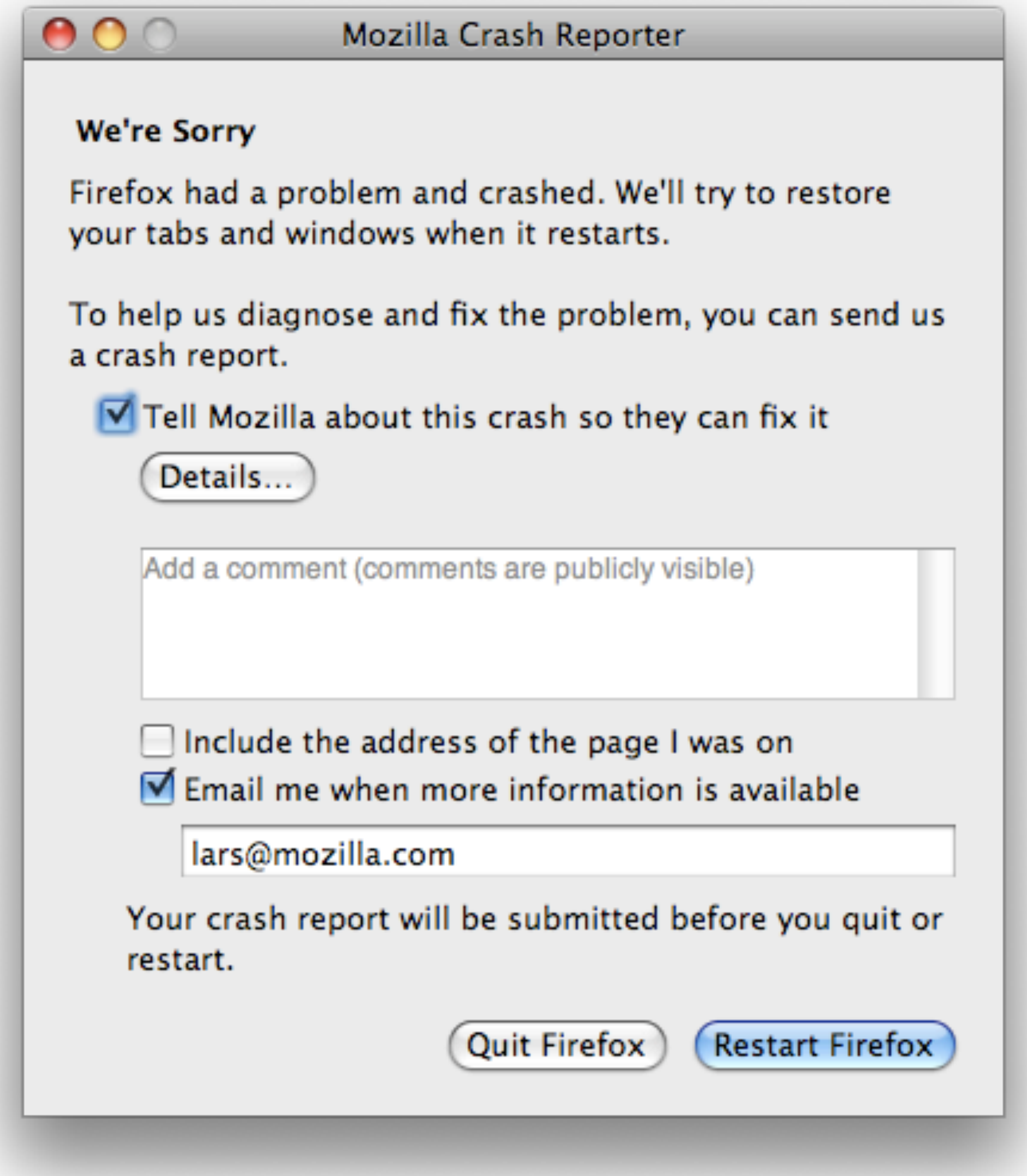
Crash Reporting:
Mozilla's Open Source Solution

K Lars Lohn
Ted Mielczarek
Austin King

Friday, July 24, 2009

Howdy, I'm Lars from Mozilla, I'm here today with my colleagues, Ted Mill-char-ek and Austin King to talk about Crash Reporting.

What is it?



Mozilla Crash Reporter

We're Sorry

Firefox had a problem and crashed. We'll try to restore your tabs and windows when it restarts.

To help us diagnose and fix the problem, you can send us a crash report.

☑ Tell Mozilla about this crash so they can fix it

Details...

Add a comment (comments are publicly visible)

□ Include the address of the page I was on

☑ Email me when more information is available

lars@mozilla.com

Your crash report will be submitted before you quit or restart.

Quit Firefox   Restart Firefox

Have you ever seen Firefox crash? Have you ever wondered what happens after you hit the “restart firefox” button?

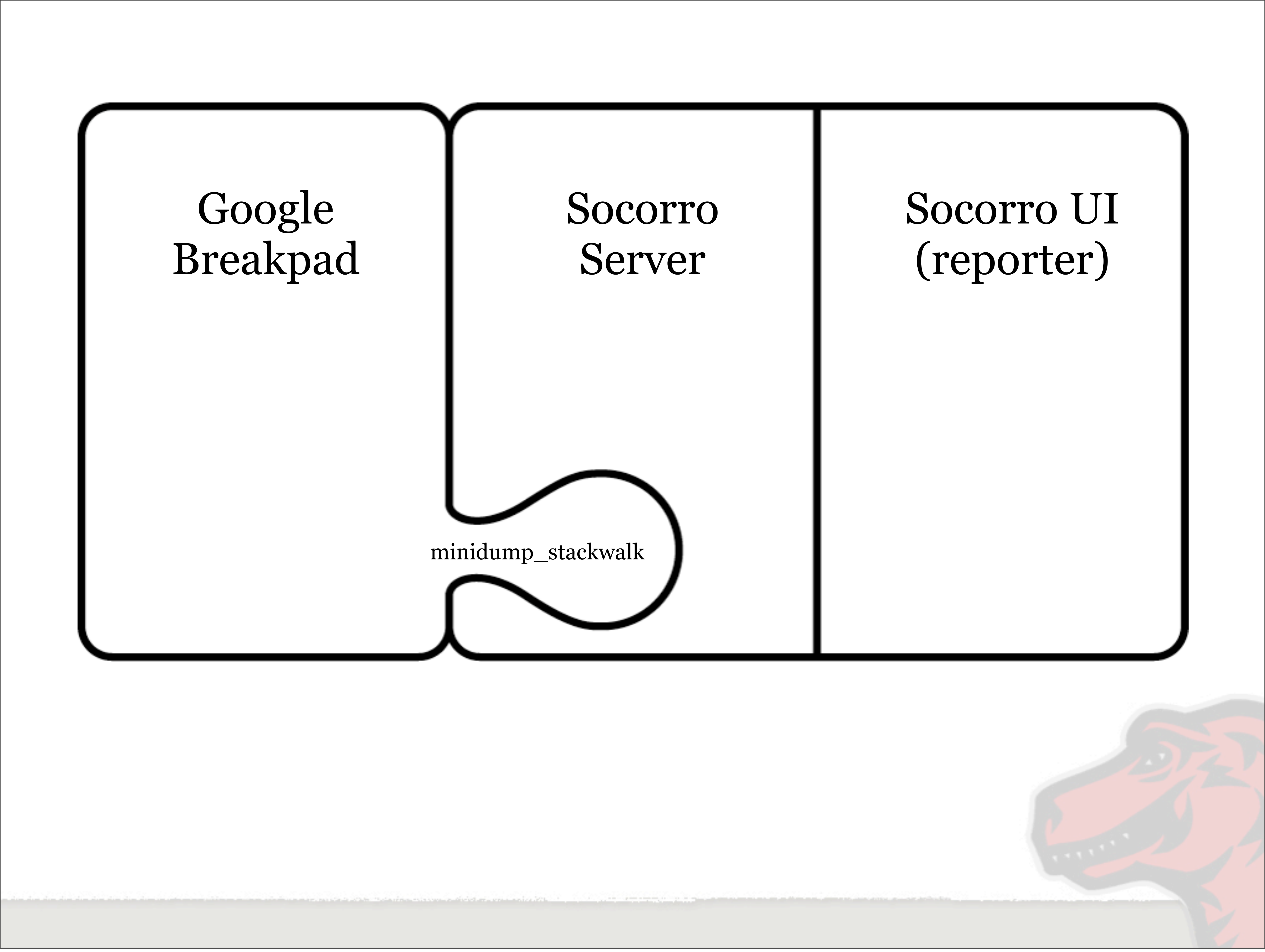
In the next 45 minutes, we’re going to take you down the rabbit hole and show you what we do. We want our processes to be open.

When you experience a problem with any Mozilla product, we want _everyone_, not just the developers, to watch the flow of information about a problem for its initial occurrence, through data collection and triage, on to Bugzilla and to an eventual resolution.

Crash reporting
is the transmission of
information
to the developers
about the state of an application
during a catastrophic failure.

Crash reporting is the transmission of information about the state of an application during a catastrophic failure to the developers.

The goal is to give the developers information that they would not otherwise have had.



Google Breakpad

Socorro Server

Socorro UI (reporter)

minidump_stackwalk

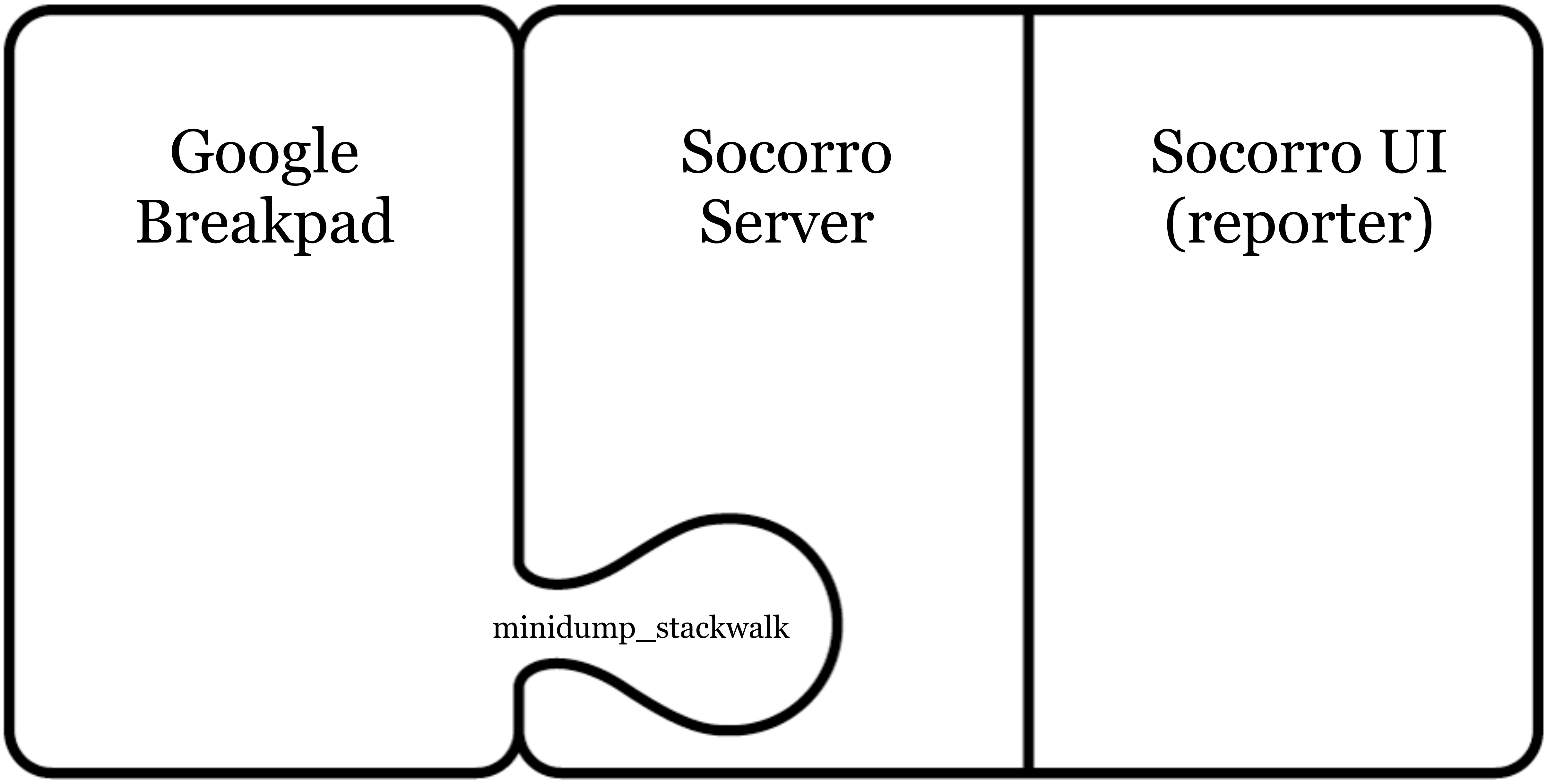
Our crash reporting system can be divided into three parts.

Breakpad - a google project - this code lives mainly within the Firefox application

Socorro (in two movements) the backend server and the user interface running at Mozilla.

written in three languages <click> <click> <click>

Ted, Austin and I are the three developers in charge of the three sections. We’re going to talk in turn about our sections.



Google Breakpad

Socorro Server

Socorro UI (reporter)

minidump_stackwalk

C++

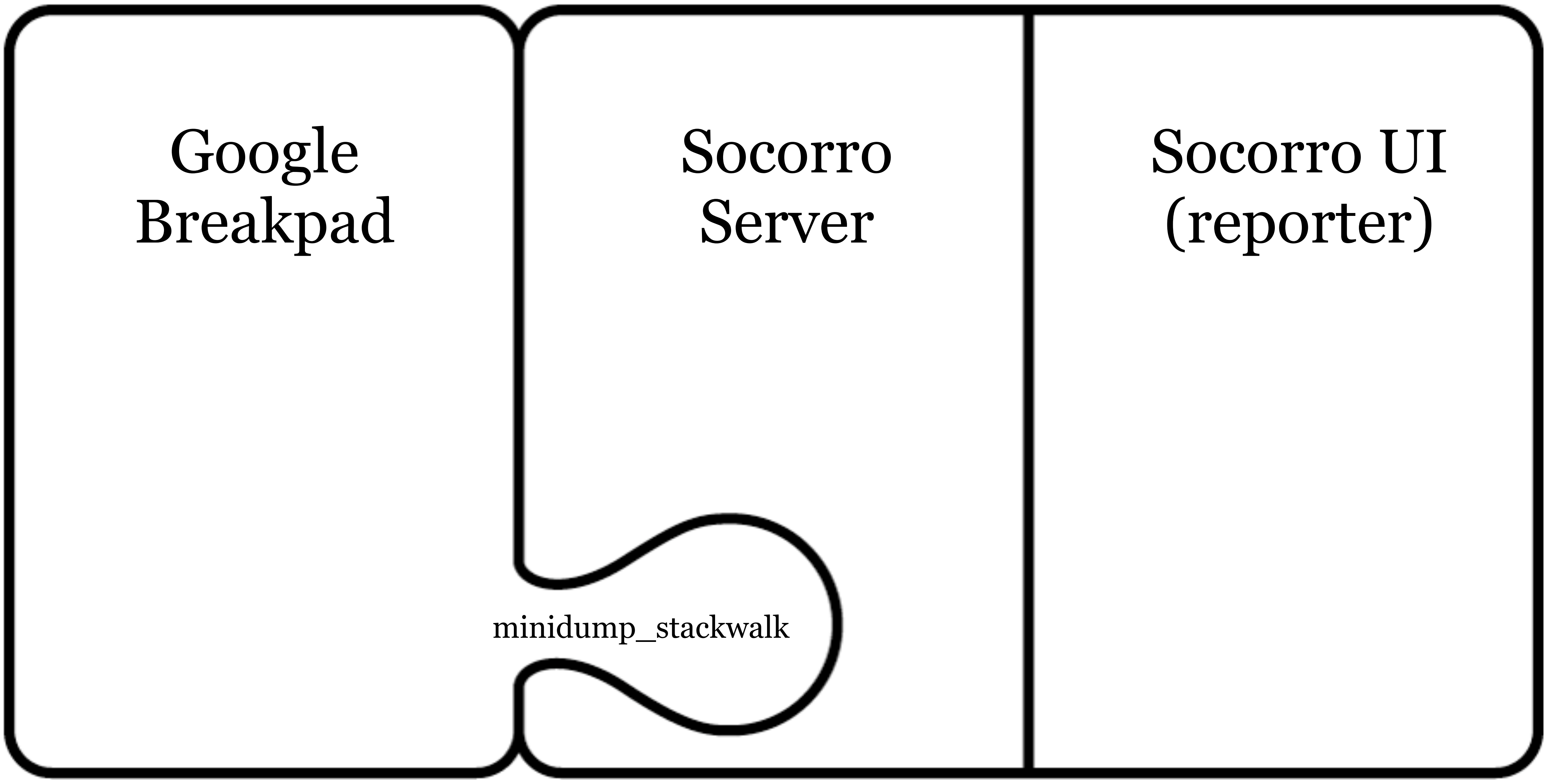
Our crash reporting system can be divided into three parts.

Breakpad - a google project - this code lives mainly within the Firefox application

Socorro (in two movements) the backend server and the user interface running at Mozilla.

written in three languages <click> <click> <click>

Ted, Austin and I are the three developers in charge of the three sections. We’re going to talk in turn about our sections.



Google Breakpad

Socorro Server

Socorro UI (reporter)

minidump_stackwalk

C++

Python

Friday, July 24, 2009

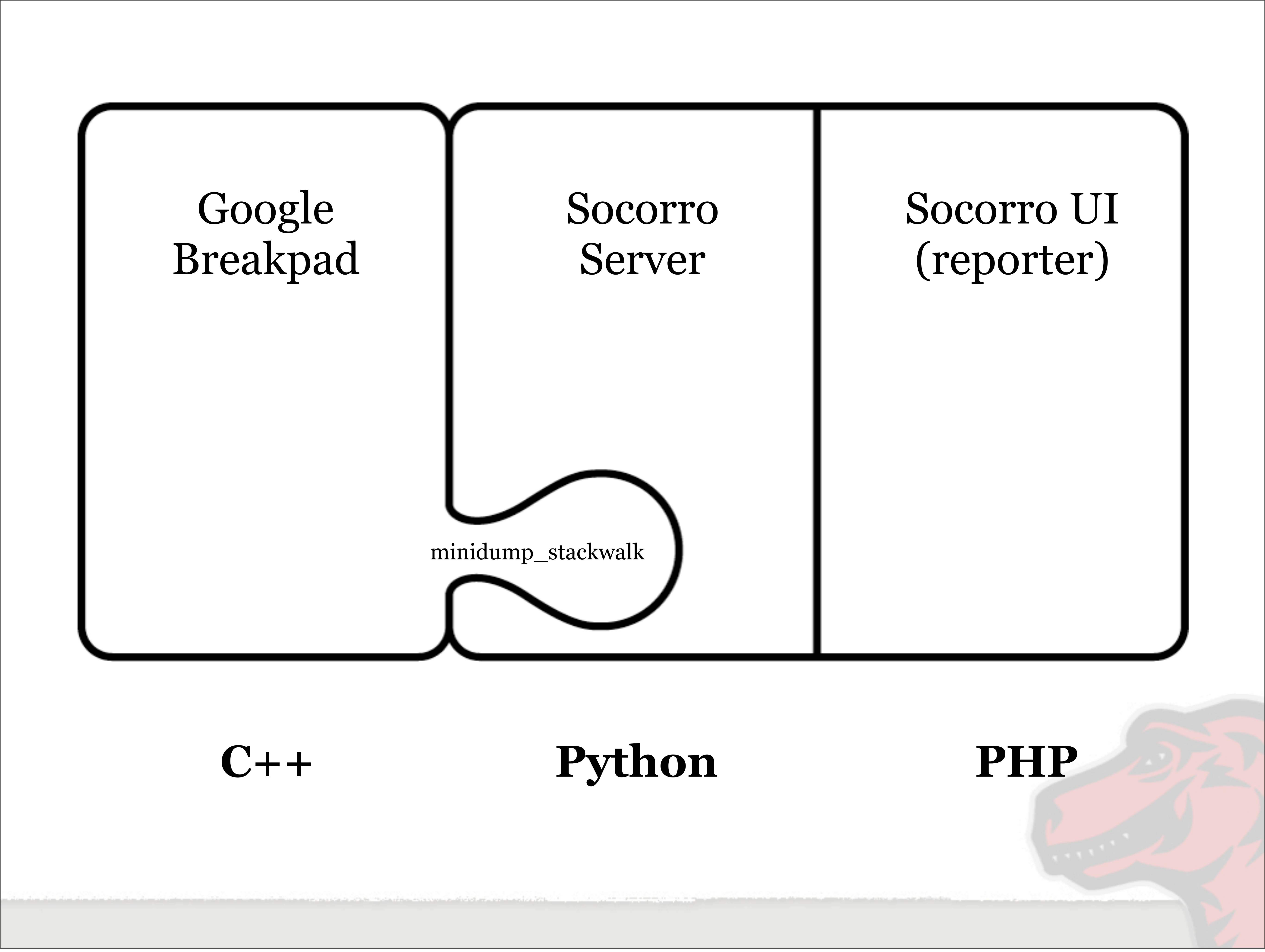
Our crash reporting system can be divided into three parts.

Breakpad - a google project - this code lives mainly within the Firefox application

Socorro (in two movements) the backend server and the user interface running at Mozilla.

written in three languages <click> <click> <click>

Ted, Austin and I are the three developers in charge of the three sections. We’re going to talk in turn about our sections.



Google Breakpad

Socorro Server

Socorro UI (reporter)

minidump_stackwalk

C++

Python

PHP

Friday, July 24, 2009

Our crash reporting system can be divided into three parts.

Breakpad – a google project – this code lives mainly within the Firefox application

Socorro (in two movements) the backend server and the user interface running at Mozilla.

written in three languages <click> <click> <click>

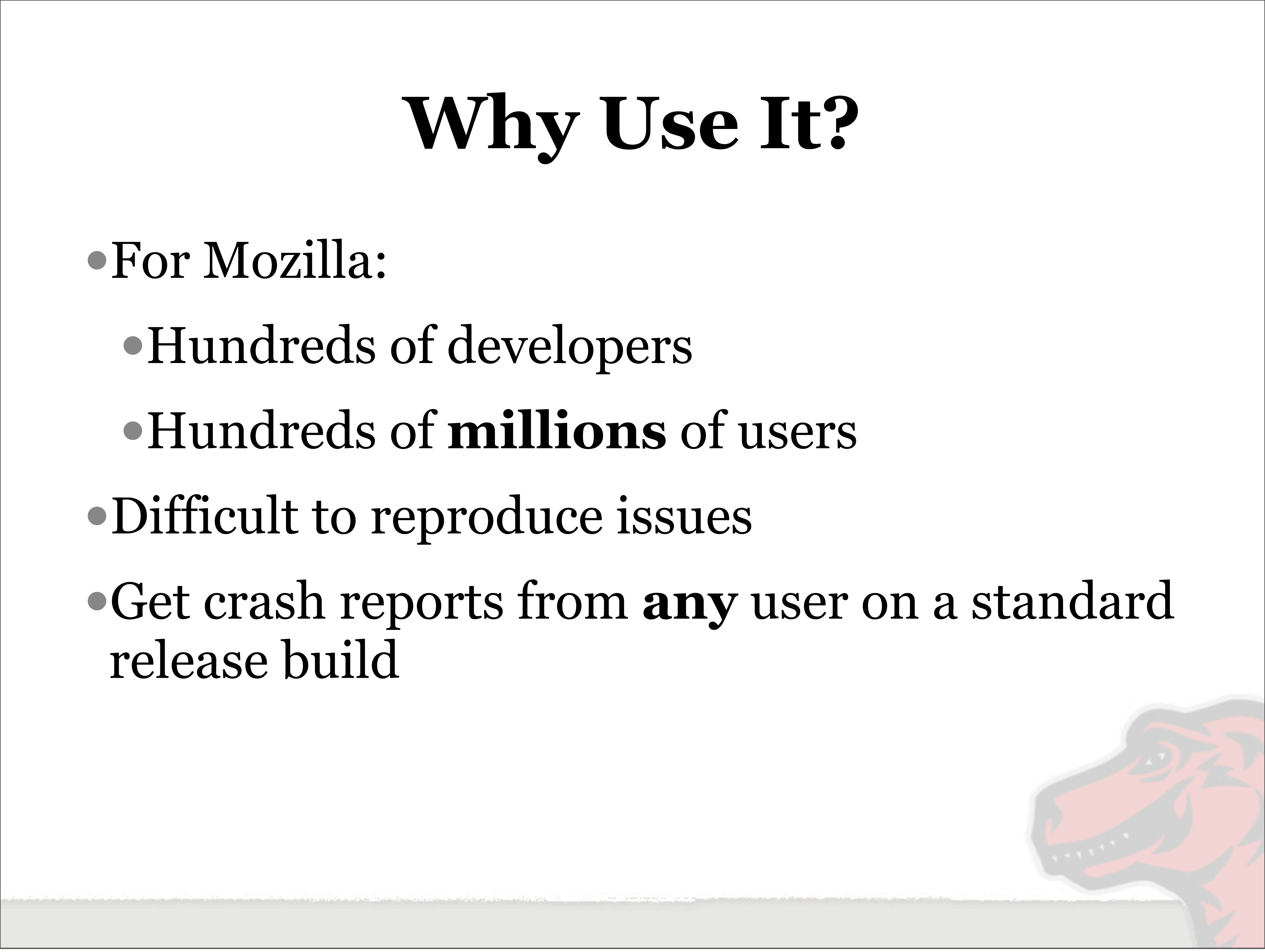
Ted, Austin and I are the three developers in charge of the three sections. We’re going to talk in turn about our sections.

Google Breakpad

Google Breakpad

- Open Source project started by Google
  - BSD licensed: http://code.google.com/p/google-breakpad/
- Client-side support for Linux (x86), Windows (x86), OS X (x86/PPC), Solaris (x86/SPARC)
- Server-side runs on POSIX systems

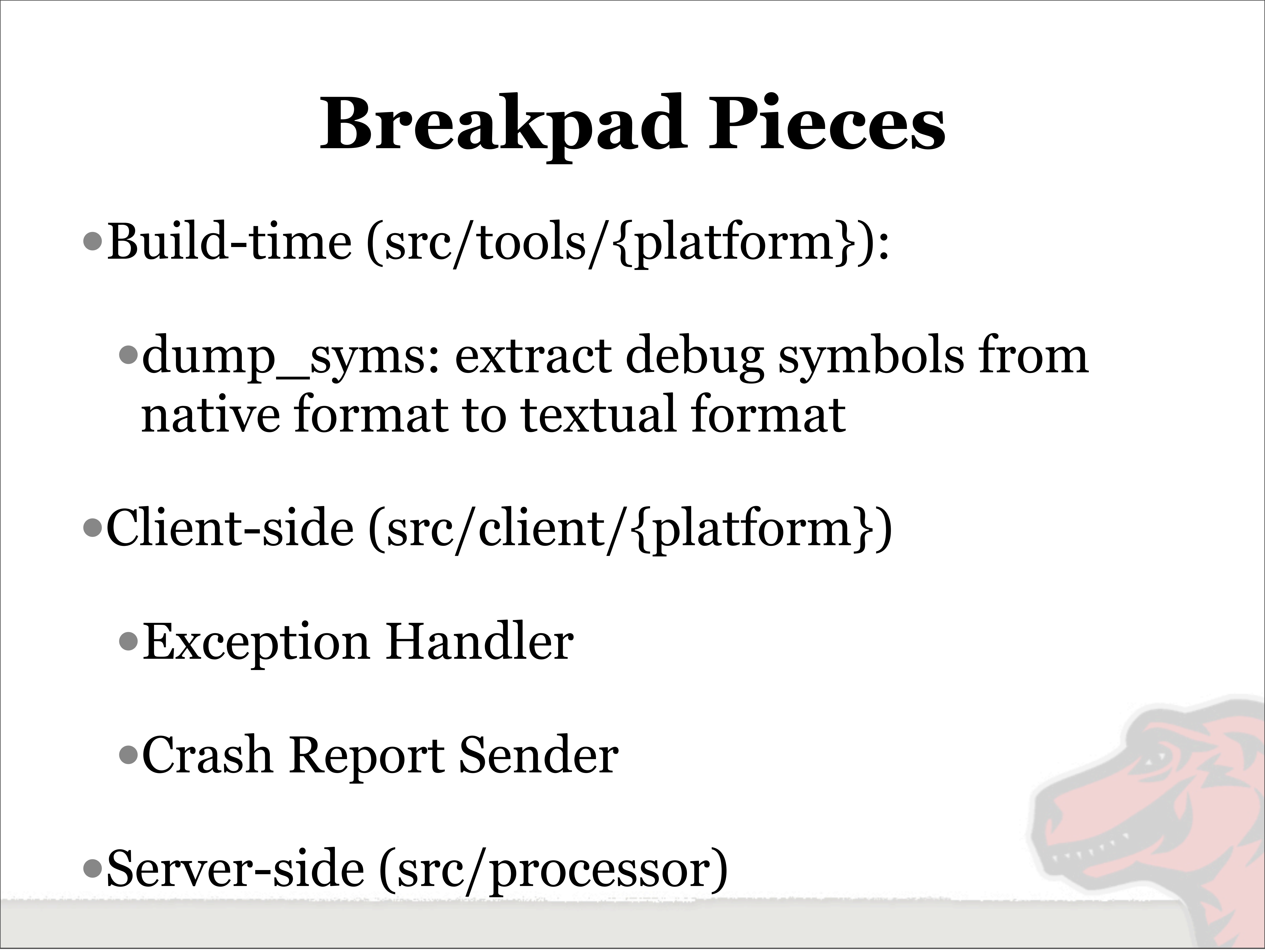
Used by a few applications you might know: Firefox, Chrome, Google Earth
Supports a number of platforms as clients, the server side runs on POSIX systems.



Why Use It?

- For Mozilla:
  - Hundreds of developers
  - Hundreds of **millions** of users
- Difficult to reproduce issues
- Get crash reports from **any** user on a standard release build

For us, it gives us these benefits:
The number of users is so much greater than the number of developers, so that larger group will always encounter problems that developers never see. You may have timing sensitive bugs or bugs that only occur when certain third party software is installed.
For Mozilla, the sheer number of webpages on the internet guarantees that users will hit unique situations every day.
In addition, every user becomes a source of information about crashes.

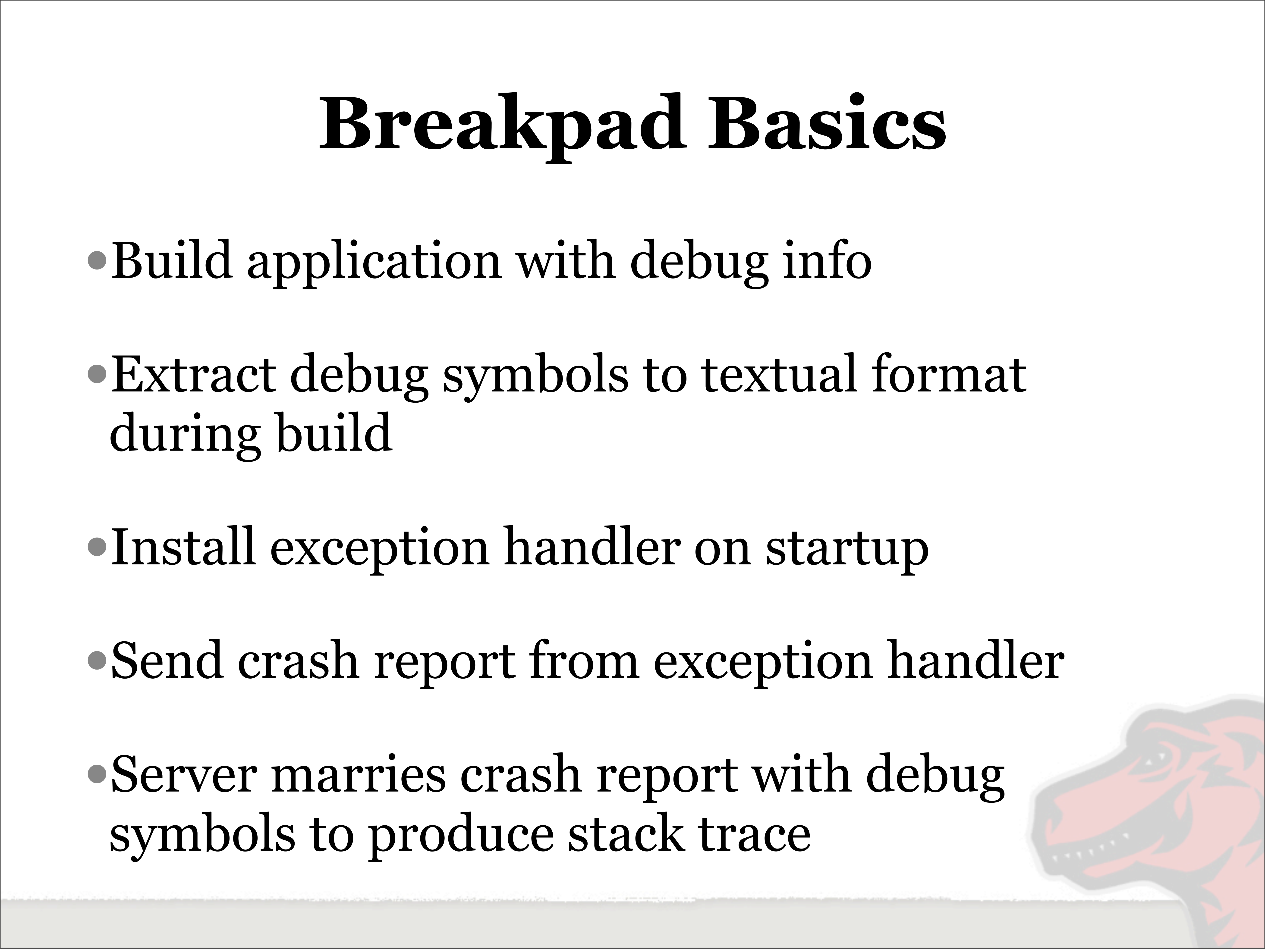


Breakpad Pieces

- Build-time (src/tools/{platform}):
  - dump_syms: extract debug symbols from native format to textual format
- Client-side (src/client/{platform})
  - Exception Handler
  - Crash Report Sender
- Server-side (src/processor)

Friday, July 24, 2009

Breakpad consists of three separate sets of code:
* build time tool to extract debug symbols to a common textual format
* two client side libraries:
  – “Exception Handler” – catching crashes and doing something about it
  – “Crash Report Sender” – sending the results to a server for handling
* server side libraries and tools for turning binary reports into useful data



Breakpad Basics

- Build application with debug info
- Extract debug symbols to textual format during build
- Install exception handler on startup
- Send crash report from exception handler
- Server marries crash report with debug symbols to produce stack trace

Using Breakpad in your application can be broken down along those lines:
Steps you take when you build your application for distribution to users (build, extract debug info)
Steps your application takes to catch and submit crashes (handler, sender)
Steps the server takes to get the useful data out of the report (processor)

Exception Handler

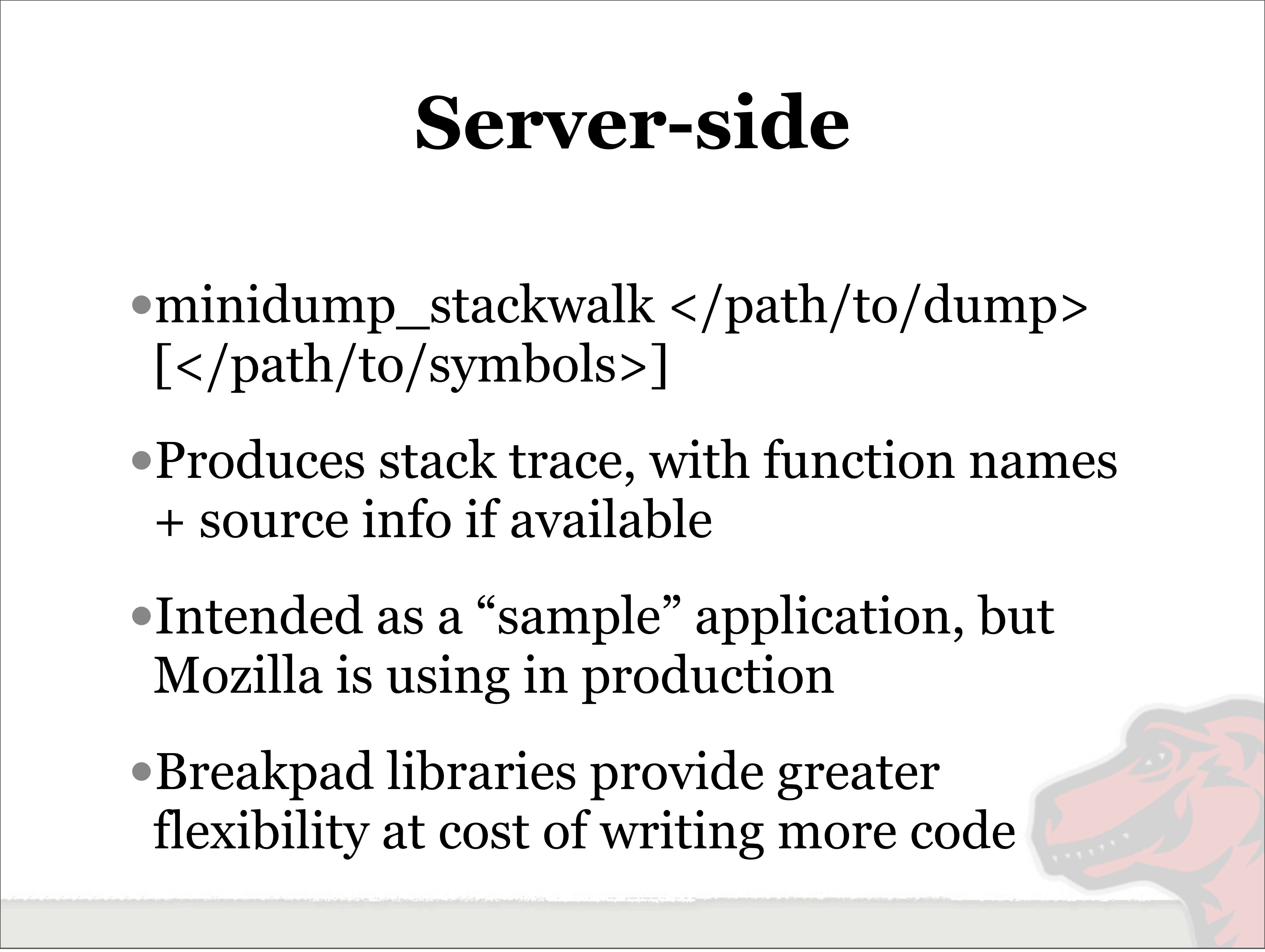
- Create ExceptionHandler object with callback
- On crash, Breakpad writes crash data and calls your callback
- Your callback does something useful

The exception handler is the key part that lets you handle crashes in your application. You provide a callback function, and when the application crashes Breakpad writes out information about the state of the application to disk, then calls your callback with the path to that data file (“minidump”). You can then do something useful with it (although probably in a separate process, since this one has crashed!) In Firefox, we spawn a separate crash reporter process.

Crash report sender

- Varies per-OS, but boils down to:
  - Send(URL, parameters, dumpfile)
  - Sends via HTTP POST
- Not included for OS X (easy to do with Cocoa)

Once you have a crash report, you need to send it to your server for processing. Breakpad provides support for sending via HTTP POST. The minidump is sent as a file upload, other params can be sent as form data.



Server-side

- minidump_stackwalk </path/to/dump> [</path/to/symbols>]
- Produces stack trace, with function names + source info if available
- Intended as a “sample” application, but Mozilla is using in production
- Breakpad libraries provide greater flexibility at cost of writing more code

The final piece of Breakpad is the processor, which can take a crash report along with the symbols from the build and produce a stack trace. The command line tool that does this is called minidump_stackwalk, it simply takes the crash report and symbol path on the command line. You can go further by using the processor code as a library, but you’ll need to write some C++ glue code.

Socorro Server

Socorro Server

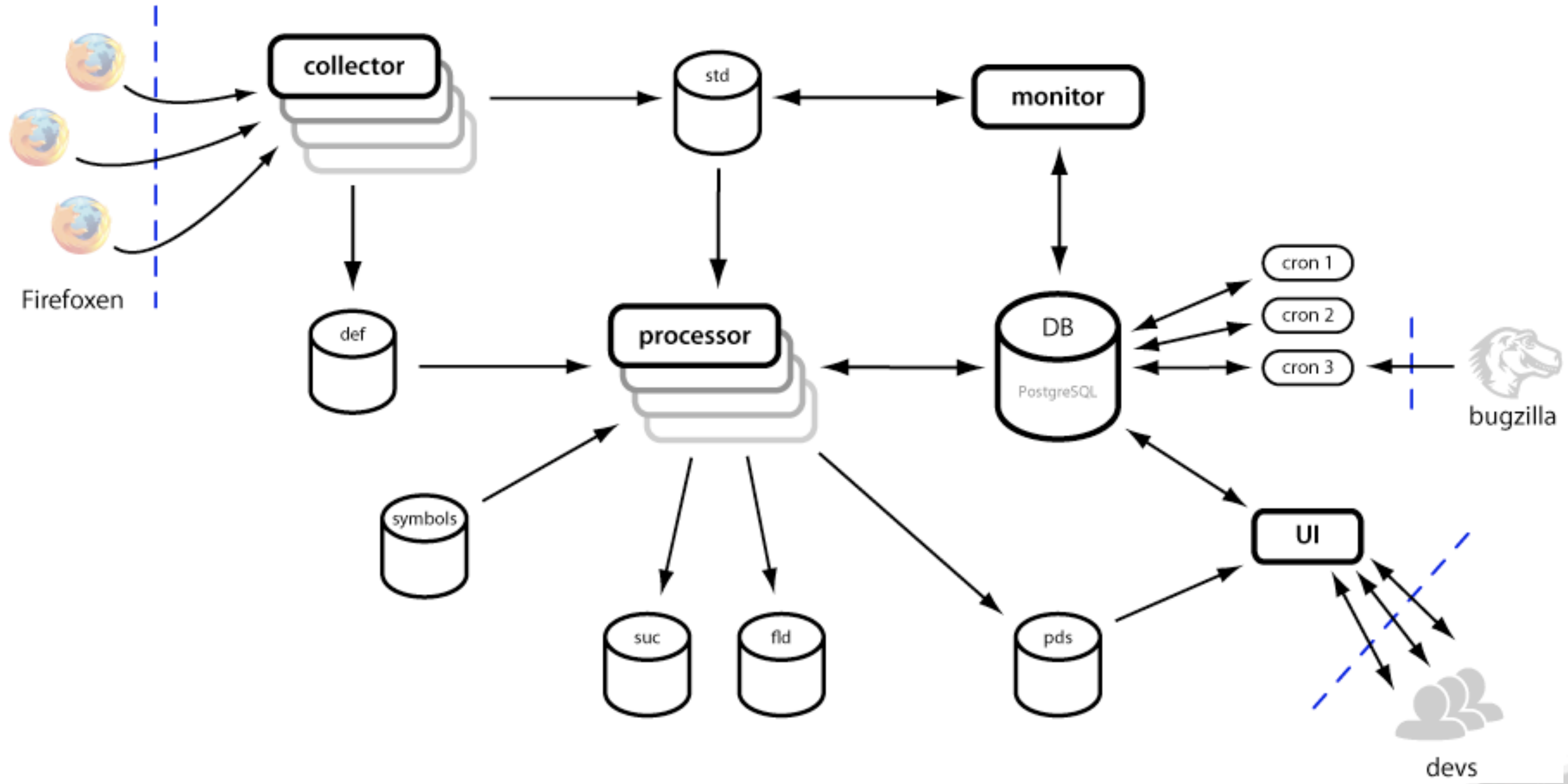
- Collector
- Monitor
- Processor
- Data Aggregators as cron jobs
- Data Cleanup as cron jobs

Friday, July 24, 2009

The Socorro Server consists of several long running processes: the Collectors, the Monitor and the Processors.

There are also several processes that run as periodic cron jobs.

These cron jobs are in charge of generating aggregate information for reports as well as periodic system maintenance of file system storage.



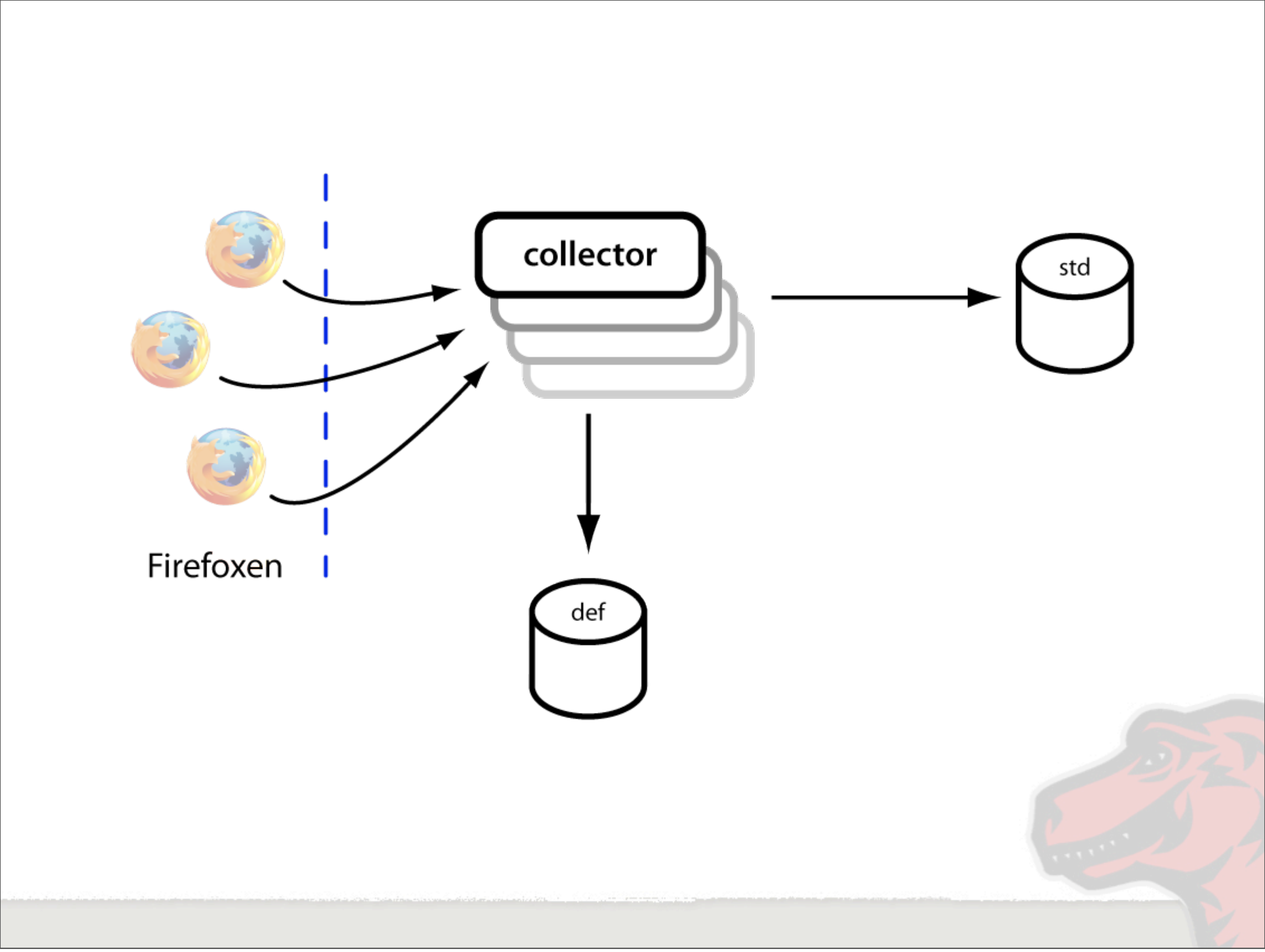
Friday, July 24, 2009

This is an overview of the data flow whole system. It shows the long running processes as well as the aggregate and clean up jobs, just indicated as cron 1 through 3. There are actually more of them, but this is an overview, so we're going to gloss over some details.

You'll notice here that we've got lots of different data storage areas. Off the the right of center, we've got our instance of PostgreSQL. This is the heart of the system: it stores data about crashes as well as serves as a queuing system to coordinate the timing of processes.

We're going to look at the data flow – how crash information actually moves through the system

First, the collector.



Friday, July 24, 2009

First we’ll focus on Collector.

This is a python script running under Apache using mod-python. When Firefox crashes, in a last ditch effort before it quits, it sends off an http post of crash information to these Collectors. There can be any number of them, load balanced out front with whatever suits your fancy. We use Netscaler.

<CLICK>

When collector receives a crash, it examines the meta information about the crash: the product, the version, etc. At this point it can make some snap decisions in a process called “throttling” on whether to pass the crash on for further processing, refuse it or dump it into a deferred storage for later use.

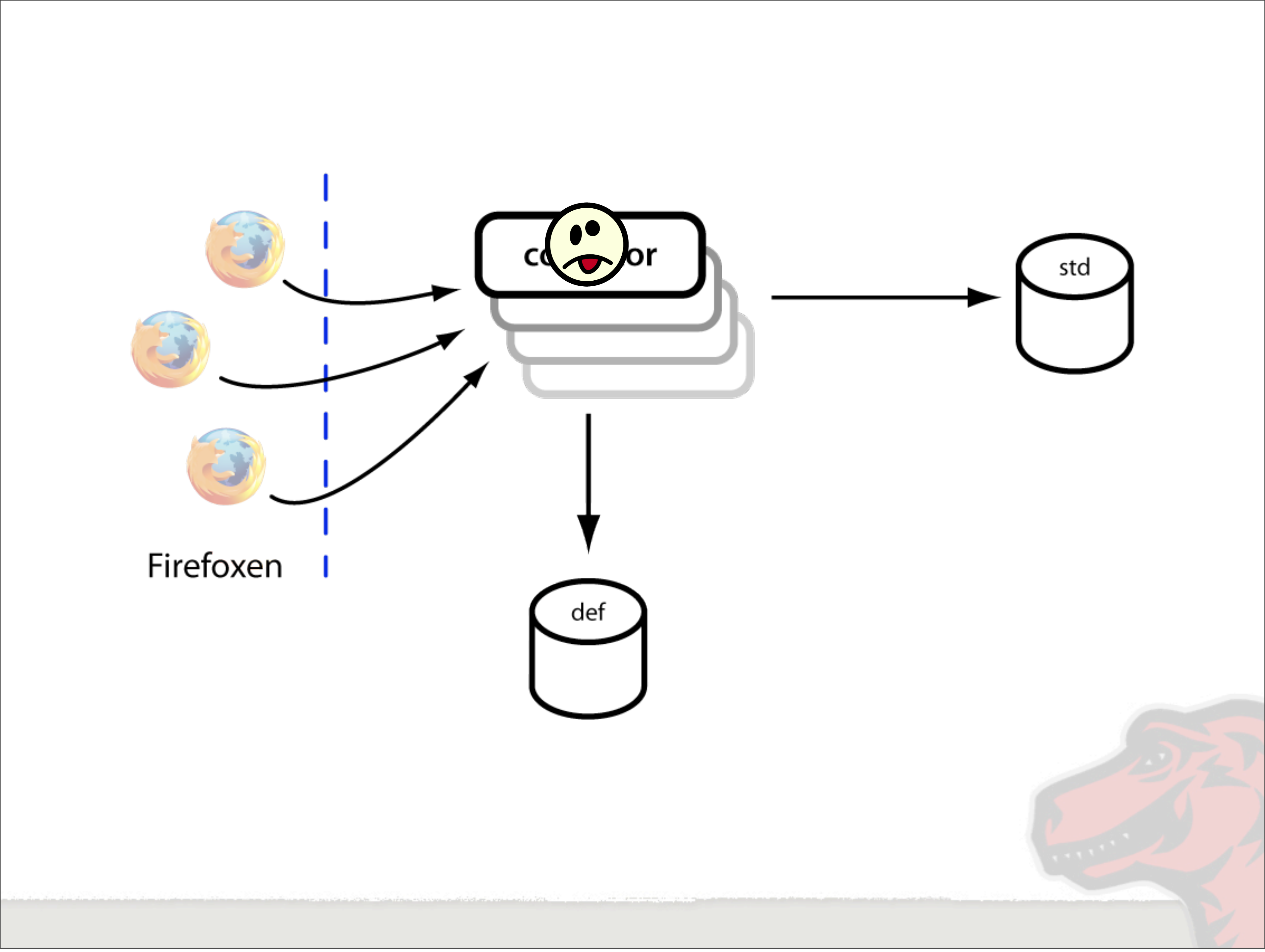
Statistically, we don’t have to collect every crash. In fact, we don’t have that much disk space.

<CLICK>

<CLICK>

<CLICK>

When we finally do accept a crash (and we accept only about 10%), it get assigned a 32 character uuid and dropped into “standard” storage.



Friday, July 24, 2009

First we'll focus on Collector.

This is a python script running under Apache using mod-python. When Firefox crashes, in a last ditch effort before it quits, it sends off an http post of crash information to these Collectors. There can be any number of them, load balanced out front with whatever suits your fancy. We use Netscaler.

<CLICK>

When collector receives a crash, it examines the meta information about the crash: the product, the version, etc. At this point it can make some snap decisions in a process called "throttling" on whether to pass the crash on for further processing, refuse it or dump it into a deferred storage for later use.

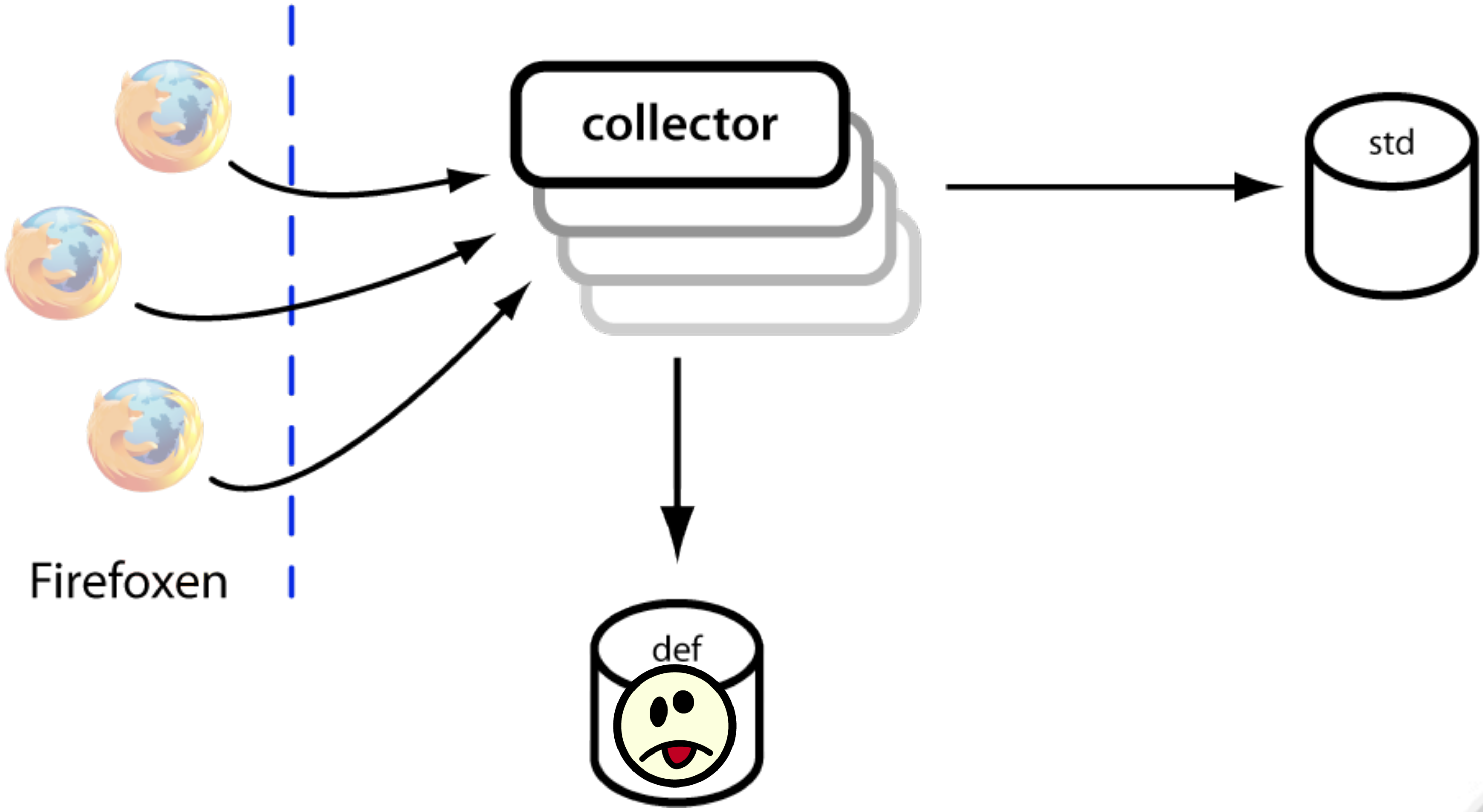
Statistically, we don't have to collect every crash. In fact, we don't have that much disk space.

<CLICK>

<CLICK>

<CLICK>

When we finally do accept a crash (and we accept only about 10%), it get assigned a 32 character uuid and dropped into "standard" storage.



Friday, July 24, 2009

First we’ll focus on Collector.

This is a python script running under Apache using mod-python. When Firefox crashes, in a last ditch effort before it quits, it sends off an http post of crash information to these Collectors. There can be any number of them, load balanced out front with whatever suits your fancy. We use Netscaler.

<CLICK>

When collector receives a crash, it examines the meta information about the crash: the product, the version, etc. At this point it can make some snap decisions in a process called “throttling” on whether to pass the crash on for further processing, refuse it or dump it into a deferred storage for later use.

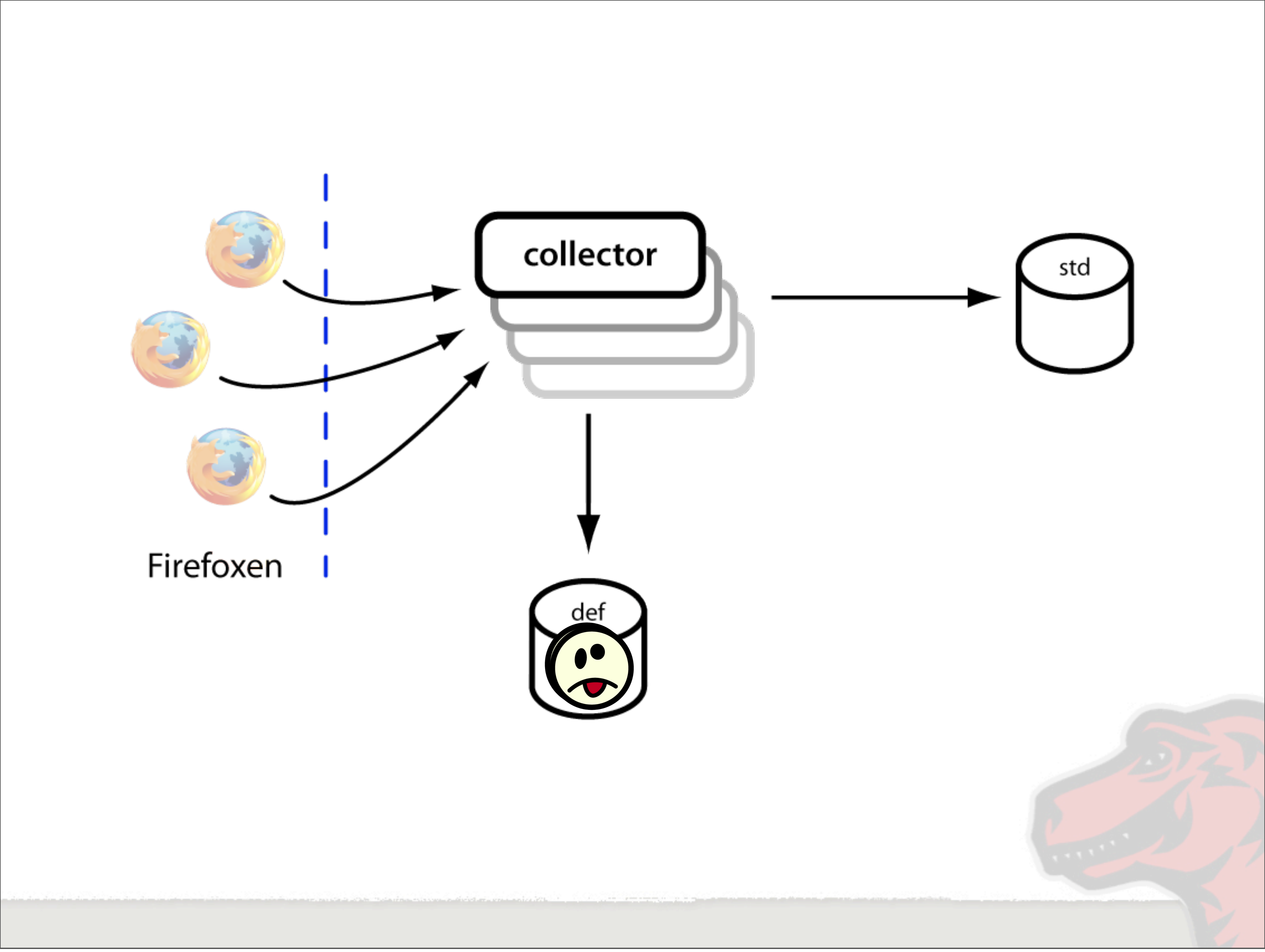
Statistically, we don’t have to collect every crash. In fact, we don’t have that much disk space.

<CLICK>

<CLICK>

<CLICK>

When we finally do accept a crash (and we accept only about 10%), it get assigned a 32 character uuid and dropped into “standard” storage.



Friday, July 24, 2009

First we'll focus on Collector.

This is a python script running under Apache using mod-python. When Firefox crashes, in a last ditch effort before it quits, it sends off an http post of crash information to these Collectors. There can be any number of them, load balanced out front with whatever suits your fancy. We use Netscaler.

<CLICK>

When collector receives a crash, it examines the meta information about the crash: the product, the version, etc. At this point it can make some snap decisions in a process called "throttling" on whether to pass the crash on for further processing, refuse it or dump it into a deferred storage for later use.

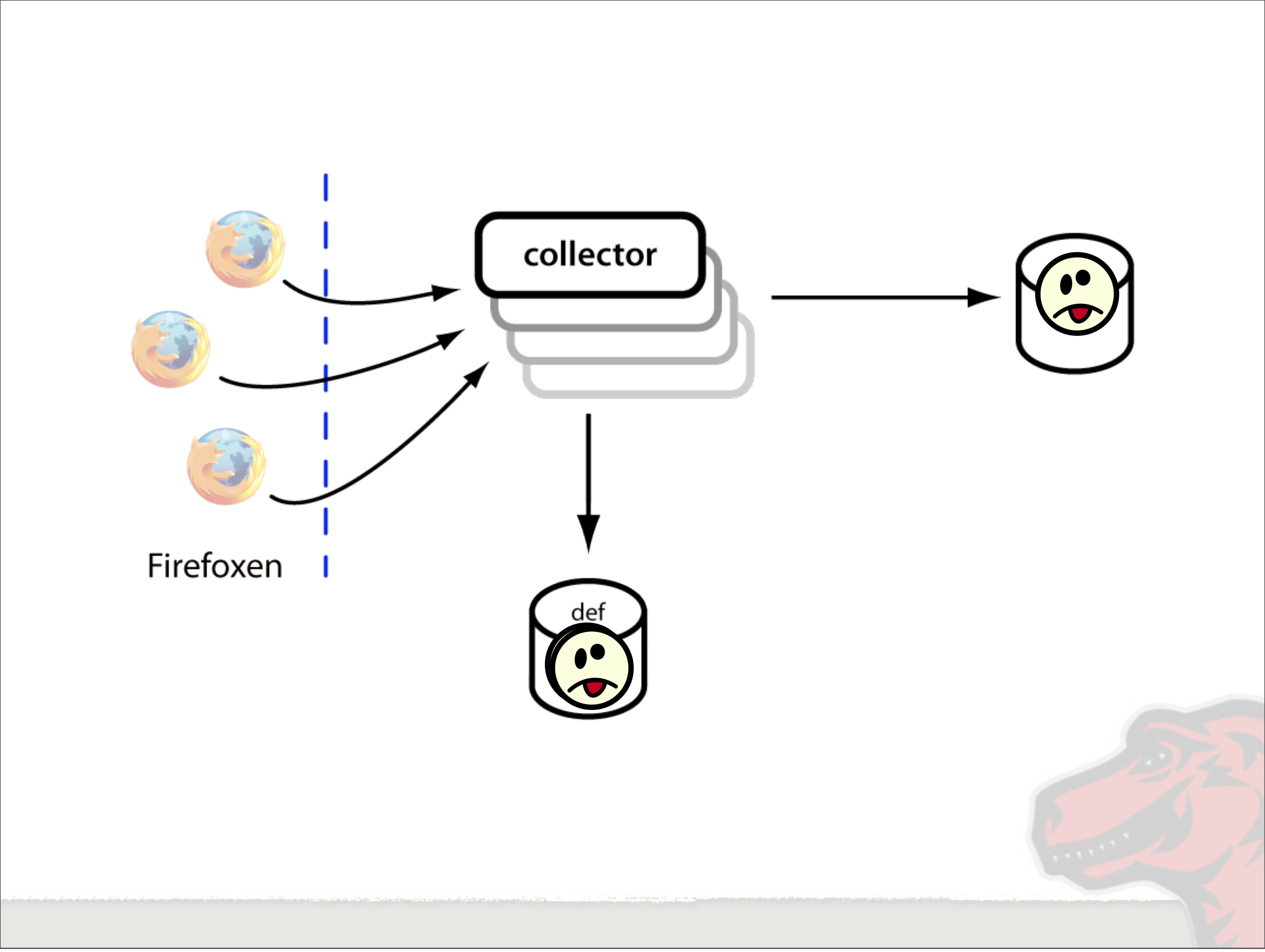
Statistically, we don't have to collect every crash. In fact, we don't have that much disk space.

<CLICK>

<CLICK>

<CLICK>

When we finally do accept a crash (and we accept only about 10%), it get assigned a 32 character uuid and dropped into "standard" storage.



Friday, July 24, 2009

First we'll focus on Collector.

This is a python script running under Apache using mod-python. When Firefox crashes, in a last ditch effort before it quits, it sends off an http post of crash information to these Collectors. There can be any number of them, load balanced out front with whatever suits your fancy. We use Netscaler.

<CLICK>

When collector receives a crash, it examines the meta information about the crash: the product, the version, etc. At this point it can make some snap decisions in a process called "throttling" on whether to pass the crash on for further processing, refuse it or dump it into a deferred storage for later use.

Statistically, we don't have to collect every crash. In fact, we don't have that much disk space.

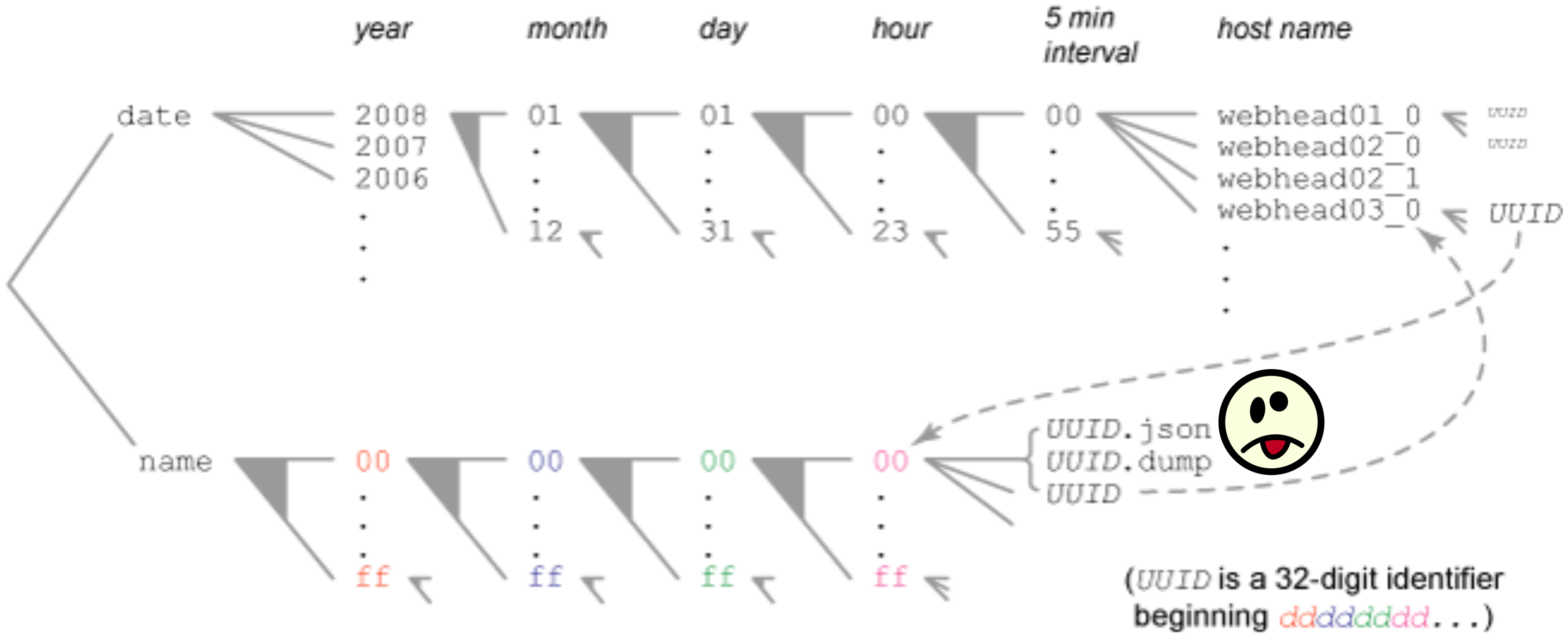
<CLICK>

<CLICK>

<CLICK>

When we finally do accept a crash (and we accept only about 10%), it get assigned a 32 character uuid and dropped into "standard" storage.

File System Structure



(UUID is a 32-digit identifier beginning dddddddd...)

Friday, July 24, 2009

We use file system storage like a hierarchical database. We want to be able to look up a crash with out having to spend any time searching. With 2 million of these things, you can't just dump them all into one directory.

We use a radix scheme to save crashes by name. Say we have a file called "aabbcc.json"

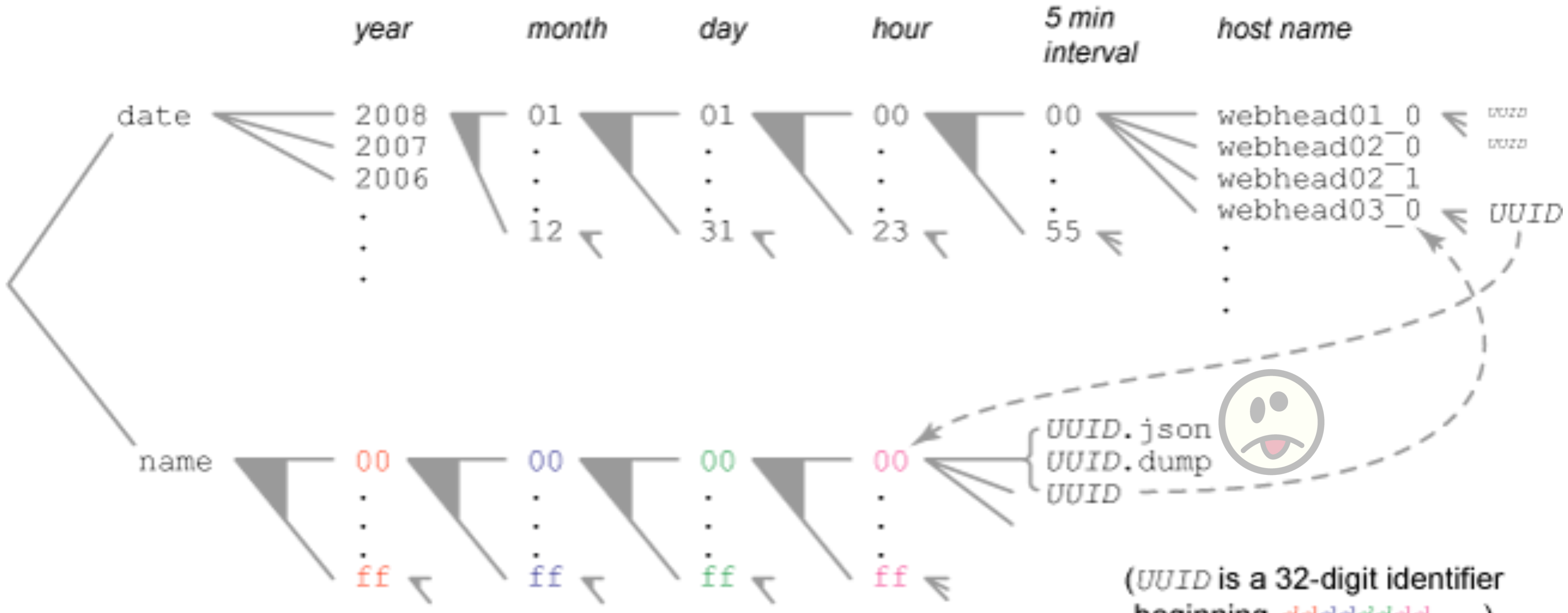
The "name" branch of this structure uses two characters at each directory level.

The "date" branch uses the same radix idea with datetimes. However, rather than having a data file at the leaf node, it has a symbolic link over to the where the data is stored in the name branch.

We can rapidly lookup crashes either by name or date, without wasting time having to search.

Back to our data flow...

File System Structure



(UUID is a 32-digit identifier beginning dddddddd...)

Friday, July 24, 2009

We use file system storage like a hierarchical database. We want to be able to look up a crash with out having to spend any time searching. With 2 million of these things, you can't just dump them all into one directory.

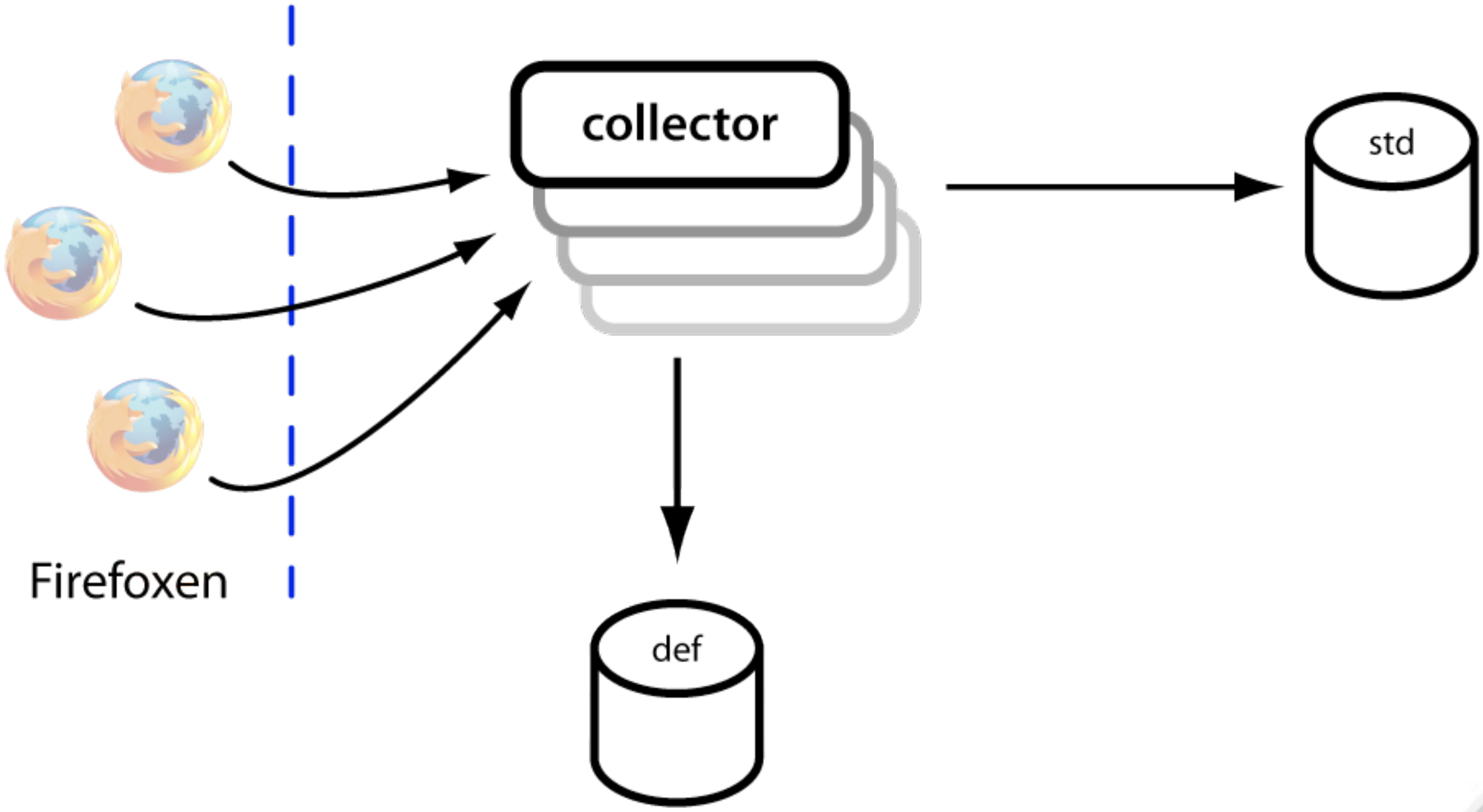
We use a radix scheme to save crashes by name. Say we have a file called "aabbcc.json"

The "name" branch of this structure uses two characters at each directory level.

The "date" branch uses the same radix idea with datetimes. However, rather than having a data file at the leaf node, it has a symbolic link over to the where the data is stored in the name branch.

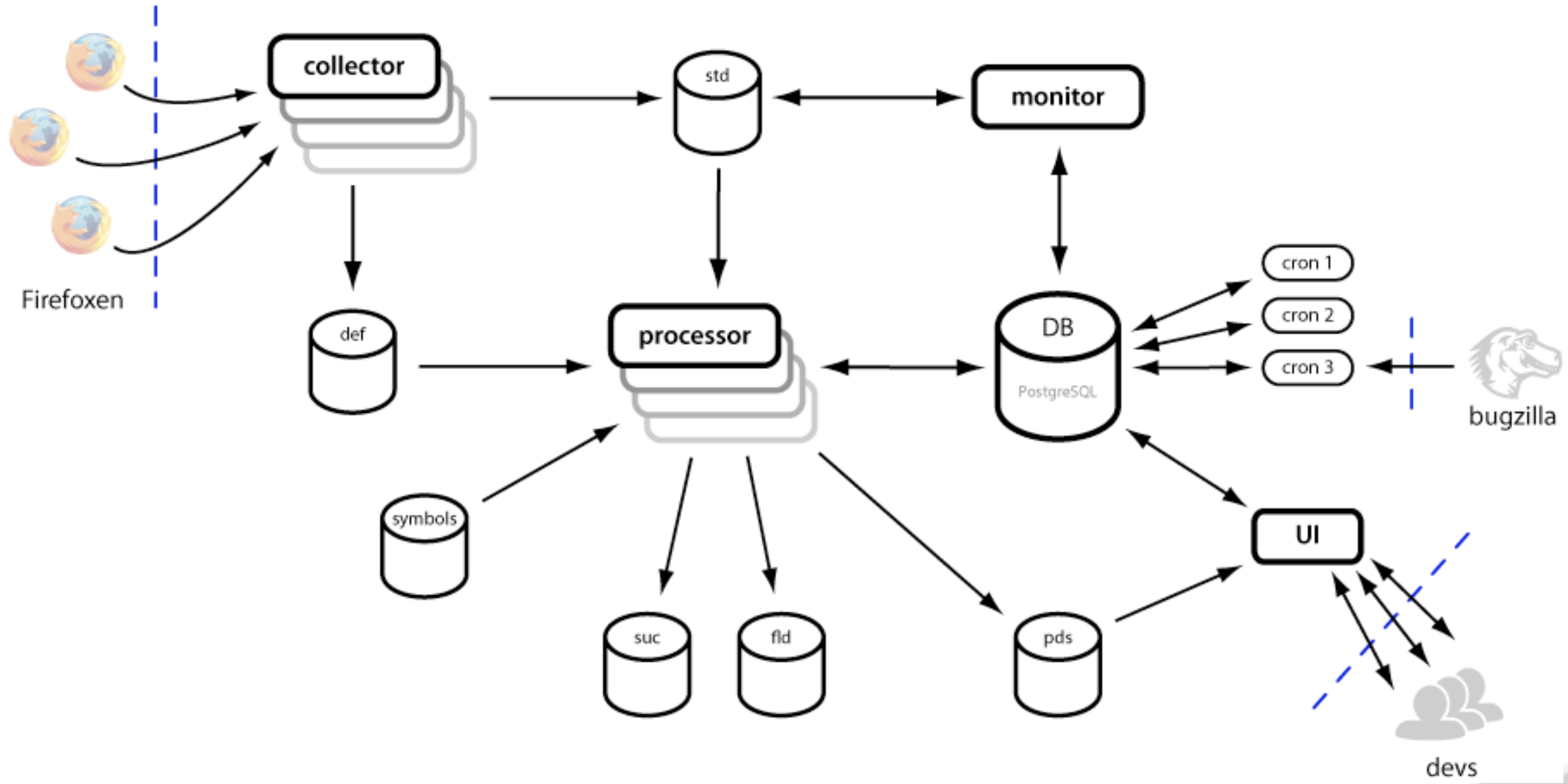
We can rapidly lookup crashes either by name or date, without wasting time having to search.

Back to our data flow...



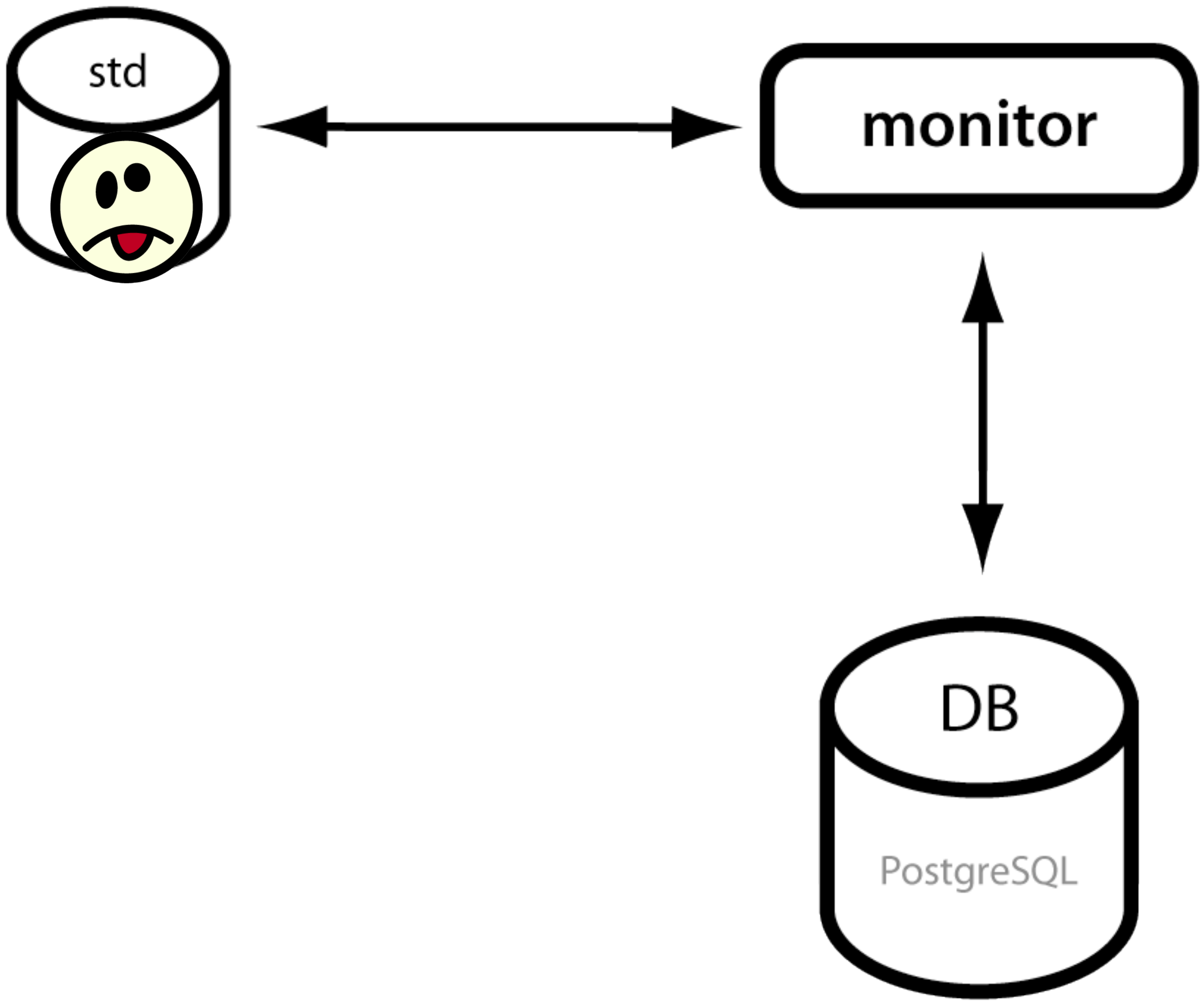
Friday, July 24, 2009

<CLICK>



Friday, July 24, 2009

Now we’re going to look at Monitor, the ring master of this circus



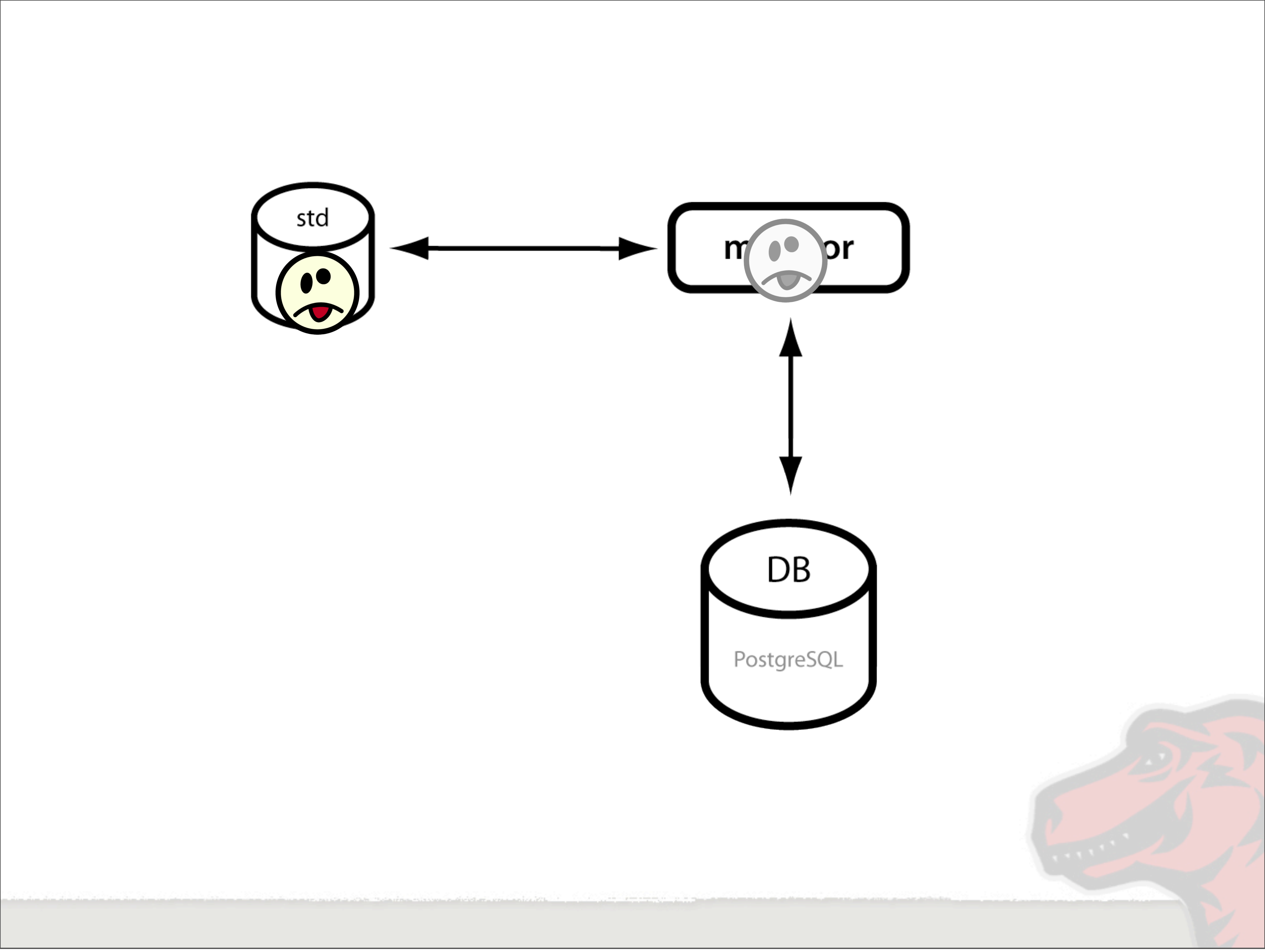
Friday, July 24, 2009

We’ve got a crash waiting in “standard” storage. The monitor can detect that it’s there by following the date directory branches of the file system.

<CLICK>

It doesn’t actually read the crash data, it just notes its existence and saves the name in the database.

<CLICK>



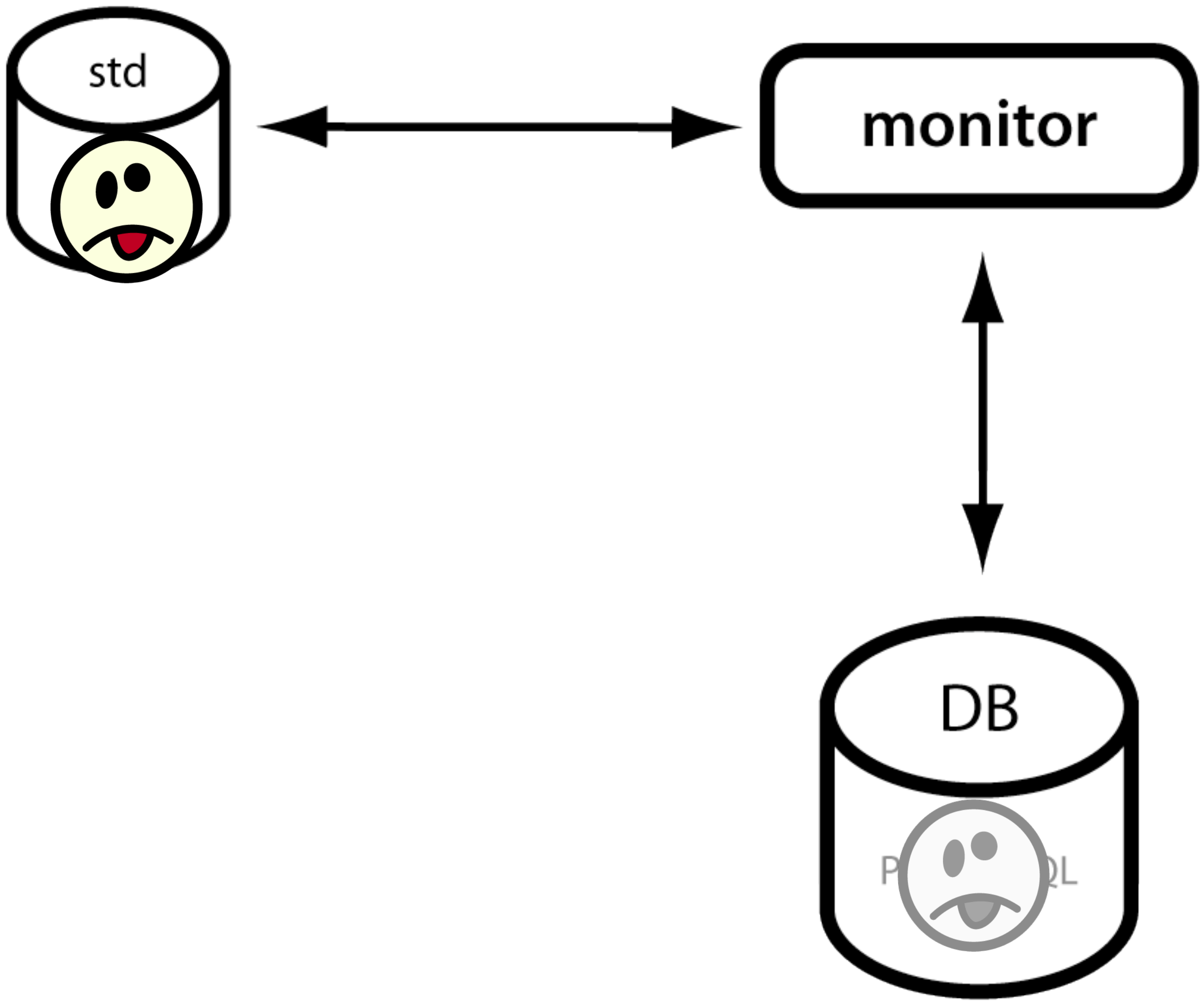
Friday, July 24, 2009

We’ve got a crash waiting in “standard” storage. The monitor can detect that it’s there by following the date directory branches of the file system.

<CLICK>

It doesn’t actually read the crash data, it just notes its existence and saves the name in the database.

<CLICK>



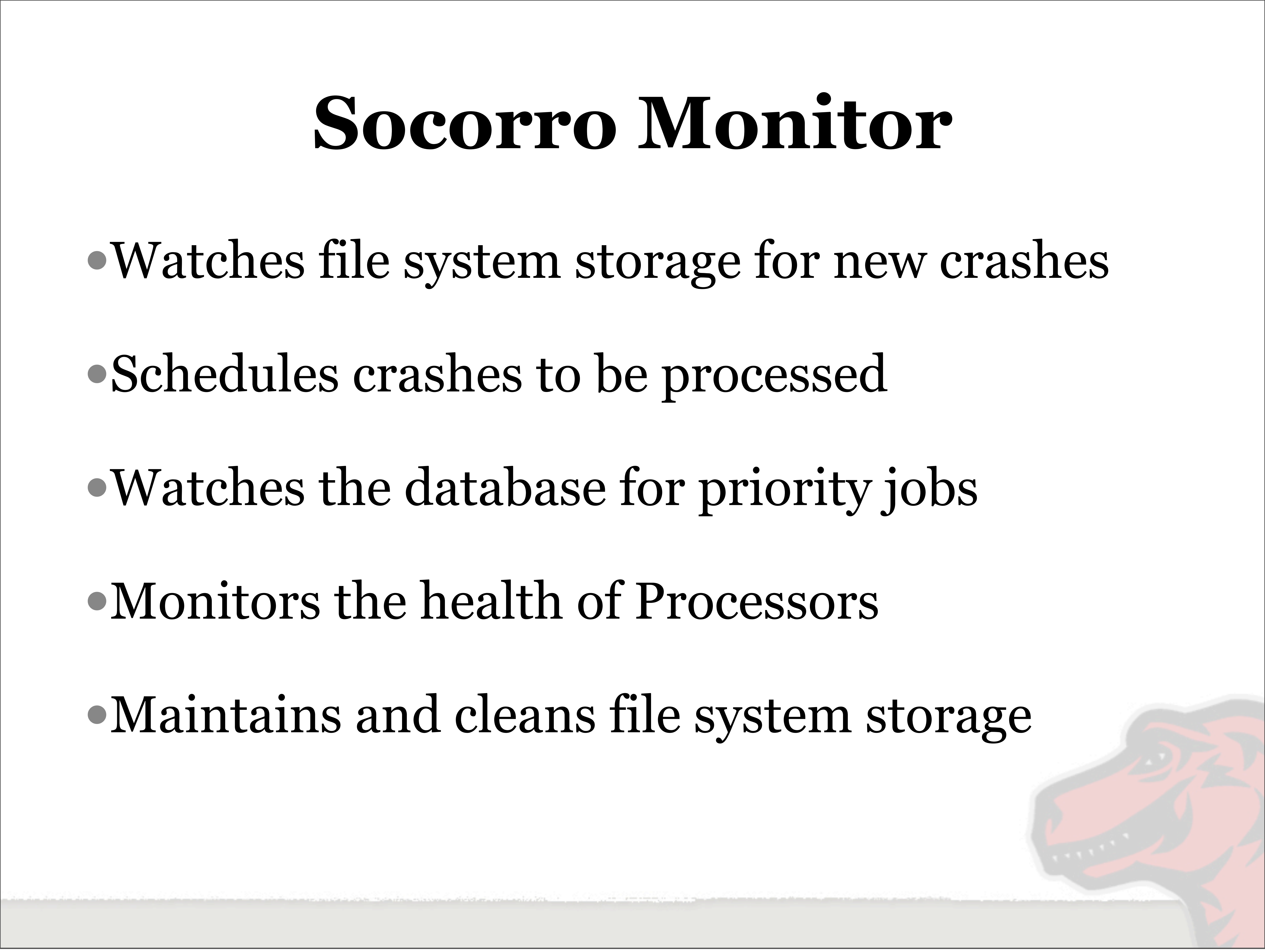
Friday, July 24, 2009

We’ve got a crash waiting in “standard” storage. The monitor can detect that it’s there by following the date directory branches of the file system.

<CLICK>

It doesn’t actually read the crash data, it just notes its existence and saves the name in the database.

<CLICK>



Socorro Monitor

- Watches file system storage for new crashes
- Schedules crashes to be processed
- Watches the database for priority jobs
- Monitors the health of Processors
- Maintains and cleans file system storage

We’ve already seen how monitor watches the file system storage and schedules crashes (at this point referred to as jobs) with processors.

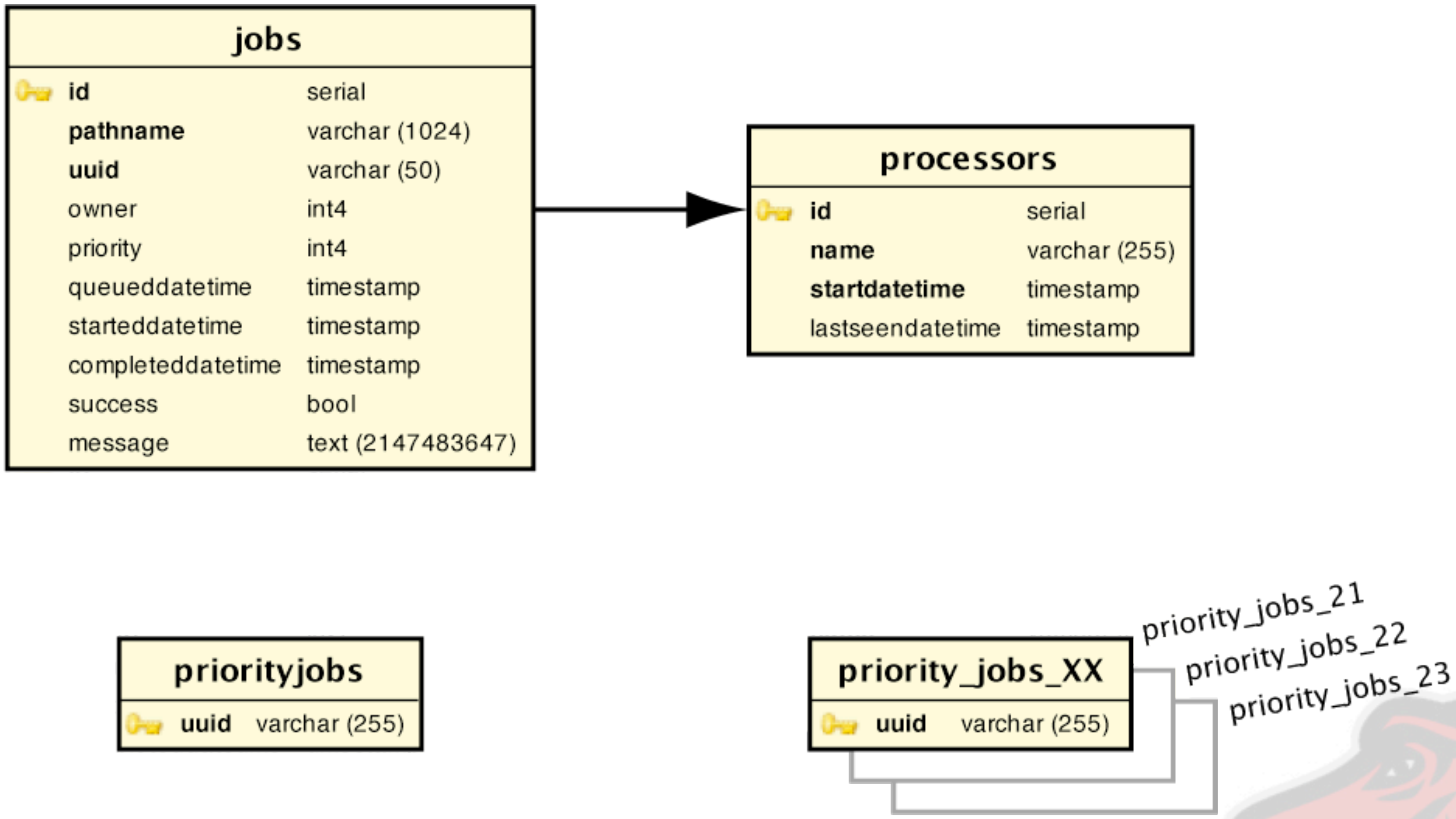
It also watches a special table in the database for priority job requests. Crash processing is not generally a real time activity. Sometimes, we need to get the processed result quickly. Since Monitor is in charge of scheduling, it can let some crashes jump the queue for immediate responses. That’s how jobs from that “deferred” storage can get processed.

Since it’s in charge of scheduling, monitor also watches the health of the processors. If a processor becomes unresponsive and is not doing its work, the monitor has the power to take the processor’s jobs away from it and hand them off to a more responsive processor.

Finally, monitor also is in charge of cleaning old files out of the file system storage. This task may be moved into a cron jon in the future.

With all this stuff to do, it is not surprising that Monitor is a multithreaded application.

Process Control



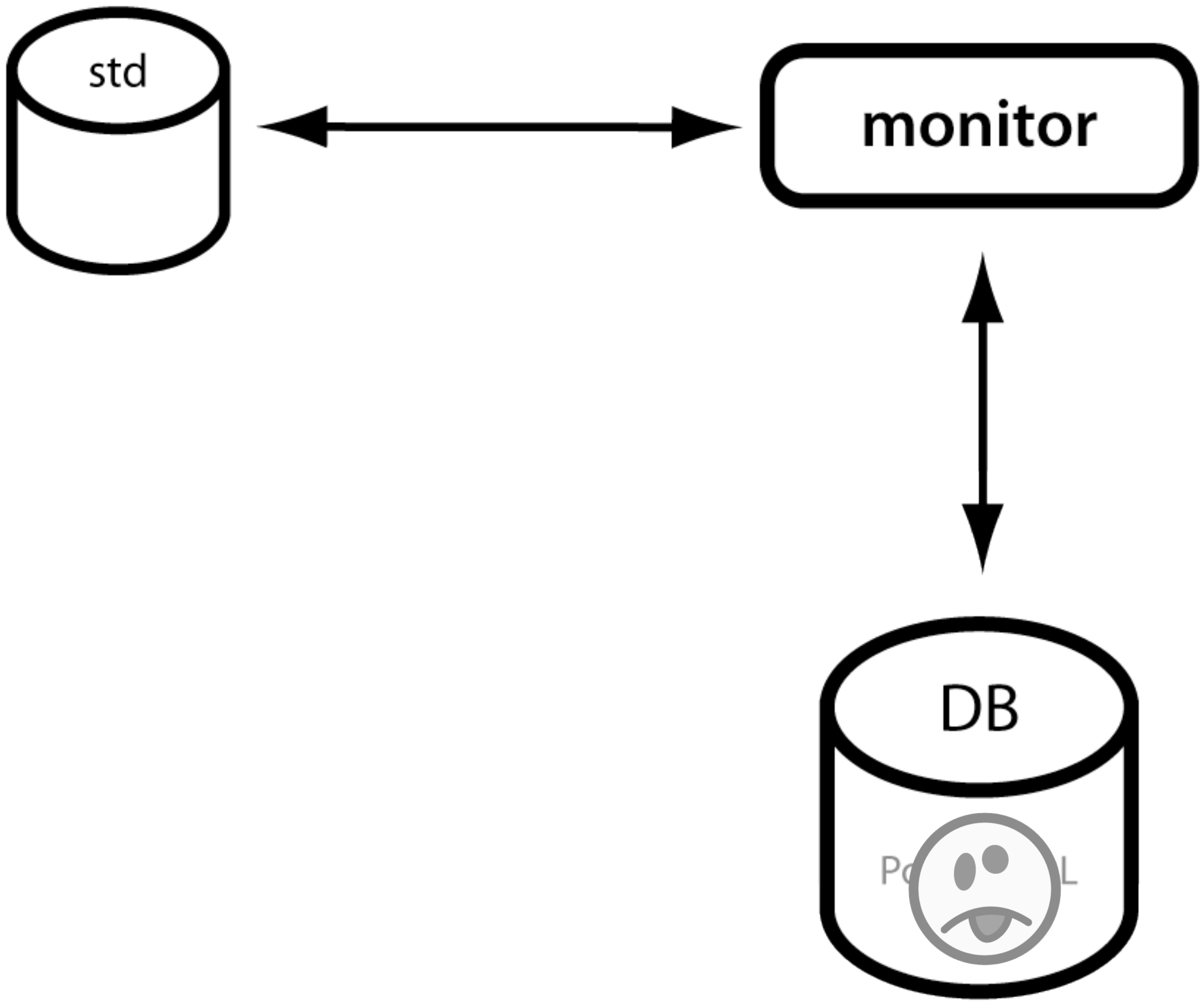
jobs	
id	serial
pathname	varchar (1024)
uuid	varchar (50)
owner	int4
priority	int4
queueddatetime	timestamp
starteddatetime	timestamp
completeddatetime	timestamp
success	bool
message	text (2147483647)

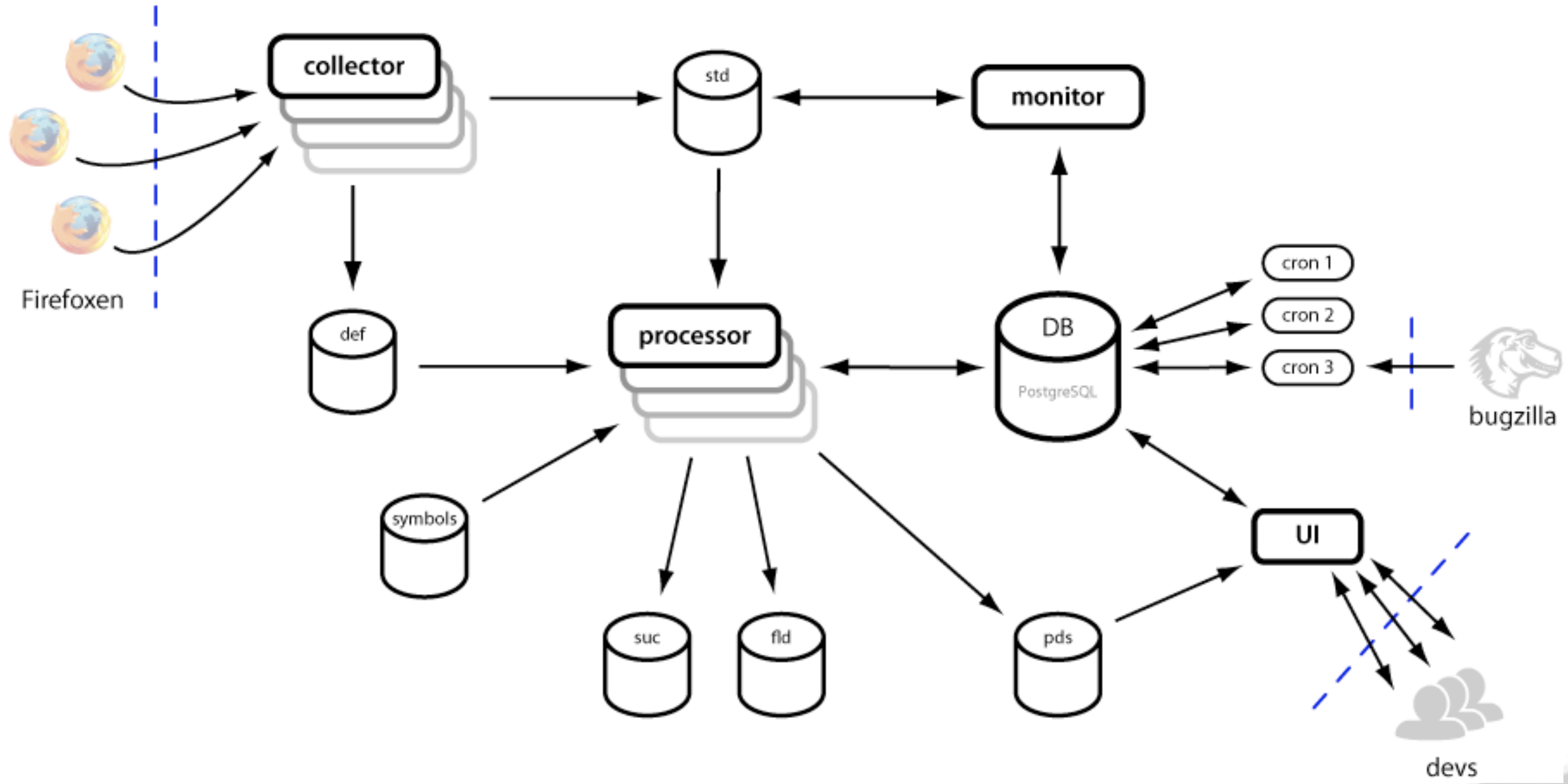
processors	
id	serial
name	varchar (255)
startdatetime	timestamp
lastseendatetime	timestamp

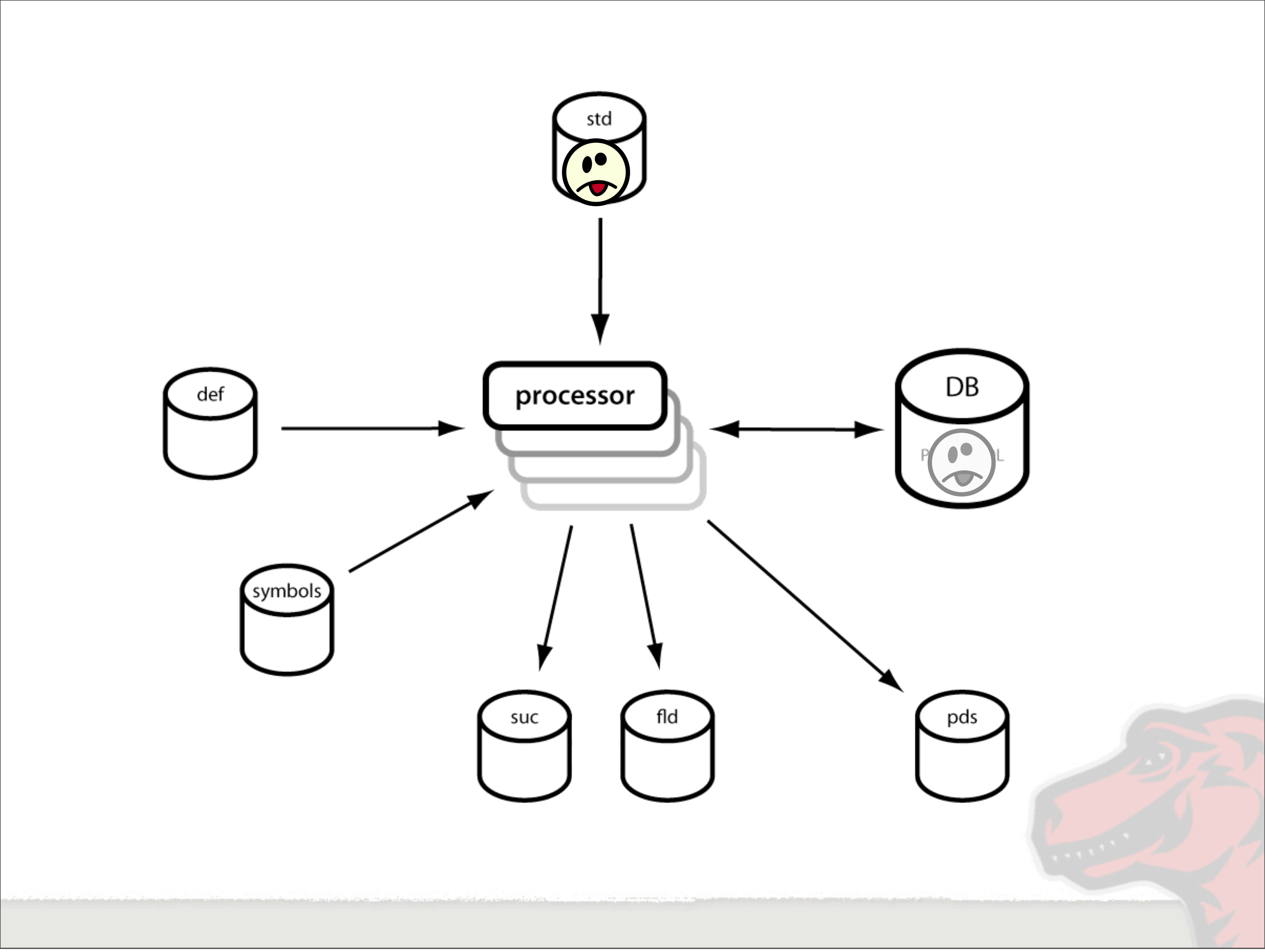
priorityjobs	
uuid	varchar (255)

priority_jobs_XX	
uuid	varchar (255)

In PostgreSQL, the Monitor uses these table for process control. Jobs are assigned to Processors by the Monitor.


std
monitor
DB
Po
L


Firefoxen
collector
std
monitor
def
processor
DB
PostgreSQL
cron 1
cron 2
cron 3
bugzilla
symbols
suc
fld
pds
UI
devs



Friday, July 24, 2009

We start with a crash dump saved in “standard” storage and meta information about the crash in the database in a queue for this processor.

<CLICK>

The processor grabs its next job from the queue and using the meta information, grabs the files to be processed from “standard” storage.

Once it has the crash dump, it invokes Google Breakpad’s minidump_stackwalk program to take the raw crash, remarry it with symbol table information and then save the results.

<CLICK>

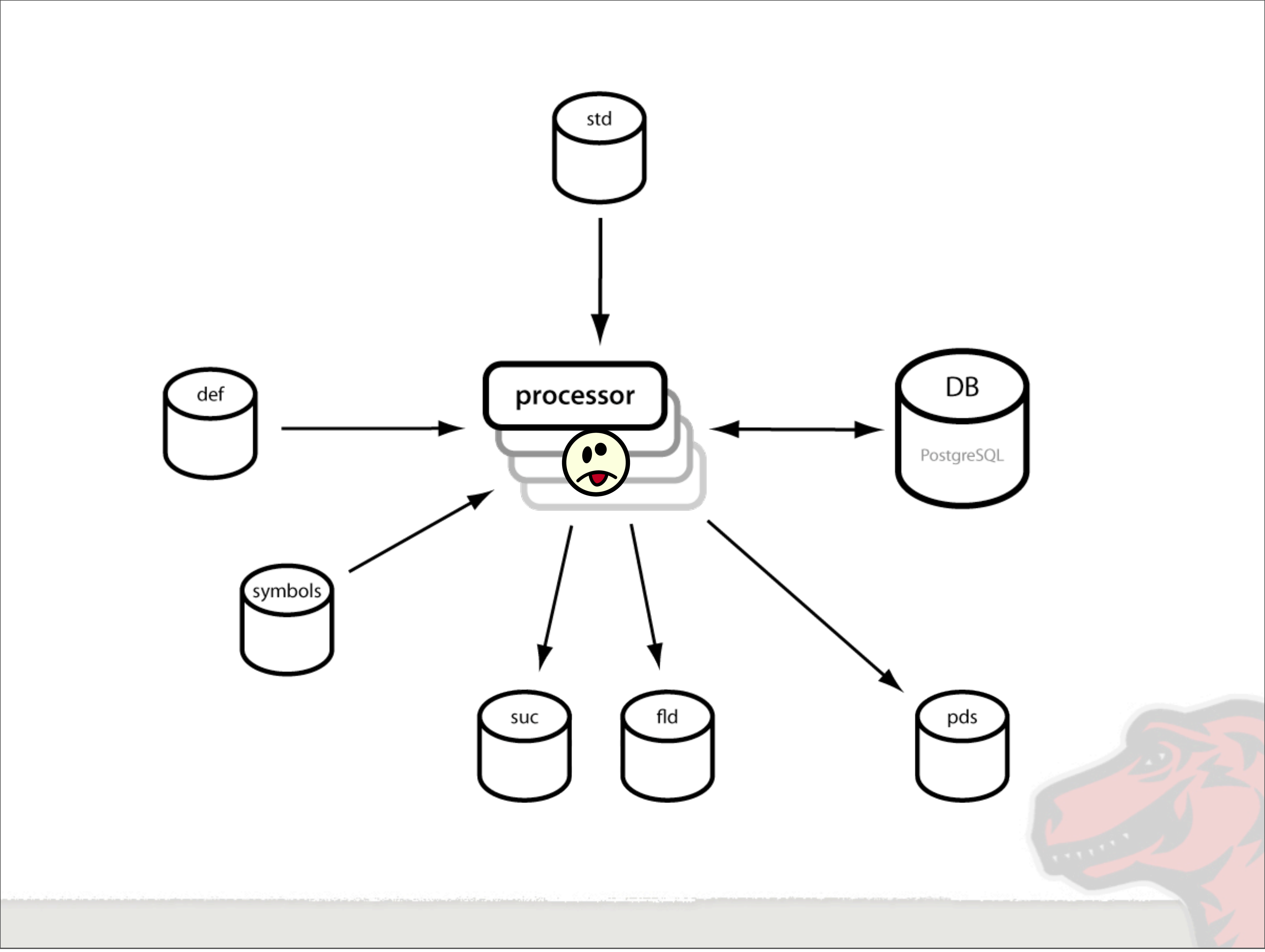
The original crash files are shuffled off to “Successful” storage. If something went wrong with minidump_stackwalk, the original files may head down to “Failed” storage instead. These are optional steps, if configured to do so, processor could just throw the originals away.

<CLICK>

The output of minidump_stackwalk is saved to processed dump storage. This will eventually be used by the UI.

<CLICK>

information parsed from the minidump_stackwalk results is saved in the database. This data is used in the cron jobs to great aggregate reports.



Friday, July 24, 2009

We start with a crash dump saved in “standard” storage and meta information about the crash in the database in a queue for this processor.

<CLICK>

The processor grabs its next job from the queue and using the meta information, grabs the files to be processed from “standard” storage.

Once it has the crash dump, it invokes Google Breakpad’s minidump_stackwalk program to take the raw crash, remarry it with symbol table information and then save the results.

<CLICK>

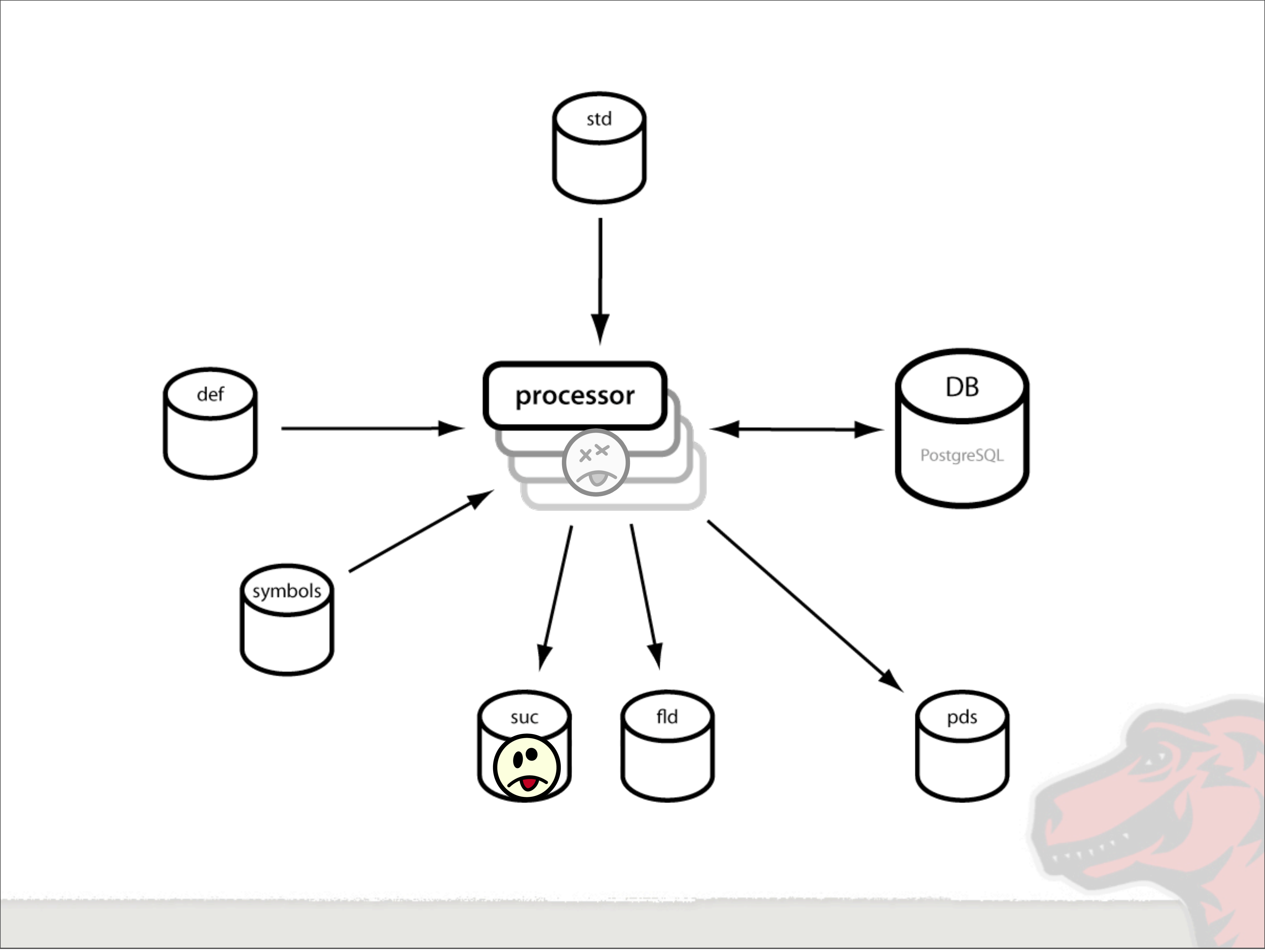
The original crash files are shuffled off to “Successful” storage. If something went wrong with minidump_stackwalk, the original files may head down to “Failed” storage instead. These are optional steps, if configured to do so, processor could just throw the originals away.

<CLICK>

The output of minidump_stackwalk is saved to processed dump storage. This will eventually be used by the UI.

<CLICK>

information parsed from the minidump_stackwalk results is saved in the database. This data is used in the cron jobs to great aggregate reports.



Friday, July 24, 2009

We start with a crash dump saved in “standard” storage and meta information about the crash in the database in a queue for this processor.

<CLICK>

The processor grabs its next job from the queue and using the meta information, grabs the files to be processed from “standard” storage.

Once it has the crash dump, it invokes Google Breakpad’s minidump_stackwalk program to take the raw crash, remarry it with symbol table information and then save the results.

<CLICK>

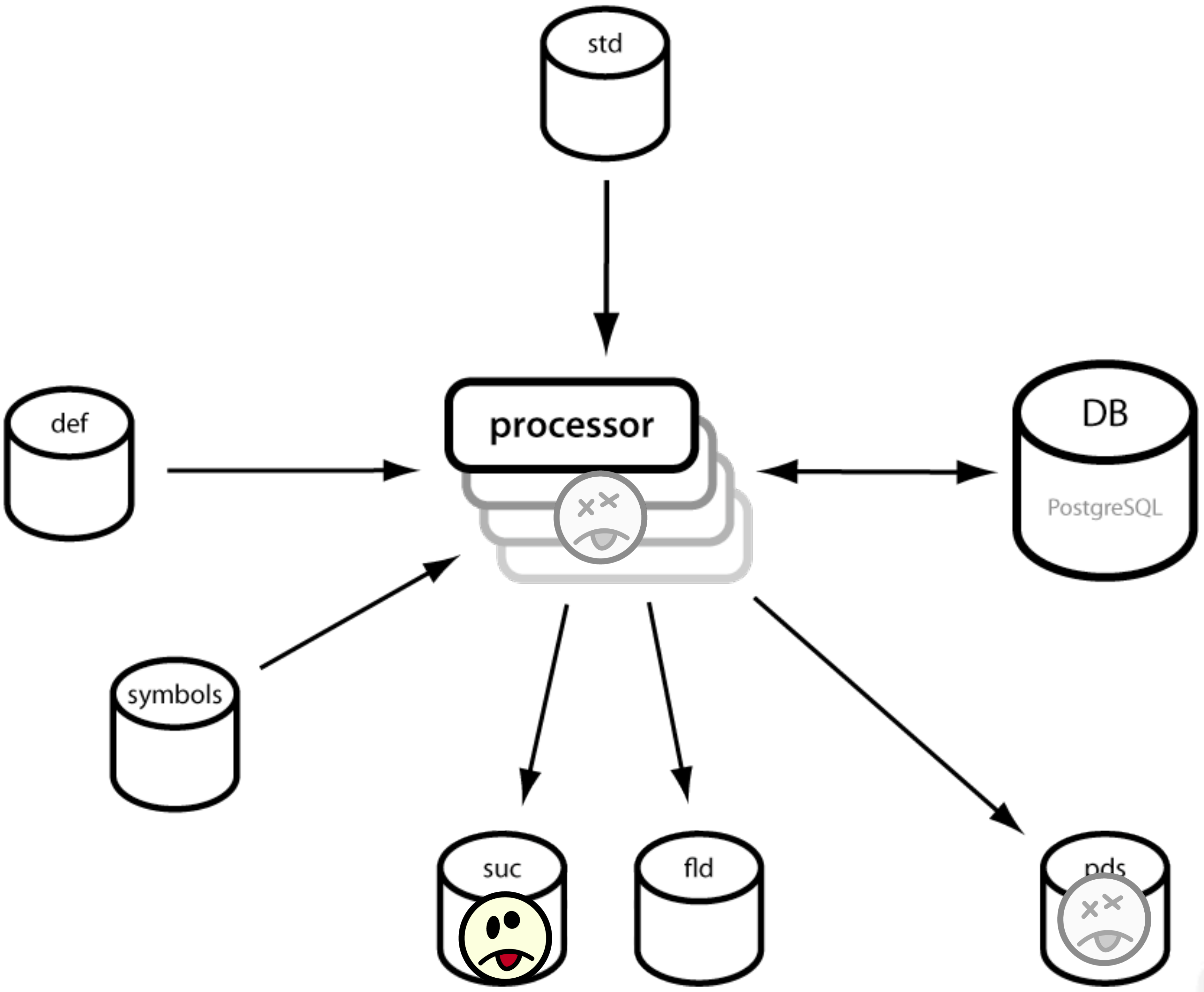
The original crash files are shuffled off to “Successful” storage. If something went wrong with minidump_stackwalk, the original files may head down to “Failed” storage instead. These are optional steps, if configured to do so, processor could just throw the originals away.

<CLICK>

The output of minidump_stackwalk is saved to processed dump storage. This will eventually be used by the UI.

<CLICK>

information parsed from the minidump_stackwalk results is saved in the database. This data is used in the cron jobs to great aggregate reports.



We start with a crash dump saved in “standard” storage and meta information about the crash in the database in a queue for this processor.

<CLICK>

The processor grabs its next job from the queue and using the meta information, grabs the files to be processed from “standard” storage.

Once it has the crash dump, it invokes Google Breakpad’s minidump_stackwalk program to take the raw crash, remarry it with symbol table information and then save the results.

<CLICK>

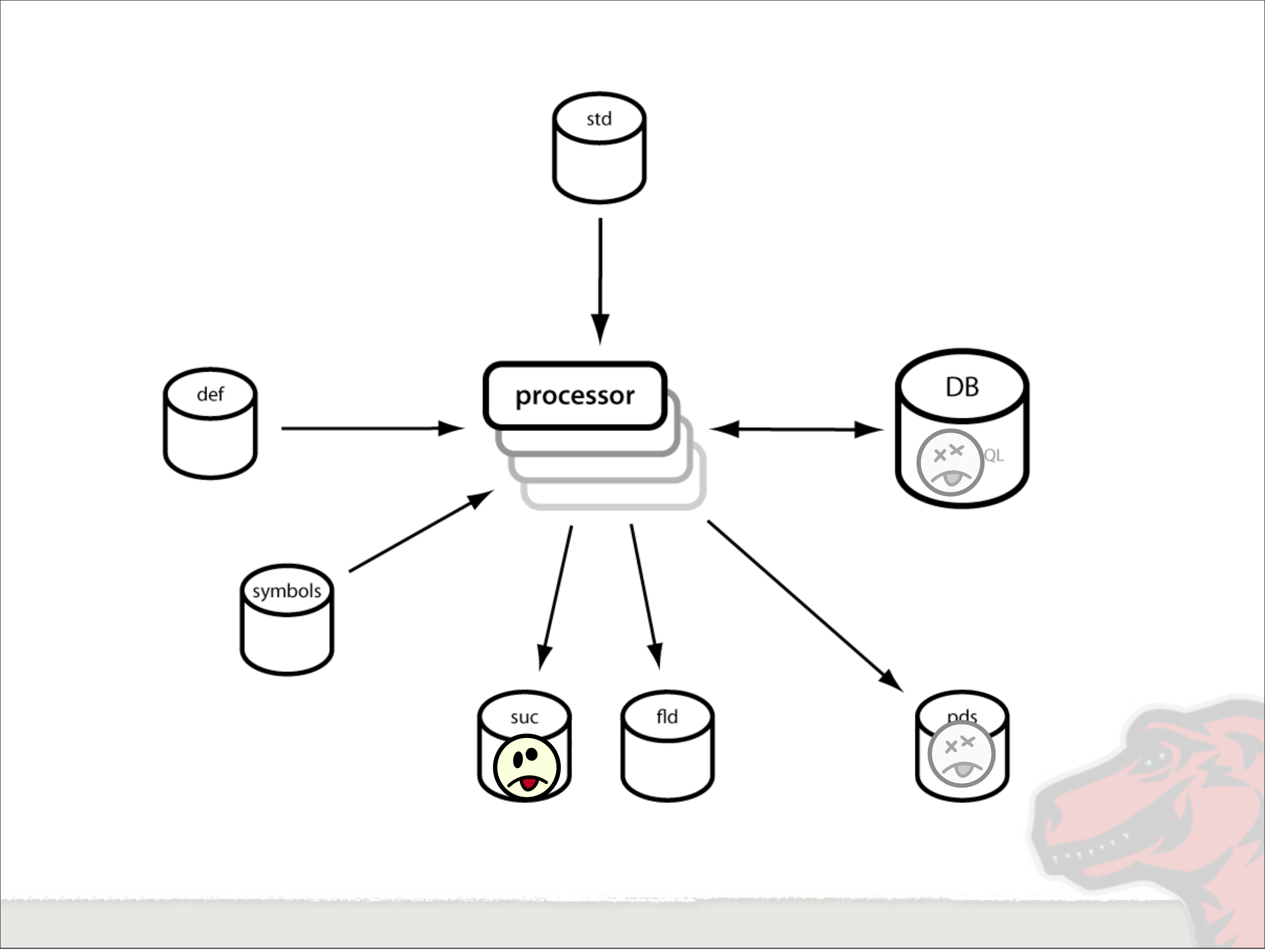
The original crash files are shuffled off to “Successful” storage. If something went wrong with minidump_stackwalk, the original files may head down to “Failed” storage instead. These are optional steps, if configured to do so, processor could just throw the originals away.

<CLICK>

The output of minidump_stackwalk is saved to processed dump storage. This will eventually be used by the UI.

<CLICK>

information parsed from the minidump_stackwalk results is saved in the database. This data is used in the cron jobs to great aggregate reports.



Friday, July 24, 2009

We start with a crash dump saved in “standard” storage and meta information about the crash in the database in a queue for this processor.

<CLICK>

The processor grabs its next job from the queue and using the meta information, grabs the files to be processed from “standard” storage.

Once it has the crash dump, it invokes Google Breakpad’s minidump_stackwalk program to take the raw crash, remarry it with symbol table information and then save the results.

<CLICK>

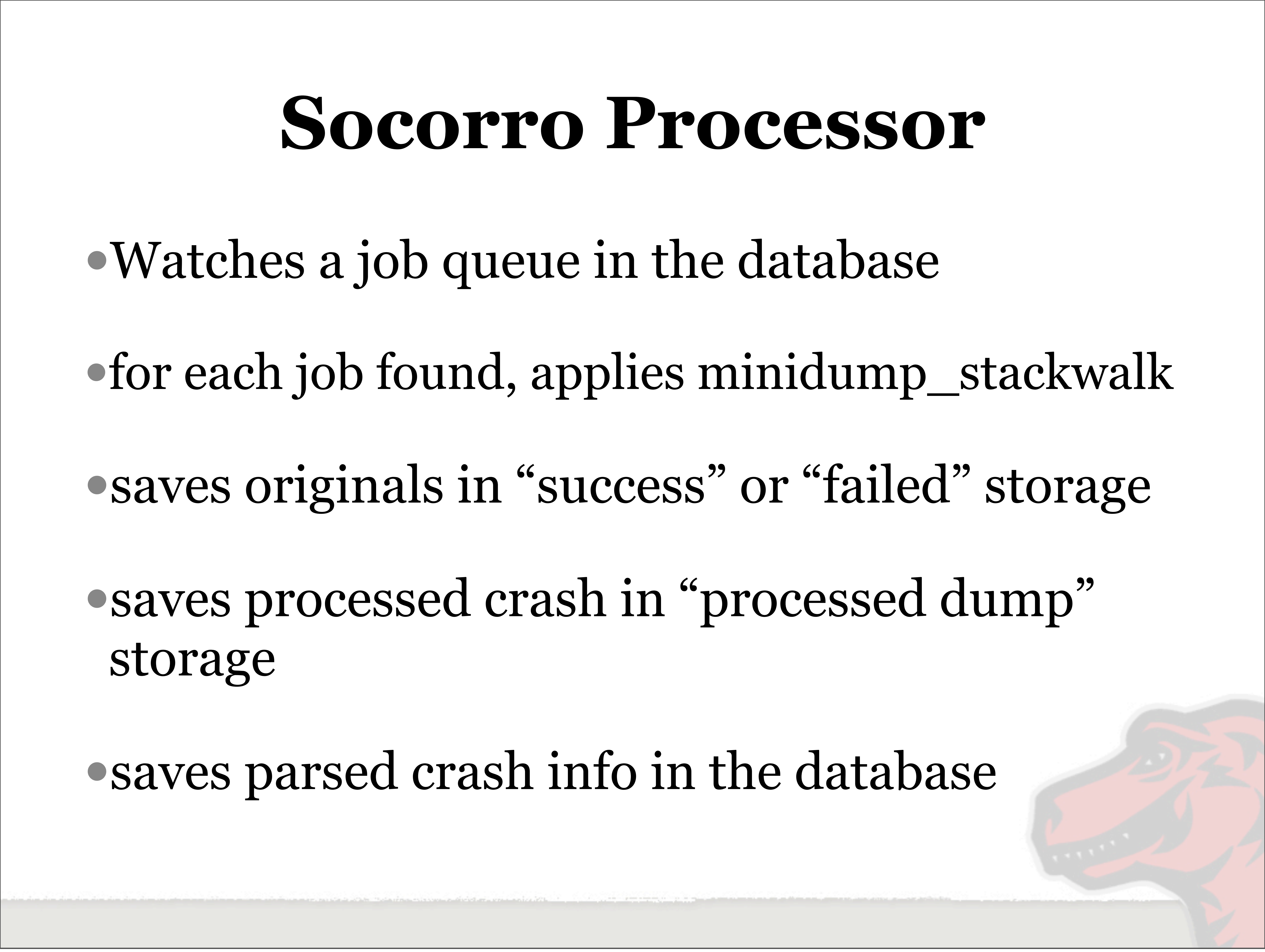
The original crash files are shuffled off to “Successful” storage. If something went wrong with minidump_stackwalk, the original files may head down to “Failed” storage instead. These are optional steps, if configured to do so, processor could just throw the originals away.

<CLICK>

The output of minidump_stackwalk is saved to processed dump storage. This will eventually be used by the UI.

<CLICK>

information parsed from the minidump_stackwalk results is saved in the database. This data is used in the cron jobs to great aggregate reports.



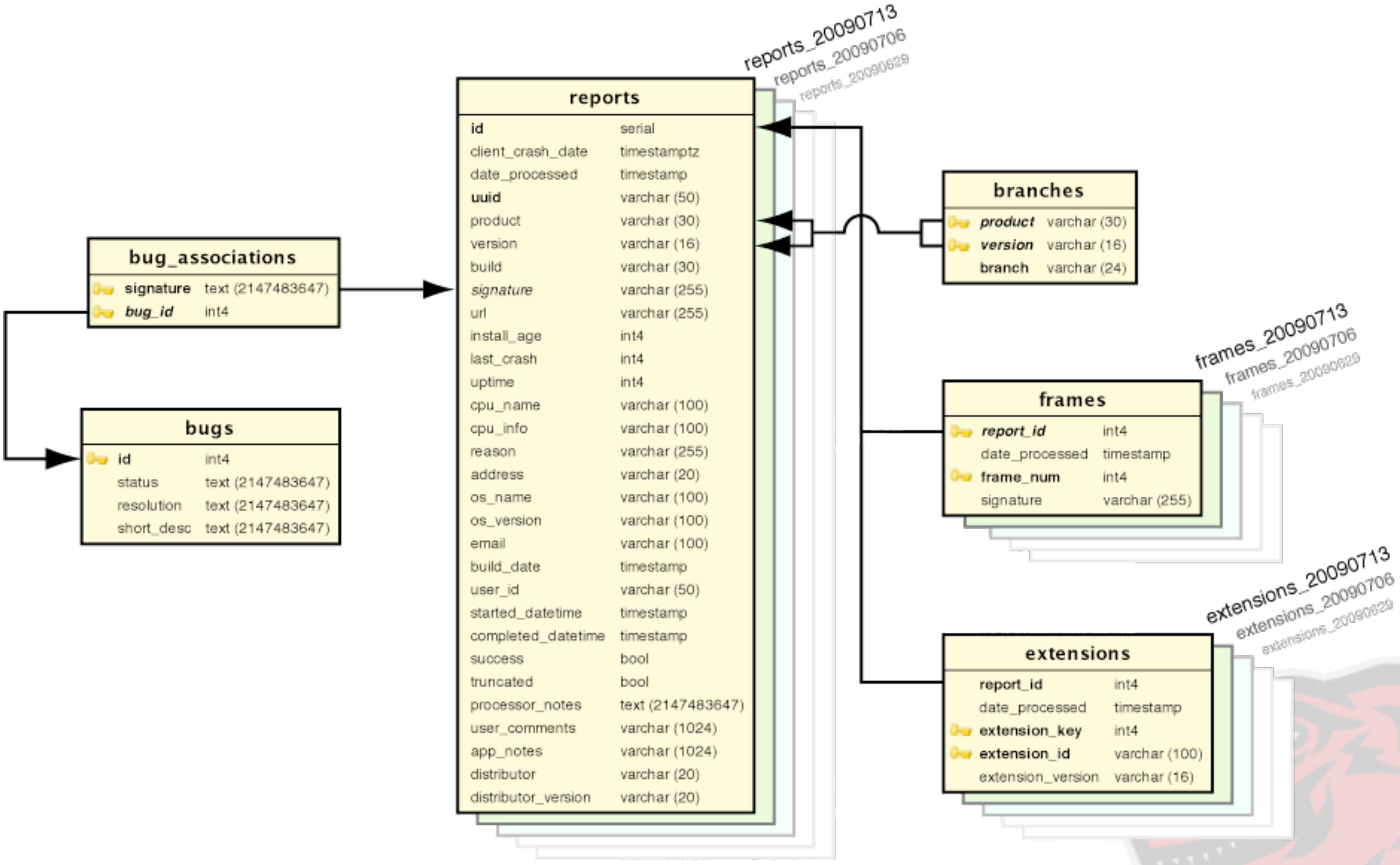
Socorro Processor

- Watches a job queue in the database
- for each job found, applies minidump_stackwalk
- saves originals in “success” or “failed” storage
- saves processed crash in “processed dump” storage
- saves parsed crash info in the database

Friday, July 24, 2009

ok, we’ve seen how the processors watch the job queue and applies minidump_stackwalk and then dumps the results into a file system _and_ the database.

Database Crash Data



bug_associations	
signature	text (2147483647)
bug_id	int4

bugs	
id	int4
status	text (2147483647)
resolution	text (2147483647)
short_desc	text (2147483647)

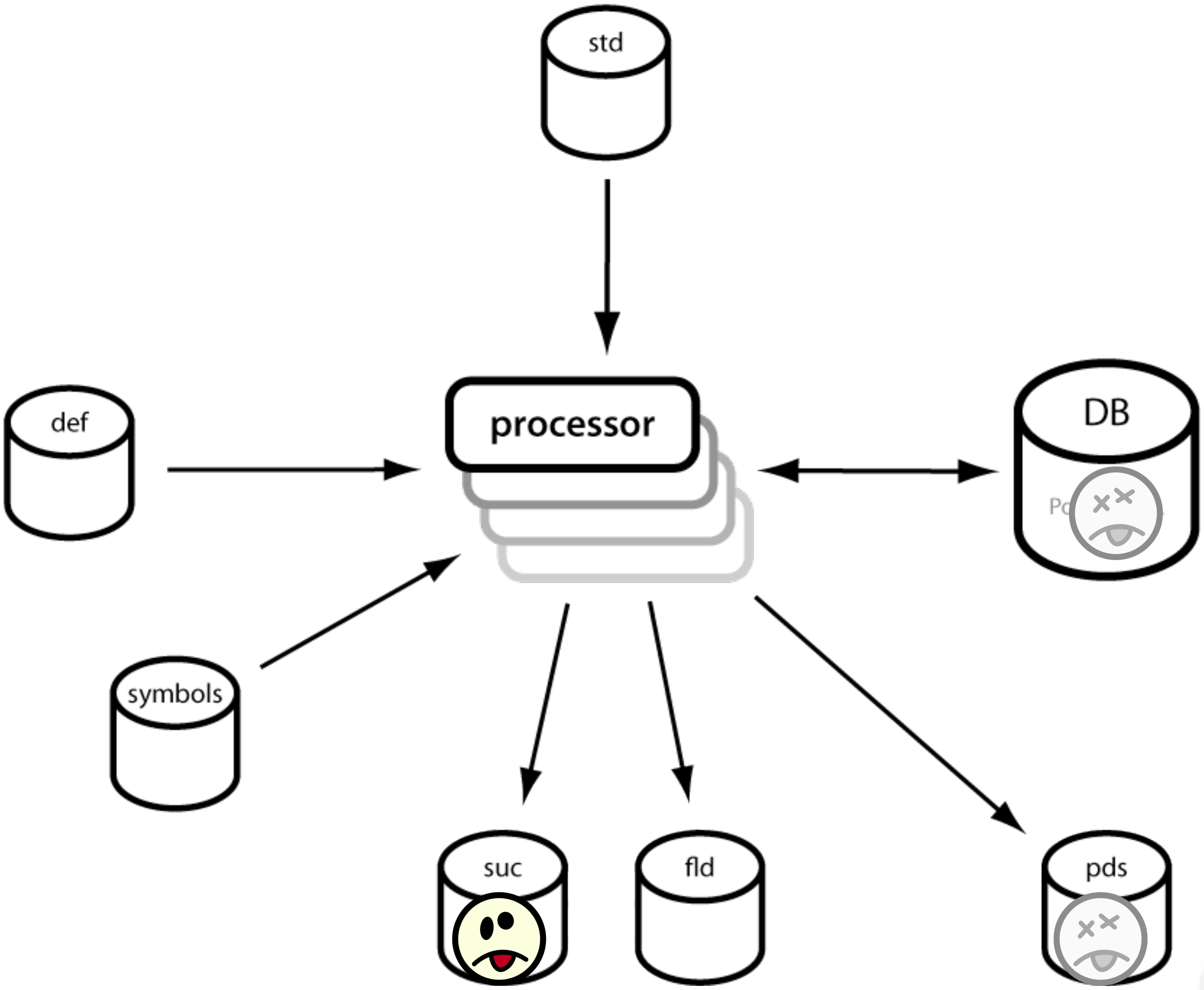
reports	
id	serial
client_crash_date	timestamptz
date_processed	timestamp
uuid	varchar (50)
product	varchar (30)
version	varchar (16)
build	varchar (30)
signature	varchar (255)
url	varchar (255)
install_age	int4
last_crash	int4
uptime	int4
cpu_name	varchar (100)
cpu_info	varchar (100)
reason	varchar (255)
address	varchar (20)
os_name	varchar (100)
os_version	varchar (100)
email	varchar (100)
build_date	timestamp
user_id	varchar (50)
started_datetime	timestamp
completed_datetime	timestamp
success	bool
truncated	bool
processor_notes	text (2147483647)
user_comments	varchar (1024)
app_notes	varchar (1024)
distributor	varchar (20)
distributor_version	varchar (20)

branches	
product	varchar (30)
version	varchar (16)
branch	varchar (24)

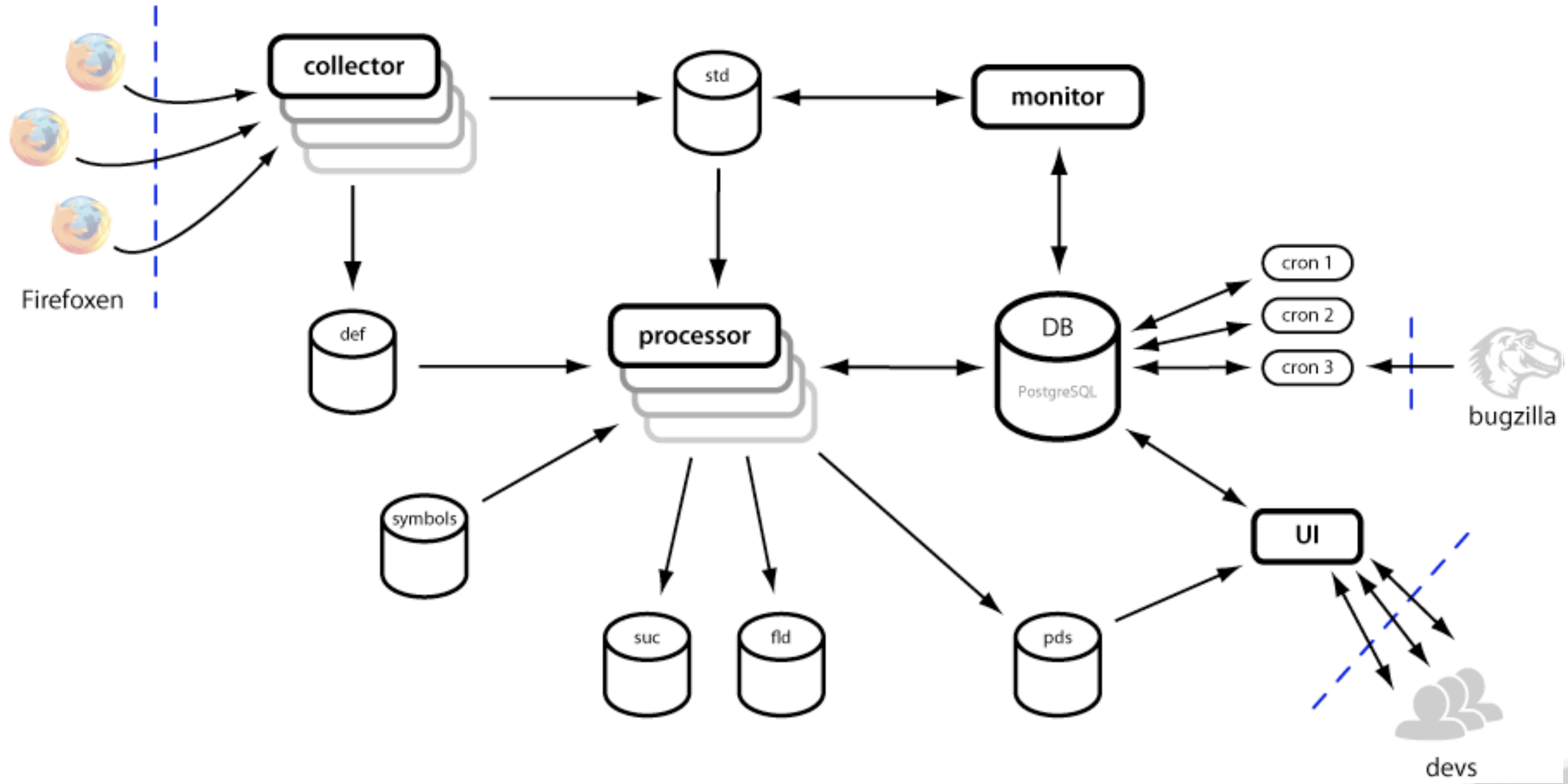
frames	
report_id	int4
date_processed	timestamp
frame_num	int4
signature	varchar (255)

extensions	
report_id	int4
date_processed	timestamp
extension_key	int4
extension_id	varchar (100)
extension_version	varchar (16)

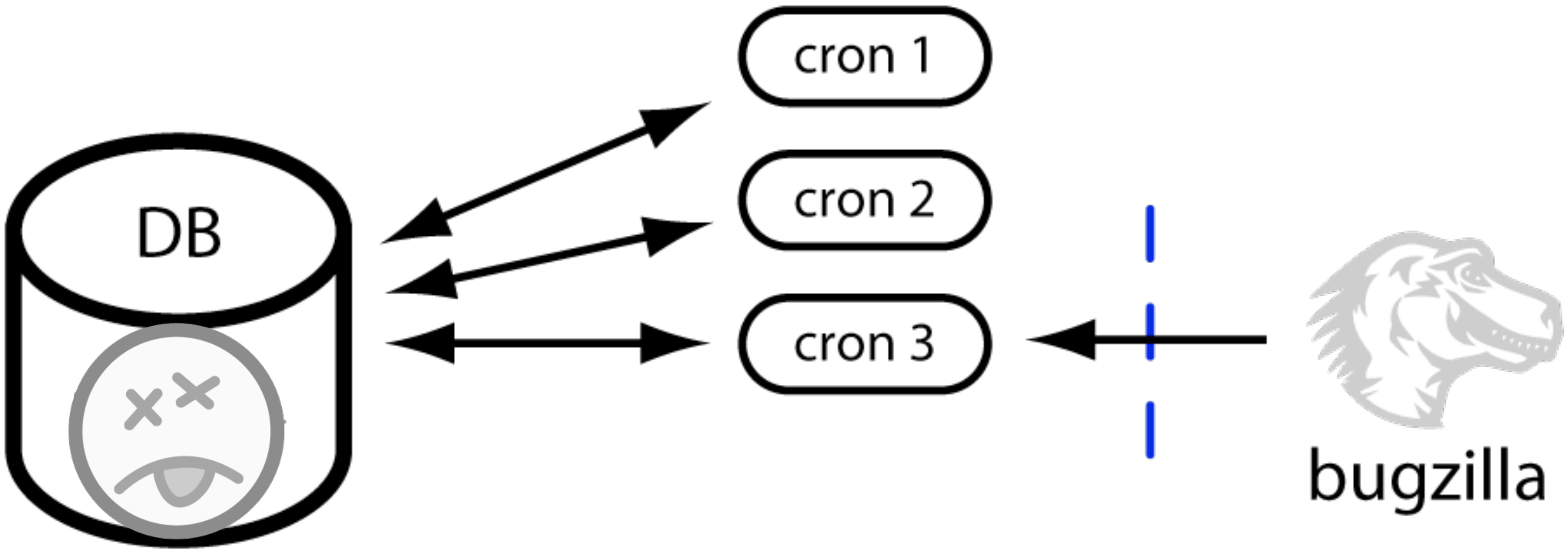
The reports, frames and extensions tables are partitioned. For efficiency in the database, there are many instances of these tables each holding data from a range in time. Partitioning is implemented using PostgreSQL table inheritance.


std
def
processor
DB
Po
symbols
suc
fld
pds

<CLICK>


collector
std
monitor
Firefoxen
def
processor
DB
PostgreSQL
cron 1
cron 2
cron 3
bugzilla
symbols
UI
suc
fld
pds
devs

Cron Jobs

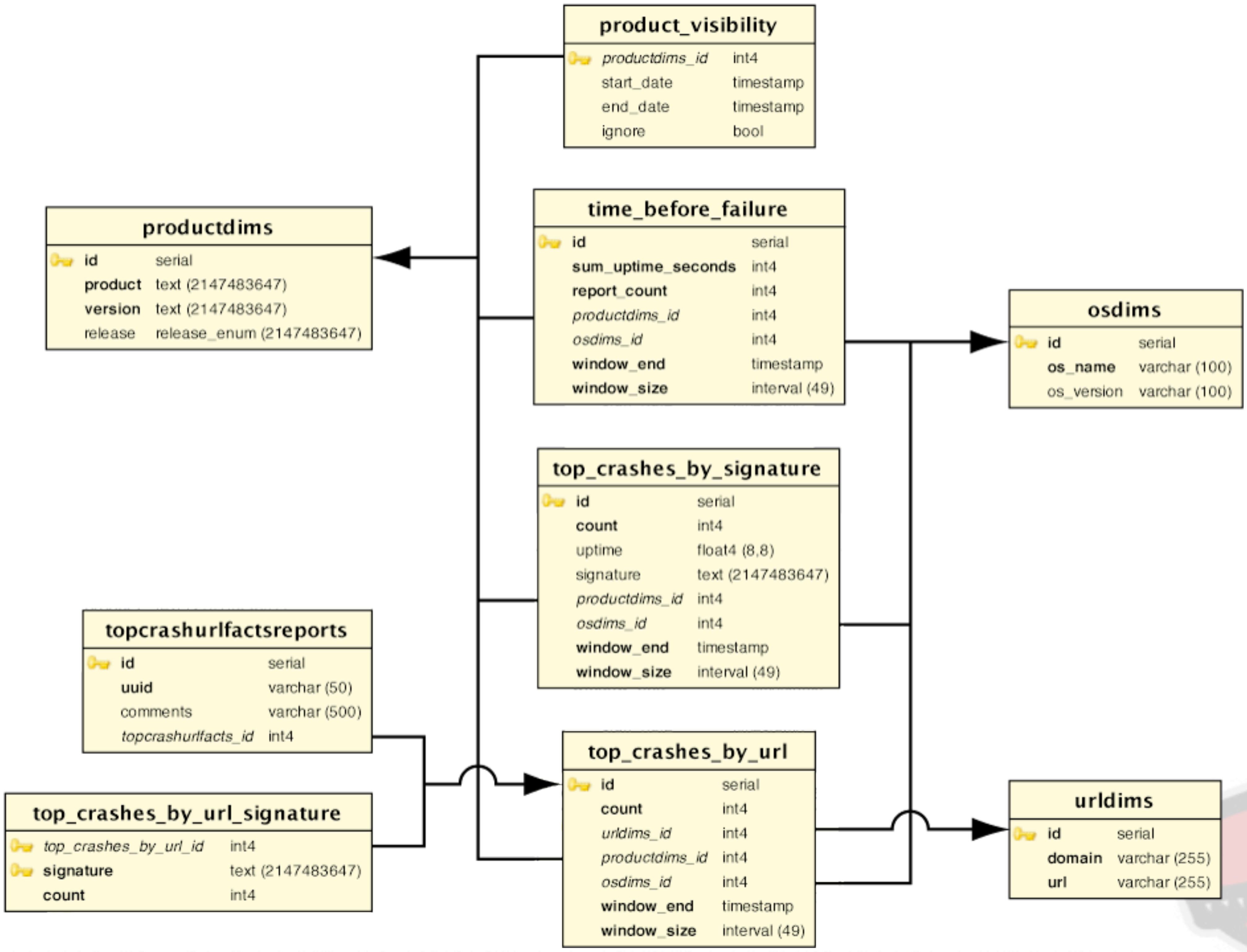


These are cluster of auxiliary applications that feed off the crash data in the database.

Cron Jobs

- mean time before failure
- top crasher by signature
- top crasher by url
- bugzilla associator
- file system cleaner

Data Aggregation Schema



product_visibility	
productdims_id	int4
start_date	timestamp
end_date	timestamp
ignore	bool

productdims	
id	serial
product	text (2147483647)
version	text (2147483647)
release	release_enum (2147483647)

time_before_failure	
id	serial
sum_uptime_seconds	int4
report_count	int4
productdims_id	int4
osdims_id	int4
window_end	timestamp
window_size	interval (49)

osdims	
id	serial
os_name	varchar (100)
os_version	varchar (100)

top_crashes_by_signature	
id	serial
count	int4
uptime	float4 (8,8)
signature	text (2147483647)
productdims_id	int4
osdims_id	int4
window_end	timestamp
window_size	interval (49)

topcrashurlfactsreports	
id	serial
uuid	varchar (50)
comments	varchar (500)
topcrashurlfacts_id	int4

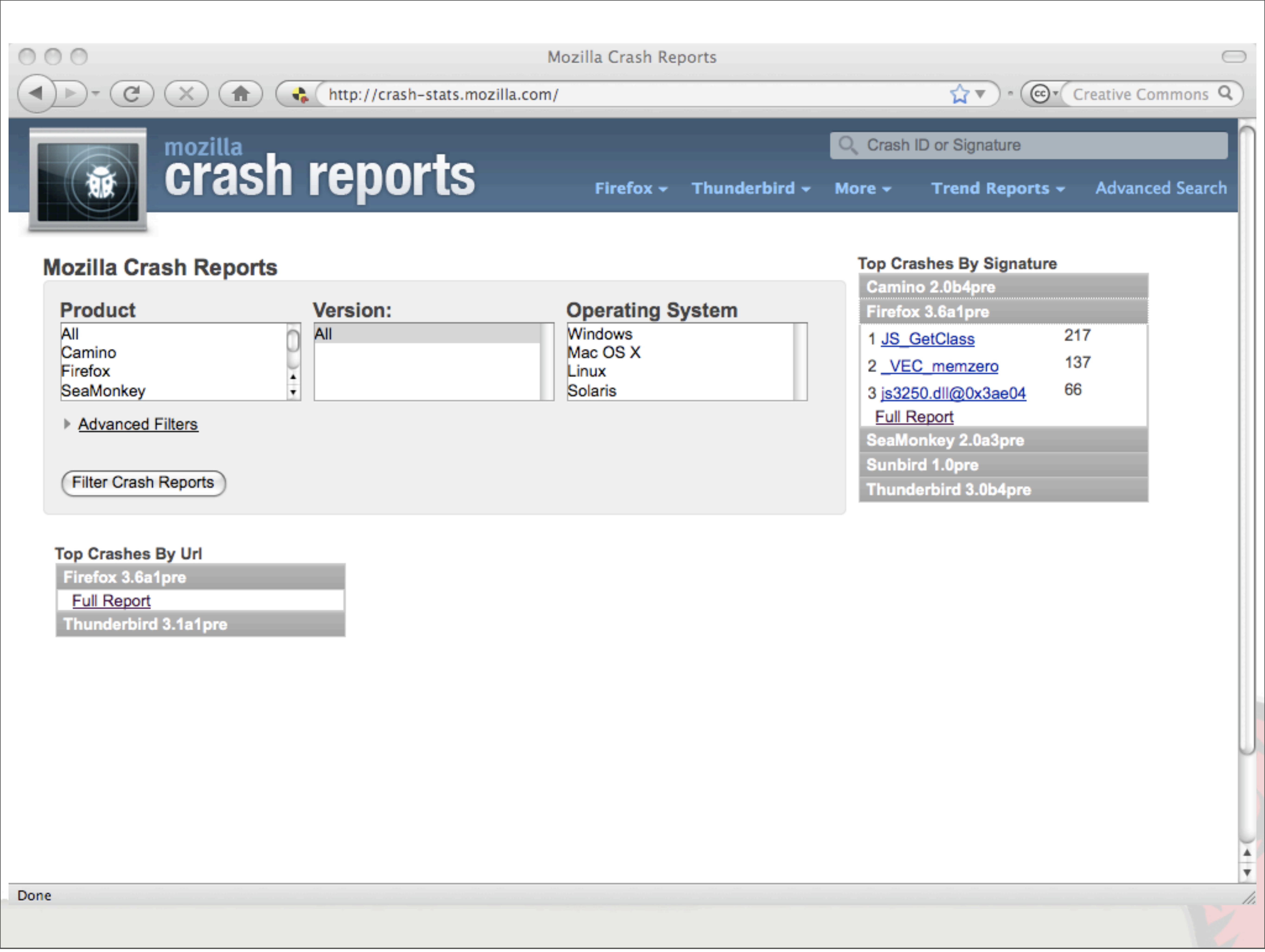
top_crashes_by_url_signature	
top_crashes_by_url_id	int4
signature	text (2147483647)
count	int4

top_crashes_by_url	
id	serial
count	int4
urldims_id	int4
productdims_id	int4
osdims_id	int4
window_end	timestamp
window_size	interval (49)

urldims	
id	serial
domain	varchar (255)
url	varchar (255)

These are the tables the support the “materialized views” of the crash data. These support the getting statistical information out of the database more quickly than trying to regenerate it on demand.

Socorro UI
(reporter)



Mozilla Crash Reports

Product	Version:	Operating System
All	All	Windows
Camino		Mac OS X
Firefox		Linux
SeaMonkey		Solaris

Advanced Filters

Filter Crash Reports

Top Crashes By Signature

Camino 2.0b4pre

Firefox 3.6a1pre

1	JS_GetClass	217
2	_VEC_memzero	137
3	js3250.dll@0x3ae04	66

Full Report

SeaMonkey 2.0a3pre

Sunbird 1.0pre

Thunderbird 3.0b4pre

Top Crashes By Url

Firefox 3.6a1pre

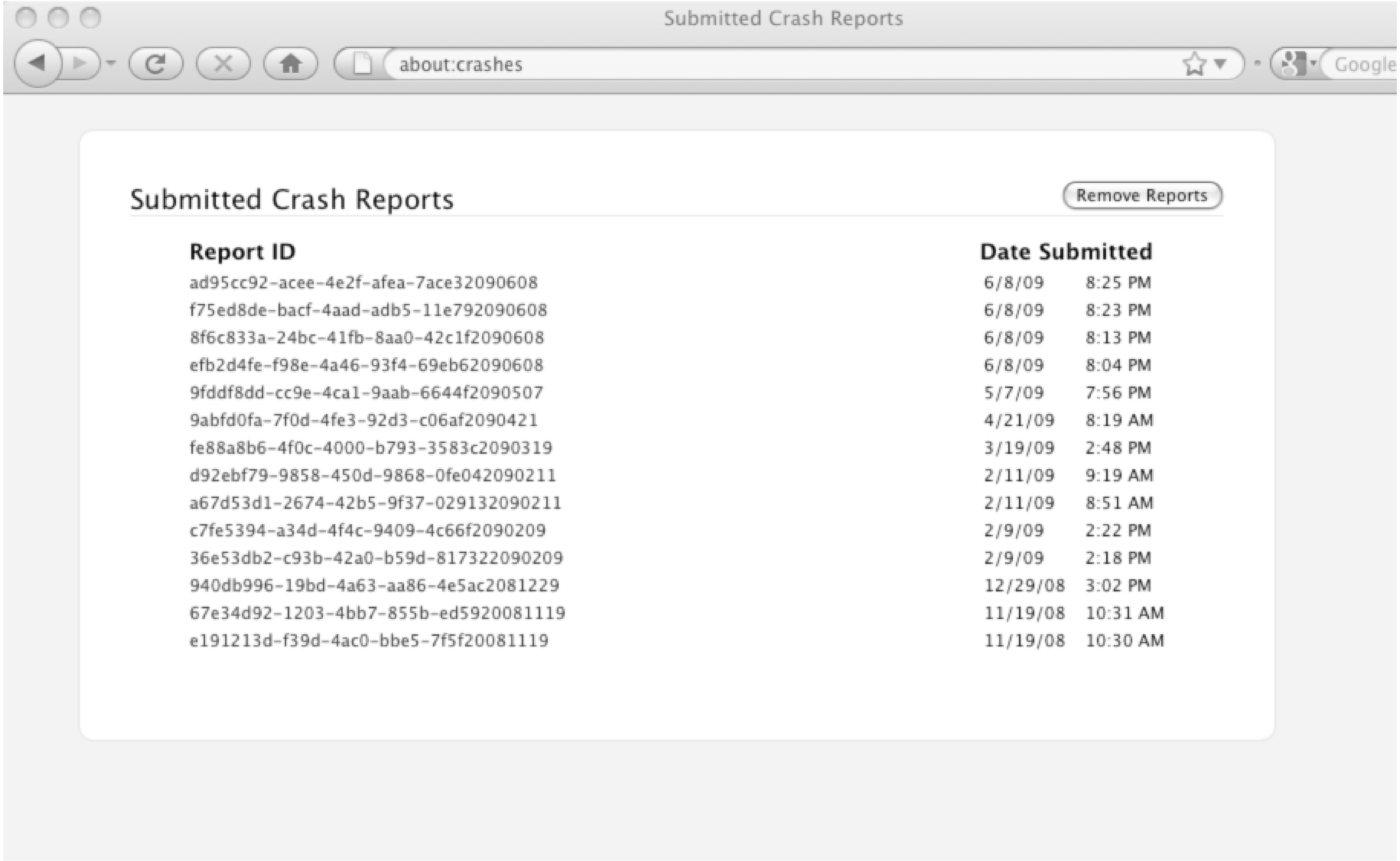
Full Report

Thunderbird 3.1a1pre

3 ways to play

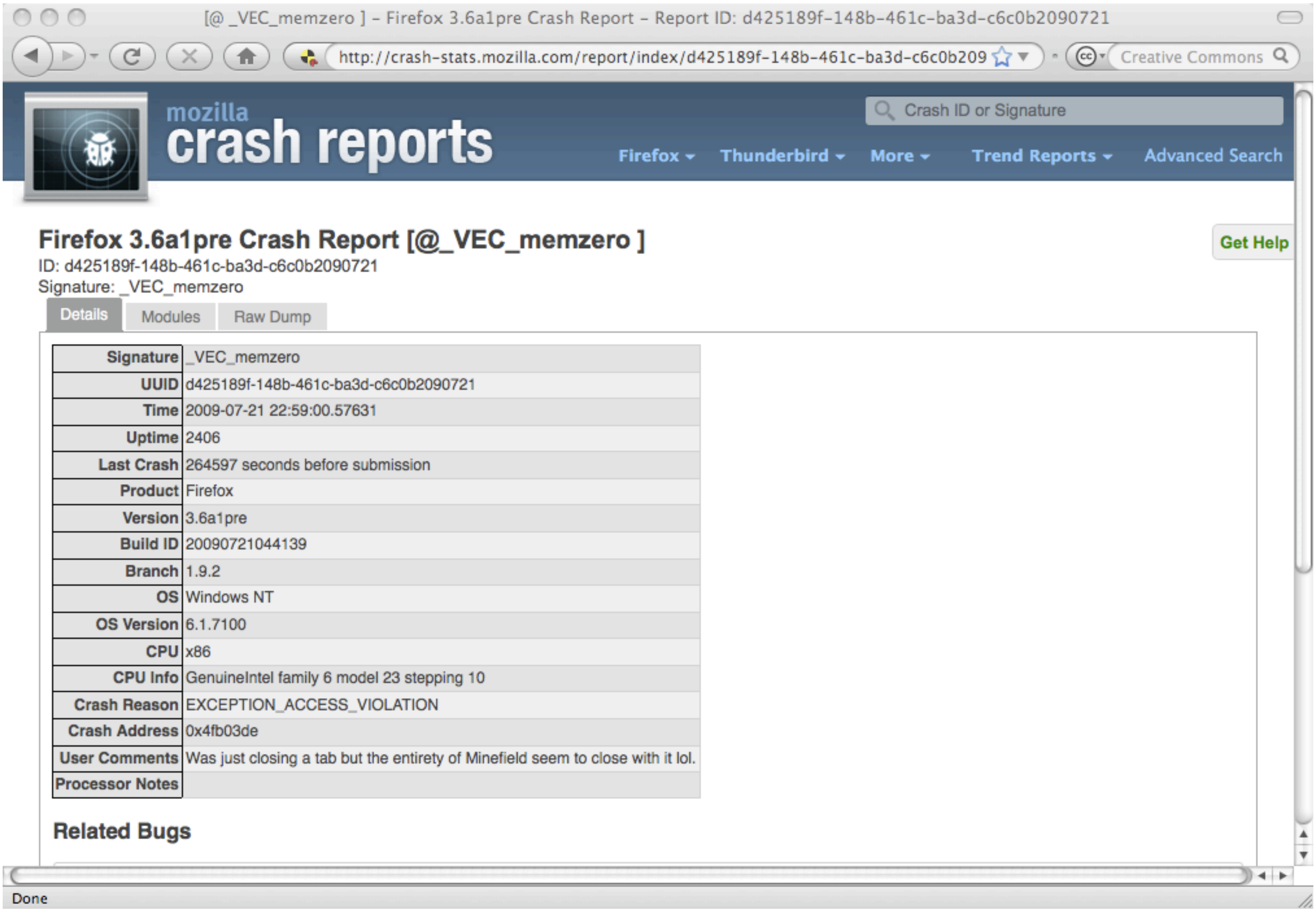
- View Your Crash
- Search
- Trend Reports

View Your Crash



Submitted Crash Reports

Report ID	Date Submitted	
ad95cc92-acee-4e2f-afea-7ace32090608	6/8/09	8:25 PM
f75ed8de-bacf-4aad-adb5-11e792090608	6/8/09	8:23 PM
8f6c833a-24bc-41fb-8aa0-42c1f2090608	6/8/09	8:13 PM
efb2d4fe-f98e-4a46-93f4-69eb62090608	6/8/09	8:04 PM
9fddf8dd-cc9e-4ca1-9aab-6644f2090507	5/7/09	7:56 PM
9abfd0fa-7f0d-4fe3-92d3-c06af2090421	4/21/09	8:19 AM
fe88a8b6-4f0c-4000-b793-3583c2090319	3/19/09	2:48 PM
d92ebf79-9858-450d-9868-0fe042090211	2/11/09	9:19 AM
a67d53d1-2674-42b5-9f37-029132090211	2/11/09	8:51 AM
c7fe5394-a34d-4f4c-9409-4c66f2090209	2/9/09	2:22 PM
36e53db2-c93b-42a0-b59d-817322090209	2/9/09	2:18 PM
940db996-19bd-4a63-aa86-4e5ac2081229	12/29/08	3:02 PM
67e34d92-1203-4bb7-855b-ed5920081119	11/19/08	10:31 AM
e191213d-f39d-4ac0-bbe5-7f5f20081119	11/19/08	10:30 AM



[@ _VEC_memzero] – Firefox 3.6a1pre Crash Report – Report ID: d425189f-148b-461c-ba3d-c6c0b2090721
http://crash-stats.mozilla.com/report/index/d425189f-148b-461c-ba3d-c6c0b209
Creative Commons
mozilla
crash reports
Crash ID or Signature
Firefox
Thunderbird
More
Trend Reports
Advanced Search
Get Help
Details
Modules
Raw Dump
Done

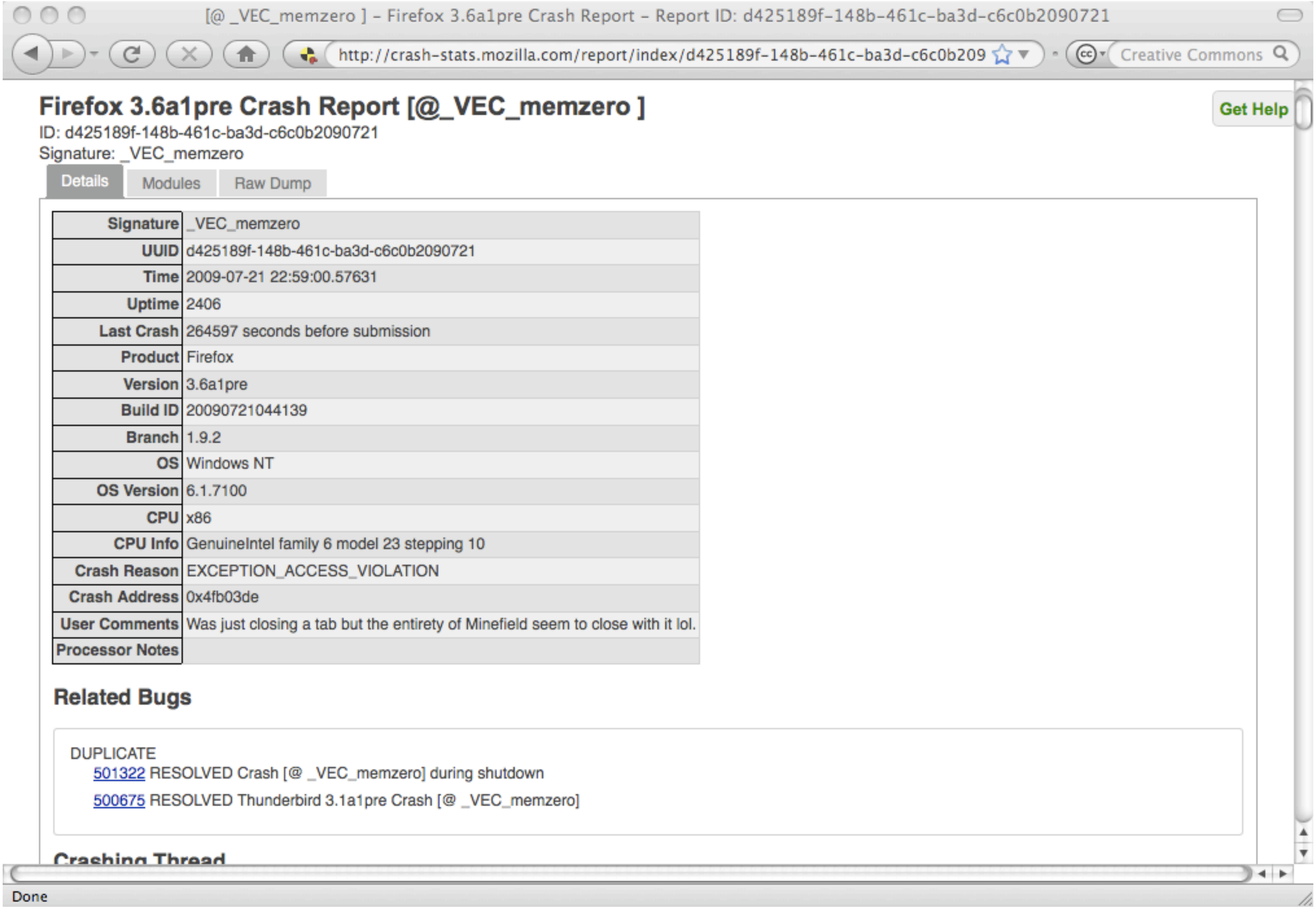
Firefox 3.6a1pre Crash Report [@_VEC_memzero]

ID: d425189f-148b-461c-ba3d-c6c0b2090721
Signature: _VEC_memzero

Signature	_VEC_memzero
UUID	d425189f-148b-461c-ba3d-c6c0b2090721
Time	2009-07-21 22:59:00.57631
Uptime	2406
Last Crash	264597 seconds before submission
Product	Firefox
Version	3.6a1pre
Build ID	20090721044139
Branch	1.9.2
OS	Windows NT
OS Version	6.1.7100
CPU	x86
CPU Info	GenuineIntel family 6 model 23 stepping 10
Crash Reason	EXCEPTION_ACCESS_VIOLATION
Crash Address	0x4fb03de
User Comments	Was just closing a tab but the entirety of Minefield seem to close with it lol.
Processor Notes	

Related Bugs

Friday, July 24, 2009


[@ _VEC_memzero] - Firefox 3.6a1pre Crash Report - Report ID: d425189f-148b-461c-ba3d-c6c0b2090721
http://crash-stats.mozilla.com/report/index/d425189f-148b-461c-ba3d-c6c0b209
Creative Commons
Get Help
Details
Modules
Raw Dump
Done

Firefox 3.6a1pre Crash Report [@_VEC_memzero]

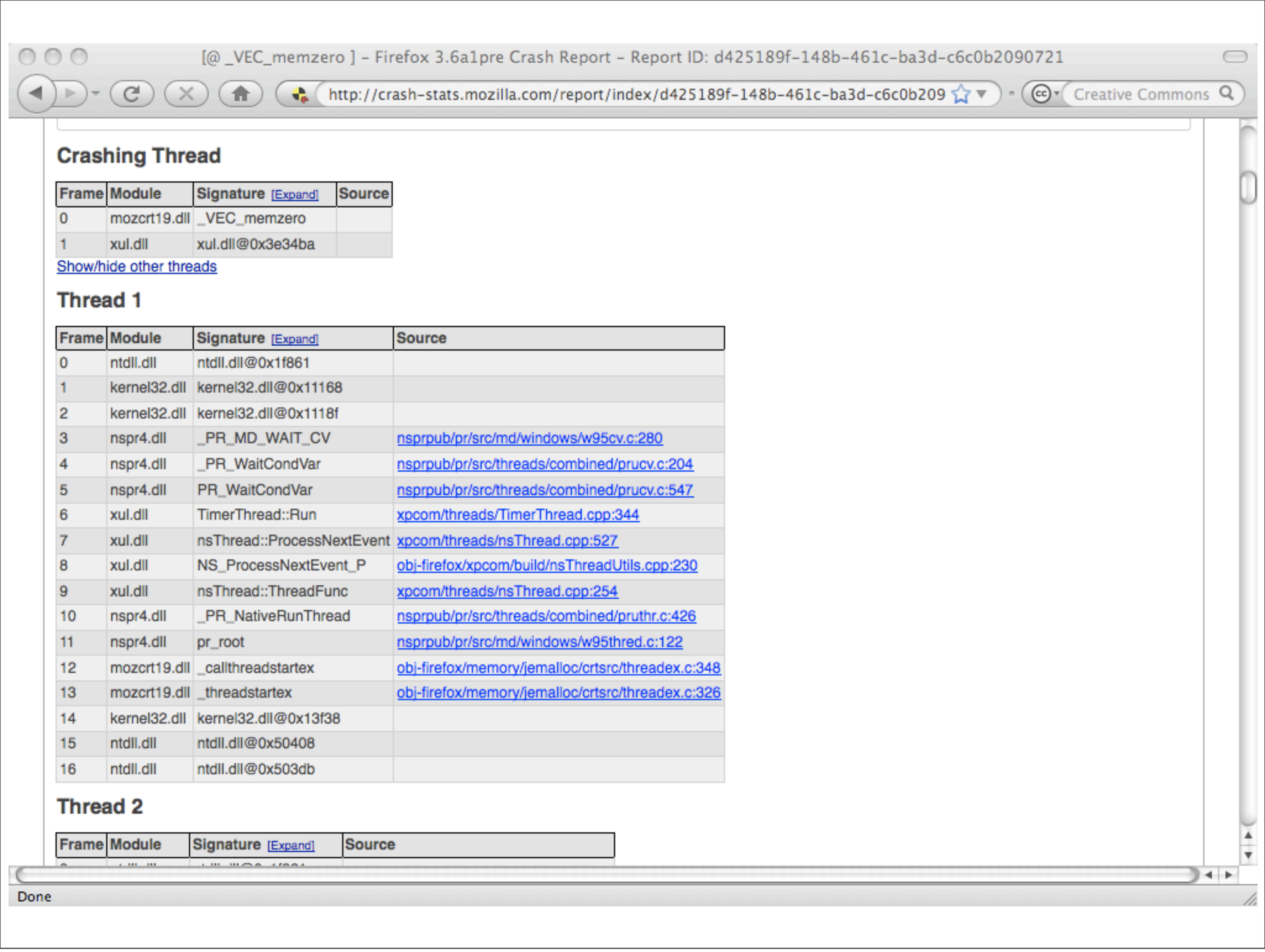
ID: d425189f-148b-461c-ba3d-c6c0b2090721
Signature: _VEC_memzero

Signature	_VEC_memzero
UUID	d425189f-148b-461c-ba3d-c6c0b2090721
Time	2009-07-21 22:59:00.57631
Uptime	2406
Last Crash	264597 seconds before submission
Product	Firefox
Version	3.6a1pre
Build ID	20090721044139
Branch	1.9.2
OS	Windows NT
OS Version	6.1.7100
CPU	x86
CPU Info	GenuineIntel family 6 model 23 stepping 10
Crash Reason	EXCEPTION_ACCESS_VIOLATION
Crash Address	0x4fb03de
User Comments	Was just closing a tab but the entirety of Minefield seem to close with it lol.
Processor Notes	

Related Bugs

DUPLICATE
501322 RESOLVED Crash [@ _VEC_memzero] during shutdown
500675 RESOLVED Thunderbird 3.1a1pre Crash [@ _VEC_memzero]

Crashing Thread



[@ _VEC_memzero] - Firefox 3.6a1pre Crash Report - Report ID: d425189f-148b-461c-ba3d-c6c0b2090721

http://crash-stats.mozilla.com/report/index/d425189f-148b-461c-ba3d-c6c0b209

Crashing Thread

Frame	Module	Signature [Expand]	Source
0	mozcrt19.dll	_VEC_memzero	
1	xul.dll	xul.dll@0x3e34ba	

Show/hide other threads

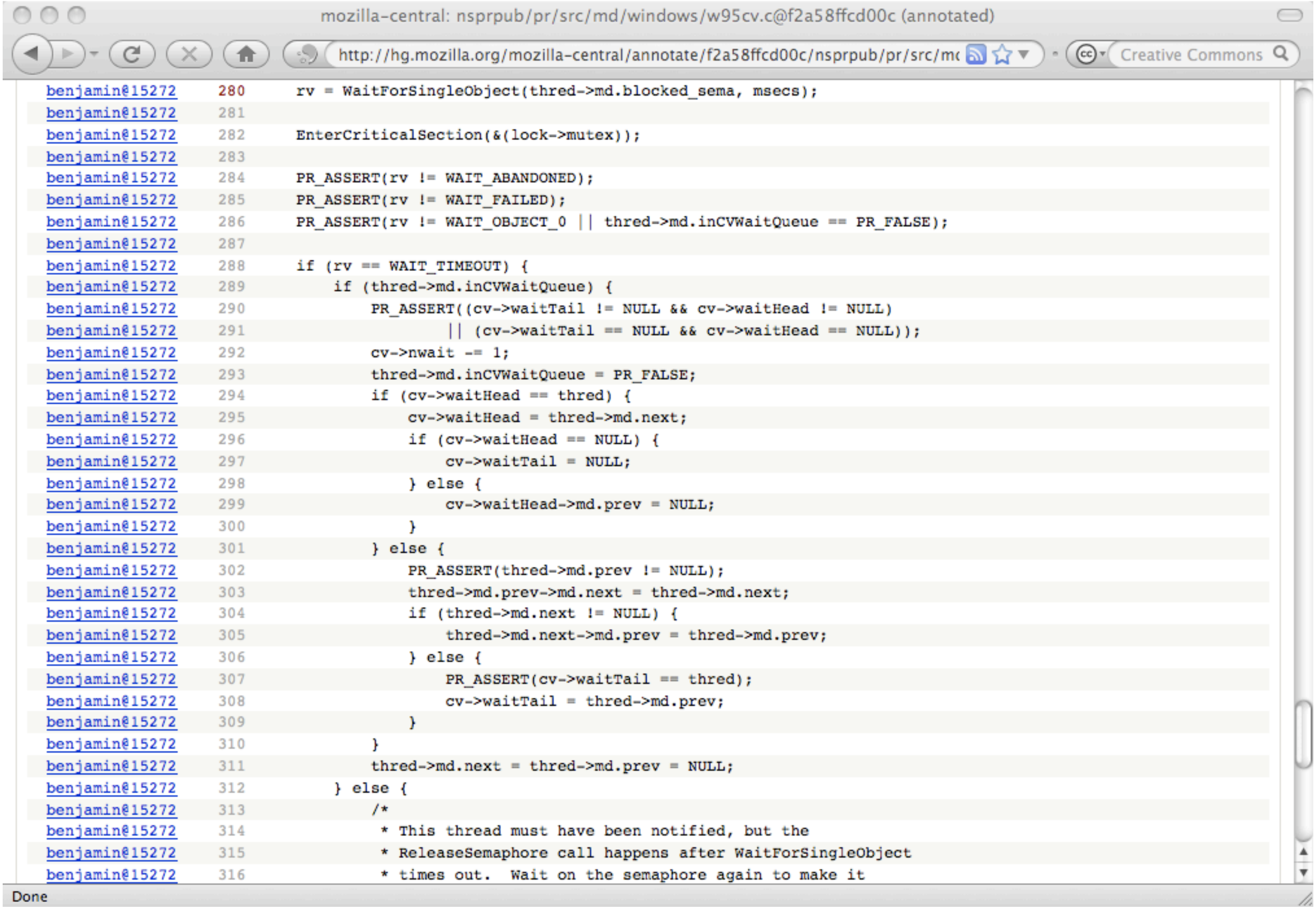
Thread 1

Frame	Module	Signature [Expand]	Source
0	ntdll.dll	ntdll.dll@0x1f861	
1	kernel32.dll	kernel32.dll@0x11168	
2	kernel32.dll	kernel32.dll@0x1118f	
3	nspr4.dll	_PR_MD_WAIT_CV	nsprpub/pr/src/md/windows/w95cv.c:280
4	nspr4.dll	_PR_WaitCondVar	nsprpub/pr/src/threads/combined/prucv.c:204
5	nspr4.dll	PR_WaitCondVar	nsprpub/pr/src/threads/combined/prucv.c:547
6	xul.dll	TimerThread::Run	xpcom/threads/TimerThread.cpp:344
7	xul.dll	nsThread::ProcessNextEvent	xpcom/threads/nsThread.cpp:527
8	xul.dll	NS_ProcessNextEvent_P	obj-firefox/xpcom/build/nsThreadUtils.cpp:230
9	xul.dll	nsThread::ThreadFunc	xpcom/threads/nsThread.cpp:254
10	nspr4.dll	_PR_NativeRunThread	nsprpub/pr/src/threads/combined/pruthr.c:426
11	nspr4.dll	pr_root	nsprpub/pr/src/md/windows/w95thred.c:122
12	mozcrt19.dll	_callthreadstartex	obj-firefox/memory/jemalloc/crtsrc/threadex.c:348
13	mozcrt19.dll	_threadstartex	obj-firefox/memory/jemalloc/crtsrc/threadex.c:326
14	kernel32.dll	kernel32.dll@0x13f38	
15	ntdll.dll	ntdll.dll@0x50408	
16	ntdll.dll	ntdll.dll@0x503db	

Thread 2

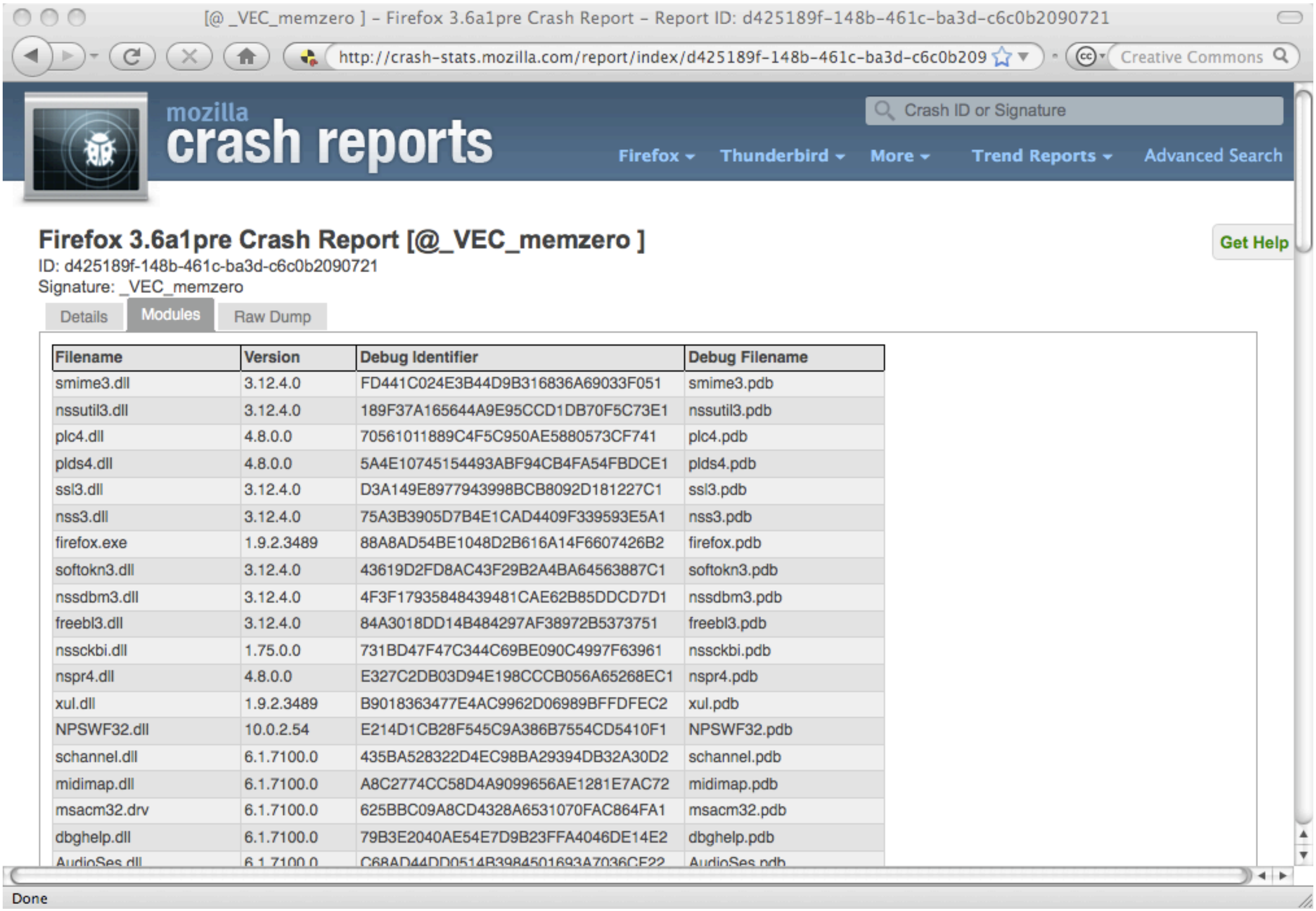
Frame	Module	Signature [Expand]	Source

Friday, July 24, 2009



mozilla-central: nsprpub/pr/src/md/windows/w95cv.c@f2a58ffcd00c (annotated)
http://hg.mozilla.org/mozilla-central/annotate/f2a58ffcd00c/nsprpub/pr/src/mc
Creative Commons
Done

```
benjamin@15272      rv = WaitForSingleObject(thred->md.blocked_sema, msecs);
benjamin@15272
benjamin@15272      EnterCriticalSection(&(lock->mutex));
benjamin@15272
benjamin@15272      PR_ASSERT(rv != WAIT_ABANDONED);
benjamin@15272      PR_ASSERT(rv != WAIT_FAILED);
benjamin@15272      PR_ASSERT(rv != WAIT_OBJECT_0 || thred->md.inCVWaitQueue == PR_FALSE);
benjamin@15272
benjamin@15272      if (rv == WAIT_TIMEOUT) {
benjamin@15272          if (thred->md.inCVWaitQueue) {
benjamin@15272              PR_ASSERT((cv->waitTail != NULL && cv->waitHead != NULL)
benjamin@15272                      || (cv->waitTail == NULL && cv->waitHead == NULL));
benjamin@15272              cv->nwait -= 1;
benjamin@15272              thred->md.inCVWaitQueue = PR_FALSE;
benjamin@15272              if (cv->waitHead == thred) {
benjamin@15272                  cv->waitHead = thred->md.next;
benjamin@15272                  if (cv->waitHead == NULL) {
benjamin@15272                      cv->waitTail = NULL;
benjamin@15272                  } else {
benjamin@15272                      cv->waitHead->md.prev = NULL;
benjamin@15272                  }
benjamin@15272              } else {
benjamin@15272                  PR_ASSERT(thred->md.prev != NULL);
benjamin@15272                  thred->md.prev->md.next = thred->md.next;
benjamin@15272                  if (thred->md.next != NULL) {
benjamin@15272                      thred->md.next->md.prev = thred->md.prev;
benjamin@15272                  } else {
benjamin@15272                      PR_ASSERT(cv->waitTail == thred);
benjamin@15272                      cv->waitTail = thred->md.prev;
benjamin@15272                  }
benjamin@15272              }
benjamin@15272              thred->md.next = thred->md.prev = NULL;
benjamin@15272          } else {
benjamin@15272              /*
benjamin@15272               * This thread must have been notified, but the
benjamin@15272               * ReleaseSemaphore call happens after WaitForSingleObject
benjamin@15272               * times out.  Wait on the semaphore again to make it
```



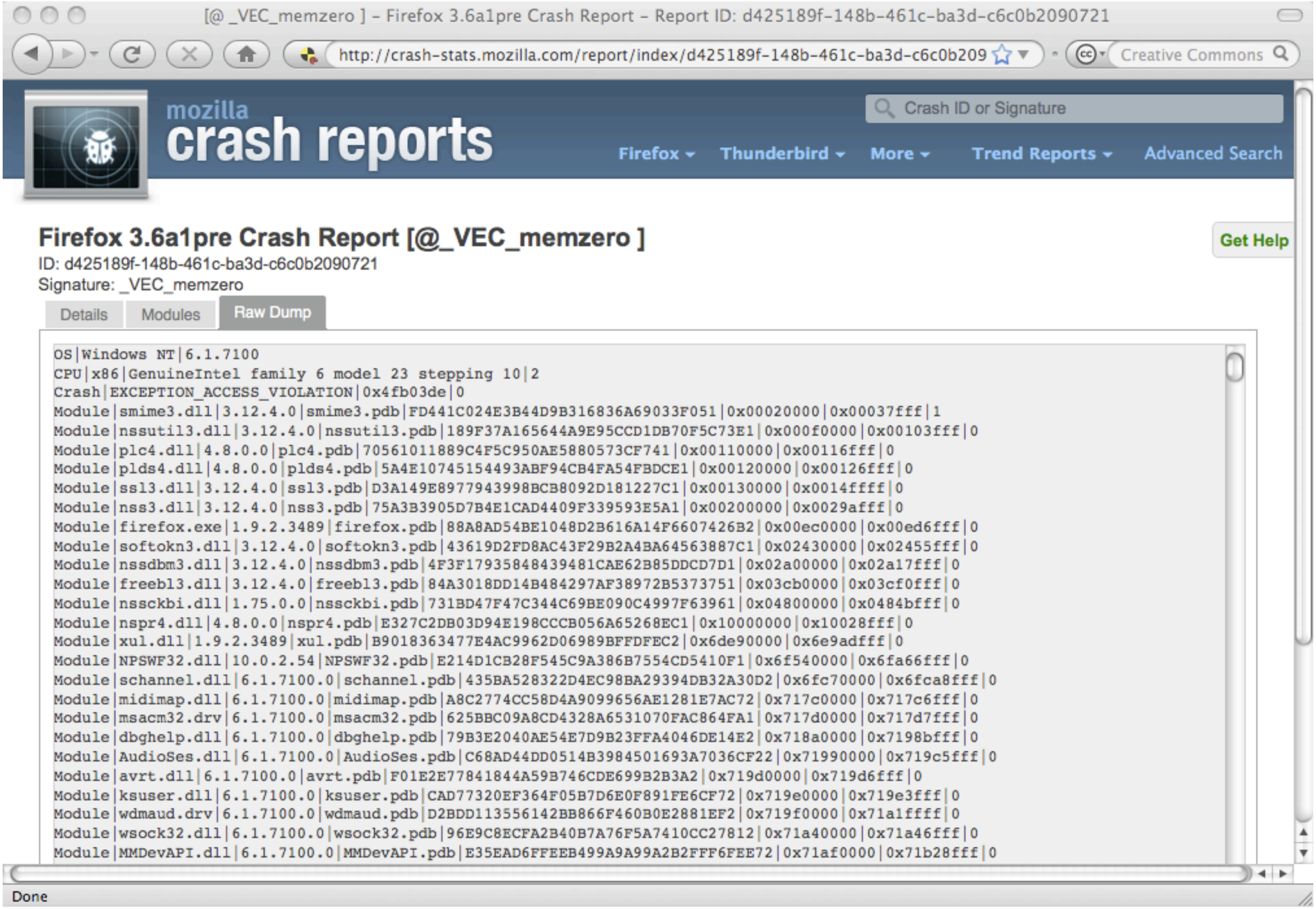
[@ _VEC_memzero] - Firefox 3.6a1pre Crash Report - Report ID: d425189f-148b-461c-ba3d-c6c0b2090721
http://crash-stats.mozilla.com/report/index/d425189f-148b-461c-ba3d-c6c0b209
Creative Commons
mozilla
crash reports
Crash ID or Signature
Firefox
Thunderbird
More
Trend Reports
Advanced Search
Get Help
Details
Modules
Raw Dump
Done

Firefox 3.6a1pre Crash Report [@_VEC_memzero]

ID: d425189f-148b-461c-ba3d-c6c0b2090721

Signature: _VEC_memzero

Filename	Version	Debug Identifier	Debug Filename
smime3.dll	3.12.4.0	FD441C024E3B44D9B316836A69033F051	smime3.pdb
nssutil3.dll	3.12.4.0	189F37A165644A9E95CCD1DB70F5C73E1	nssutil3.pdb
plc4.dll	4.8.0.0	70561011889C4F5C950AE5880573CF741	plc4.pdb
plds4.dll	4.8.0.0	5A4E10745154493ABF94CB4FA54FBDCE1	plds4.pdb
ssl3.dll	3.12.4.0	D3A149E8977943998BCB8092D181227C1	ssl3.pdb
nss3.dll	3.12.4.0	75A3B3905D7B4E1CAD4409F339593E5A1	nss3.pdb
firefox.exe	1.9.2.3489	88A8AD54BE1048D2B616A14F6607426B2	firefox.pdb
softokn3.dll	3.12.4.0	43619D2FD8AC43F29B2A4BA64563887C1	softokn3.pdb
nssdbm3.dll	3.12.4.0	4F3F17935848439481CAE62B85DDCD7D1	nssdbm3.pdb
freebl3.dll	3.12.4.0	84A3018DD14B484297AF38972B5373751	freebl3.pdb
nssckbi.dll	1.75.0.0	731BD47F47C344C69BE090C4997F63961	nssckbi.pdb
nspr4.dll	4.8.0.0	E327C2DB03D94E198CCCB056A65268EC1	nspr4.pdb
xul.dll	1.9.2.3489	B9018363477E4AC9962D06989BFFDFEC2	xul.pdb
NPSWF32.dll	10.0.2.54	E214D1CB28F545C9A386B7554CD5410F1	NPSWF32.pdb
schannel.dll	6.1.7100.0	435BA528322D4EC98BA29394DB32A30D2	schannel.pdb
midimap.dll	6.1.7100.0	A8C2774CC58D4A9099656AE1281E7AC72	midimap.pdb
msacm32.drv	6.1.7100.0	625BBC09A8CD4328A6531070FAC864FA1	msacm32.pdb
dbghelp.dll	6.1.7100.0	79B3E2040AE54E7D9B23FFA4046DE14E2	dbghelp.pdb
AudioSes.dll	6.1.7100.0	C68AD44DD0514B3984501693A7036CE22	AudioSes.pdb


[@ _VEC_memzero] - Firefox 3.6a1pre Crash Report - Report ID: d425189f-148b-461c-ba3d-c6c0b2090721
http://crash-stats.mozilla.com/report/index/d425189f-148b-461c-ba3d-c6c0b209
Creative Commons
mozilla
crash reports
Crash ID or Signature
Firefox
Thunderbird
More
Trend Reports
Advanced Search
Get Help
Details
Modules
Raw Dump
Done

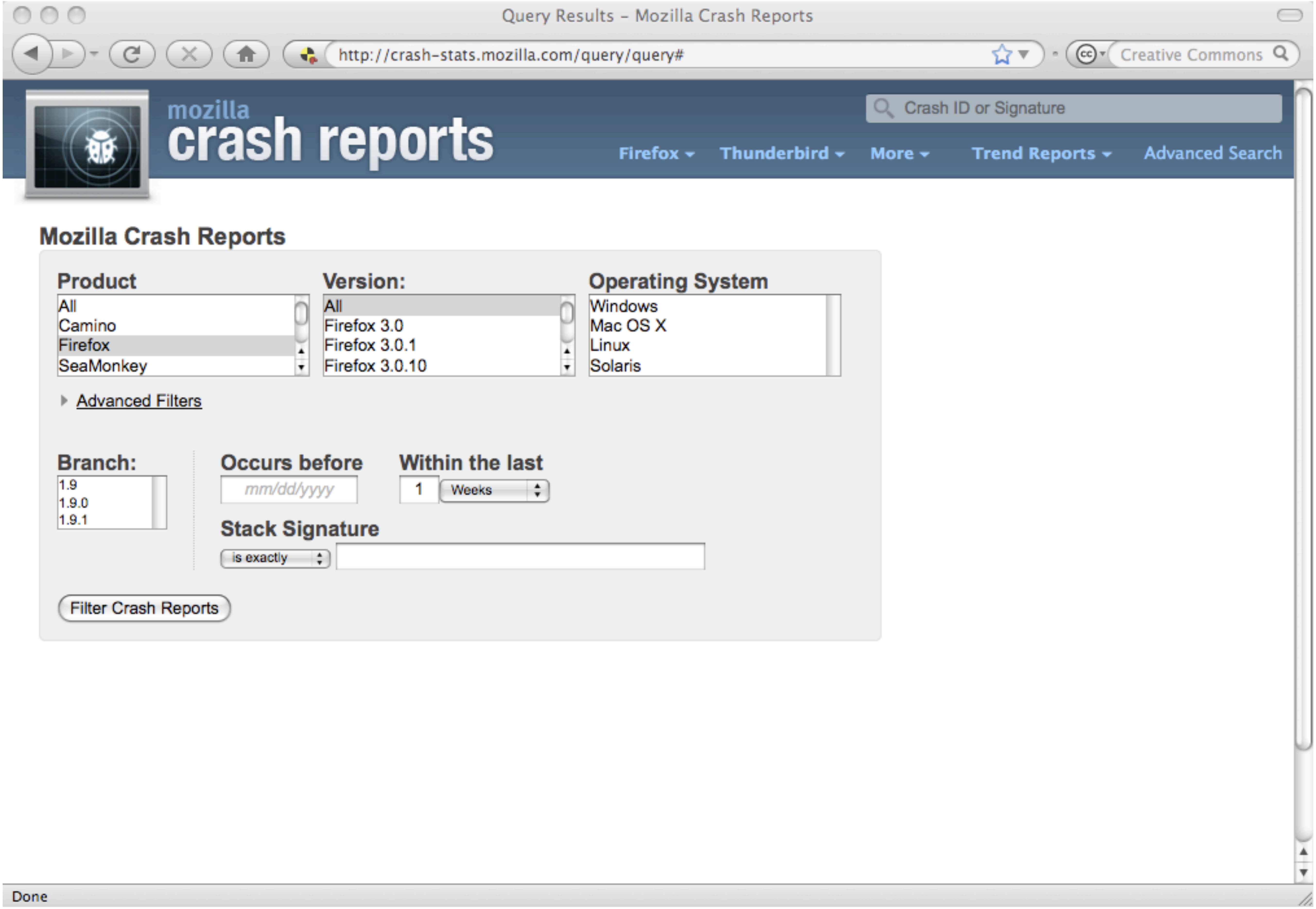
Firefox 3.6a1pre Crash Report [@_VEC_memzero]

ID: d425189f-148b-461c-ba3d-c6c0b2090721

Signature: _VEC_memzero

```
OS|Windows NT|6.1.7100
CPU|x86|GenuineIntel family 6 model 23 stepping 10|2
Crash|EXCEPTION_ACCESS_VIOLATION|0x4fb03de|0
Module|smime3.dll|3.12.4.0|smime3.pdb|FD441C024E3B44D9B316836A69033F051|0x00020000|0x00037fff|1
Module|nssutil3.dll|3.12.4.0|nssutil3.pdb|189F37A165644A9E95CCD1DB70F5C73E1|0x000f0000|0x00103fff|0
Module|plc4.dll|4.8.0.0|plc4.pdb|70561011889C4F5C950AE5880573CF741|0x00110000|0x00116fff|0
Module|plds4.dll|4.8.0.0|plds4.pdb|5A4E10745154493ABF94CB4FA54FBDCE1|0x00120000|0x00126fff|0
Module|ssl3.dll|3.12.4.0|ssl3.pdb|D3A149E8977943998BCB8092D181227C1|0x00130000|0x0014ffff|0
Module|nss3.dll|3.12.4.0|nss3.pdb|75A3B3905D7B4E1CAD4409F339593E5A1|0x00200000|0x0029afff|0
Module|firefox.exe|1.9.2.3489|firefox.pdb|88A8AD54BE1048D2B616A14F6607426B2|0x00ec0000|0x00ed6fff|0
Module|softokn3.dll|3.12.4.0|softokn3.pdb|43619D2FD8AC43F29B2A4BA64563887C1|0x02430000|0x02455fff|0
Module|nssdbm3.dll|3.12.4.0|nssdbm3.pdb|4F3F17935848439481CAE62B85DDCD7D1|0x02a00000|0x02a17fff|0
Module|freebl3.dll|3.12.4.0|freebl3.pdb|84A3018DD14B484297AF38972B5373751|0x03cb0000|0x03cf0fff|0
Module|nssckbi.dll|1.75.0.0|nssckbi.pdb|731BD47F47C344C69BE090C4997F63961|0x04800000|0x0484bfff|0
Module|nspr4.dll|4.8.0.0|nspr4.pdb|E327C2DB03D94E198CCCB056A65268EC1|0x10000000|0x10028fff|0
Module|xul.dll|1.9.2.3489|xul.pdb|B9018363477E4AC9962D06989BFFDFEC2|0x6de90000|0x6e9adfff|0
Module|NPSWF32.dll|10.0.2.54|NPSWF32.pdb|E214D1CB28F545C9A386B7554CD5410F1|0x6f540000|0x6fa66fff|0
Module|schannel.dll|6.1.7100.0|schannel.pdb|435BA528322D4EC98BA29394DB32A30D2|0x6fc70000|0x6fca8fff|0
Module|midimap.dll|6.1.7100.0|midimap.pdb|A8C2774CC58D4A9099656AE1281E7AC72|0x717c0000|0x717c6fff|0
Module|msacm32.drv|6.1.7100.0|msacm32.pdb|625BBC09A8CD4328A6531070FAC864FA1|0x717d0000|0x717d7fff|0
Module|dbghelp.dll|6.1.7100.0|dbghelp.pdb|79B3E2040AE54E7D9B23FFA4046DE14E2|0x718a0000|0x7198bfff|0
Module|AudioSes.dll|6.1.7100.0|AudioSes.pdb|C68AD44DD0514B3984501693A7036CF22|0x71990000|0x719c5fff|0
Module|avrt.dll|6.1.7100.0|avrt.pdb|F01E2E77841844A59B746CDE699B2B3A2|0x719d0000|0x719d6fff|0
Module|ksuser.dll|6.1.7100.0|ksuser.pdb|CAD77320EF364F05B7D6E0F891FE6CF72|0x719e0000|0x719e3fff|0
Module|wdmaud.drv|6.1.7100.0|wdmaud.pdb|D2BDD113556142BB866F460B0E2881EF2|0x719f0000|0x71a1ffff|0
Module|wsock32.dll|6.1.7100.0|wsock32.pdb|96E9C8ECFA2B40B7A76F5A7410CC27812|0x71a40000|0x71a46fff|0
Module|MMDevAPI.dll|6.1.7100.0|MMDevAPI.pdb|E35EAD6FFEEB499A9A99A2B2FFF6FEE72|0x71af0000|0x71b28fff|0
```

Search



Mozilla Crash Reports

Product: All, Camino, Firefox (selected), SeaMonkey

Version: All (selected), Firefox 3.0, Firefox 3.0.1, Firefox 3.0.10

Operating System: Windows, Mac OS X, Linux, Solaris

Advanced Filters

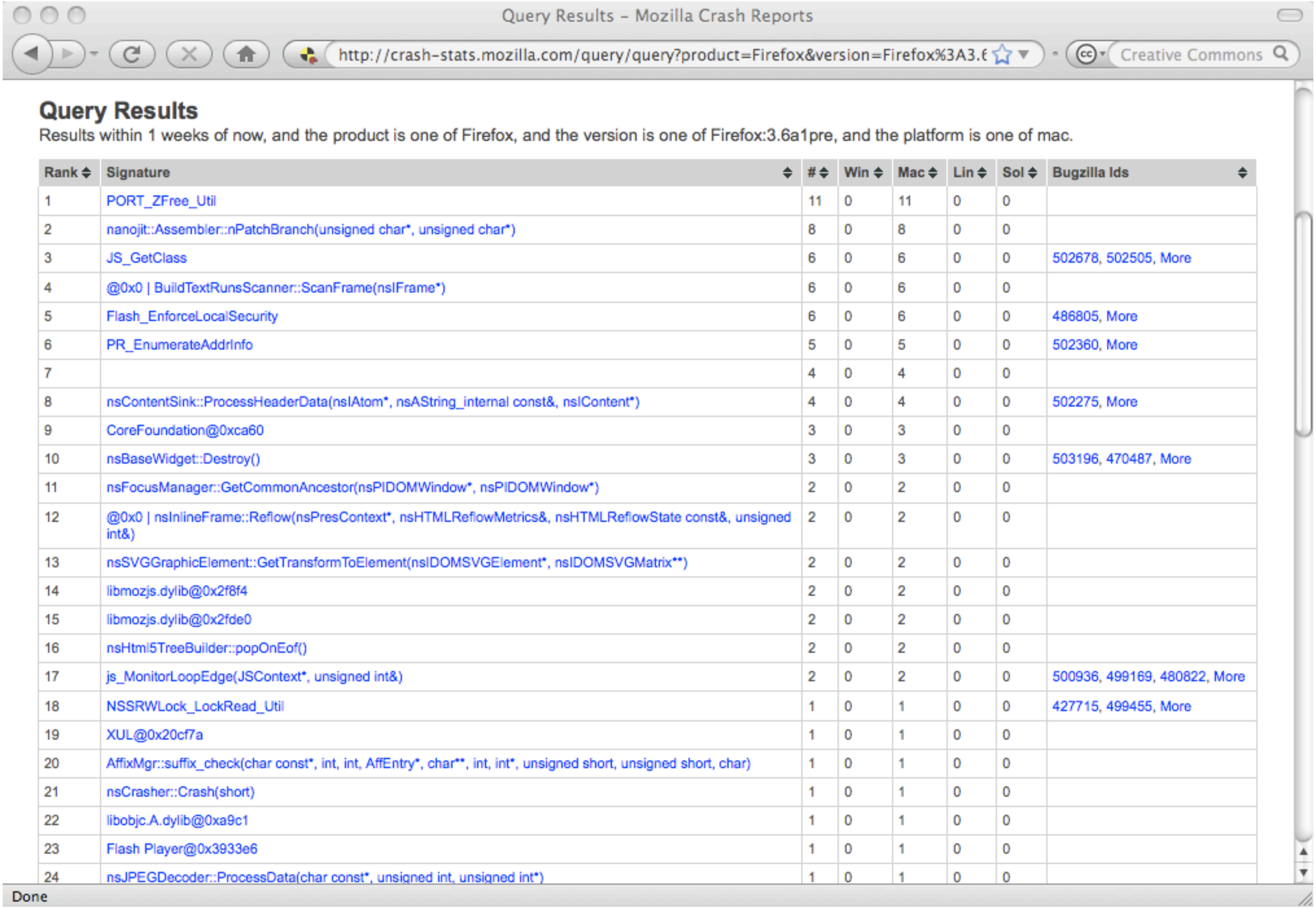
Branch: 1.9, 1.9.0, 1.9.1

Occurs before: mm/dd/yyyy

Within the last: 1 Weeks

Stack Signature: is exactly

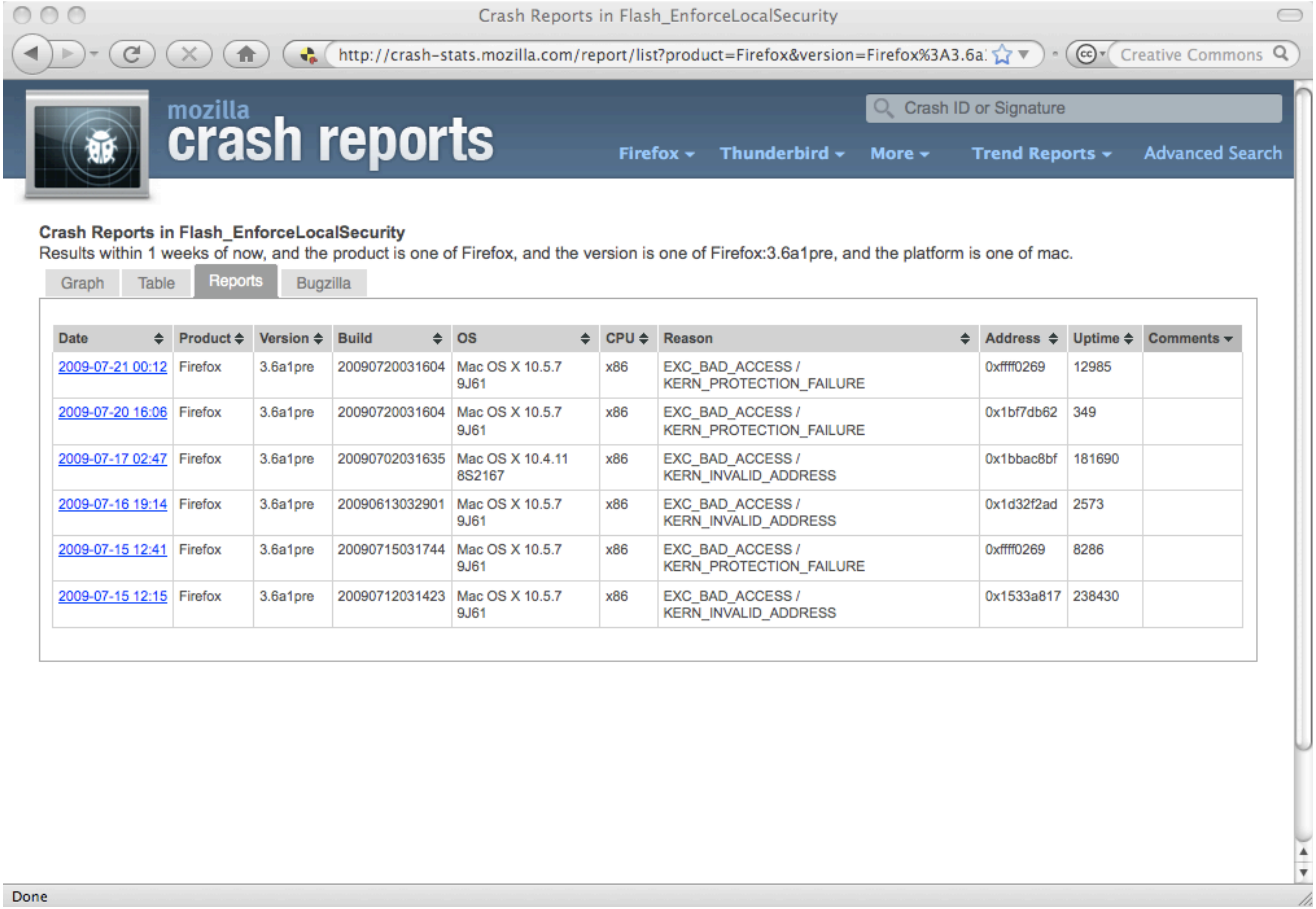
Friday, July 24, 2009


Query Results - Mozilla Crash Reports
http://crash-stats.mozilla.com/query/query?product=Firefox&version=Firefox%3A3.6
Creative Commons
Done

Query Results

Results within 1 weeks of now, and the product is one of Firefox, and the version is one of Firefox:3.6a1pre, and the platform is one of mac.

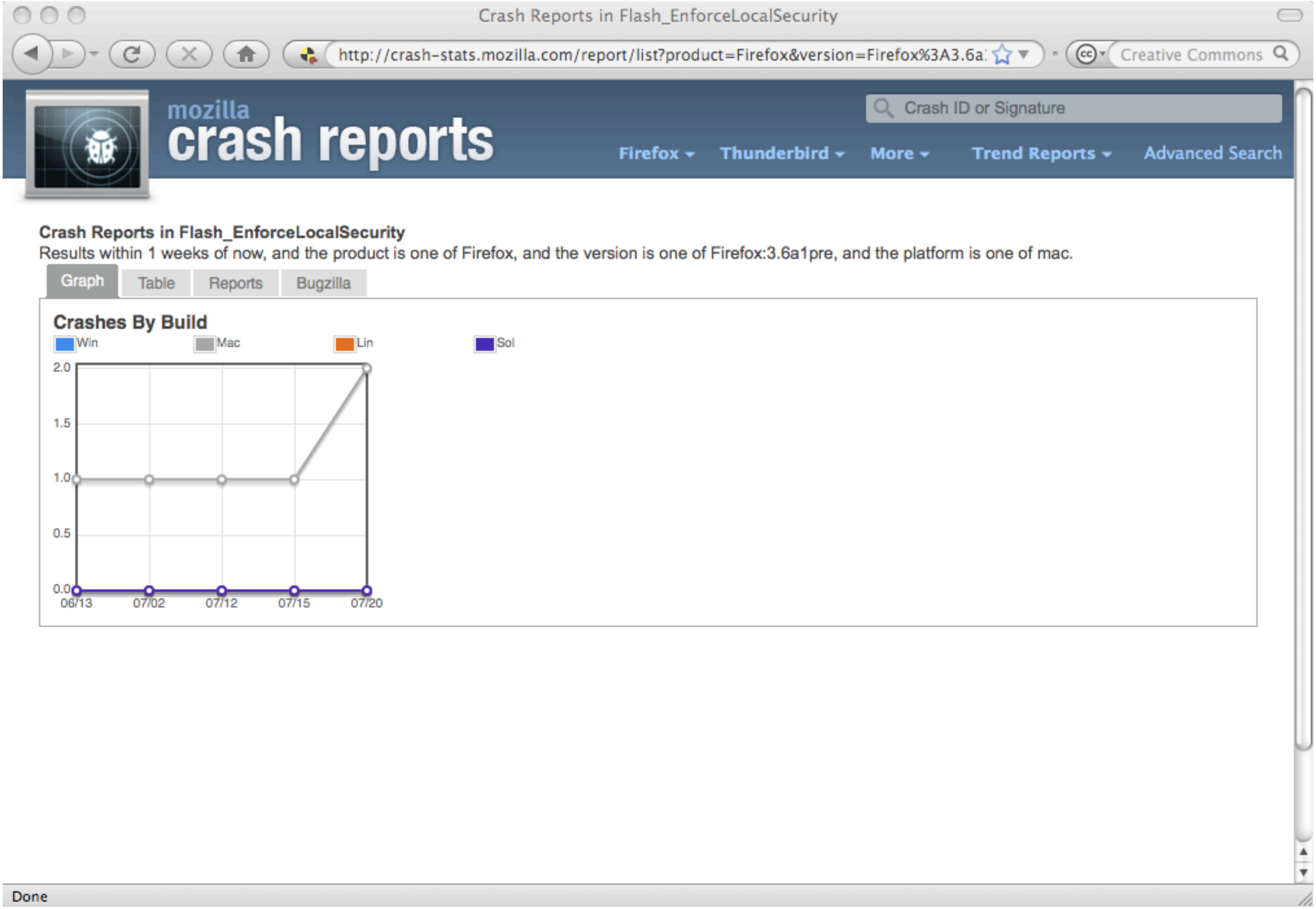
Rank	Signature	#	Win	Mac	Lin	Sol	Bugzilla Ids
1	PORT_ZFree_Util	11	0	11	0	0	
2	nanojit::Assembler::nPatchBranch(unsigned char*, unsigned char*)	8	0	8	0	0	
3	JS_GetClass	6	0	6	0	0	502678, 502505, More
4	@0x0 | BuildTextRunsScanner::ScanFrame(nsIFrame*)	6	0	6	0	0	
5	Flash_EnforceLocalSecurity	6	0	6	0	0	486805, More
6	PR_EnumerateAddrInfo	5	0	5	0	0	502360, More
7		4	0	4	0	0	
8	nsContentSink::ProcessHeaderData(nsIAtom*, nsAString_internal const&, nsIContent*)	4	0	4	0	0	502275, More
9	CoreFoundation@0xca60	3	0	3	0	0	
10	nsBaseWidget::Destroy()	3	0	3	0	0	503196, 470487, More
11	nsFocusManager::GetCommonAncestor(nsPIDOMWindow*, nsPIDOMWindow*)	2	0	2	0	0	
12	@0x0 | nsInlineFrame::Reflow(nsPresContext*, nsHTMLReflowMetrics&, nsHTMLReflowState const&, unsigned int&)	2	0	2	0	0	
13	nsSVGGraphicElement::GetTransformToElement(nsIDOMSVGElement*, nsIDOMSVGMatrix**)	2	0	2	0	0	
14	libmozjs.dylib@0x2f8f4	2	0	2	0	0	
15	libmozjs.dylib@0x2fde0	2	0	2	0	0	
16	nsHtml5TreeBuilder::popOnEof()	2	0	2	0	0	
17	js_MonitorLoopEdge(JSContext*, unsigned int&)	2	0	2	0	0	500936, 499169, 480822, More
18	NSSRWLock_LockRead_Util	1	0	1	0	0	427715, 499455, More
19	XUL@0x20cf7a	1	0	1	0	0	
20	AffixMgr::suffix_check(char const*, int, int, AffEntry*, char**, int, int*, unsigned short, unsigned short, char)	1	0	1	0	0	
21	nsCrasher::Crash(short)	1	0	1	0	0	
22	libobjc.A.dylib@0xa9c1	1	0	1	0	0	
23	Flash Player@0x3933e6	1	0	1	0	0	
24	nsJPEGDecoder::ProcessData(char const*, unsigned int, unsigned int*)	1	0	1	0	0	



Crash Reports in Flash_EnforceLocalSecurity

Results within 1 weeks of now, and the product is one of Firefox, and the version is one of Firefox:3.6a1pre, and the platform is one of mac.

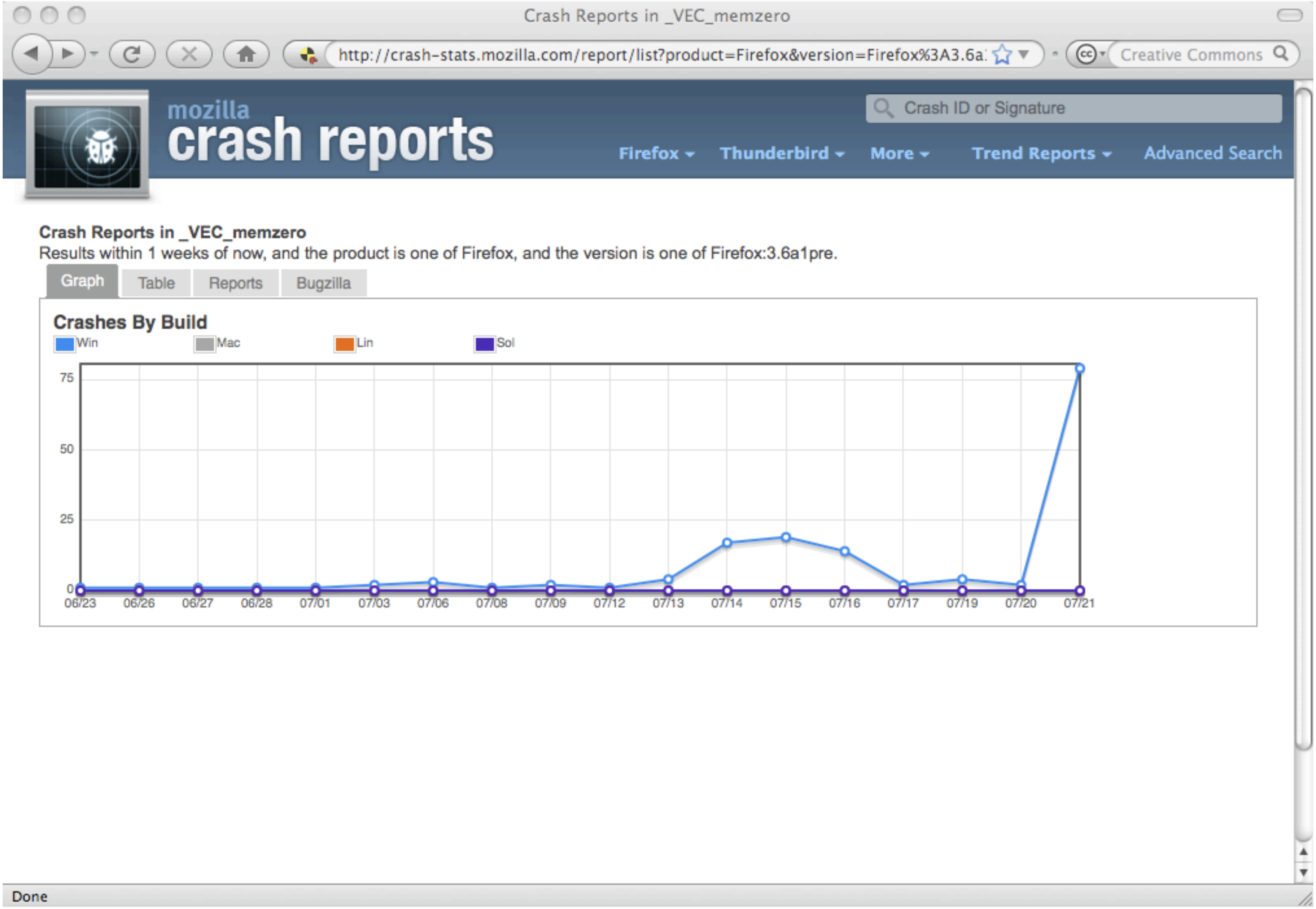
Date	Product	Version	Build	OS	CPU	Reason	Address	Uptime	Comments
2009-07-21 00:12	Firefox	3.6a1pre	20090720031604	Mac OS X 10.5.7 9J61	x86	EXC_BAD_ACCESS / KERN_PROTECTION_FAILURE	0xffff0269	12985	
2009-07-20 16:06	Firefox	3.6a1pre	20090720031604	Mac OS X 10.5.7 9J61	x86	EXC_BAD_ACCESS / KERN_PROTECTION_FAILURE	0x1bf7db62	349	
2009-07-17 02:47	Firefox	3.6a1pre	20090702031635	Mac OS X 10.4.11 8S2167	x86	EXC_BAD_ACCESS / KERN_INVALID_ADDRESS	0x1bbac8bf	181690	
2009-07-16 19:14	Firefox	3.6a1pre	20090613032901	Mac OS X 10.5.7 9J61	x86	EXC_BAD_ACCESS / KERN_INVALID_ADDRESS	0x1d32f2ad	2573	
2009-07-15 12:41	Firefox	3.6a1pre	20090715031744	Mac OS X 10.5.7 9J61	x86	EXC_BAD_ACCESS / KERN_PROTECTION_FAILURE	0xffff0269	8286	
2009-07-15 12:15	Firefox	3.6a1pre	20090712031423	Mac OS X 10.5.7 9J61	x86	EXC_BAD_ACCESS / KERN_INVALID_ADDRESS	0x1533a817	238430	



Crash Reports in Flash_EnforceLocalSecurity
http://crash-stats.mozilla.com/report/list?product=Firefox&version=Firefox%3A3.6a
Creative Commons
Crash ID or Signature
mozilla
crash reports
Firefox
Thunderbird
More
Trend Reports
Advanced Search

Crash Reports in Flash_EnforceLocalSecurity

Results within 1 weeks of now, and the product is one of Firefox, and the version is one of Firefox:3.6a1pre, and the platform is one of mac.



Crash Reports in _VEC_memzero

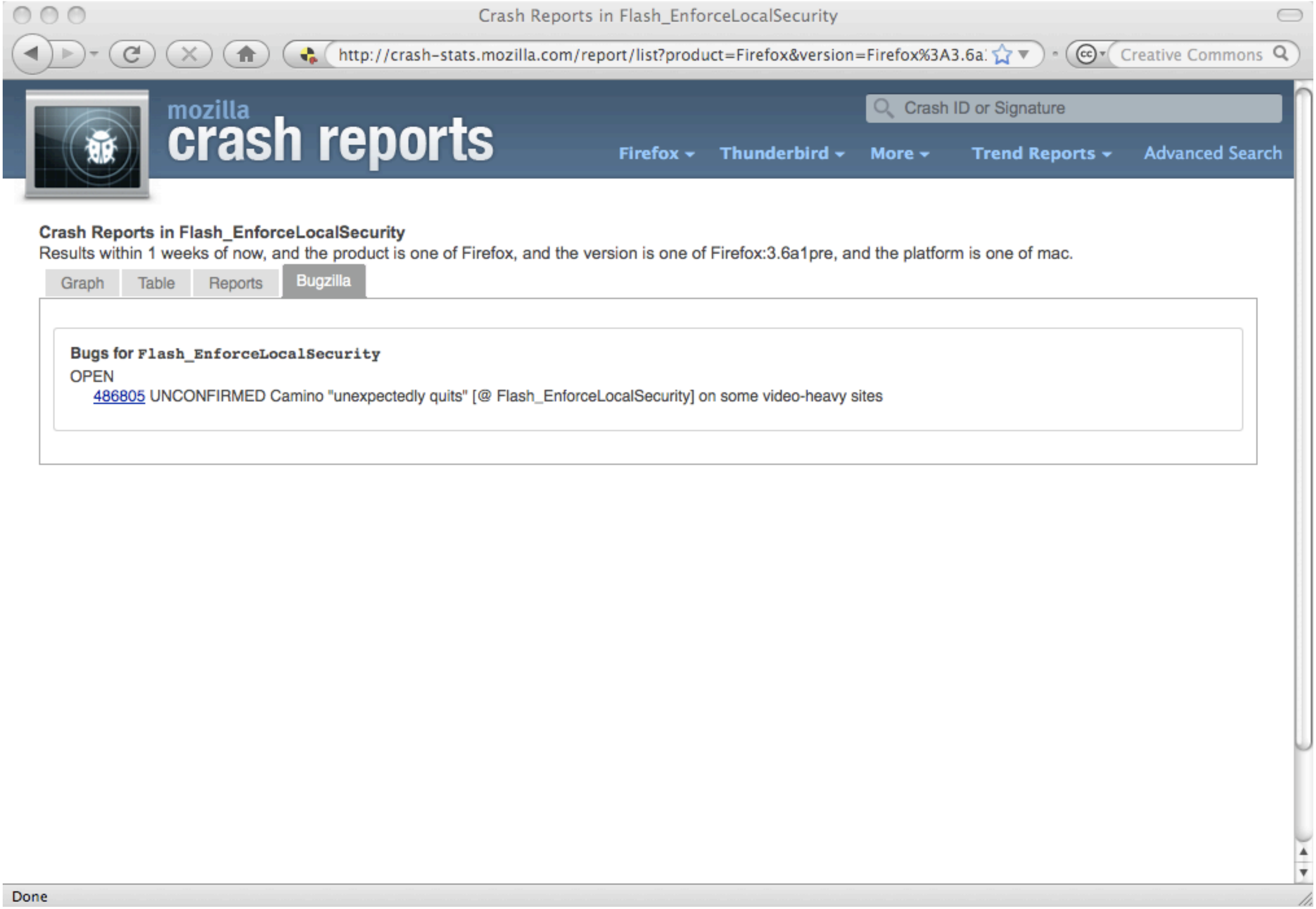
http://crash-stats.mozilla.com/report/list?product=Firefox&version=Firefox%3A3.6a

Crash Reports in _VEC_memzero

Results within 1 weeks of now, and the product is one of Firefox, and the version is one of Firefox:3.6a1pre.

Crashes By Build

Friday, July 24, 2009



Crash Reports in Flash_EnforceLocalSecurity

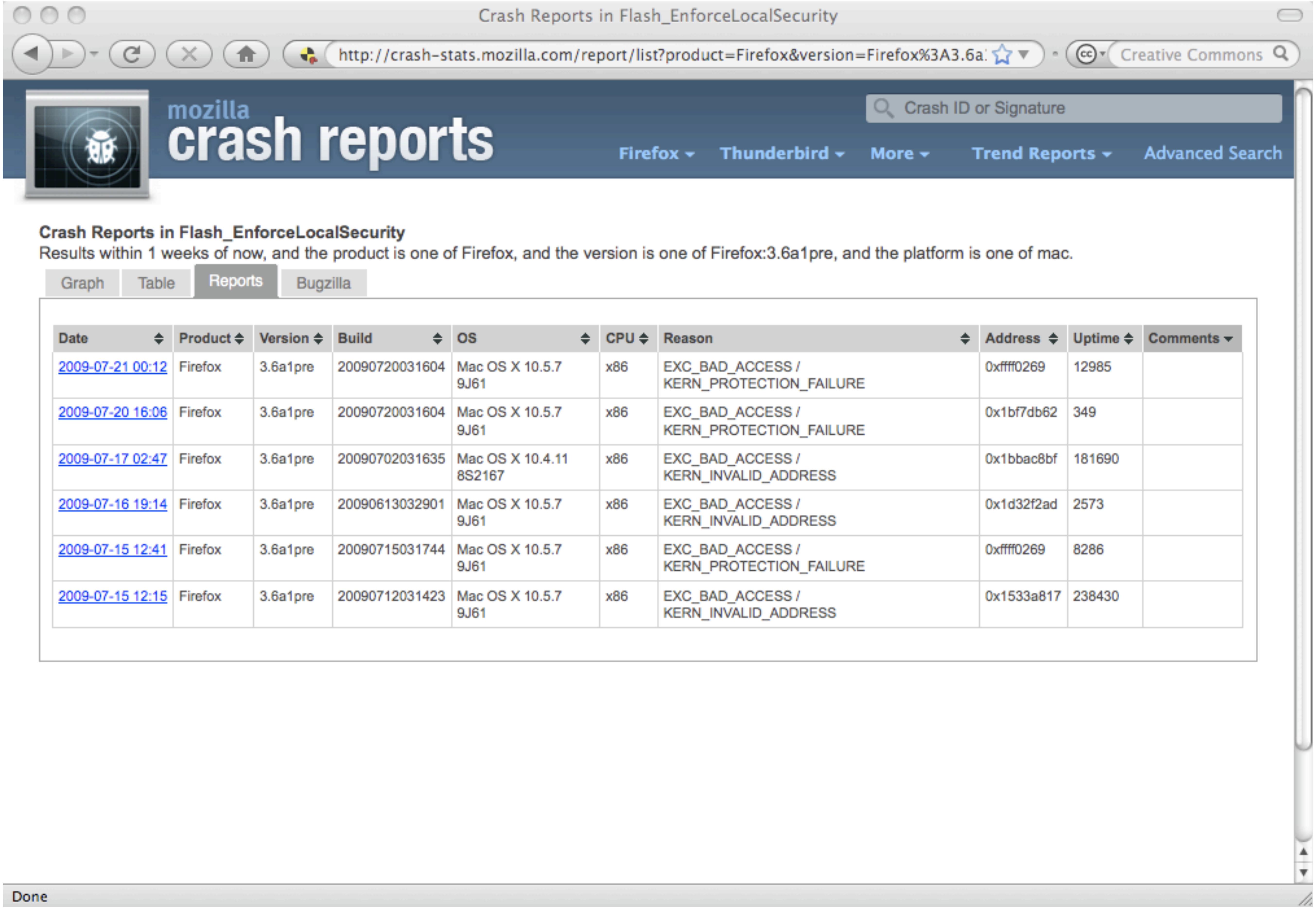
Results within 1 weeks of now, and the product is one of Firefox, and the version is one of Firefox:3.6a1pre, and the platform is one of mac.

Bugs for `Flash_EnforceLocalSecurity`

OPEN

486805 UNCONFIRMED Camino "unexpectedly quits" [@ Flash_EnforceLocalSecurity] on some video-heavy sites

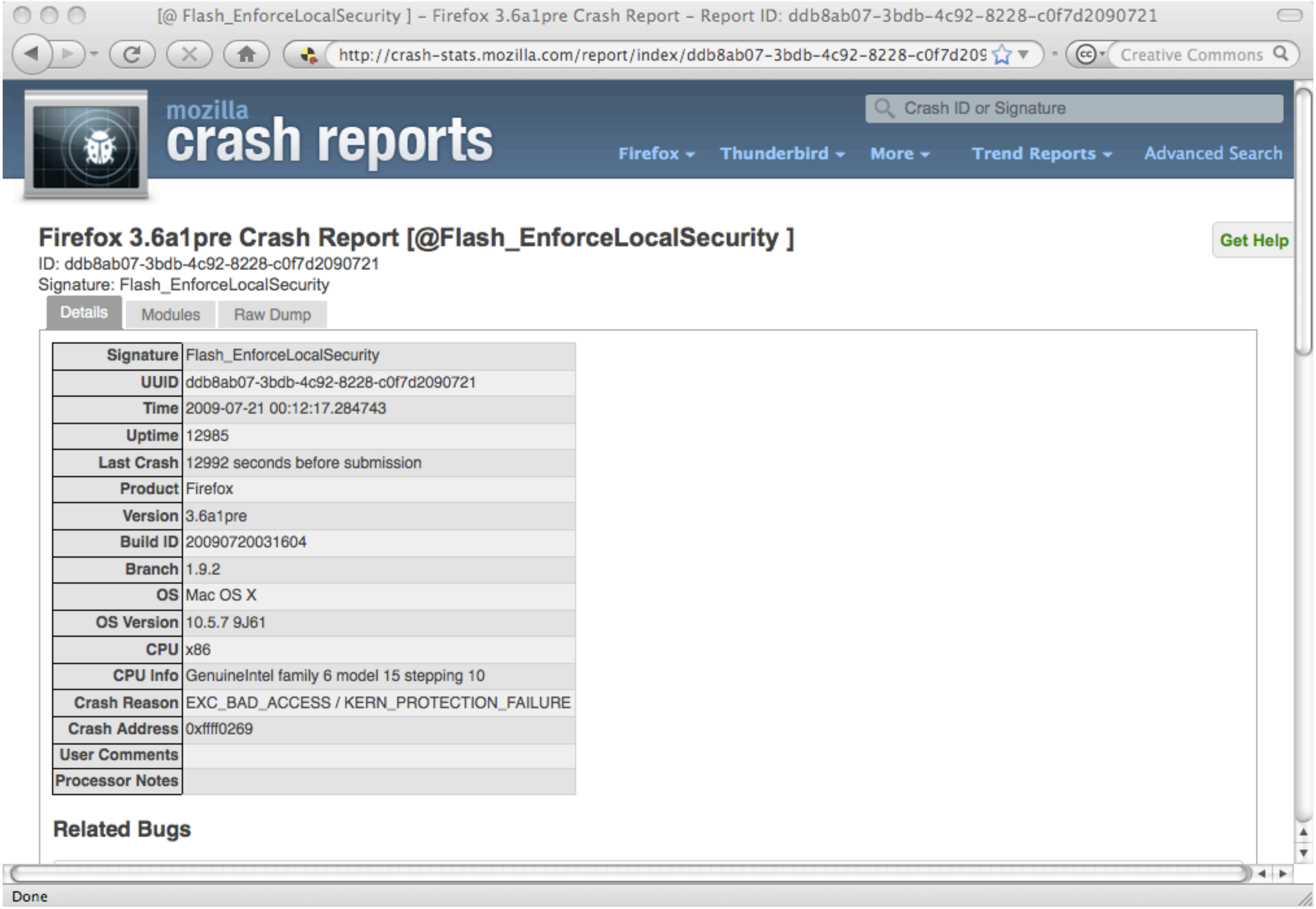
Friday, July 24, 2009



Crash Reports in Flash_EnforceLocalSecurity

Results within 1 weeks of now, and the product is one of Firefox, and the version is one of Firefox:3.6a1pre, and the platform is one of mac.

Date	Product	Version	Build	OS	CPU	Reason	Address	Uptime	Comments
2009-07-21 00:12	Firefox	3.6a1pre	20090720031604	Mac OS X 10.5.7 9J61	x86	EXC_BAD_ACCESS / KERN_PROTECTION_FAILURE	0xffff0269	12985	
2009-07-20 16:06	Firefox	3.6a1pre	20090720031604	Mac OS X 10.5.7 9J61	x86	EXC_BAD_ACCESS / KERN_PROTECTION_FAILURE	0x1bf7db62	349	
2009-07-17 02:47	Firefox	3.6a1pre	20090702031635	Mac OS X 10.4.11 8S2167	x86	EXC_BAD_ACCESS / KERN_INVALID_ADDRESS	0x1bbac8bf	181690	
2009-07-16 19:14	Firefox	3.6a1pre	20090613032901	Mac OS X 10.5.7 9J61	x86	EXC_BAD_ACCESS / KERN_INVALID_ADDRESS	0x1d32f2ad	2573	
2009-07-15 12:41	Firefox	3.6a1pre	20090715031744	Mac OS X 10.5.7 9J61	x86	EXC_BAD_ACCESS / KERN_PROTECTION_FAILURE	0xffff0269	8286	
2009-07-15 12:15	Firefox	3.6a1pre	20090712031423	Mac OS X 10.5.7 9J61	x86	EXC_BAD_ACCESS / KERN_INVALID_ADDRESS	0x1533a817	238430	



[@ Flash_EnforceLocalSecurity] - Firefox 3.6a1pre Crash Report - Report ID: ddb8ab07-3bdb-4c92-8228-c0f7d2090721
http://crash-stats.mozilla.com/report/index/ddb8ab07-3bdb-4c92-8228-c0f7d209
Creative Commons
mozilla
crash reports
Crash ID or Signature
Firefox
Thunderbird
More
Trend Reports
Advanced Search
Get Help
Details
Modules
Raw Dump
Done

Firefox 3.6a1pre Crash Report [@Flash_EnforceLocalSecurity]

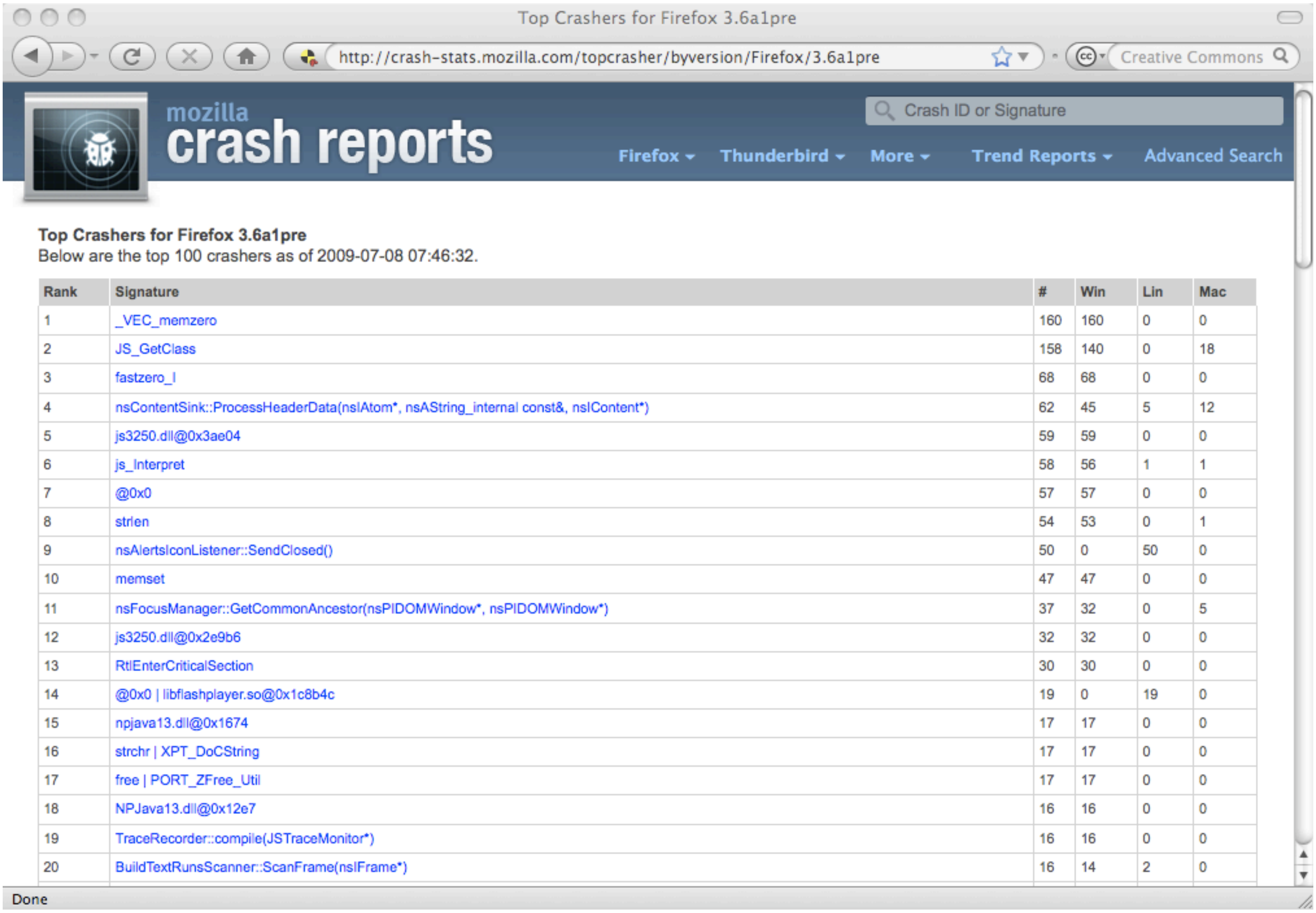
ID: ddb8ab07-3bdb-4c92-8228-c0f7d2090721
Signature: Flash_EnforceLocalSecurity

Signature	Flash_EnforceLocalSecurity
UUID	ddb8ab07-3bdb-4c92-8228-c0f7d2090721
Time	2009-07-21 00:12:17.284743
Uptime	12985
Last Crash	12992 seconds before submission
Product	Firefox
Version	3.6a1pre
Build ID	20090720031604
Branch	1.9.2
OS	Mac OS X
OS Version	10.5.7 9J61
CPU	x86
CPU Info	GenuineIntel family 6 model 15 stepping 10
Crash Reason	EXC_BAD_ACCESS / KERN_PROTECTION_FAILURE
Crash Address	0xffff0269
User Comments	
Processor Notes	

Related Bugs

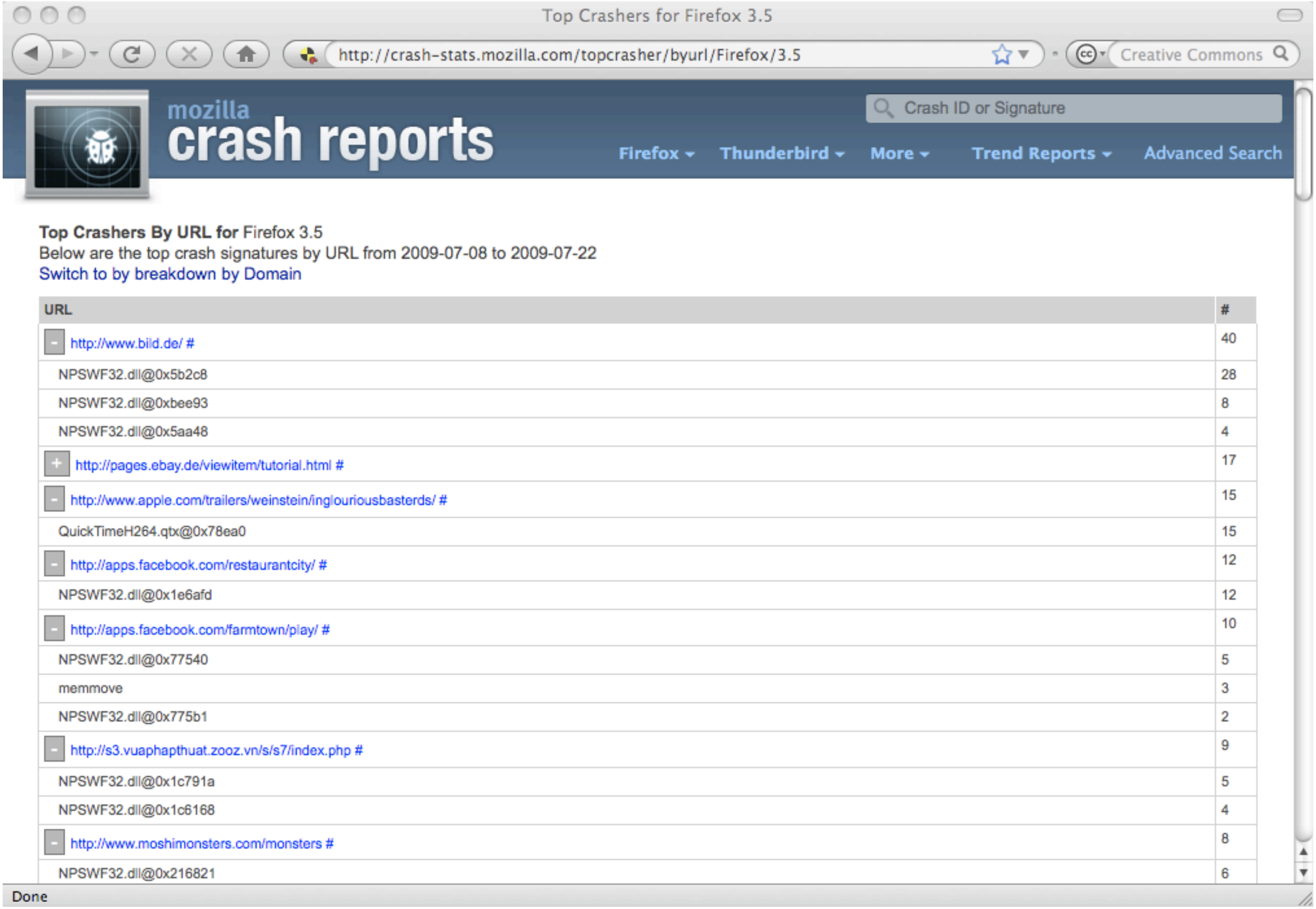
Friday, July 24, 2009

Trend Reports



Top Crashers for Firefox 3.6a1pre
Below are the top 100 crashers as of 2009-07-08 07:46:32.

Rank	Signature	#	Win	Lin	Mac
1	_VEC_memzero	160	160	0	0
2	JS_GetClass	158	140	0	18
3	fastzero_I	68	68	0	0
4	nsContentSink::ProcessHeaderData(nsIAtom*, nsAString_internal const&, nsIContent*)	62	45	5	12
5	js3250.dll@0x3ae04	59	59	0	0
6	js_Interpret	58	56	1	1
7	@0x0	57	57	0	0
8	strlen	54	53	0	1
9	nsAlertsIconListener::SendClosed()	50	0	50	0
10	memset	47	47	0	0
11	nsFocusManager::GetCommonAncestor(nsPIDOMWindow*, nsPIDOMWindow*)	37	32	0	5
12	js3250.dll@0x2e9b6	32	32	0	0
13	RtlEnterCriticalSection	30	30	0	0
14	@0x0 | libflashplayer.so@0x1c8b4c	19	0	19	0
15	npjava13.dll@0x1674	17	17	0	0
16	strchr | XPT_DoCString	17	17	0	0
17	free | PORT_ZFree_Util	17	17	0	0
18	NPJava13.dll@0x12e7	16	16	0	0
19	TraceRecorder::compile(JSTraceMonitor*)	16	16	0	0
20	BuildTextRunsScanner::ScanFrame(nsIFrame*)	16	14	2	0

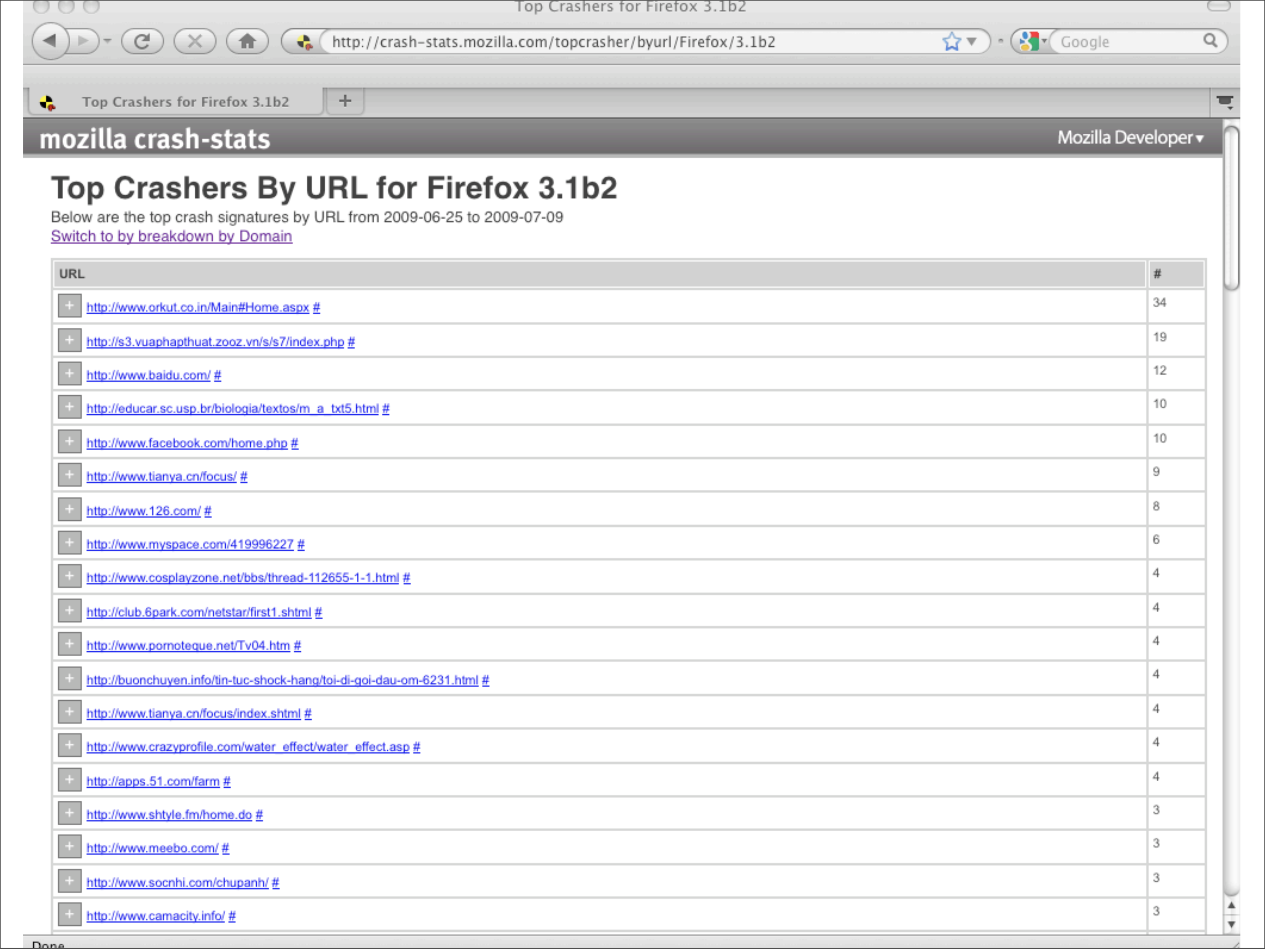


Top Crashers By URL for Firefox 3.5

Below are the top crash signatures by URL from 2009-07-08 to 2009-07-22

Switch to by breakdown by Domain

URL	#
- http://www.bild.de/ #	40
NPSWF32.dll@0x5b2c8	28
NPSWF32.dll@0xbee93	8
NPSWF32.dll@0x5aa48	4
+ http://pages.ebay.de/viewitem/tutorial.html #	17
- http://www.apple.com/trailers/weinstein/inglouriousbasterds/ #	15
QuickTimeH264.qtx@0x78ea0	15
- http://apps.facebook.com/restaurantcity/ #	12
NPSWF32.dll@0x1e6afd	12
- http://apps.facebook.com/farmtown/play/ #	10
NPSWF32.dll@0x77540	5
memmove	3
NPSWF32.dll@0x775b1	2
- http://s3.vuaphapthuat.zooz.vn/s/s7/index.php #	9
NPSWF32.dll@0x1c791a	5
NPSWF32.dll@0x1c6168	4
- http://www.moshimonsters.com/monsters #	8
NPSWF32.dll@0x216821	6

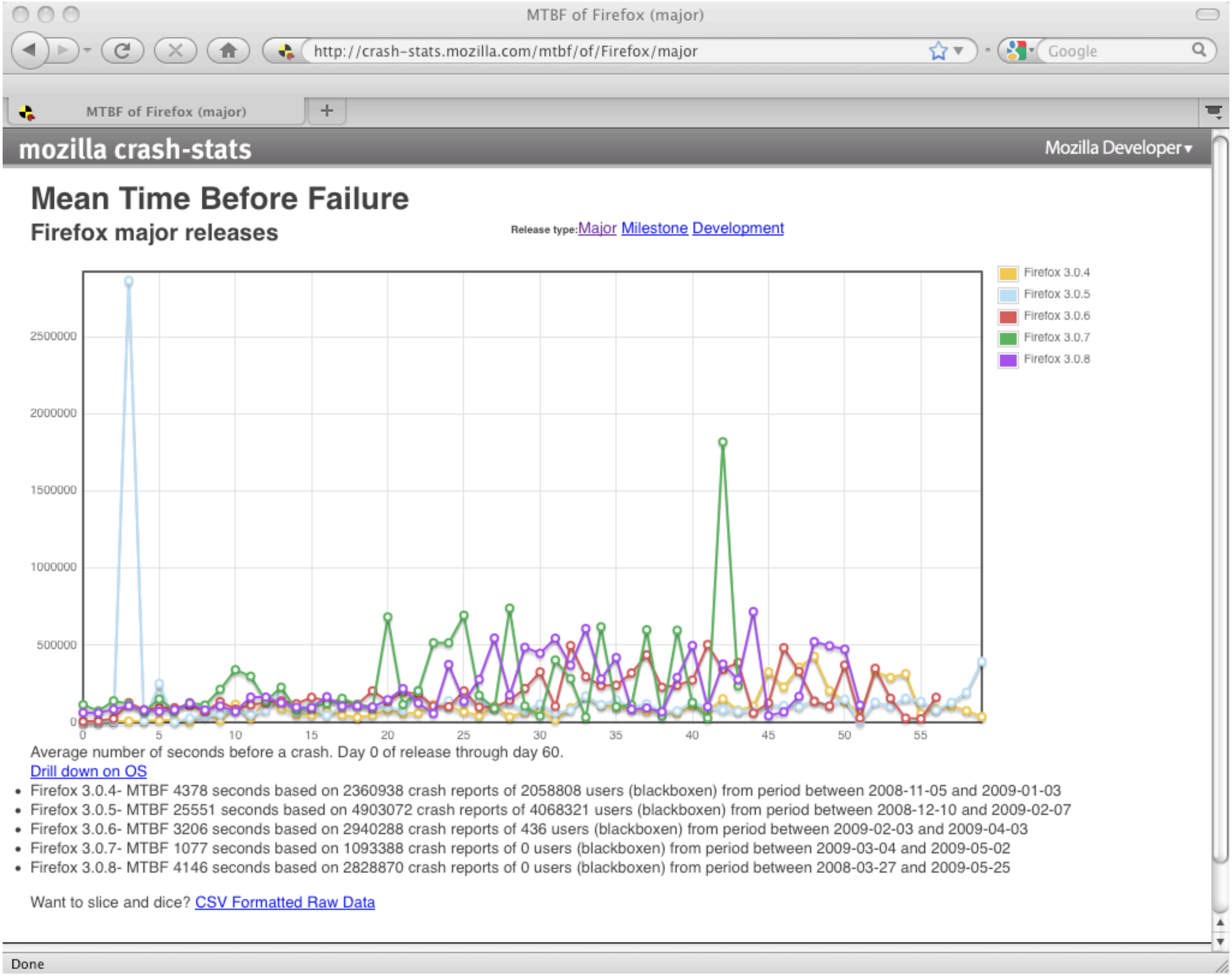

Top Crashers for Firefox 3.1b2
http://crash-stats.mozilla.com/topcrasher/byurl/Firefox/3.1b2
Google
Top Crashers for Firefox 3.1b2
+
mozilla crash-stats
Mozilla Developer ▾
Done

Top Crashers By URL for Firefox 3.1b2

Below are the top crash signatures by URL from 2009-06-25 to 2009-07-09

Switch to by breakdown by Domain

URL	#
+ http://www.orkut.co.in/Main#Home.aspx #	34
+ http://s3.vuaphapthuat.zooz.vn/s/s7/index.php #	19
+ http://www.baidu.com/ #	12
+ http://educar.sc.usp.br/biologia/textos/m_a_txt5.html #	10
+ http://www.facebook.com/home.php #	10
+ http://www.tianya.cn/focus/ #	9
+ http://www.126.com/ #	8
+ http://www.myspace.com/419996227 #	6
+ http://www.cosplayzone.net/bbs/thread-112655-1-1.html #	4
+ http://club.6park.com/netstar/first1.shtml #	4
+ http://www.pornoteque.net/Tv04.htm #	4
+ http://buonchuyen.info/tin-tuc-shock-hang/toi-di-goi-dau-om-6231.html #	4
+ http://www.tianya.cn/focus/index.shtml #	4
+ http://www.crazyprofile.com/water_effect/water_effect.asp #	4
+ http://apps.51.com/farm #	4
+ http://www.shtyle.fm/home.do #	3
+ http://www.meebo.com/ #	3
+ http://www.socnhi.com/chupanh/ #	3
+ http://www.camacity.info/ #	3



Mean Time Before Failure

Firefox major releases

Release type: Major Milestone Development

Average number of seconds before a crash. Day 0 of release through day 60.

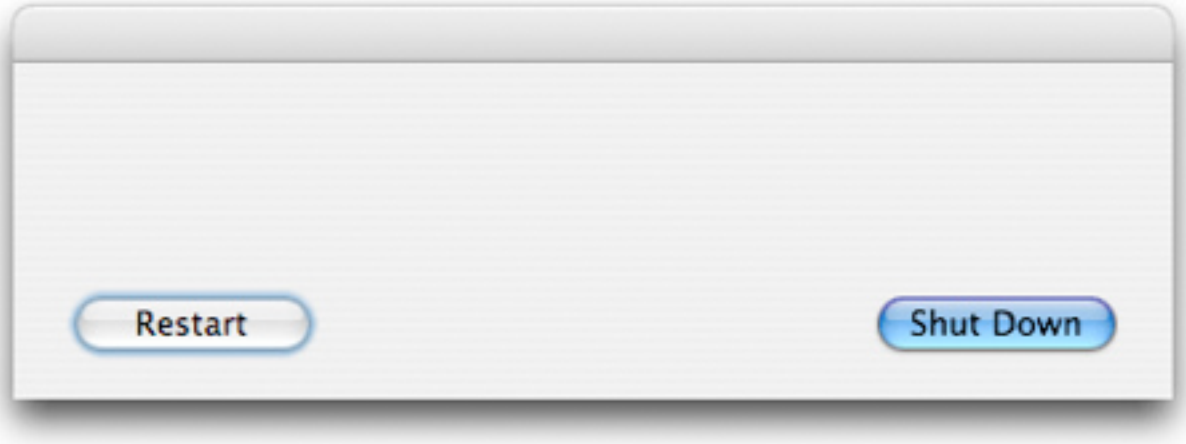
Drill down on OS

- Firefox 3.0.4- MTBF 4378 seconds based on 2360938 crash reports of 2058808 users (blackboxen) from period between 2008-11-05 and 2009-01-03
- Firefox 3.0.5- MTBF 25551 seconds based on 4903072 crash reports of 4068321 users (blackboxen) from period between 2008-12-10 and 2009-02-07
- Firefox 3.0.6- MTBF 3206 seconds based on 2940288 crash reports of 436 users (blackboxen) from period between 2009-02-03 and 2009-04-03
- Firefox 3.0.7- MTBF 1077 seconds based on 1093388 crash reports of 0 users (blackboxen) from period between 2009-03-04 and 2009-05-02
- Firefox 3.0.8- MTBF 4146 seconds based on 2828870 crash reports of 0 users (blackboxen) from period between 2008-03-27 and 2009-05-25

Want to slice and dice? CSV Formatted Raw Data

The Why and the How

Open Wins



Screenshot: Adriano Castro on Flickr http://www.flickr.com/photos/acastro

Open Has Limits

- Mozilla values privacy
- Project has vacillated over fields like email
- Urls and other data are aggregated and truncated
- QA and Devs would love more types of crash data, but privacy concerns trump these enhancements

Development

- Evolutionary - Incremental
- Community Driven - Bugzilla
- No “benevolent dictator”
- Driven by the quest for Quality Software

Technology

- PHP / Kohana
- jQuery
- flot
- ezComponents
- Postgresql
- Memcached

The Internet

Crash Reporting:
Mozilla’s Open Source Solution

IRC: #breakpad (irc.mozilla.com)

http://code.google.com/p/google-breakpad/

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