GLAST Large Area Telescope

LAT Deadtime

Warren Focke
SLAC
I&T Science Verification Analysis and Calibration
Engineering Physicist
focke@slac.stanford.edu
650-926-4713

Outline

- EvtTicks reminder
  - & example
- Deadtime distributions
  - not just upper limits

Reminder - EvtTicks

- Currently the best measure of event time we have.
  - Assigned at trigger time.
    - Other times (for example, EvtSecond, EvtNanoSecond) are assigned much (and upredictably) later.
  - Running count of LAT ticks (nominally 50 ns) since shortly (<128 s) before run began.
- Variable EvtTicks is in SVAC tuple.
  - Stored in a double, but values are integers.
  - Calculated from GemTriggerTime, GemOnePpsSeconds, GemOnePpsTime, EvtSecond, EvtNanoSecond (all in SVAC tuple).
  - Details in extra slides at end of talk
- Will need a new algorithm when we get GPS time.

Example

- Can be used to get precise time between any two events
  - they do have to be from the same run
- E.g., suppose you want to look at times between CAL-only events
  - Apply a cut on GemConditionWord
  - GemDeltaEventTime is then not useful, since it gives the time since the last event that triggered, not the last one that passed the cut
  - but $EvtTicks_i$ – $EvtTicks_{i-1}$ is still valid

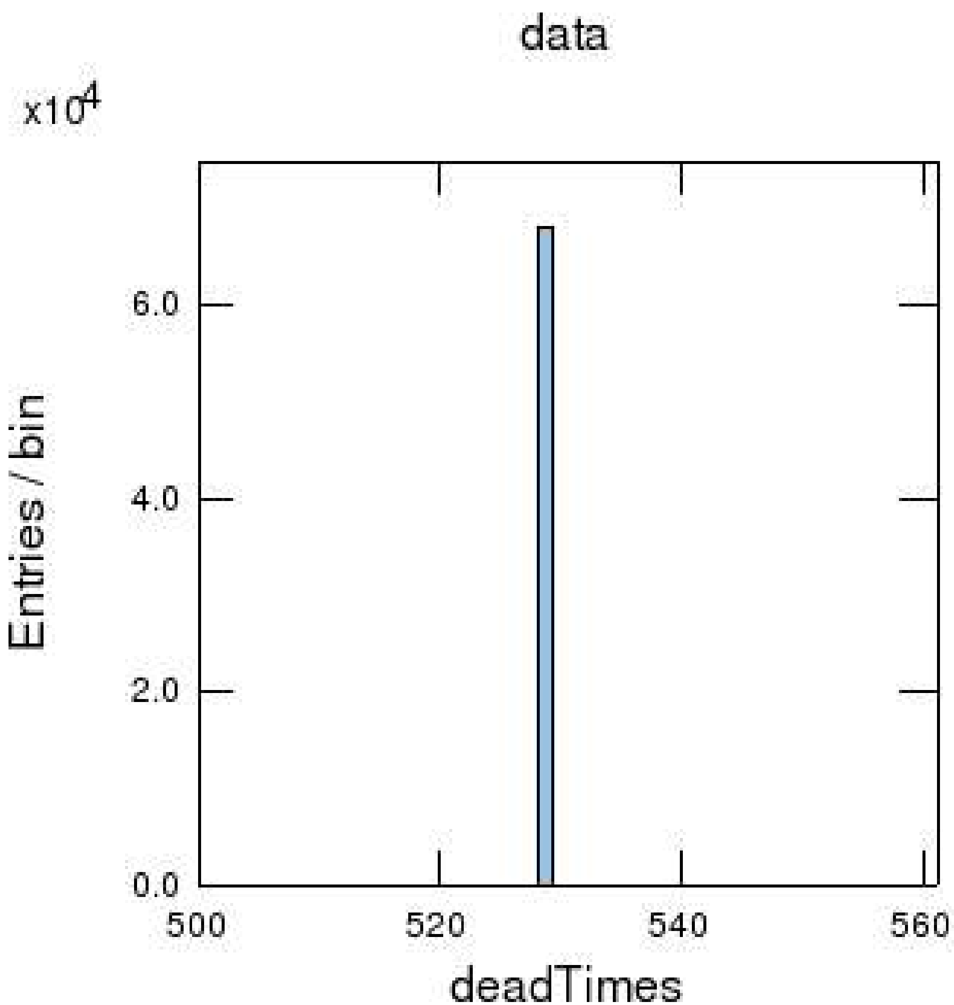
More SVAC tuple variables

- All measured in LAT ticks (50 ns)
- GemDeltaEventTime (GDET)
  - time since last event
    - only if triggered & read out
  - saturates
  - 16 bits = 3.3ms
- GemLiveTime (LIVE)
  - only increments when LAT not busy
  - running counter (rolls over)
  - 25 bits = 1.7s

Deadtime Algorithm

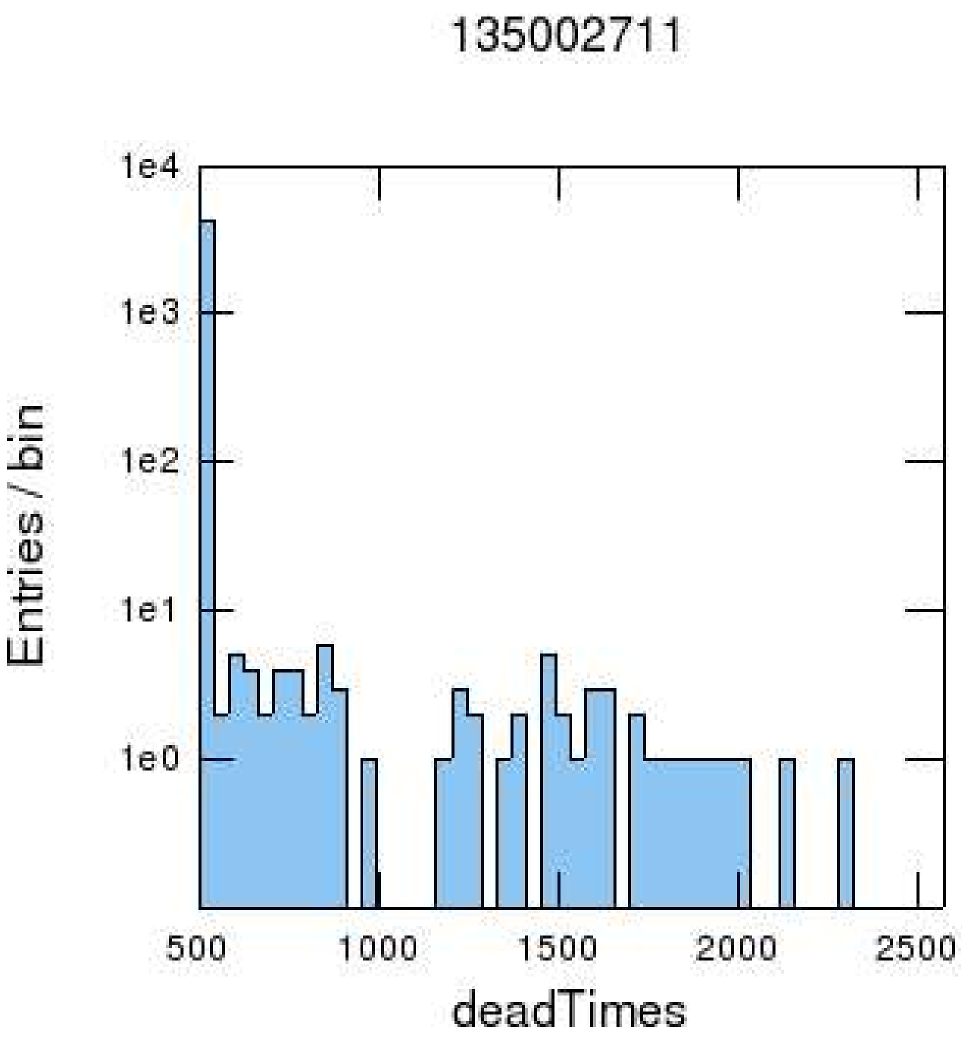
- $DLT_i = LIVE_i - LIVE_{i-1}$
  - + 2**25 if < 0
- Apply cut
  - GDET == delta EvtTicks
    - no missing events in between
  - && GDET < 2**16 - 1
    - not saturated
- $DeadTime_i = GDET_i - DLT_i$
- Make histogram

"Flight-Like" Deadtime



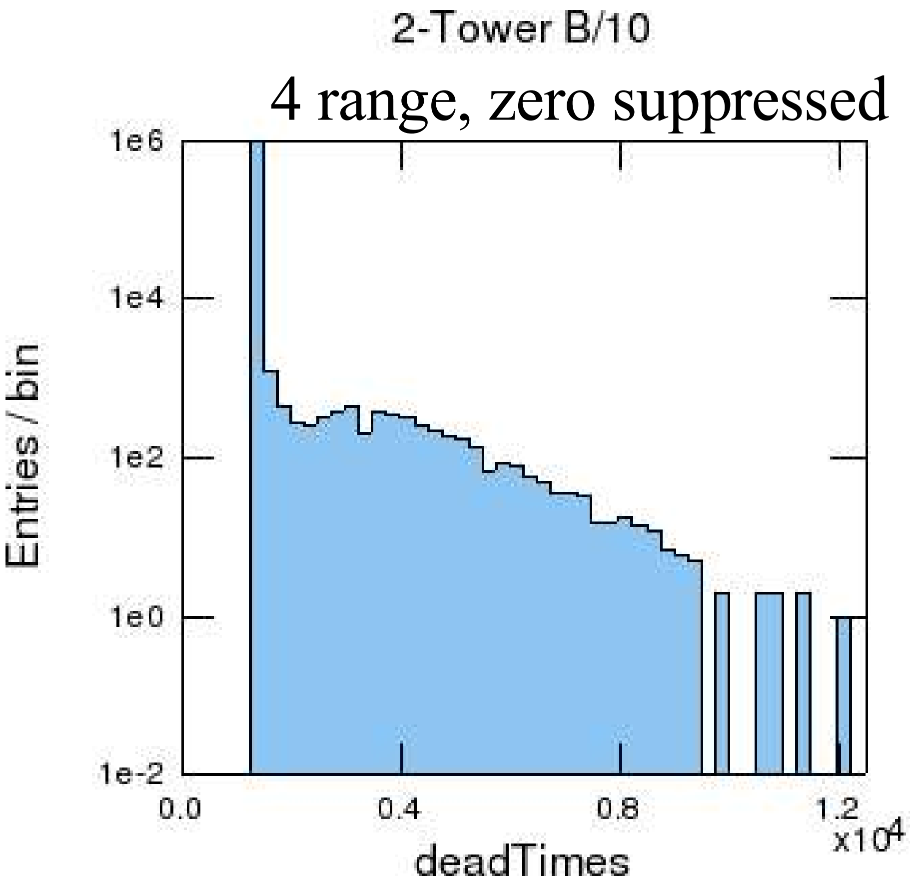
- 2 Tower Baseline (1/1) run
  - 135002052
- Observed deadtimes all == 529 ticks (26.45μs)
  - as predicted
    - Sweet!
  - consistent with observed minimum event separation of 530
- This is also true for 6-tower B/2 runs
- and 2-tower external trigger

DOH!

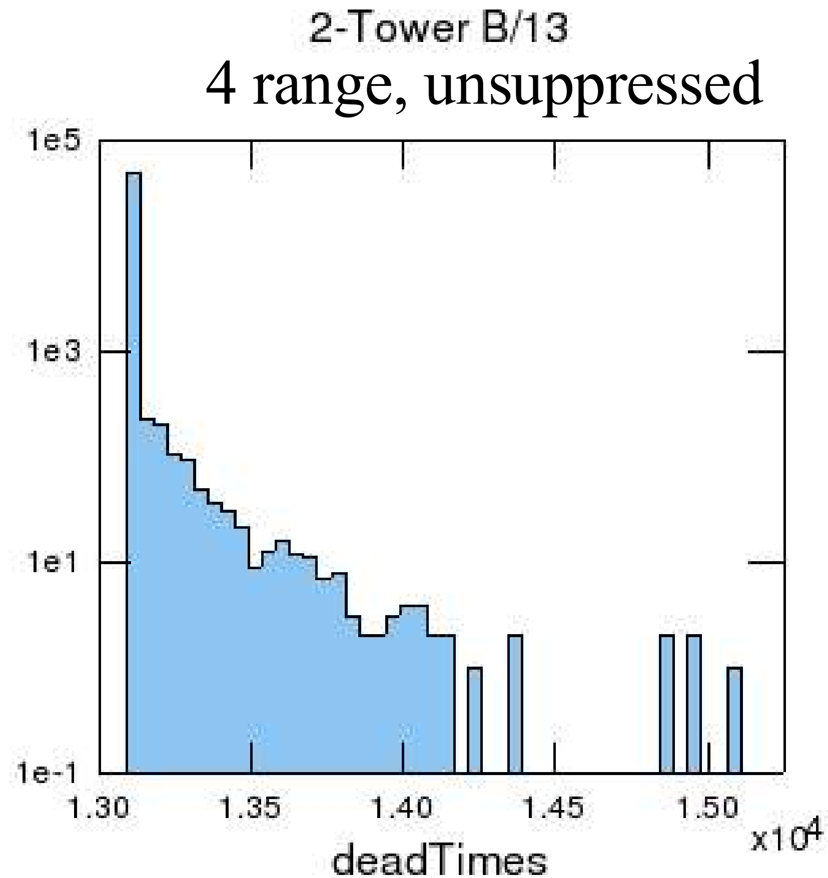


- 4 Tower Baseline run
  - 135002711
- 4 tower data have incorrect CAL LAC thresholds
  - causes too much data to be read out
    - which causes more deadtime
- Minimum is still 529

Non-Flightlike Runs



4 range, zero suppressed

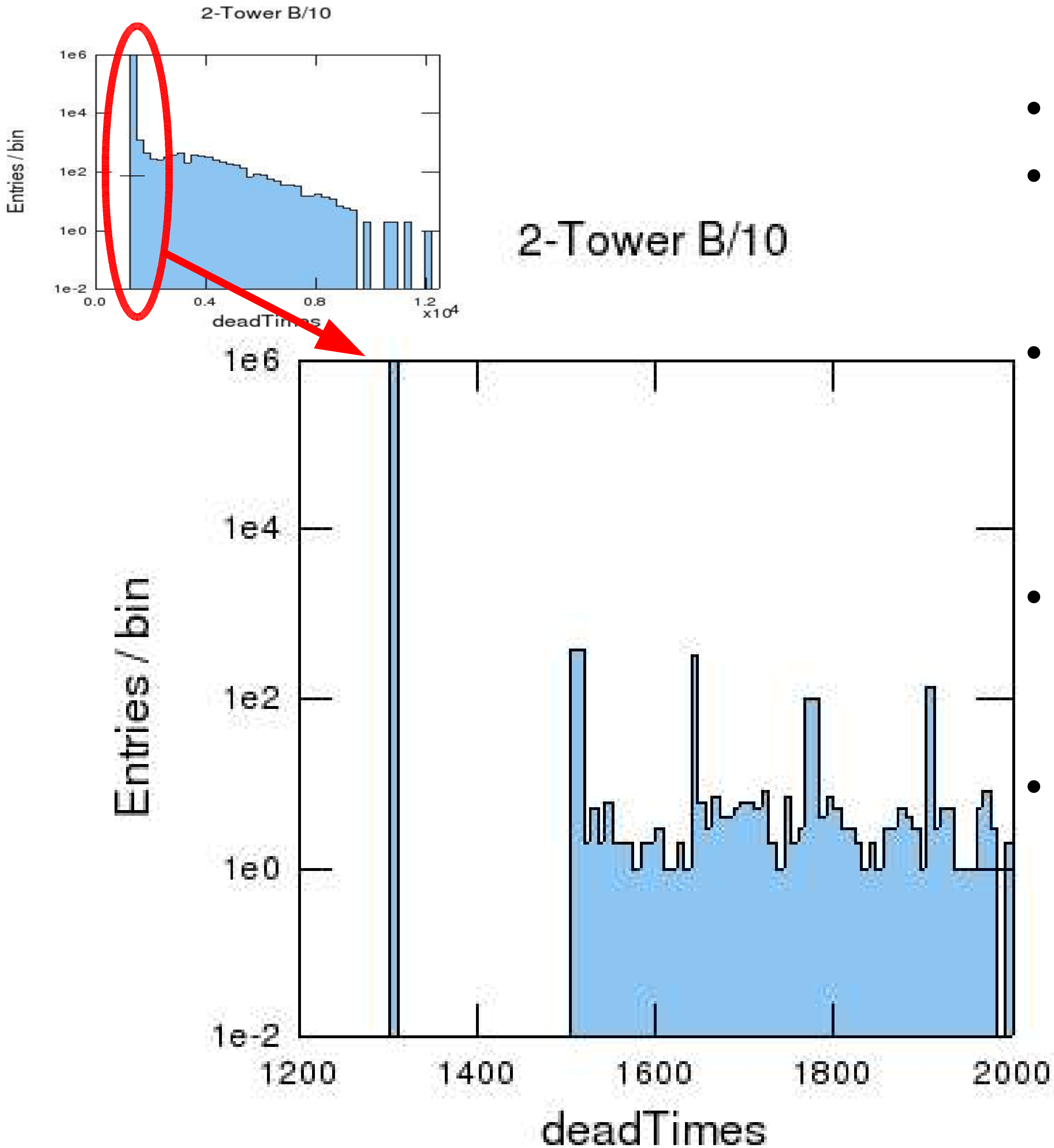


4 range, unsuppressed

Minimum deadtime = 1308
Consistent with prediction and observed minimum event separation of 1309

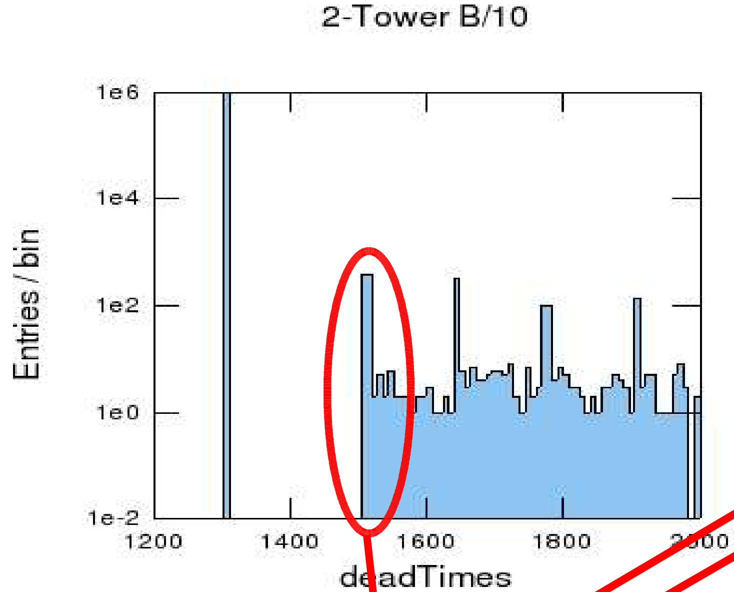
Minimum deadtime = 13128
Consistent with observed minimum event separation of 13129

Quantization



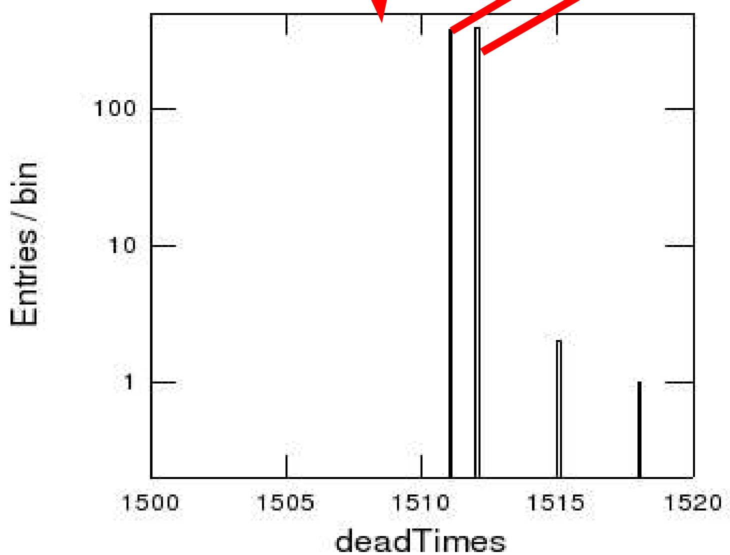
- **4 range, zero suppressed**
- **Gap of 203 ticks between main peak & next-higher value**
- **Smaller peaks are separated by 132 ticks**
  - **This is the time required to transmit 4 logs**
- **Smaller peaks are 2 ticks wide**
  - **see next page**
- **This is not seen in B/13 (4 range, unsuppressed)**
  - **probably "washed out" by 10x longer times**

Double Peaks



Delays from trigger TACK to shaper hold

CAL (ticks (ns))		TKR (ticks (ns))	
Tower	Delay	Tower	Delay
0	44 (2200ns)	0	0 (0ns)
4	45 (2250ns)	4	0 (0ns)



- Double peak seems due to different values of CAL TACK delay in different towers
- Main peak would be double, too, but both towers always contribute at least that much deadtime, so the longer one wins
- 6 tower runs seem mostly consistent with this interpretation
  - but a bit odd
    - but they're screwy anyway

Conclusion

- Deadtime is behaving as expected for flight-like runs
- Configuration errors cause unexpected behavior

GEM Timing Variables (in SVAC tuple)

- LAT timebase is a running counter of ticks (50ns)
  - 25 bits, rolls on overflow (1.67 s)
- GemTriggerTime samples timebase at window close time
- GemOnePpsTime samples timebase when 1PPS signal received
- GemOnePpsSeconds is incremented on 1PPS signal
  - 7 bits, rolls on overflow (128 s)
- Timebase can overflow between 1PPS and event
  - But only once, so we can detect it:
    - GemTriggerTime < GemOnePpsTime
- GemOnePpsSeconds overflows every 128 s
  - Not likely to roll more than once between events
    - But if it does we can't detect it from GEM variables
  - Can use other timestamps to detect multiple overflows

Coarser Timestamps (in SVAC tuple)

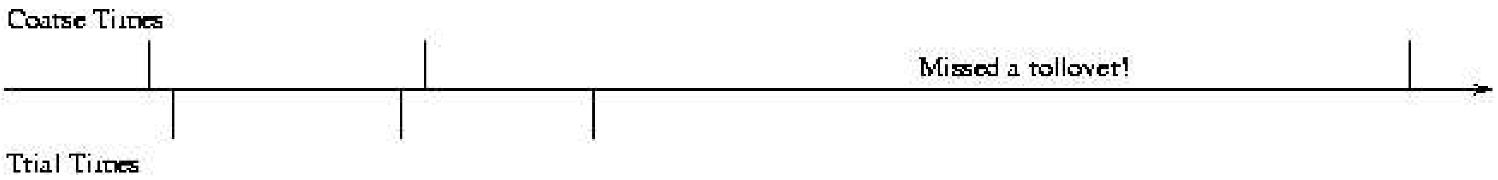
- EvtSecond, EvtNanoSecond come from vxWorks realtime clock (RTC)
  - Updated at 50 Hz
- EvtUpperTime, EvtLowerTime come from SBC CPU cycle counter
  - Updated at ~16 Mhz
  - But 1/60e-9 is closer
  - But we don't really know for sure, and even if we did, it varies by 1 part in ~1e6 (http://www-glast.slac.stanford.edu/IntegrationTest/Weekly%20Minutes/2004-02-12/EMTiming.ppt)
  - Sampled at event build time, not trigger time
    - Queuing can have odd effects

First 2 Tries

- Try to calculate when GemOnePpsSeconds will roll over based on event time using seconds/nanoseconds or upper/lower
  - This is folly
  - Don't know the offsets between the time streams, or even their relative rates, well enough to predict rollovers down to the event
- Try to use long gaps (> 128 s) in seconds/nanoseconds or upper/lower
  - Better, but still doesn't always work
  - Can give spurious rollovers for 64 < gaps < 128 s
  - Coarseness of other timestamps means you can't make an exact cutoff, and there's always a chance of a long separation sneaking into the uncertain region

Third Try

- Use GemOnePpsSeconds, GemOnePpsTime and GemTriggerTime to make trial timestamps, based on assumption that obvious rollovers are the only ones.
  - see next slide
- Compare delta times between events for trial times with deltas from coarser timestamps
- Differences should be within 10-20 ms, unless we missed a PPS rollover
  - Then they will cluster around multiples of 128 s
- Correct trial times if we missed any rollovers
  - Add an appropriate multiple of 128 s (round the difference between deltas to nearest multiple of 128) to all events after the missed roll



Details

- $trialTime_i$ = (nPpsRoll * 128 + $OnePpsSeconds_{last}$) * 20e6 + ($TriggerTime_i$ – $OnePpsTime_{last}$)
  - correct for obvious rollovers
    - $OnePpsSeconds_{last}$ < $OnePpsSeconds_{last-1}$
      - nPpsRoll += 1
    - $TriggerTime_i$ < $OnePpsTime_{last}$
      - TriggerTime += 2**25
- This assumes that $OnePpsTime_i$ – $OnePpsTime_{i-1}$ == 20e6
  - currently true, OnePps signal is faked from LAT clock
  - won't be true (?) when we get a GPS