two visualization tools for log files

Eugene Kirpichov, 2010
ekirpichev@griddynamics.com, ekirpichov@gmail.com

Plan

- intro / philosophy
- splot, tplot
  - purpose
  - basics
  - plenty of examples
  - options
- installation

Intro

- I wanted to visualize the behavior of my code
- I only had logs
- I found no existing tools to be good enough
  - suggestions welcome

So I wrote them

tplot (for timeplot)

http://code.google.com/p/timeplot

splot (for stateplot)

http://code.google.com/p/stateplot

P.S. both are in Haskell – “for fun”, but turned out to pay its weight in gold

All new features took a couple lines of code and usually worked immediately

This helped when I *really* needed a feature quickly

Philosophy

- I want to see X!
- If X is in the log, you're almost done.

Easily map logs to tools' input, let tools do the rest.

Philosophy

Do not depend on log format

- `cat log | text-processing oneliner | plot`

Do not depend on domain

- Visualize arbitrary “signals”

Mode of operation

```
cat log.txt | awk '/…/{something simple}'
            | tplot some parameters …
```

You can use perl or whatever, but awk is really freakin' damn simple.

You can learn enough of awk in 1 minute.

`/REGEX/{print something}`, and `$n` is n-th field.

Actually awk is *very* powerful, but simple things are simple

Philosophy

The “awk {*something simple*}” part is <u>really</u> simple

- Adapt to typical log messages

Philosophy

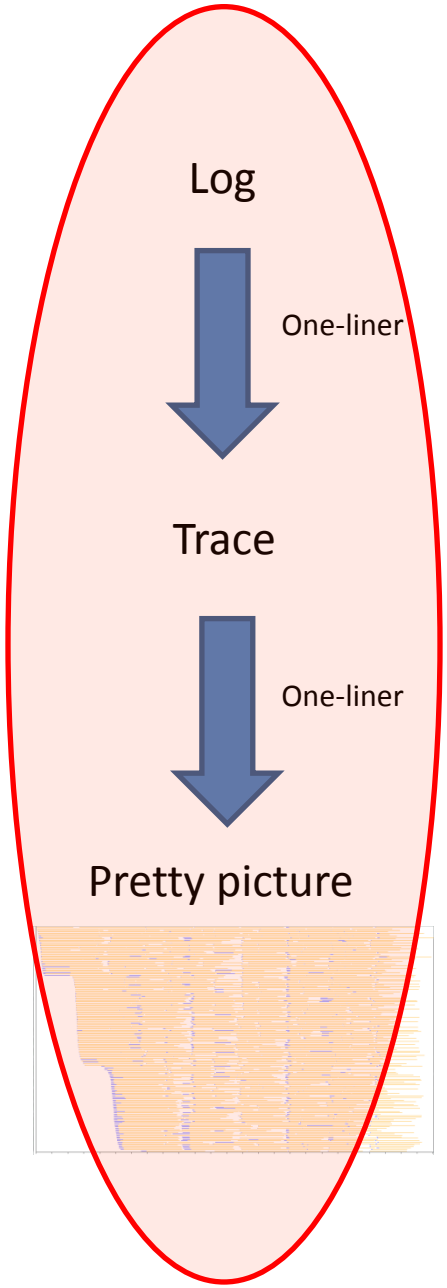
What are typical log messages?

- `An error happened`
- `The request took 100`
- `Machine UNIT001 started/finished reading data`
- `The current temperature is 96 F`
- `Search returned 974 results`
- `URL responded: NOT_FOUND`
- …

Philosophy

What are typical questions?

- Show me the big picture!
- Show me X over time!
- Analyze X over time!
  - Percentiles
  - Buckets
- How did X and Y behave together?
- …




```
UNIT006 2010-11-13 06:23:27.975 P5872 Info Begin 9a444fde86544c7195
```

```
awk ‘{time=$2 “ “ $3; core=$1 “ “ $4} \
     /Begin /       {print time " >" core " blue"} \
     /GetCommonData/{print time " >" core " orange"} \
     /End /         {print time " <" core}‘ \
```

```
2010-12-07 13:52:44.738 >UNIT01-P2368 blue
2010-12-07 13:52:44.912 >UNIT01-P2368 orange
2010-12-07 13:52:44.912 <UNIT01-P2368
```

```
splot -bh 1 -w 1400 -h 800 -expire 10000
```

Tool 1 - splot

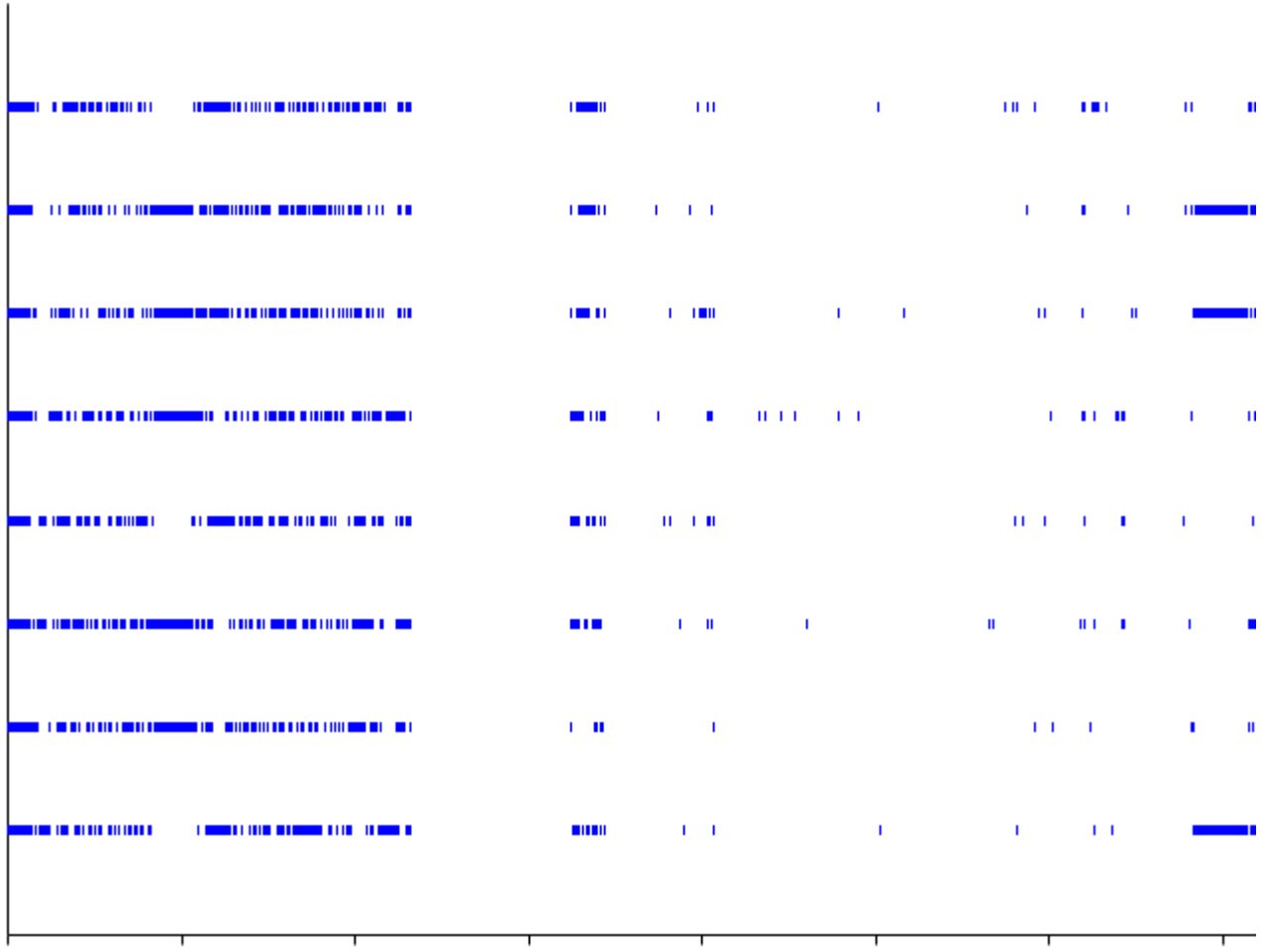
“state plot”

you’ve got a zillion workers

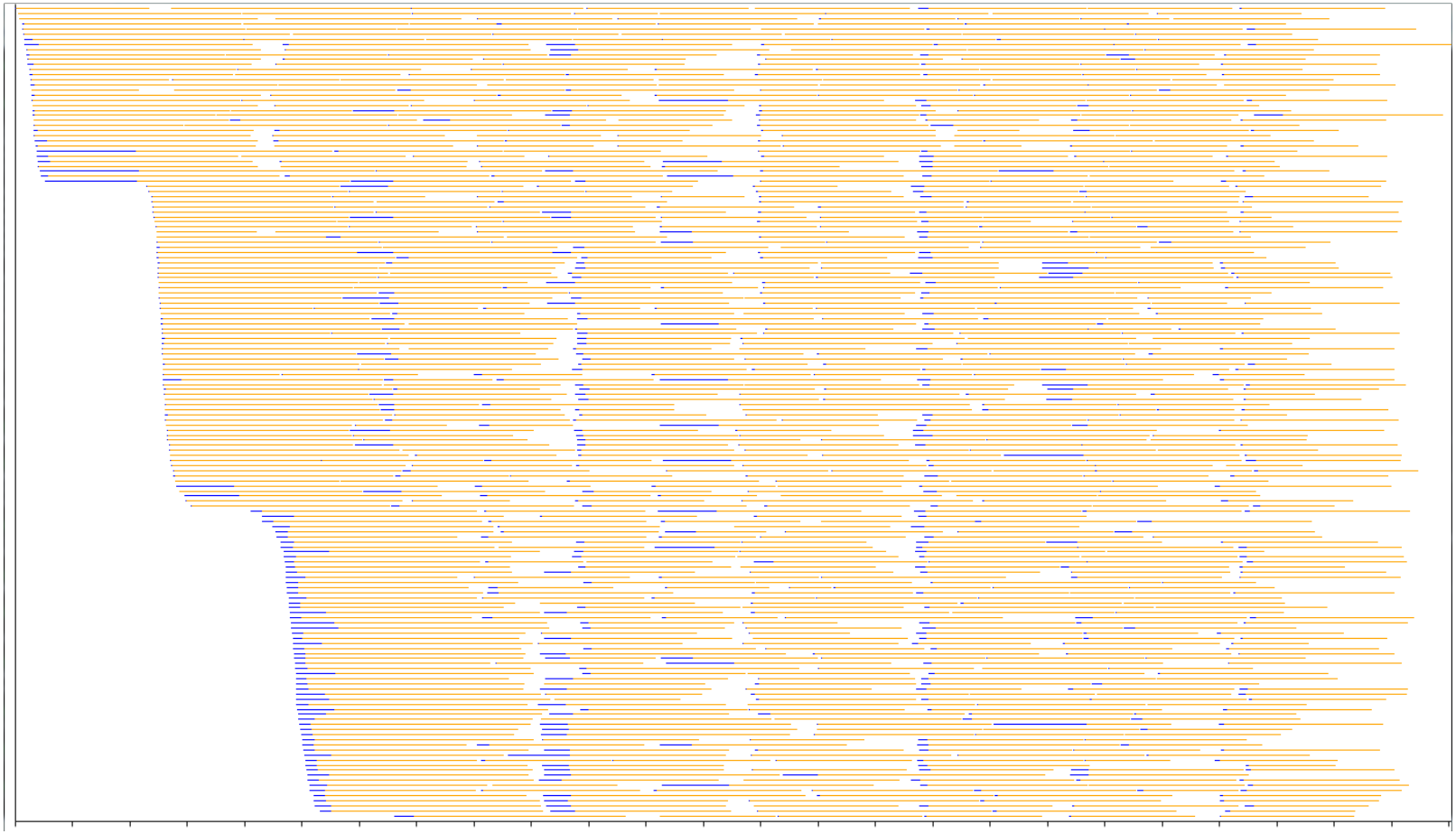
they all work on something

what is the big picture?

1 actor = 1 thread



1 actor = 1 cluster core



How to use it?

```
Usage: splot [-o PNGFILE] [-w WIDTH] [-h HEIGHT] [-bh BARHEIGHT] [-tf TIMEFORMAT]
             [-tickInterval TICKINTERVAL]
  -o PNGFILE    - filename to which the output will be written in PNG format.
                  If omitted, it will be shown in a window.
  -w, -h        - width and height of the resulting picture. Default 640x480.
  -bh           - height of the bar depicting each individual process. Default 5 pixels.
                  Use 1 or so if you have a lot of them.
  -tf           - time format, as in http://linux.die.net/man/3/strptime but with
                  fractional seconds supported via %O - will parse 12.4039 or 12,4039
  -tickInterval - ticks on the X axis will be this often (in millis).
  -sort SORT    - sort tracks by SORT, where: 'time' - sort by time of first event,
                  'name' - sort by track name

Input is read from stdin. Example input (speaks for itself):
2010-10-21 16:45:09,431 >foo green
2010-10-21 16:45:09,441 >bar green
2010-10-21 16:45:10,611 >foo yellow
2010-10-21 16:45:10,721 >foo red
2010-10-21 16:45:10,930 >bar blue
2010-10-21 16:45:11,322 <foo
2010-10-21 16:45:12,508 <bar

'>FOO COLOR' means 'start a bar of color COLOR on track FOO',
'<FOO' means 'end the current bar for FOO'.
```

BLA BLA BLA

Log

```
UNIT006 2010-11-13 06:23:27.975 P5872 Info Begin 9a444fde86544c7195
```

One-liner

```
awk '{time=$2 " " $3; core=$1 " " $4} \
     /Begin /       {print time " >" core " blue"} \
     /GetCommonData/{print time " >" core " orange"} \
     /End /         {print time " <" core}' \
```

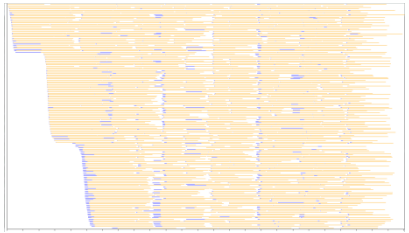
Trace (stdin)

```
2010-12-07 13:52:44.738 >UNIT01-P2368 blue
2010-12-07 13:52:44.912 >UNIT01-P2368 orange
2010-12-07 13:52:44.912 <UNIT01-P2368
```

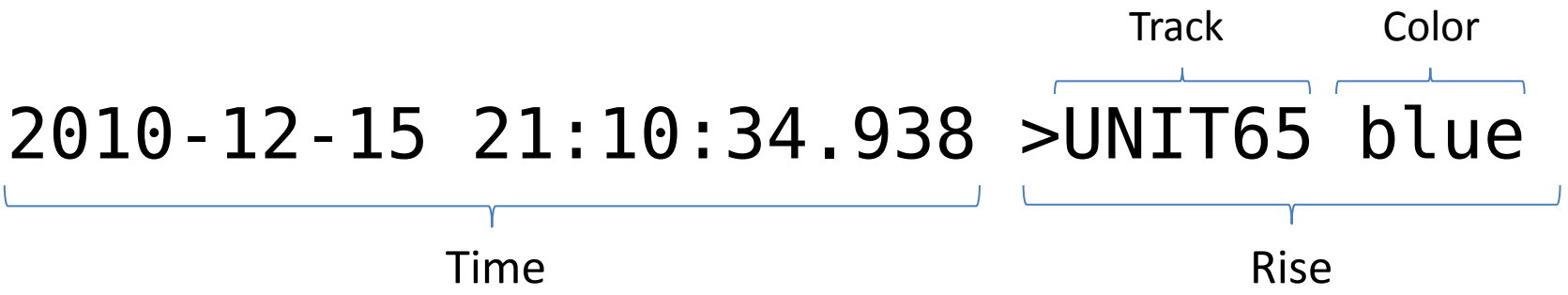
One-liner

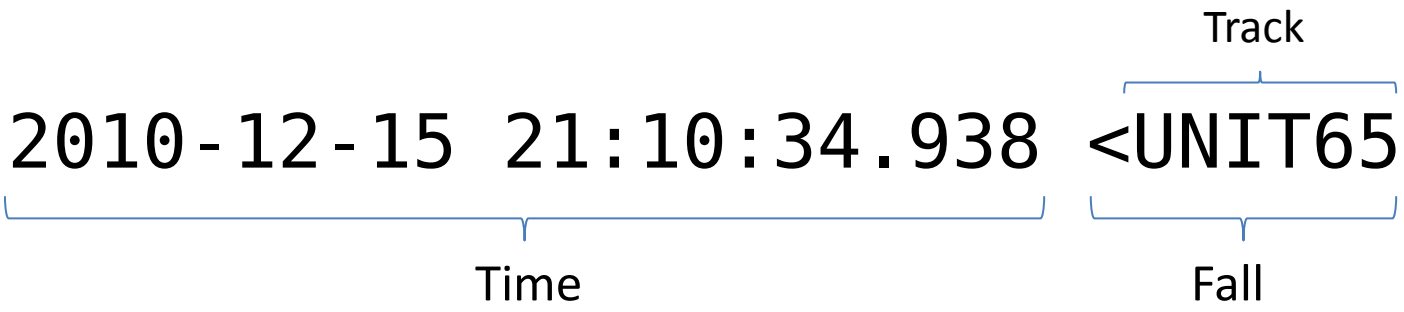
```
splot -bh 1 -w 1400 -h 800 -expire 10000
```

Pretty picture



Trace format





Trace format

```
TIME >ACTOR COLOR
TIME <ACTOR

2010-12-07 13:52:44.738 >UNIT01-P2368 blue
2010-12-07 13:52:44.912 >UNIT01-P2368 orange
2010-12-07 13:52:44.912 <UNIT01-P2368
```

Log

```
UNIT006 2010-11-13 06:23:27.975 P5872 Info Begin 9a444fde86544c7195
```

One-liner

```
awk '{time=$2 " " $3; core=$1 " " $4} \
     /Begin /       {print time " >" core " blue"} \
     /GetCommonData/{print time " >" core " orange"} \
     /End /         {print time " <" core}' \
```

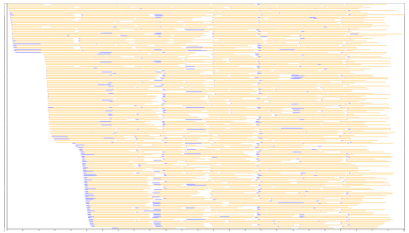
Trace

```
2010-12-07 13:52:44.738 >UNIT01-P2368 blue
2010-12-07 13:52:44.912 >UNIT01-P2368 orange
2010-12-07 13:52:44.912 <UNIT01-P2368
```

One-liner

```
splot -bh 1 -w 1400 -h 800 -expire 10000
```

Pretty picture

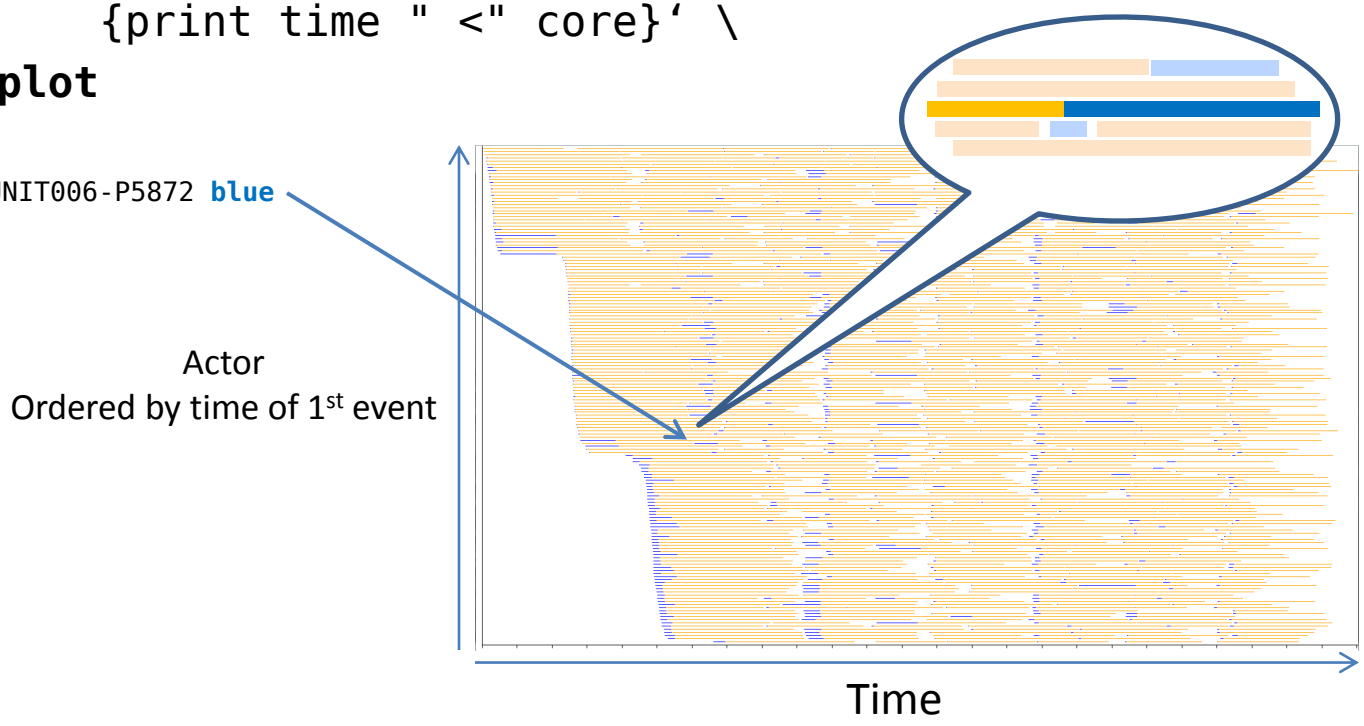


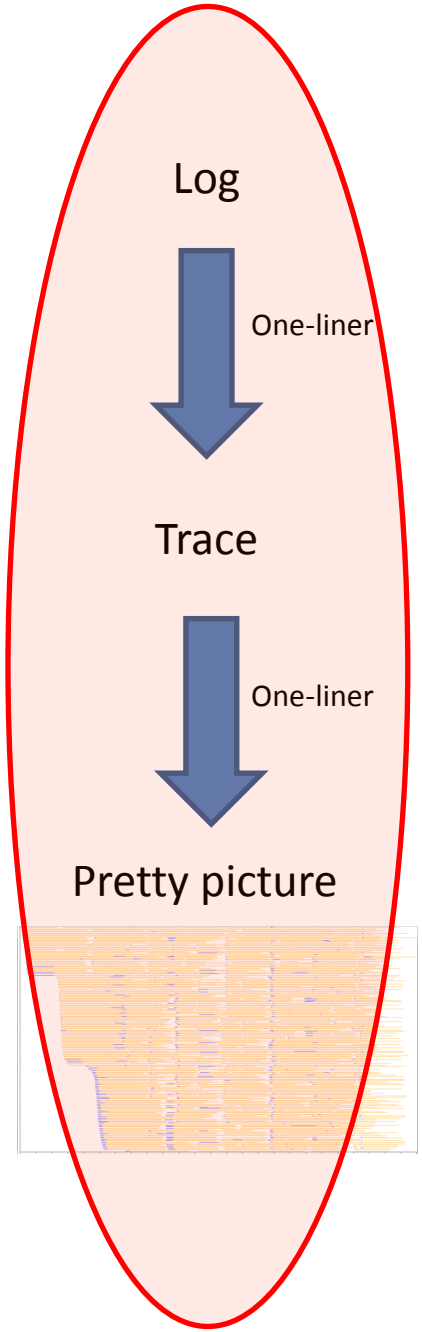
From log to trace

```
UNIT006 2010-11-13 06:23:27.975 P5872 Info Begin 9a444fde86544c7
```

```
awk ‘{time=$3 “ “ $4; core=$1 “ “ $9} \
     /Begin /        {print time " >" core " blue"} \
     /GetCommonData/{print time " >" core " orange"} \
     /End /          {print time " <" core}‘ \
     log.txt | splot
```

```
2010-11-13 06:23:27.975 >UNIT006-P5872 blue
```






```
UNIT006 2010-11-13 06:23:27.975 P5872 Info Begin 9a444fde86544c7195
```

```
awk ‘{time=$2 “ “ $3; core=$1 “ “ $4} \
     /Begin /       {print time " >" core " blue"} \
     /GetCommonData/{print time " >" core " orange"} \
     /End /         {print time " <" core}‘ \
```

```
2010-12-07 13:52:44.738 >UNIT01-P2368 blue
2010-12-07 13:52:44.912 >UNIT01-P2368 orange
2010-12-07 13:52:44.912 <UNIT01-P2368
```

```
splot -bh 1 -w 1400 -h 800 -expire 10000
```

How to create a PNG

```
-o out.png
```

How to change window size

```
-w 1400 -h 800
```

What if the bars are too thick?

```
-bh 1
```

(for example, 1 bar per process, 2000 processes)

What if the ticks are too often?

```
-tickInterval 5000
```

(for example, the log spans 1.5 hours)

What if the time format is not

`%Y-%m-%d %H:%M:%OS`?

`-tf '[%H-%M-%OS %Y/%m/%d]'`

(man strptime)
(%OS for fractional seconds)

What if I want to sort on actor name (not time of first event)?

`-sort name`

(for example, your actor names are like “JOB-MACHINE-PID”, and you want to clearly differentiate jobs in output)

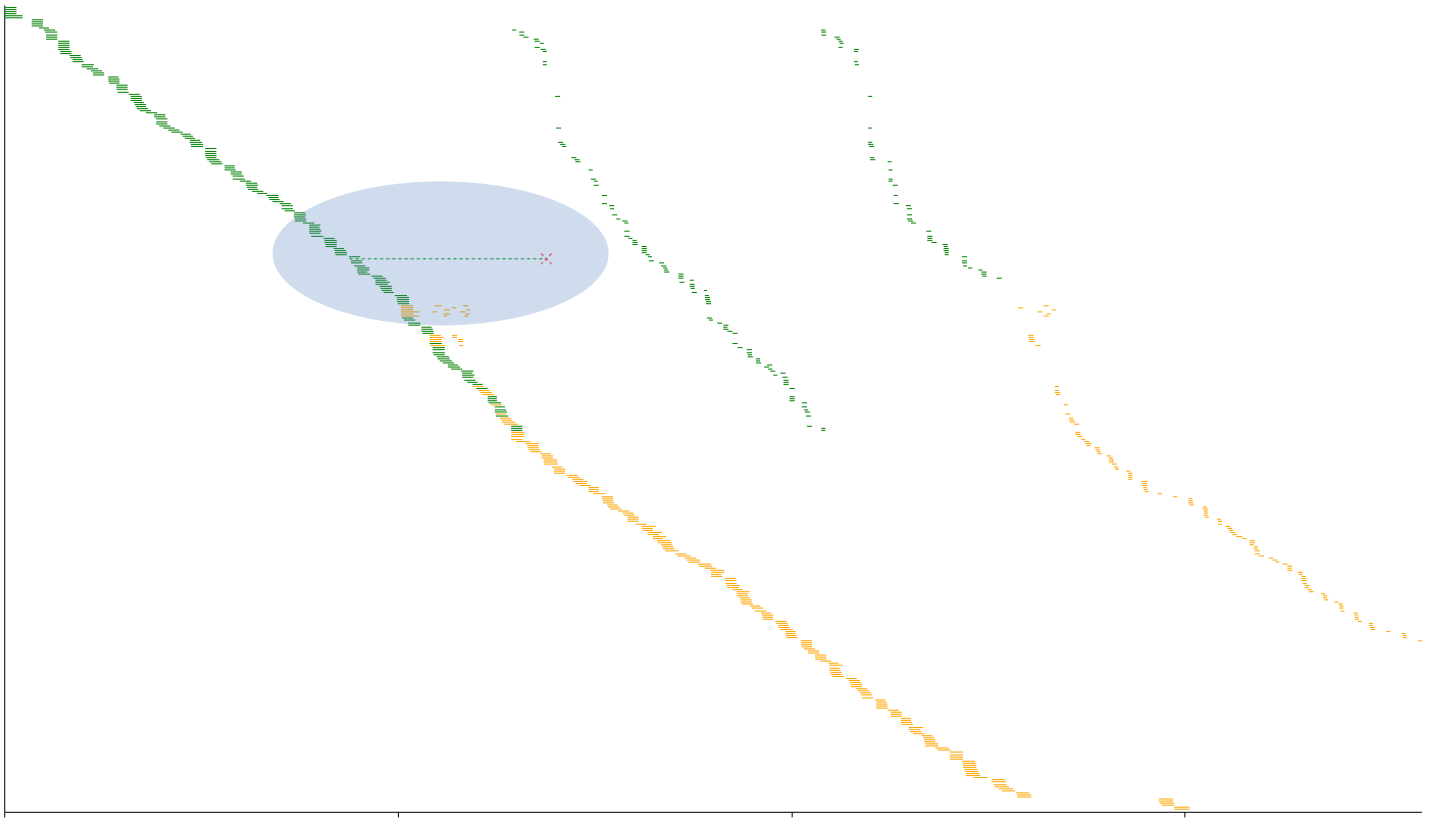
What if the ‘<‘ is lost?

- Process was killed before it said ‘Done with X’

- Assume X takes no more than T
- Use “`-expire T`”

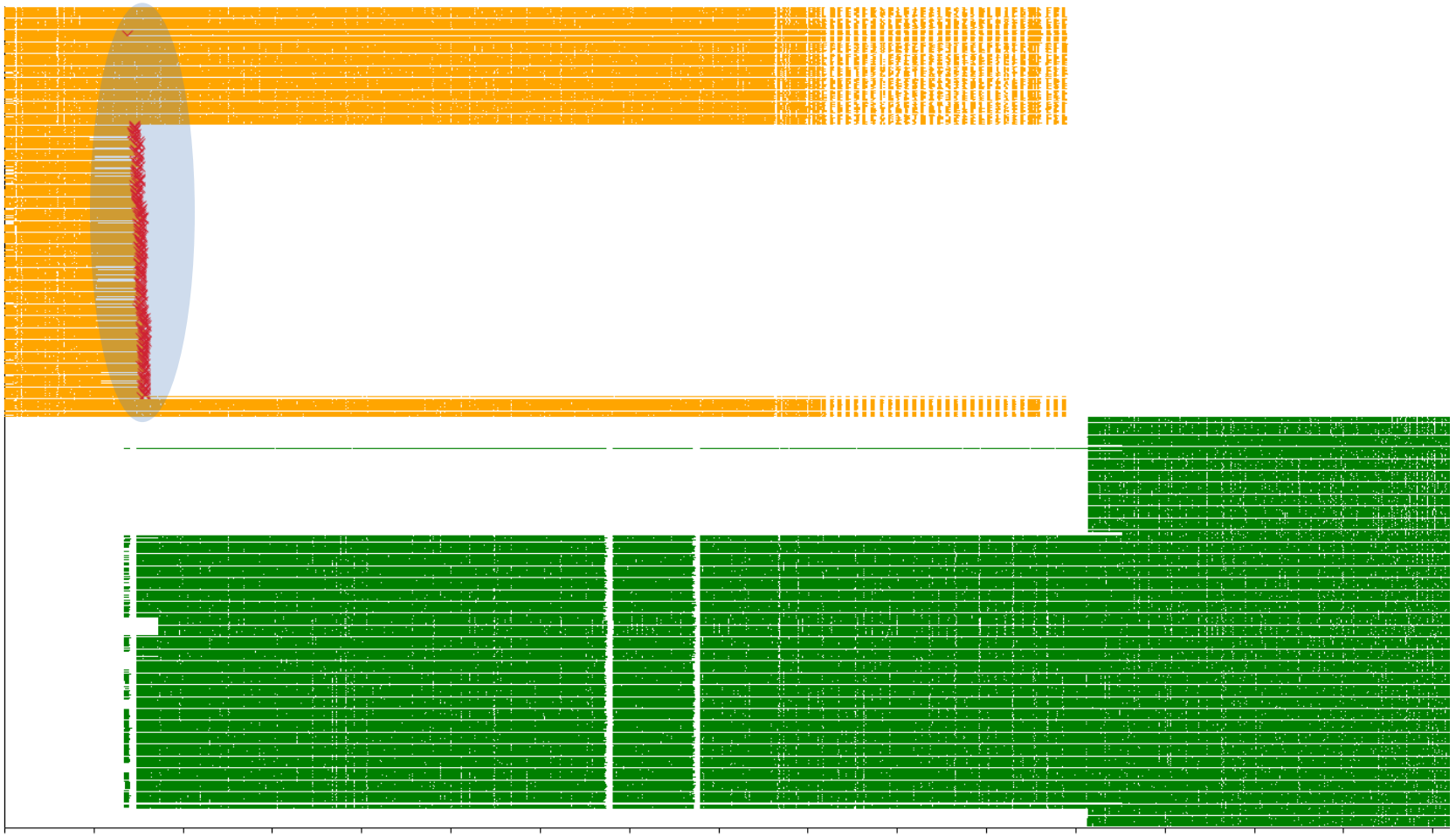
What if the ‘<‘ is lost?

```
-expire 10000
```



What if the ‘<‘ is lost?

```
-expire 10000
```



What if the ‘>’ is lost?

- You’re processing a log in pieces

- Use ‘`-phantom COLOR`’.
- Tracks starting with ‘<‘ will be prepended with ‘`>COLOR`’.

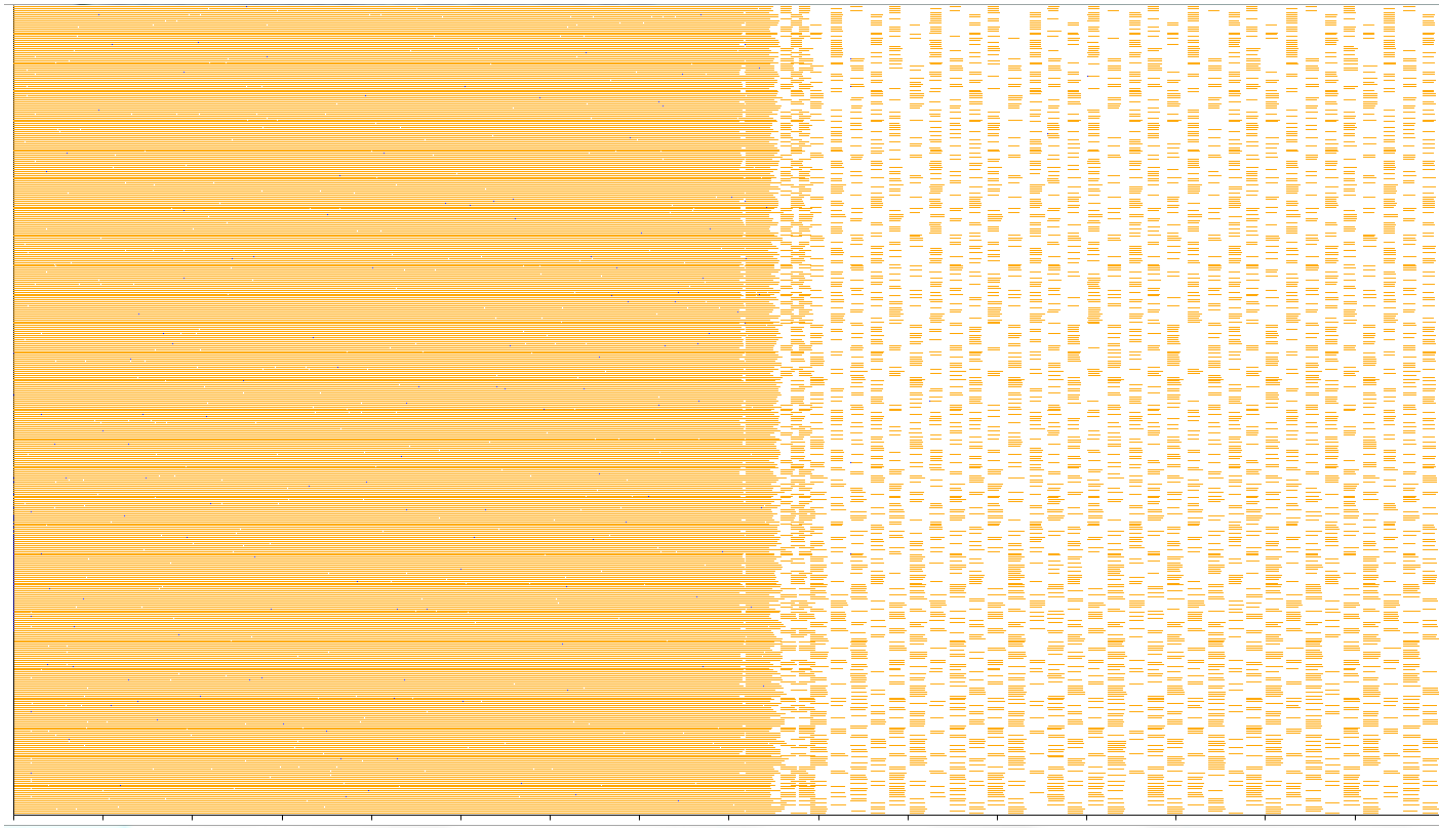
So what can it do, again?

It can help you see a pattern

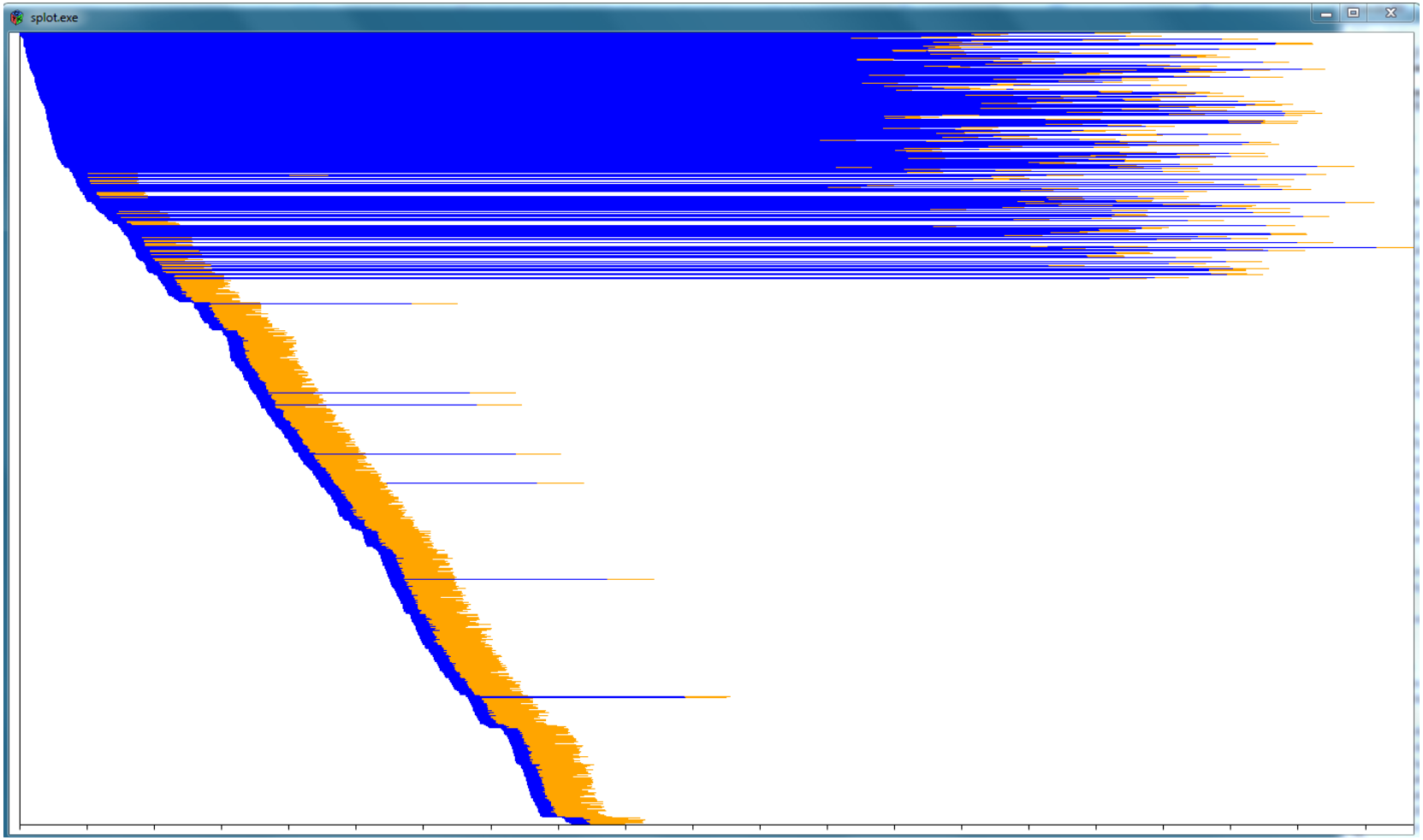
that is hard/impossible to see by other means
(just like any other visualization)

P.S. All examples below are drawn by one-liners.

N jobs run concurrently,
then all but one finish,
it continues sequentially:
too sequentially to saturate the cluster.

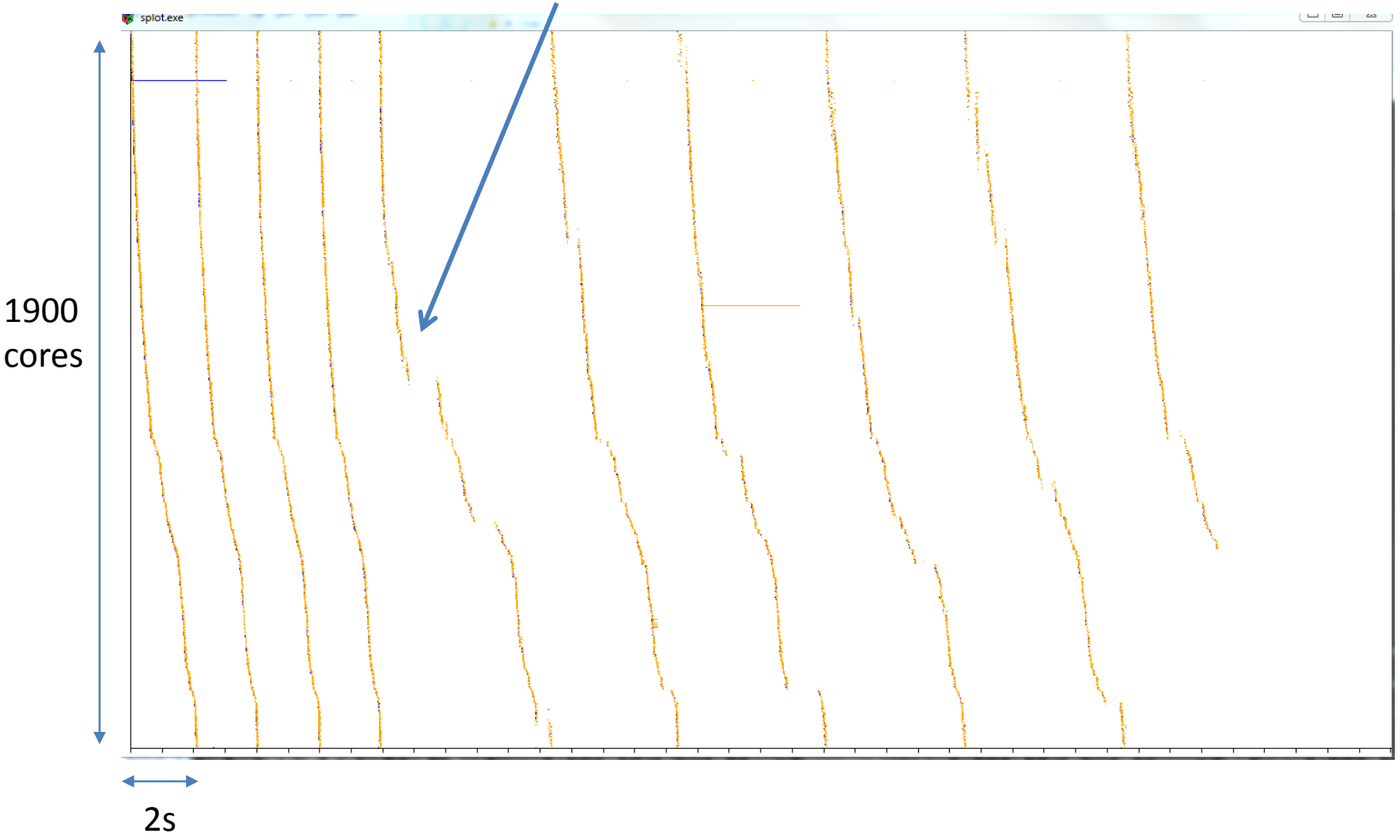


For the early tasks, fetching data takes a pathologically large time.
Sometimes it takes a lot of time for other tasks, too, but not *that* much.

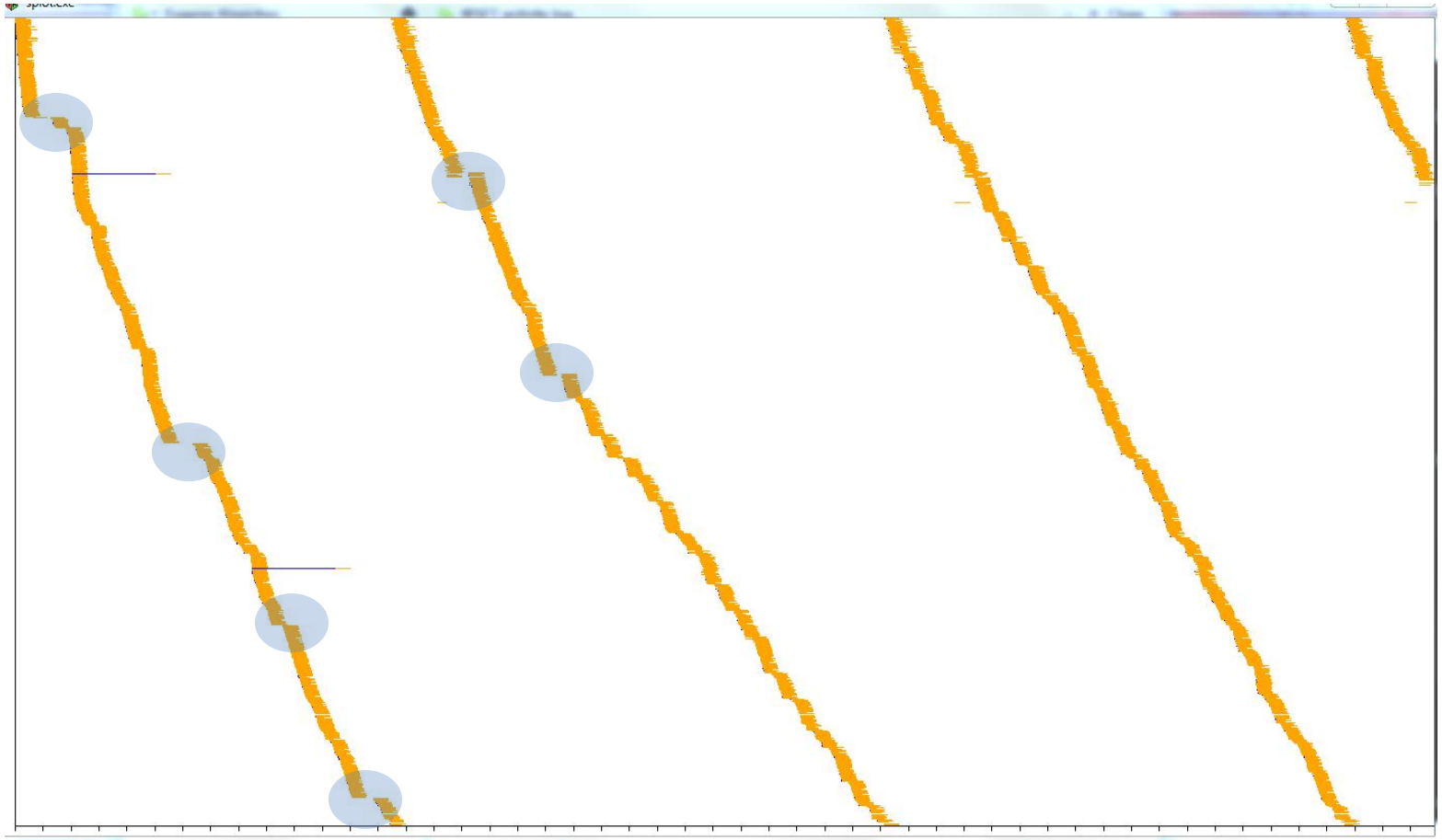


We’re spraying tasks all over the cluster as fast as we can (900/s),
but they are just too short.

Spraying slows down over time.



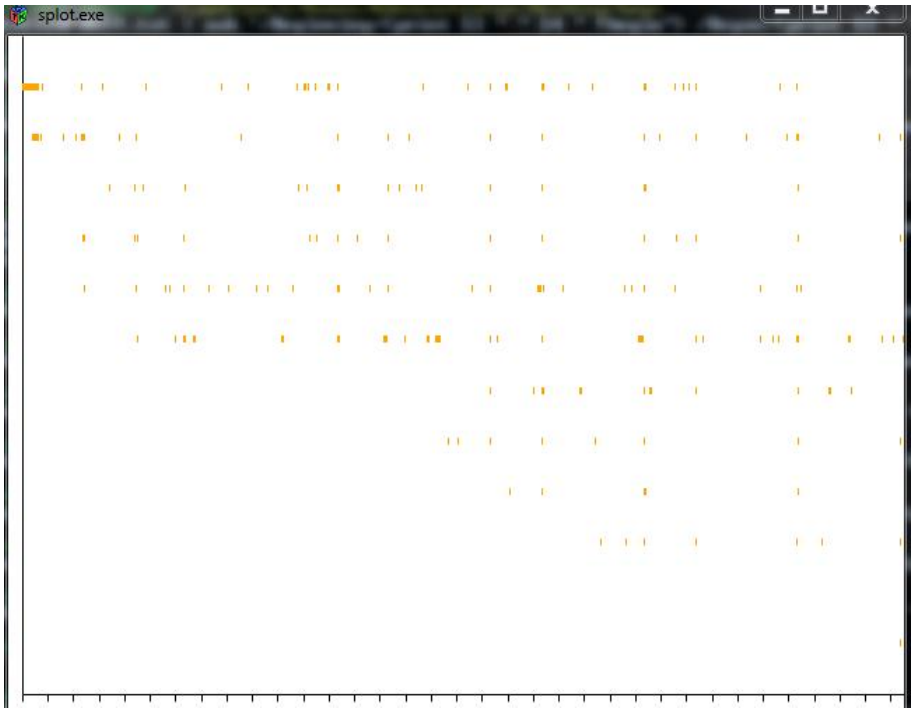
Utilization is better, but there are some strange pauses.



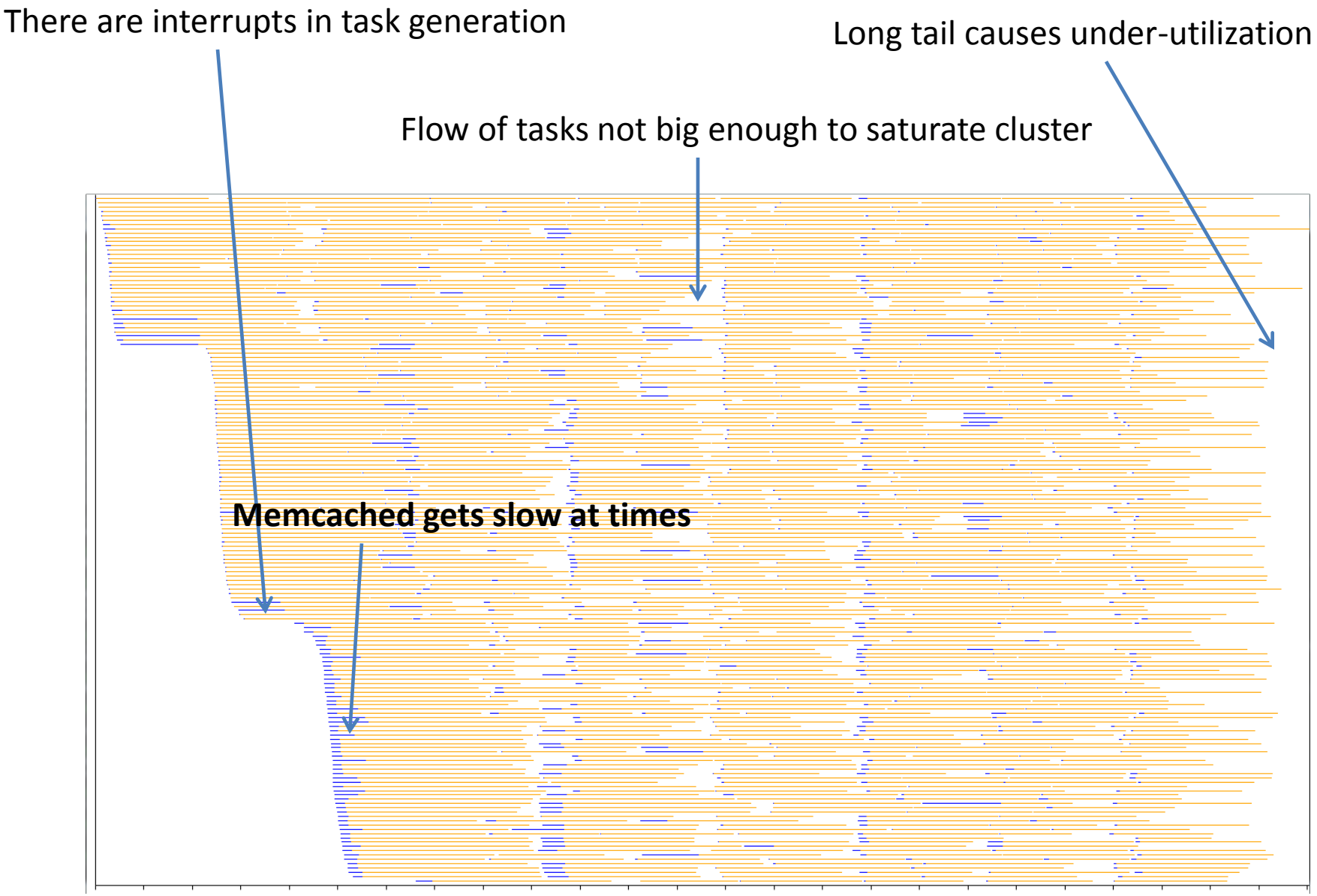
Component A calls component B

Component A's impression

Component B's impression



Diagnosis: Slow inter-component transport!


There are interrupts in task generation
Long tail causes under-utilization
Flow of tasks not big enough to saturate cluster
Memcached gets slow at times

Guidelines

What are the actors?

- Processes
  - Name them like “MACHINE-PID-THREAD” or “JOB-MACHINE-PID-THREAD”
  - Make sure your log is verbose enough for that
- Tasks
  - Better show those who process them (not tasks themselves)

Guidelines

What are the states?

- Example: “fetch data”, then “compute”, then “write result”
- **Make sure your log shows boundaries**

```
{time=…; actor=…}
/Started fetching/{print time “ >” actor “ blue”}
/Computing…/      {print time “ >” actor “ orange”}
/Done computing/  {print time “ >” actor “ green”}
/Written result/  {print time “ <“ actor}
```

Guidelines

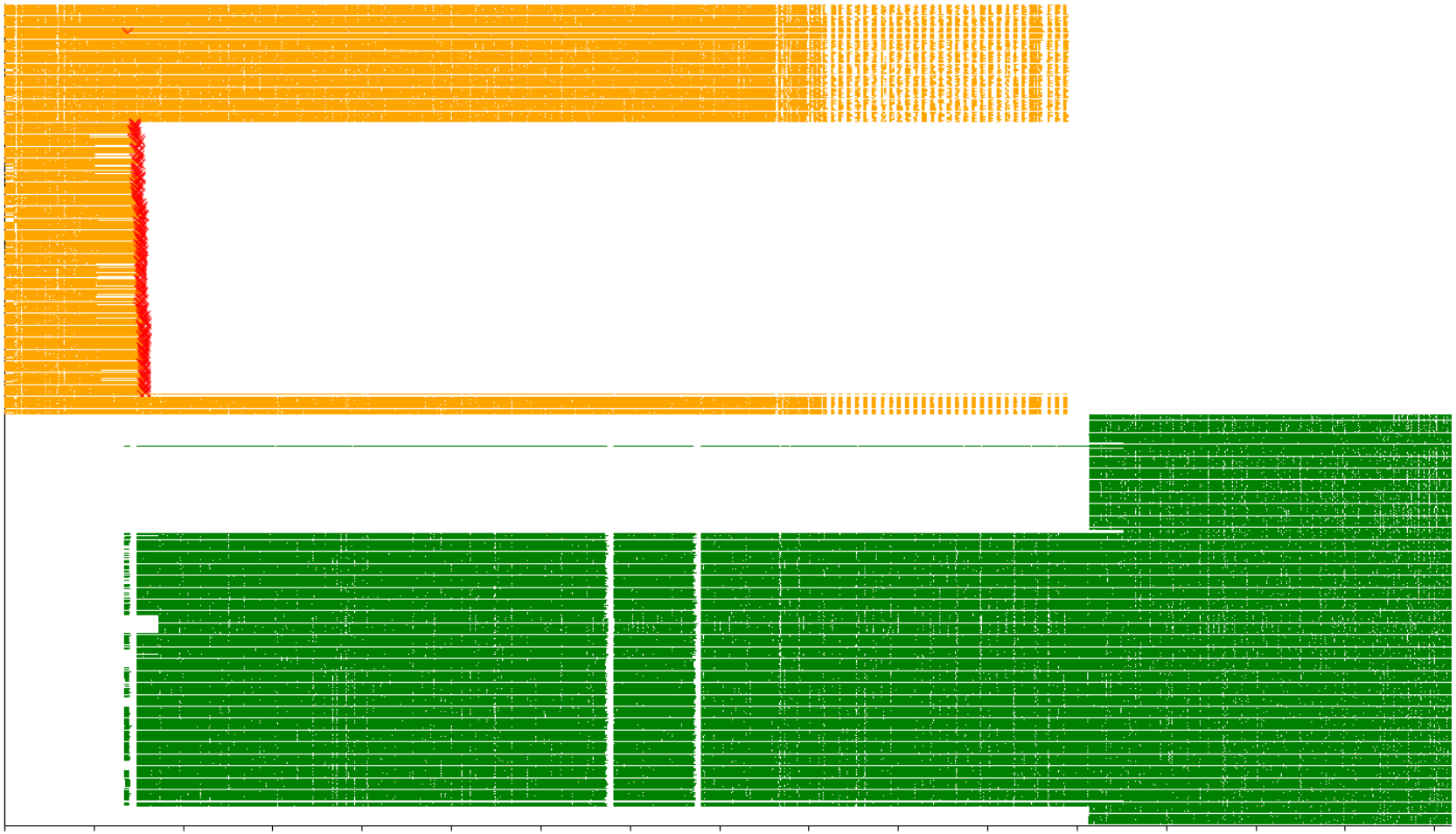
How to differentiate between actor groups?

- Make them of different colour

Log: …. Deliver **JOBID.TASKID**

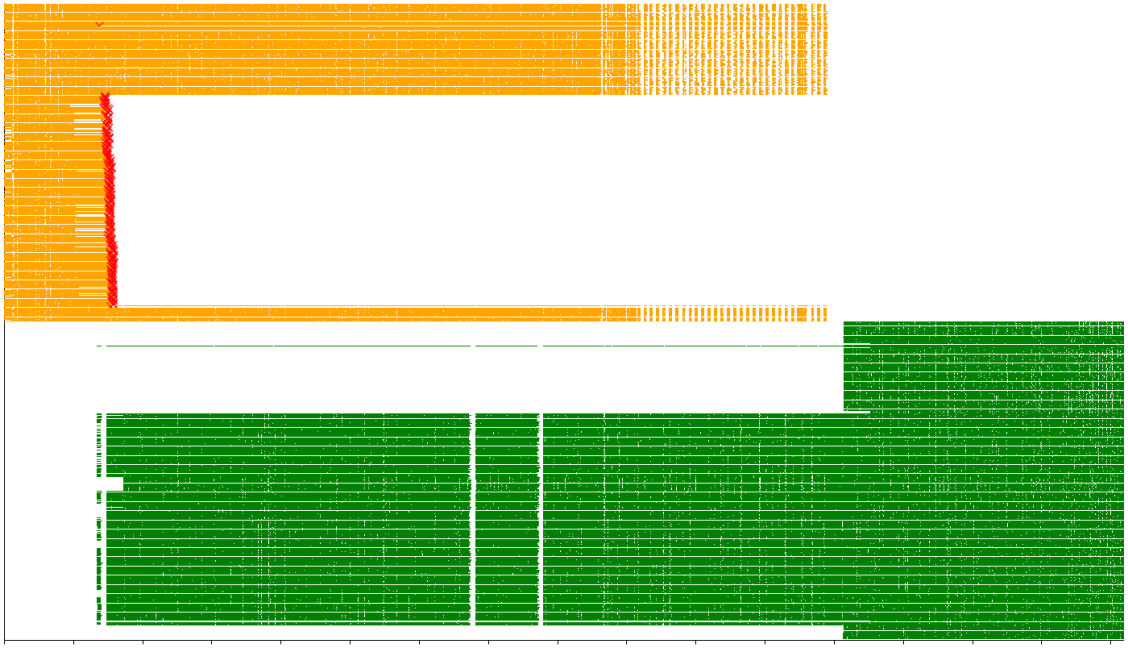
```
BEGIN    {color[0]="green"; color[1]="orange"}
         {time=$3 " " $4; core=$1 "-" $9}
/Deliver/{id=$NF; sub(/\..*/,"",id); job[core]=id;
          print time " >" job[core] "-" core " " color[id%2]}
/End /   {print time " <" job[core] "-" core}
```

That’s how it looks. One job preempts the other.
(1st job’s processes are killed, 2nd’s are spawned)



Example

```
awk 'BEGIN{color[0]="green"; color[1]="orange"}                              \
     {time=$3 " " $4; core=$1 "-" $9}                                        \
     /Deliver/{id=$NF; sub(/\..*/,"",id); job[core]=id;                      \
               print time " >" job[core] "-" core " " color[id%2]}           \
     /End /   {print time " <" job[core] "-" core}‘                          \
     "$f"                                                                    \
  | splot -bh 1 -w 1400 -h 800 -tickInterval 30000 -o "$f.utilization.png“   \
    -sort time -expire 15000
```



P.S.

To draw a “global picture” you’ll need a global time axis.

Try **greg** – http://code.google.com/p/greg

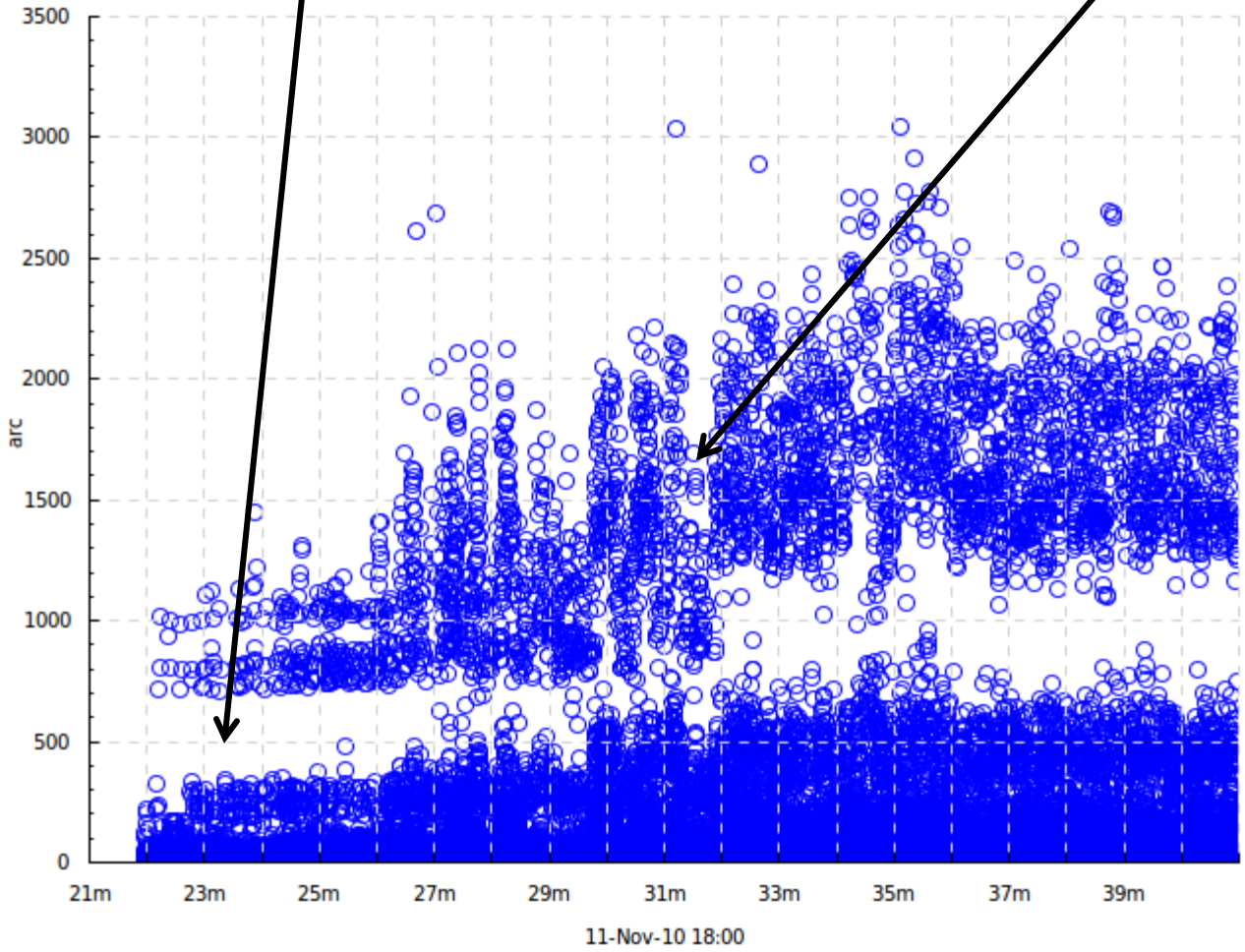
Tool 2 - tplot

"time plot"

how do these quantitative characteristics change together over time?

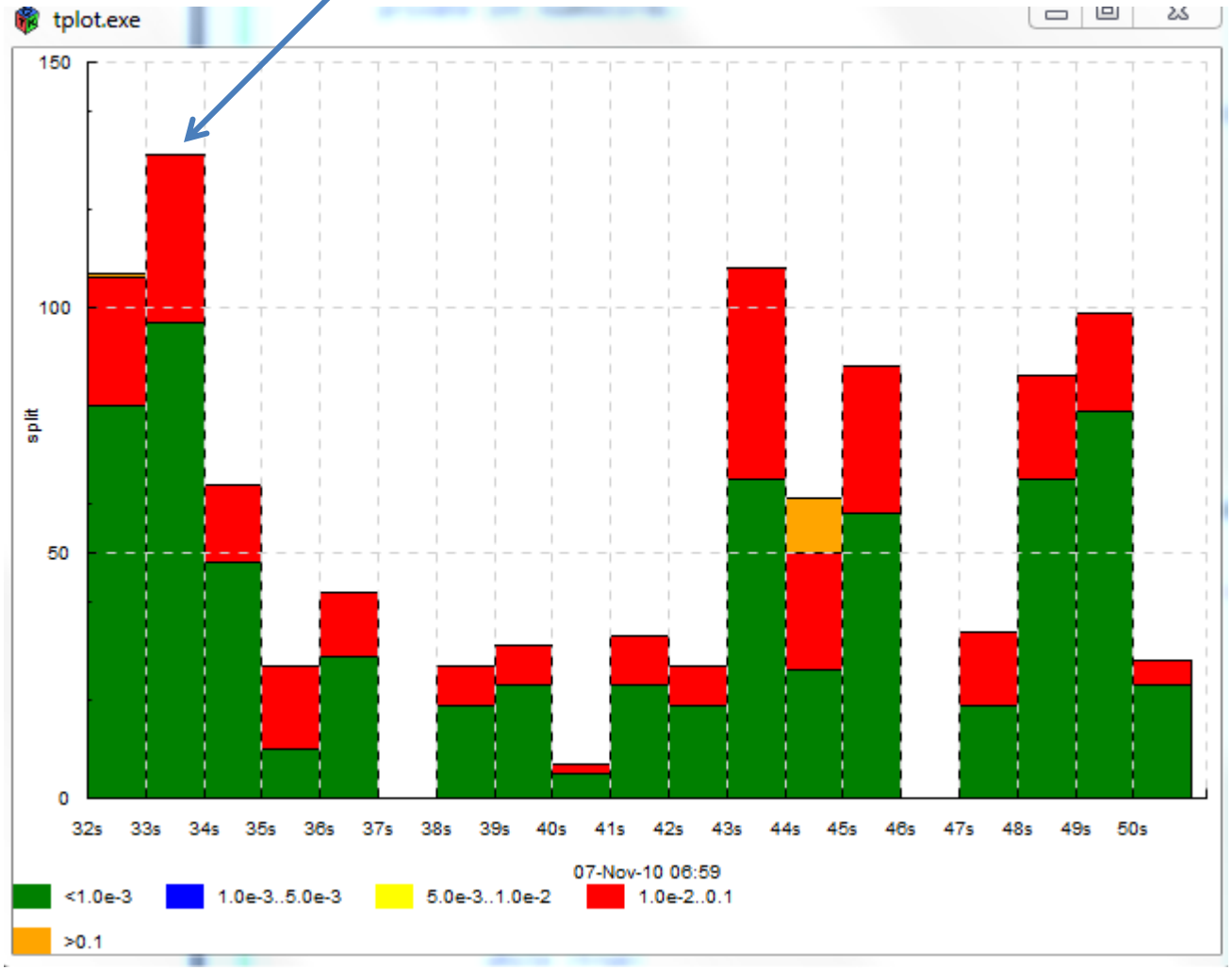
When load increased, both hits and misses got more expensive

Request times are clustered in 2 groups (cache **hits** and **misses**)

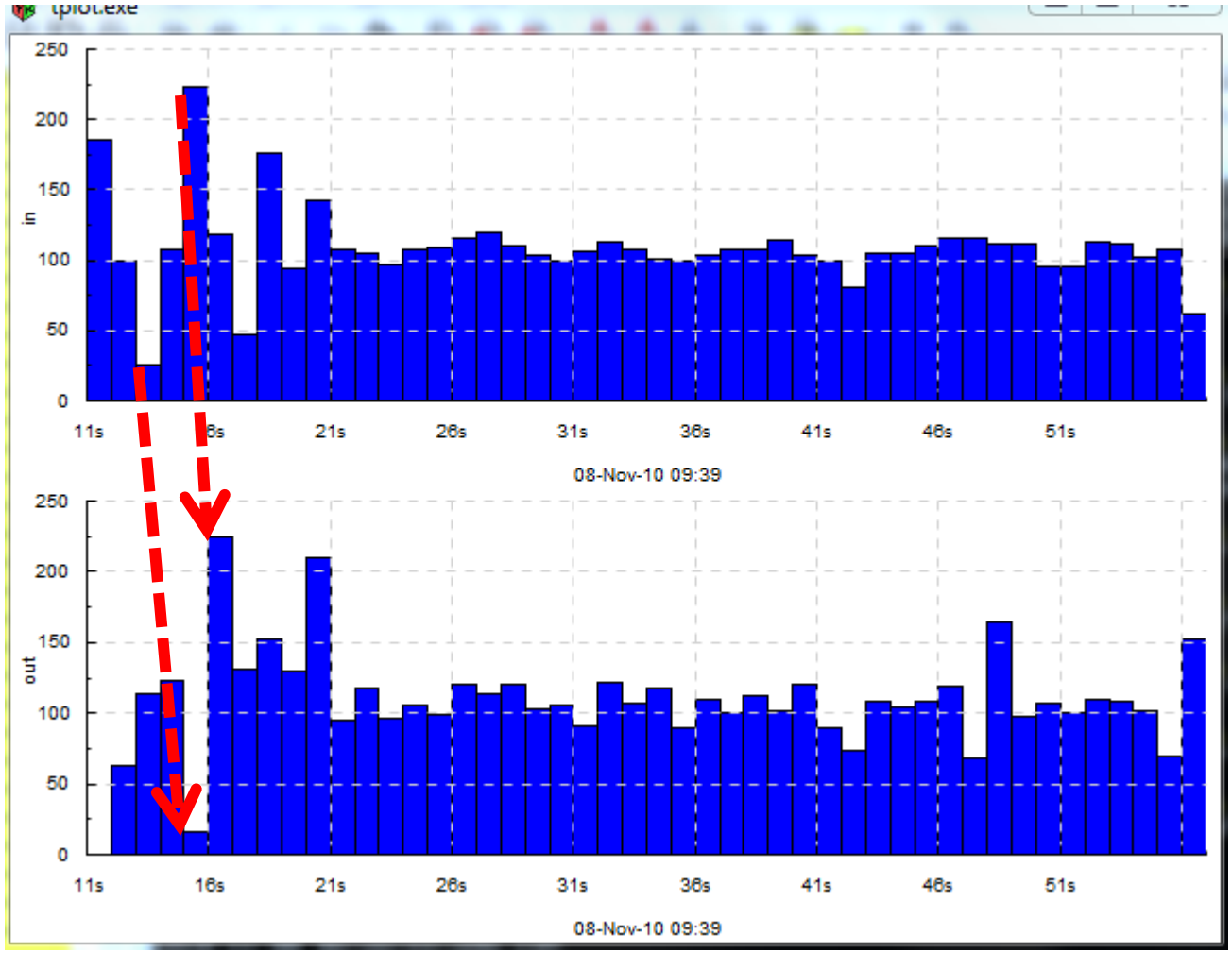


"arc" is for "arcadia" (Yandex Server) – it's from a load test of rabota.yandex.ru

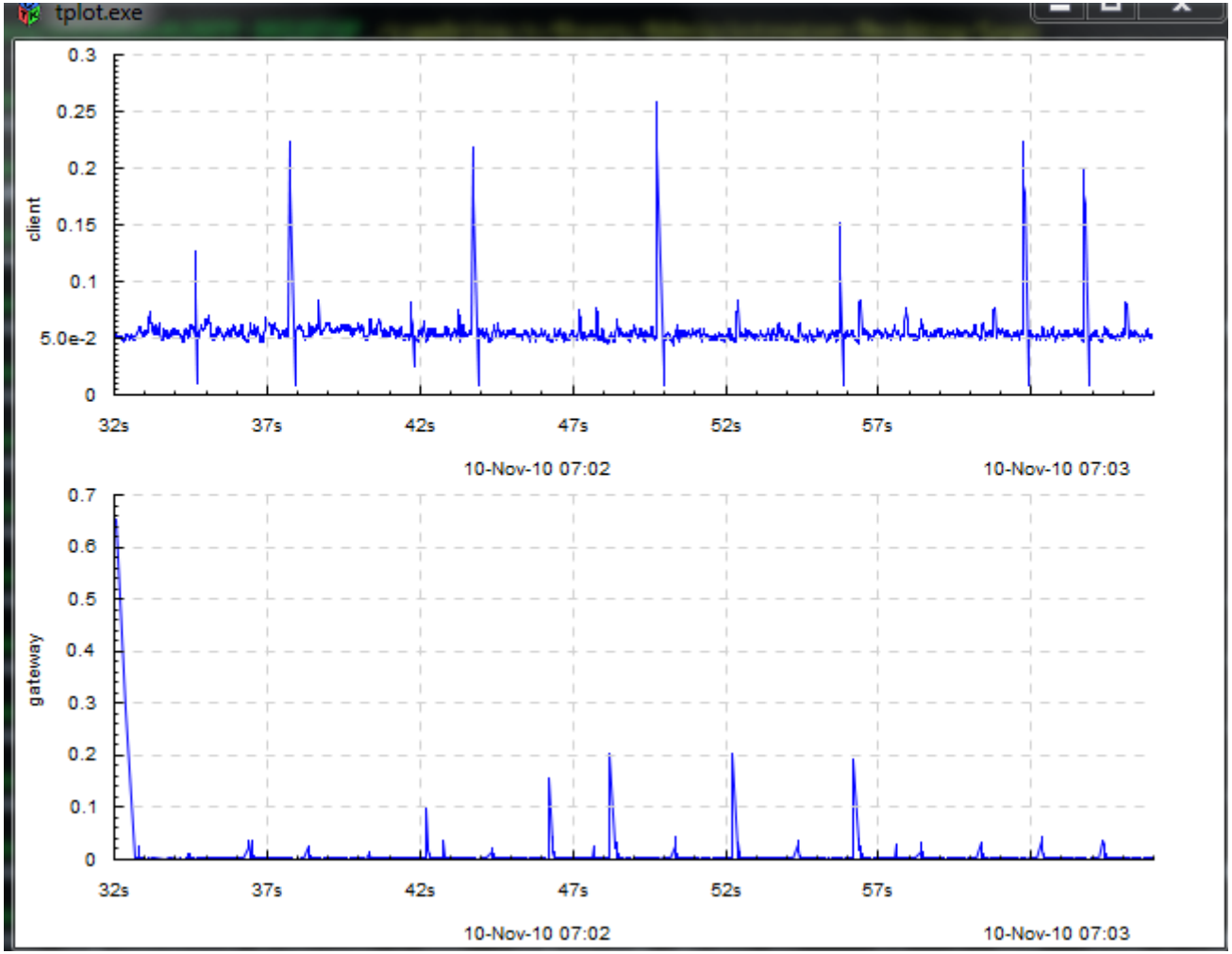
Some “splits” are slow. Their impact is not negligible at all



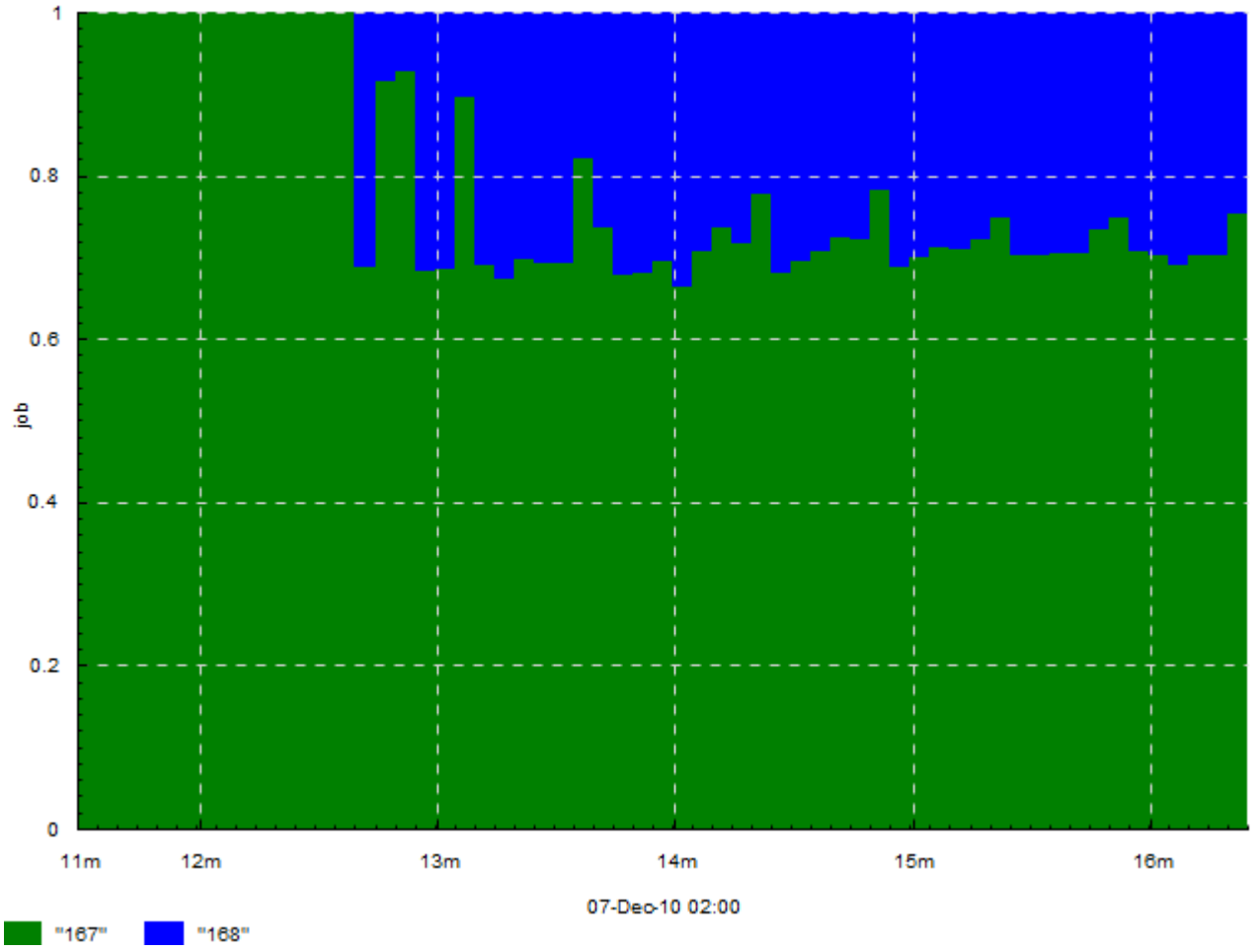
We’re basically keeping up with the flow of tasks, lagging ~1s behind.



Client calls Gateway. Client thinks it takes 50ms, Gateway thinks it takes ~2ms

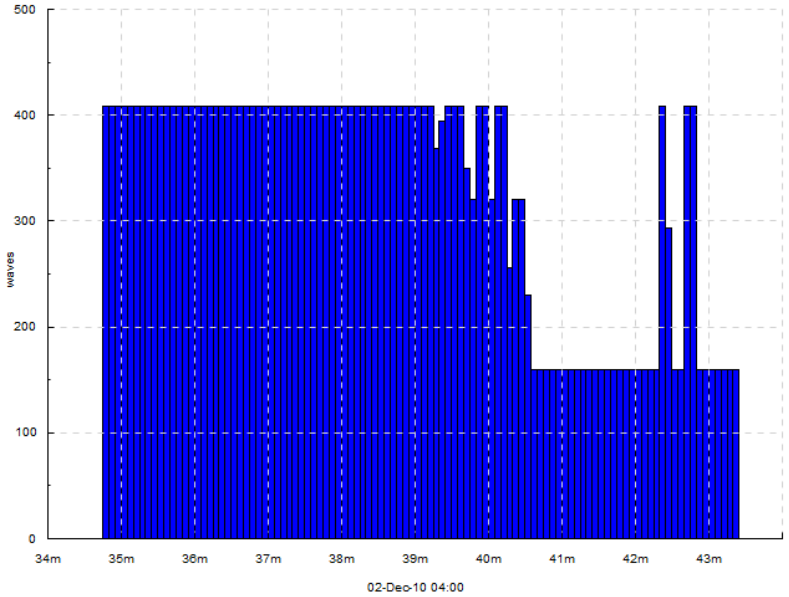


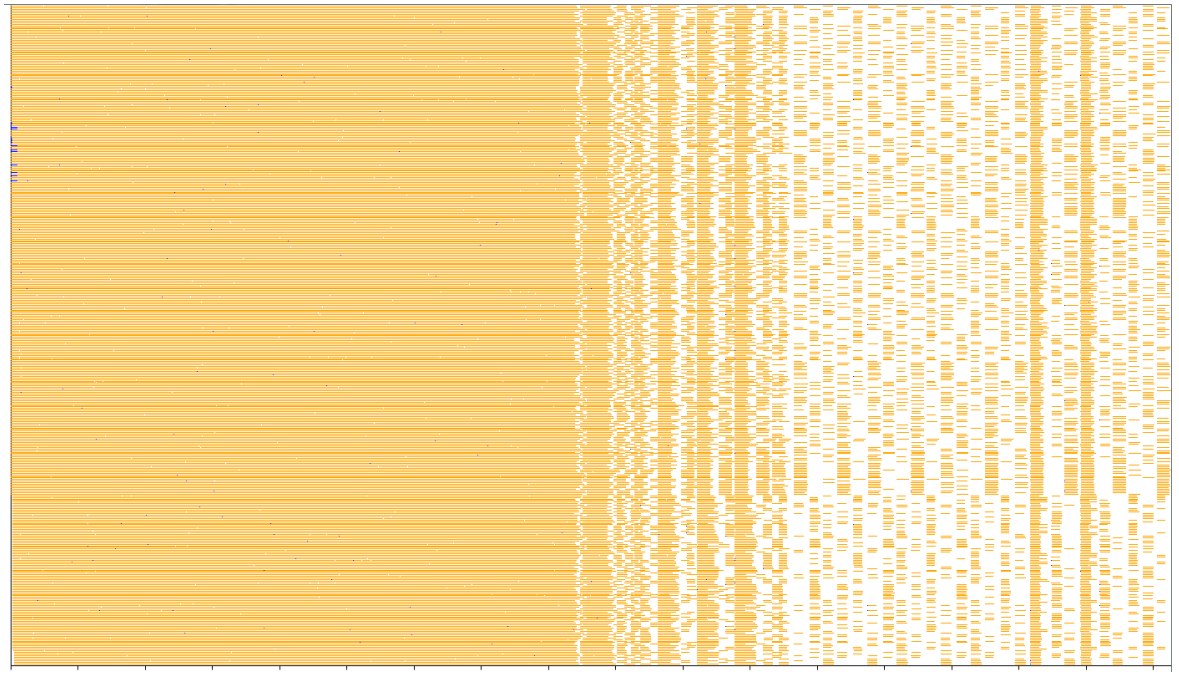
Job 168 preempts job 167 and see how cluster usage share changes.



See anything in common?

Numbes of “waves” being processed by cluster at each moment





It has slightly more options than splot

```
Usage: tplot [-o OFILE] [-of {png|pdf|ps|svg|x}] [-or 640x480] -if IFILE [-tf TF]
            [-k Pat1 Kind1 -k Pat2 Kind2 ...] [-dk KindN] [-fromTime TIME] [-toTime TIME]
  -o  OFILE - output file (required if -of is not x)
  -of       - output format (x means draw result in a window, default: extension of -o)
              x is only available if you installed timeplot with --flags=gtk
  -or       - output resolution (default 640x480)
  -if IFILE - input file; '-' means 'read from stdin'
  -tf TF    - time format: 'num' means that times are floating-point numbers
              (for instance, seconds elapsed since an event); 'date PATTERN' means that times are dates
              in the format specified by PATTERN - see http://linux.die.net/man/3/strptime,
              for example, [%Y-%m-%d %H:%M:%S] parses dates like [2009-10-20 16:52:43].
              We also support %OS for fractional seconds (i.e. %OS will parse 12.4039 or 12,4039).
              Default: 'date %Y-%m-%d %H:%M:%OS'
  -k P K    - set diagram kind for tracks matching regex P (in the format of regex-tdfa, which
              is at least POSIX-compliant and supports some GNU extensions) to K
              (-k clauses are matched till first success)
  -dk       - set default diagram kind
  -fromTime - filter records whose time is >= this time (formatted according to -tf)
  -toTime   - filter records whose time is <  this time (formatted according to -tf)

Input format: lines of the following form:
1234 >A - at time 1234, activity A has begun
1234 <A - at time 1234, activity A has ended
1234 !B - at time 1234, pulse event B has occured
1234 @B COLOR - at time 1234, the status of B became such that it is appropriate to draw it with color COLOR :)
1234 =C VAL - at time 1234, parameter C had numeric value VAL (for example, HTTP response time)
1234 =D `EVENT - at time 1234, event EVENT occured in process D (for example, HTTP response code)
It is assumed that many events of the same kind may occur at once.
Diagram kinds:
  'none' - do not plot this track
  'event' is for event diagrams: activities are drawn like --[===]--- , pulse events like --|--
  'duration XXXX' - plot any kind of diagram over the *durations* of events on a track (delimited by > ... <)
     for example 'duration quantile 300 0.25,0.5,0.75' will plot these quantiles of durations of the events.
     This is useful where your log looks like 'Started processing' ... 'Finished processing': you can plot
     processing durations without computing them yourself.
  'duration[C] XXXX' - same as 'duration', but of a track's name we only take the part before character C.
     For example, if you have processes named 'MACHINE-PID' (i.e. UNIT027-8532) say 'begin something' /
     'end something' and you're interested in the properties of per-machine durations, use duration[-].
  'count N' is for activity counts: a 'histogram' is drawn with granularity of N time units, where
     the bin corresponding to [t..t+N) has value 'what was the maximal number of active events
     in that interval', or 'what was the number of impulses in that interval'.
  'freq N [TYPE]' is for event frequency histograms: a histogram of type TYPE (stacked or
     clustered, default clustered) is drawn for each time bin of size N, about the distribution
     of various ` events
  'hist N [TYPE]' is for event frequency histograms: a histogram of type TYPE (stacked or
     clustered, default clustered) is drawn for each time bin of size N, about the counts of
     various ` events
  'quantile N q1,q2,..' (example: quantile 100 0.25,0.5,0.75) - a bar chart of corresponding
     quantiles in time bins of size N
  'binf N v1,v2,..' (example: binf 100 1,2,5,10) - a bar chart of frequency of values falling
     into bins min..v1, v1..v2, .., v2..max in time bins of size N
  'binh N v1,v2,..' (example: binf 100 1,2,5,10) - a bar chart of counts of values falling
     into bins min..v1, v1..v2, .., v2..max in time bins of size N
  'lines'  - a simple line plot of numeric values
  'dots'   - a simple dot plot of numeric values
  'cumsum' - a simple line plot of the sum of the numeric values
  'sum N'  - a simple line plot of the sum of the numeric values in time bins of size N
N is measured in units or in seconds.
```

But it's used the same way

Log file

```
INFO 2010-12-02 07:08:10.422 [Pool-1] A task arrived
INFO 2010-12-02 07:08:10.440 [Pool-2] A task arrived
INFO 2010-12-02 07:08:10.518 [Pool-3] Task finished
```

One-liner

```
awk '{t=$2 " " $3} \
     /arrived/{print t " >running"; print t " !begin/5s"} \
     /finished/{print t " <running"; print t " !end/5s"}'
```

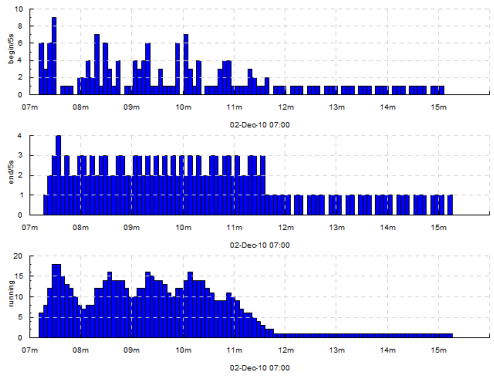
Trace file (input for tools)

```
2010-12-02 07:08:10.422 !begin/5s
2010-12-02 07:08:10.422 >running
2010-12-02 07:08:10.440 !begin/5s
2010-12-02 07:08:10.440 >running
2010-12-02 07:08:10.518 !end/5s
2010-12-02 07:08:10.518 <running
```

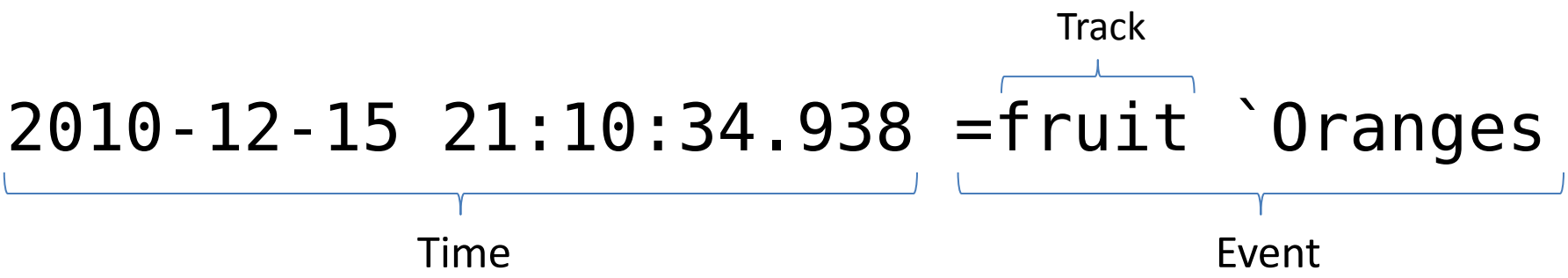
One-liner

```
tplot -dk 'count 5' -if - -of x -or 1400x800
```

Pretty picture



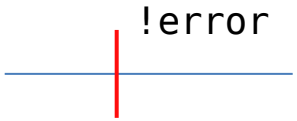
Trace format



Event types

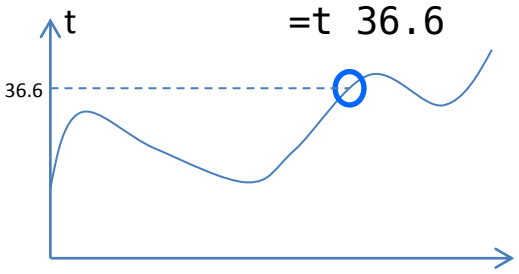
Activity start

Activity end

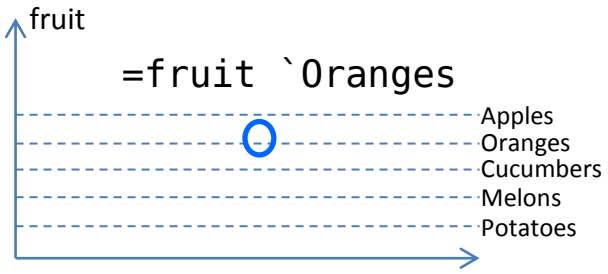


Impulse

Colored activity start



Continuous observation



Categorical (discrete) observation

How to map logs to that?

Log: `Starting “Reduce” phase…`

- `TIME >reduce`
- When “>” finished, say “<“.

Log: `TASKID – fetching data…`

- `TIME >num-fetching-data`
- `TIME @state-of-TASKID blue`
- `TIME =state-of-TASKID `fetching-data`
- `TIME =state `fetching-data`

How to map logs to that?

Log: `GET /image.php`

- `TIME =url `/image.php`
- `TIME >/image.php-MACHINE.THREADID`
  - Do not fear – see use case later
  - Say “<“ when you’ve generated the response

Log: `Accessing database DB002 – error NOTFOUND!`

- `TIME !error-DB002`
- `TIME =error-DB002 `NOTFOUND`
- `TIME =who-failed `DB002`

How to map logs to that?

Log: `Search returned 973 results`

- `TIME =search-results 973`

Log: `Request took 34ms`

- `TIME =response-time 34`

But it's used the same way

Log file

```
INFO 2010-12-02 07:08:10.422 [Pool-1] A task arrived
INFO 2010-12-02 07:08:10.440 [Pool-2] A task arrived
INFO 2010-12-02 07:08:10.518 [Pool-3] Task finished
```

One-liner

```
awk '{t=$2 " " $3} \
     /arrived/{print t " >running"; print t " !begin/5s"} \
     /finished/{print t " <running"; print t " !end/5s"}'
```

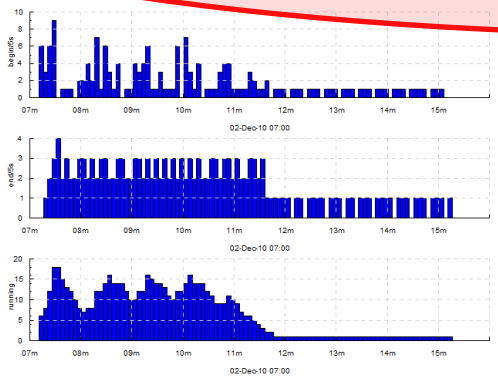
Trace file (input for tools)

```
2010-12-02 07:08:10.422 !begin/5s
2010-12-02 07:08:10.422 >running
2010-12-02 07:08:10.440 !begin/5s
2010-12-02 07:08:10.440 >running
2010-12-02 07:08:10.518 !end/5s
2010-12-02 07:08:10.518 <running
```

One-liner

```
tplot -dk 'count 5' -if - -of x -or 1400x800
```

Pretty picture



Let tools do the rest

- Choose diagram kinds
- Map the trace to diagrams
- 1 diagram per track

```
-k REGEX1 KIND1
-k REGEX2 KIND2
…
-dk DEFAULT-KIND
```

```
-k search-results ‘quantile 1 0.5,0.75,0.95’ -k return-code ‘freq 1’ -dk none
```

Choose ~~your poison~~ diagram kind

‘none’ - do not plot this track
'event' is for event diagrams: activities are drawn like --[===]--- , pulse events like --|--
'duration XXXX' - plot any kind of diagram over the *durations* of events on a track (delimited by > ... <)
for example 'duration quantile 300 0.25,0.5,0.75' will plot these quantiles of durations of the events.
This is useful where your log looks like 'Started processing' ... 'Finished processing': you can plot
processing durations without computing them yourself.
'duration[C] XXXX' - same as 'duration', but of a track's name we only take the part before character C.
For example, if you have processes named 'MACHINE-PID' (i.e. UNIT027-8532) say 'begin something' /
'end something' and you're interested in the properties of per-machine durations, use duration[-].
'count N' is for activity counts: a 'histogram' is drawn with granularity of N time units, where
the bin corresponding to [t..t+N) has value 'what was the maximal number of active events
in that interval', or 'what was the number of impulses in that interval'.
'freq N [TYPE]' is for event frequency histograms: a histogram of type TYPE (stacked or
clustered, default clustered) is drawn for each time bin of size N, about the distribution
of various ` events
'hist N [TYPE]' is for event frequency histograms: a histogram of type TYPE (stacked or
clustered, default clustered) is drawn for each time bin of size N, about the counts of
various ` events
'quantile N q1,q2,..' (example: quantile 100 0.25,0.5,0.75) - a bar chart of corresponding
quantiles in time bins of size N
'binf N v1,v2,..' (example: binf 100 1,2,5,10) - a bar chart of frequency of values falling
into bins min..v1, v1..v2, .., v2..max in time bins of size N
'binh N v1,v2,..' (example: binf 100 1,2,5,10) - a bar chart of counts of values falling
into bins min..v1, v1..v2, .., v2..max in time bins of size N
'lines' - a simple line plot of numeric values
'dots' - a simple dot plot of numeric values
'cumsum' - a simple line plot of the sum of the numeric values
'sum N' - a simple line plot of the sum of the numeric values in time bins of size N

TL;DR

‘none’

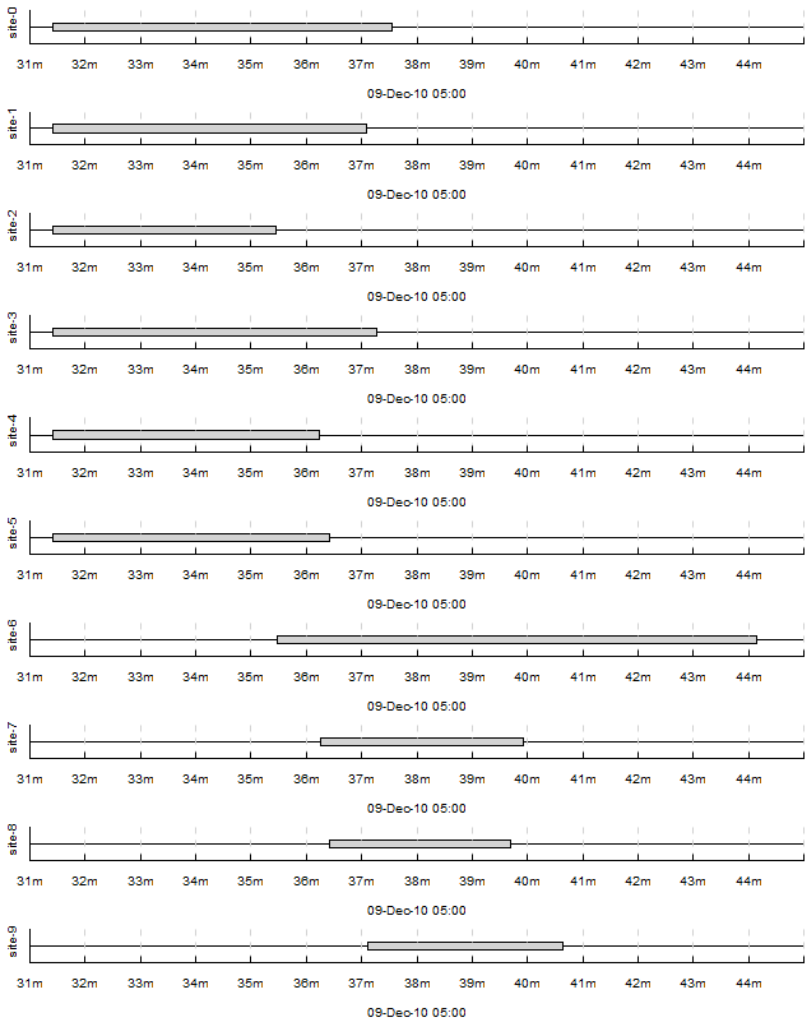
‘event’

'event' is for event diagrams: activities are drawn like --[===]--- , pulse events like --|--

Which ‘computation sites’ were active at any given time?

```
12/9/2010 5:31:25 >site-0
12/9/2010 5:31:25 >site-4
12/9/2010 5:31:25 >site-1
12/9/2010 5:31:25 >site-5
12/9/2010 5:31:25 >site-3
12/9/2010 5:31:25 >site-2
12/9/2010 5:35:27 <site-2
12/9/2010 5:35:28 >site-6
12/9/2010 5:36:14 <site-4
12/9/2010 5:36:15 >site-7
…
```

-k site event



‘event’

Other uses:

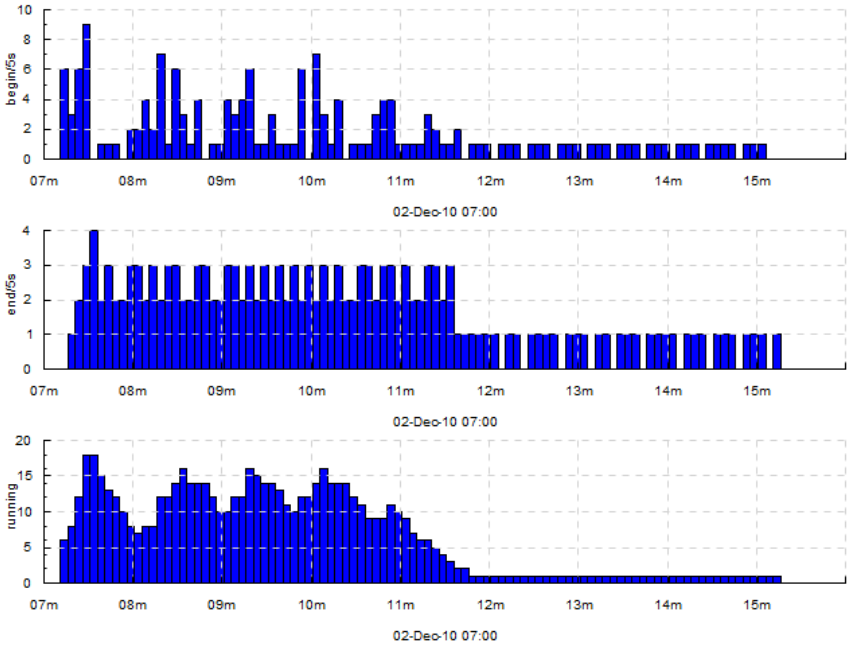
- How did a long activity influence the rest?
  - Like “data reloading” etc
- Which machines were doing anything at any given time?
  - Log: “`machine X started/finished Y`”
  - Trace: “>X” / “<X”
  - `event :` when was X > 0

‘count’

```
INFO 2010-12-02 07:08:10.422 [Pool-1] A task arrived
INFO 2010-12-02 07:08:10.440 [Pool-2] A task arrived
INFO 2010-12-02 07:08:10.518 [Pool-3] Task finished
```

```
…
2010-12-02 07:08:10.422 !begin/5s
2010-12-02 07:08:10.422 >running
2010-12-02 07:08:10.440 !begin/5s
2010-12-02 07:08:10.440 >running
2010-12-02 07:08:10.518 !end/5s
2010-12-02 07:08:10.518 <running
…
```

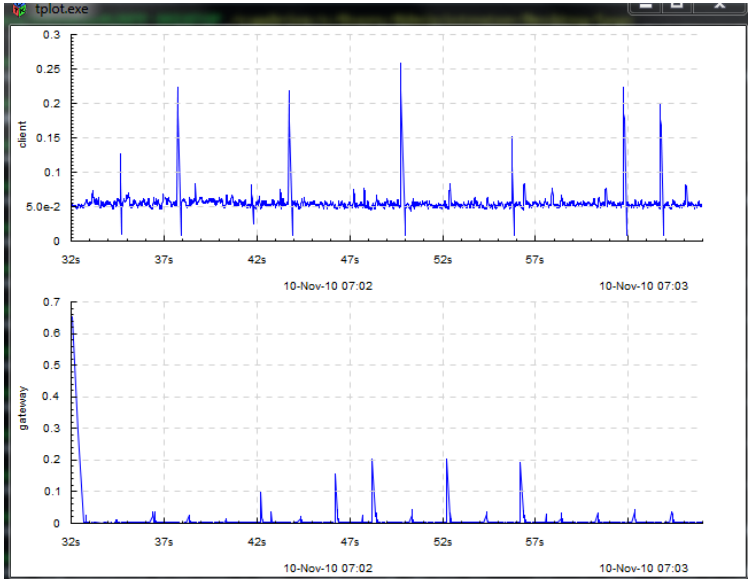
```
count 5
```



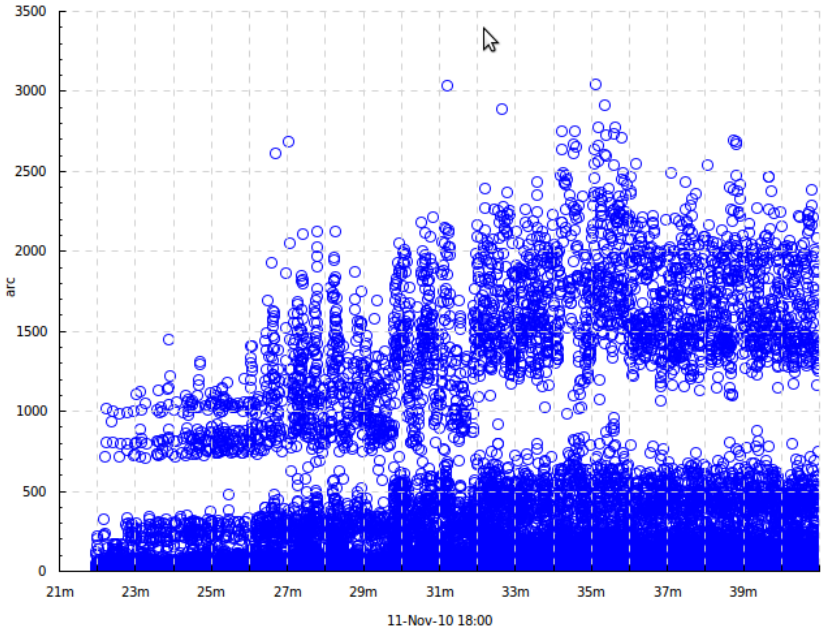
Tasks started/finished per 5s

Max active tasks per 5s

‘lines’ and ‘dots’



-dk lines



2010-11-11 18:00:27.24.343 =arc 1089.3

Nothing special

-dk dots

‘sum’ and ‘cumsum’

`sum N`

- `lines` over sum of values in bins 0..N, N..2N etc seconds

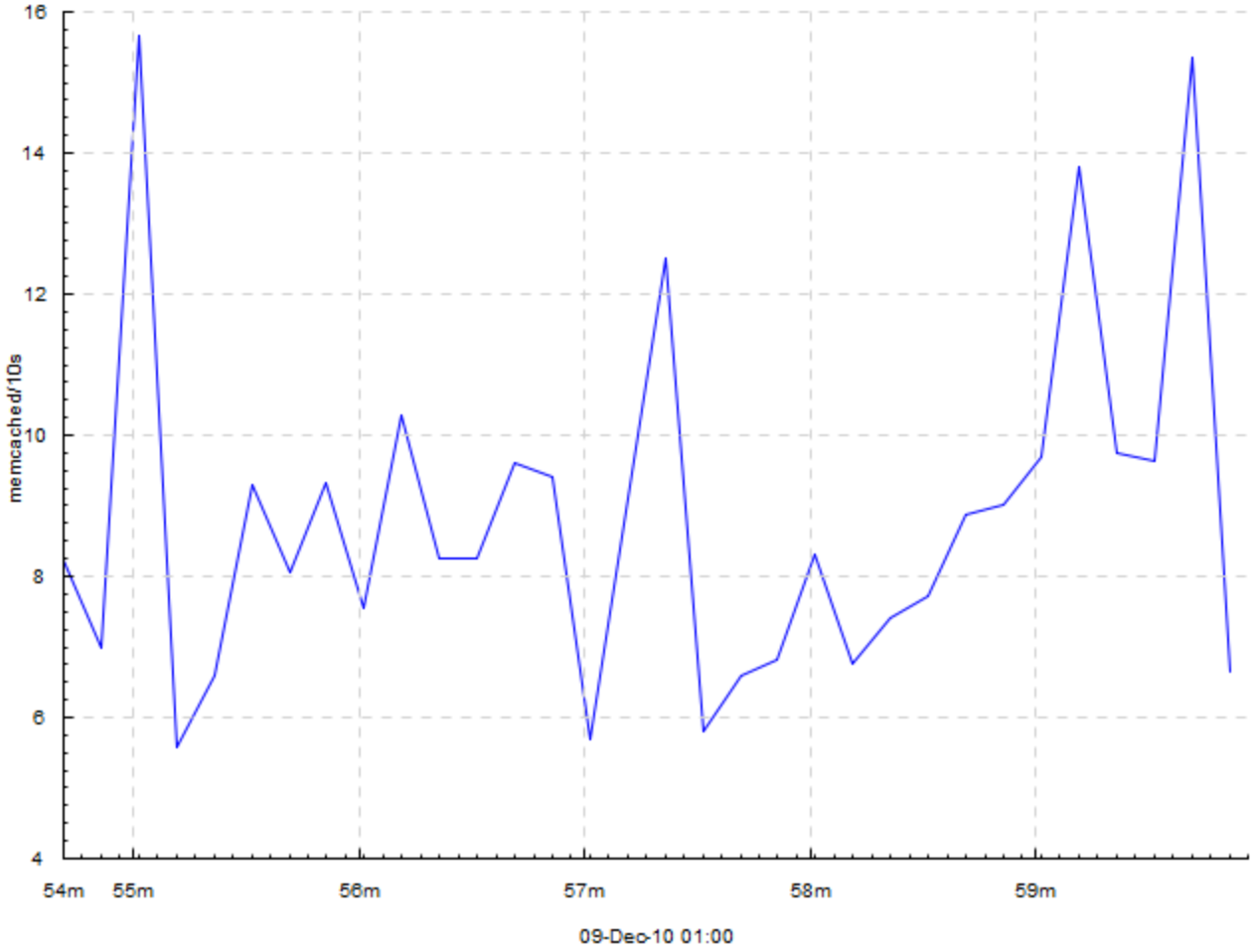
`cumsum`

- `lines` over sum of values from the beginning of the log

How much time memcached took on a 360-node cluster, in each 10-second interval

```
2010-12-09 01:00:57.738 =memcached/10s 0.059
```

It was quite unstable.




```
-k memcached ‘sum 10’
```

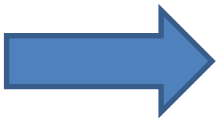
Ok, actually
`duration[-] sum 10`

Stay tuned!

Records flow:

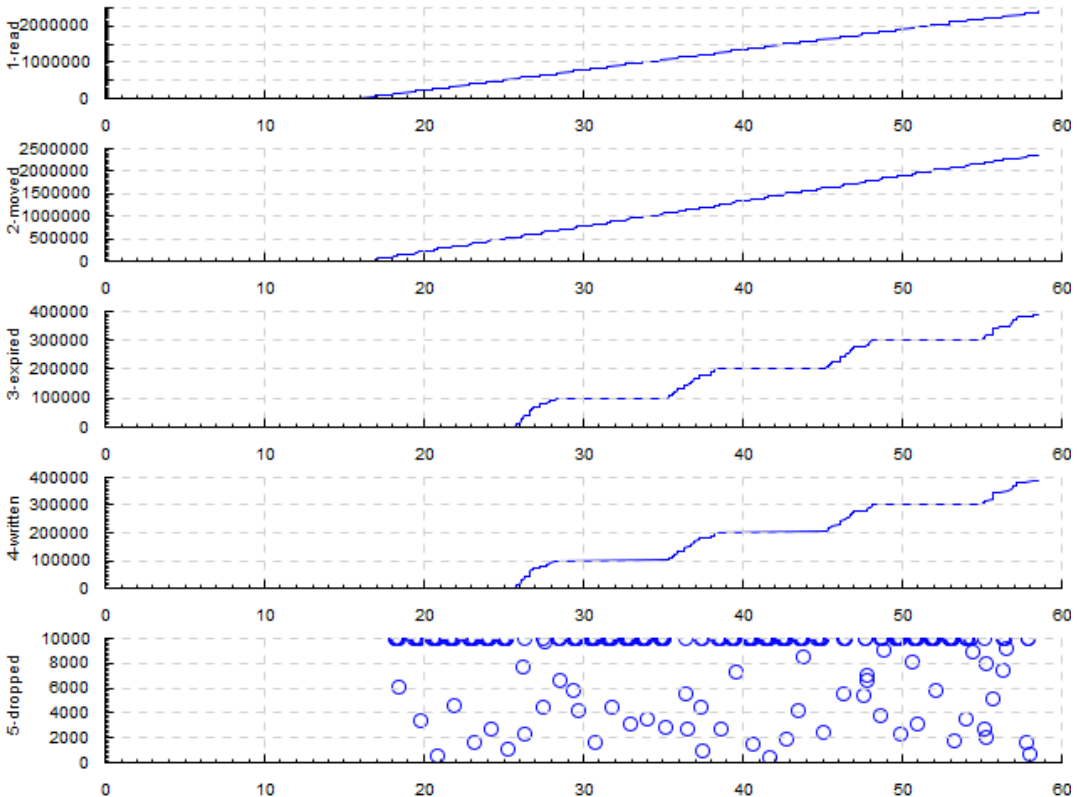
read from socket to “uncalibrated” queue ➔ **moved** to “time-buffered” queue (TQ) ➔ **expired** ➔ **written** to console

```
00013846      58.27768326    [332] Dequeued uncalibrated: 0
00013847      58.29270172    [332] TQ dequeued 0 entries
00013848      58.57195663    [332] Dequeued uncalibrated: 0
00013849      58.57243347    [332] TQ dequeued 59 entries
00013850      58.57282257    [332] Dequeued uncalibrated: 0
00013851      58.57336044    [332] Read 10000 entries from socket
00013852      58.57374191    [332] Read 10000 entries from socket
00013853      58.57406998    [332] Read 10000 entries from socket
00013854      58.57439423    [332] Read 10000 entries from socket
00013855      58.57477188    [332] Read 10000 entries from socket
00013856      58.57511139    [332] Writer dequeued 59 records
```



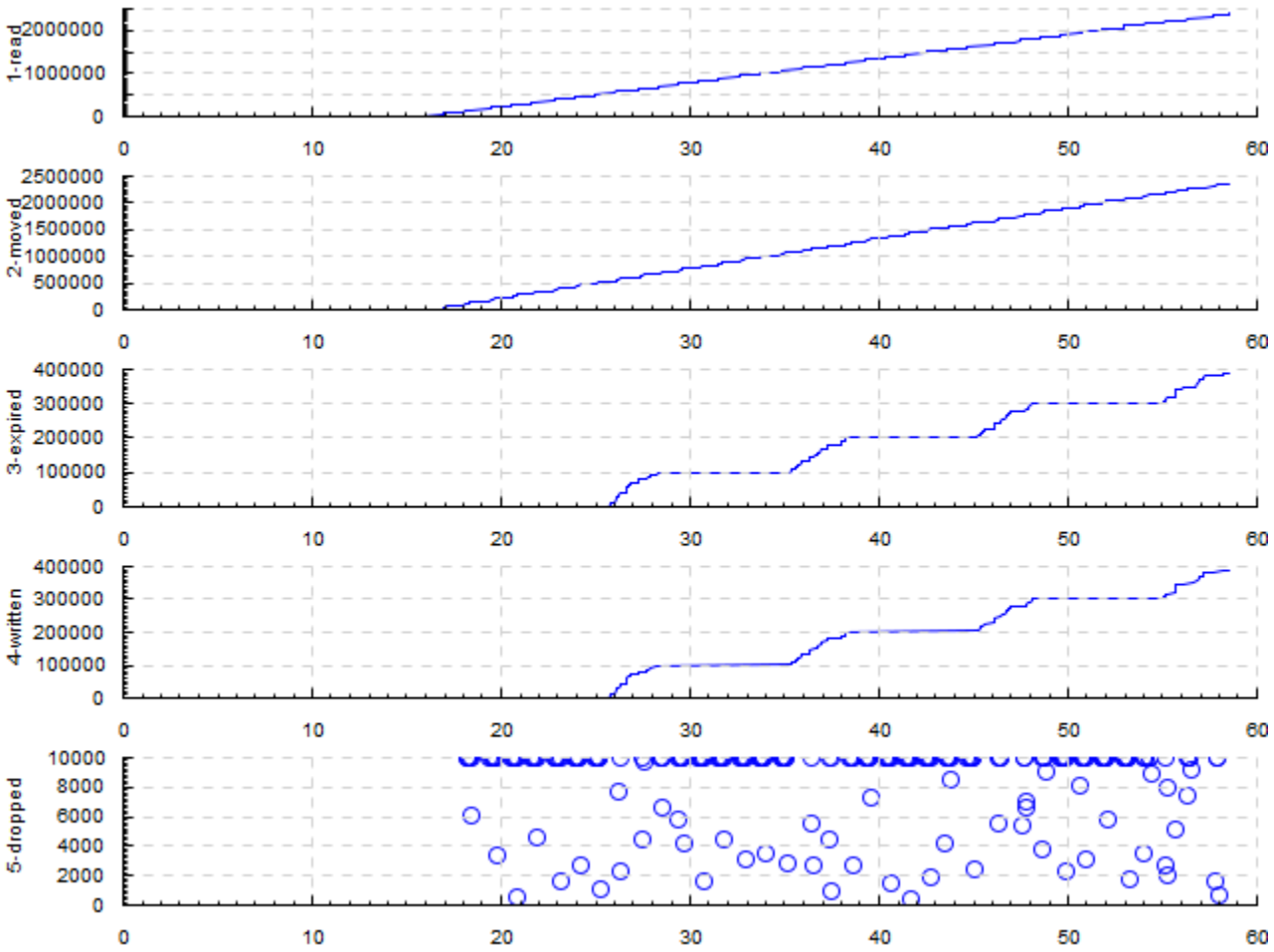
```
58.27768326 =2-moved 0
58.29270172 =3-expired 0
58.57195663 =2-moved 0
58.57243347 =3-expired 59
58.57282257 =2-moved 0
58.57336044 =1-read 10000
58.57374191 =1-read 10000
58.57406998 =1-read 10000
58.57439423 =1-read 10000
58.57477188 =1-read 10000
58.57511139 =4-written 59
```

```
-k dropped dots
-dk cumsum
```



Records flow:

read from socket to “uncalibrated” queue ➔ **moved** to “time-buffered” queue (TQ) ➔ **expired** ➔ **written** to console

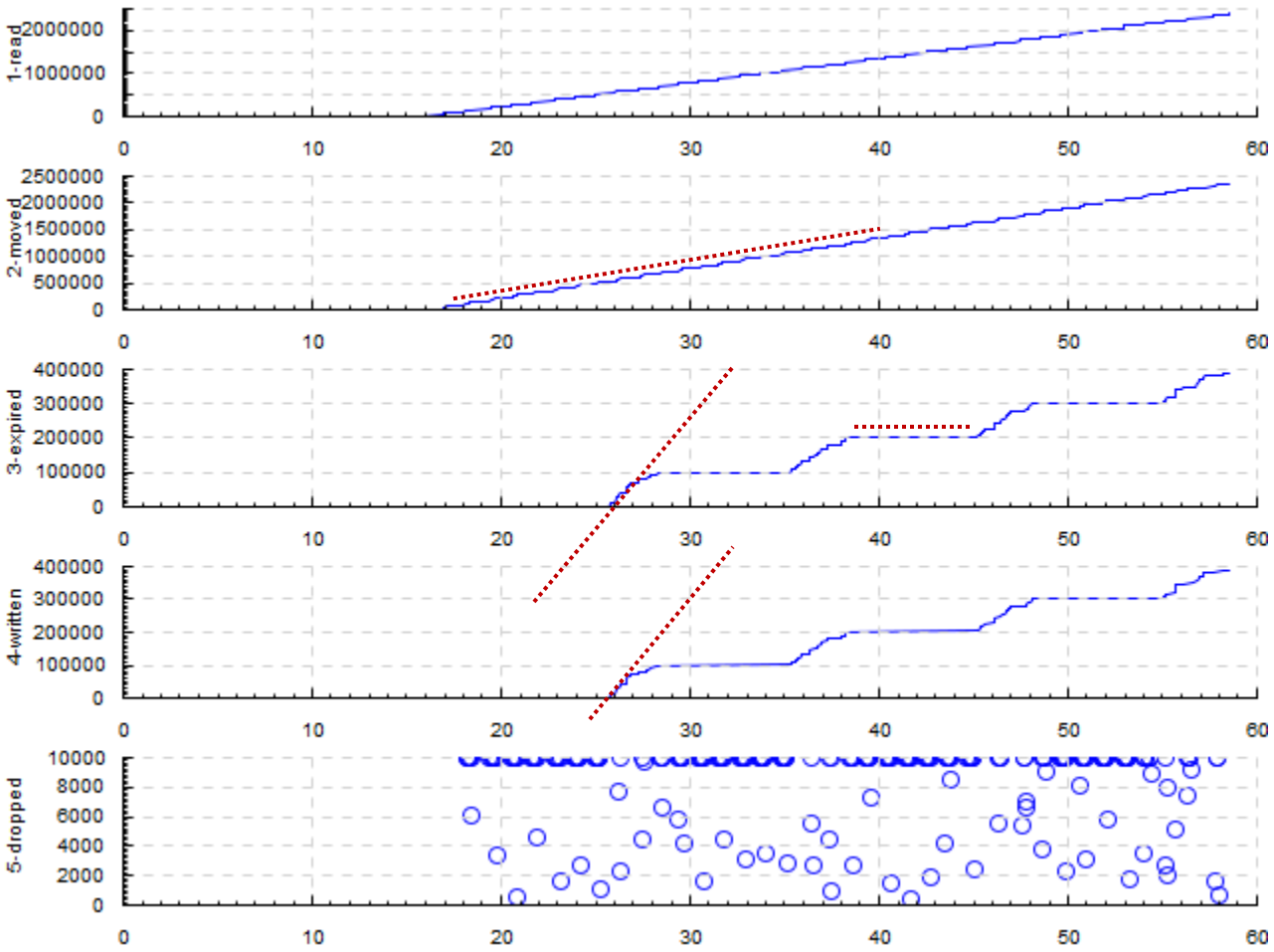


These two guys keep up.

We ‘move’ as fast as we ‘read’.

Records flow:

read from socket to “uncalibrated” queue ➔ **moved** to “time-buffered” queue (TQ) ➔ **expired** ➔ **written** to console

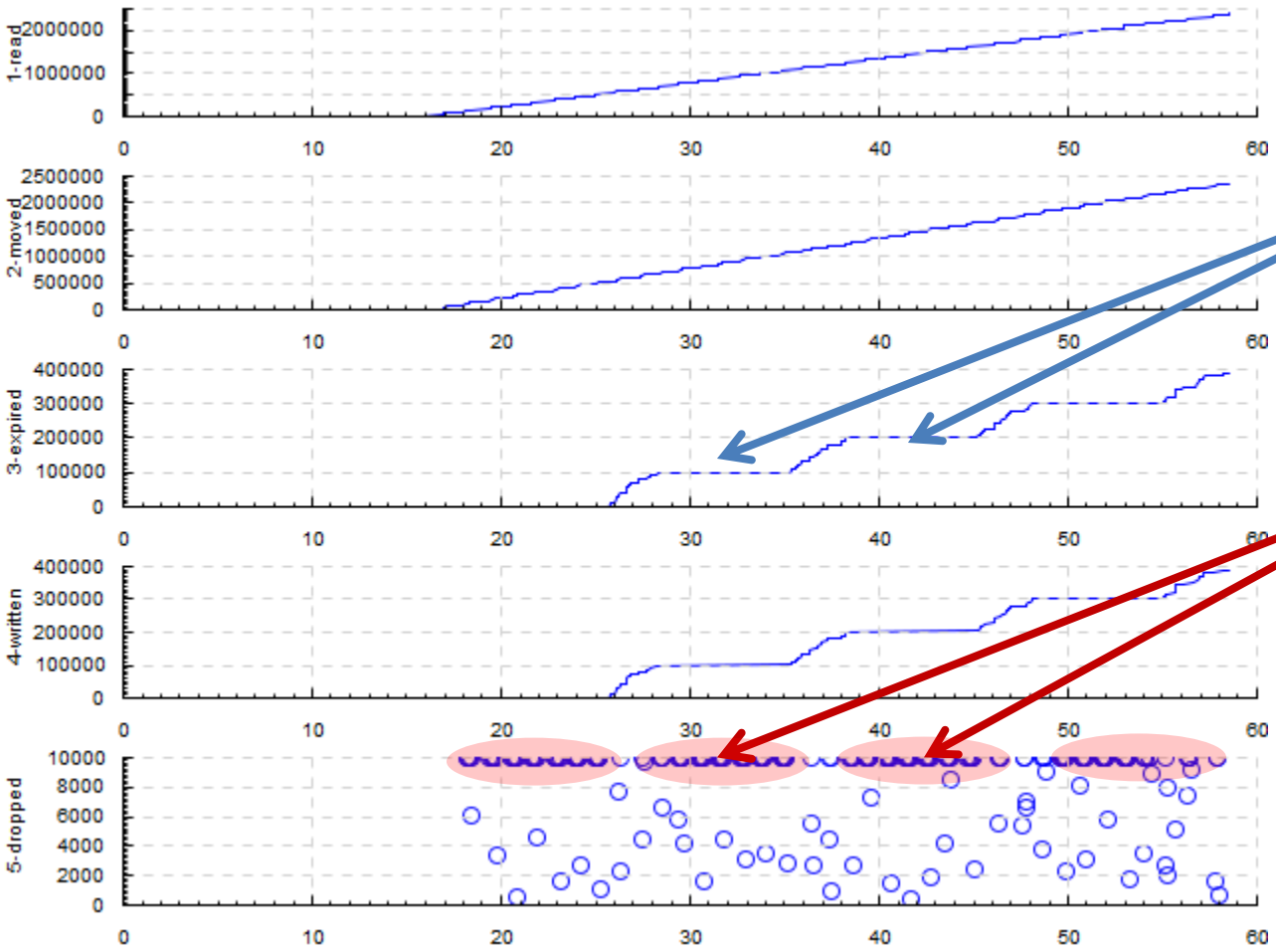


We dequeue ‘**expired**’ entries even faster
And we could **write** them equally fast

But most of the time
we don’t have any expired entries!

Records flow:

read from socket to “uncalibrated” queue ➔ **moved** to “time-buffered” queue (TQ) ➔ **expired** ➔ **written** to console

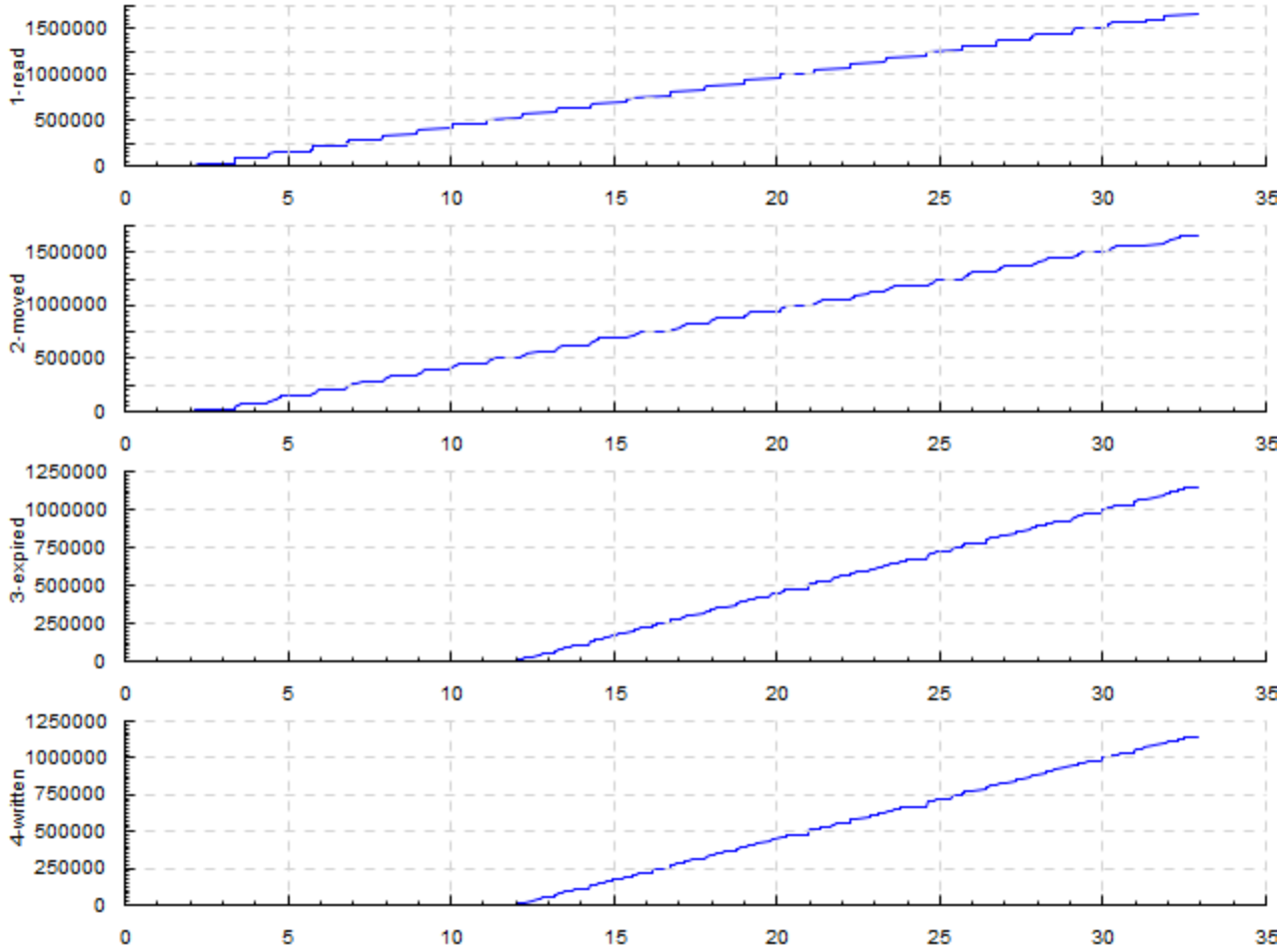


Why are no entries expired here?

Because they’re **dropped** from TQ!

We should increase TQ buffer size

increase buffer 10x…
…et voila

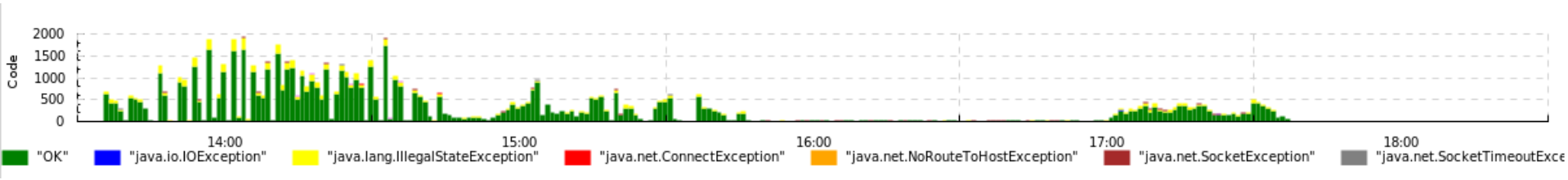


Hey, where's that 'dropped' graph? ;)

‘freq’ and ‘hist’

`-dk ‘hist 60’`

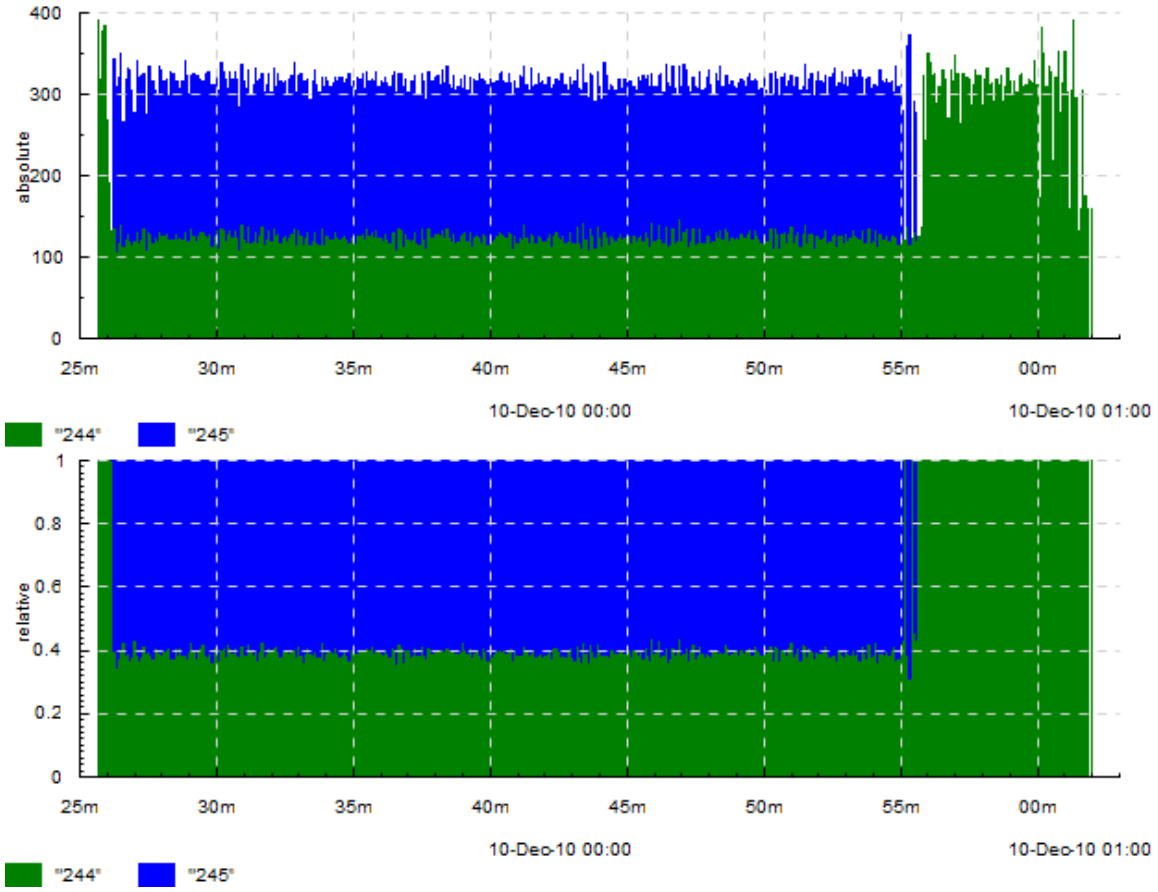
Return codes of a pinger program.




```
14:05:23 =Code `java.io.IOException
```

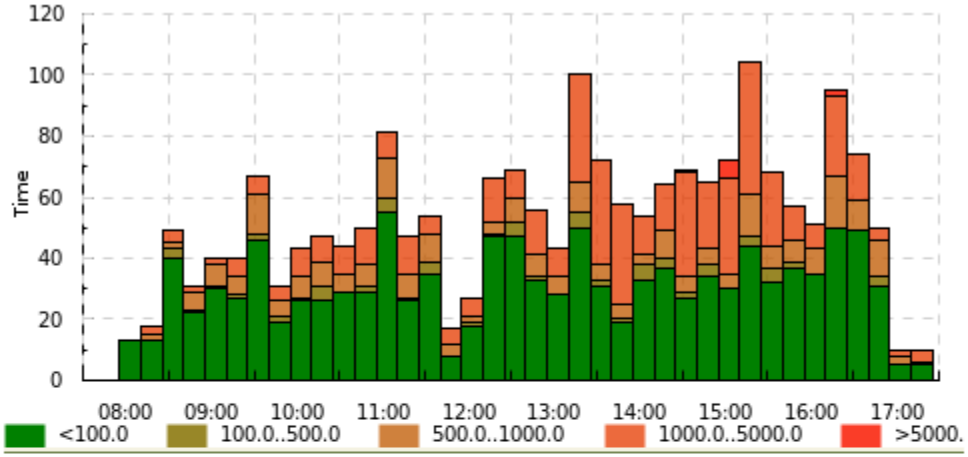
two jobs competing for a cluster

```
-k relative ‘freq 5 stacked’ -k absolute ‘hist 5 stacked’
```



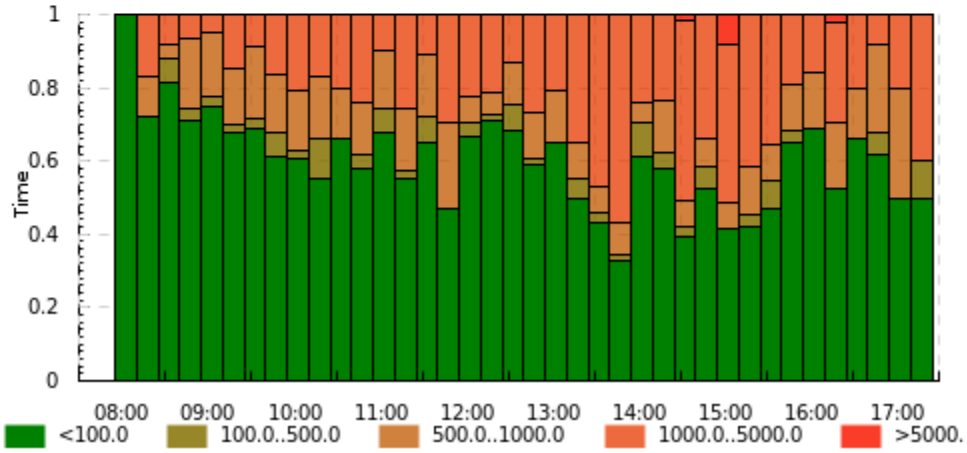

```
2010-12-10 00:00:30.422 =absolute `244
2010-12-10 00:00:30.422 =relative `244
```

‘binh’ and ‘binf’

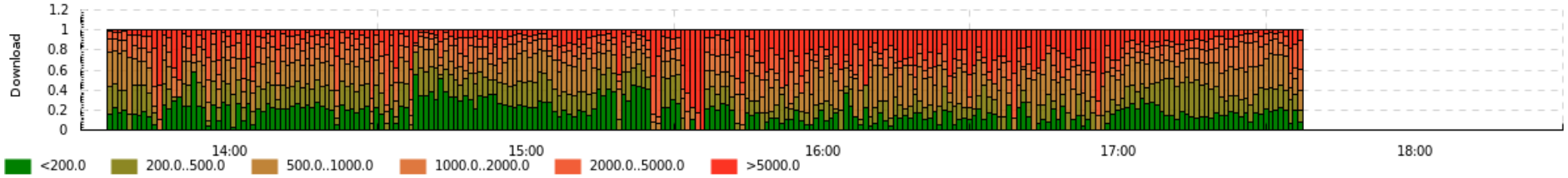


08:00:42 =Time 35.8

-dk ‘bin**h** 15 100,500,1000,5000’



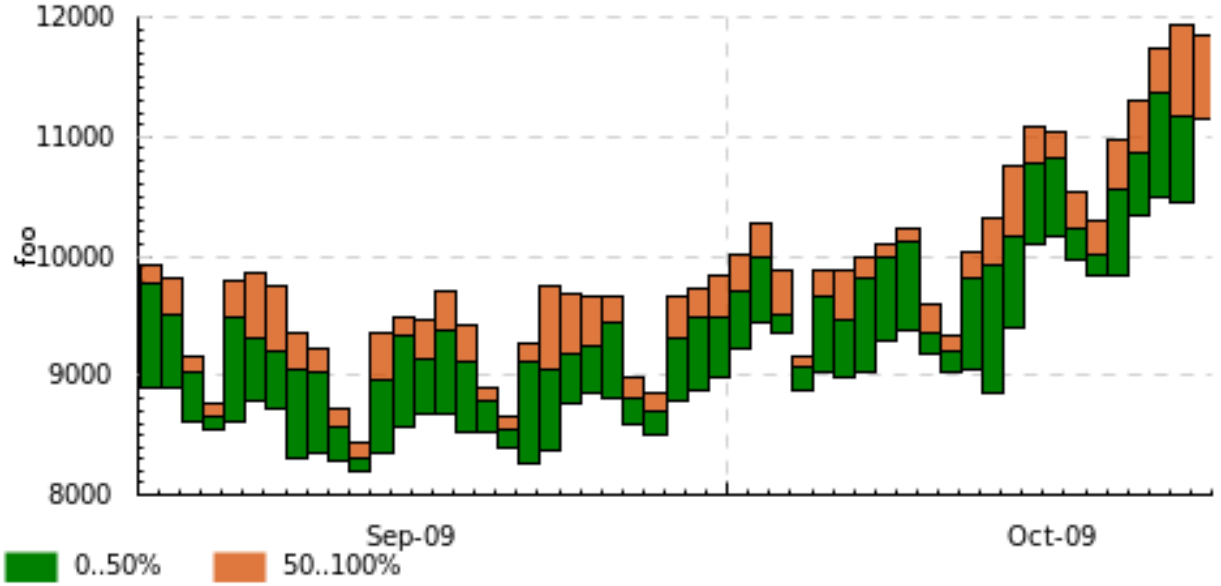
-dk ‘bin**f** 15 100,500,1000,5000’



-dk ‘binf 60 200,500,1000,2000,5000’

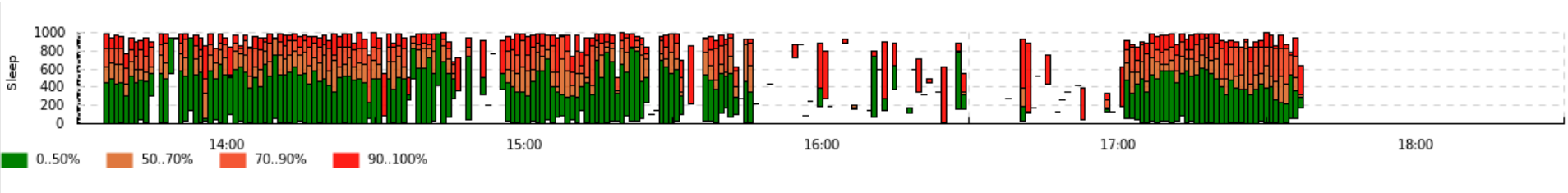
‘quantile’

```
-dk ‘quantile 3600 0.5’
```



min/max/med
of some supersecret value
from Yandex ☺

```
-dk ‘quantile 60 0.5,0.7,0.9’
```



How long a “polite” pinger program usually had to wait for a host

‘duration’

- Log: “Started quizzling”, “Finished quizzling”
- We wonder about quizzling durations

- Trace `>quizzle`, `<quizzle`
- ‘`duration XXX`’ plots XXX over durations
- Examples:
  - `duration dots`
  - `duration sum 10`
  - `duration binh 100,200,500`
  - `duration quantile 1 0.5,0.75,0.95`

- `duration[SEP]` is more universal

‘duration[SEP]’

- Log: “UNIT035 started quizzling”, “UNIT035 Finished quizzling”
- We wonder about quizzling durations

- Trace `>quizzle@UNIT035`, `<quizzle@UNIT035`
- ‘`duration[SEP] XXX`’ plots XXX over durations
- **Like ‘duration XXX’ but durations of all actors go to 1 track**
- Examples:
  - `duration[@] dots`
  - `duration[@] sum 10`
  - `duration[@] binh 100,200,500`
  - `duration[@] quantile 1 0.5,0.75,0.95`

UNIT011 is on blade 1, UNIT051 is on blade 5, memcached is on blade 1.

```
UNIT011 2010-12-09 01:54:41.927 P3964  Info Begin 390256d1-ce56-4f23-8428-1e1b109ab61c/51
UNIT011 2010-12-09 01:54:41.928 P3964 Debug GetCommonData 390256d1-ce56-4f23-8428-1e1b10
UNIT051 2010-12-09 01:54:42.045 P3832  Info Begin 390256d1-ce56-4f23-8428-1e1b109ab61c/99
UNIT051 2010-12-09 01:54:42.045 P3164  Info Begin 390256d1-ce56-4f23-8428-1e1b109ab61c/98
UNIT051 2010-12-09 01:54:42.046 P3164 Debug GetCommonData 390256d1-ce56-4f23-8428-1e1b109ab61
UNIT051 2010-12-09 01:54:42.046 P3832 Debug GetCommonData 390256d1-ce56-4f23-8428-1e1b10
UNIT011 2010-12-09 01:54:42.132 P2740  Info Begin 390256d1-ce56-4f23-8428-1e1b109ab
UNIT011 2010-12-09 01:54:42.132 P4032  Info Begin 390256d1-ce56-4f23-8428-1e1b
UNIT011 2010-12-09 01:54:42.133 P2740 Debug GetCommonData 390256d1-ce56-4f
UNIT011 2010-12-09 01:54:42.133 P4032 Debug GetCommonData 390256d1-ce
```

```
awk '{t=$2 " " $3; p="memcached-" $1 "." $4}
     /Begin /{print t " >" p}
     /GetCommonData /{print t " <" p}'
```

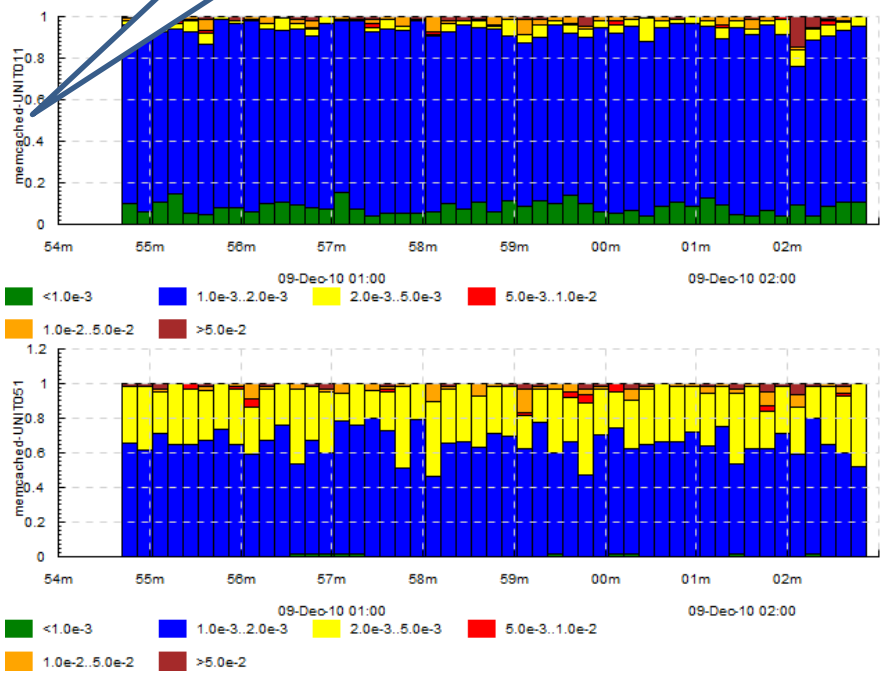
```
2010-12-09 01:54:41.853 <memcached-UNIT011.P3964
2010-12-09 01:54:41.927 >memcached-UNIT011.P3964
2010-12-09 01:54:42.001 <memcached-UNIT051.P3164
2010-12-09 01:54:42.002 <memcached-UNIT051.P3832
2010-12-09 01:54:42.045 >memcached-UNIT051.P3832
2010-12-09 01:54:42.045 >memcached-UNIT051.P3164
2010-12-09 01:54:42.128 <memcached-UNIT011.P2740
```

```
-dk 'duration[.] binf 10 0.001,0.002,0.005,0.01,0.05'
```

<memcached-UNIT011.~~P3964~~

↓

memcached-UNIT011



So, apparently, memcached access times for blade 1 are smaller.

Who'd have thought ☺

To reiterate

- Take your log
- Trivially map it to a trace (I use ‘awk’)

```
/PATTERN/{print something to trace}
```

- Choose diagram kinds and map trace to them

```
-k REGEX KIND, -dk DEFAULT-KIND
```

- Plot!

```
cat log.txt | awk ‘…’ | tplot -k …
```

Options

How to specify input?

`-if -`	read trace from stdin
`-if FILE`	read trace from FILE

How to specify output?

`-of x`	output to a window
`-o FILE.{png,svg,pdf,ps}`	output to a file

How to produce a bigger image?

`-or WIDTHxHEIGHT`	Output resolution (default 640x480)

How to specify time format?

`-tf num`	time is a real number
`-tf 'date %Y-%m-%d %H:%M:%OS'`	time is a date in format of strptime

What if I need just a part of the log?

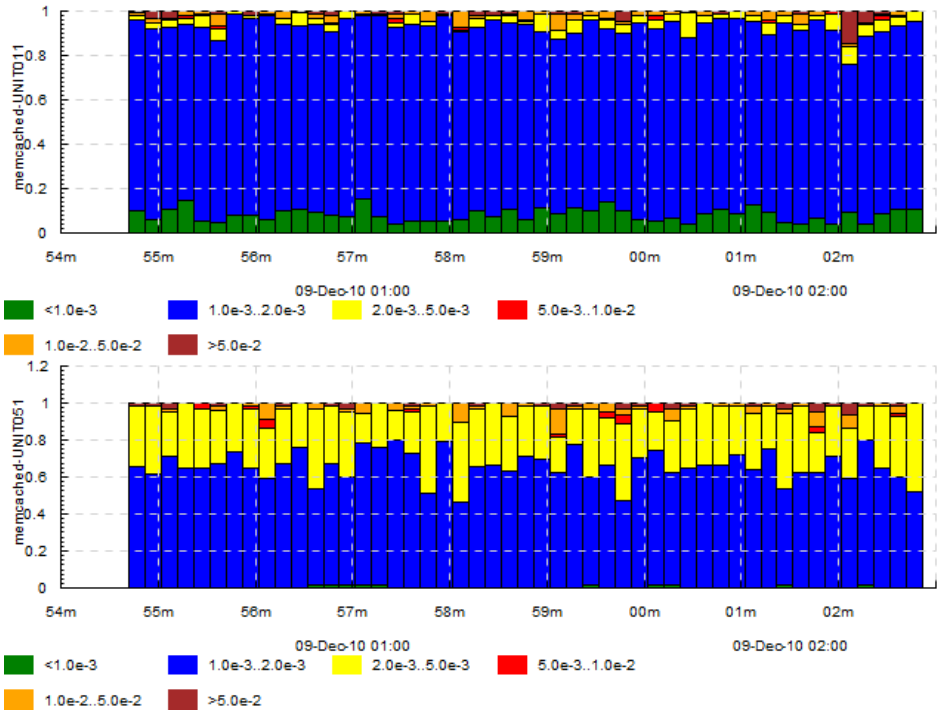
```
-fromTime ‘2010-09-12 14:33:00’
-toTime   ‘2010-09-12 16:00:00’
```

Specify one or both, in format of `-tf`.

(much better than doing this in the shell pipeline)

Example

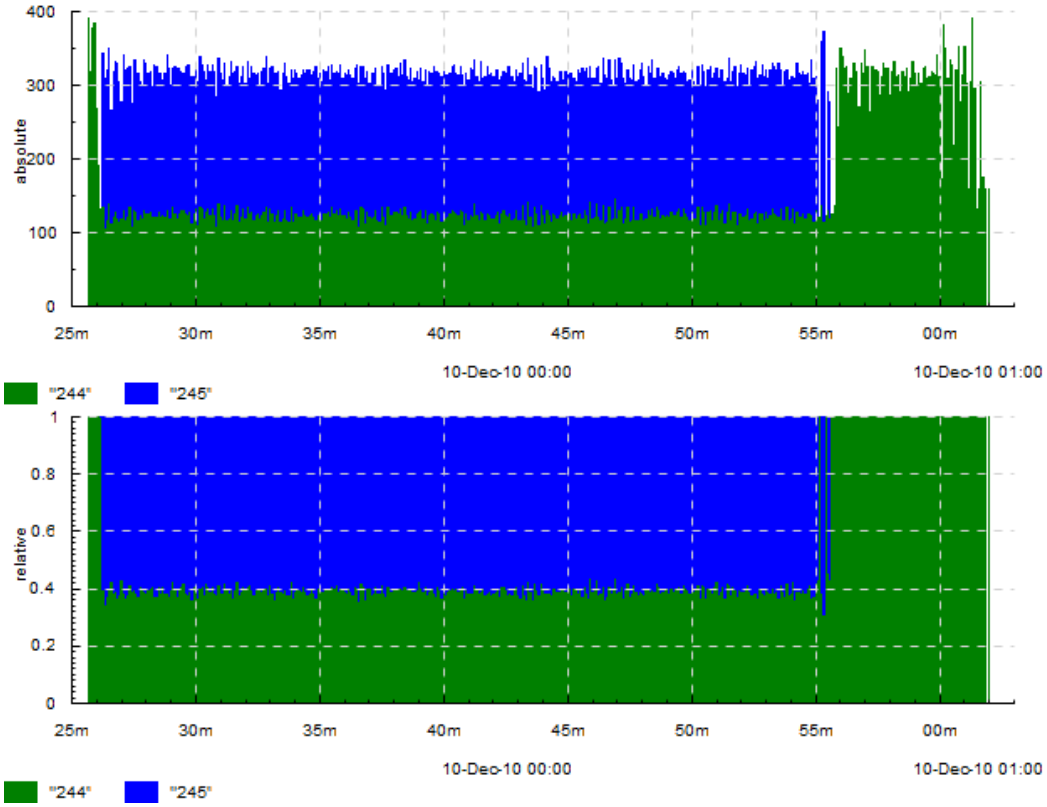
```
 cat log.txt
| grep 'UNIT0[15]1'
| awk '/Begin /        {print $3 " " $4 " >memcached-" $1 "." $9}
       /GetCommonData /{print $3 " " $4 " <memcached-" $1 "." $9}'
| tplot -if - -dk 'duration[.] binf 10 0.001,0.002,0.005,0.01,0.05' -of x
```



Example

Deliver 244.390256d1-ce56-4f23-8428-1e1b109ab61c/51

```
awk '{time=$3 " " $4; core=$1 "-" $9}                                    \
     /Deliver/{id=$NF; sub(/\..*/,"",id);                                \
               print time " =relative `" id;                             \
               print time " =absolute `" id }'                           \
    "$f"                                                                 \
   | tplot -if - -k relative 'freq 5 stacked' -k absolute 'hist 5 stacked'  \
     -o "$f.share.png"
```



How much data can they handle?

200-300Mb is ok.

If you have more, split it.

Into 10-minute bins:

```
awk '{hour=substr($4,0,4); sub(/:/,"-",hour); \
     print >(FILENAME "-" hour "0")}' $log
```

P.S. Installation on Linux

- Install http://hackage.haskell.org/platform/
- `cabal update`
- `cabal install gtk`
- `cabal install timeplot`
- `cabal install splot`

P.S. Installation on Windows

- Install http://hackage.haskell.org/platform/
- Follow http://jystic.com/2010/10/20/installing-gtk2hs-on-windows/
- `cabal update`
- `cabal install timeplot`
- `cabal install splot`
- Install http://www.cygwin.com/ for awk or whatever
- Use cygwin shell

That's it

Thanks

If you liked the presentation,
The best way to say your "thanks" is
to **use the tools** and to **spread the word**

OK, the *really* best way is to contribute
http://code.google.com/p/timeplot
http://code.google.com/p/stateplot