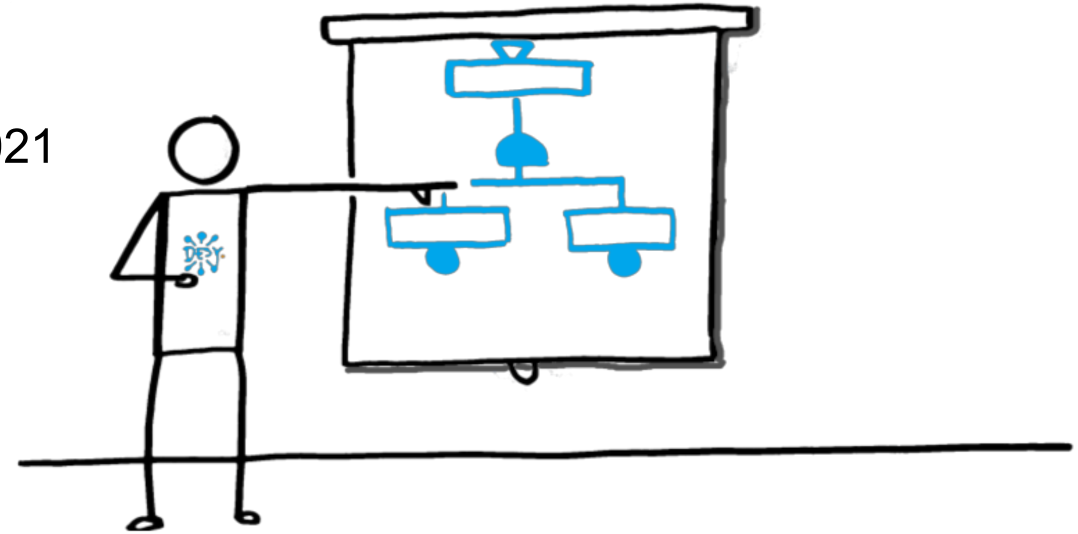
The Trip Event Logger for Online Fault Diagnosis at the European XFEL

Overview and Current Status

Jan Timm

IPAC 2021 Brazil - Hamburg, May the 26th 2021



HELMHOLTZ RESEARCH FOR GRAND CHALLENGES

DESY.

Motivation, Intentions and Goals

Faults mean downtime, identifying and understanding faults can help increase runtime.

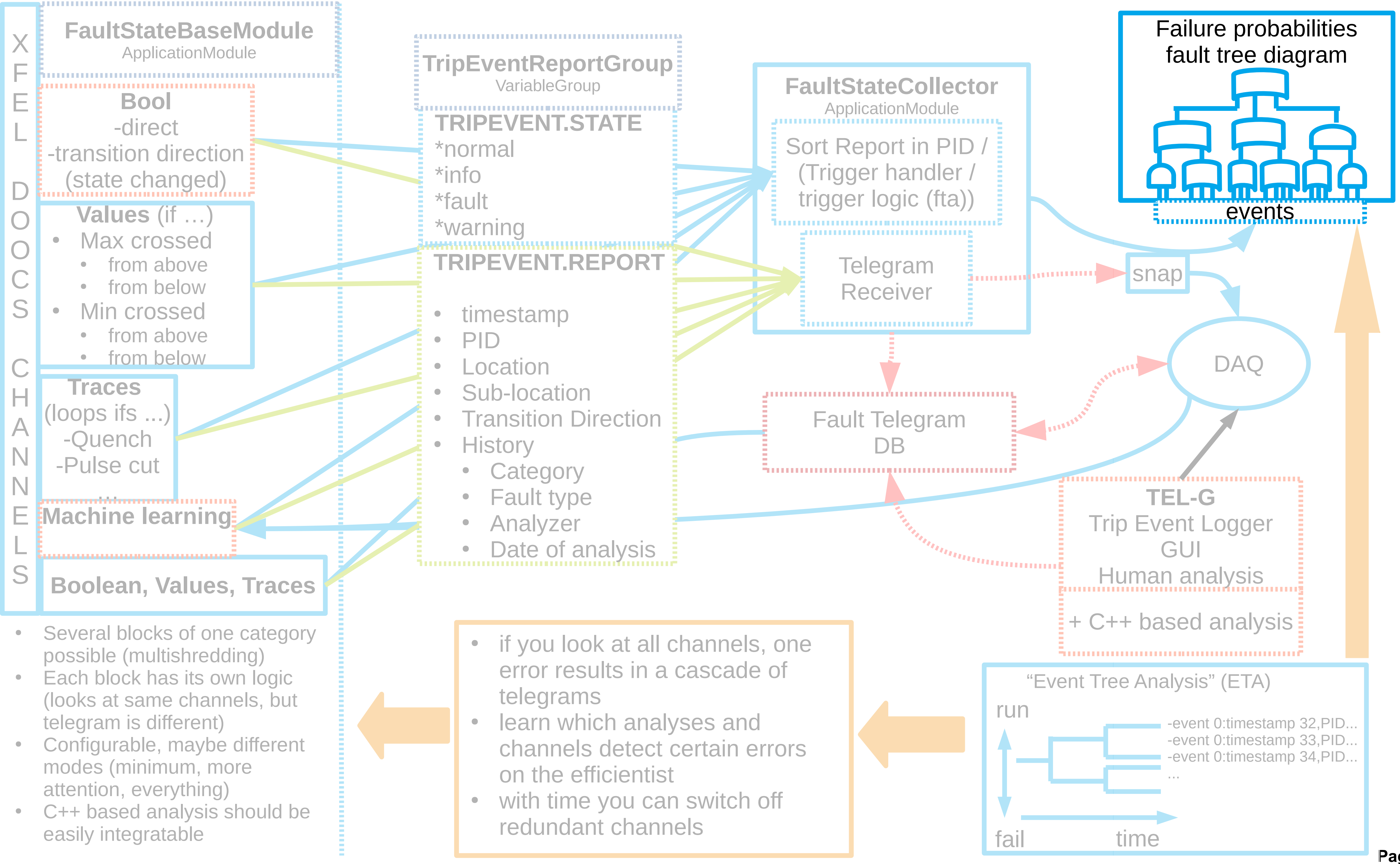
Once you have identified the source of a fault, you may be able to avoid it.

If the understanding of the fault goes far enough and one has appropriate sensors, one can prevent a fault also, before it occurs.

However, not only simple sensors are needed for this, analyses of the data are also necessary in some cases.

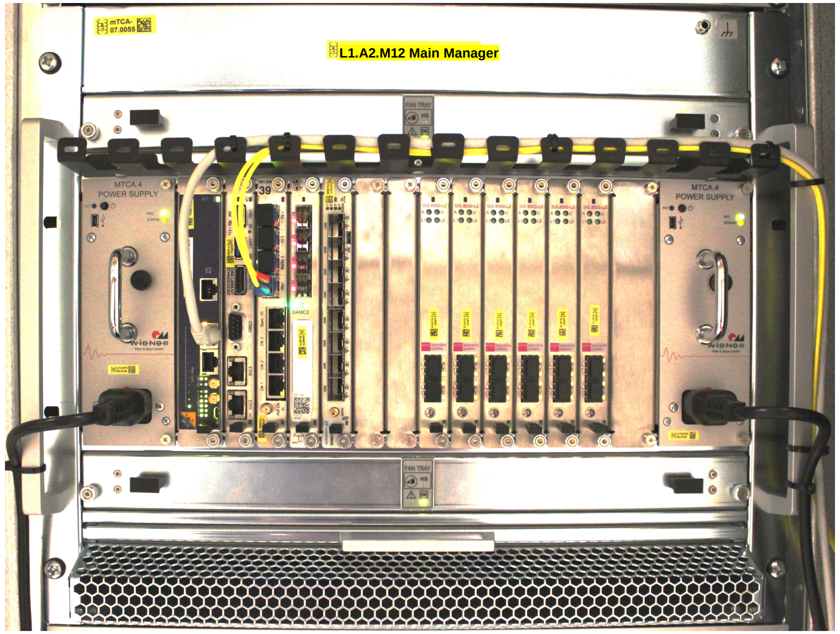
But also a fast, maybe automatic detection, can avoid a long and tedious troubleshooting and you can go directly to fixing the faults.

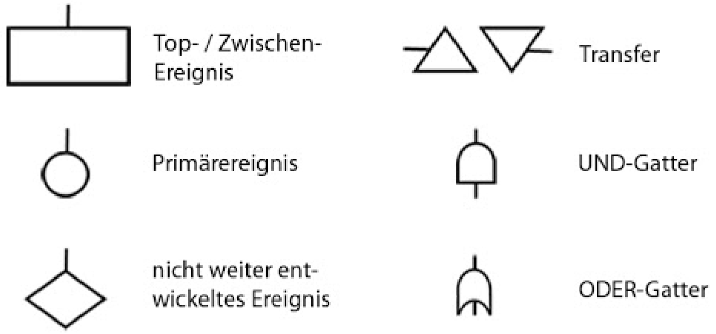
Fault tree analyses are a popular and suitable method for this.

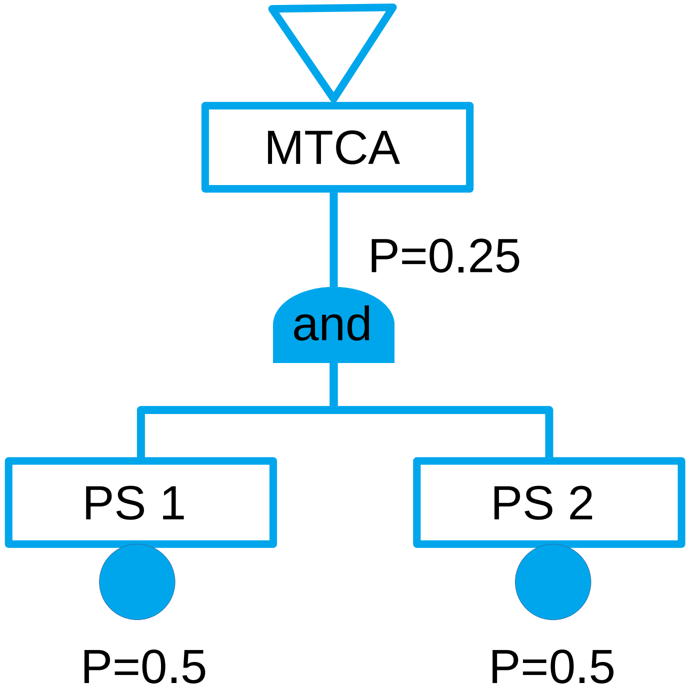

XFEL DOOCS CHANNELS
FaultStateBaseModule
ApplicationModule
Bool
-direct
-transition direction
(state changed)
Values (if …)
• Max crossed
• from above
• from below
• Min crossed
• from above
• from below
Traces
(loops ifs …)
-Quench
-Pulse cut
…
Machine learning
Boolean, Values, Traces
• Several blocks of one category possible (multishredding)
• Each block has its own logic (looks at same channels, but telegram is different)
• Configurable, maybe different modes (minimum, more attention, everything)
• C++ based analysis should be easily integratable
TripEventReportGroup
VariableGroup
TRIPEVENT.STATE
*normal
*info
*fault
*warning
TRIPEVENT.REPORT
• timestamp
• PID
• Location
• Sub-location
• Transition Direction
• History
• Category
• Fault type
• Analyzer
• Date of analysis
FaultStateCollector
ApplicationModule
Sort Report in PID / (Trigger handler / trigger logic (fta))
Telegram Receiver
Fault Telegram DB
Failure probabilities fault tree diagram
events
snap
DAQ
TEL-G
Trip Event Logger
GUI
Human analysis
+ C++ based analysis
• if you look at all channels, one error results in a cascade of telegrams
• learn which analyses and channels detect certain errors on the efficientist
• with time you can switch off redundant channels
"Event Tree Analysis" (ETA)
run
fail
time
-event 0:timestamp 32,PID...
-event 0:timestamp 33,PID...
-event 0:timestamp 34,PID...
...

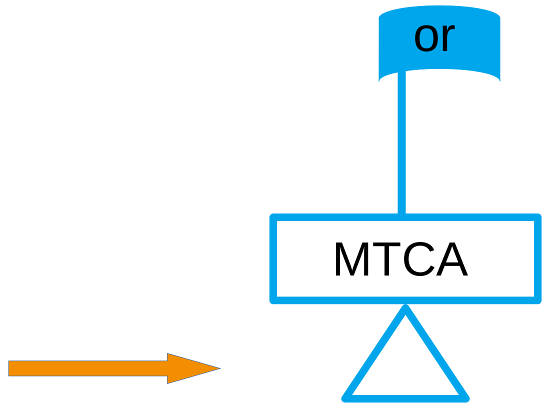
Fault Tree Example

MTCA Crate with two power supply









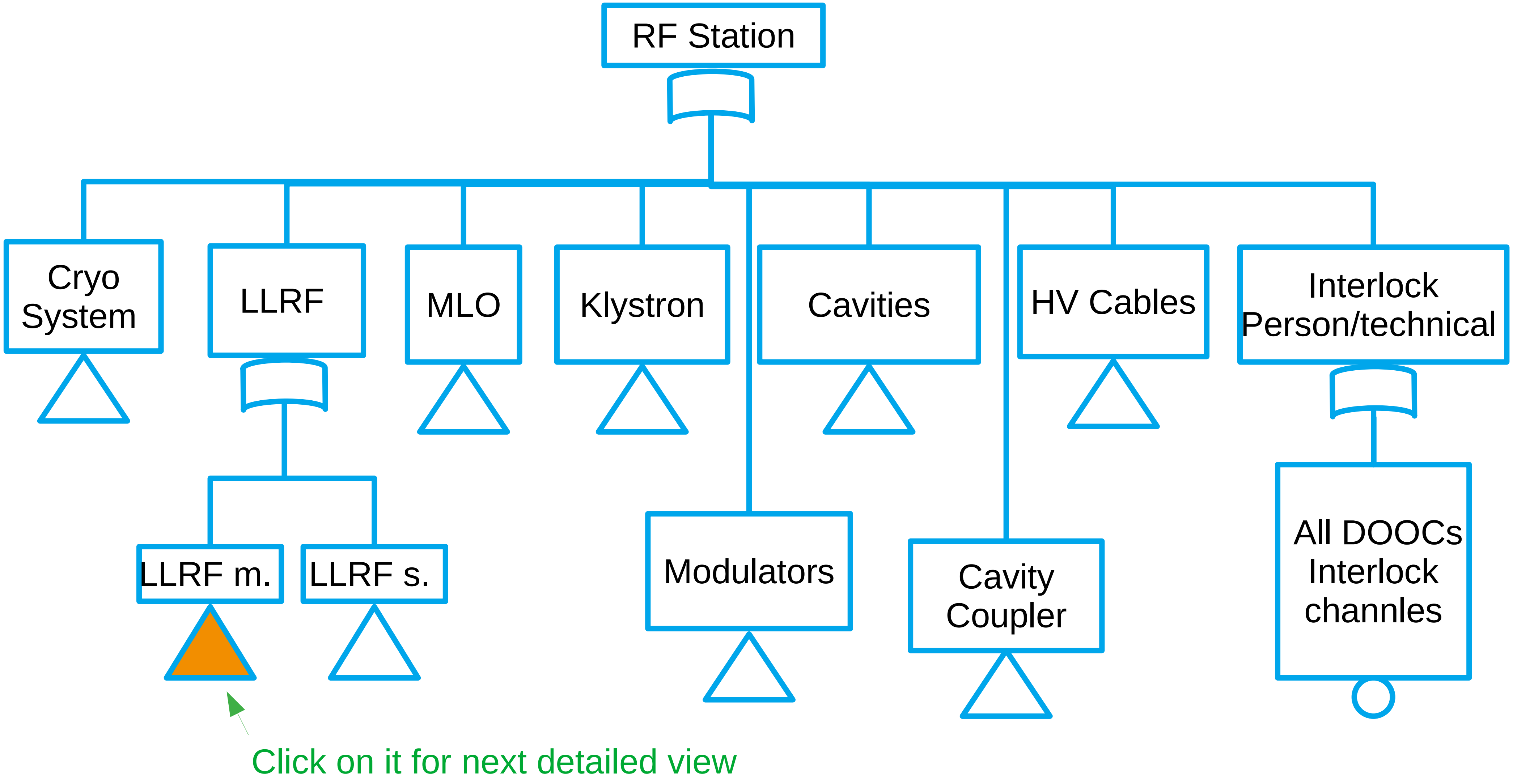
The events can follow different models, e.g. „law of decay“:
P = 1 - e^(-λt)

The fault tree follows a simple logic:
P(A and B) = P(A) * P(B)
P(A or B) = P(A) + P(B) – P(A and B)

If you have **sensors** that tell you what **state** the component/**event** is in and which are connected to a **P**, then you can automatically calculate a certain probability of failure.

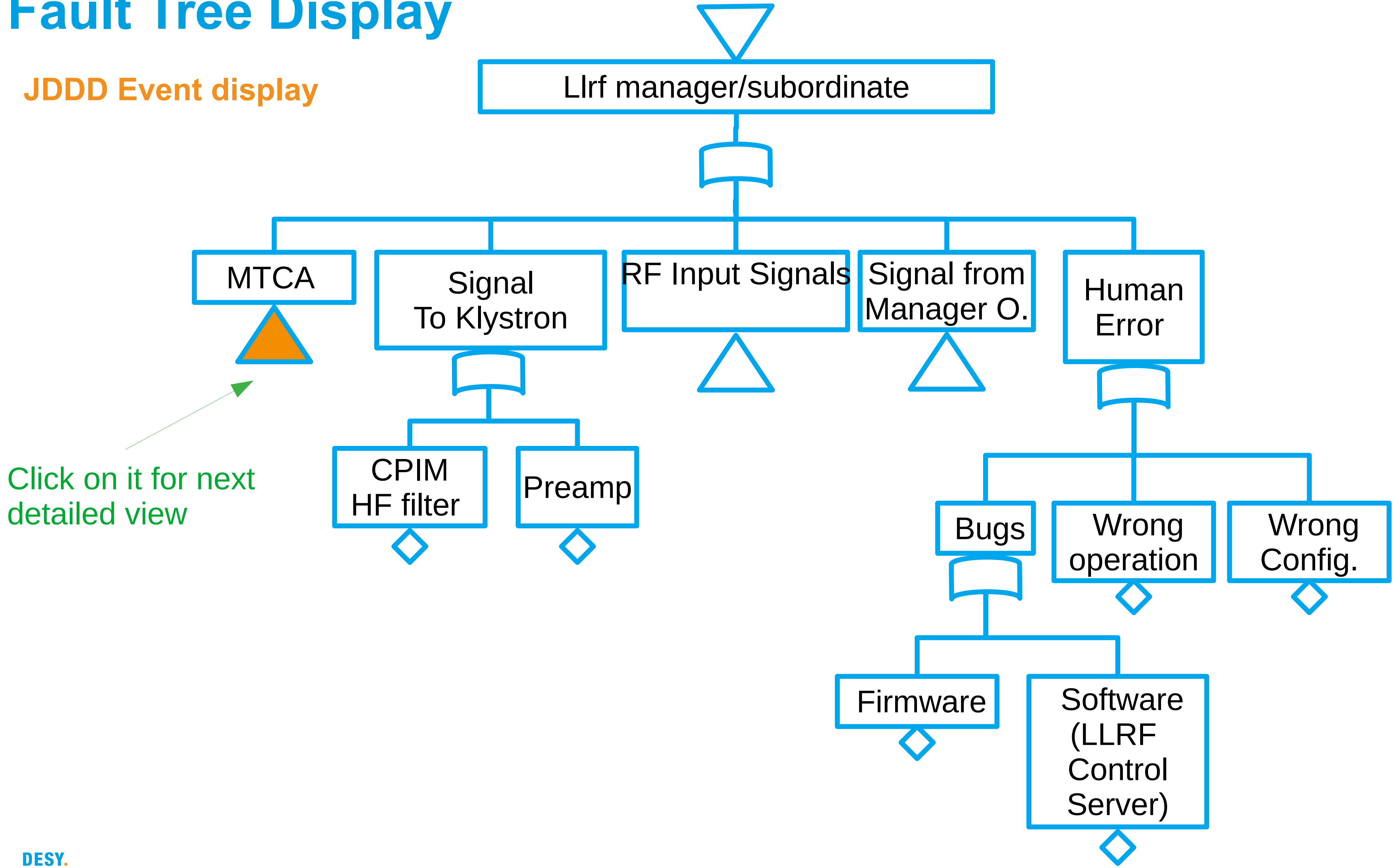
Fault Tree Display

JDDD Event display



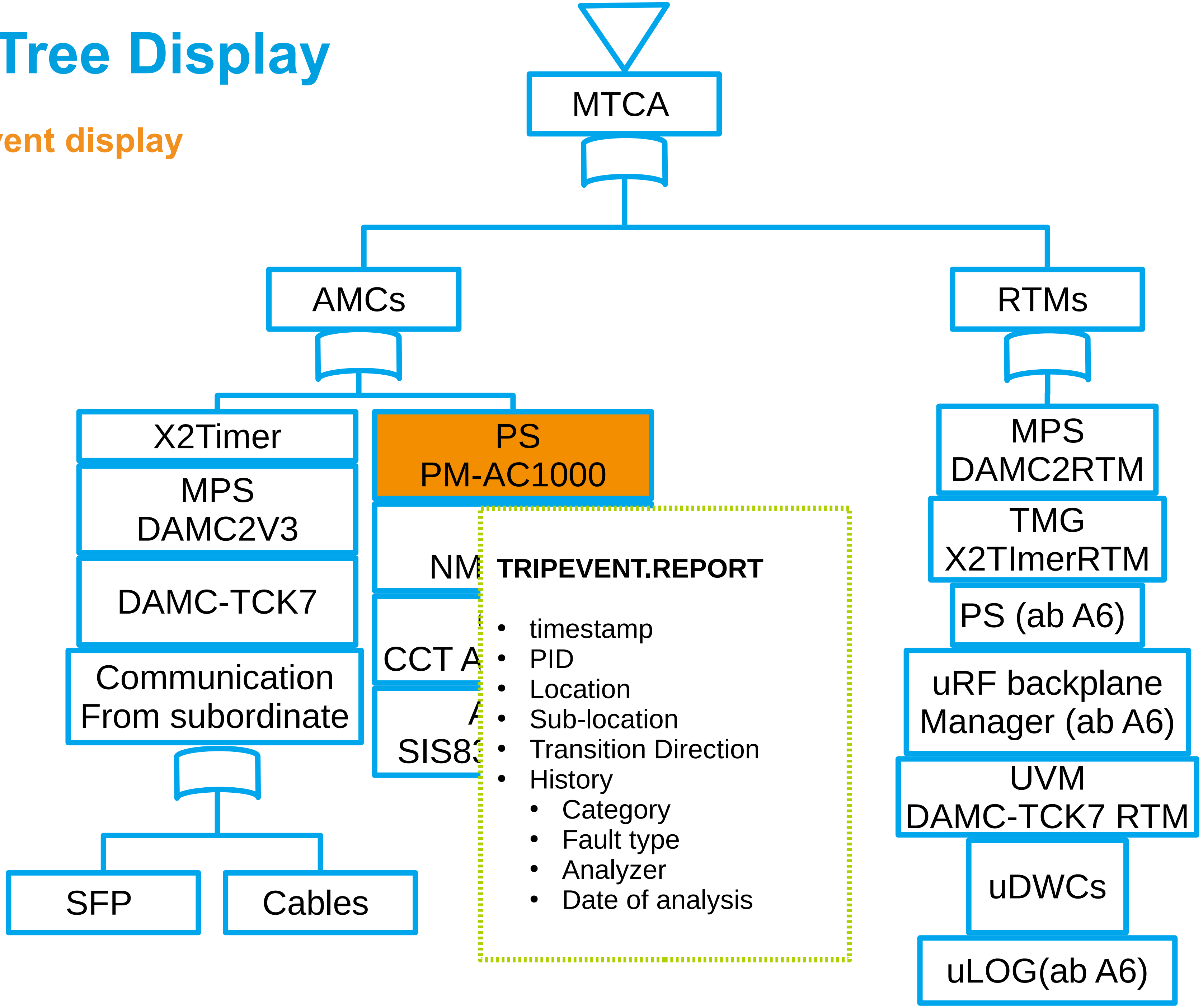
Fault Tree Display

JDDD Event display



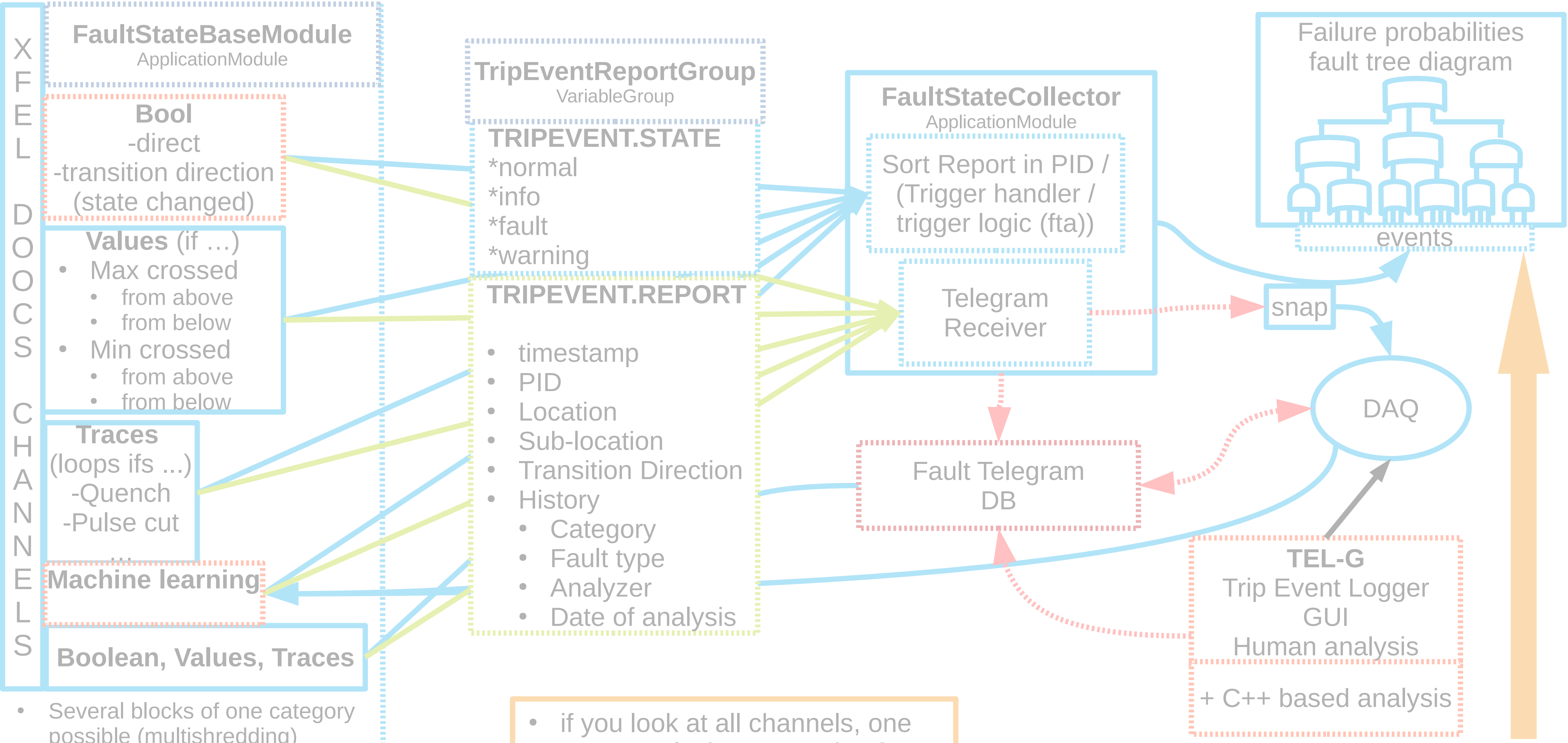
Fault Tree Display

JDDD Event display



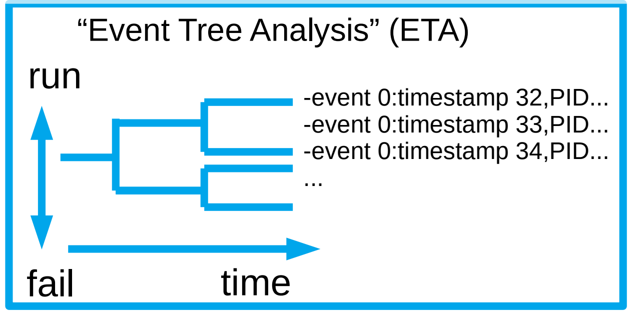
TRIPEVENT.REPORT

- timestamp
- PID
- Location
- Sub-location
- Transition Direction
- History
  - Category
  - Fault type
  - Analyzer
  - Date of analysis



- Several blocks of one category possible (multishredding)
- Each block has its own logic (looks at same channels, but telegram is different)
- Configurable, maybe different modes (minimum, more attention, everything)
- C++ based analysis should be easily integratable

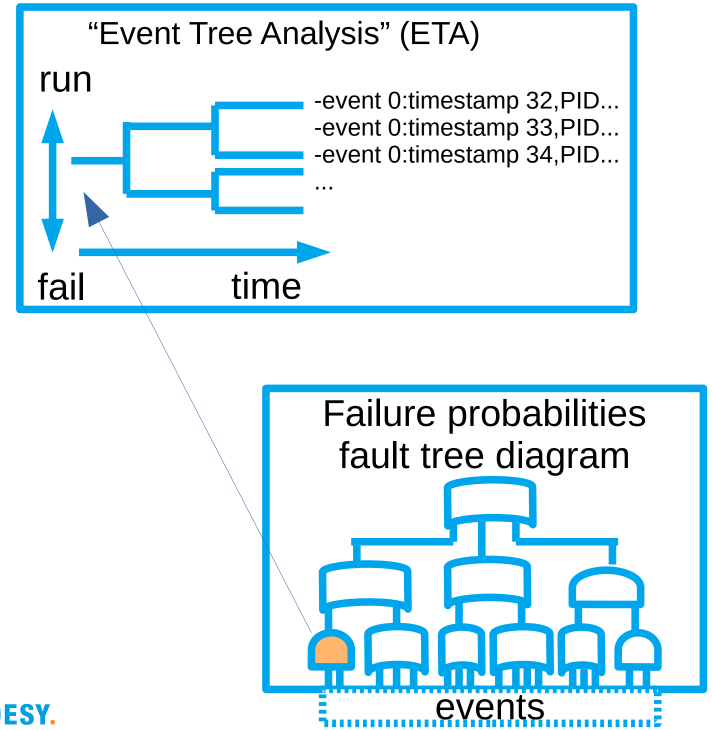
- if you look at all channels, one error results in a cascade of telegrams
- learn which analyses and channels detect certain errors on the efficientist
- with time you can switch off redundant channels

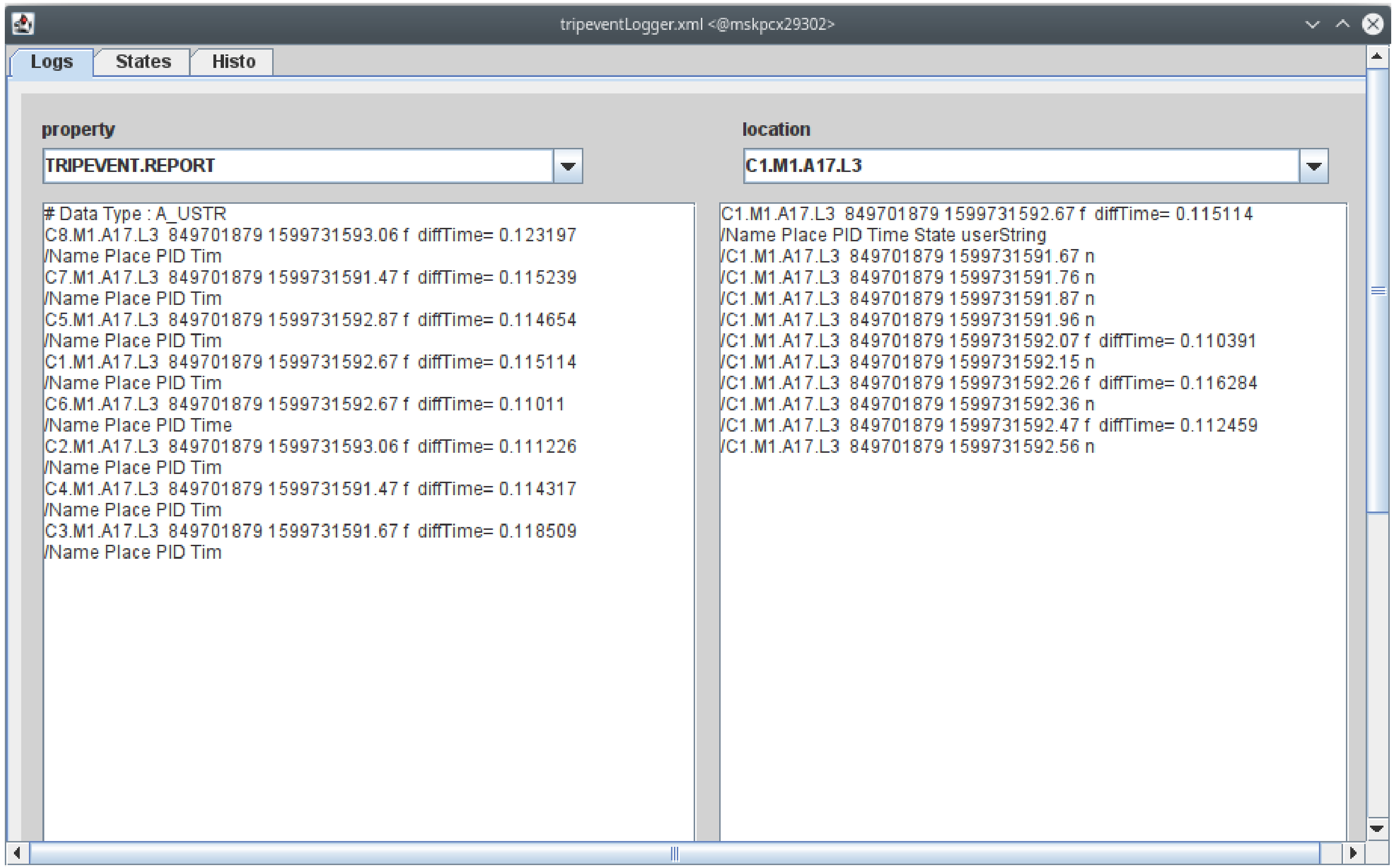


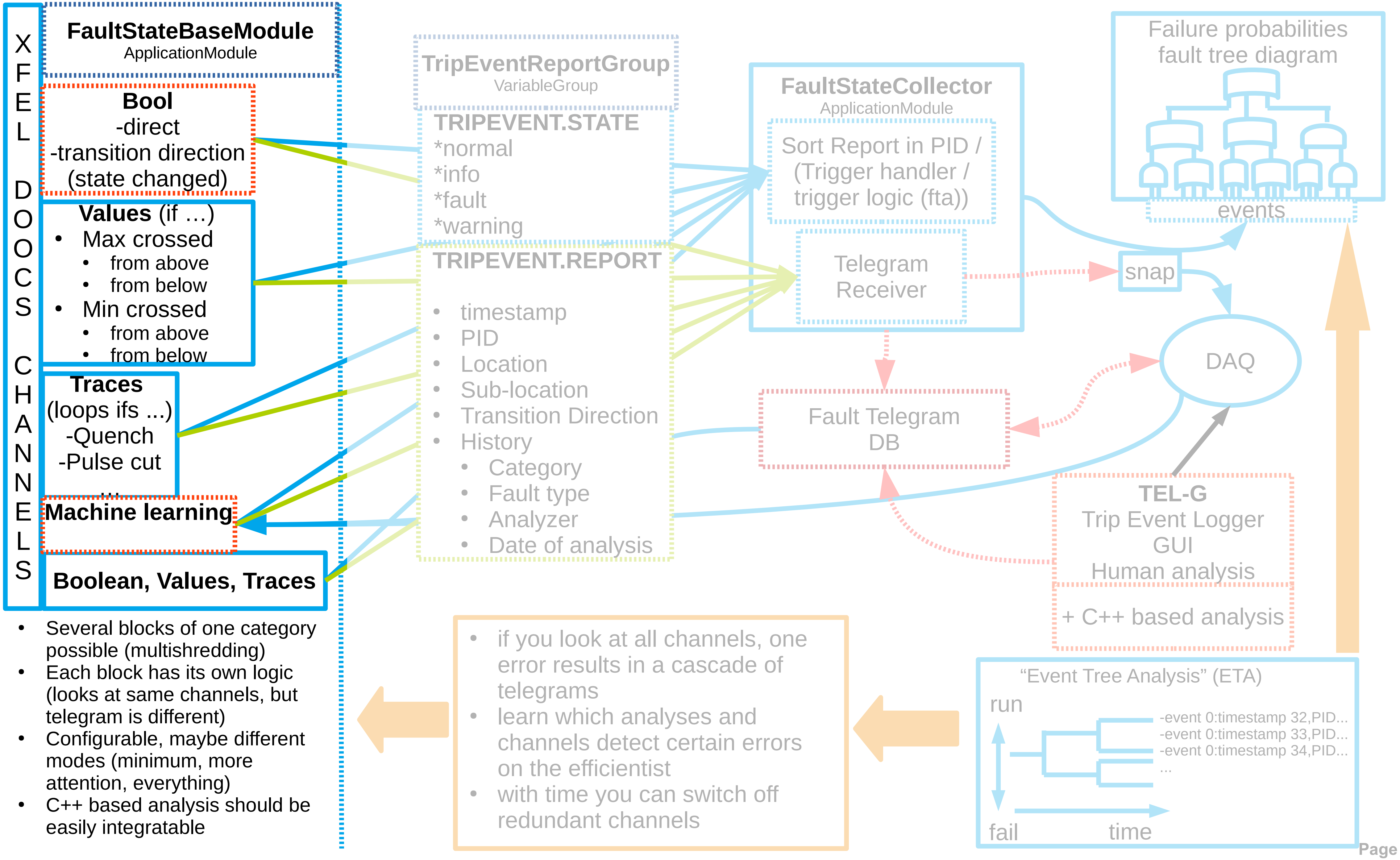
Event Tree (Analysis)

The other way around.

State of all events/components analyzed by a module at the moment of the very first transition to a fault state.






XFEL DOOCS CHANNELS
FaultStateBaseModule
ApplicationModule
Bool
-direct
-transition direction
(state changed)
Values (if …)
• Max crossed
• from above
• from below
• Min crossed
• from above
• from below
Traces
(loops ifs …)
-Quench
-Pulse cut
…
Machine learning
Boolean, Values, Traces
• Several blocks of one category possible (multishredding)
• Each block has its own logic (looks at same channels, but telegram is different)
• Configurable, maybe different modes (minimum, more attention, everything)
• C++ based analysis should be easily integratable
TripEventReportGroup
VariableGroup
TRIPEVENT.STATE
*normal
*info
*fault
*warning
TRIPEVENT.REPORT
• timestamp
• PID
• Location
• Sub-location
• Transition Direction
• History
• Category
• Fault type
• Analyzer
• Date of analysis
FaultStateCollector
ApplicationModule
Sort Report in PID /
(Trigger handler /
trigger logic (fta))
Telegram
Receiver
Fault Telegram
DB
Failure probabilities
fault tree diagram
events
snap
DAQ
TEL-G
Trip Event Logger
GUI
Human analysis
+ C++ based analysis
• if you look at all channels, one error results in a cascade of telegrams
• learn which analyses and channels detect certain errors on the efficientist
• with time you can switch off redundant channels
"Event Tree Analysis" (ETA)
run
fail
time
-event 0:timestamp 32,PID...
-event 0:timestamp 33,PID...
-event 0:timestamp 34,PID...
...

Analysis module

C++ written analysis libraries. Each represents an event / component in the Fault/Event Tree

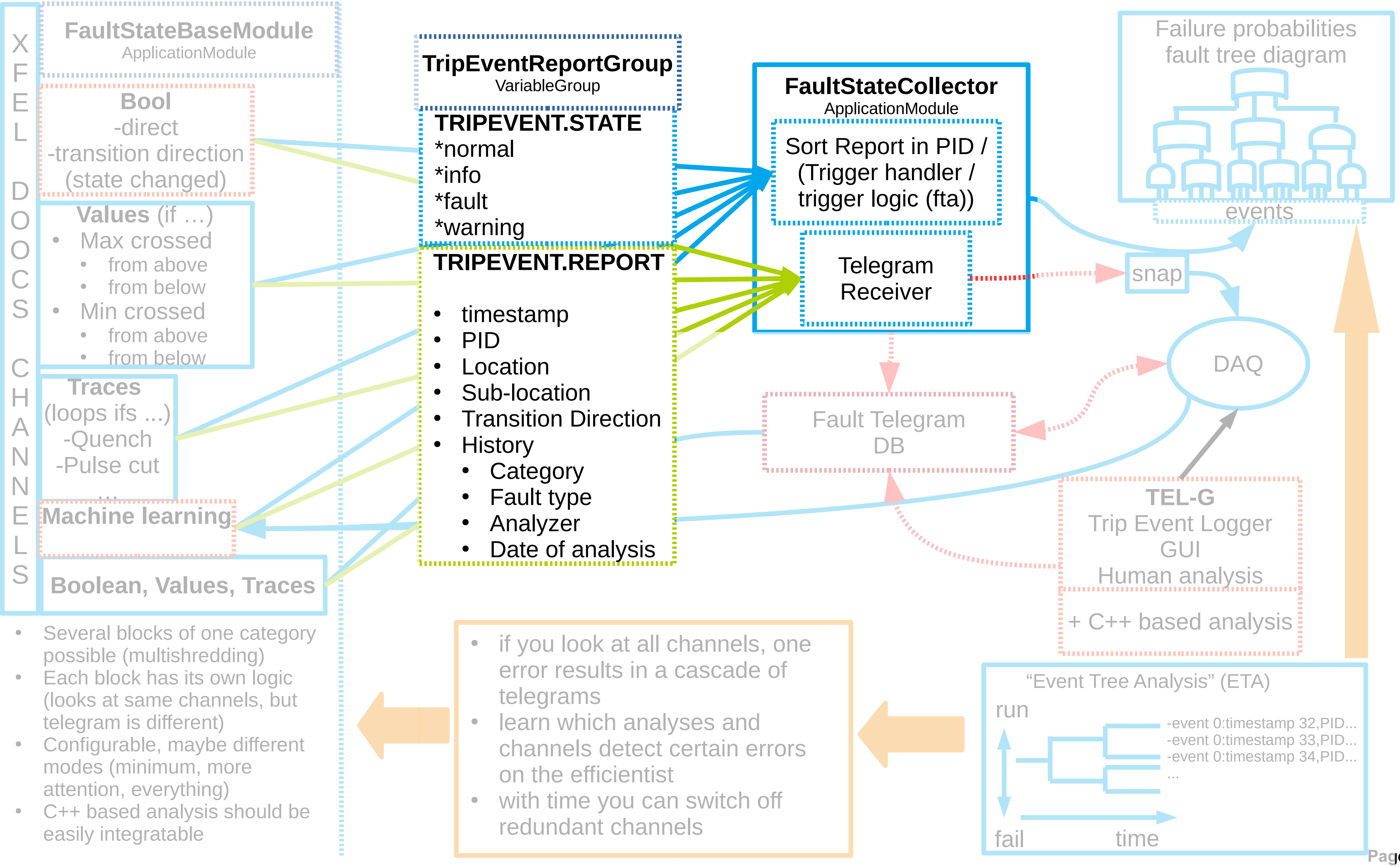
- Input in our case are the 6 cavity signals for cavities (for other components this could also be traces, scalar or bools).
  - But also meta data and parameter about the operation mode of the machine if necessary.
- The output could also be a traces, scalars etc.
  - e.g. written in hdf5 for further investigation, especially in the case of offline analysis,
  - but you must also evaluate the output for the TEL and break it down to at least 3 states: off, normal, (warning1 .. warningN), failure.
- Each module must know its location, time and macropulse number.

Bool
-direct
-transition direction
(state changed)

Values (if …)
- Max crossed
  - from above
  - from below
- Min crossed
  - from above
  - from below

Traces
(loops ifs ...)
-Quench
-Pulse cut
…

Machine learning


XFEL DOOCS CHANNELS
FaultStateBaseModule
ApplicationModule
Bool
-direct
-transition direction
(state changed)
Values (if …)
• Max crossed
• from above
• from below
• Min crossed
• from above
• from below
Traces
(loops ifs …)
-Quench
-Pulse cut
…
Machine learning
Boolean, Values, Traces
• Several blocks of one category possible (multishredding)
• Each block has its own logic (looks at same channels, but telegram is different)
• Configurable, maybe different modes (minimum, more attention, everything)
• C++ based analysis should be easily integratable
TripEventReportGroup
VariableGroup
TRIPEVENT.STATE
*normal
*info
*fault
*warning
TRIPEVENT.REPORT
• timestamp
• PID
• Location
• Sub-location
• Transition Direction
• History
• Category
• Fault type
• Analyzer
• Date of analysis
FaultStateCollector
ApplicationModule
Sort Report in PID / (Trigger handler / trigger logic (fta))
Telegram Receiver
Fault Telegram DB
Failure probabilities fault tree diagram
events
snap
DAQ
TEL-G
Trip Event Logger GUI
Human analysis
+ C++ based analysis
• if you look at all channels, one error results in a cascade of telegrams
• learn which analyses and channels detect certain errors on the efficientist
• with time you can switch off redundant channels
"Event Tree Analysis" (ETA)
run
fail
time
-event 0:timestamp 32,PID...
-event 0:timestamp 33,PID...
-event 0:timestamp 34,PID...
...

DESY.

Application Core Analysis module

C++ written analysis libraries.

- Input are the results of analysis module and the states
  - But also meta data and parameter about the operation mode of the machine if necessary.
- The output:
  - State and Report
- Each module must know its location, time and macropulse number.
- Intervention

Fault State Collector

- Define a duration fault time, starts with the first fault.
- Sort Events by time.
- **Intervention**

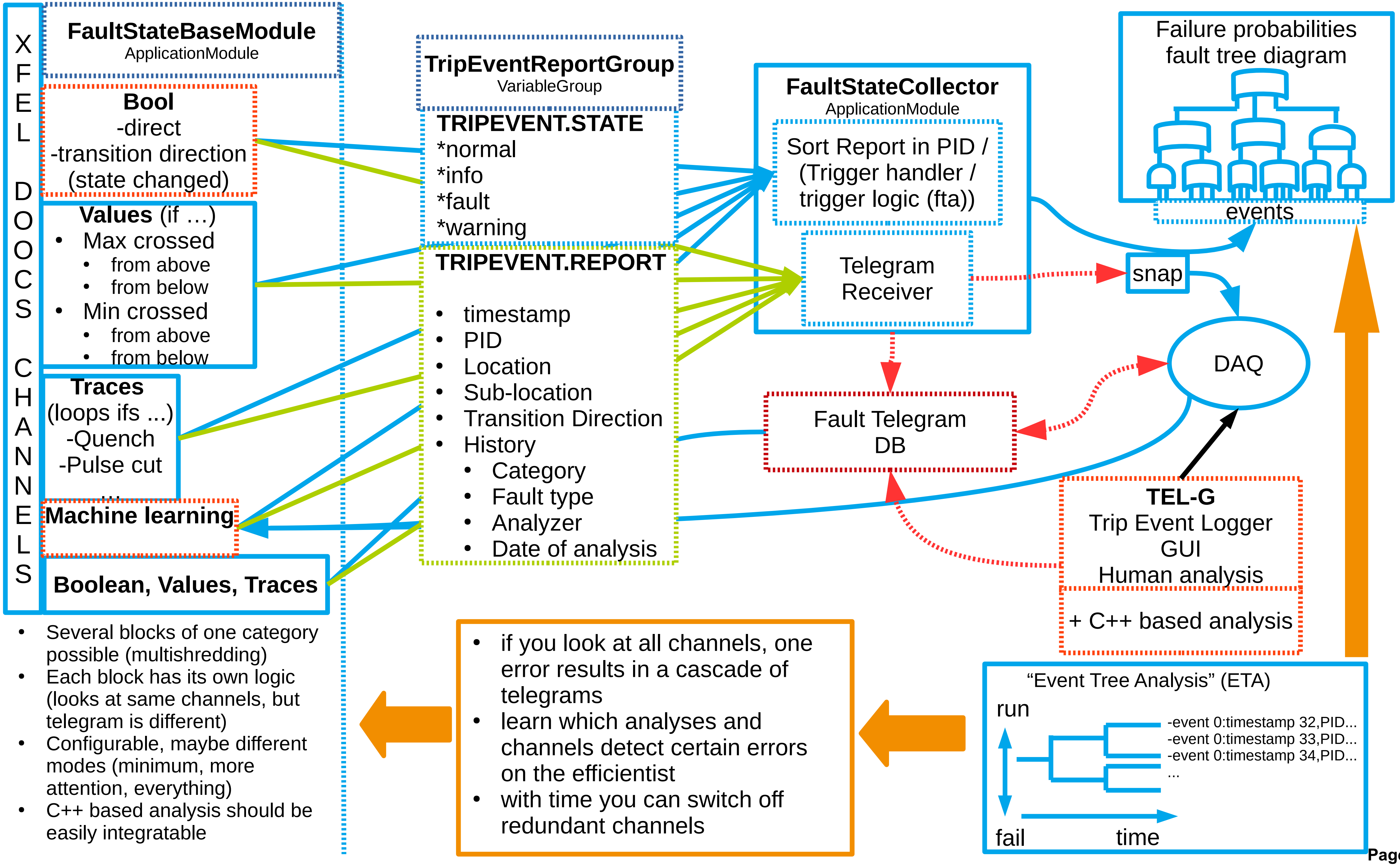
FaultStateBaseModule
ApplicationModule

TripEventReportGroup
Variable(Consistensy)Group

TRIPEVENT.STATE
*normal
*info
*fault
*warning

TRIPEVENT.REPORT

- timestamp
- macro pulse number
- Location
- Sub-location
- Transition Direction
- History
  - Category
  - Fault type
  - Analyzer
  - Date of analysis


XFEL DOOCS CHANNELS
FaultStateBaseModule
ApplicationModule
Bool
-direct
-transition direction
(state changed)
Values (if …)
• Max crossed
• from above
• from below
• Min crossed
• from above
• from below
Traces
(loops ifs …)
-Quench
-Pulse cut
…
Machine learning
Boolean, Values, Traces
TripEventReportGroup
VariableGroup
TRIPEVENT.STATE
*normal
*info
*fault
*warning
TRIPEVENT.REPORT
• timestamp
• PID
• Location
• Sub-location
• Transition Direction
• History
• Category
• Fault type
• Analyzer
• Date of analysis
FaultStateCollector
ApplicationModule
Sort Report in PID / (Trigger handler / trigger logic (fta))
Telegram Receiver
Failure probabilities fault tree diagram
events
snap
DAQ
Fault Telegram DB
TEL-G
Trip Event Logger
GUI
Human analysis
+ C++ based analysis
• Several blocks of one category possible (multishredding)
• Each block has its own logic (looks at same channels, but telegram is different)
• Configurable, maybe different modes (minimum, more attention, everything)
• C++ based analysis should be easily integratable
• if you look at all channels, one error results in a cascade of telegrams
• learn which analyses and channels detect certain errors on the efficientist
• with time you can switch off redundant channels
“Event Tree Analysis” (ETA)
run
fail
time
-event 0:timestamp 32,PID...
-event 0:timestamp 33,PID...
-event 0:timestamp 34,PID...
...

Implementation

The previous slide shows something like the workflow we want to follow.

The next slides are about the actual implementation and how the trip event logger is connected to the different components like DAQ, control system, HPC cluster and analysis methods.

Within the Application Core based Trip Event Logger is really the logic that collects all the information and triggers the sending of a fault telegram. Since we are sitting directly in the control system, decisions could also be made here to anticipate possible faults.

DOOCS
Control System
OPC-UA
EPICS
(TANGO)

Trip Event Logger

ChimeraTK
Application Core
Analysis Module

TRIPEVENT
Reports
States

Fault State Collector
-Event tree
(global state of all modules
at the moment of first transition
to a fault)
Not yet implemented

DOCCS DAQ Snap
Sample reduced
MCS DAQ Data

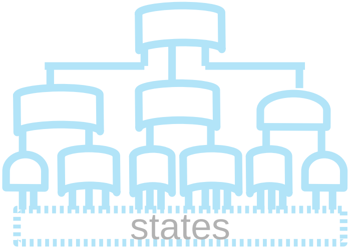
Not yet automated

Analysis Modules

- C++
- *MATLAB*
(-PYTHON)
(-NN)
-…

setData(&raw)
setData(&ttf2_daq_getdata)
GetParameter()
Calculate()
WriteResults("results.h5")
Evaluate() → state

Fault Tree
Event display
Failure probability calculation
Not yet implemented



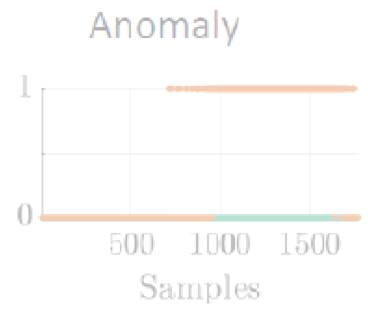
Data Sets
For long term use

-Labled data
-Trainings data
-Simulated Data
.raw (MCS DAQ)
.hdf5

Ladybug
Offline Module
Analysis adapter
for HPC
Command line tool

Other scripts
Docker container
File and
data set manger

Detailed Results



Trip Event Logger

It should detect failures and than report them with a failure telegram.

Telegrams are collected, as for fault tree analysis and for collecting data of interest.

- Telegram: failure state, location, time, macropulse number, (link to data, failure probability).
- ChimeraTK Application Core [1] [2]
  - This not only enables monitoring, but also allows you to intervene in the control system.
  - There are several control system backends. (DOOCS, OPC-UA, EPICS, *(TANGO)*)
  - Thread management is under the hood.
  - Full sample rate. Scalable, if enough computing power is available.
- An analysis of the parity space will be part of this soon (and the first module), developed by Ayla Nawaz [3].

DOOCS
Control System
OPC-UA
EPICS
(TANGO)

Trip Event Logger

ChimeraTK Application Core Analysis Module

TRIPEVENT
Reports
States

DOCCS DAQ Snap
Sample reduced
MCS DAQ Data

Not yet automated

Data Sets
For long term use

-Labled data
-Trainings data
-Simulated Data
.raw (MCS DAQ)
.hdf5

Ladybug
Offline Module
Analysis adapter
for HPC
Command line tool

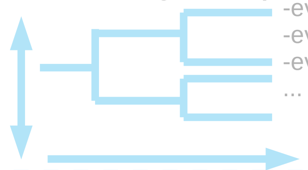
Other scripts
Docker container
File and
data set manger

Analysis Modules

- C++
- *MATLAB*
(-PYTHON)
(-NN)
-…

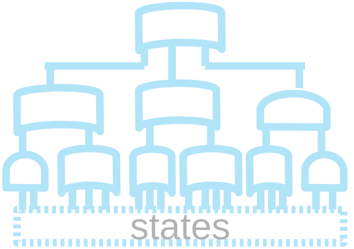
setData(&ctk)
setData(&ttf2_daq_getdata)
GetParameter()
Calculate()
WriteResults(“results.h5”)
Evaluate() → state

Fault State Collector
-Event tree
(global state of all modules
at the moment of first transition
to a fault)
Not yet implemented

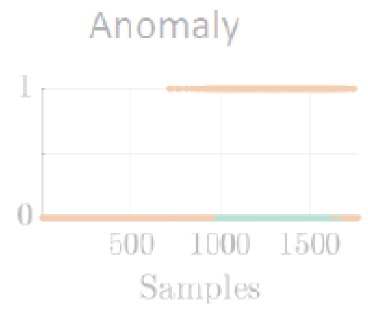


-event 0:timestamp 32,PID...
-event 0:timestamp 33,PID...
-event 0:timestamp 34,PID...
...

Fault Tree
Event display
Failure probability calculation
Not yet implemented



Detailed Results



Method development for anomaly detection

From MATLAB to C++

- Current workflow:
  - Development of analysis in MATLAB on sample reduced DAQ data.
  - Automatically generated C MATLAB code with semi-autmomatic integration in our tools for the MAXWELL cluster.

- Development of analysis modules directly in C++
  - This guarantees us a good cooperation with the rest of the Helmholtz Gemeinschaft.
  - There are ideas to use CINT the C++ Interpreter from ROOT @ CERN.

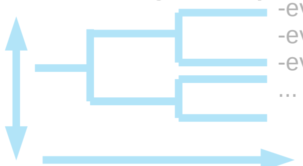
DOOCS
Control System
OPC-UA
EPICS
(TANGO)

Trip Event Logger

ChimeraTK
Application Core
Analysis Module

TRIPEVENT
Reports
States

Fault State Collector
-Event tree
(global state of all modules
at the moment of first transition
to a fault)
Not yet implemented



-event 0:timestamp 32,PID...
-event 0:timestamp 33,PID...
-event 0:timestamp 34,PID...
...

DOCCS DAQ Snap
Sample reduced
MCS DAQ Data

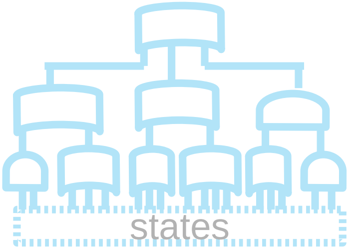
Not yet automated

Analysis Modules

- C++
- *MATLAB*
(-PYTHON)
(-NN)
-...

setData(&raw)
setData(&ttf2_daq_getdata)
GetParameter()
Calculate()
WriteResults("results.h5")
Evaluate() → state

Fault Tree
Event display
Failure probability calculation
Not yet implemented



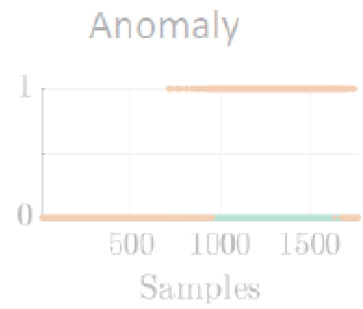
Ladybug
Offline Module
Analysis adapter
for HPC
Command line tool

Other scripts
Docker container
File and
data set manger

Data Sets
For long term use

-Labled data
-Trainings data
-Simulated Data
.raw (MCS DAQ)
.hdf5

Detailed Results



Tools for Maxwell

Ladybug

With these modules you can also analyses DAQ data.

Very primitive parallelization

- The structure of the problem and the high amount of data allow to choose the simplest parallelization.
- You start the job as often as CPUs are in the node.
- The number of macropulses investigated determines the job duration.

- Docker container with all dependencies, ubuntu16, DOOCS, ChimeraTK, ARMADILLO...
- Ladybug is connecting the modules with ttf2looper and the DAQ data.
- With bash scripts and a dataset manager you can submit jobs based on slurm e.g. to the MAXWELL cluster.
- Merge tool for handy result data files.

DOOCS
Control System
OPC-UA
EPICS
(TANGO)

Trip Event Logger

ChimeraTK
Application Core
Analysis Module

TRIPEVENT
Reports
States

Fault State Collector
-Event tree
(global state of all modules
at the moment of first transition
to a fault)
Not yet implemented

DOCCS DAQ Snap
Sample reduced
MCS DAQ Data

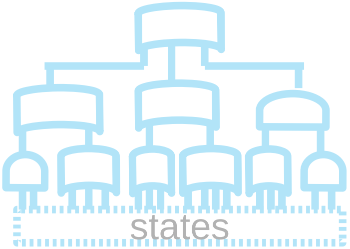
Not yet automated

Analysis Modules

- C++
- *MATLAB*
(-PYTHON)
(-NN)
-…

setData(&raw)
setData(&ttf2_daq_getdata)
GetParameter()
Calculate()
WriteResults(“results.h5”)
Evaluate() → state

Fault Tree
Event display
Failure probability calculation
Not yet implemented



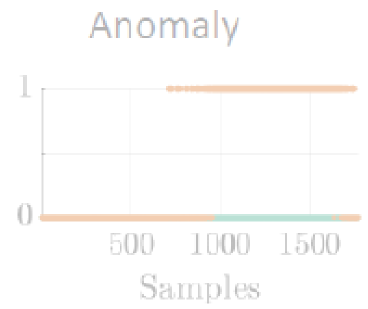
Data Sets
For long term use

-Labled data
-Trainings data
-Simulated Data
.raw (MCS DAQ)
.hdf5

Ladybug
Offline Module
Analysis adapter
for HPC
Command line tool

Other scripts
Docker container
File and
data set manger

Detailed Results



DAQ

The current DAQ system is provided by MCS.

- Only sample reduced rate (1,8k samples, float)
- MCS-DOOCS data format
  - Very fast and smart!
  - Unwiedly in C/C++ → A small library can help here (ttf2looper.h)
  - Not common, hdf5 would be better.
- Not all channels of interest are available in the DAQ, but maybe in DOOCS history.
- Ring buffer with ~ 1 week samples reduced data on dCache, now available on Maxwell.
- Local histories.

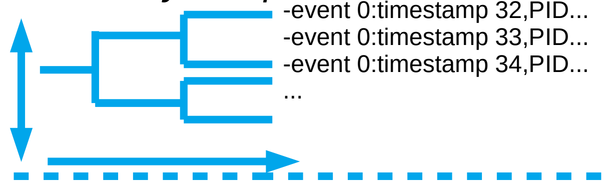
DOOCS
Control System
OPC-UA
EPICS
(TANGO)

Trip Event Logger

ChimeraTK
Application Core
Analysis Module

TRIPEVENT
Reports
States

Fault State Collector
-Event tree
(global state of all modules
at the moment of first transition
to a fault)
Not yet implemented



DOCCS DAQ Snap
Sample reduced
MCS DAQ Data

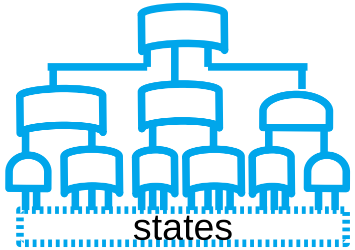
Not yet automated

Analysis Modules

- C++
- *MATLAB*
(-PYTHON)
(-NN)
-…

setData(&raw)
setData(&ttf2_daq_getdata)
GetParameter()
Calculate()
WriteResults(“results.h5”)
Evaluate() → state

Fault Tree
Event display
Failure probability calculation
Not yet implemented



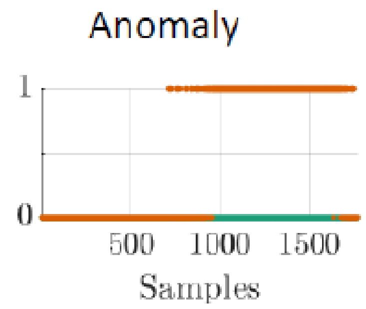
Data Sets
For long term use

-Labled data
-Trainings data
-Simulated Data
.raw (MCS DAQ)
.hdf5

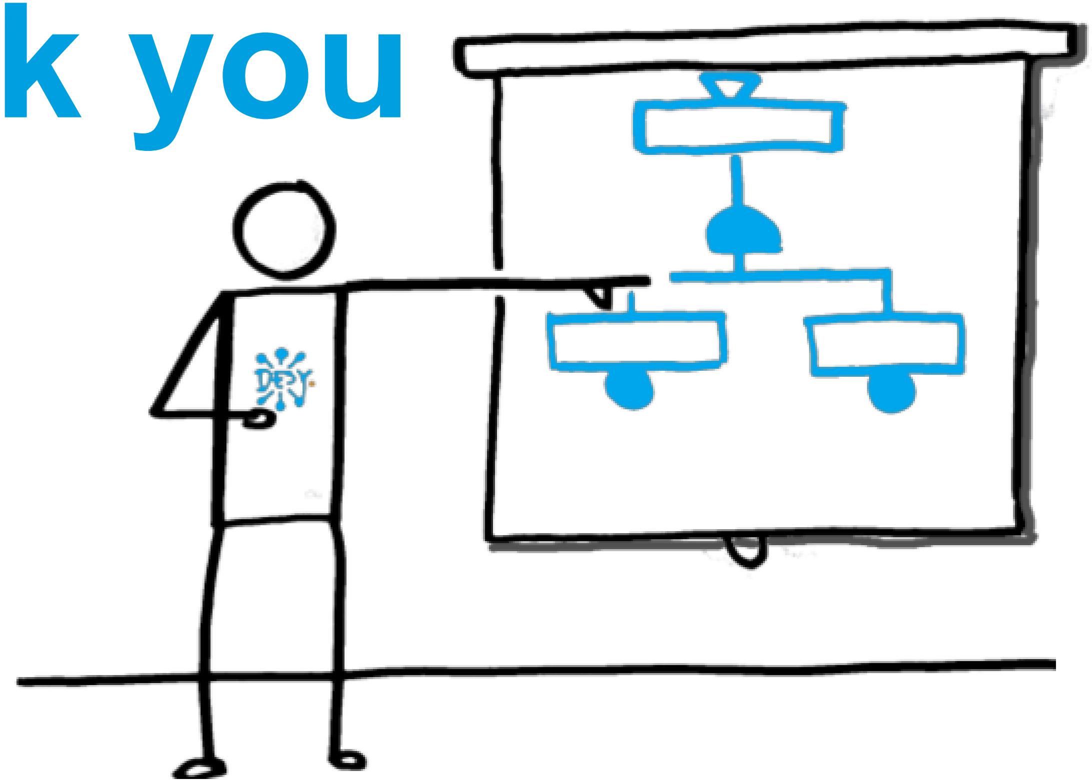
Ladybug
Offline Module
Analysis adapter
for HPC
Command line tool

Other scripts
Docker container
File and
data set manger

Detailed Results



Thank you



If you are interested in working on the project, there will be a job posting soon. Feel free to contact us:

Annika Eichler
annika.eichler@desy.de

Julien Branlard
julien.branlard@desy.de

Holger Schlarb
holger.schlarb@desy.de

Contact

DESY. Deutsches Elektronen-Synchrotron

www.desy.de

Jan Timm
DESY - MCS4
Jan.horst.karl.timm@desy.de

References

[1] Varghese, G. et al. “ChimeraTK - A Software Tool Kit for Control Applications.” (2017).

[2] ChimeraTK Repository, https://github.com/ChimeraTK

[3] Nawaz, A. et al. “Anomaly Detection for the European XFEL using a Nonlinear Parity Space Method.” IFAC-PapersOnLine 51 (2018): 1379-1386.