# Trigger functions

# Zabbix supports almost 100 different functions:

| | |
|---|---|
| Aggregate functions | avg, count, min, max, sum, kurtosis etc. |
| Bitwise functions | bitand, bitlshift, bitnot, bitor, bitrshift, bitxor |
| Date and time functions | date, dayofmonth, dayofweek, now, time |
| History functions | last, first, change, logseverity, monoinc, nodata, etc. |
| Trend-based functions | trendavg, trendcount, trendmax, trendmin, trendsum |
| Mathematical functions | abs, cos, sin, tan, ceil, floor, degrees, e, exp, log, rand, etc. |
| Operator functions | in, between |
| Prediction functions | forecast, timeleft |
| String functions | ascii, bitlength, char, concat, find, left, length, trim, mid etc. |
| Foreach functions | avg_foreach, last_foreach, sum_foreach (calculated items only) |

Advanced functions and absolute time shift periods are discussed in ZCP 6.0

Most of the functions require one or multiple parameters:

- /host/key is a common mandatory first parameter for history functions

```
last(/prod/system.cpu.load)
```

- Other parameters are placed after the /host/key separated by a comma
- If an evaluation period or range is required, it always goes as a second parameter

```
min(/prod/vm.memory.size[free],1h)
```

- More than one parameter may be required for some functions

```
count(/prod/log[/var/log/myApp.log],10m,"like","error" )
```

/host/key and evaluation period or range parameters must never be quoted

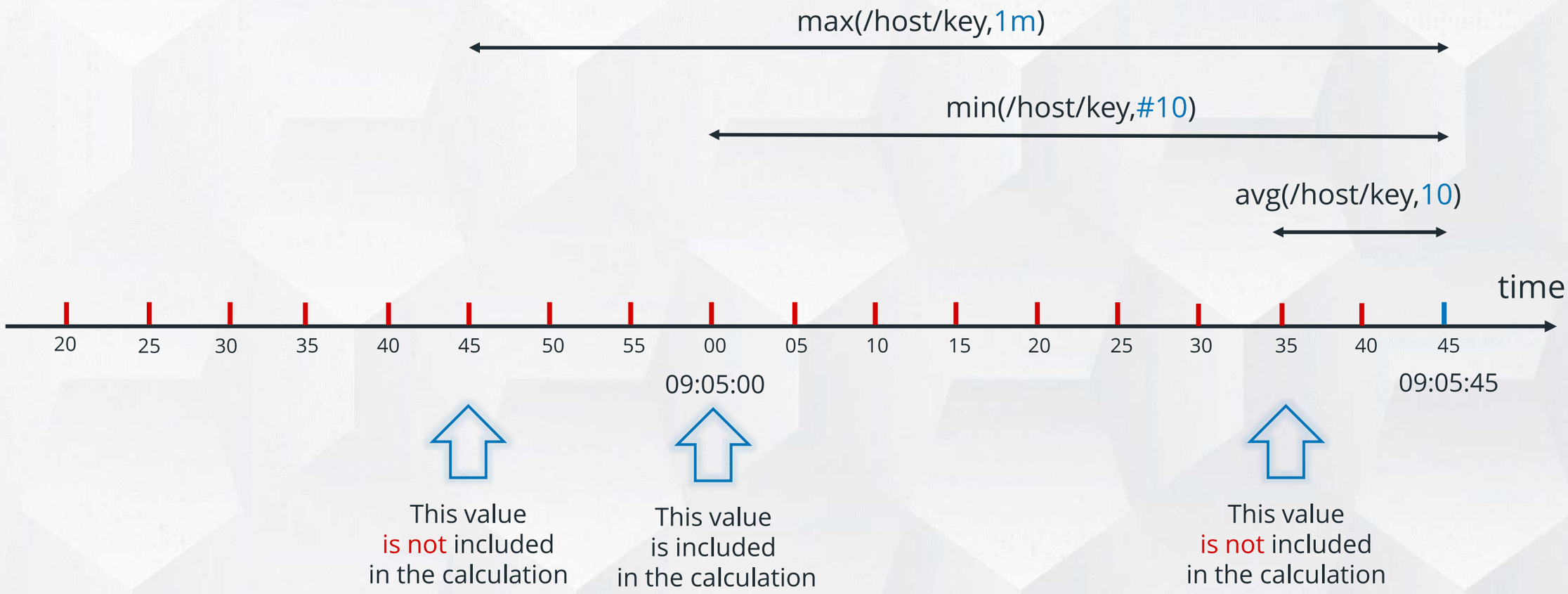
Most of numeric functions accept time or number of values as a parameter:

- Seconds will be used if no time suffix is specified
- Time suffixes may be used to specify time units (10s, 5m, 1h, etc.)
- If preceded by a hashtag #, the parameter indicates the number of values (#5, #10, etc.)

| | | |
|---|---|---|
| sum(/host/key,10m) |  | Sum of values in the last 10 minutes |
| sum(/host/key,10s) |  | Sum of values in the last 10 seconds |
| sum(/host/key,10) |  | Sum of values in the last 10 seconds |
| sum(/host/key,#10) |  | Sum of the last 10 values |

## Example:

- Item update interval: 5s
- Item collected every minute at 00, 05, 10, 15 ,etc.



# Mathematical operations can be applied to the trigger functions:

- To the result of trigger function

avg(/host/net.if.in[eth0,bytes],10m) * 8 > 10M  result is multiplied by 8

- Between results of trigger functions

min(/host/system.cpu.load,5m) / last(/host/system.cpu.num) > 1.5  Load per CPU

- To the results of multiple trigger functions

last(/host/proc.num) / last(/host/kernel.maxproc) * 100 > 80  80 percent calculated

Mathematical operations must be added after the function

| Correct | Correct | Wrong |
|---|---|---|
| min(/host/key,#5) | min(/host/key,#5)*10 | min(/host/key,#5*10) |

Aggregate functions will accept other expressions as function parameters:

function(function_a(),function_b(),...)

- Function beginning with host and item key will use time period as parameter

min(/host/key1,1h) — smallest value from 1 hour of historical data

- Function beginning with other expressions will use them as parameters

min(avg(/host/key1,1h),min(/host/key2,#5)*10) — smallest value from the result of other expressions

function1

function2

## The following operators are supported for triggers:

- Unary minus - (change the sign of an operand)
- Mathematical operations ( +, -, *, / )
- Compare (<, <=, >, >=, =, <>)
- Logical operators (and, or, not)
  - ✓ Case-sensitive and must be in lowercase
  - ✓ Must be surrounded by spaces or parentheses.

## Notes:

- Most operators expect numerical variables
- Operators = or <> can be used to compare strings

last(/host/vfs.file.cksum[/etc/passwd],#1) <> last(/host/vfs.file.cksum[/etc/passwd],#2)

last(/host1/system.hw.macaddr[eth0,short],#1) = last(/host2/system.hw.macaddr[eth0,short],#1)

# FUNCTION COMPARE

When building trigger expressions, it is possible to compare function against:

- Fixed value

```
last(/prod/agent.version) <> '6.0.0'
```

- User macro

```
min(/prod/system.cpu.load,5m) > {$CPU.LOAD}
```

- Result of another trigger function

```
last(/node1/hw.macaddr) = last(/node2/hw.macaddr)
```

- Result of calculation

```
last(/node1/system.cpu.num) > last(/node2/system.cpu.num) * 1.5
```

# An optional time shift is supported in the time parameter:

- This parameter allows to reference the data from a period of time in the past
- Time shift starts with now (the current time), followed by:
  - ✓ +N<time unit> to add N time units
  - ✓ -N<time unit> to subtract N time units

```
avg(/host/key,4h:now-1d)
```

- Complex expressions using multiple time units in the calculation are supported

```
avg(/host/key,6h:now-1d+8h)
```


https://www.zabbix.com/documentation/6.0/en/manual/config/triggers/expression

# The last() function returns the last received value:

- The time period is not supported

  last(/host/key)

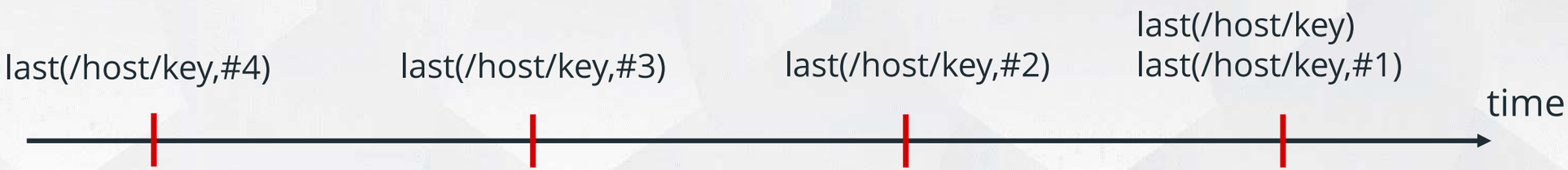
- Using hashtag # will denote the Nth previous value

  last(/host/key,#1) ⇨ Last value

  last(/host/key,#3) ⇨ 3rd previous value

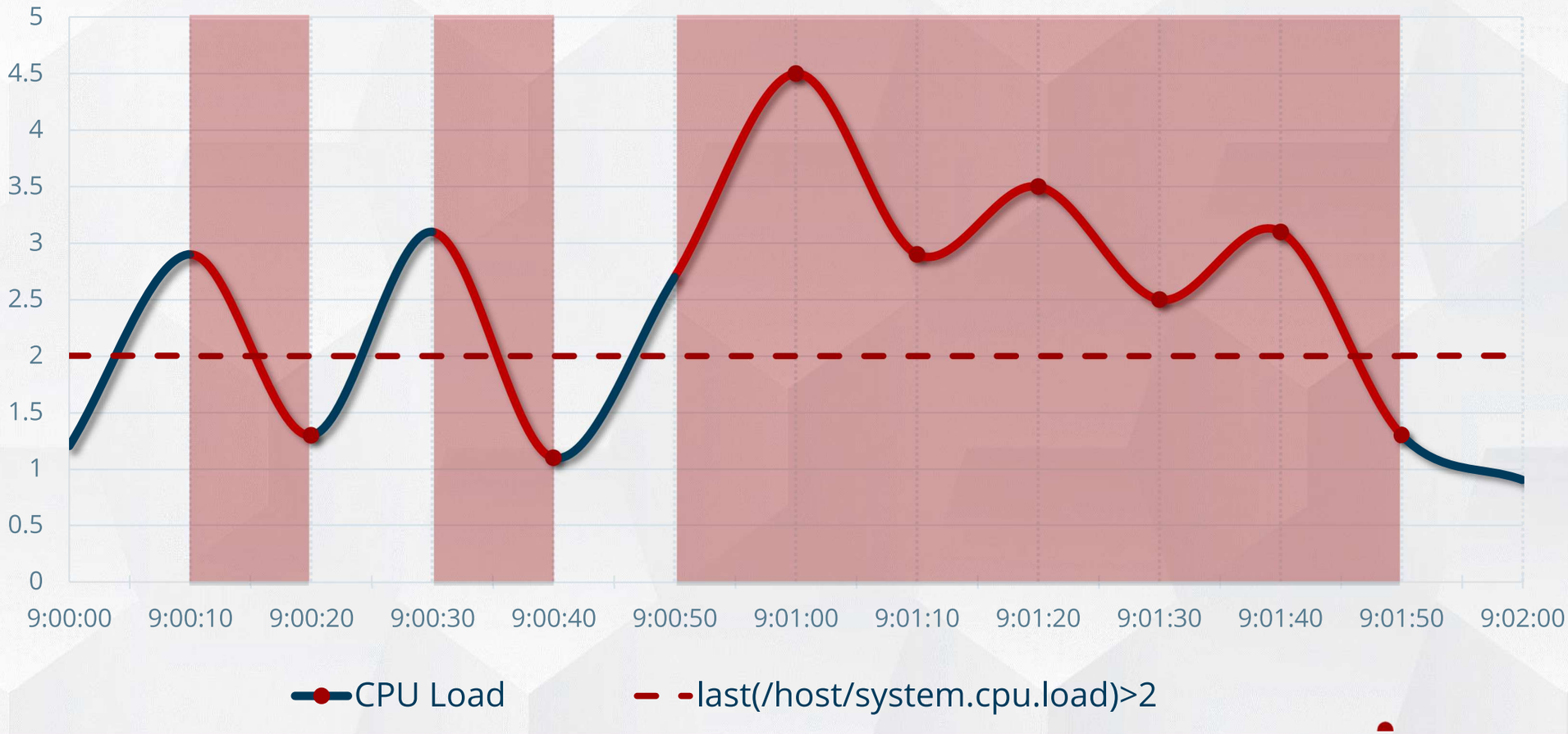
- Time shift parameters are supported

  last(/host/key,#1:now-1h) Value received one hour ago



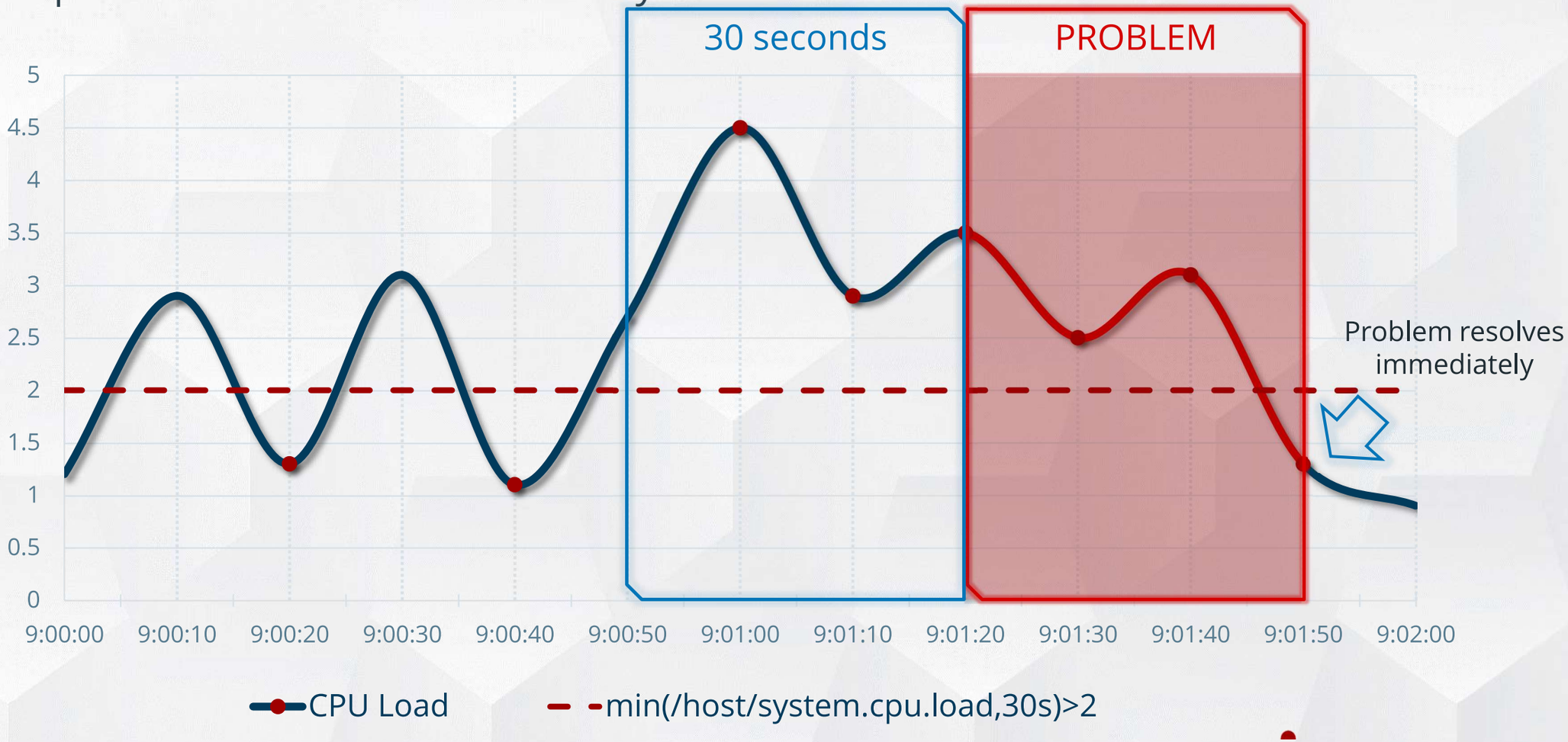
# Function last() is very sensitive:

- Every received value which exceeds the threshold will generate a new alert
- This can lead to "trigger flapping"



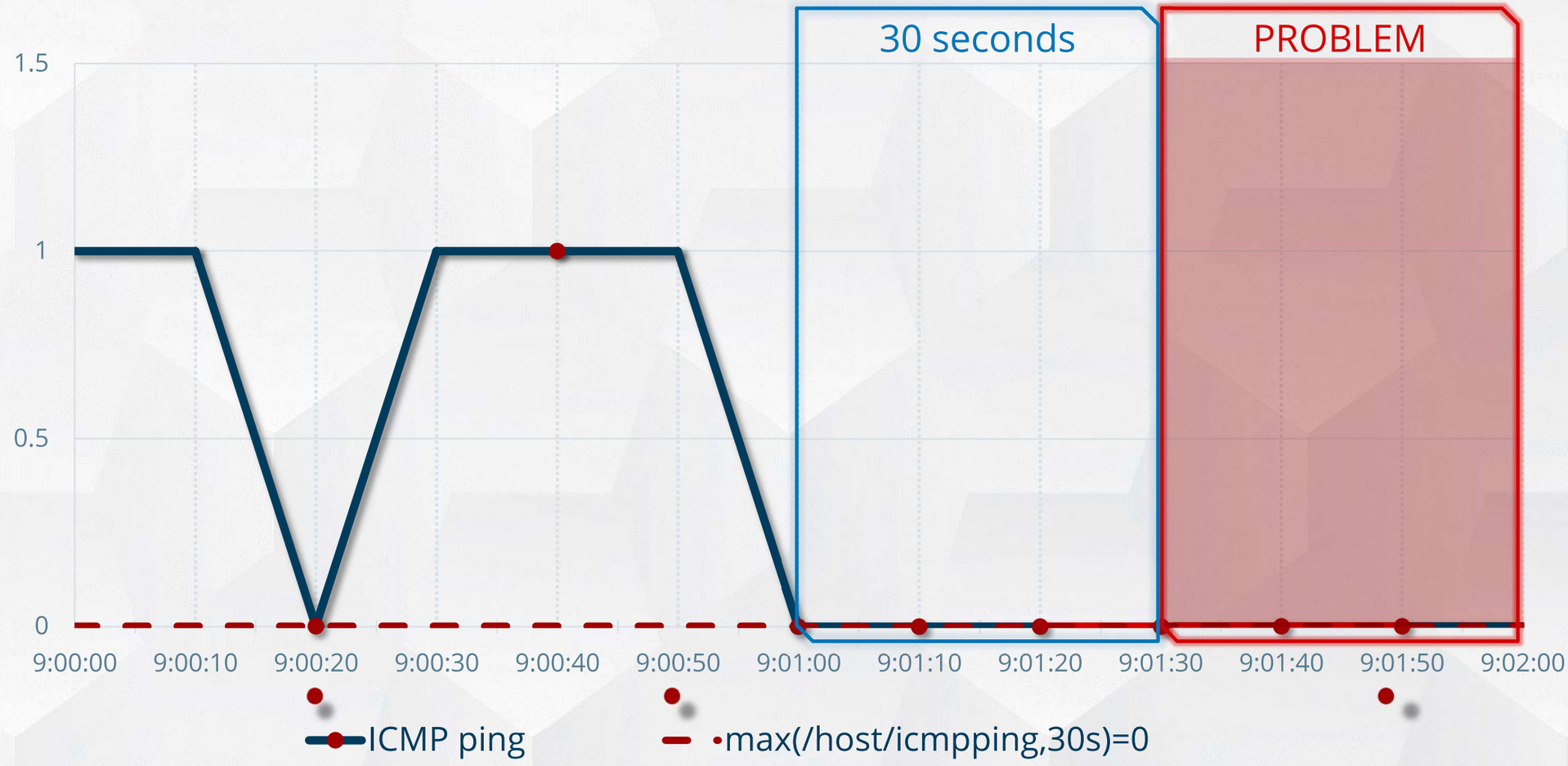
# Function min() is a simple way to reduce false problem detection:

- Minimum value must drop below the threshold for some time period to detect a problem
- The problem will resolve immediately



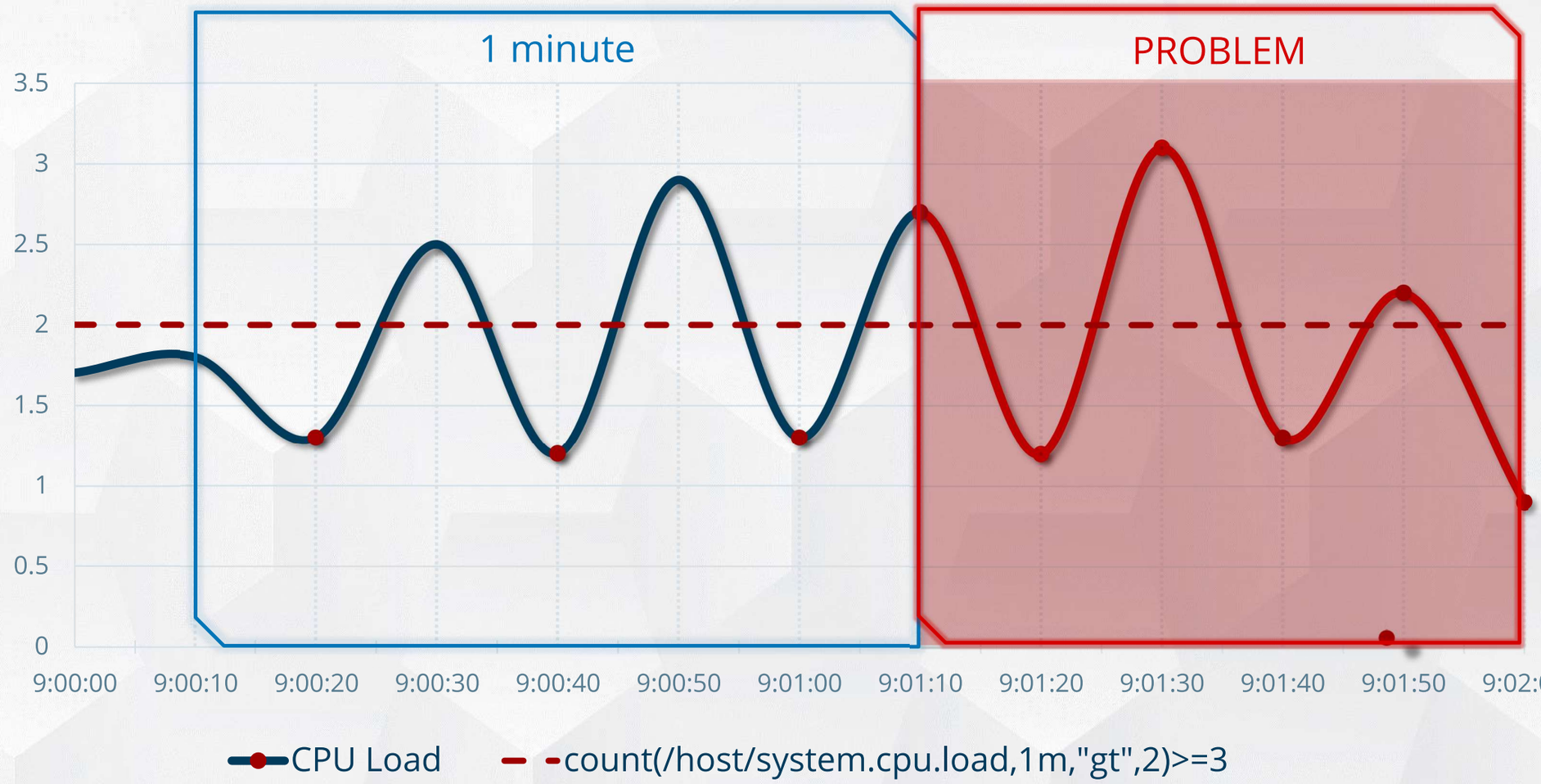
# Function max() may be used to detect availability issues:

- Multiple availability checks in a row must fail to detect a problem



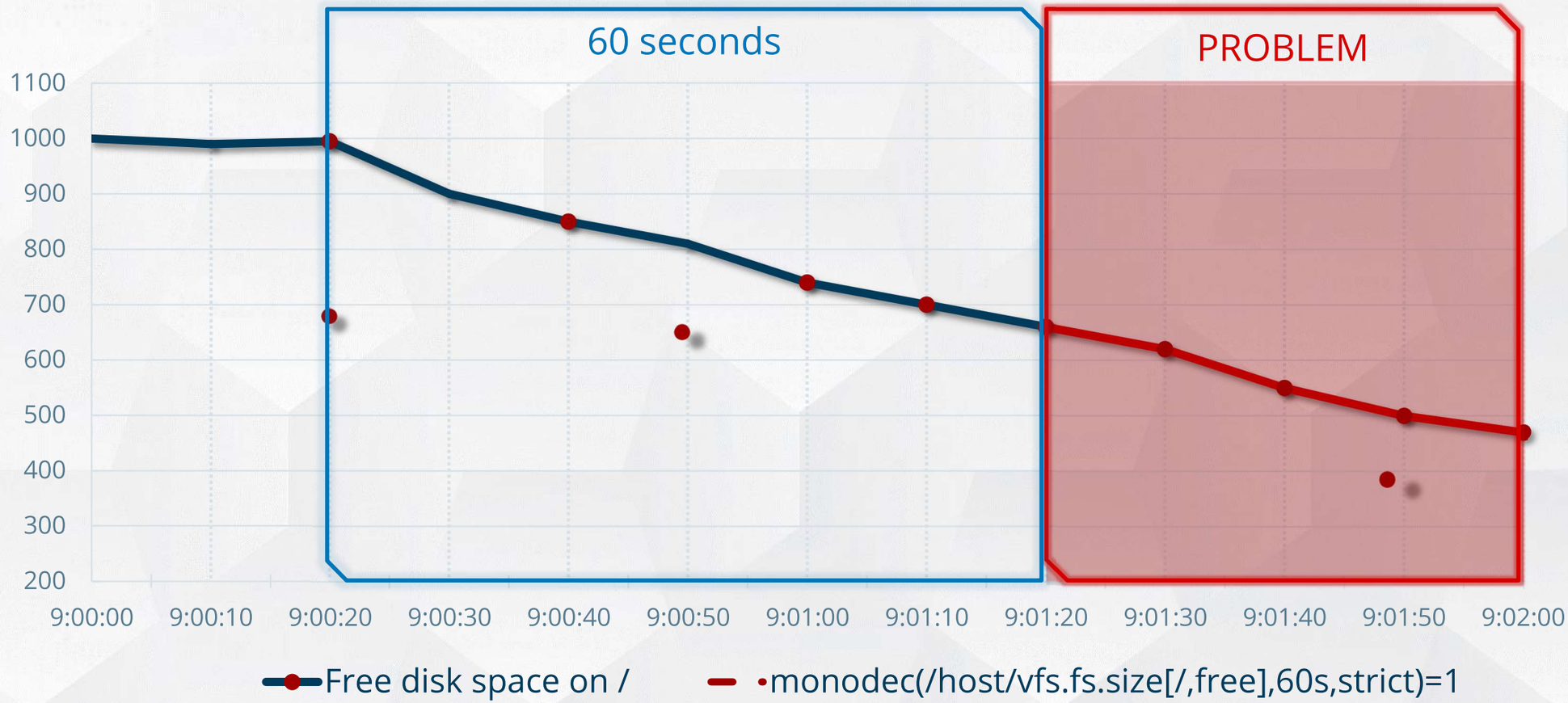
# Function count() is another option:

- More than one anomaly will be required to detect a problem



# Monotonic functions can be used to monitor queues or disk space:

- monoinc() detects monotonic increase of values collected
- monodec() detects monotonic decrease of values collected



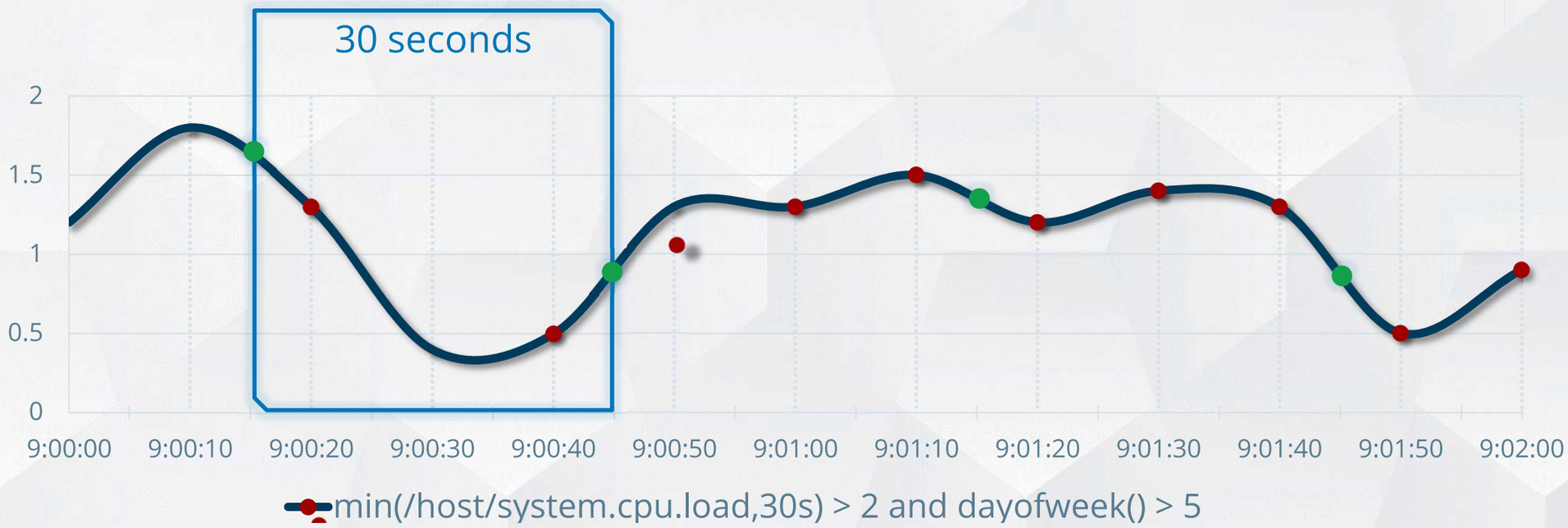
# PRACTICAL SETUP

1) On the host "Training-VM-XX":
   - ✓ Replace last() function for CPU load triggers with a minimum for 1 minute
2) Use "cat /dev/urandom | md5sum" command to test triggers
3) Create a new trigger to compare memory usage:
   - ✓ Compare average free memory for current hour with average free memory for previous hour
   - ✓ Generate alert if the free memory has decreased by more than 25%

# Time-based functions

# All time-based functions are recalculated every 30 seconds:

- If both time-based and non-time-based functions are used in an expression, it is recalculated when a new value is received and additionally every 30 seconds
- Recalculation schedule is distributed evenly between all time-based functions
- Example trigger is recalculated:
  - ✓ every 10 seconds based on the item update interval
  - ✓ additionally, every 30 seconds because time-based function is used in the expression



# DATE AND TIME FUNCTIONS

All date and time functions are time-based:

| | |
|---|---|
| date | current date in YYYYMMDD format |
| time | current time in HHMMSS format |
| dayofweek | current day of week 1-7 (Mon - 1, Sun - 7) |
| dayofmonth | current day of month 1-31 |
| now | current time in HHMMSS format |

These functions can be used to specify time periods from trigger calculation

- Detect problems only on weekends

```
min(/host/system.cpu.load,30s) > 2 and dayofweek() > 5
```

- Ignore scheduled backups between 01:00:00 and 03:00:00

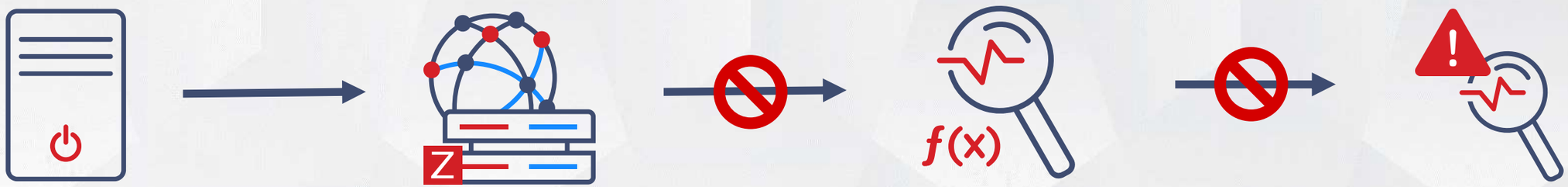
```
avg(/host/system.cpu.util[,iowait],5m) > 5  and (time() < 010000 or time() > 030000)
```

All date and time functions use Zabbix server time zone!

# Time-based history function checks for no data received:

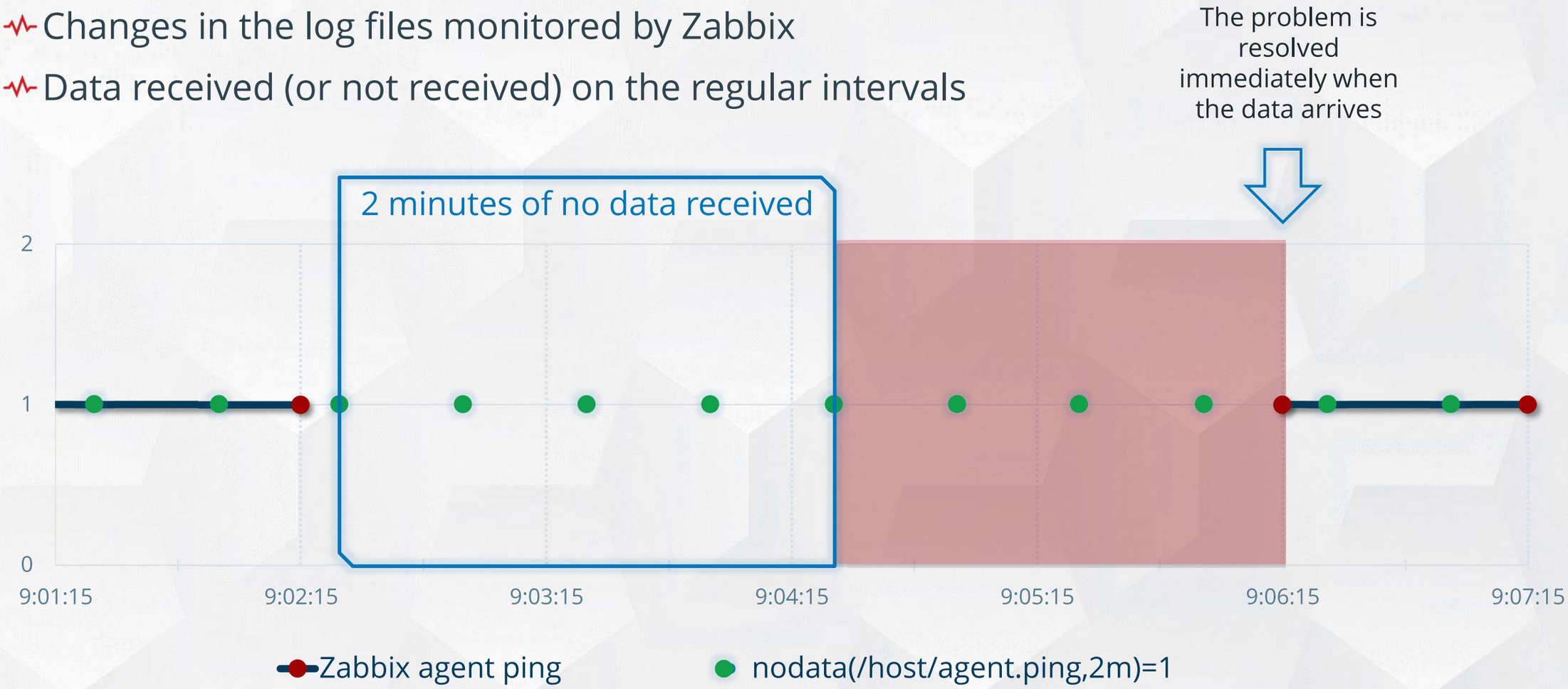
nodata(/host/key,time period,<mode>)

- Time period should not be less than 30 seconds (nodata(0) is not allowed)
- Returns:
  - ✓ 1 - if no data received during the defined period of time
  - ✓ 0 - otherwise
- The 'nodata' triggers monitored by proxy are, by default, sensitive to proxy availability
  - ✓ They will not fire if the data is expected from a proxy, which is currently offline
  - ✓ "strict" mode will ignore proxy availability



# Function nodata() can be used to detect:

- Zabbix agent availability (agent.ping item never returns 0)
- Changes in the log files monitored by Zabbix
- Data received (or not received) on the regular intervals



Zabbix server time zone is used to calculate time-based functions

- User time zone settings may differ from Zabbix server time zone

Time-based functions in triggers with multiple event generation mode will create a new problem every 30 seconds

- The trigger will be evaluated every 30 seconds even if there isn't any new data received

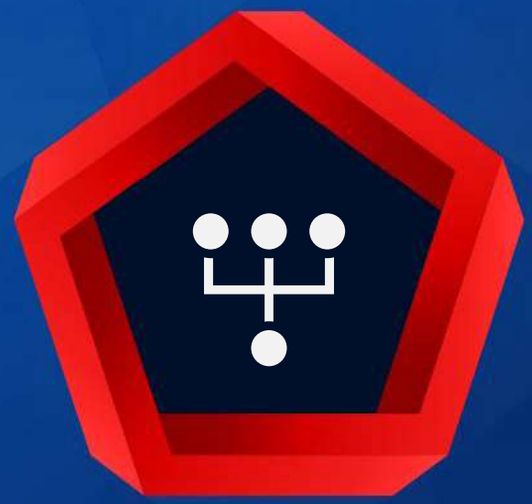
Nodata() function may return false positives if:

- There are time differences between Zabbix server, proxy and agent
- Discard unchanged preprocessing steps are used
- History is not saved for the item

Nodata() function is evaluated for "not supported" items also

# PRACTICAL SETUP

1) On the "Training-VM-XX active checks" host
   - ✓ Create Zabbix agent ping item with the 10 second update interval
   - ✓ Create a trigger to check agent ping last value
2) Stop Zabbix agent on your virtual machine
3) Wait 1 minute to test if the trigger detects a problem
4) On the Training-VM-XX active checks host:
   - ✓ Replace the last() trigger function with no data received for 1 minute
5) Wait 1 minute to test if the trigger detects a problem
6) Start Zabbix agent on your virtual machine



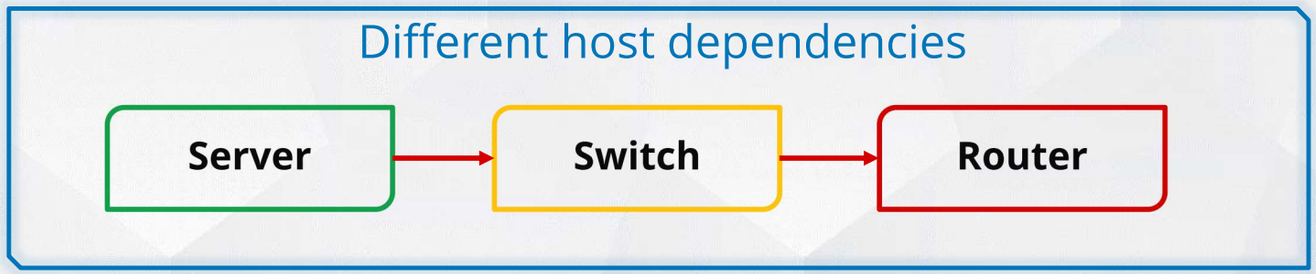
# TRIGGER DEPENDENCIES

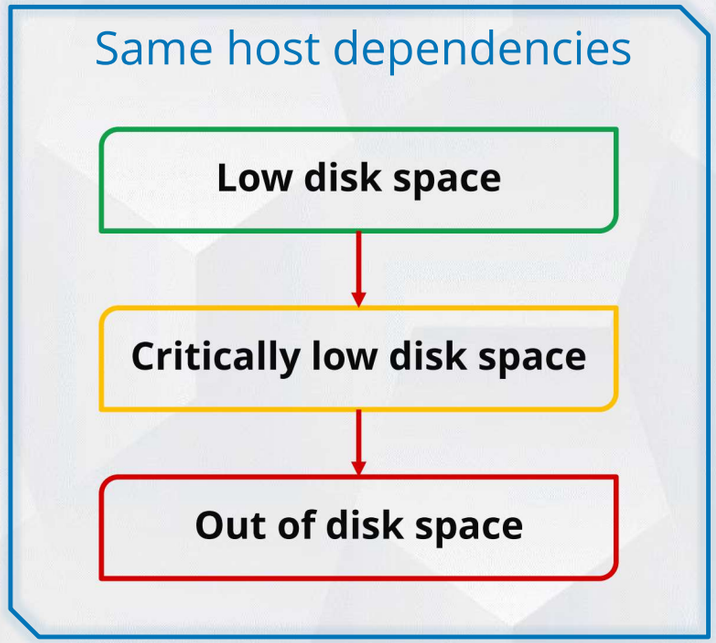
## Dependencies can be defined between multiple triggers:

- Problems will be suppressed if the trigger they depend on is in the PROBLEM state
- Zabbix does not support dependencies between hosts directly

## Dependencies between triggers can be defined:

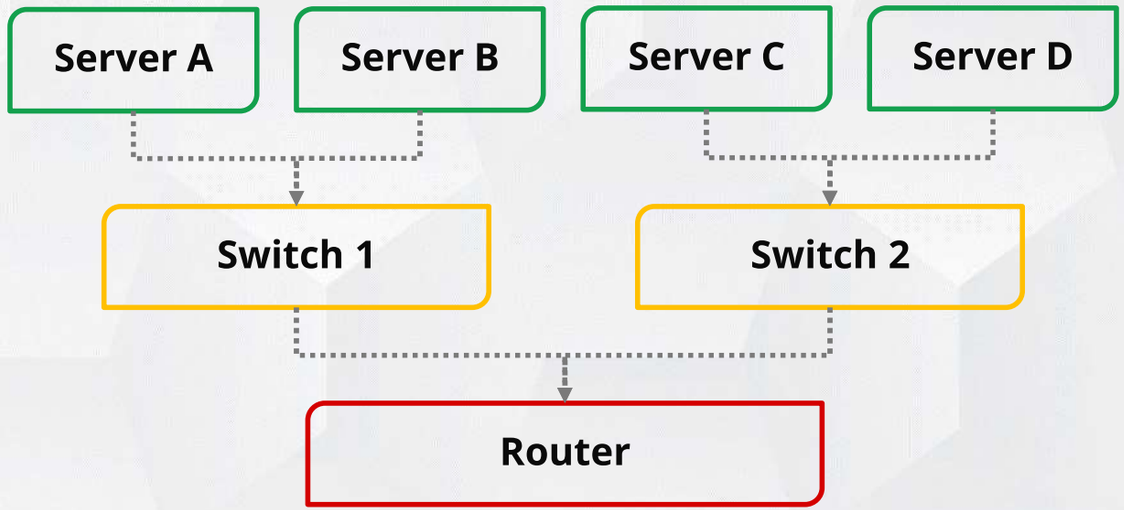
- On the same host:
  - ✓ Problem level (different severities)
- Different hosts:
  - ✓ Network devices
  - ✓ Applications
  - ✓ Other resources



It is possible to create complex architecture with multi-level dependency:

- Multiple levels:
  - ✓ Server A > Switch 1 > Router
- Multiple dependencies:
  - ✓ Switch 1 > Router
  - ✓ Switch 2 > Router




If, for example, a router is down, and dependencies are defined:

- Problems generated by the dependent triggers will be suppressed and hidden
- Zabbix will not execute actions for the dependent trigger
- The dependent trigger will be re-evaluated and will change its state only after the parent trigger returns to the OK state and new metrics are received.

# DEPENDENT TRIGGER EVALUATION

## Dependent trigger will only be re-evaluated when:

- Parent trigger has changed its state to "OK"
- A new value is received for item used in dependent trigger expression

## Triggers are evaluated independently of their dependencies:

- It is possible that problem with a dependent trigger will be detected first
- In this scenario, the dependent trigger will fire as usual
  - ✓ It will become suppressed later when problem with a parent trigger will be detected

# PRACTICAL SETUP

1) On the host "Training-VM-XX":
   - ✓ Create user macro for very high CPU load ( >2.5) with a "High" severity
   - ✓ Create a trigger to detect very high CPU load
2) Create a dependency between triggers for high and very high CPU load
3) Use "yes > /dev/null" command to test this setup