LTTng & Trace Compass and Open Tracing API

Naser Ezzati, Polytechnique

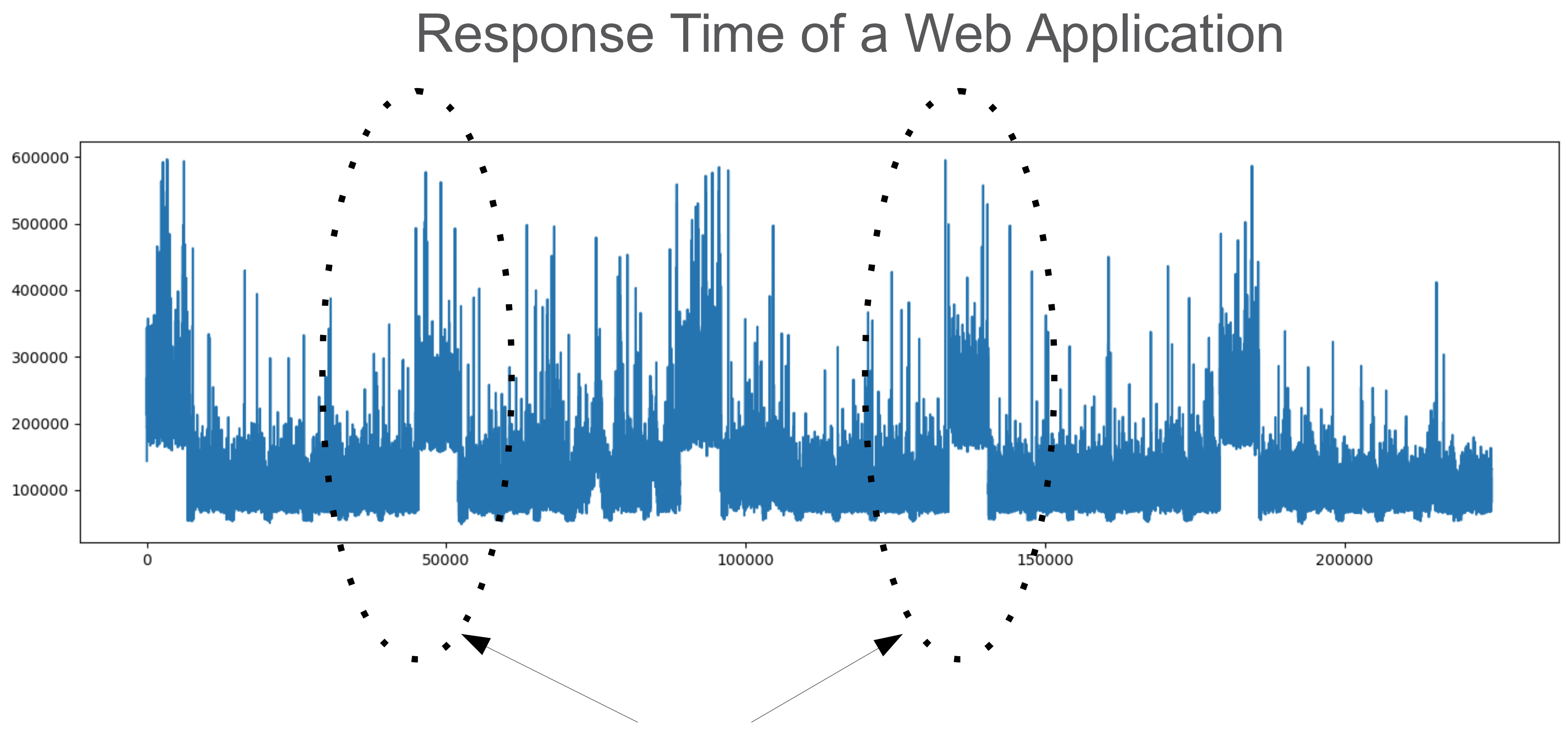
May 2017

AGENDA

Recent Updates

- Last time: LAMP Stack Instrumentation
- APACHE,, PHP, MySQL
- PHP (standard extension)
- NGINX
- PostgreSQL,
- MongoDB (coming)
-
- Linked Analysis
- Usecase!

Example:



What happens?

There are slow-downs in the speed every few minutes. What are the reasons behind?!

PHP Request Anatomy

- PHP Is a scripting language
  - compiles any file you ask it to run, obtain OPCodes from compilation, run them, and trash them away immediately.

- Parse, compile, execute, forget

  Parse, compile, execute, forget

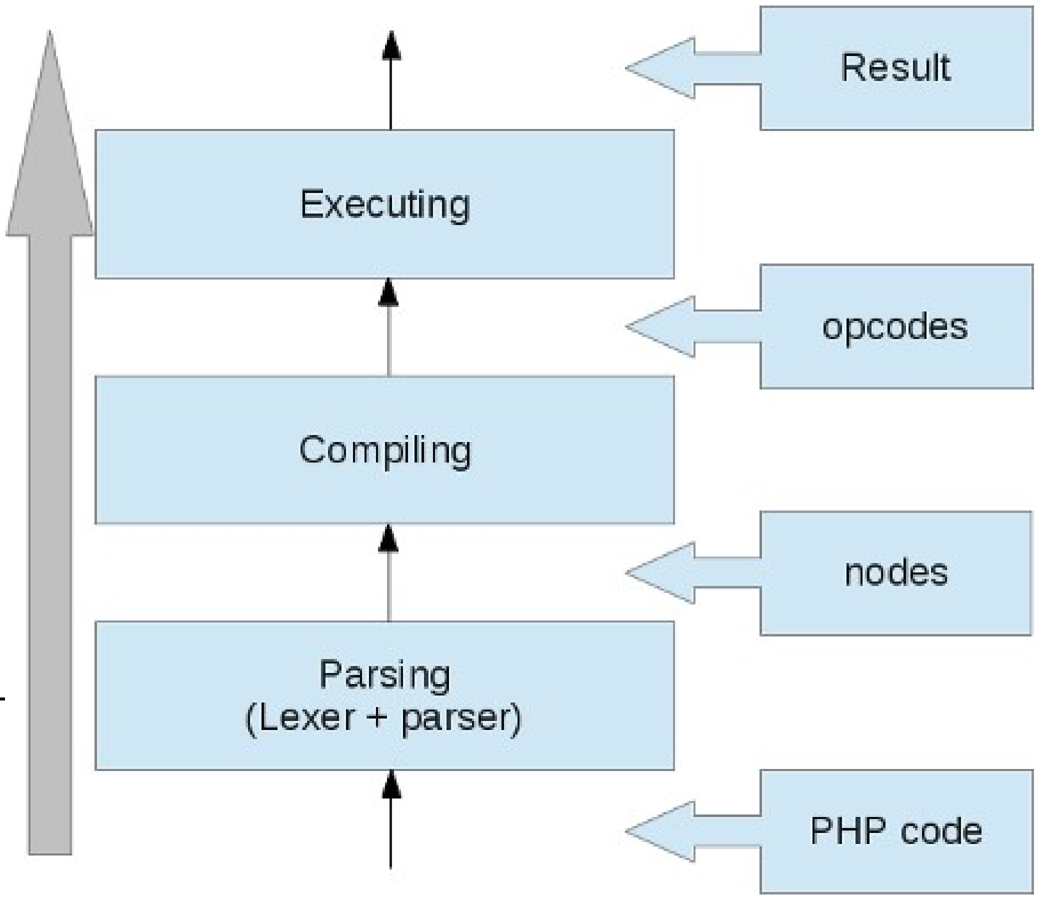
  Parse, compile, execute, forget

  ...

PHP "forgets" everything it's done in request N-1, when it comes to run request N.

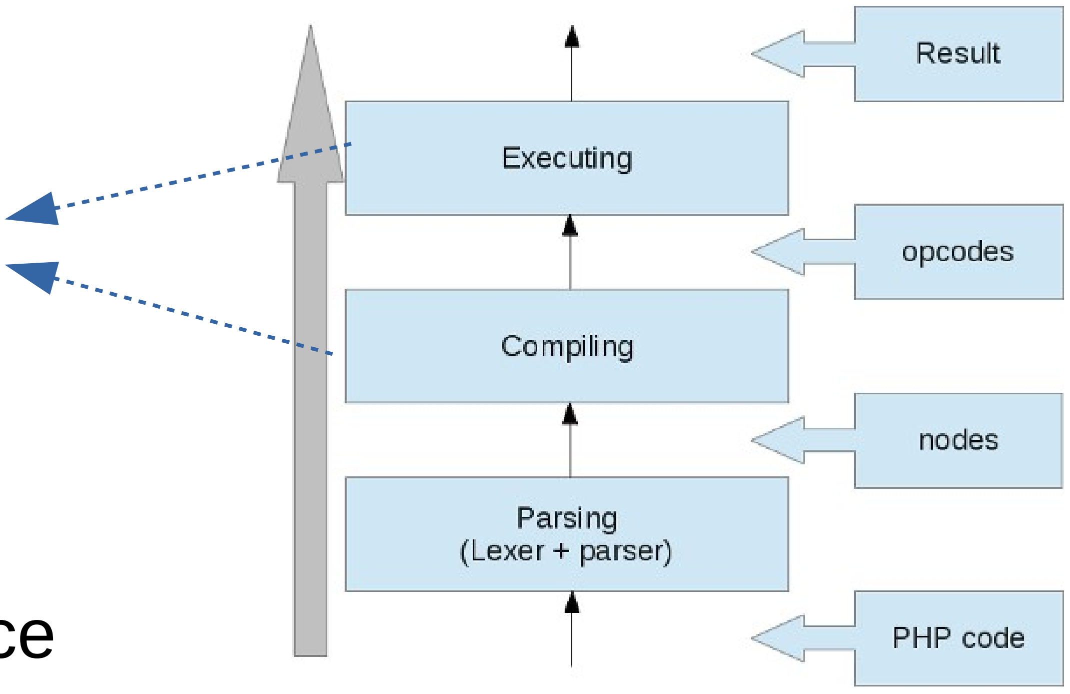
- Even if it calls the same scripts

    several times.

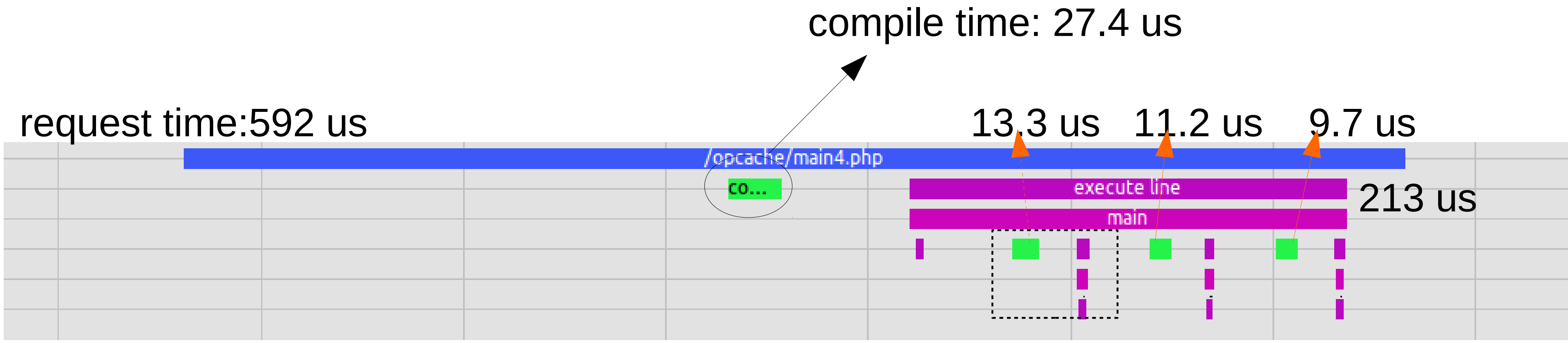


PHP Request Anatomy (2)

- Which one is the longest?
  - It depends!
  - Let's see what trace data gives us.



Compile Time: UST Events



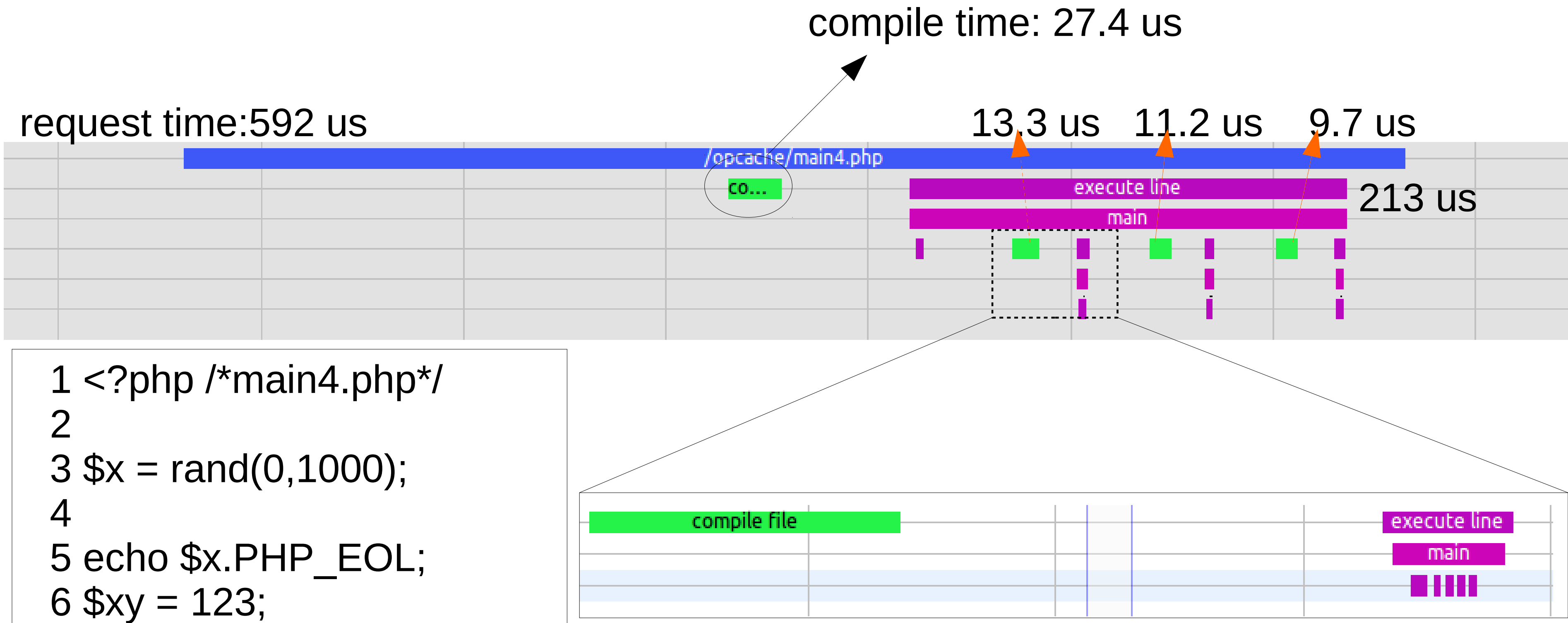
```php
<?php /*main4.php*/

$x = rand(0,1000);

echo $x.PHP_EOL;
$xy = 123;
include 'folder1/'.$xy.'.php';
include 'folder2/'.$xy.'.php';
include 'folder3/'.$xy.'.php';

echo $x.PHP_EOL;
?>
```

Compile Time: UST Events



```
<?php /*main4.php*/

$x = rand(0,1000);

echo $x.PHP_EOL;
$xy = 123;
include 'folder1/'.$xy.'.php';
include 'folder2/'.$xy.'.php';
include 'folder3/'.$xy.'.php';

echo $x.PHP_EOL;
?>
```

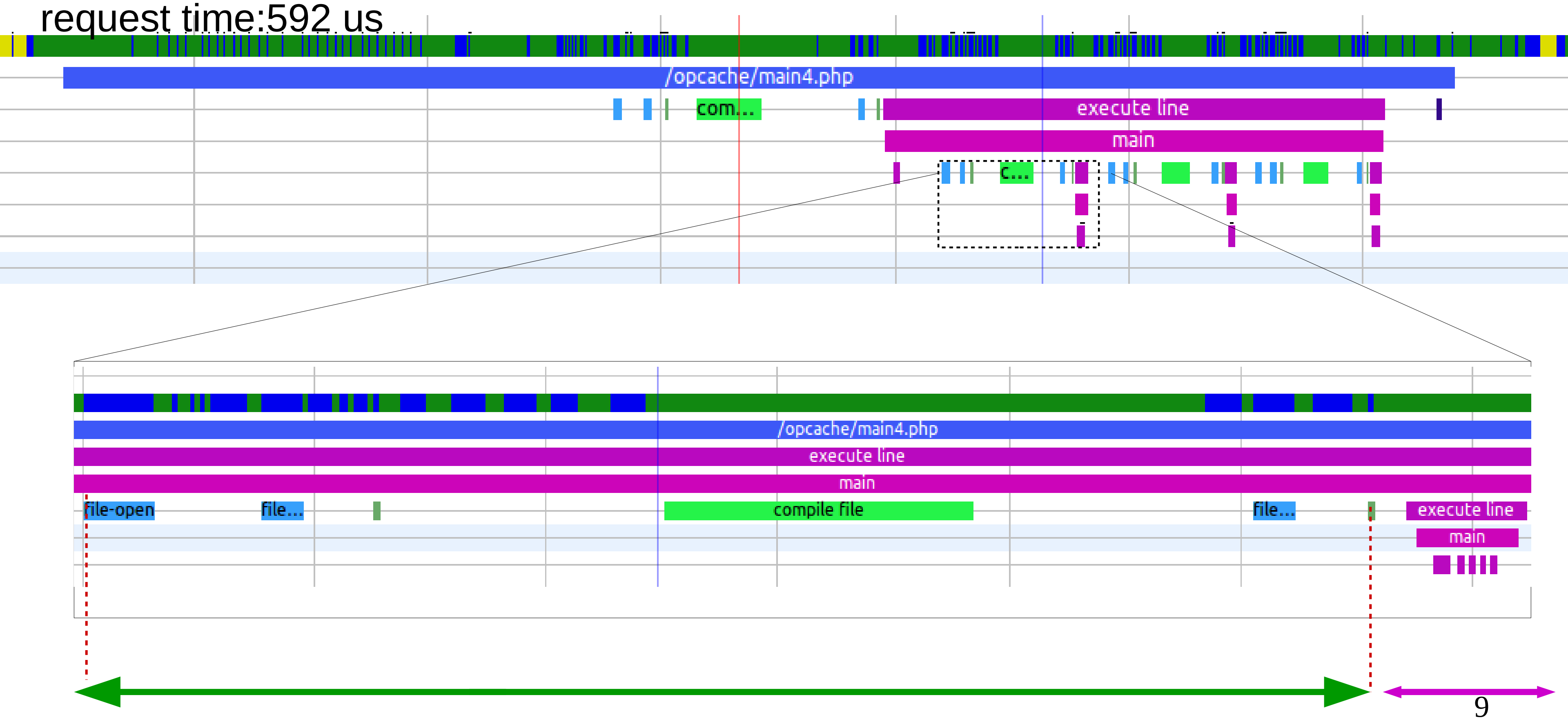
Compilation time: 60us (~ 10 % of the request time)

But, let's go deeper!

Kernel + UST Events

Now looks much more!

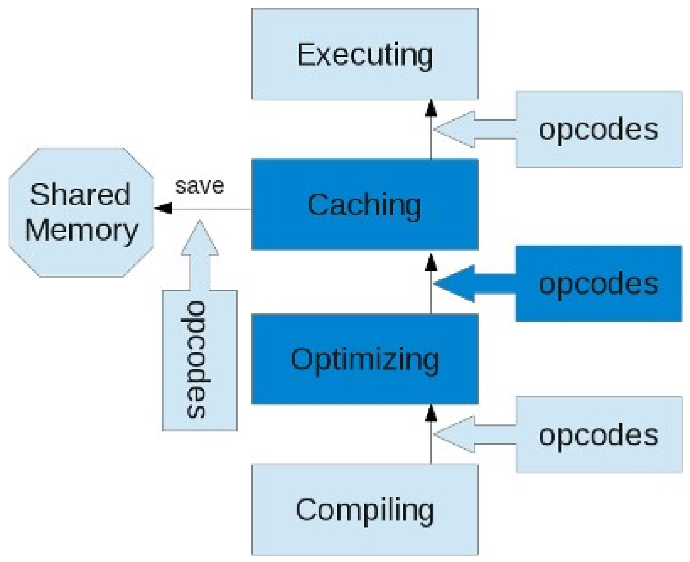
request time:592 us



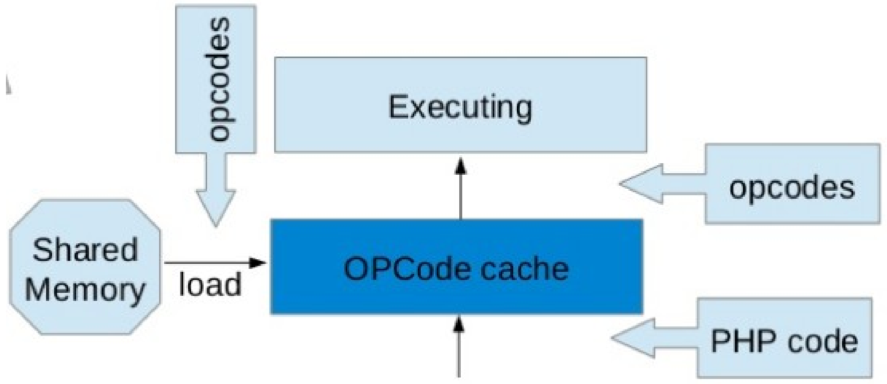
Parse, Compile, Optimize (55us)

Execute(5.2us)

Solution: Opcode Cache (Opcache)

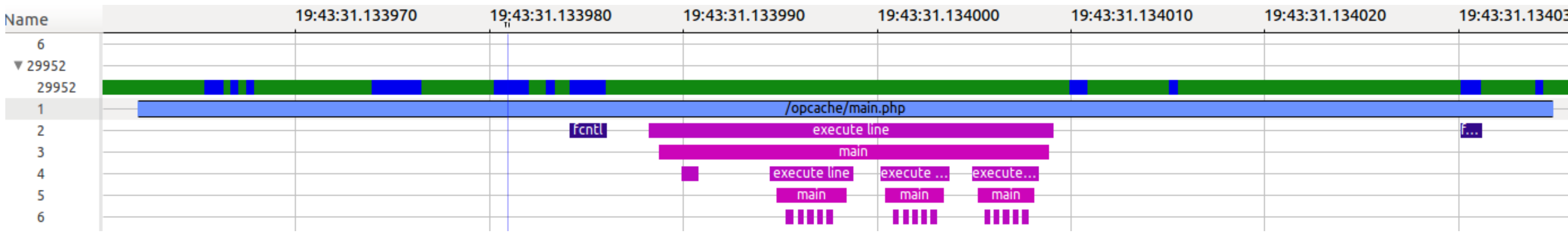


Cache at first run



Load from cache after

Solution: Opcode Cache (Opcache)

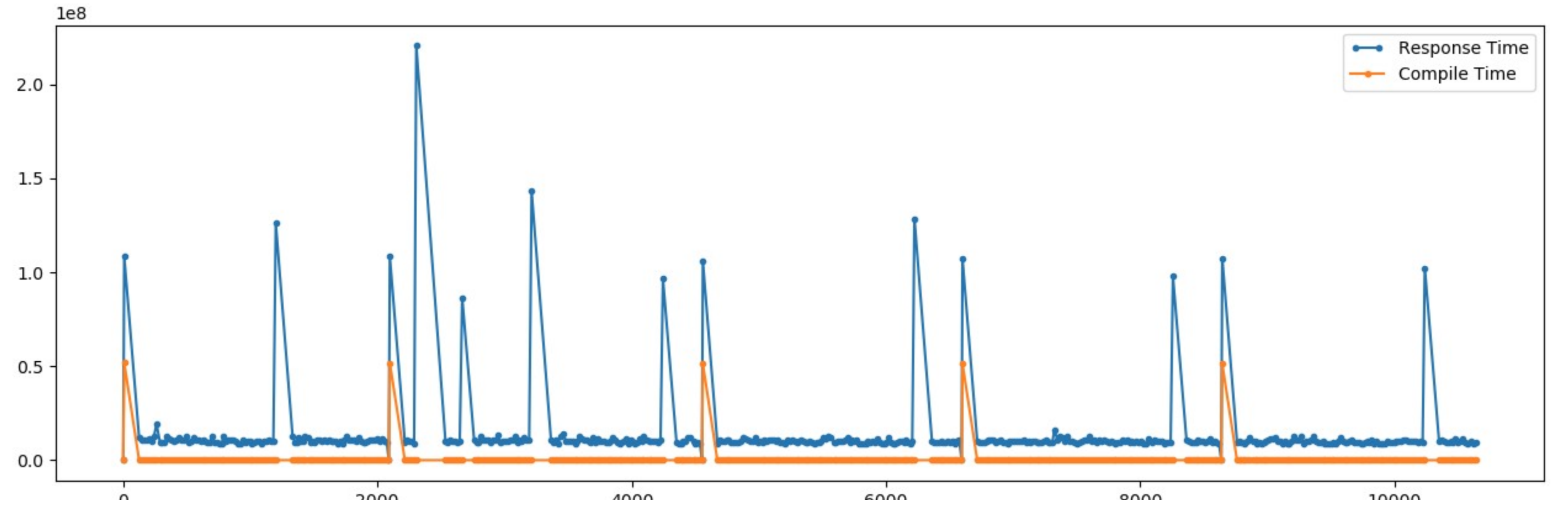


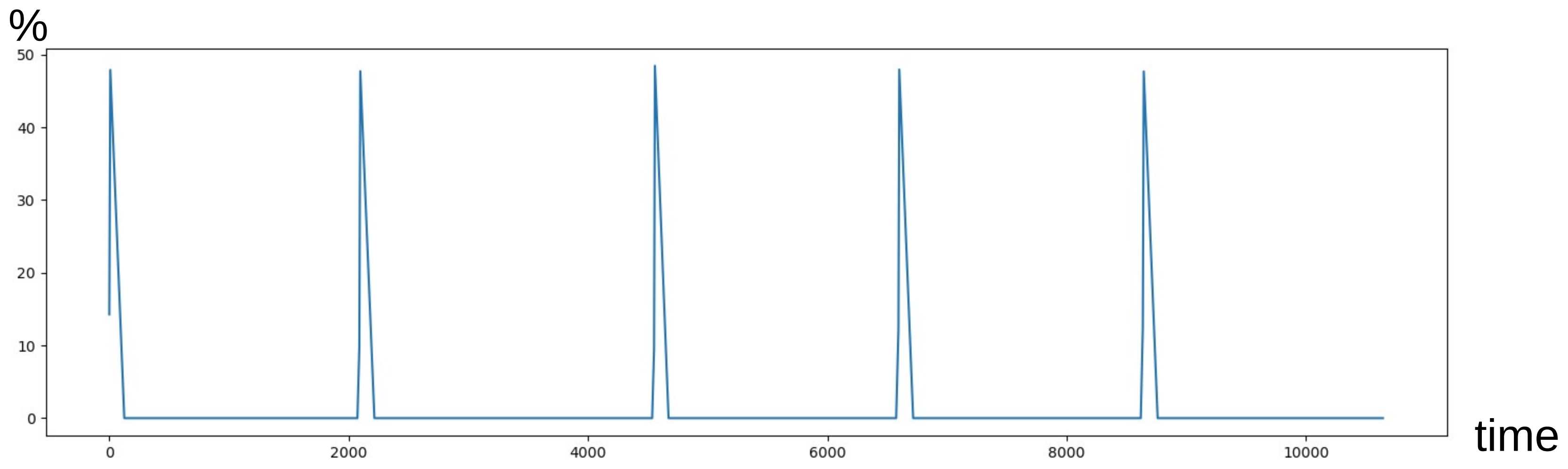
Loads from cache

Request time: 73 us

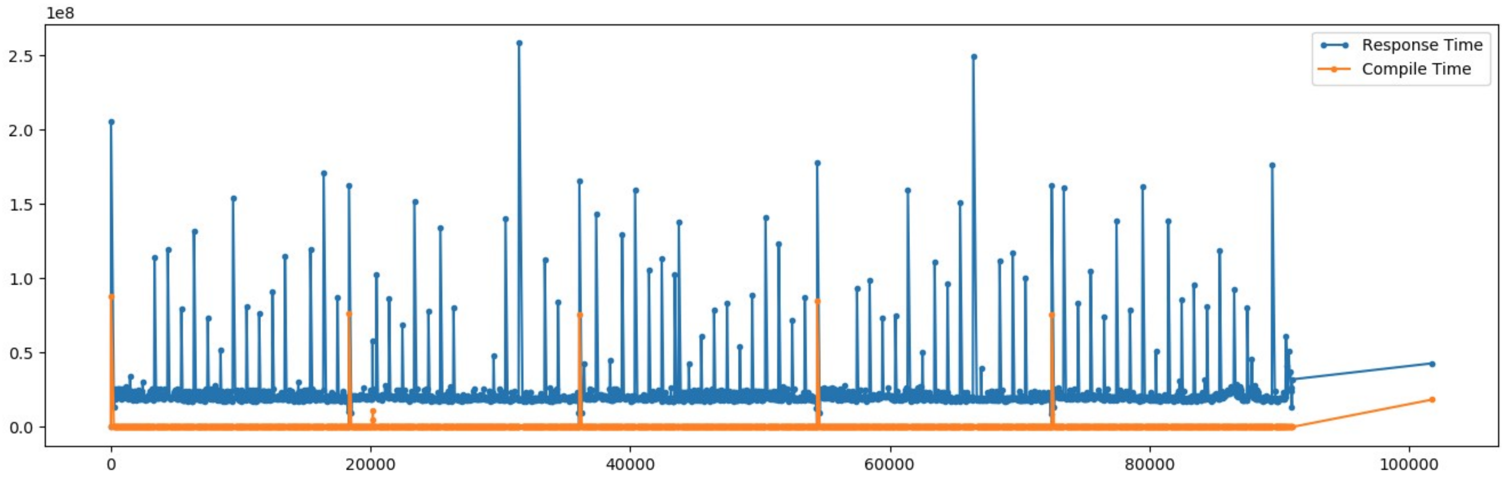
was: 592 us

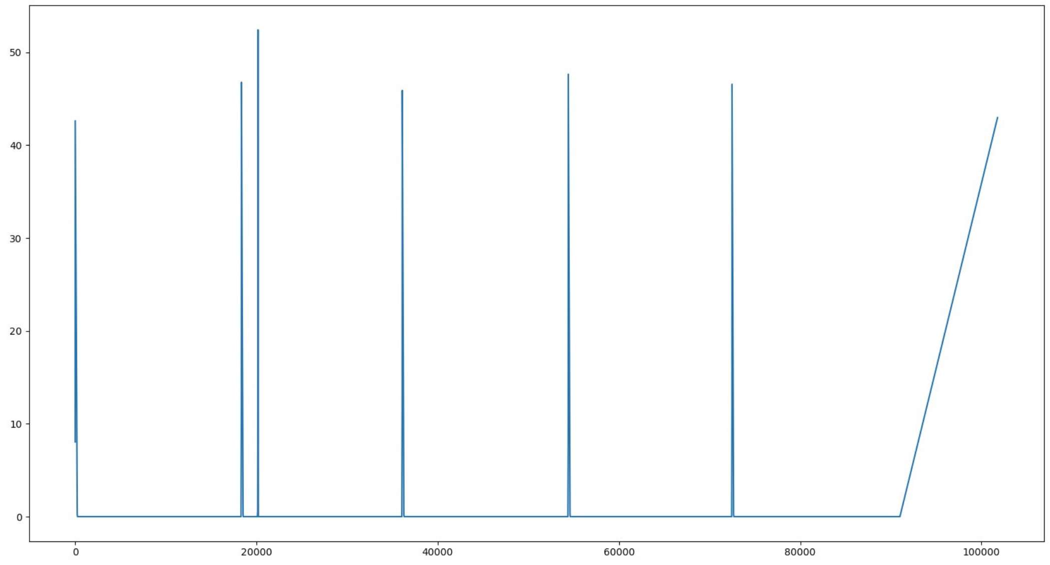
Compile Time: Drupal



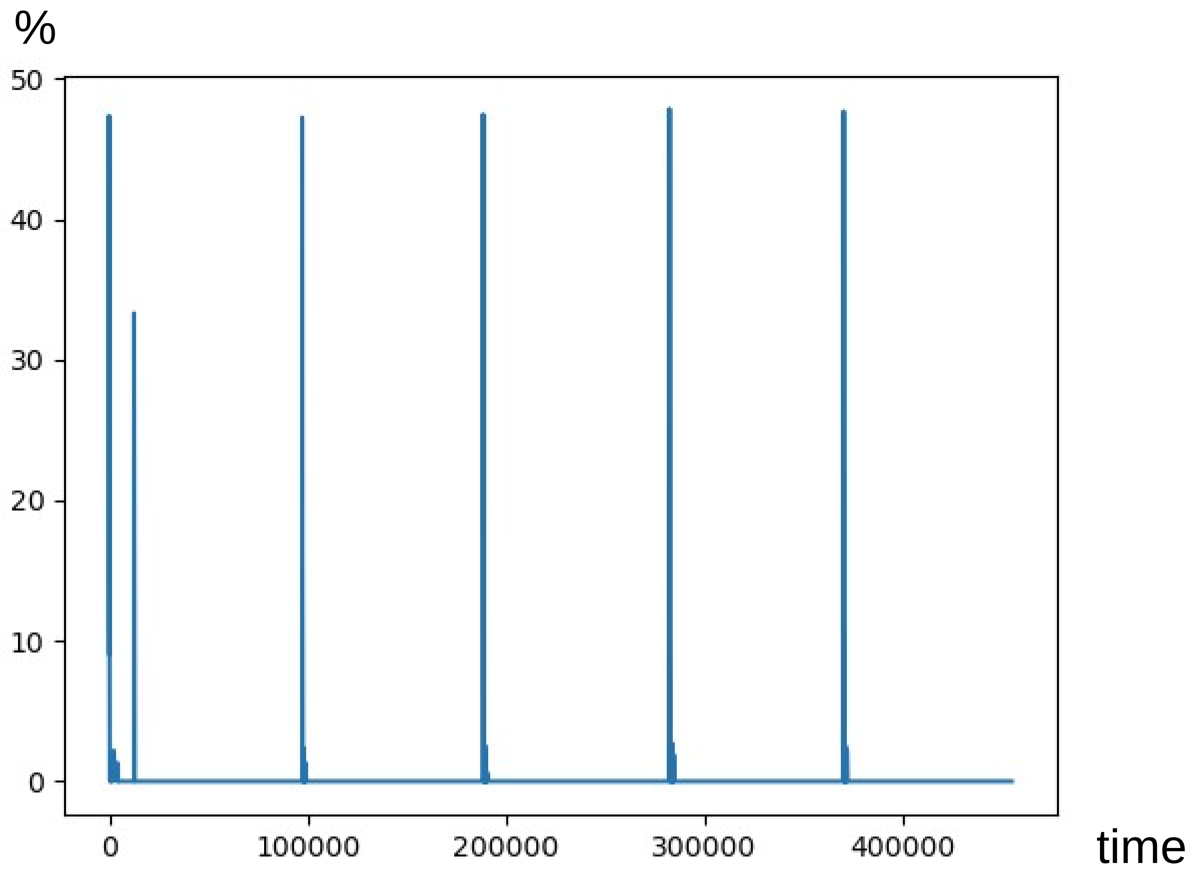


Compile Time: WordPress

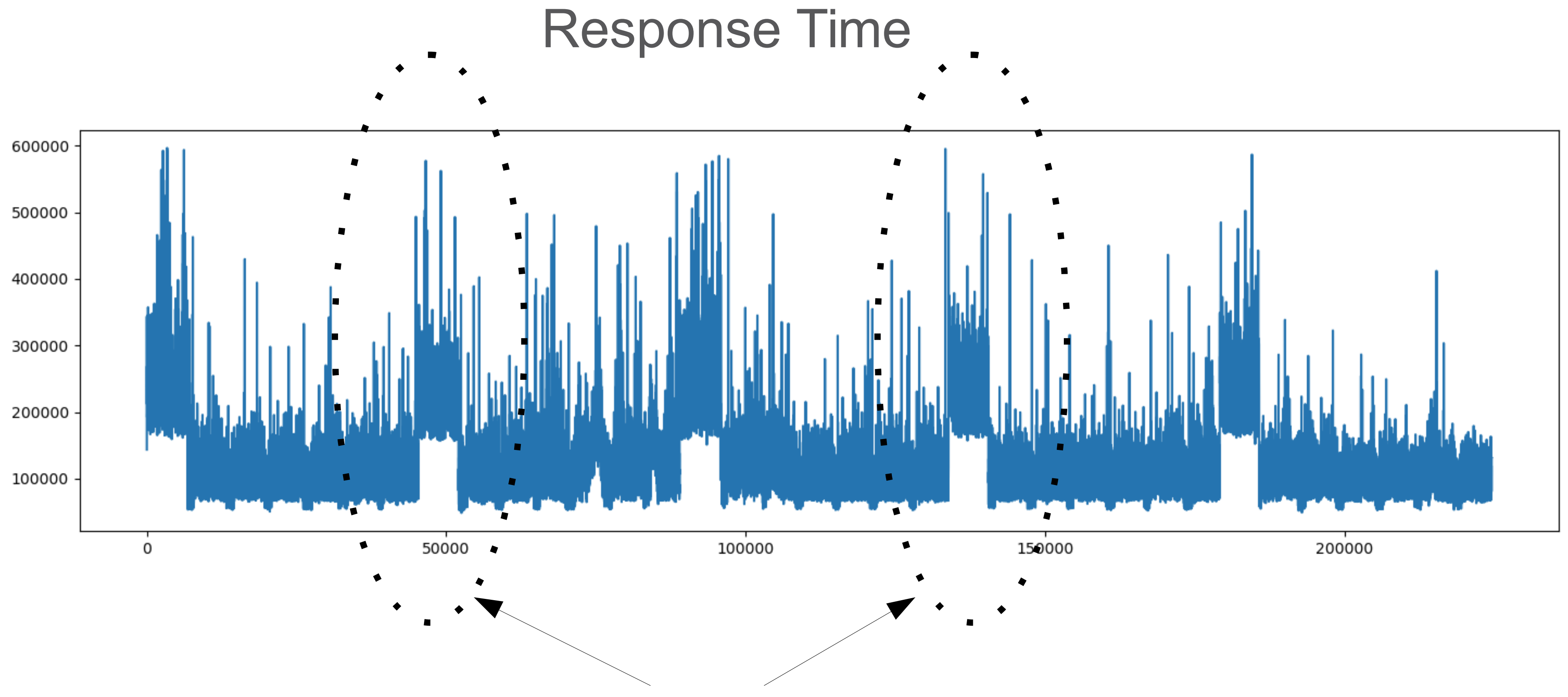




Compile Time: MediaWiki

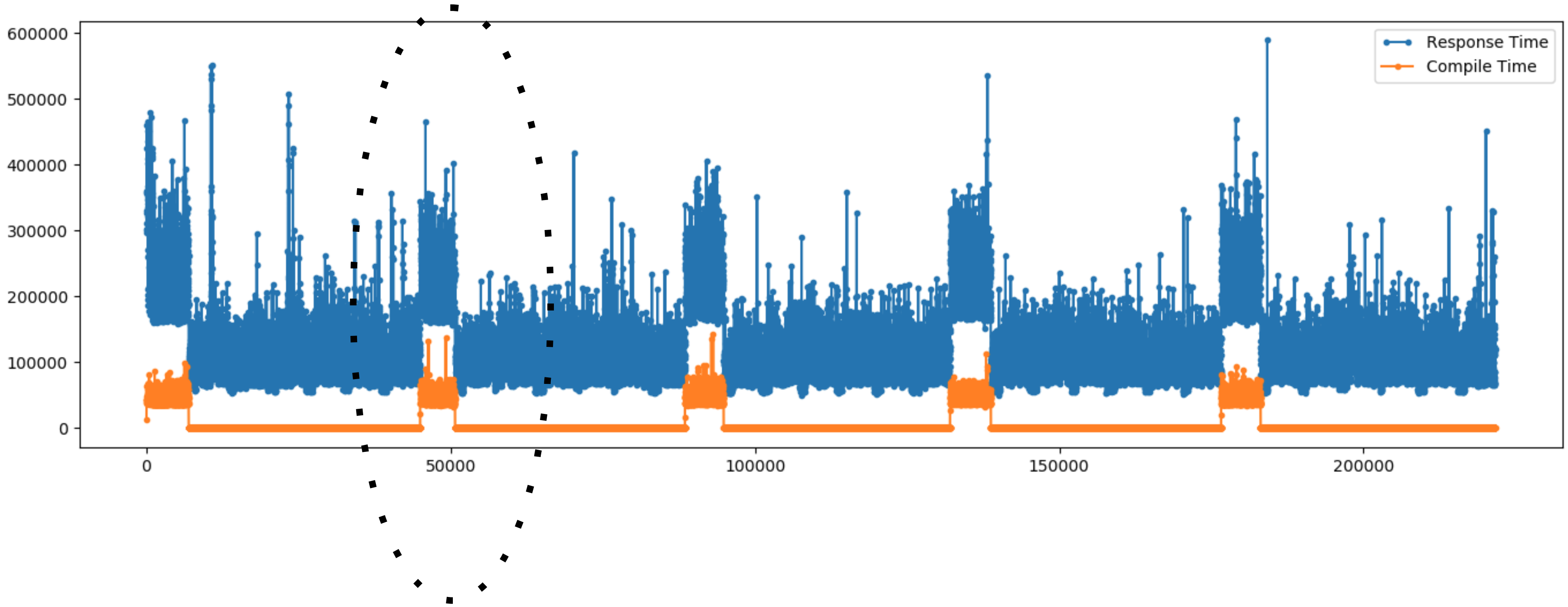


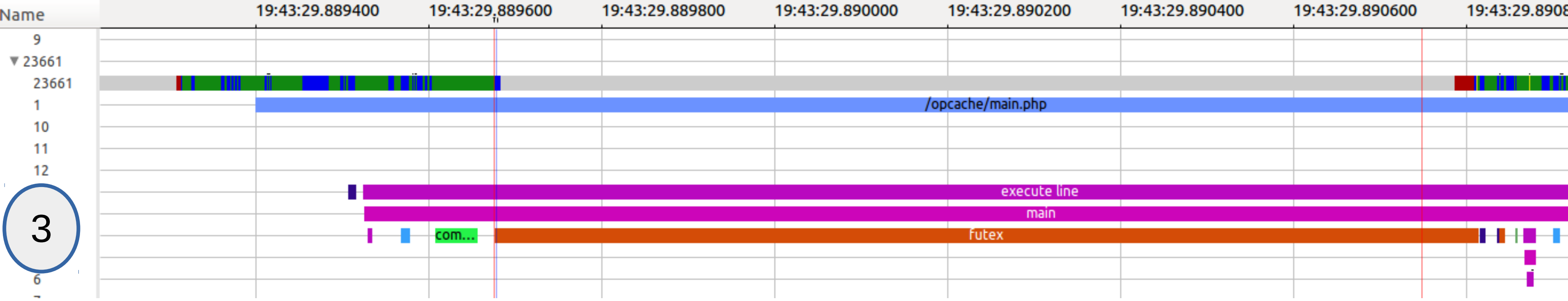
Let's back to our example

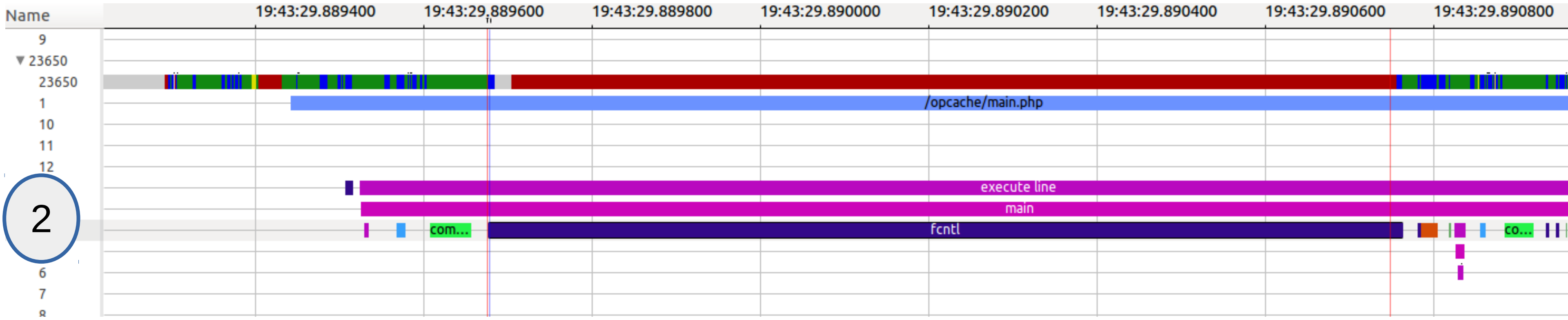


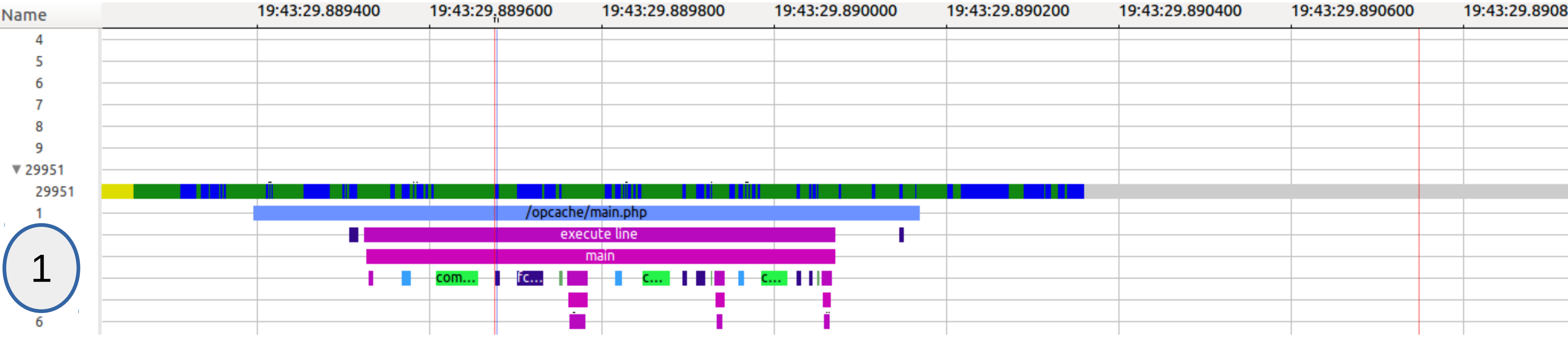
What happens?

Response Time vs Compile Time

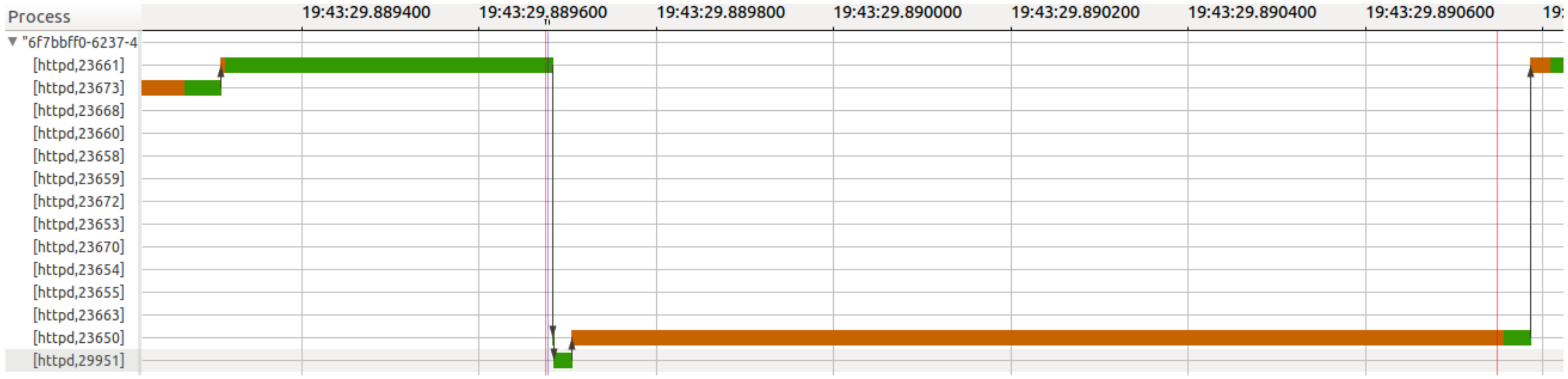



Name
19:43:29.889400
19:43:29.889600
19:43:29.889800
19:43:29.890000
19:43:29.890200
19:43:29.890400
19:43:29.890600
9
23661
23661
1
10
11
12
6
/opcache/main.php
execute line
main
com...
futex
3


Name
19:43:29.889400
19:43:29.889600
19:43:29.889800
19:43:29.890000
19:43:29.890200
19:43:29.890400
19:43:29.890600
19:43:29.890800
9
23650
23650
1
10
11
12
6
7
/opcache/main.php
execute line
main
com...
fcntl
co...
2

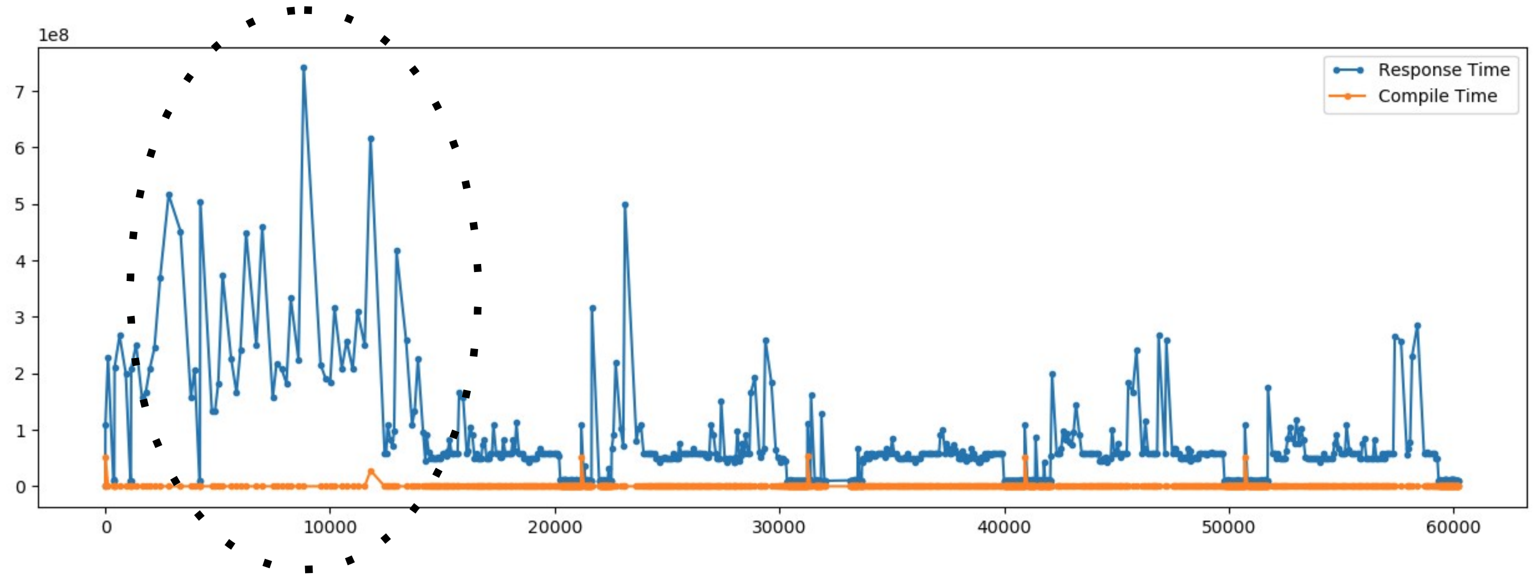

Name
19:43:29.889400
19:43:29.889600
19:43:29.889800
19:43:29.890000
19:43:29.890200
19:43:29.890400
19:43:29.890600
4
5
6
7
8
9
29951
29951
1
6
/opcache/main.php
execute line
main
com...
fc...
c...
c...
1

Critical Flow



Every PHP process that is willing to write into shared memory will lock every other process willing to write into shared memory as well.

Example2: Drupal Website

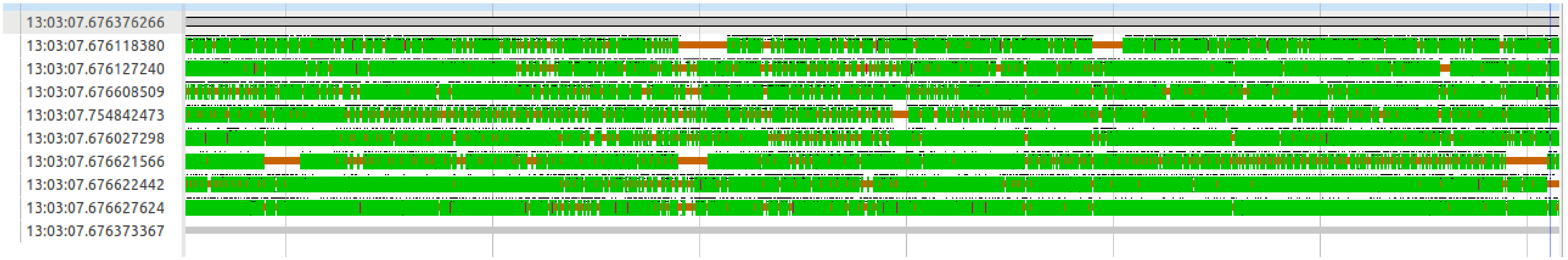


Not compilation issue anymore! What causes those latencies?

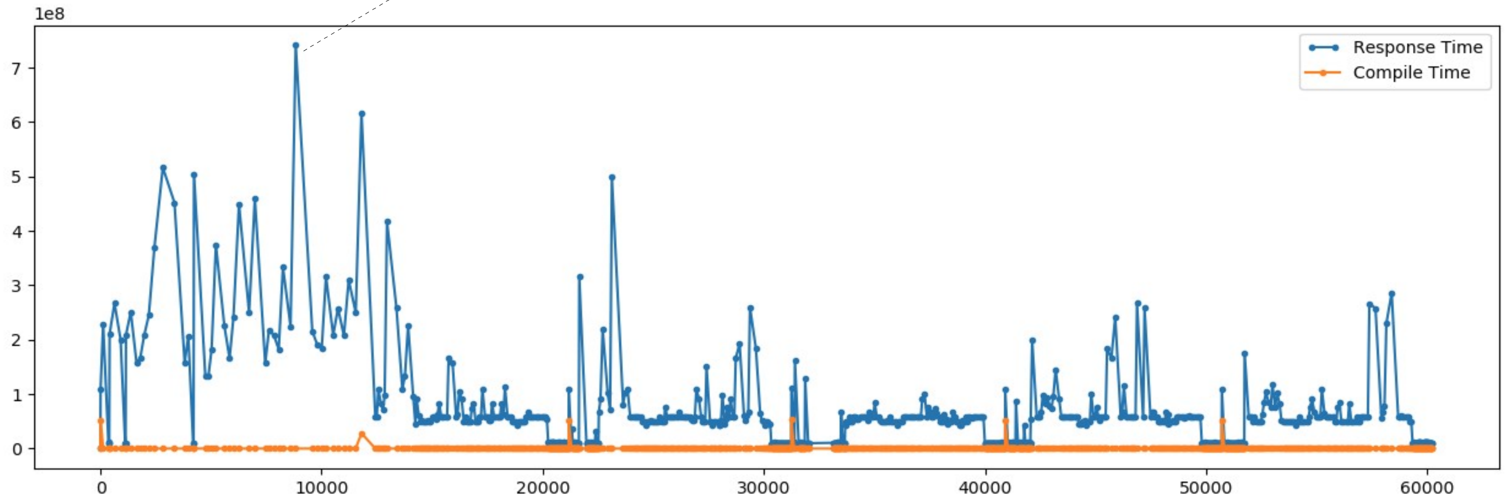
Php itself?
Databse query?
The web server?
A background process (or another vm) in the system?
A problem in the network?
...

Maybe, we need more data?

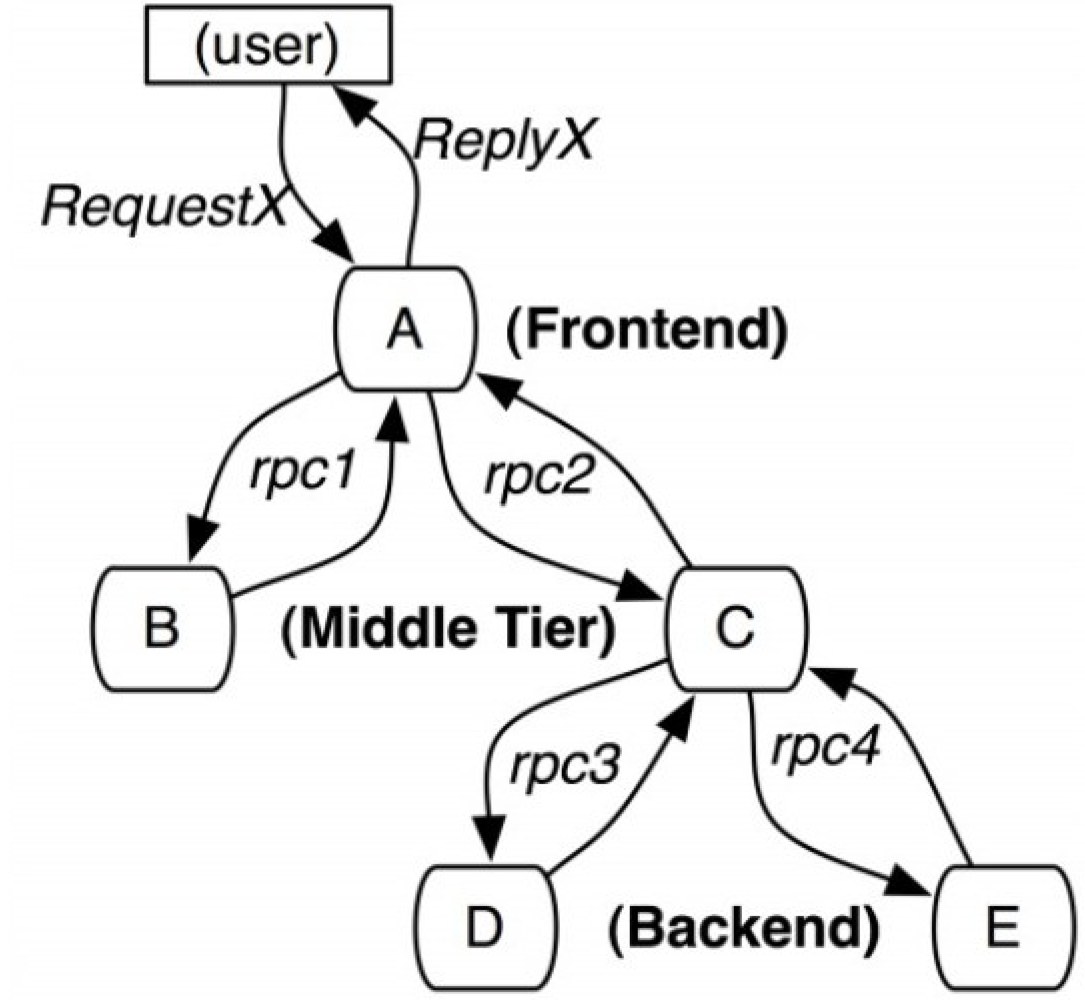
We trace all components and get more data !



How can I only follow a specific?

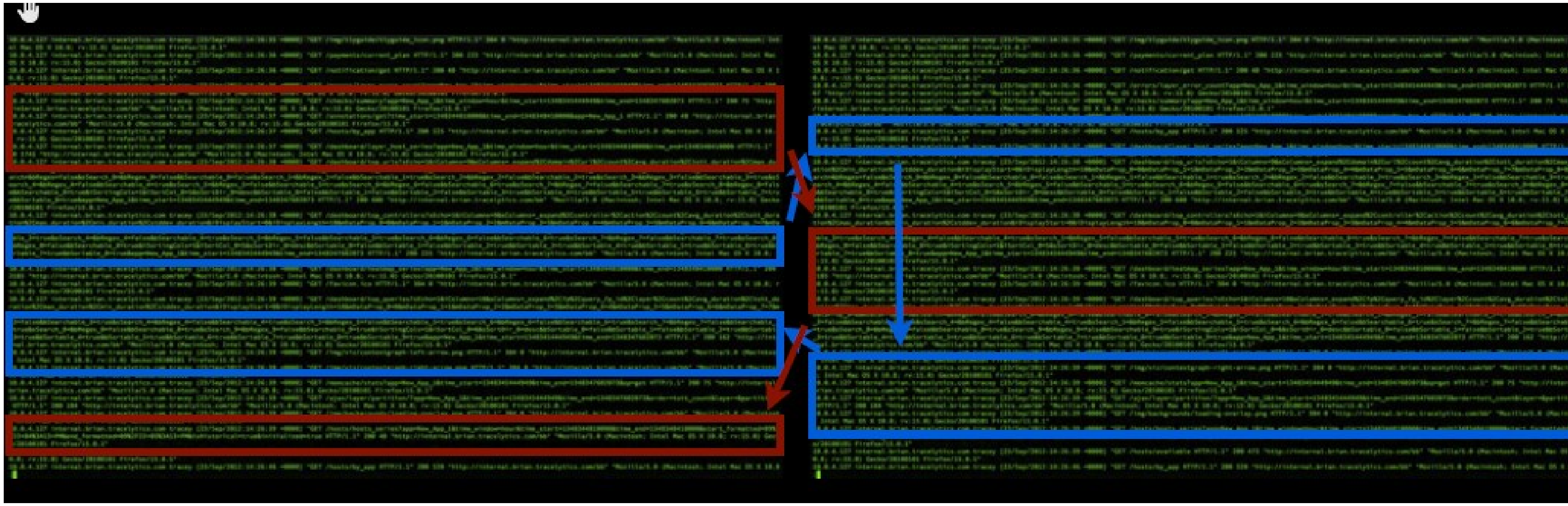


A General Case: Latency in a Distributed System



Data from all components

Causal Relationship (causal flow of control)



```
[illegible]
```

OpenTracing API

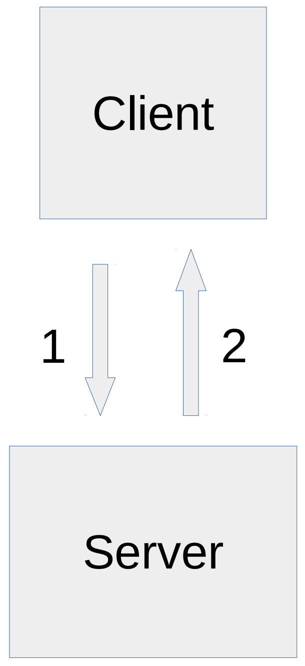
- The OpenTracing project provides a standard, portable API for distributed tracing instrumentation.
  - Ver 1.0 annonuced in Aug 2016.
- Consistent semantics across languages
  - Makes it easy for developers to add (or switch) tracing implementations with an O(1) configuration change

Uses for OpenTracing

- Logging - Easy to output to any logging tool.
- Metrics/Alerting - Measure based on tags, span timing, log data.
- Critical Path Analysis - Drill down into request latency in very low granularity.
- Context Propagation - Use baggage to carry request and user ID’s, etc.
- System Topology Analysis - Identify bottlenecks due to shared resources.

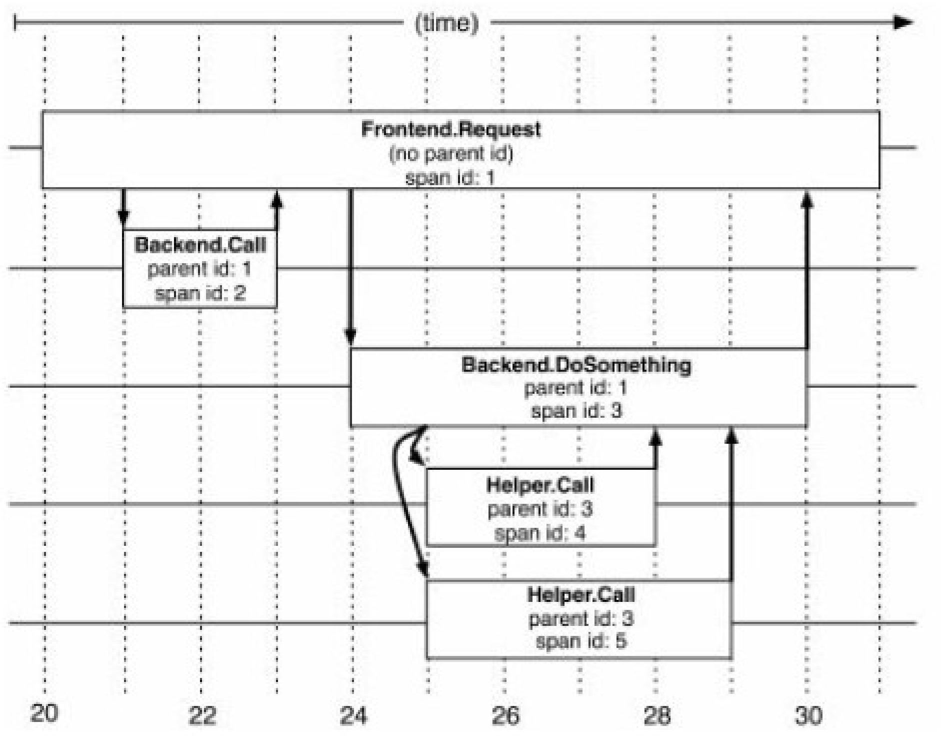
Trace Definistion in OpenTracing

- At the highest level, a trace tells the story of a transaction or workflow as it propagates through a (potentially distributed) system.
- In OpenTracing, a trace is a directed acyclic graph (DAG) of "spans"
  - Named, timed operations representing a contiguous segment of work in that trace
  - Each component in a distributed trace will contribute its own span or spans

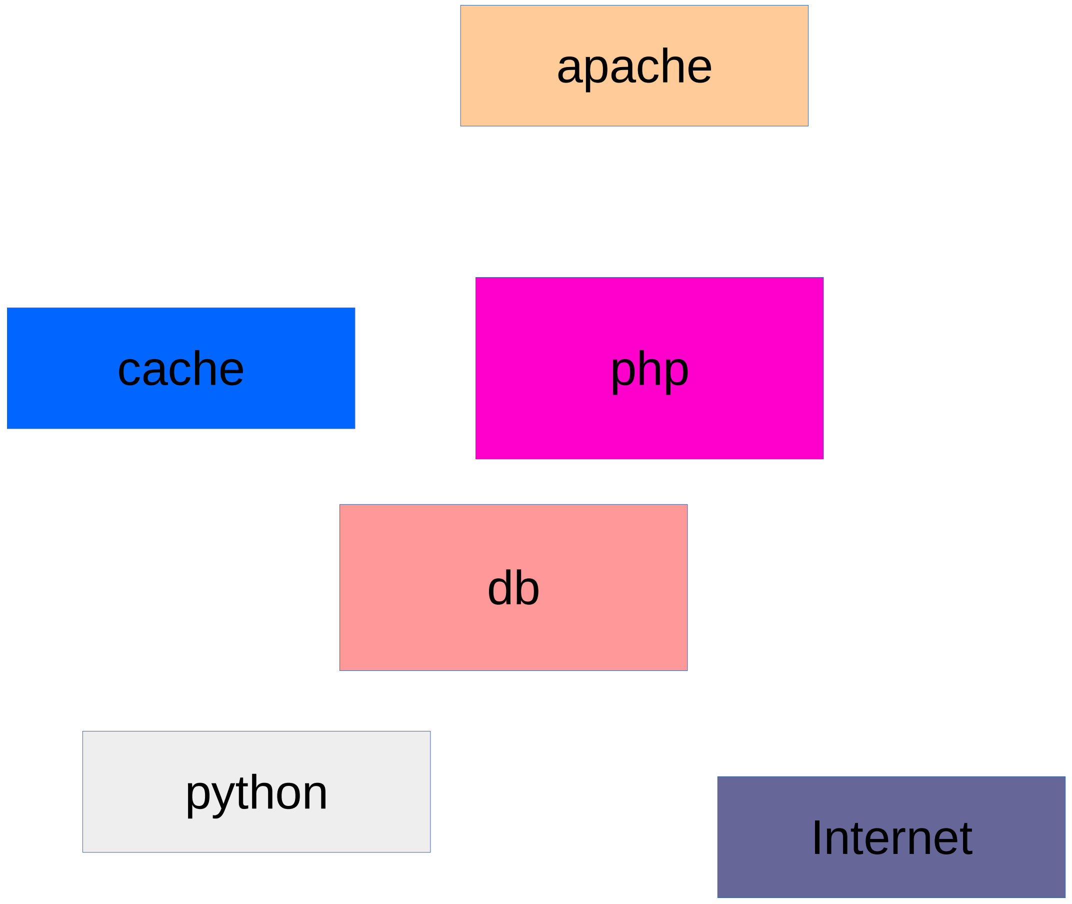


Data Model

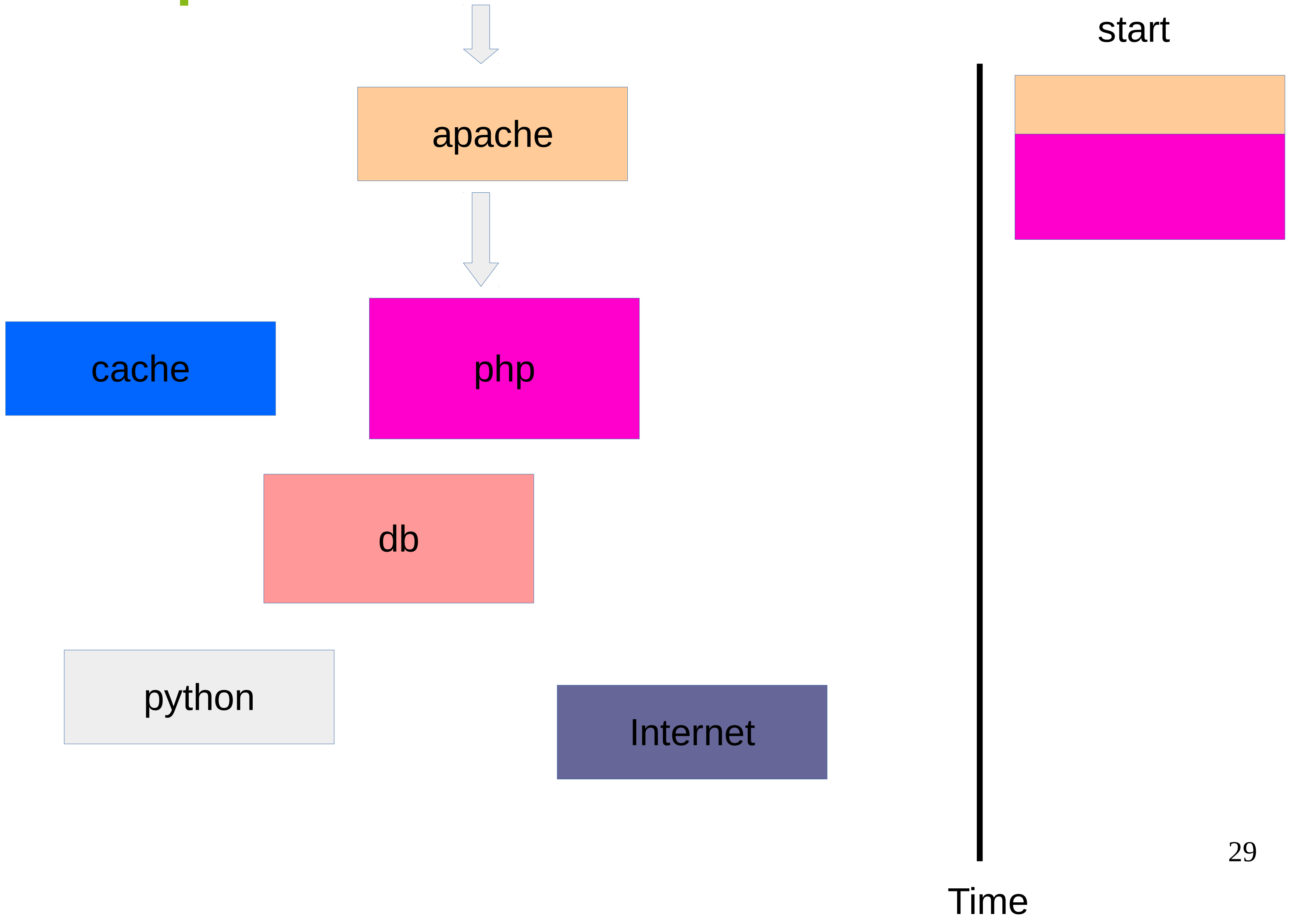
- Common libraries for several programming languages
  - Javascript
  - Python
  - Java
  - C
  - C#
  - Go
  - Ruby
  - PHP
- Trace as a tree of nested calls
  - Spans
  - Trace trees



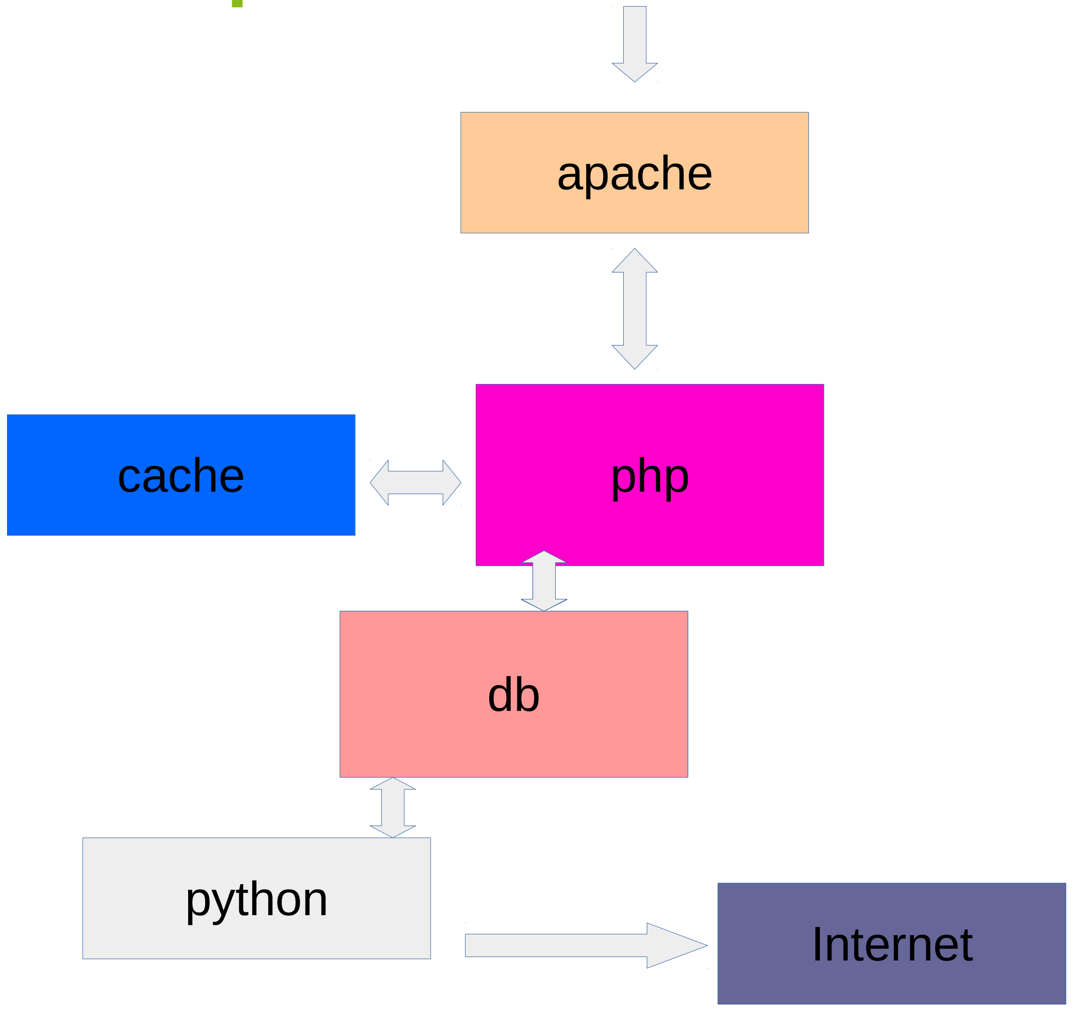
Causal Flow of Control

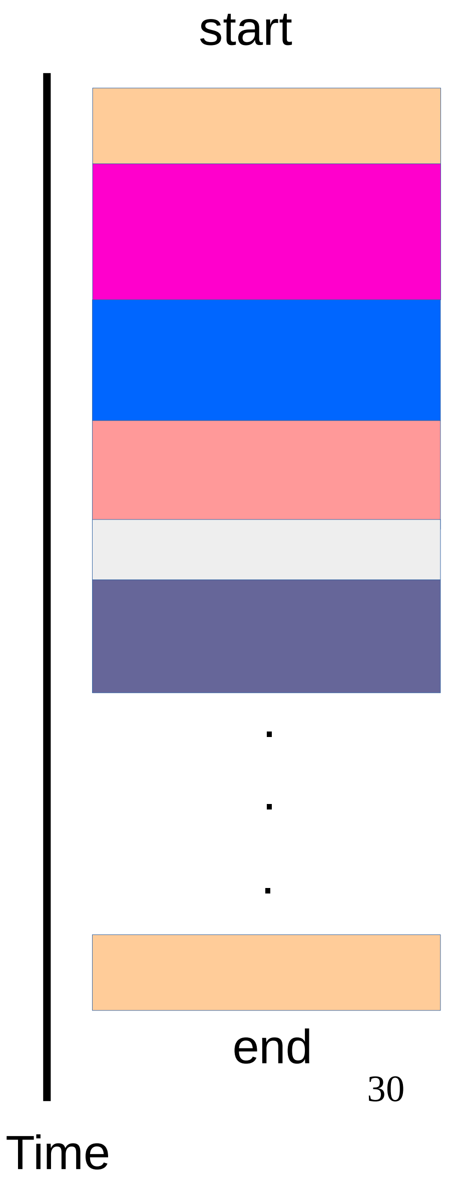


Example: Causal Flow of Control

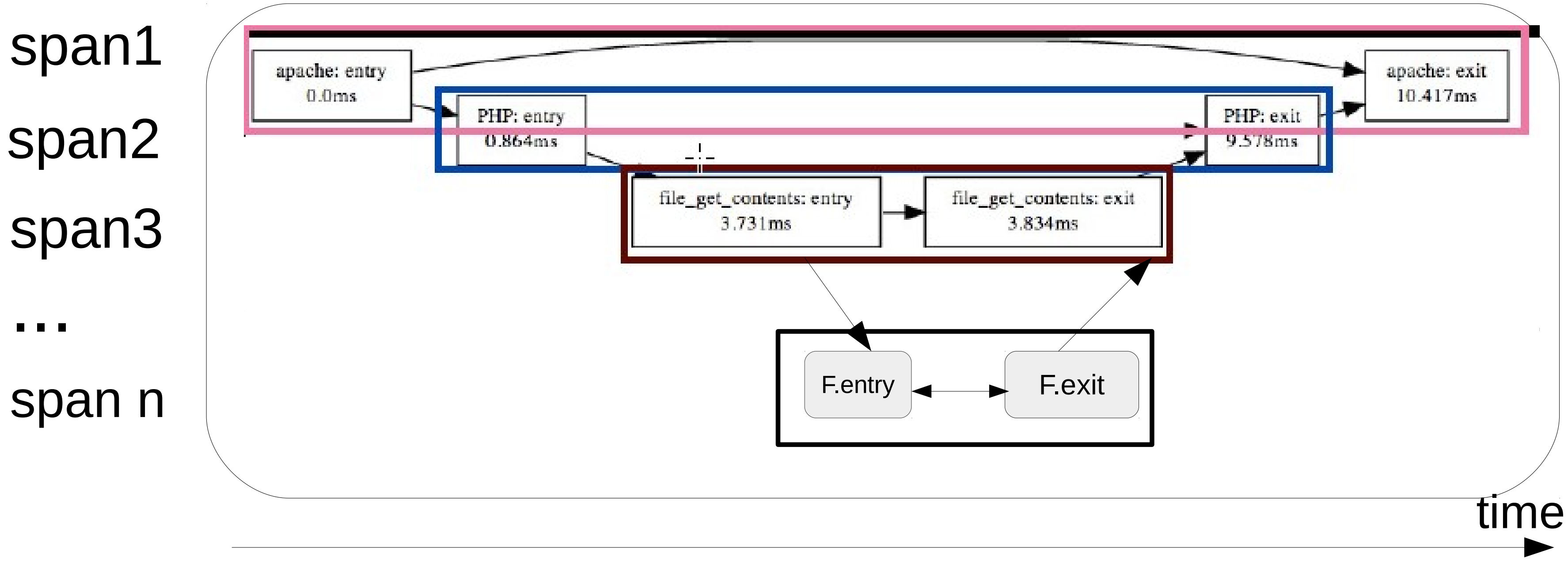


Example: Causal Flow of Control





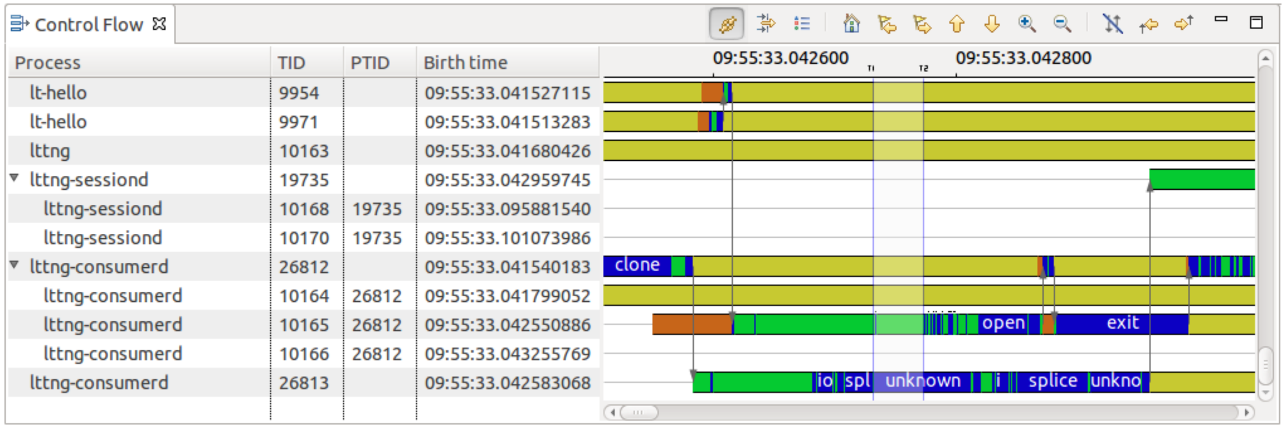
Timeline View



- This type of visualization adds the **context of time**, the **hierarchy** of the services involved, and the serial or parallel nature of the process/task execution.
- This view **highlights the system's critical path**. By focusing on the critical path, attention can focus on the area of code where the most valuable improvements can be made.
  - For example, you might want to trace the resource allocation spans inside an API request down to the underlying blocking calls.

- Can we have this in LTTng and Trace Compass?
  - How to relate the traces/events/spans?
  - How to visualize them?

- We already have some notions of LINK in trace compass.
  - Arrows in control flow diagram
    - Arrows: scheduler switches from one process to another for a given CPU
  - Critical path analysis
  - Call stack
    - Function calls
  - **Why not link everything?**



Process	TID	PTID	Birth time
lt-hello	9954		09:55:33.041527115
lt-hello	9971		09:55:33.041513283
lttng	10163		09:55:33.041680426
▾ lttng-sessiond	19735		09:55:33.042959745
lttng-sessiond	10168	19735	09:55:33.095881540
lttng-sessiond	10170	19735	09:55:33.101073986
▾ lttng-consumerd	26812		09:55:33.041540183
lttng-consumerd	10164	26812	09:55:33.041799052
lttng-consumerd	10165	26812	09:55:33.042550886
lttng-consumerd	10166	26812	09:55:33.043255769
lttng-consumerd	26813		09:55:33.042583068

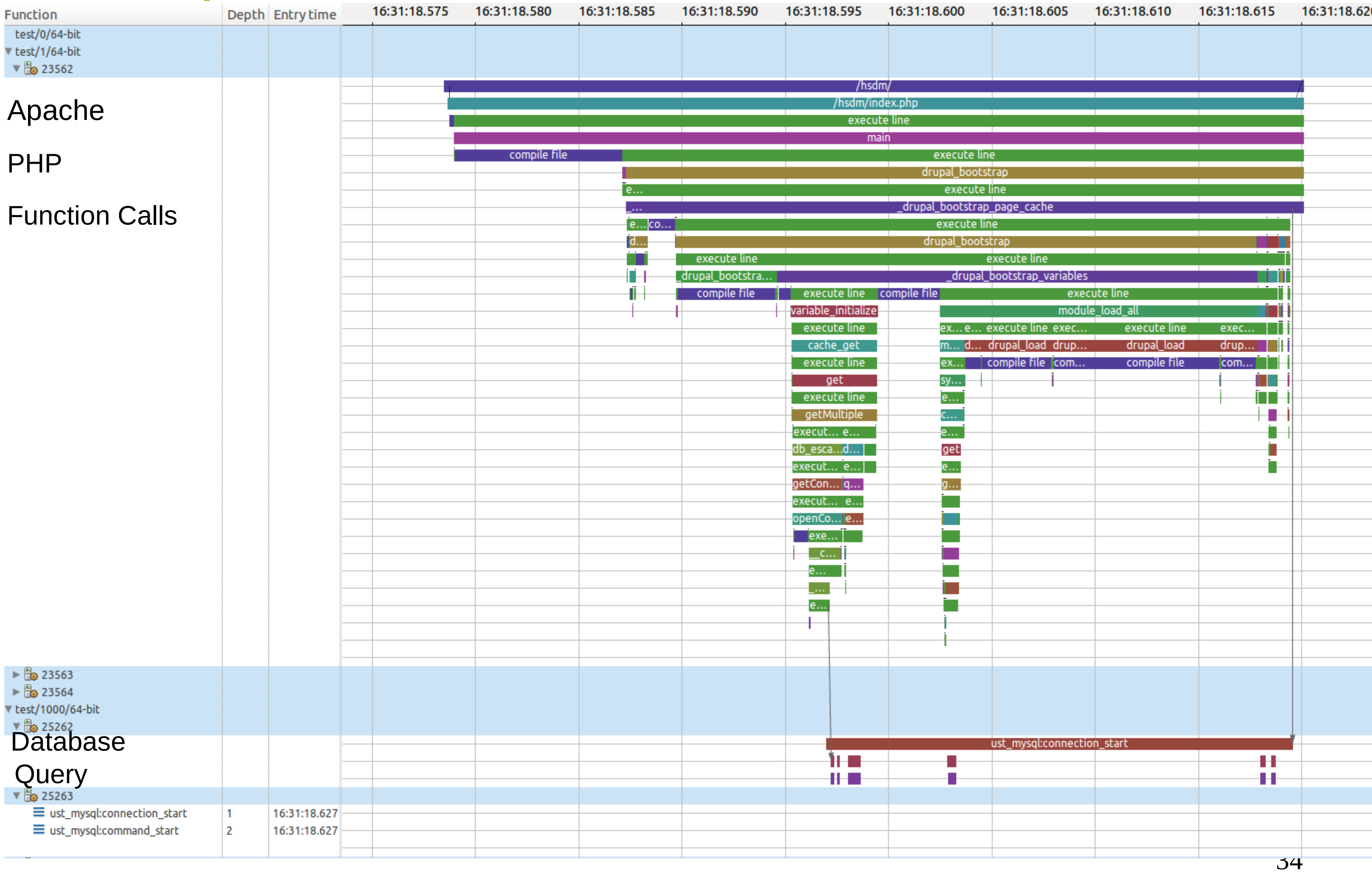
Example

Apache

PHP

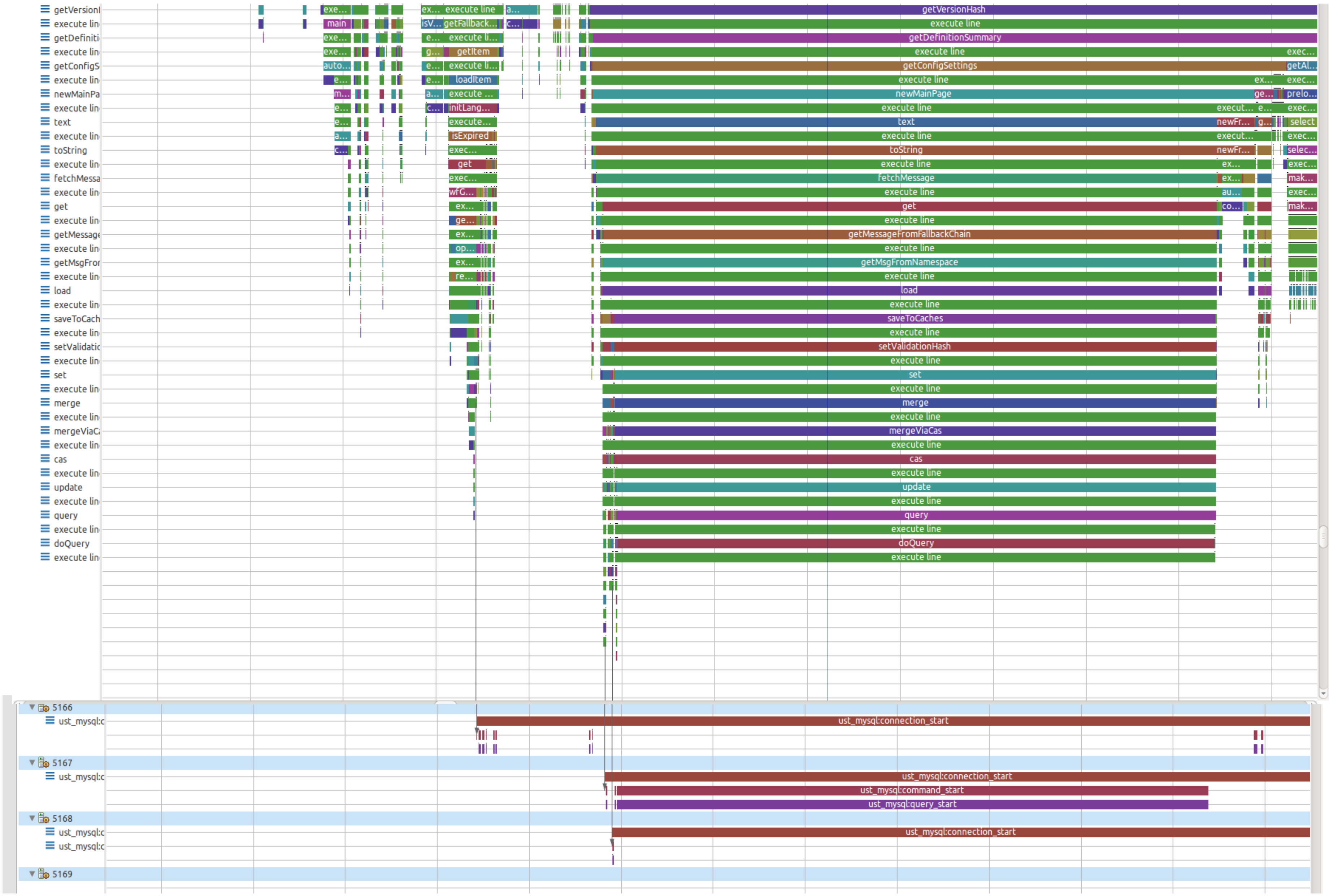
Function Calls

Database Query

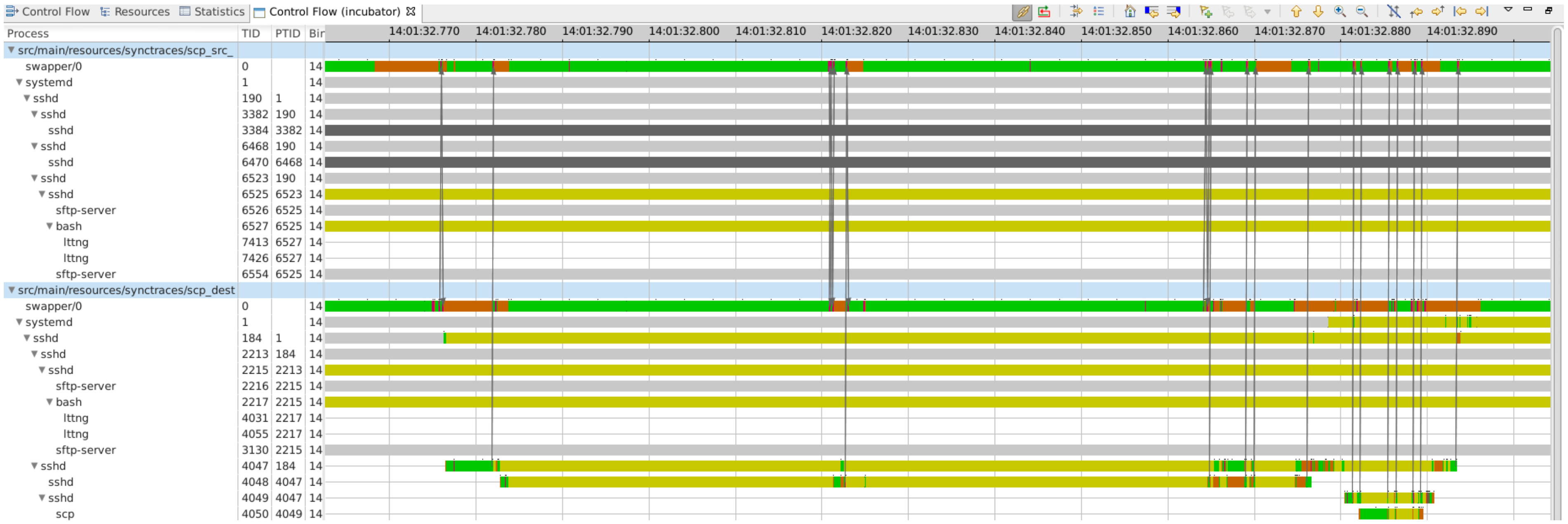


Function	Depth	Entry time
test/0/64-bit		
▼ test/1/64-bit		
▼ 23562		
▶ 23563		
▶ 23564		
▼ test/1000/64-bit		
▼ 25262		
▼ 25263		
ust_mysql:connection_start	1	16:31:18.627
ust_mysql:command_start	2	16:31:18.627

Example (cntd)



Linked Traces & Linked Events => Linked logics

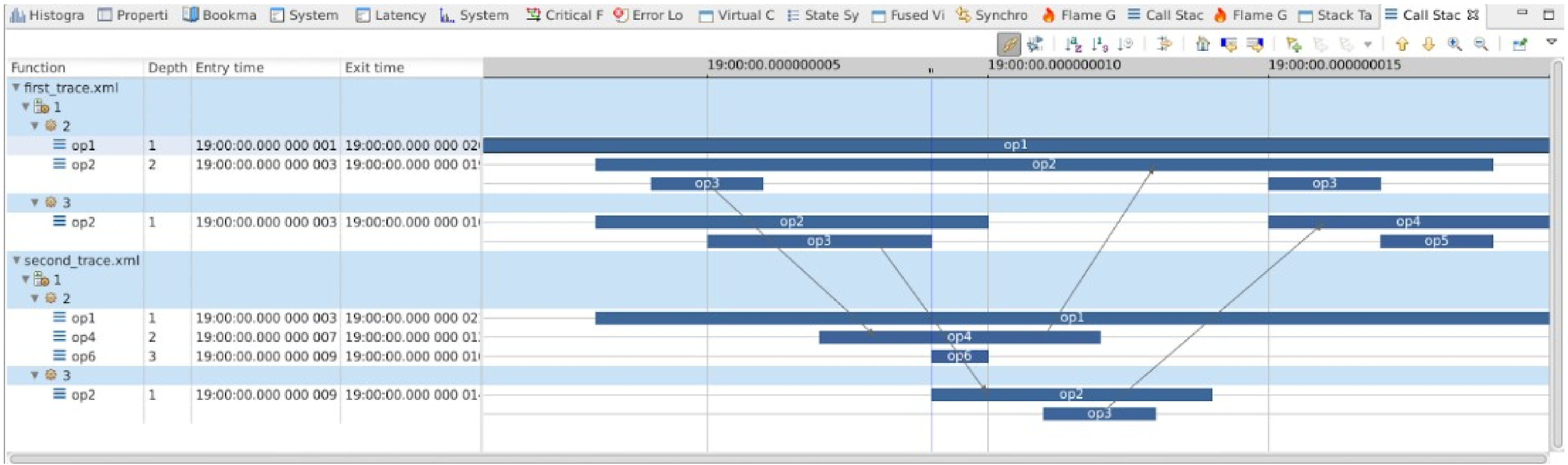


Process	TID	PTID	Bir
▼ src/main/resources/synctraces/scp_src_			
swapper/0	0		14
▼ systemd	1		14
▼ sshd	190	1	14
▼ sshd	3382	190	14
sshd	3384	3382	14
▼ sshd	6468	190	14
sshd	6470	6468	14
▼ sshd	6523	190	14
▼ sshd	6525	6523	14
sftp-server	6526	6525	14
▼ bash	6527	6525	14
lttng	7413	6527	14
lttng	7426	6527	14
sftp-server	6554	6525	14
▼ src/main/resources/synctraces/scp_dest			
swapper/0	0		14
▼ systemd	1		14
▼ sshd	184	1	14
▼ sshd	2213	184	14
▼ sshd	2215	2213	14
sftp-server	2216	2215	14
▼ bash	2217	2215	14
lttng	4031	2217	14
lttng	4055	2217	14
sftp-server	3130	2215	14
▼ sshd	4047	184	14
sshd	4048	4047	14
▼ sshd	4049	4047	14
scp	4050	4049	14

Conclusion

- EventMatching Class in Trace Compass
  - Manual
    - Hardcoded links between event
      - Event a from trace 1 matches to event b from trace 2

- We need a standard way to define the semantic relationships
    - To deinfe the events and fields that match to each other
      - Everyone can define and trace the links they need in the logics of their code/application/system
    - The standard can be Opentracing Api
  - In LTTng:
    - To support the above standard
  - In Trace Compass
    - A gneric Event Matching class to analyse and visualze the links between events
    - View!



Function	Depth	Entry time	Exit time
first_trace.xml			
1			
2			
op1	1	19:00:00.000 000 001	19:00:00.000 000 02
op2	2	19:00:00.000 000 003	19:00:00.000 000 01
3			
op2	1	19:00:00.000 000 003	19:00:00.000 000 01
second_trace.xml			
1			
2			
op1	1	19:00:00.000 000 003	19:00:00.000 000 02
op4	2	19:00:00.000 000 007	19:00:00.000 000 01
op6	3	19:00:00.000 000 009	19:00:00.000 000 01
3			
op2	1	19:00:00.000 000 009	19:00:00.000 000 01

Q&A