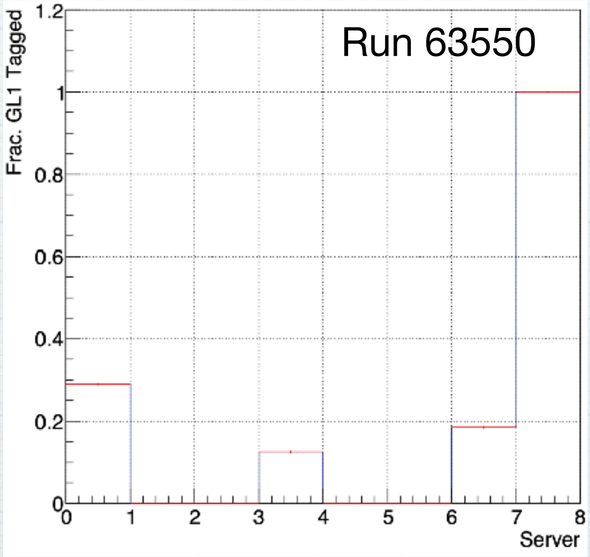
Good case

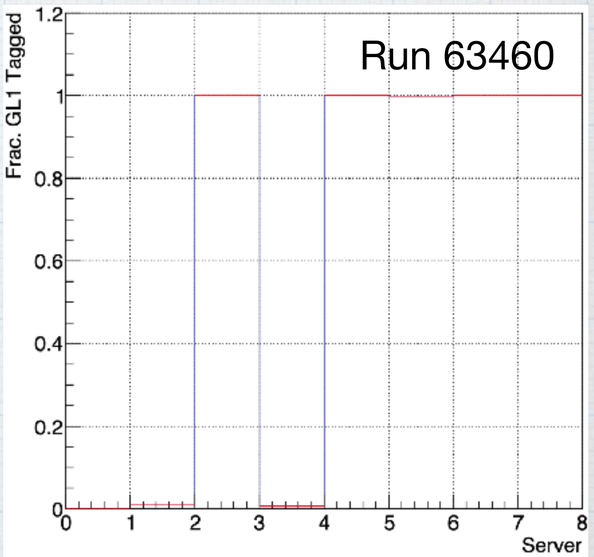


Bad cases



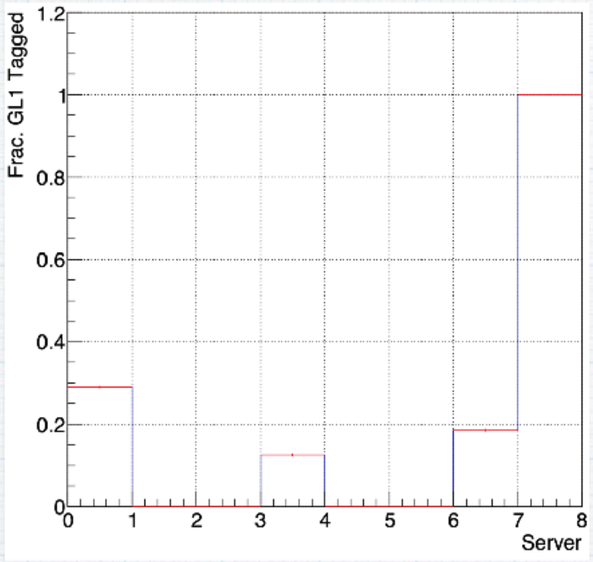




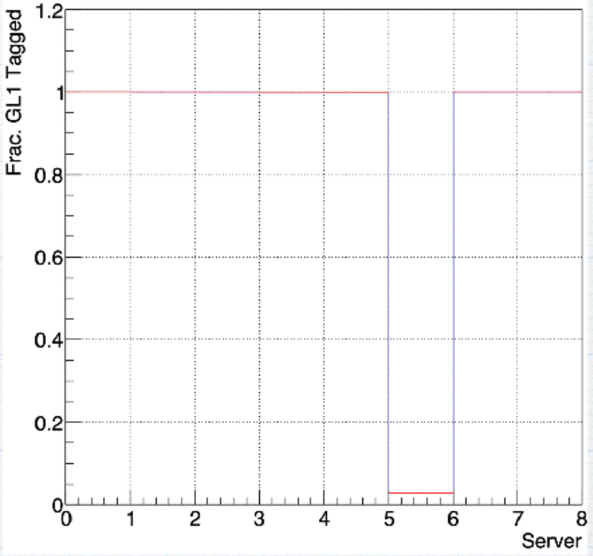


How bad???

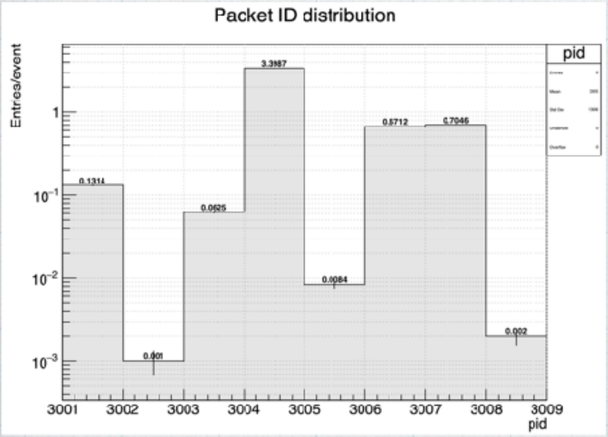
Run 63550

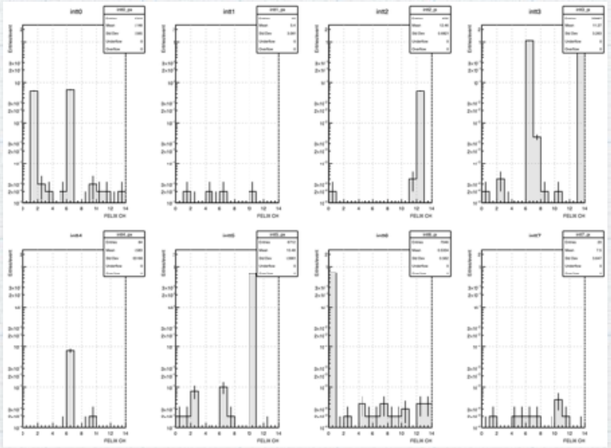


Run 63551

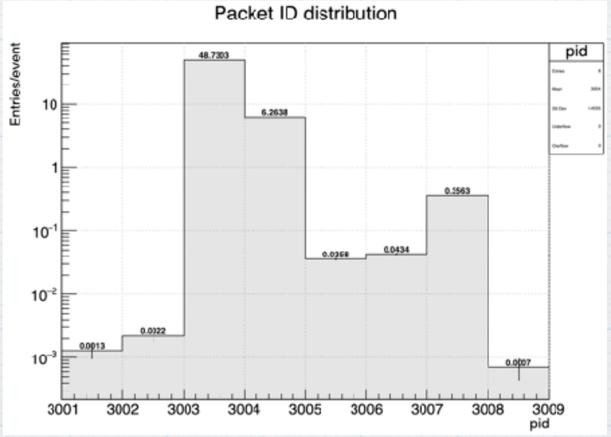


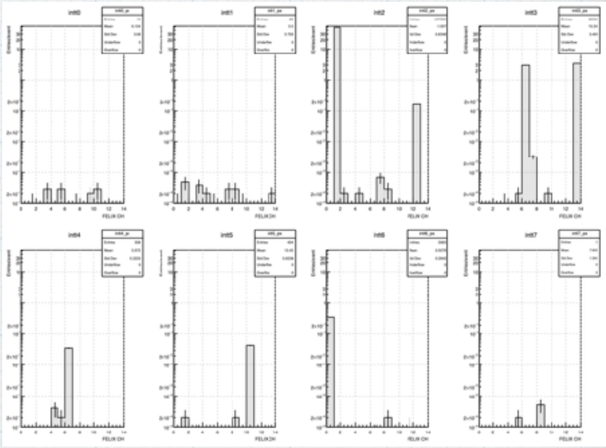
InttRawHit QA





InttRawHit QA





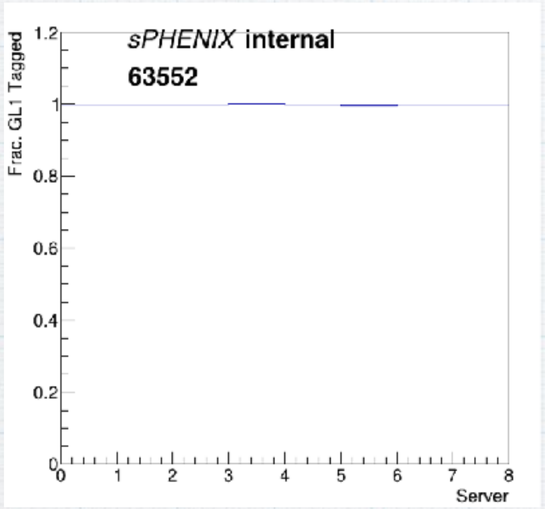
0% tag fraction doesn't mean no hit. If GL1 data is also used

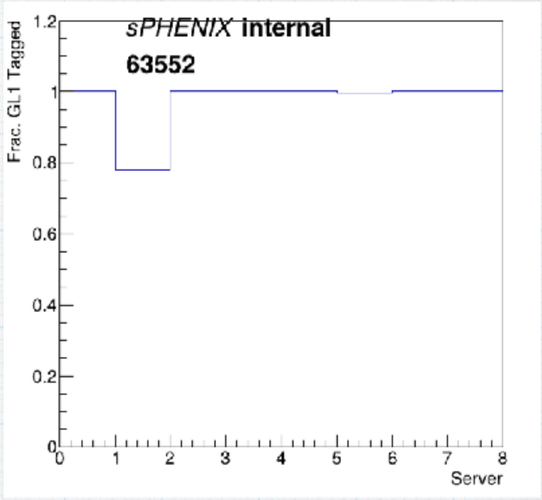
```
Gl1RawHitMap is empty - we are done
```

Timing dependence of GL1-INTT matching at packet level?

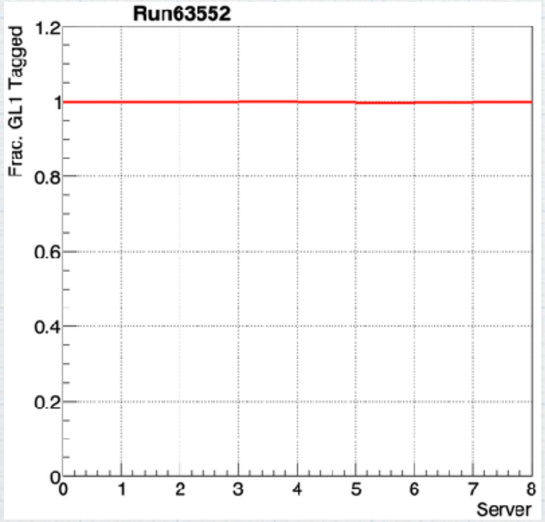
1k events | 2k events | 5k events | 10k events

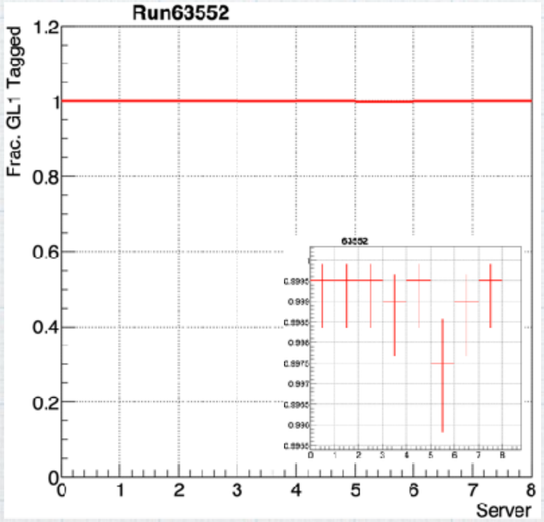
by Jaein

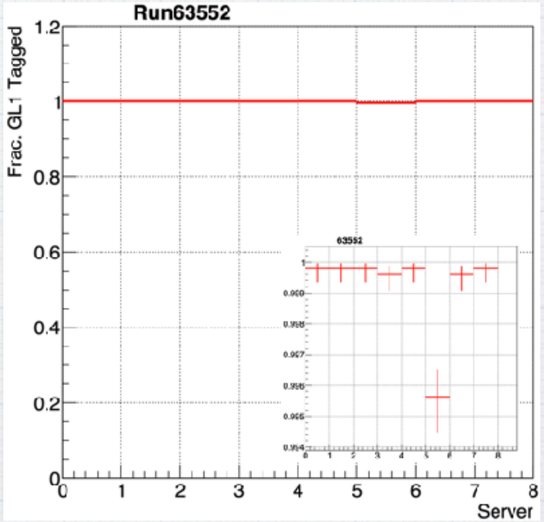


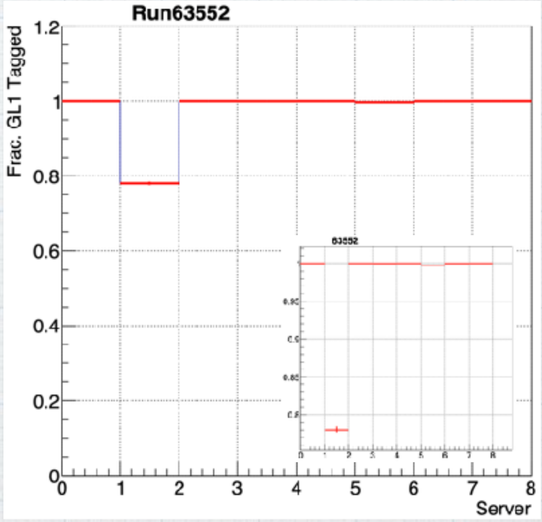


by Genki









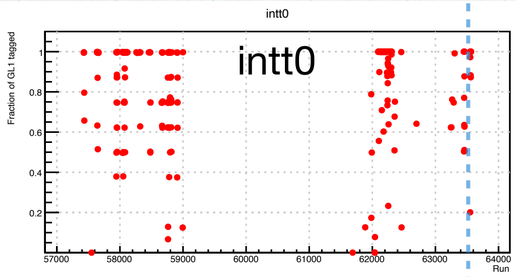
Lots of cosmic results: Joe already made them

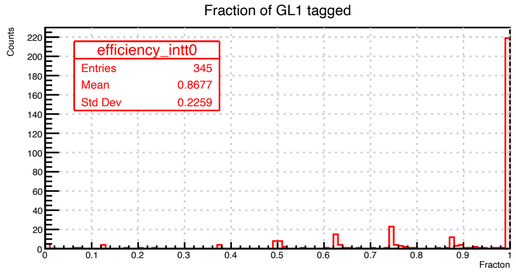
- `/sphenix/data/data02/sphnxpro/QAhtml/aggregated/run3auau/cosmics/new_nocdbtag_v000/`
- Results of 929 cosmic runs are there. 318 runs are OK to use.
- I also analyzed 88 new cosmic runs (63000–63564) with Joe's codes.
  - All events are being processed for 48 runs. They are running more than 2 days and never end… and killed.
  - 10k -- 1M events are processed for other runs. Statistics is not important to find loss > 5%.

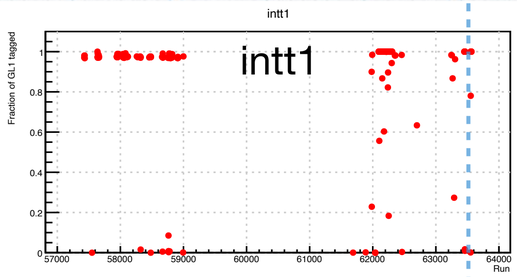
Lots of cosmic results

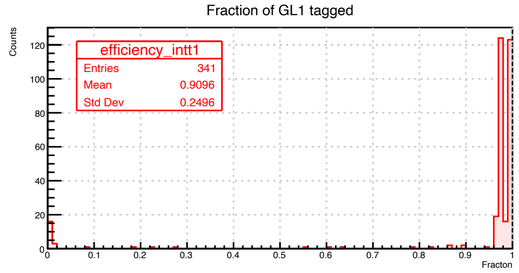
hot channel modification

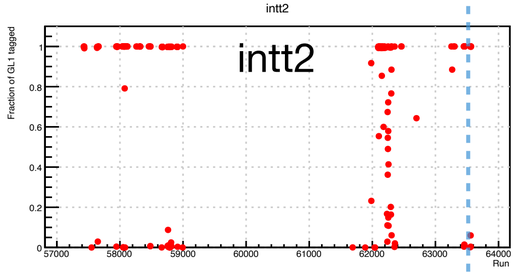
hot channel modification

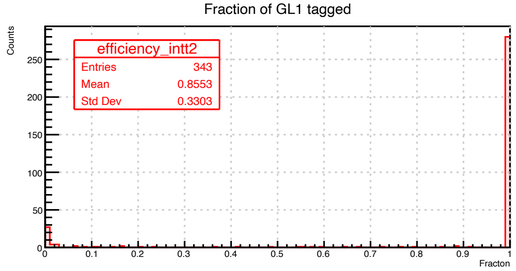


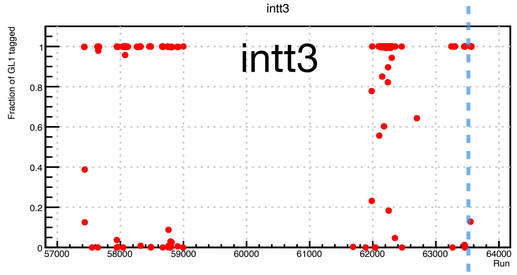


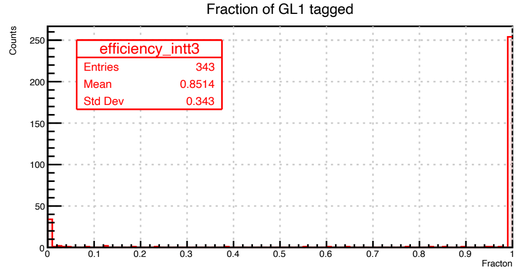


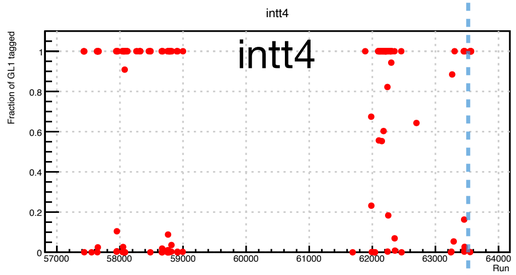


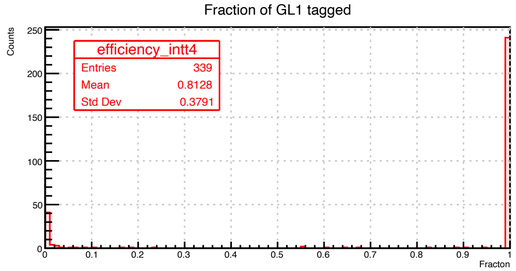


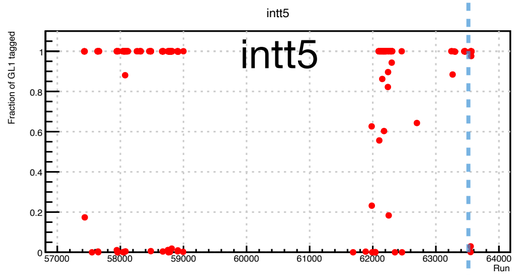


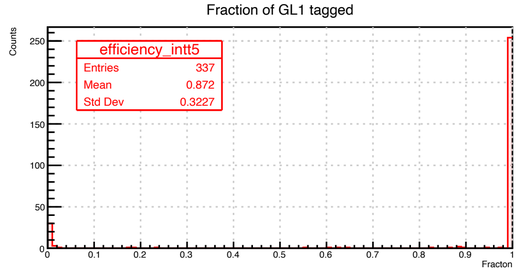


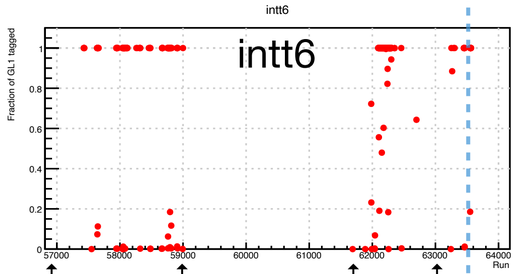


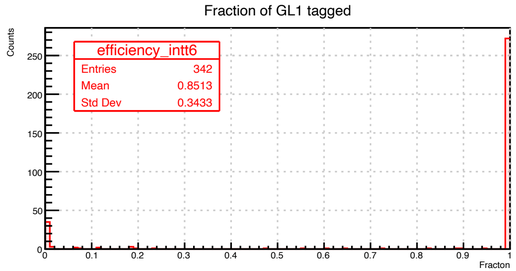


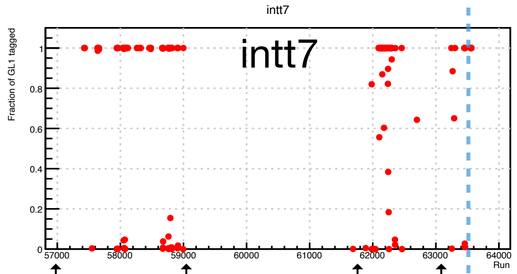


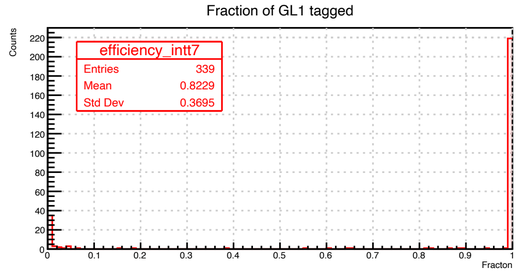












Feb/28 Mar/24 Apr/12 Apr/26

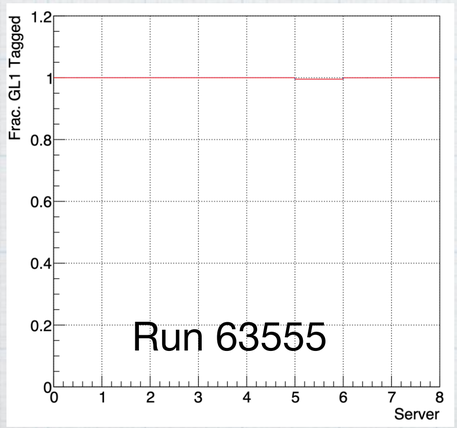
Feb/28 Mar/24 Apr/12 Apr/26

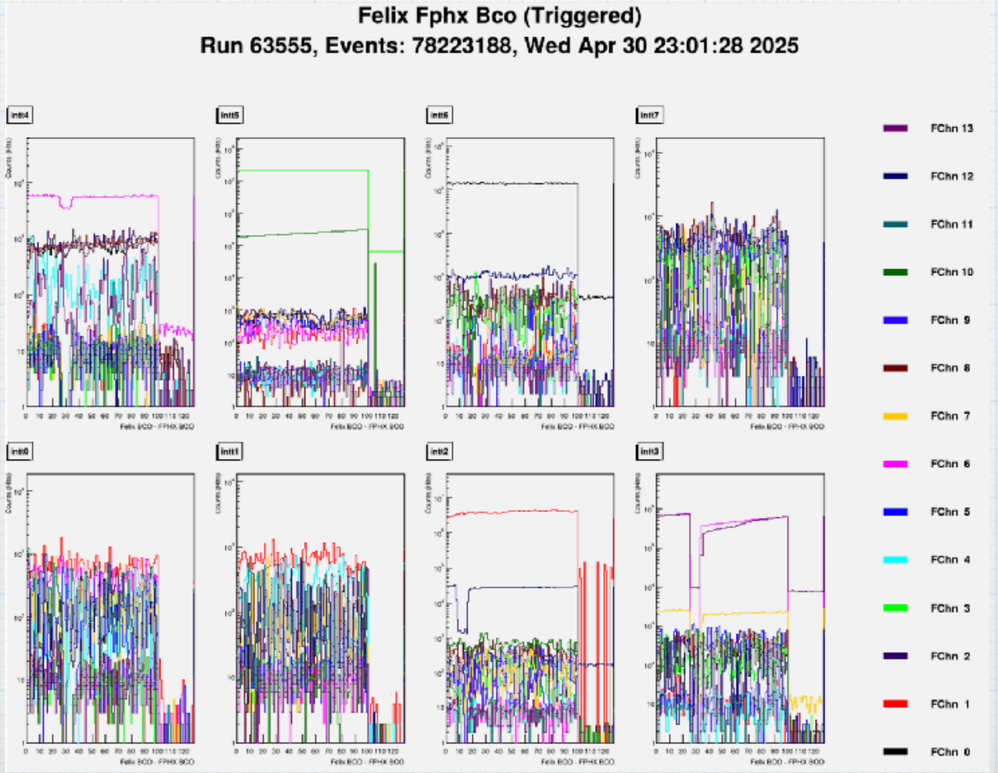
What was the situation last year?

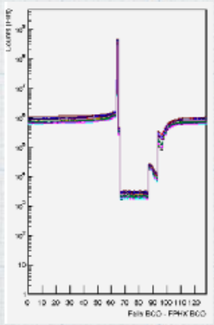
- We took a month for GL1-INTT matching study last year, but it was for the streaming readout mode.
- According to Jaebeom, trigger mode runs of pp and AuAu had beed checked. Where are the results?
- It’s better to check the situation last year, anyway. Unfortunately, most of evt files in luster were removed. Some files are /sphenix/lustre01/sphnxpro/fromhpss/physics/INTT/physics. But none of runs have a complete set of files. I requested Chris to transfer some runs. It will be completed in this weekend.

What can be done next?

- Communication with Raul: Done. He has no idea for now.
- Touching evt files directly with pmonitor: ongoing
- Consistency check within INTT: not highest priority
- Hit timing wrt GL1 check: There is no useful information in such noisy condition too low hit rate.
- Hit timing wrt GL1 check with tracking to remove noise: not highest priority
- Hot channel mask: Don’t expect improvement by hot channel mask.







if collisions happen…

Change on GTM firmware?

Joseph Osborn 10:53

Hi all, doing some more investigating into the cosmic data... @Gregory Ottino and I noticed that there were no INTT hits in the produced cosmic data (do not panic, there are hits in the prdfs as indicated by the INTT pooler). I dug into the event combiner and found the reason no raw hits are created. The GTM and GL1 BCO seem to be consistently offset by 2 clocks instead of 1 (as was the case last year) in some of the recent data. The event combiner adds hits to the GL1 based on a clock diff of only 1, which is why no raw hits were being written out. What this means is that the problem is resolved here if we set the negative bco range to 0, so that an allowed matching range of GTM-GL1 BCO == 2 is allowed here.

So the questions are:

1. Is this change expected? I checked in a couple of runs and it seems to be the case going back several weeks (62461,62244, 60165)
2. The fix for the problem is easy - just to set the negative BCO range to 0 instead of 1. But this means we have a time dependent change that needs to be dealt with, since this was not the case in triggered mode in AuAu last year and even in cosmics earlier this year in March. What is the best way to handle this, something in the CDB or in the database that looks up whether the INTT is in streaming or triggered mode?

Raul Cecato 11:32

The triggered mode uses the usual lvl1 signal from the GTM, while the streaming mode uses a modebit as a fake trigger. If some relative timing between them changed on the GTM side this might be causing some issue, since the INTT fw itself hasn't changed in a long time. *Edited*

Joseph Osborn 11:41

If it is worth anything it looks like the change happened between runs 58668 and 58791. I will just update the pooler to check for a bco difference of 2 from then on. I narrowed it down to the change happening on March 21, starting with 58753

Edited

2

JAEBEOm PARK 12:06

I think that is the time when the GTM firmware got changed so could be that

Raul Cecato 12:06

that's around 03/21 and 03/23

JAEBEOm PARK 12:06

yes

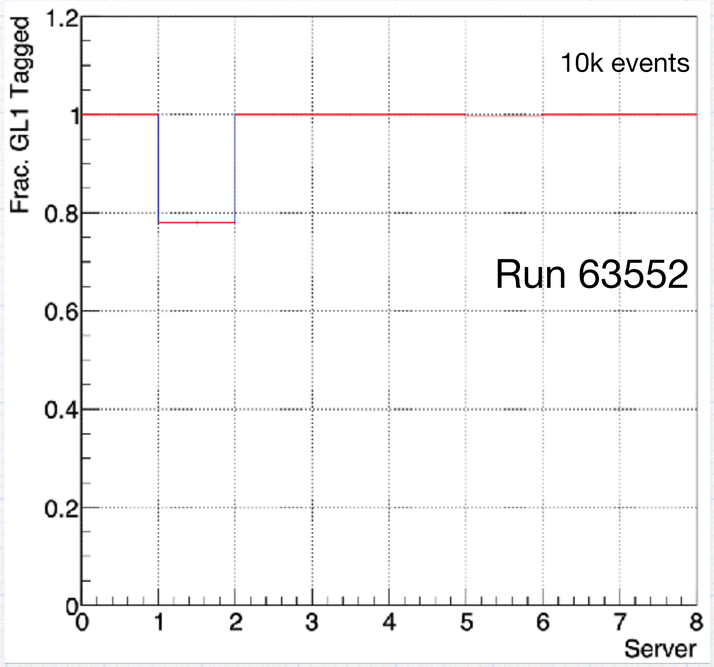
Joseph Osborn 12:07

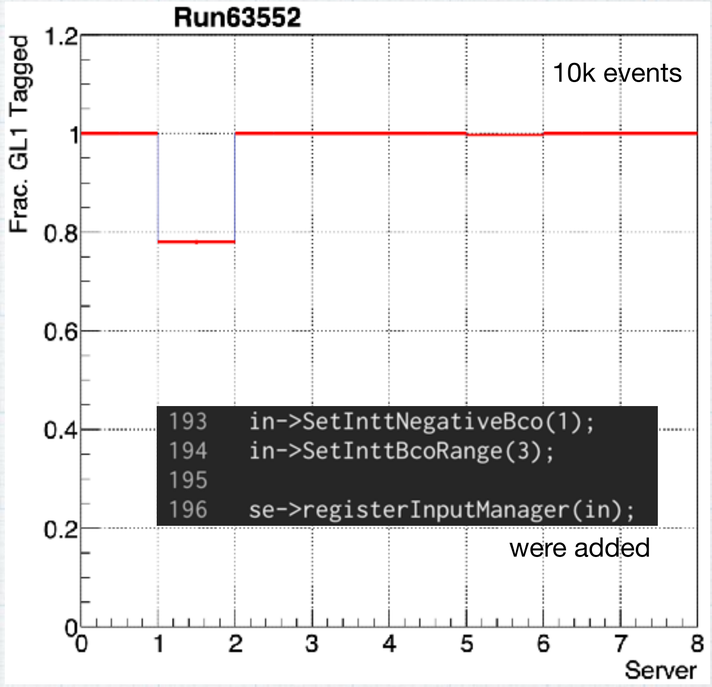
Right, the date was March 21. The last GTM-GL1 diff==1 run was 58677 at 9 AM and then the next run taken at 8 PM has diff==2. So it is probably associated to the gtm firmware change

1

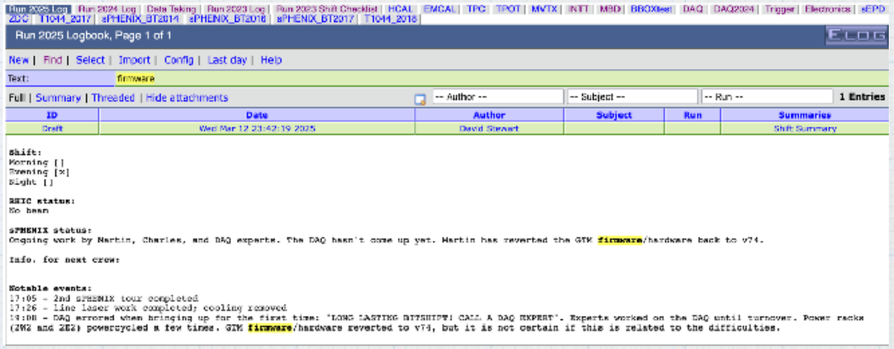
Joseph Osborn 12:16

Note this is also an indication that the BCO matching QA was wrong, because the matching was only performed within a clock diff of 2. Since the gtm firmware update moved the clock diff, we should really be checking within 3 ticks for all the BCOs to be there





Cosmetics was changed. But only cosmetics.
The fraction of intt1 is exactly same as before changing the BCO configuration.



Run 2025 Logbook, Page 1 of 1

New | Find | Select | Import | Config | Last day | Help

Text: firmware

Full | Summary | Threaded | Hide attachments

ID	Date	Author	Subject	Run	Summaries
Draft	Wed Mar 12 23:42:19 2025	David Stewart			Shift Summary

Shift:
Morning [ ]
Evening [X]
Night [ ]

RHIC status:
No beam

sPHENIX status:
Ongoing work by Martin, Charles, and DAQ experts. The DAQ hasn't come up yet. Martin has reverted the GTM firmware/hardware back to v74.

Info. for next crew:

Notable events:
17:05 - 2nd sPHENIX tour completed
17:26 - Line laser work completed; cooling removed
19:08 - DAQ screwed when bringing up for the first time: "LONG LASTING BITSHIFT! CALL A DAQ EXPERT". Experts worked on the DAQ until turnover. Power racks (2W2 and 2E2) powercycled a few times. GTM firmware/hardware reverted to v74, but it is not certain if this is related to the difficulties.

GTM firmware was reverted on Mar/12.
Is there any log about the change on Mar/21?

backup

Joe's QA: Details

TrackingAnalysis / SiliconBcoQA /

osbornjd add readme

Name
..
scripts
Fun4All_SingleStream_Combiner.C
PlotRatios.C
README.md
gl1_makelist.sh
intt_makelist.sh
mvtx_makelist.sh

Link

```
void Fun4All_SingleStream_Combiner(int nEvents = 0,
                                   const int runnumber = 30117,
                                   const string &outdir = "./",
                                   const string& histdir = "./",
                                   const string &type = "beam",
                                   const int neventsper = 100,
                                   const string &dbtag = "ProdA_2024",
                                   const string &input_gl1file = "gl1daq.list",
                                   const string &input_tpcfile00 = "tpc00.list",
                                   const string &input_inttfile00 = "intt4.list",
                                   const string &input_mvtxfile00 = "mvtx0.list",
                                   const string &input_tpotfile = "tpot.list")
{
// GL1 which provides the beam clock reference (if we ran with GL1)
  vector<string> gl1_infile;
  gl1_infile.push_back(input_gl1file);


// MVTX
  vector<string> mvtx_infile;
  mvtx_infile.push_back(input_mvtxfile00);

// INTT
  vector<string> intt_infile;
  intt_infile.push_back(input_inttfile00);

  vector<string> tpc_infile;
  tpc_infile.push_back(input_tpcfile00);

// TPOT
  vector<string> tpot_infile;
  tpot_infile.push_back(input_tpotfile);

  Fun4AllServer *se = Fun4AllServer::instance();
  se->Verbosity(1);
  recoConsts *rc = recoConsts::instance();
  CDBInterface::instance()->Verbosity(1);
  rc->set_StringFlag("CDB_GLOBALTAG", dbtag );
  Fun4AllStreamingInputManager *in = new Fun4AllStreamingInputManager("Comb");
  in->Verbosity(0);

// create and register input managers
  int i = 0;

  for (auto iter : gl1_infile)
  {
    if (isGood(iter))
    {
      SingleGl1PoolInput *gl1_sngl = new SingleGl1PoolInput("GL1_" + to_string(i));
      //    gl1_sngl->Verbosity(3);
      gl1_sngl->AddListFile(iter);
      in->registerStreamingInput(gl1_sngl, InputManagerType::GL1);
      i++;
    }
  }
  i = 0;
```

```
int Fun4AllStreamingInputManager::run(const int /*nevents*/)
{
  int iret = 0;
  if (m_gl1_registered_flag)  // Gl1 first to get the reference
  {
    iret += FillGl1();
  }
  if (m_intt_registered_flag)
  {
    iret += FillIntt();
  }
```

```
    unsigned int refbcobitshift = m_RefBCO & 0x3FU;
    h_refbco_intt->Fill(refbcobitshift);
    bool allpackets = true;
    int allpacketsallfees = 0;
    for (auto &p : m_InttInputVector)
    {
      // this is on a per packet basis
      auto bcl_stack = p->BclkStackMap();
      auto feebclstack = p->getFeeGTMLVL1BCOMap();
      int packet_id = bcl_stack.begin()->first;
      int histo_to_fill = (packet_id % 10) - 1;

      std::set<int> feeidset;
      int fee = 0;
      for (auto &[feeid, gtmbcoset] : feebclstack)
      {
        for (auto &bcl : gtmbcoset)
        {
          auto diff = (m_RefBCO > bcl) ? m_RefBCO - bcl : bcl - m_RefBCO;
          h_bcodiff_intt[histo_to_fill]->Fill(feeid, diff);
          if (diff <= m_intt_bco_range)
          {  // diff is whatever the bco range is set as (2 for triggered, 120 for strobe)
            h_gl1taggedfee_intt[histo_to_fill][fee]->Fill(refbcobitshift);
            feeidset.insert(feeid);
            // this fee was tagged, go to the next one
```

Joe's QA: Details

Fun4AllStreamingInputManager.cc

`FillIntt` function

```
int Fun4AllStreamingInputManager::FillIntt()
{
  int iret = FillInttPool();
  if (iret)
  {
    return iret;
  }

  // unsigned int alldone = 0;
  //     std::cout << "stashed intt BCOs: " << m_InttRawHitMap.size() << std::endl;
  InttRawHitContainer *inttcont = findNode::getClass<InttRawHitContainer>(m_topNode, "INTTRAWHIT");
  if (!inttcont)
  {
    inttcont = findNode::getClass<InttRawHitContainer>(m_topNode, (*(m_InttInputVector.begin()))->getHitContainerName());
    if (!inttcont)
    {
      std::cout << PHWHERE << "Could not find InttRawHitContainer node in topNode" << std::endl;
      gSystem->Exit(1);
      exit(1);
    }
  }
  //  std::cout << "before filling m_InttRawHitMap size: " <<  m_InttRawHitMap.size() << std::endl;
  // !m_InttRawHitMap.empty() is implicitely handled and the check is expensive
  // FillInttPool() contains this check already and will return non zero
  // so here m_InttRawHitMap will always contain entries
  uint64_t select_crossings = m_intt_bco_range;
  if (m_RefBCO == 0)
  {
    m_RefBCO = m_InttRawHitMap.begin()->first;
  }
  select_crossings += m_RefBCO;
  if (Verbosity() > 2)
  {
    std::cout << "select INTT crossings"
              << " from 0x" << std::hex << m_RefBCO - m_intt_negative_bco
              << " to 0x" << select_crossings - m_intt_negative_bco
              << " for ref BCO " << m_RefBCO
              << std::dec << std::endl;
  }

  while (m_InttRawHitMap.begin()->first < m_RefBCO - m_intt_negative_bco)
  {
    if (Verbosity() > 2)
    {
      std::cout << "Intt BCO: 0x" << std::hex << m_InttRawHitMap.begin()->first
                << " corrected for negative offset: 0x" << m_InttRawHitMap.begin()->first + m_intt_negative_bco
                << " smaller than GL1 BCO: 0x" << m_RefBCO
                << " corrected for range: 0x" << select_crossings
                << std::dec << " diff: " << (m_RefBCO - m_InttRawHitMap.begin()->first)
                << ", ditching this bco" << std::dec << std::endl;
    }
    for (auto iter : m_InttInputVector)
    {
      iter->CleanupUsedPackets(m_InttRawHitMap.begin()->first);
      iter->clearPacketBClkStackMap(m_InttRawHitMap.begin()->first);
      iter->clearFeeGTML1BCOMap(m_InttRawHitMap.begin()->first);
    }

    m_InttRawHitMap.begin()->second.InttRawHitVector.clear();
    m_InttRawHitMap.erase(m_InttRawHitMap.begin());

    iret = FillInttPool();
    if (iret)
    {
      return iret;
    }
  }
  unsigned int refbcobitshift = m_RefBCO & 0x3FU;
  h_refbco_intt->Fill(refbcobitshift);
  bool allpackets = true;
  int allpacketsallfees = 0;
  for (auto &p : m_InttInputVector)
  {
    // this is on a per packet basis
    auto bcl_stack = p->BclkStackMap();
    auto feebclstack = p->getFeeGTML1BCOMap();
    int packet_id = bcl_stack.begin()->first;
    int histo_to_fill = (packet_id % 10) - 1;

    std::set<int> feeidset;
    int fee = 0;
    for (auto &[feeid, gtmbcoset] : feebclstack)
    {
      for (auto &bcl : gtmbcoset)
      {
        auto diff = (m_RefBCO > bcl) ? m_RefBCO - bcl : bcl - m_RefBCO;
        h_bcodiff_intt[histo_to_fill]->Fill(feeid, diff);
        if (diff <= m_intt_bco_range)
        {  // diff is whatever the bco range is set as (2 for triggered, 120 for strobe)
          h_gl1taggedfee_intt[histo_to_fill][fee]->Fill(refbcobitshift);
          feeidset.insert(feeid);
          // this fee was tagged, go to the next one
          break;
        }
      }
      fee++;
    }

    if (feeidset.size() == 14)
    {
      allpacketsallfees++;
      h_taggedAllFees_intt[histo_to_fill]->Fill(refbcobitshift);
    }
    feeidset.clear();
    bool thispacket = false;

    for (auto &[packetid, gtmbcoset] : bcl_stack)
    {
      for (auto &gtmbco : gtmbcoset)
      {
        auto diff = (m_RefBCO > gtmbco) ? m_RefBCO - gtmbco : gtmbco - m_RefBCO;
        if (diff < m_intt_bco_range)
        {
          thispacket = true;
          h_gl1tagged_intt[histo_to_fill]->Fill(refbcobitshift);
          break;
        }
      }
    }

    if (thispacket == false)
    {
      allpackets = false;
    }
  }
  if (allpackets)
  {
    h_taggedAll_intt->Fill(refbcobitshift);
  }
  if (allpacketsallfees == (int) m_InttInputVector.size())
  {
    h_taggedAllFee_intt->Fill(refbcobitshift);
  }

  //  std::cout << "Checking diff " << (m_InttRawHitMap.begin()->first - (select_crossings - m_intt_negative_bco)) << std::endl;
  while (m_InttRawHitMap.begin()->first <= select_crossings - m_intt_negative_bco)
  {
    for (auto intthititer : m_InttRawHitMap.begin()->second.InttRawHitVector)
    {
      if (Verbosity() > 1)
      {
        std::cout << "Adding intt hit with bco 0x" << std::hex
                  << intthititer->get_bco() << std::dec << std::endl;
        //        intthititer->identify();
      }
      inttcont->AddHit(intthititer);
    }
    for (auto iter : m_InttInputVector)
    {
      iter->CleanupUsedPackets(m_InttRawHitMap.begin()->first);
      if (m_intt_negative_bco < 2)  // triggered mode
      {
        iter->clearPacketBClkStackMap(m_InttRawHitMap.begin()->first);
        iter->clearFeeGTML1BCOMap(m_InttRawHitMap.begin()->first);
      }
    }
    m_InttRawHitMap.begin()->second.InttRawHitVector.clear();
    m_InttRawHitMap.erase(m_InttRawHitMap.begin());
    if (m_InttRawHitMap.empty())
    {
      break;
    }
  }
  return 0;
}
```

Joe's QA: Details

Fun4AllStreamingInputManager.cc

FillIntt function

```
642  int Fun4AllStreamingInputManager::FillIntt()
643  {
644    int iret = FillInttPool();
645    if (iret)
646    {
647      return iret;
648    }
649
650    // unsigned int alldone = 0;
651    //     std::cout << "stashed intt BCOs: " << m_InttRawHitMap.size() << std::endl;
652    InttRawHitContainer *inttcont = findNode::getClass<InttRawHitContainer>(m_topNode, "INTTRAWHIT");
653    if (!inttcont)
654    {
655      inttcont = findNode::getClass<InttRawHitContainer>(m_topNode, (*(m_InttInputVector.begin()))->getHitContainerName());
656      if (!inttcont)
657      {
658        std::cout << PHWHERE << "Could not find InttRawHitContainer node in topNode" << std::endl;
659        gSystem->Exit(1);
660        exit(1);
661      }
662    }
663    //  std::cout << "before filling m_InttRawHitMap size: " <<  m_InttRawHitMap.size() << std::endl;
664    // !m_InttRawHitMap.empty() is implicitely handled and the check is expensive
665    // FillInttPool() contains this check already and will return non zero
666    // so here m_InttRawHitMap will always contain entries
667    uint64_t select_crossings = m_intt_bco_range;
668    if (m_RefBCO == 0)
669    {
670      m_RefBCO = m_InttRawHitMap.begin()->first;
671    }
672    select_crossings += m_RefBCO;
673    if (Verbosity() > 2)
674    {
675      std::cout << "select INTT crossings"
676                << " from 0x" << std::hex << m_RefBCO - m_intt_negative_bco
677                << " to 0x" << select_crossings - m_intt_negative_bco
678                << " for ref BCO " << m_RefBCO
679                << std::dec << std::endl;
680    }
681
682    while (m_InttRawHitMap.begin()->first < m_RefBCO - m_intt_negative_bco)
683    {
684      if (Verbosity() > 2)
685      {
686        std::cout << "Intt BCO: 0x" << std::hex << m_InttRawHitMap.begin()->first
687                  << " corrected for negative offset: 0x" << m_InttRawHitMap.begin()->first + m_intt_negative_bco
688                  << " smaller than GL1 BCO: 0x" << m_RefBCO
689                  << " corrected for range: 0x" << select_crossings
690                  << std::dec << " diff: " << (m_RefBCO - m_InttRawHitMap.begin()->first)
691                  << ", ditching this bco" << std::dec << std::endl;
692      }
693      for (auto iter : m_InttInputVector)
694      {
695        iter->CleanupUsedPackets(m_InttRawHitMap.begin()->first);
696        iter->clearPacketBClkStackMap(m_InttRawHitMap.begin()->first);
697        iter->clearFeeGTML1BCOMap(m_InttRawHitMap.begin()->first);
698      }
699
700      m_InttRawHitMap.begin()->second.InttRawHitVector.clear();
701      m_InttRawHitMap.erase(m_InttRawHitMap.begin());
702
703      iret = FillInttPool();
704      if (iret)
705      {
706        return iret;
707      }
708    }
```

```
710    unsigned int refbcobitshift = m_RefBCO & 0x3FU;
711    h_refbco_intt->Fill(refbcobitshift);
712    bool allpackets = true;
713    int allpacketsallfees = 0;
714    for (auto &p : m_InttInputVector)
715    {
716      // this is on a per packet basis
717      auto bcl_stack = p->BclkStackMap();
718      auto feebclstack = p->getFeeGTML1BCOMap();
719      int packet_id = bcl_stack.begin()->first;
720      int histo_to_fill = (packet_id % 10) - 1;
721
722      std::set<int> feeidset;
723      int fee = 0;
724      for (auto &[feeid, gtmbcoset] : feebclstack)
725      {
726        for (auto &bcl : gtmbcoset)
727        {
728          auto diff = (m_RefBCO > bcl) ? m_RefBCO - bcl : bcl - m_RefBCO;
729          h_bcodiff_intt[histo_to_fill]->Fill(feeid, diff);
730          if (diff <= m_intt_bco_range)
731          {  // diff is whatever the bco range is set as (2 for triggered, 120 for strobe)
732            h_gl1taggedfee_intt[histo_to_fill][fee]->Fill(refbcobitshift);
733            feeidset.insert(feeid);
734            // this fee was tagged, go to the next one
735            break;
736          }
737        }
738        fee++;
739      }
740
741      if (feeidset.size() == 14)
742      {
743        allpacketsallfees++;
744        h_taggedAllFees_intt[histo_to_fill]->Fill(refbcobitshift);
745      }
746      feeidset.clear();
747      bool thispacket = false;
748
749      for (auto &[packetid, gtmbcoset] : bcl_stack)
750      {
751        for (auto &gtmbco : gtmbcoset)
752        {
753          auto diff = (m_RefBCO > gtmbco) ? m_RefBCO - gtmbco : gtmbco - m_RefBCO;
754          if (diff < m_intt_bco_range)
755          {
756            thispacket = true;
757            h_gl1tagged_intt[histo_to_fill]->Fill(refbcobitshift);
758            break;
759          }
760        }
761      }
762
763      if (thispacket == false)
764      {
765        allpackets = false;
766      }
767    }
768    if (allpackets)
769    {
770      h_taggedAll_intt->Fill(refbcobitshift);
771    }
772    if (allpacketsallfees == (int) m_InttInputVector.size())
773    {
774      h_taggedAllFee_intt->Fill(refbcobitshift);
775    }
776
```

GL1 reference hist

SingleInttPoolInput.cc

FillPool function

```
for (int j = 0; j < num_hits; j++)
{
  uint64_t gtm_bco = pool->iValue(j, "BCO");
  if (gtm_bco < minBCO)
  {
    // std::cout << "dropping hit with bco 0x" << std::hex
    //           << gtm_bco << ", min bco: 0x" << minBCO
    //           << std::endl;
    continue;
  }
  auto newhit = std::make_unique<InttRawHitv2>();
  int FEE = pool->iValue(j, "FEE");
  newhit->set_packetid(pool->getIdentifier());
  newhit->set_fee(FEE);
  newhit->set_bco(gtm_bco);
  newhit->set_adc(pool->iValue(j, "ADC"));
  newhit->set_amplitude(pool->iValue(j, "AMPLITUDE"));
  newhit->set_chip_id(pool->iValue(j, "CHIP_ID"));
  newhit->set_channel_id(pool->iValue(j, "CHANNEL_ID"));
  newhit->set_word(pool->iValue(j, "DATAWORD"));
  newhit->set_FPHX_BCO(pool->iValue(j, "FPHX_BCO"));
  newhit->set_full_FPHX(pool->iValue(j, "FULL_FPHX"));
  newhit->set_full_ROC(pool->iValue(j, "FULL_ROC"));
  newhit->set_event_counter(pool->iValue(j, "EVENT_COUNTER"));
  gtm_bco += m_Rollover[FEE];

  if (gtm_bco < m_PreviousClock[FEE])
  {
    m_Rollover[FEE] += 0x10000000000;
    gtm_bco += 0x10000000000;  // rollover makes sure our bclks are ascending even if we roll over the 40 bit counter
  }
  m_PreviousClock[FEE] = gtm_bco;
  m_BeamClockFEE[gtm_bco].insert(FEE);
  m_FEEBclkMap[FEE] = gtm_bco;
  if (Verbosity() > 2)
  {
    std::cout << "evtno: " << EventSequence
              << ", hits: " << j
              << ", nr_hits: " << num_hits
              << ", FEE: " << FEE
              << ", bco: 0x" << std::hex << gtm_bco << std::dec
              << ", min bco: 0x" << std::hex << minBCO << std::dec
              << ", channel: " << newhit->get_channel_id()
              << ", evt_counter: " << newhit->get_event_counter() << std::endl;
  }
  if (StreamingInputManager())
  {
    StreamingInputManager()->AddInttRawHit(gtm_bco, newhit.get());
  }
  m_InttRawHitMap[gtm_bco].push_back(newhit.release());
```

intt_pool

Joe's QA: Details

`FillIntt` function

```
int Fun4AllStreamingInputManager::FillIntt()
{
  int iret = FillInttPool();
  if (iret)
  {
    return iret;
  }

  // unsigned int alldone = 0;
  //     std::cout << "stashed intt BCOs: " << m_InttRawHitMap.size() << std::endl;
  InttRawHitContainer *inttcont = findNode::getClass<InttRawHitContainer>(m_topNode, "INTTRAWHIT");
  if (!inttcont)
  {
    inttcont = findNode::getClass<InttRawHitContainer>(m_topNode, (*(m_InttInputVector.begin()))->getHitContainerName());
    if (!inttcont)
    {
      std::cout << PHWHERE << "Could not find InttRawHitContainer node in topNode" << std::endl;
      gSystem->Exit(1);
      exit(1);
    }
  }
  //  std::cout << "before filling m_InttRawHitMap size: " <<  m_InttRawHitMap.size() << std::endl;
  // !m_InttRawHitMap.empty() is implicitely handled and the check is expensive
  // FillInttPool() contains this check already and will return non zero
  // so here m_InttRawHitMap will always contain entries
  uint64_t select_crossings = m_intt_bco_range;
  if (m_RefBCO == 0)
  {
    m_RefBCO = m_InttRawHitMap.begin()->first;
  }
  select_crossings += m_RefBCO;
  if (Verbosity() > 2)
  {
    std::cout << "select INTT crossings"
              << " from 0x" << std::hex << m_RefBCO - m_intt_negative_bco
              << " to 0x" << select_crossings - m_intt_negative_bco
              << " for ref BCO " << m_RefBCO
              << std::dec << std::endl;
  }

  while (m_InttRawHitMap.begin()->first < m_RefBCO - m_intt_negative_bco)
  {
    if (Verbosity() > 2)
    {
      std::cout << "Intt BCO: 0x" << std::hex << m_InttRawHitMap.begin()->first
                << " corrected for negative offset: 0x" << m_InttRawHitMap.begin()->first + m_intt_negative_bco
                << " smaller than GL1 BCO: 0x" << m_RefBCO
                << " corrected for range: 0x" << select_crossings
                << std::dec << " diff: " << (m_RefBCO - m_InttRawHitMap.begin()->first)
                << ", ditching this bco" << std::dec << std::endl;
    }
    for (auto iter : m_InttInputVector)
    {
      iter->CleanupUsedPackets(m_InttRawHitMap.begin()->first);
      iter->clearPacketBClkStackMap(m_InttRawHitMap.begin()->first);
      iter->clearFeeGTML1BCOMap(m_InttRawHitMap.begin()->first);
    }

    m_InttRawHitMap.begin()->second.InttRawHitVector.clear();
    m_InttRawHitMap.erase(m_InttRawHitMap.begin());

    iret = FillInttPool();
    if (iret)
    {
      return iret;
    }
  }
```

```
  unsigned int refbcobitshift = m_RefBCO & 0x3FU;
  h_refbco_intt->Fill(refbcobitshift);
  bool allpackets = true;
  int allpacketsallfees = 0;
  for (auto &p : m_InttInputVector)
  {
    // this is on a per packet basis
    auto bcl_stack = p->BclkStackMap();
    auto feebclstack = p->getFeeGTML1BCOMap();
    int packet_id = bcl_stack.begin()->first;
    int histo_to_fill = (packet_id % 10) - 1;

    std::set<int> feeidset;
    int fee = 0;
    for (auto &[feeid, gtmbcoset] : feebclstack)
    {
      for (auto &bcl : gtmbcoset)
      {
        auto diff = (m_RefBCO > bcl) ? m_RefBCO - bcl : bcl - m_RefBCO;
        h_bcodiff_intt[histo_to_fill]->Fill(feeid, diff);
        if (diff <= m_intt_bco_range)
        {  // diff is whatever the bco range is set as (2 for triggered, 12 for strobe)
          h_gl1taggedfee_intt[histo_to_fill][fee]->Fill(refbcobitshift);
          feeidset.insert(feeid);
          // this fee was tagged, go to the next one
          break;
        }
      }
      fee++;
    }

    if (feeidset.size() == 14)
    {
      allpacketsallfees++;
      h_taggedAllFees_intt[histo_to_fill]->Fill(refbcobitshift);
    }
    feeidset.clear();
    bool thispacket = false;

    for (auto &[packetid, gtmbcoset] : bcl_stack)
    {
      for (auto &gtmbco : gtmbcoset)
      {
        auto diff = (m_RefBCO > gtmbco) ? m_RefBCO - gtmbco : gtmbco - m_RefBCO;
        if (diff < m_intt_bco_range)
        {
          thispacket = true;
          h_gl1tagged_intt[histo_to_fill]->Fill(refbcobitshift);
          break;
        }
      }
    }

    if (thispacket == false)
    {
      allpackets = false;
    }
  }
  if (allpackets)
  {
    h_taggedAll_intt->Fill(refbcobitshift);
  }
  if (allpacketsallfees == (int) m_InttInputVector.size())
  {
    h_taggedAllFee_intt->Fill(refbcobitshift);
  }
```

INTT hists

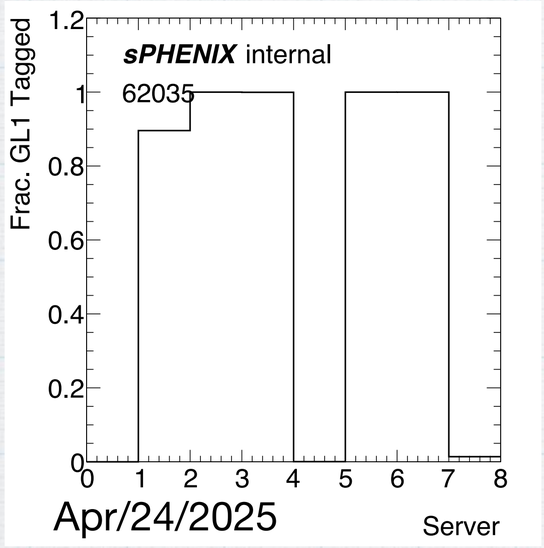
If BCO of all 14 half-ladders has small difference from the reference BCO value, it's counted. The reference BCO is BCO of the first InttRawHit. If event matching of GL1 and INTT (intt_pool?) is perfect, the reference BCO is GL1 BCO.

```
  //  std::cout << "Checking diff " << (m_InttRawHitMap.begin()->first - (select_crossings - m_intt_negative_bco)) << std::endl;
  while (m_InttRawHitMap.begin()->first <= select_crossings - m_intt_negative_bco)
  {
    for (auto intthititer : m_InttRawHitMap.begin()->second.InttRawHitVector)
    {  GL1 reference hist
      if (Verbosity() > 1)
      {
        std::cout << "Adding intt hit with bco 0x" << std::hex
                  << intthititer->get_bco() << std::dec << std::endl;
        //        intthititer->identify();
      }
      inttcont->AddHit(intthititer);
    }
    for (auto iter : m_InttInputVector)
    {
      iter->CleanupUsedPackets(m_InttRawHitMap.begin()->first);
      if (m_intt_negative_bco < 2)  // triggered mode
      {
        iter->clearPacketBClkStackMap(m_InttRawHitMap.begin()->first);
        iter->clearFeeGTML1BCOMap(m_InttRawHitMap.begin()->first);
      }
    }
    m_InttRawHitMap.begin()->second.InttRawHitVector.clear();
    m_InttRawHitMap.erase(m_InttRawHitMap.begin());
    if (m_InttRawHitMap.empty())
    {
      break;
    }
  }
  return 0;
}
```

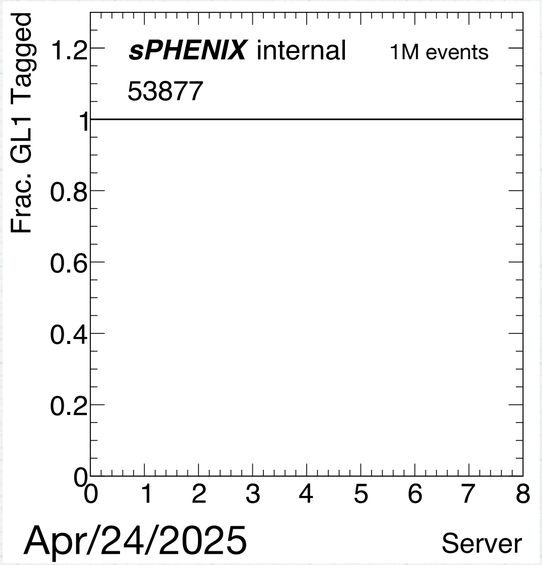
from INTT group

- GL1-INTT matching study was also done last year, and INTT found to be good. The detector condition/configuration has not been changed since last year.
- The only difference from last year is condition of hot channel masking. We don’t expect this is the cause of low matching ratio, but it’s good to apply the same mask. We will try if there is no objection.
- I’m trying another way to confirm the matching with pmonitor. This was done by Sam and Jaebeom last year. It’s ongoing.
- Consistency check within INTT is also good to try.

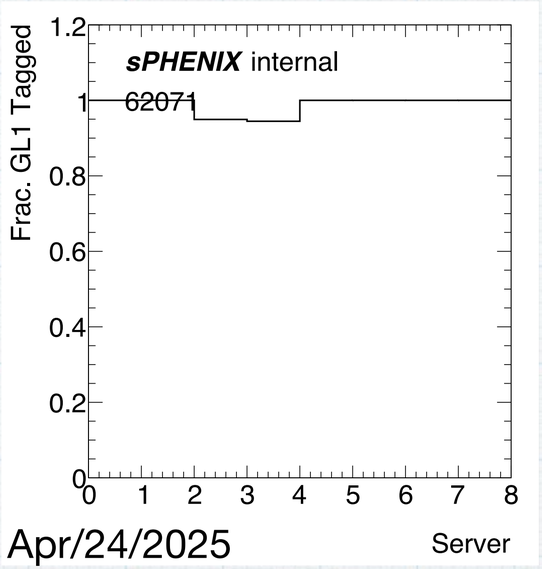
Joe’s study



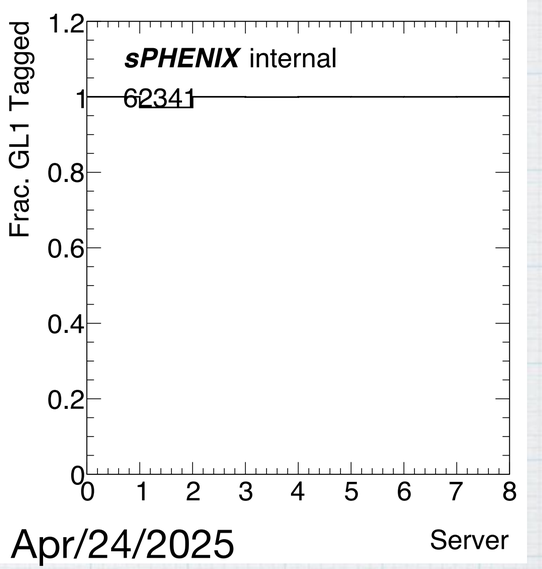
Apr/24/2025
DAQ meeting



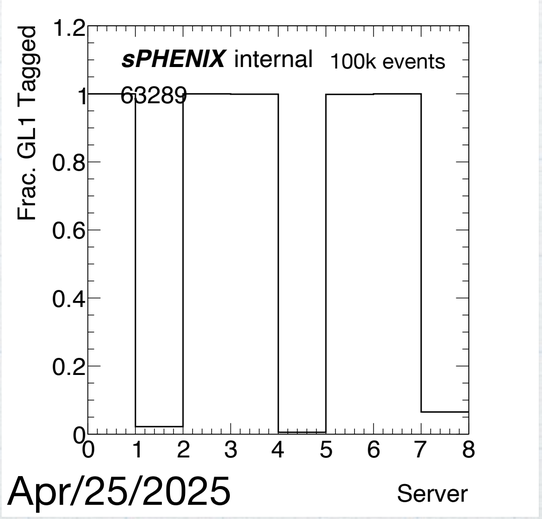
Apr/24/2025
DAQ meeting



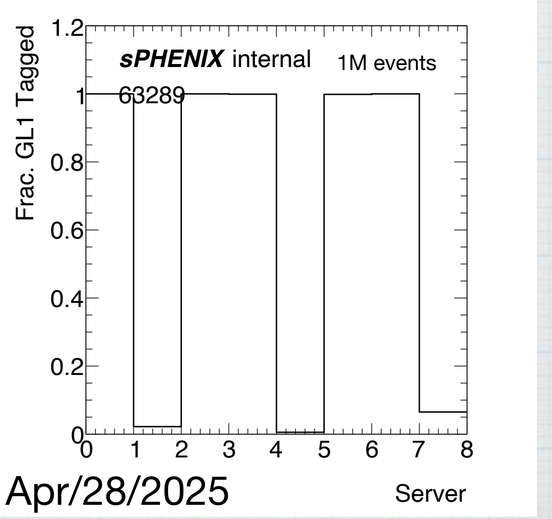
Apr/24/2025
DAQ meeting



Apr/24/2025
DAQ meeting



Apr/25/2025



Apr/28/2025