Sto

A Better Way To Store And Query Profiler Data

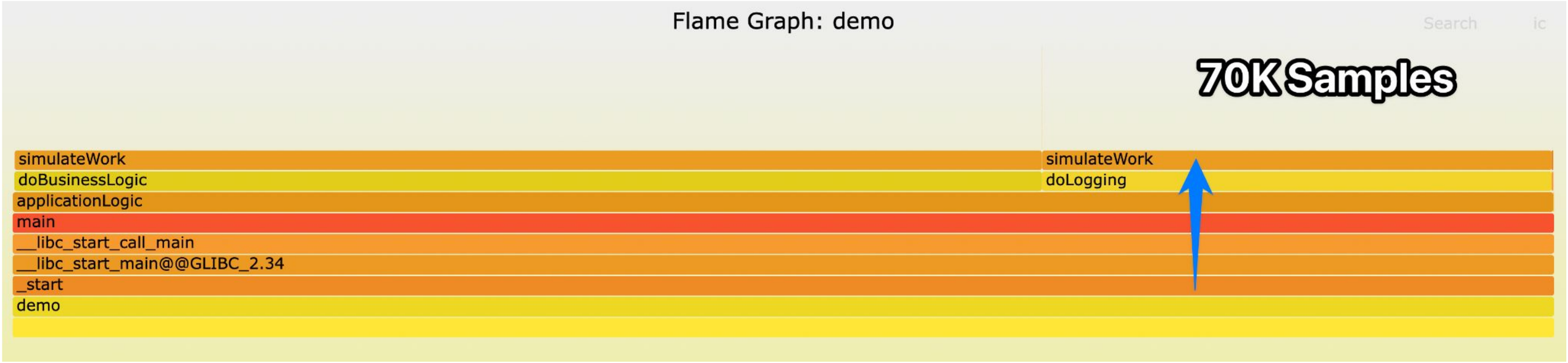
Patrick Somaru
Production Engineer

Meta

What is profiler data?

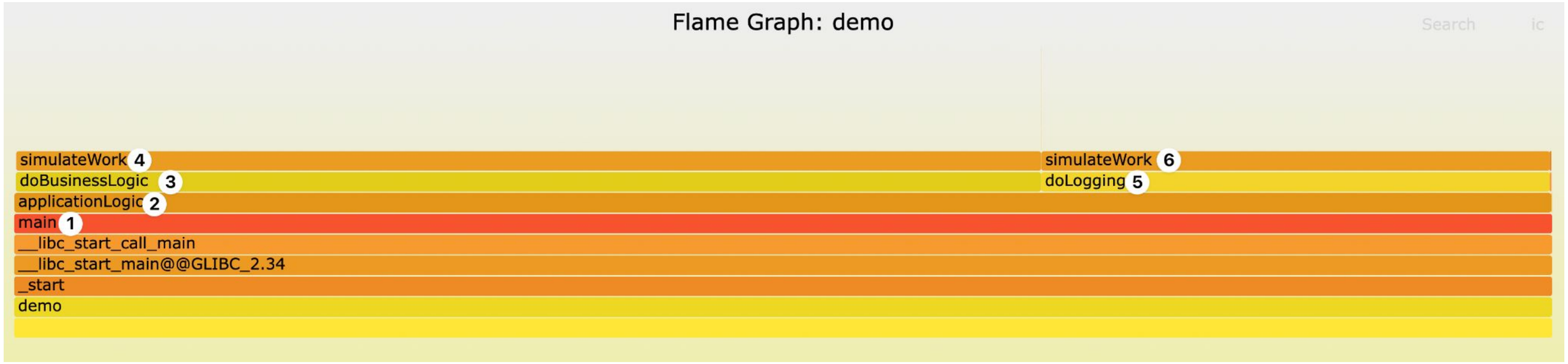
```
demo 26015 14943.211794:   18447857 cycles:u:
                    11a1 simulateWork+0x1c (/home/pat/repos/sto/demo/demo)
  /home/pat/repos/sto/demo/demo.c:5
                    11d4 doLogging+0x15 (/home/pat/repos/sto/demo/demo)
  /home/pat/repos/sto/demo/demo.c:9
                    1221 applicationLogic+0x4a (/home/pat/repos/sto/demo/demo)
  /home/pat/repos/sto/demo/demo.c:18
                    125c main+0x2c (/home/pat/repos/sto/demo/demo)
  /home/pat/repos/sto/demo/demo.c:22
                   23312 __libc_start_call_main+0x82 (/lib64/libc.so.6)
  /usr/src/debug/sys-libs/glibc-2.36-r5/glibc-2.36/csu/../sysdeps/nptl/libc_start_call_main.h:74
                   233d8 __libc_start_main@@GLIBC_2.34+0x88 (/lib64/libc.so.6)
  /usr/src/debug/sys-libs/glibc-2.36-r5/glibc-2.36/csu/../csu/libc-start.c:128
                    1091 _start+0x21 (/home/pat/repos/sto/demo/demo)
  /usr/src/debug/sys-libs/glibc-2.36-r5/glibc-2.36/csu/../sysdeps/x86_64/start.S:117
```

1 Sample

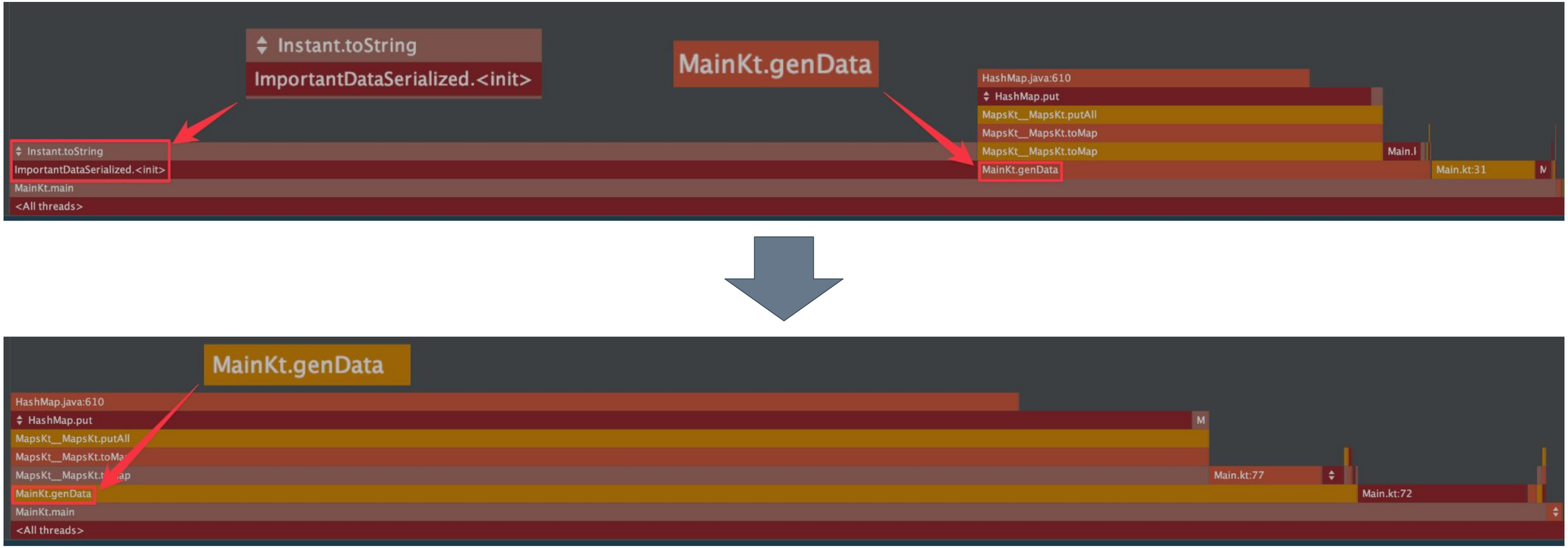


Tying things back to code

```c
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>

void simulateWork(int x) { for(int i = 10000*x; i>0; i--); }

void doBusinessLogic(int x) { simulateWork(x); }

void doLogging(int x) { simulateWork(x); }

void applicationLogic(int logPct) {
    if(rand() % 100 < logPct) {
        doBusinessLogic(1);
        doLogging(1);
    } else {
        doBusinessLogic(1);
    }
}

int main() {
    printf("%ld\n", (long)getpid());
    while(1){ applicationLogic(50); }
}
```

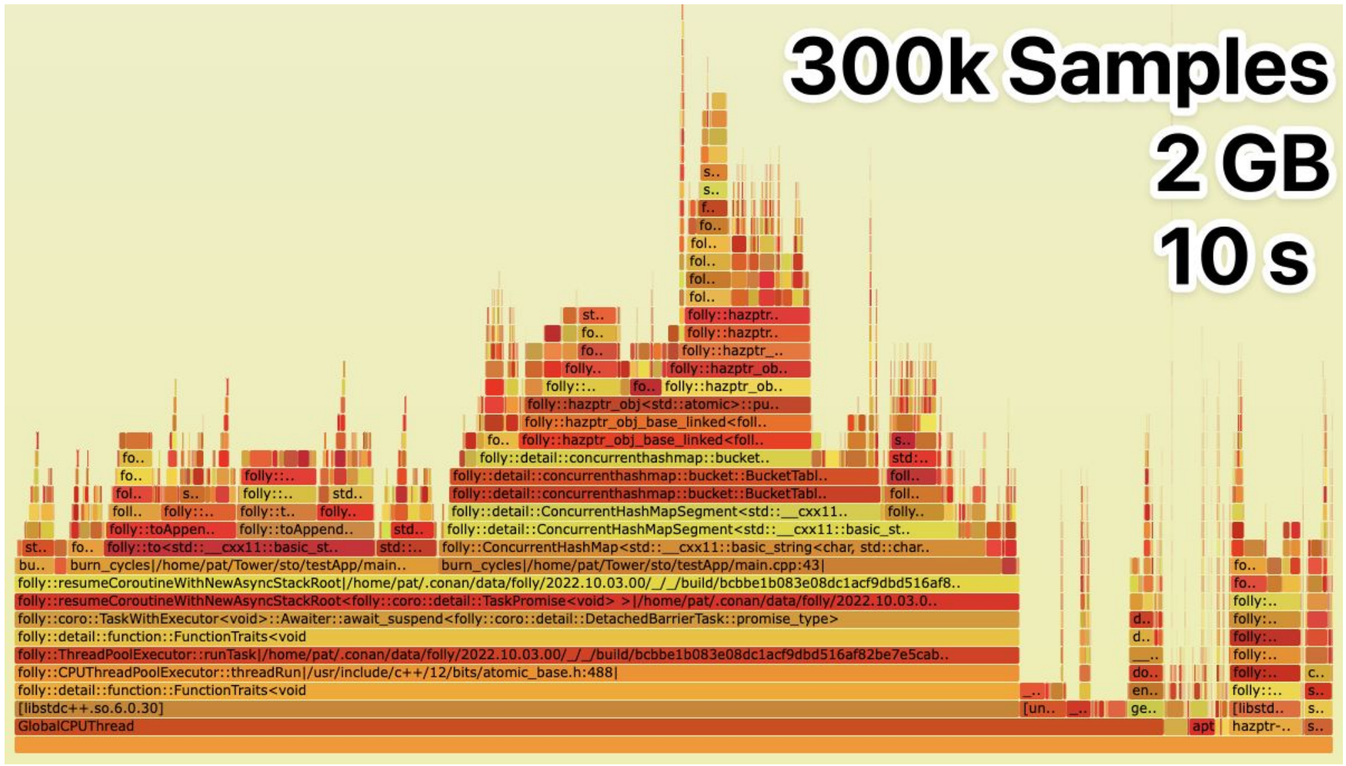


Why use Profilers?



Profiler Data

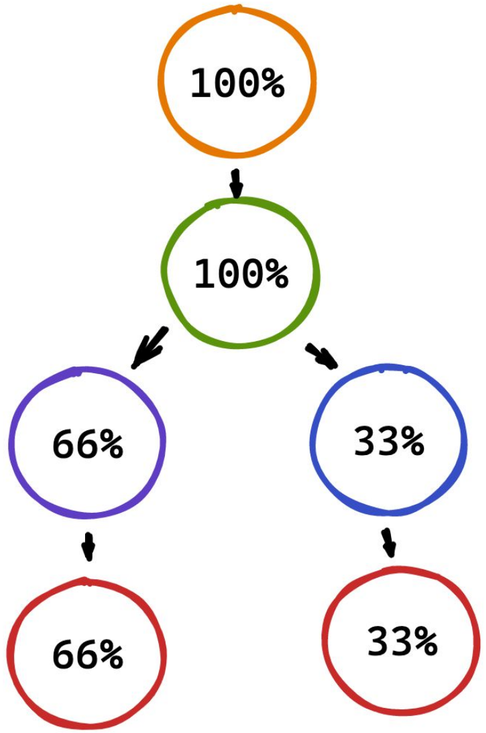
Challenges such as?



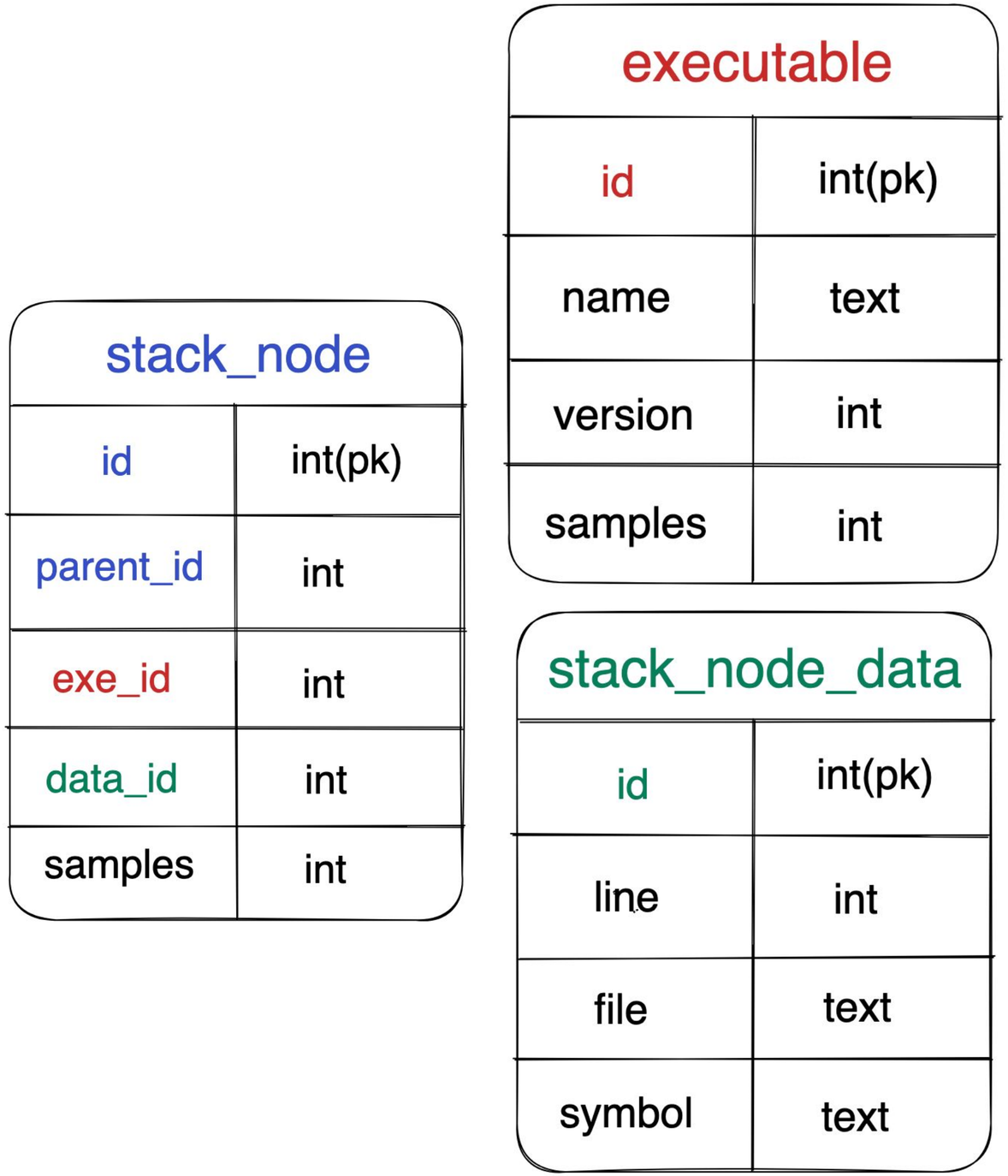
What We Found

Introducing Sto: by storing profiler data in a DAG, we can reduce it's footprint by **~10000x**. Think, 2gb => 5mb for suboptimal cases.

```
void simulateWork() { ... }

void doBusinessLogic() { simulateWork(); }

void doLogging() { simulateWork(); }

void applicationLogic() {
    if( ... 50%) {
        doBusinessLogic();
        doLogging();
    } else {
        doBusinessLogic();
    }
}

int main() { applicationLogic(); }
```



Sto Primitives



Primitive 1: Stack Node Data

stack_node_data	
id	int(pk)
line	int
file	text
symbol	text

```
/path/to/demo.c
01: void simulateWork() {  ...  }
02:
03: void doBusinessLogic() { simulateWork(); }
04:
05: void doLogging() { simulateWork(); }
06:
07: void applicationLogic() {
08:     if( ... 50%) {
09:         doBusinessLogic();
10:         doLogging();
11:     } else {
12:         doBusinessLogic();
13:     }
14: }
15:
16: int main() { applicationLogic(); }
```

id	file	symbol	line number
324	/path/to/demo.c	simulateWork	01
47653	/path/to/demo.c	doBusinessLogic	03
2345	/path/to/demo.c	doLogging	05
56742	/path/to/demo.c	applicationLogic	07
1234	/path/to/demo.c	main	16

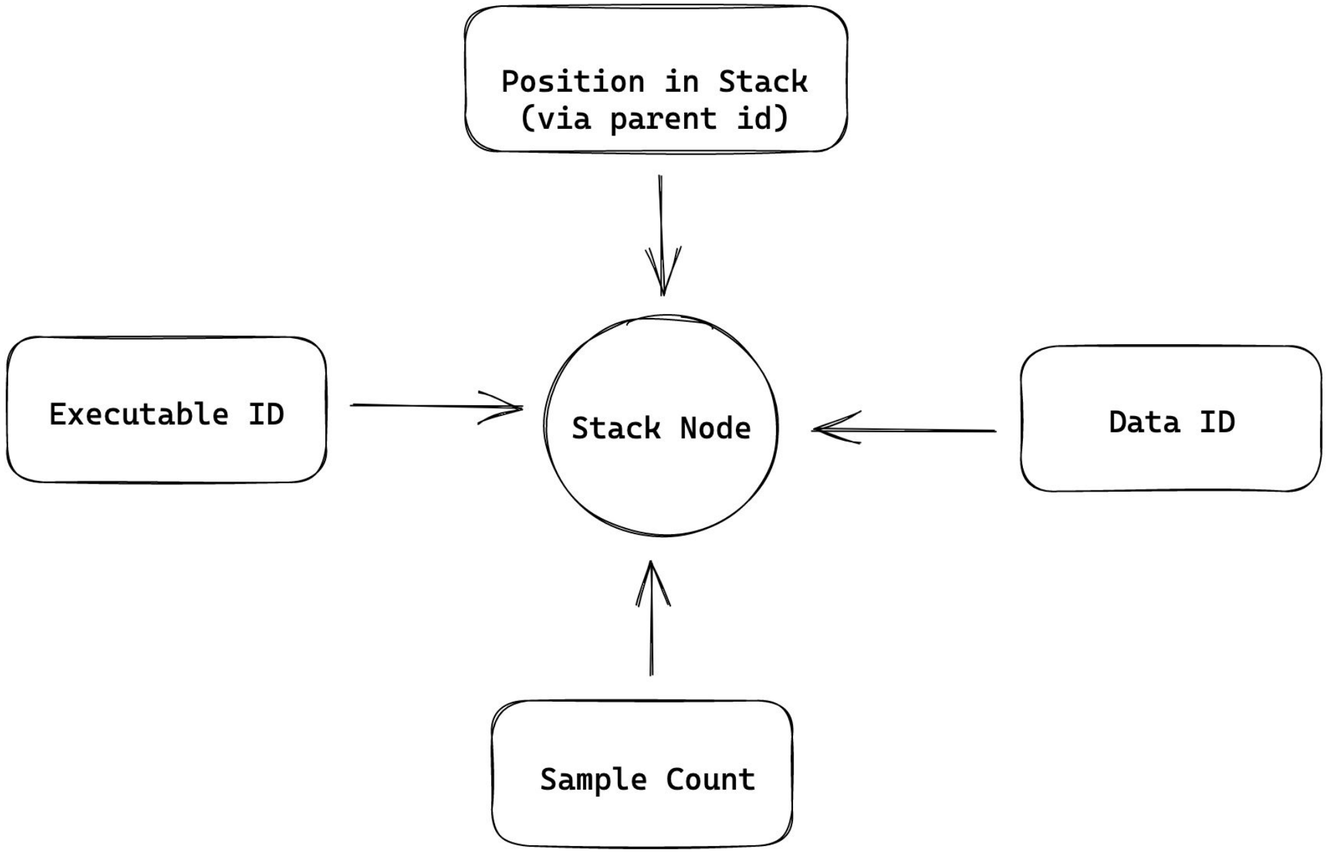
Primitive 2: Executable

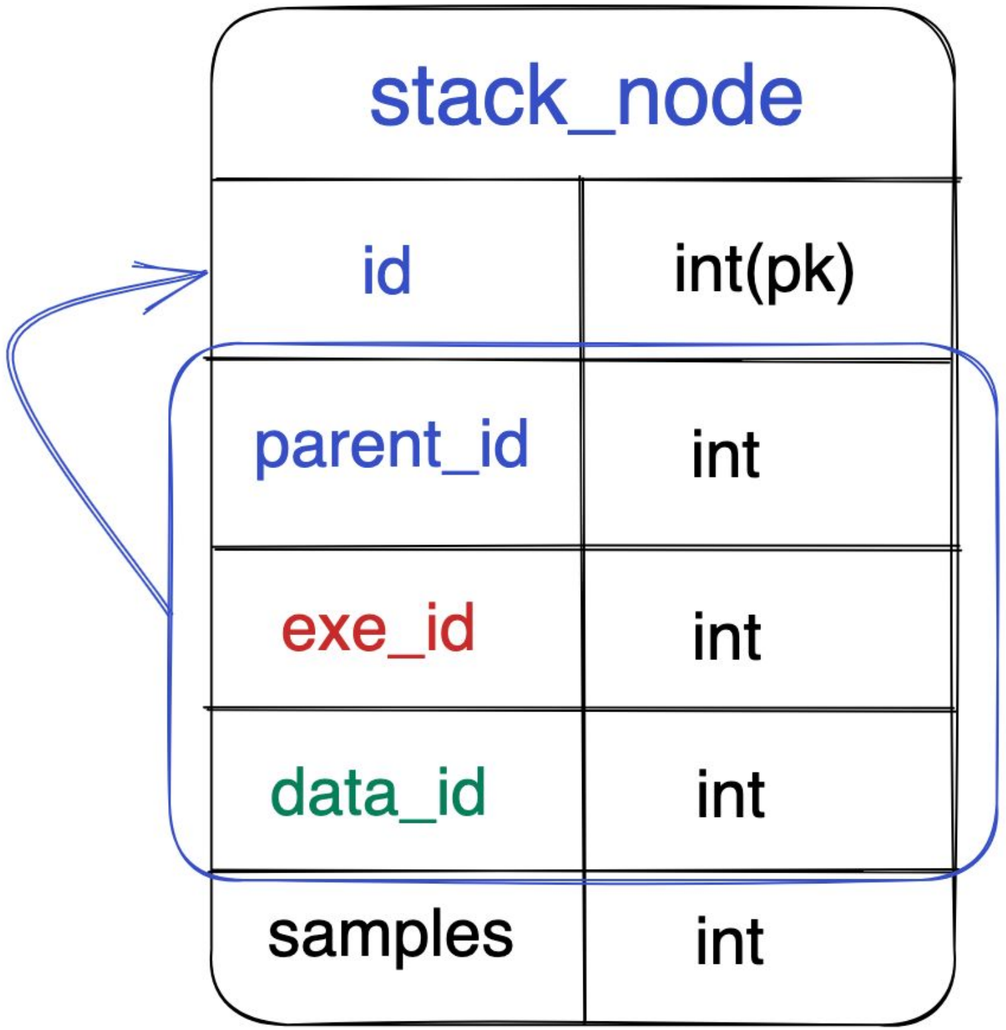
executable	
id	int(pk)
name	text
version	int
samples	int

/path/to/demo →

id	name	version	samples
hash(name,version)	cli argument	cli argument	calculated by cli

Primitive 3: Stack Node

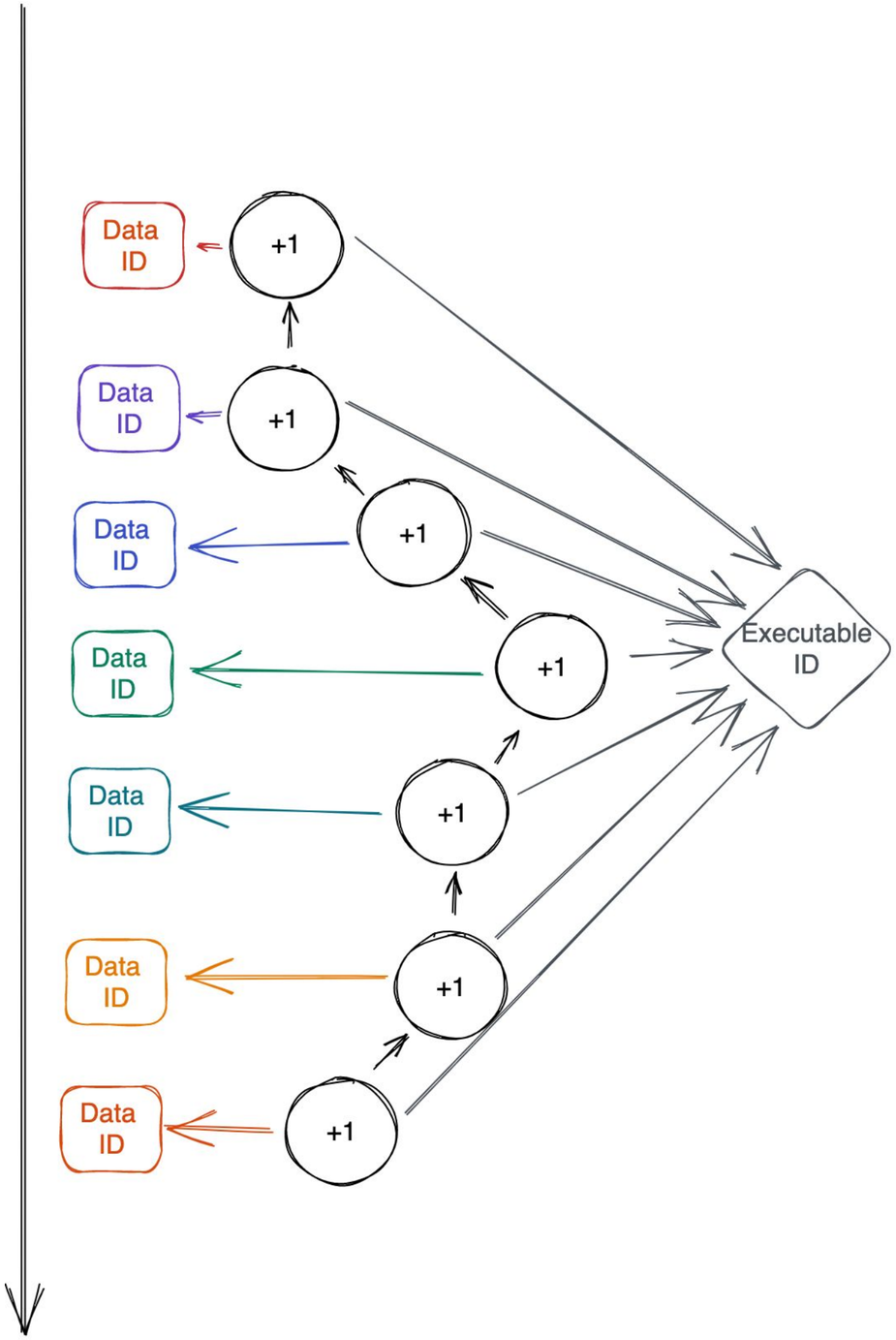




What We Developed

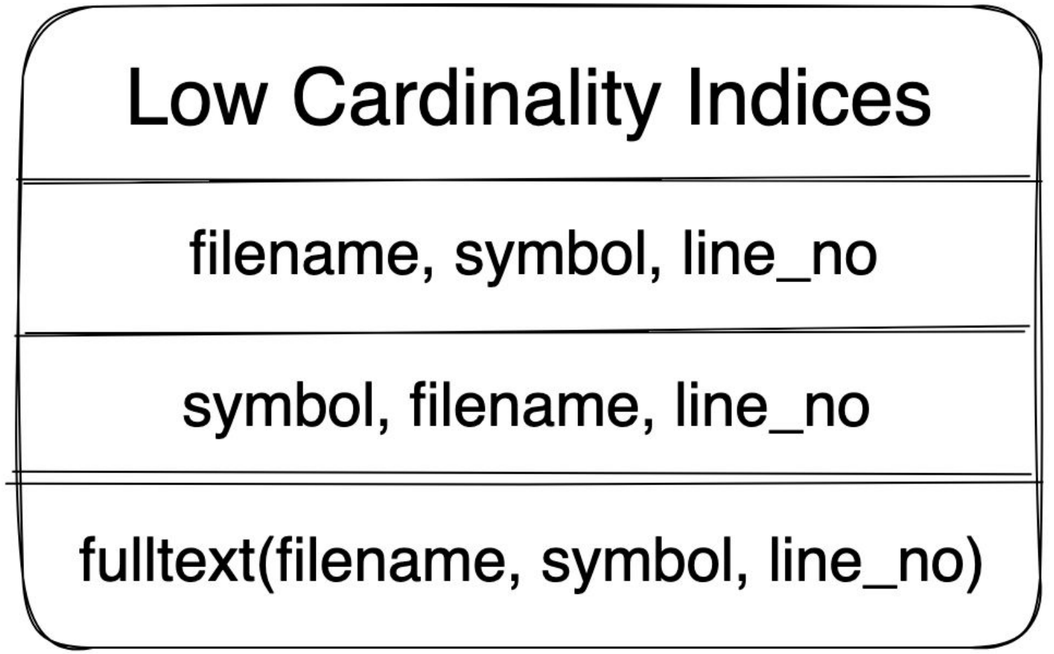
Conversion Process

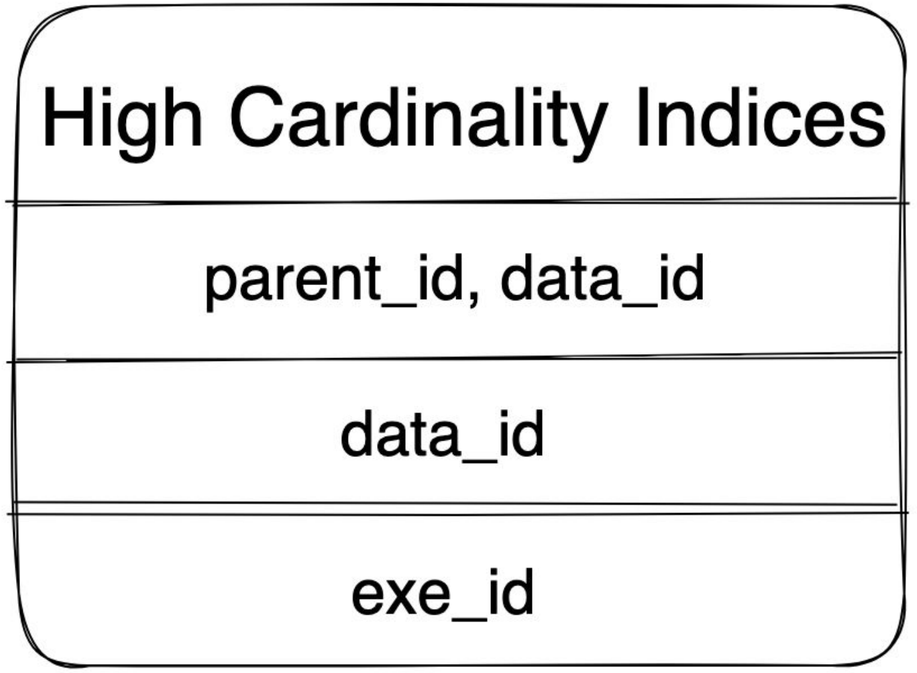
Symbol	Filename	Line Number	Executable ID
_start	../sysdeps/x86_64/start.S	117	36413
__libc_start_main@GLIBC_2.25	../csu/libc-start.c	368	36413
__libc_start_call_main	../sysdeps/nptl/libc_start_call_main.h	58	36413
main	/path/to/demo.c	22	36413
applicationLogic	/path/to/demo.c	18	36413
doLogging	/path/to/demo.c	9	36413
simulateWork	/path/to/demo.c	5	36413



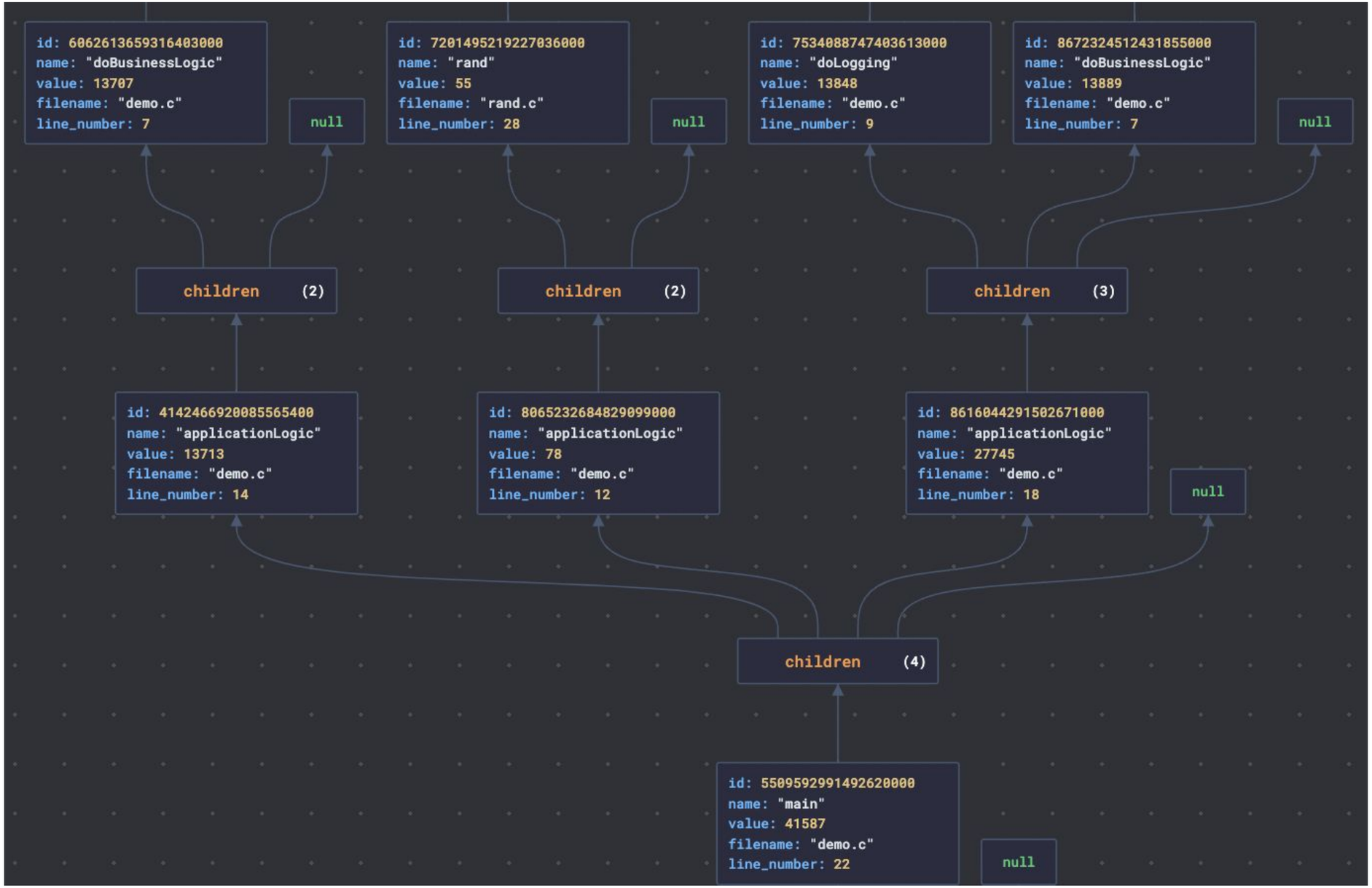
How we made this data queryable.

```
select subtree( rootid: stack_node.id)
from stack_node
         inner join executable on stack_node.executable_id = executable.id
where executable.basename = 'demo'
  and executable.build_id = 'one'
  and stack_node.parent_id is null;
```



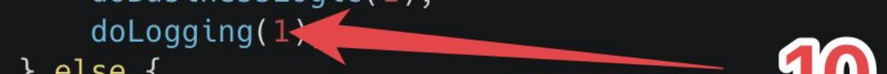


Recursive Queries Without Recursive Difficulty



Let’s Find a Regression

```c
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>

void simulateWork(int x) { for(int i = 10000*x; i>0; i--); }

void doBusinessLogic(int x) { simulateWork(x); }

void doLogging(int x) { simulateWork(x); }

void applicationLogic(int logPct) {
    if(rand() % 100 < logPct) {
        doBusinessLogic(1);
        doLogging(1);
    } else {
        doBusinessLogic(1);
    }
}

int main() {
    printf("%ld\n", (long)getpid());
    while(1){ applicationLogic(50); }
}
```




```
desktop /home/pat/repos/sto # target/release/cli --pid 15040 --binary demo --version two
```

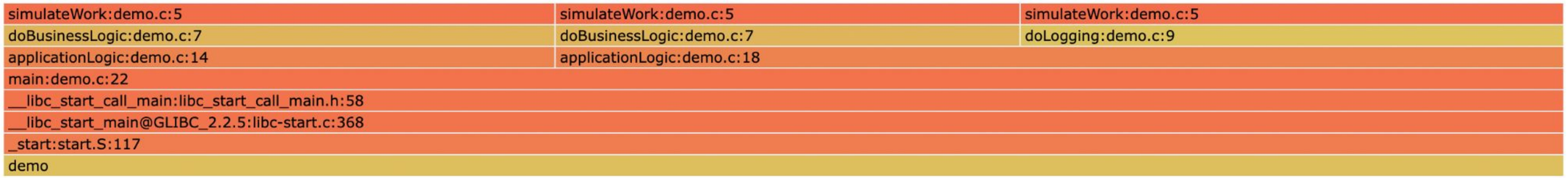
Found it, Generically

```
select * from findRegressions( start_time: CURRENT_DATE-1);
```

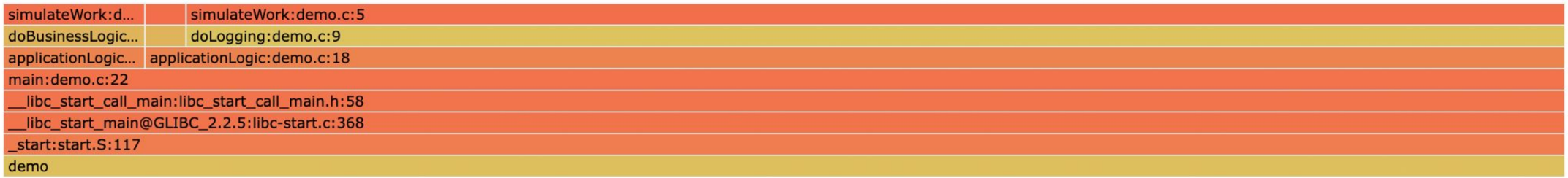
	basename	a.build_id	b.build_id	file	symbol	pct_diff
1	demo	one	two	demo.c	doLogging	153.485987696514012168
2	demo	one	two	demo.c	applicationLogic	40.690019701119600207
3	demo	one	two	demo.c	simulateWork	0.0072469019494166614999

What did that regression look like?

id	Name	Version	Samples	Raw Data Size	Sto Data Size	Storage Size Reduction
2619496695096638500	demo	one	1228800	34.64 MB	95.21 KB	373x



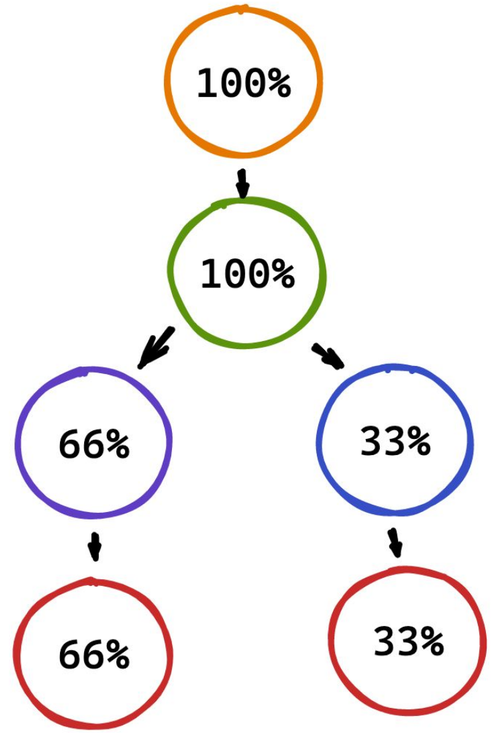
id	Name	Version	Samples	Raw Data Size	Sto Data Size	Storage Size Reduction
5600443421022226000	demo	two	1894400	53.37 MB	129.30 KB	423x



Conclusion

Call graphs are graphs. Sto makes them more easily storable and queryable.

```
void simulateWork() { ... }

void doBusinessLogic() { simulateWork(); }

void doLogging() { simulateWork(); }

void applicationLogic() {
    if( ... 50%) {
        doBusinessLogic();
        doLogging();
    } else {
        doBusinessLogic();
    }
}

int main() { applicationLogic(); }
```



Thank You!

All code in this talk is available to
play with via one-click deploy at
https://github.com/likewhatevs/sto
Please take a look!!!

Patrick Somaru
Production Engineer

Meta