All your cores are belong to us

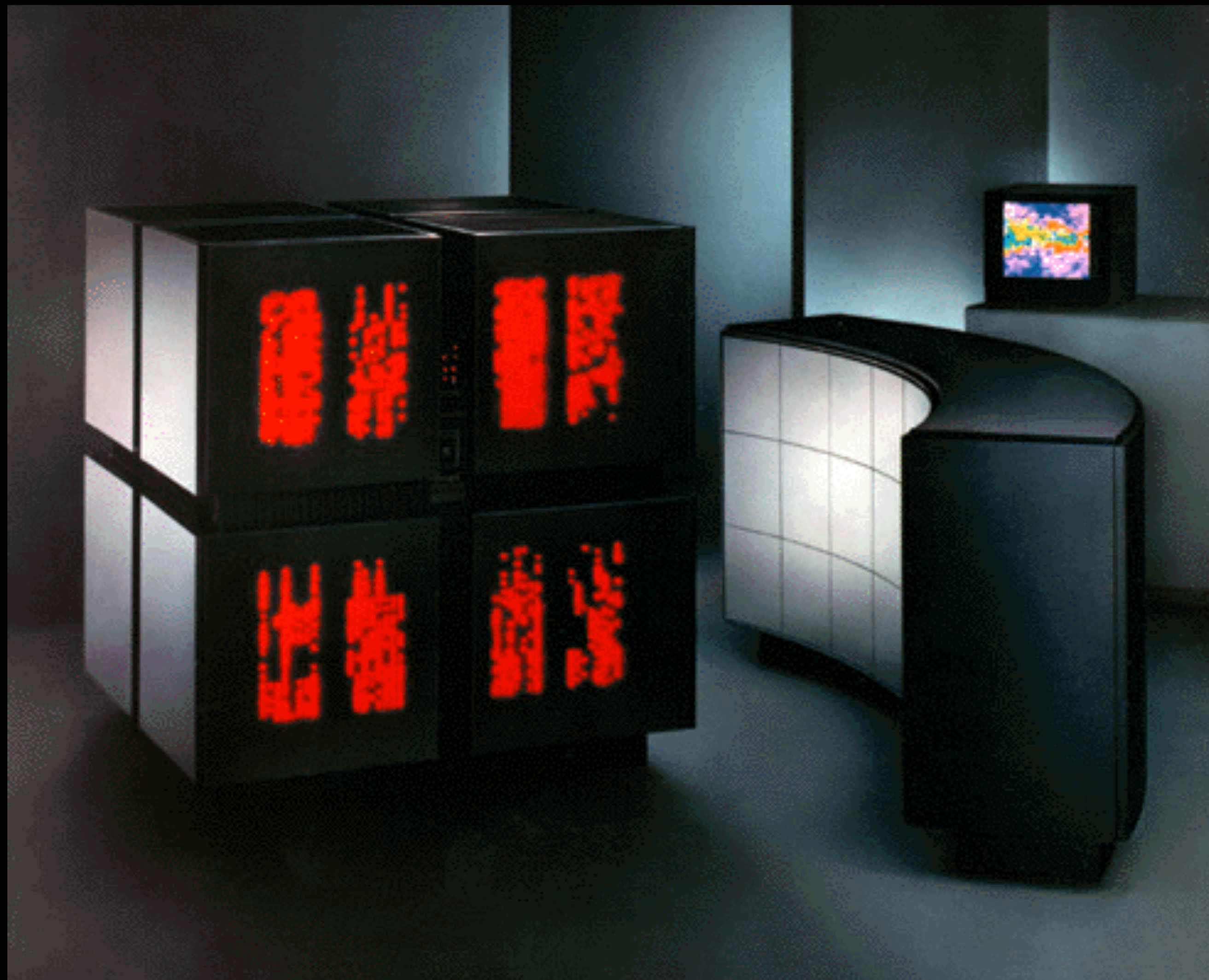
Achieving Manycore Scale

Alexander Gounares
Concurix Corporation
April 23rd, 2013

The manycore era is here.

“Zero Wing”, 1991

The manycore era is here.



Connection Machine CM-2

Big Data: Facebook stats (August 2012)

2.5 Billion content items shared per day

2.7 Billion “Likes” per day

300 Million photos per day

500+ Terabytes new data ingested per day

105 Terabytes data processed every 30 minutes

100+ Petabytes in no-sql Hadoop storage

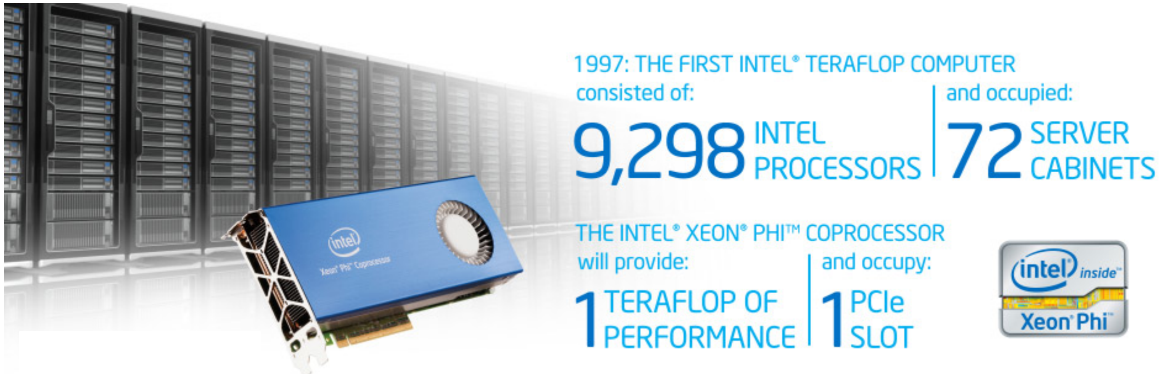
Disk is the new Tape

Memory is the new Disk

Modern Cloud Service Model

Incoming Data Streams → In-memory Real Time Data processing → Huge noSQL Storage and Analytics

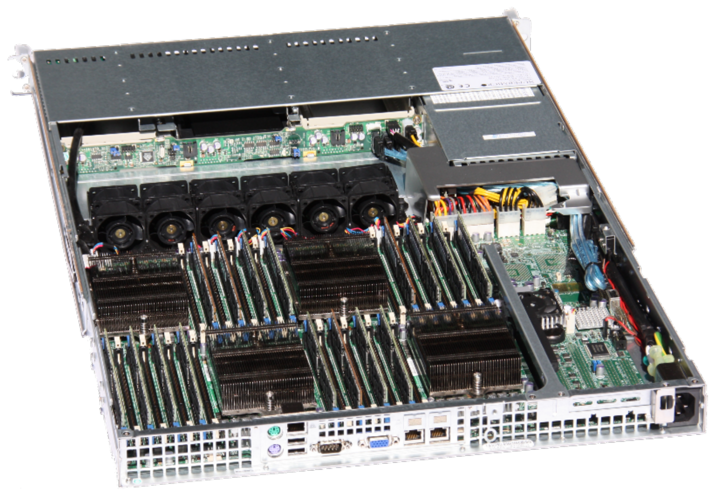
The Manycore era is here now



~$4.2K

AMD Opteron family 15h
64 cores: 16 per chip x 4 sockets
Streaming SIMD extensions (SSE4)
128 GB RAM—512GB max
8 NUMA Domains with Hypertransport

~$4.7K



The manycore era is here.

The manycore era is here.

The manycore era is here.

Traditional software does not scale on manycore

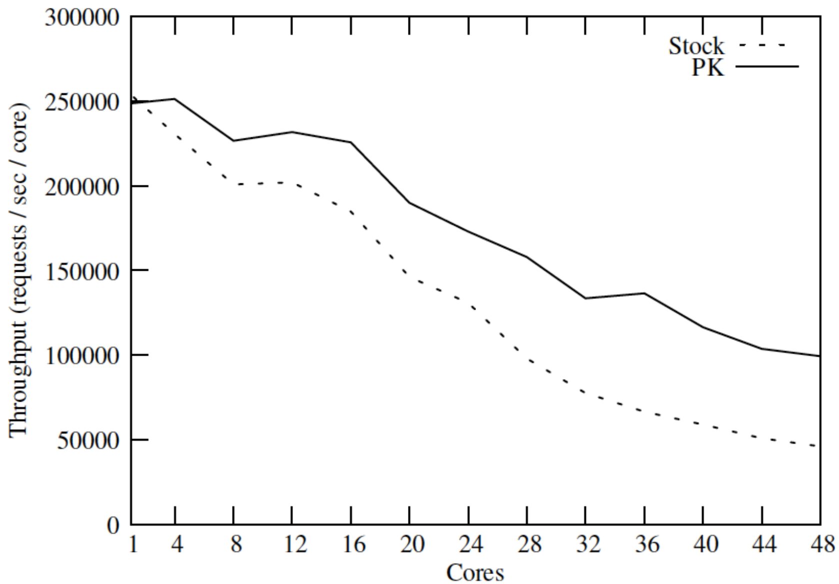
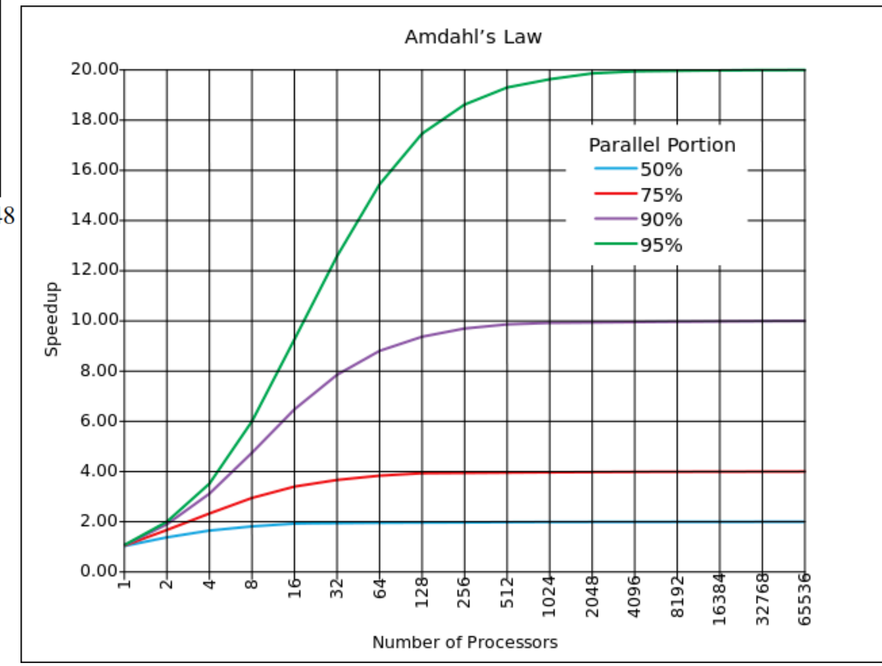


Figure 5: memcached throughput.

http://pdos.csail.mit.edu/papers/linux:osdi10.pdf

Locks in traditional O-O languages limit scalability per Amdahl's Law



Erlang and NodeJS to the rescue!

Erlang	Node.JS
Single assignment	Server side Javascript with Google V8 engine
No shared state	Single threaded *asynchronous* callback model
Lightweight processes	Multi-core via OS processes (1 or more per core)
Message passing	Existing libraries wrapped in closures—functional programming for the uninitiated
Recursion and pattern matching	Message passing via [domain] sockets
Concurrency built in	
Supervisor based reliability model	

The manycore era is here.

```
-module(tut15).

-export([start/0, ping/2, pong/0]).

ping(0, Pong_PID) ->
    Pong_PID ! finished,
    io:format("ping finished~n", []);

ping(N, Pong_PID) ->
    Pong_PID ! {ping, self()},
    receive
        pong ->
            io:format("Ping received pong~n", [])
    end,
    ping(N - 1, Pong_PID).

pong() ->
    receive
        finished ->
            io:format("Pong finished~n", []);
        {ping, Ping_PID} ->
            io:format("Pong received ping~n", []),
            Ping_PID ! pong,
            pong()
    end.

start() ->
    Pong_PID = spawn(tut15, pong, []),
    spawn(tut15, ping, [3, Pong_PID]).
```

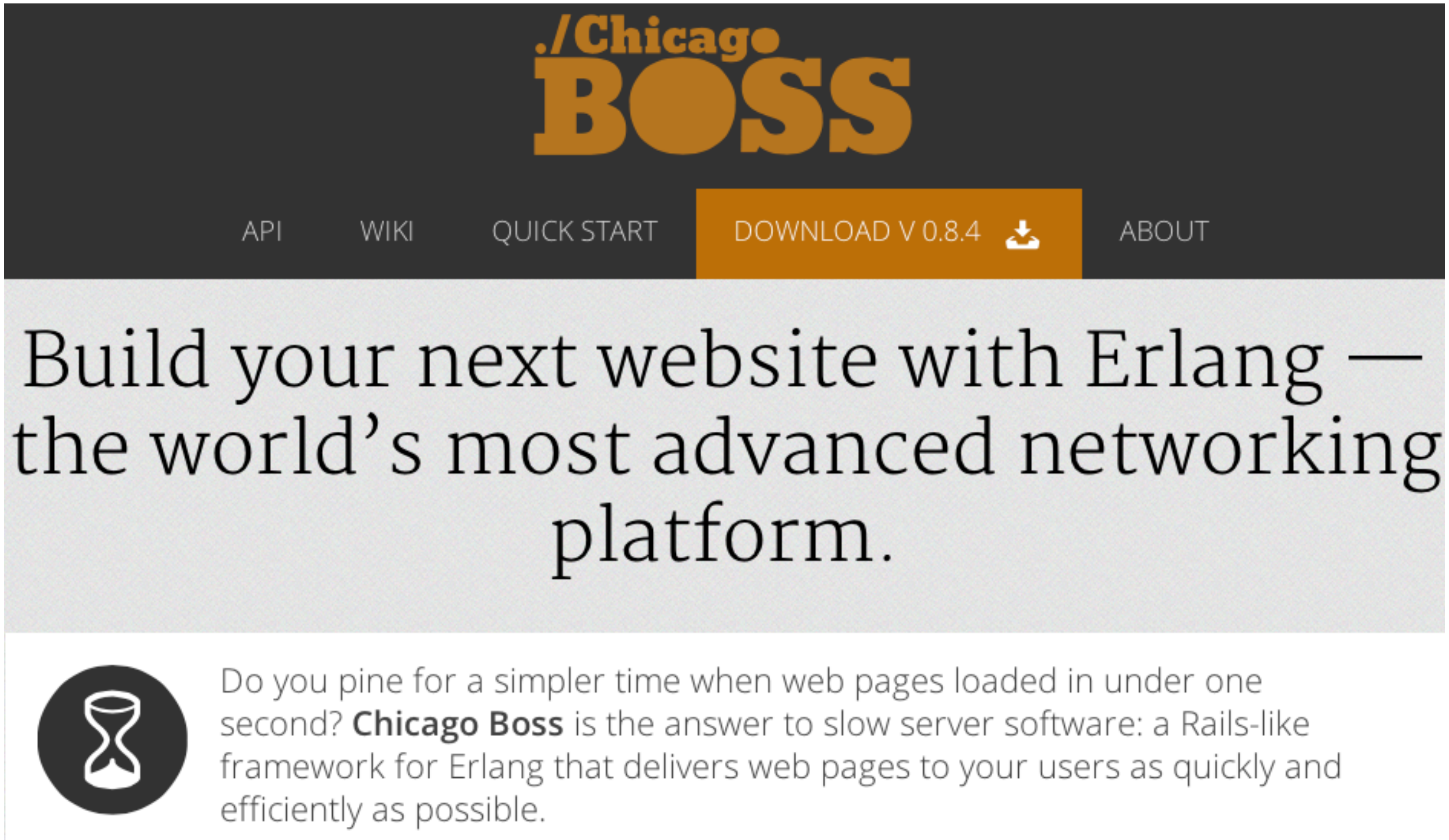
NodeJS example

```
// Simplest HTTP server
var http = require('http'),
    port = 8000;

var server = http.createServer(function (request, response) {
  response.writeHead(200, {"Content-Type": "text/plain"});
  response.end("Hello World\n");
});

server.listen(port);
console.log("Listening on <insert your favorite ip>:" + port);
```

Now something more complex….

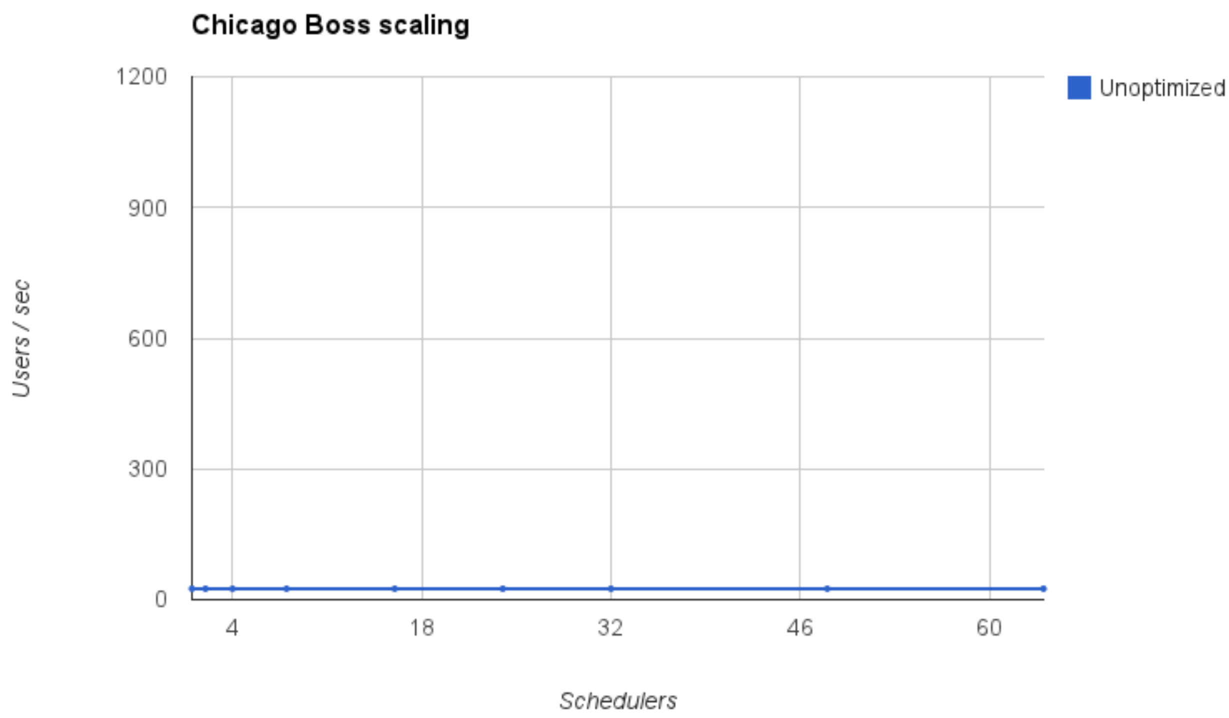
The manycore era is here.

Ruby Rails Erlang Style!



The manycore era is here.

Real world Chicago Boss application (concurix.com) December 2012

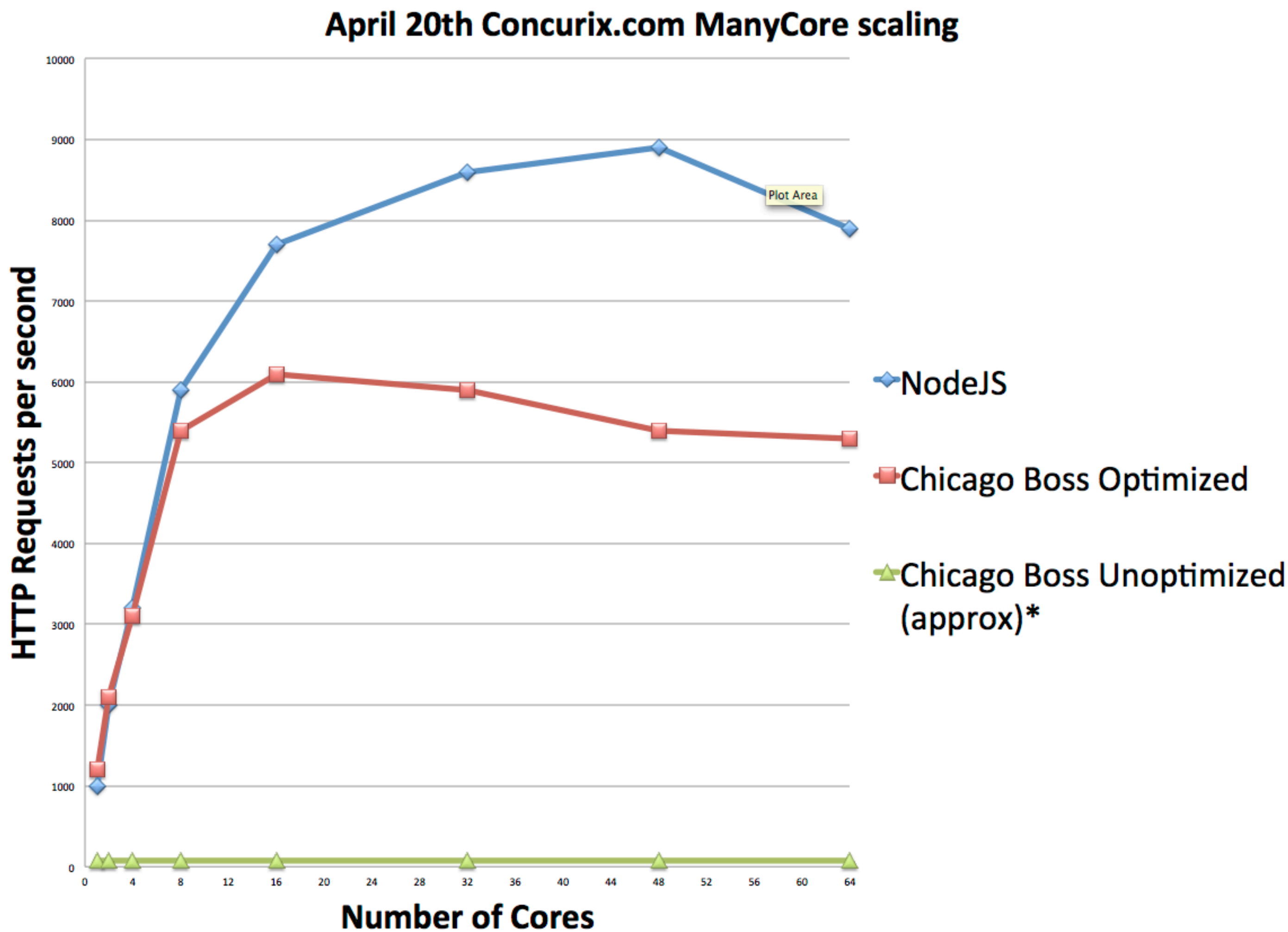


The manycore era is here.

Back to Amdahl's Law—it's all about the locks

The manycore era is here.

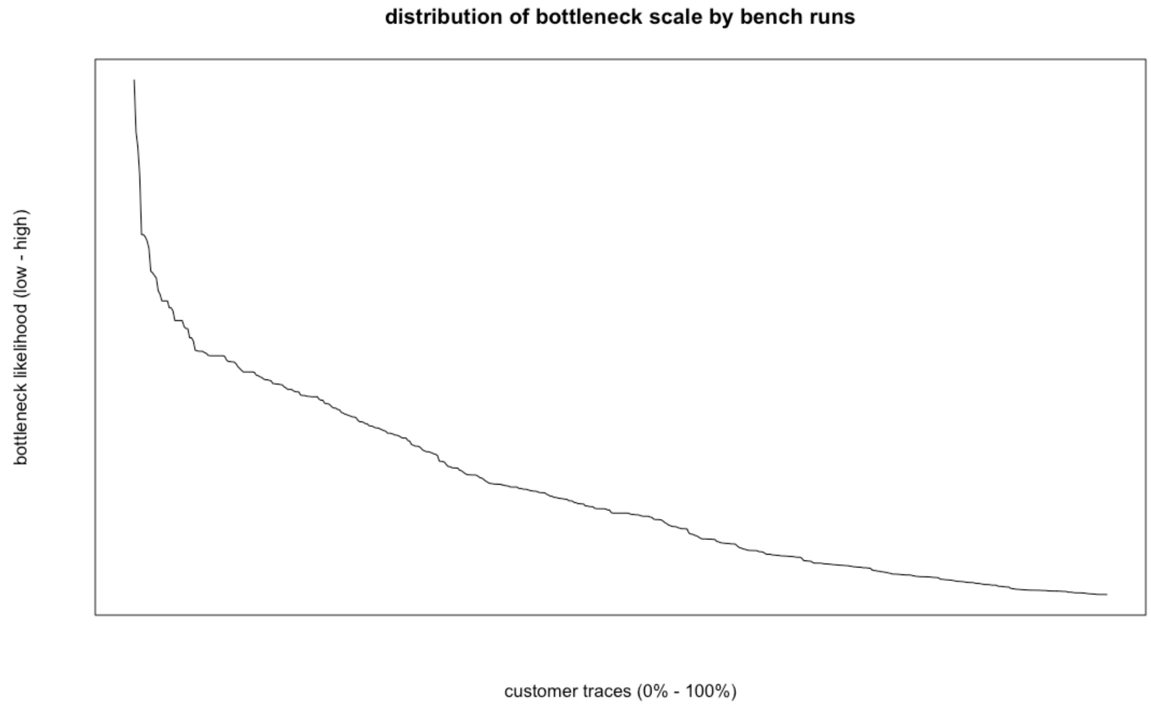
Where we are now… 45x+!!!



*: A different load tester was used in the December 2012 numbers, this is rough approximation of the December numbers in the new tool.

The manycore era is here.

We are not alone…



Concurix has over 1.2 million (and growing!) profile data sets.

100% of these profiles show at least 1 bottleneck candidate

How we did it...

The manycore era is here.

Profiling with Real Time Visualizations and Big Data

Gprof!

```
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  ms/call  ms/call  name
 33.34      0.02     0.02     7208     0.00     0.00  open
 16.67      0.03     0.01      244     0.04     0.12  offtime
 16.67      0.04     0.01        8     1.25     1.25  memccpy
 16.67      0.05     0.01        7     1.43     1.43  write
 16.67      0.06     0.01                             mcount
  0.00      0.06     0.00      236     0.00     0.00  tzset
  0.00      0.06     0.00      192     0.00     0.00  tolower
  0.00      0.06     0.00       47     0.00     0.00  strlen
  0.00      0.06     0.00       45     0.00     0.00  strchr
  0.00      0.06     0.00        1     0.00    50.00  main
  0.00      0.06     0.00        1     0.00     0.00  memcpy
  0.00      0.06     0.00        1     0.00    10.11  print
  0.00      0.06     0.00        1     0.00     0.00  profil
  0.00      0.06     0.00        1     0.00    50.00  report
```

Source: http://linuxforengineers.blogspot.com/2012/07/its-time-to-speed-up-your-code-with.html

Can we do better…?

The manycore era is here.

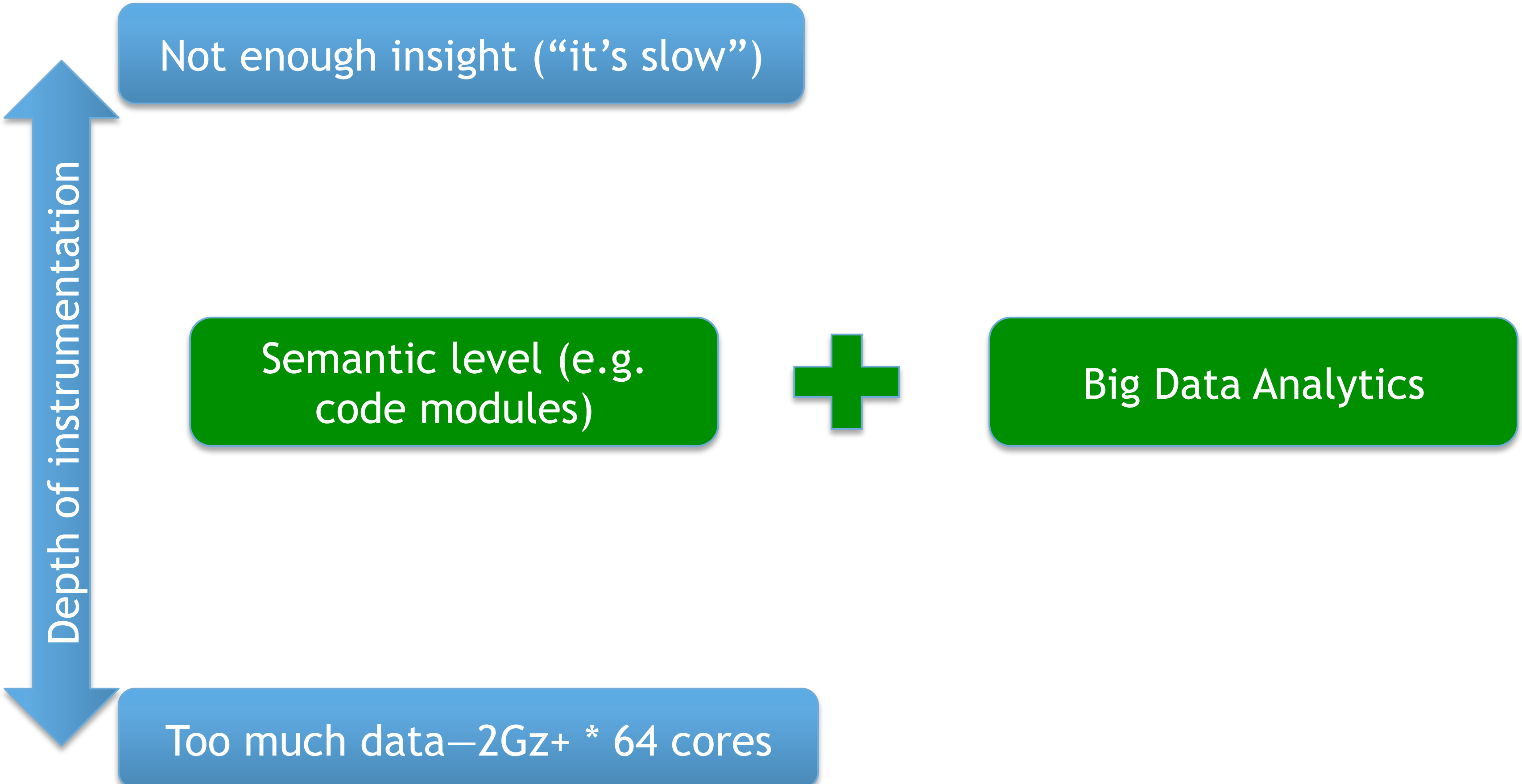
Yes!

Not enough insight ("it's slow")

Depth of instrumentation

Too much data—2Gz+ * 64 cores

Automatically detect and instrument a semantic "middle"



Measuring Similarity

If we line up the data in an ordered vector, treat the vector as a point in N dimensional space, then similarity between data sets can be measured by distance between the points

- One implementation is the cosine similarity $$\theta = \arccos\left(\frac{a \cdot b}{\sqrt{\sum_{k=1}^{n} a^2_k}\sqrt{\sum_{k=1}^{n} b^2_k}} \right)$$

Application example

- Identify transitions
  - Data = the number of messages sent between any pair of sender-receiver, during a time window
  - Vector = line up the data along all possible pairs, , like "mochiweb-to-poolboy", in fixed order
  - Data for each time window is a vector of length of NxN (N = number of processes)
  - Big change in similarity between the data vectors means big shift in message passing activity
- Benchmark repeatability

```
1.000 0.999 0.923 0.999 benchrun-399
----- 1.000 0.917 0.999 benchrun-403
----- ----- 1.000 0.910 benchrun-404
----- ----- ----- 1.000 benchrun-406
```

Clustering

Grouping data points by similarity on some metric

Algorithm: repeat until converge

Assignment to clusters:

$$Cluster^{(t)}_i = \{ p : \|p - center^{(t)}_i\| \le \|p - center^{(t)}_j\| \ \forall j \le k \}$$

Update cluster cer

$$center^{(t+1)}_i = \frac{1}{|Cluster^{(t)}_i|} \sum_{p \in Cluster^{(t)}_i} p$$

Applications

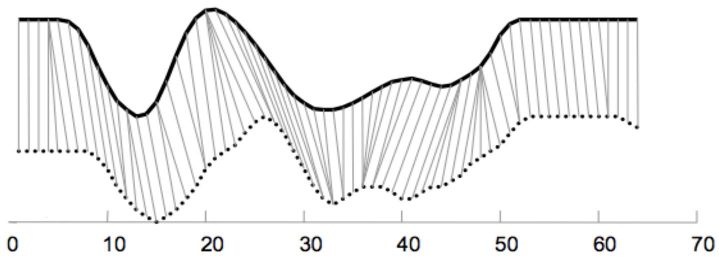
Often used for discovery tasks when there are little prior knowledge about data

Example: Bucket the many processes to a few types according to their activities over time

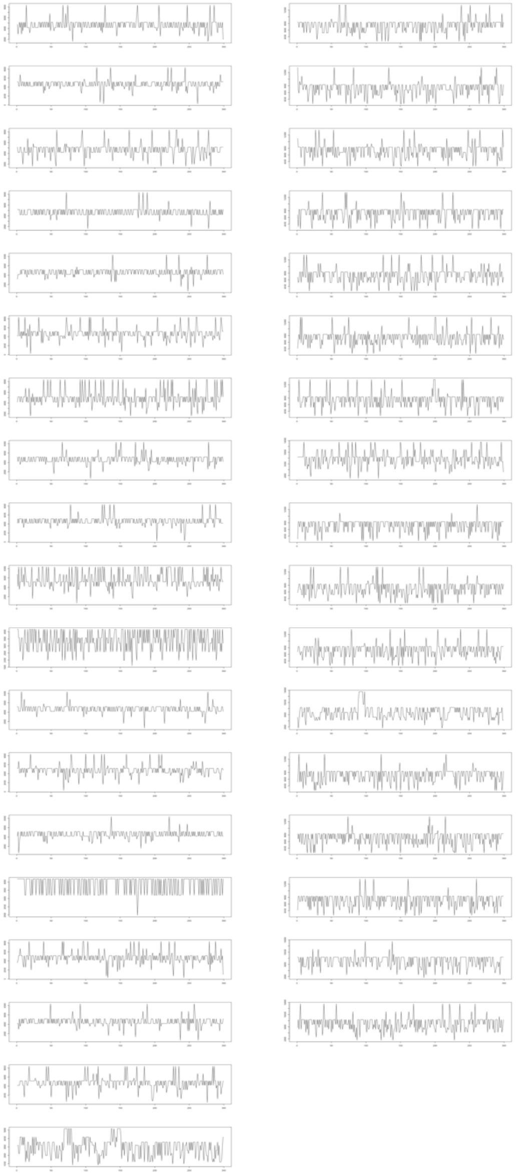
The manycore era is here.

Time series analysis

Use “dynamic time warping” distance to measure how well two time series match



Diff	DTW
$a == b$	$\lvert a - b \rvert$
INSERT	Shift time out
DELETE	Shift time in



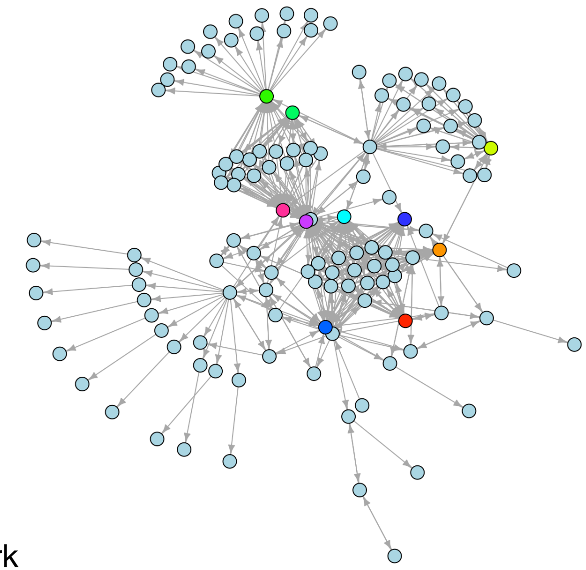
The manycore era is here.

Network analysis: centrality

Centrality: relative importance of a vertex in the network

Computation

- Degree centrality $C_D(v) = degree(v)$
- Closeness centrality $C_C(v) = \frac{n-1}{\sum_{u \in V} shortest\text{-}distance(v,u)}$
- Eigenvector centrality $C_{EC}(v) = \lambda \sum_{\{u,v\} \in E} C_{EC}(u)$
- Betweenness centrality $C_B(v) = \sum_{s \neq v \neq t \in V} \frac{\sigma_{st}(v)}{\sigma_{st}}$



Application

- Detecting most important process in message passing network
- Bottleneck detection

The manycore era is here.

Predictive Bottleneck Detection

Build a statistical model of how an application performs, and try to predict when bottlenecks can occur *before* they occur.

We accomplish this by watching the relationship between events and time over time…or:

fit a line time = a + b * task to points {($task_1$, $time_1$), ($task_2$, $time_2$), …, ($task_N$, $time_N$)}

$$\sigma_{task} = \sqrt{\frac{1}{(N-1)}\sum_{i=1}^{N}\left(task_i - \overline{task}\right)^2} \qquad \sigma_{time} = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}\left(time_i - \overline{time}\right)^2}$$

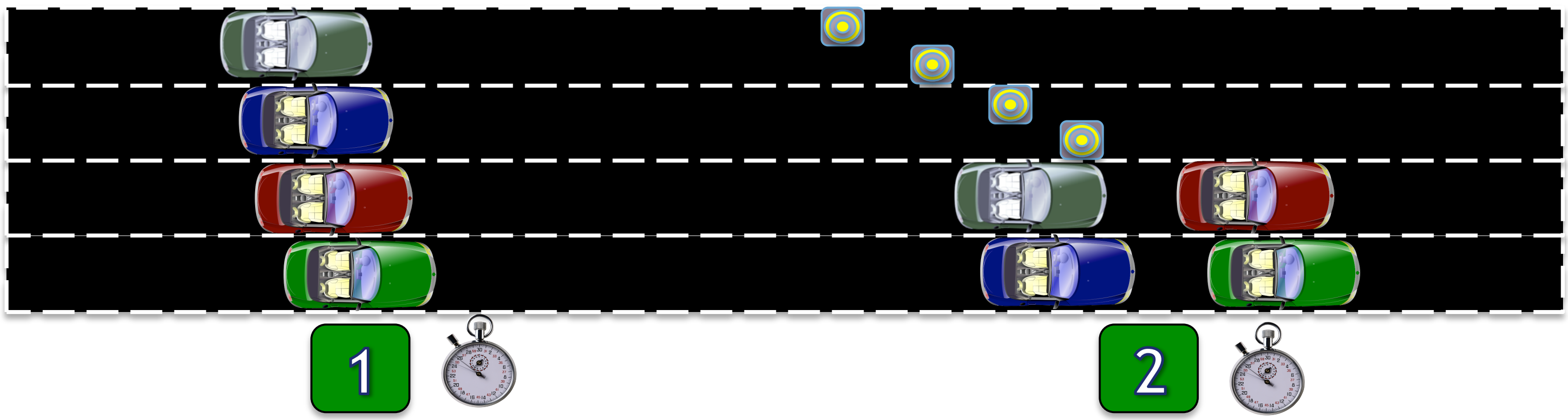
$$r_{task,time} = \frac{\sum_{i=1}^{N}\left(task_i - \overline{task}\right)\left(time_i - \overline{time}\right)}{(N-1)\sigma_{task}\sigma_{time}} \qquad b = r_{task,time}\frac{\sigma_{time}}{\sigma_{task}}$$

$$a = \overline{time} - b * \overline{task}$$

The manycore era is here.

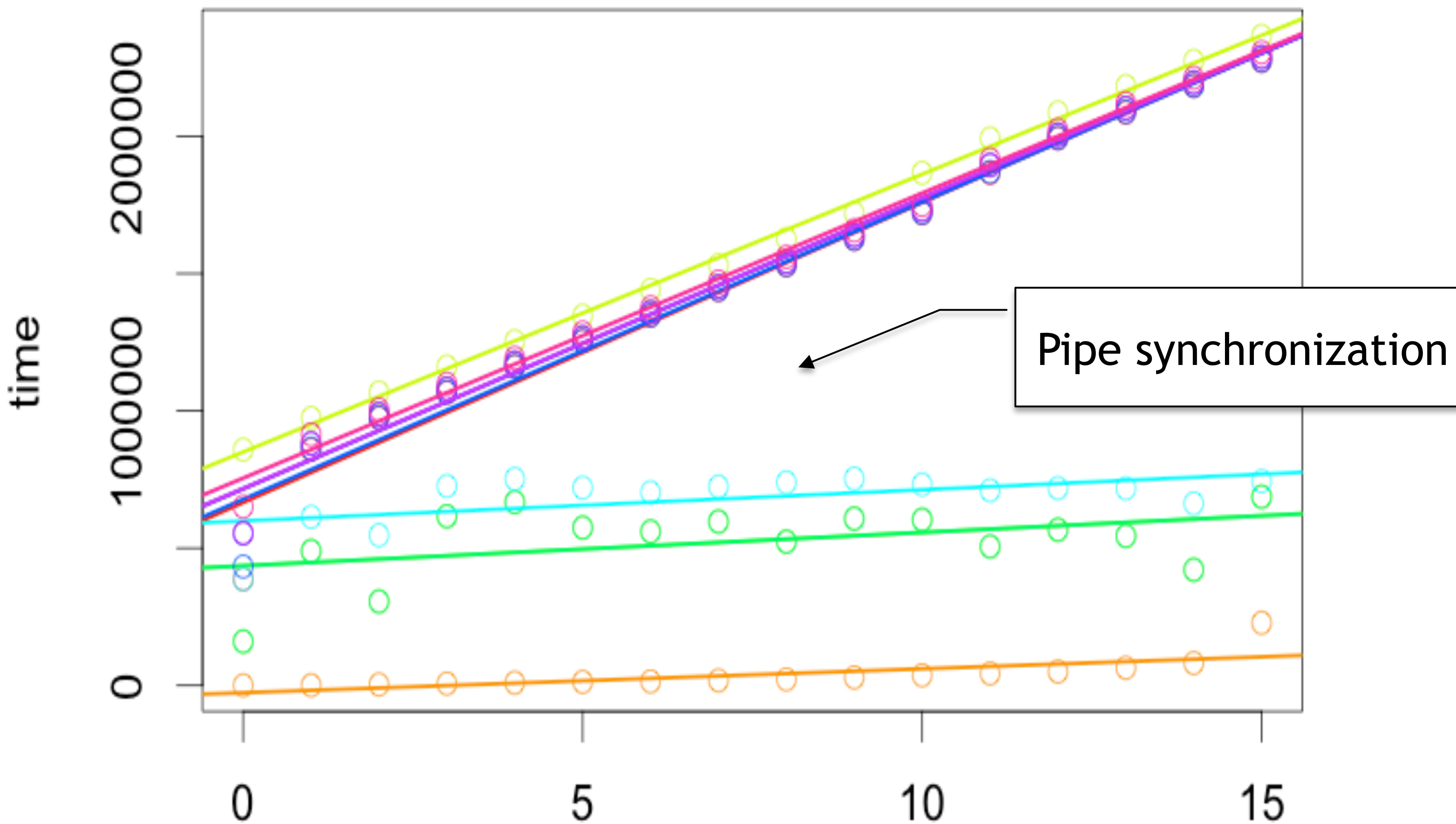
Bottleneck detection (simplified)

Order cars by arrival



Later cars are delayed more

Python MapReduce Bottleneck



Works cross-language: C and Erlang implemented, NodeJS soon

The manycore era is here.

The Bet

The manycore era is here.

Code for the bet…

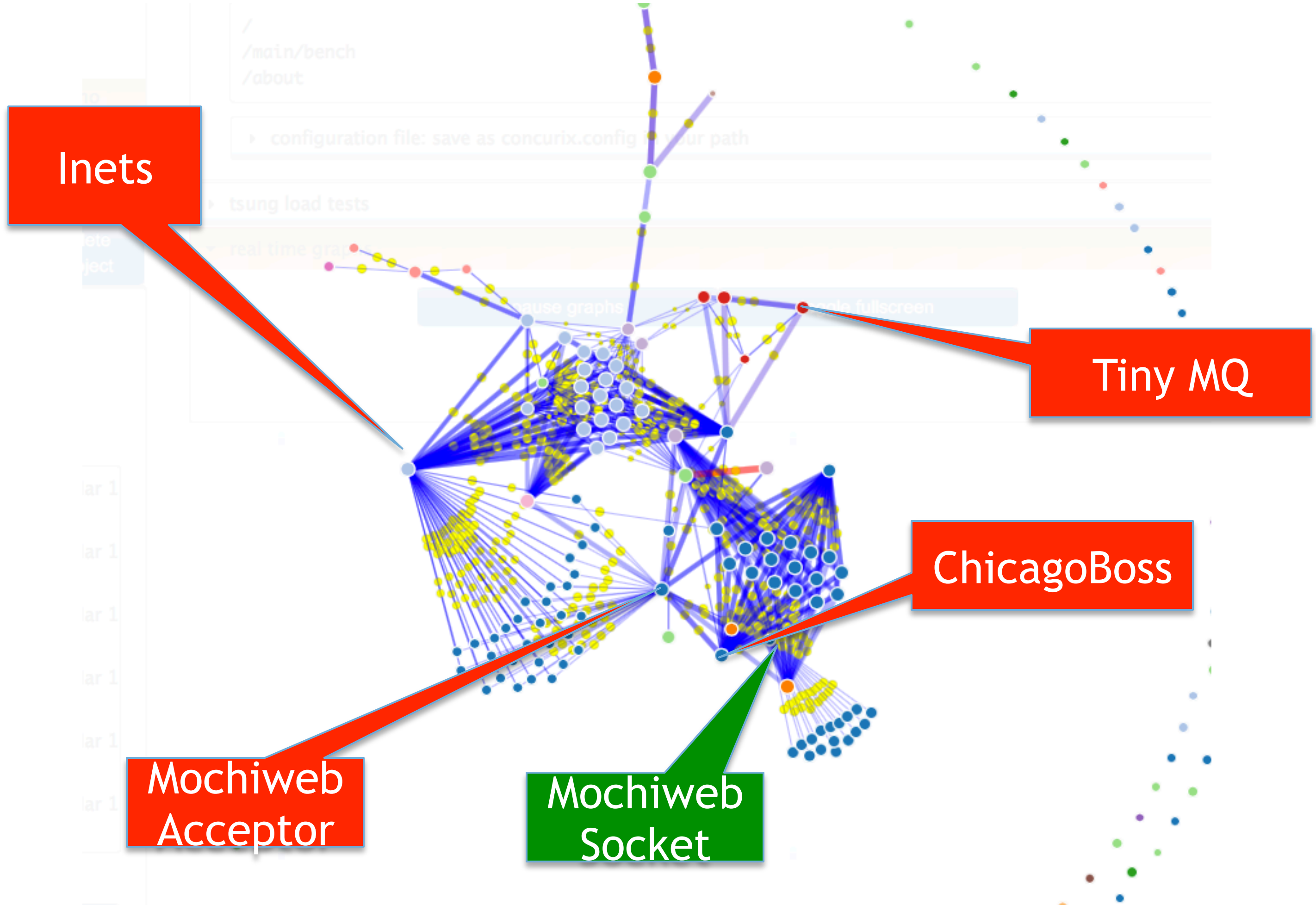
```
reload_routes() ->
    gen_server:call(boss_web, reload_routes).

reload_translation(Locale) ->
    gen_server:call(boss_web, {reload_translation, Locale}).

reload_all_translations() ->
    gen_server:call(boss_web, reload_all_translations).

reload_init_scripts() ->
    gen_server:call(boss_web, reload_init_scripts).

get_all_routes() ->
    gen_server:call(boss_web, get_all_routes).

get_all_models() ->
    gen_server:call(boss_web, get_all_models).

get_all_applications() ->
    gen_server:call(boss_web, get_all_applications).

base_url(App) ->
    gen_server:call(boss_web, {base_url, App}).

domains(App) ->
    gen_server:call(boss_web, {domains, App}).

static_prefix(App) ->
    gen_server:call(boss_web, {static_prefix, App}).

translator_pid(AppName) ->
    gen_server:call(boss_web, {translator_pid, AppName}).

router_pid(AppName) ->
    gen_server:call(boss_web, {router_pid, AppName}).

application_info(App) ->
    gen_server:call(boss_web, {application_info, App}).
```

ON TARGET

DEMO

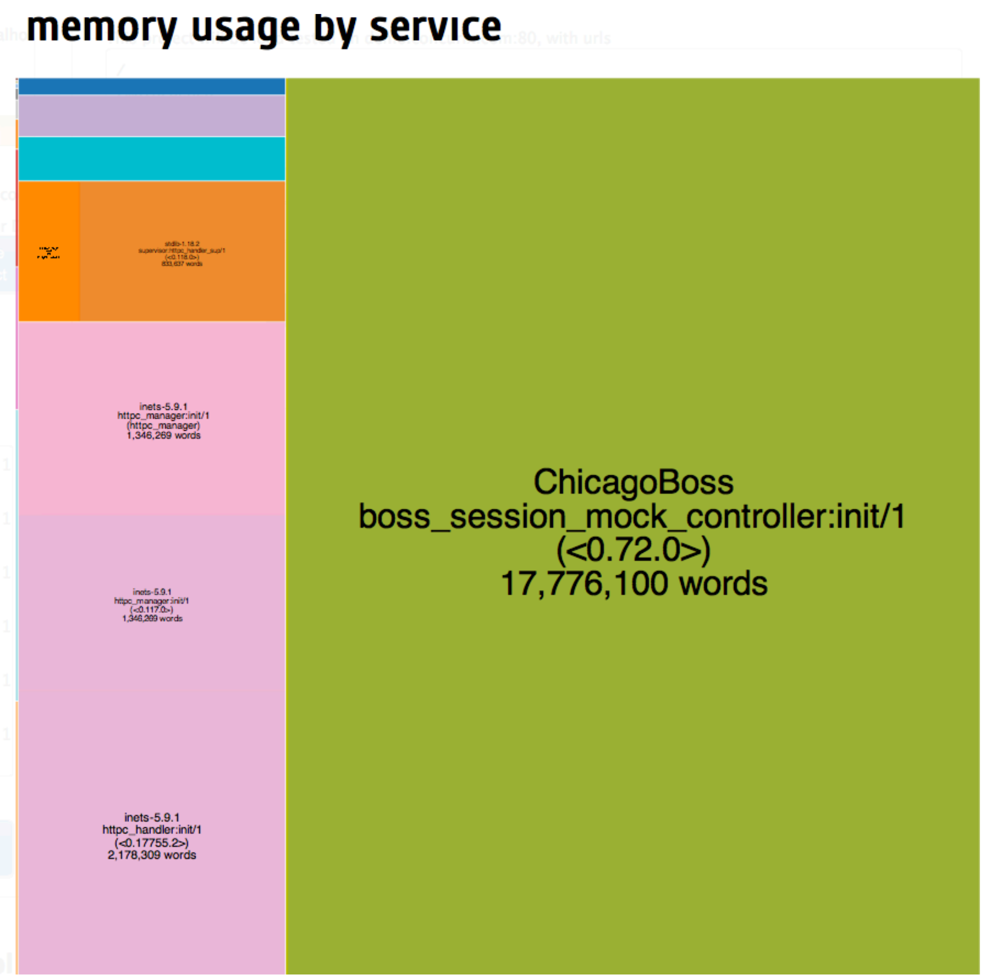
The manycore era is here.

Message passing between processes



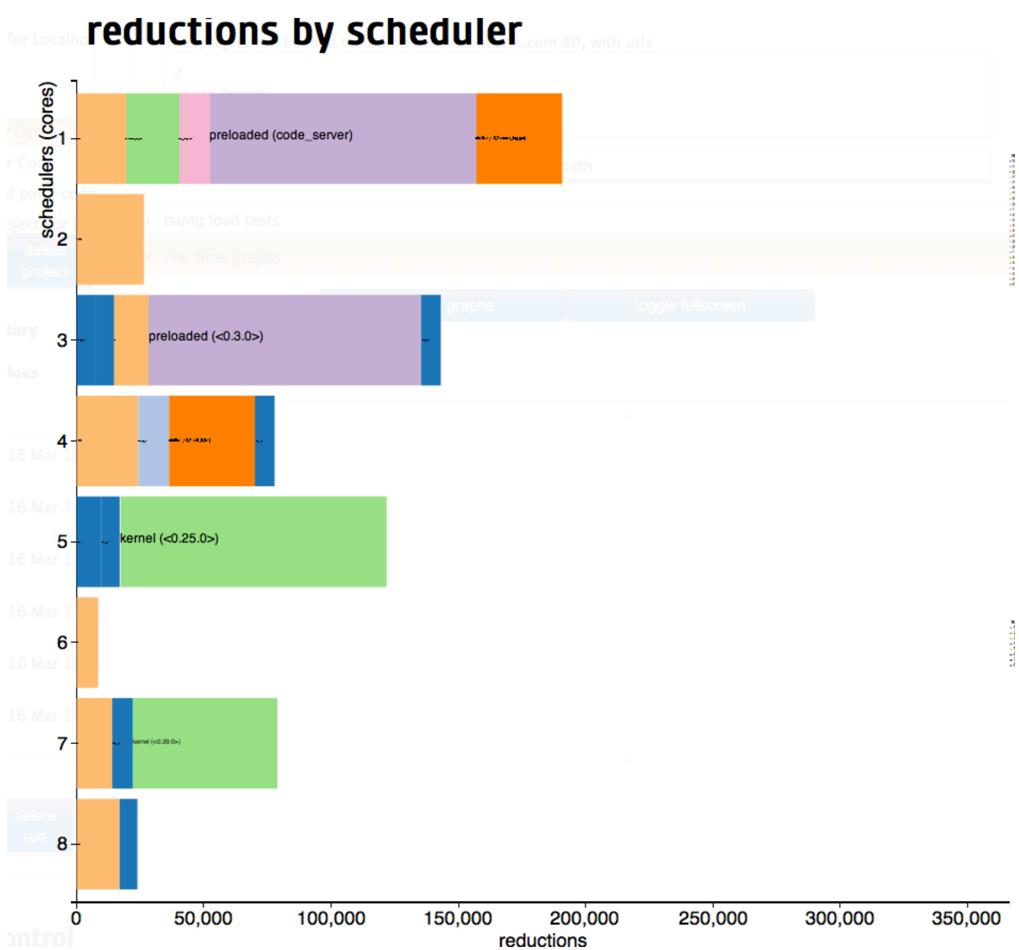
The manycore era is here.

Excessive Memory Usage



The manycore era is here.

Uneven CPU Core utilization



Use the +sbt nnts flag to lock threads to schedulers!

Try it yourself--Erlang!

1. Add concurix_runtime to your rebar.config file:

```
{concurix_runtime, ".*",
     {git, "https://github.com/Concurix/cx_runtime.git"}}
```

2. Start the concurix_runtime system:

```
concurix_runtime:start()
```

3. Navigate to http://concurix.com/main/bench

ERLANG

Try it yourself--NodeJS! (available mid May, 2013)

1. Install the Concurix NodeJS runtime:

```
>npm install concurixjs
```

2. Start the concurix_runtime system:

```
var cx = require('concurixjs')
cx.start();
```

3. Navigate to http://concurix.com/main/bench

Promising future

More languages / frameworks

RAILS python C android

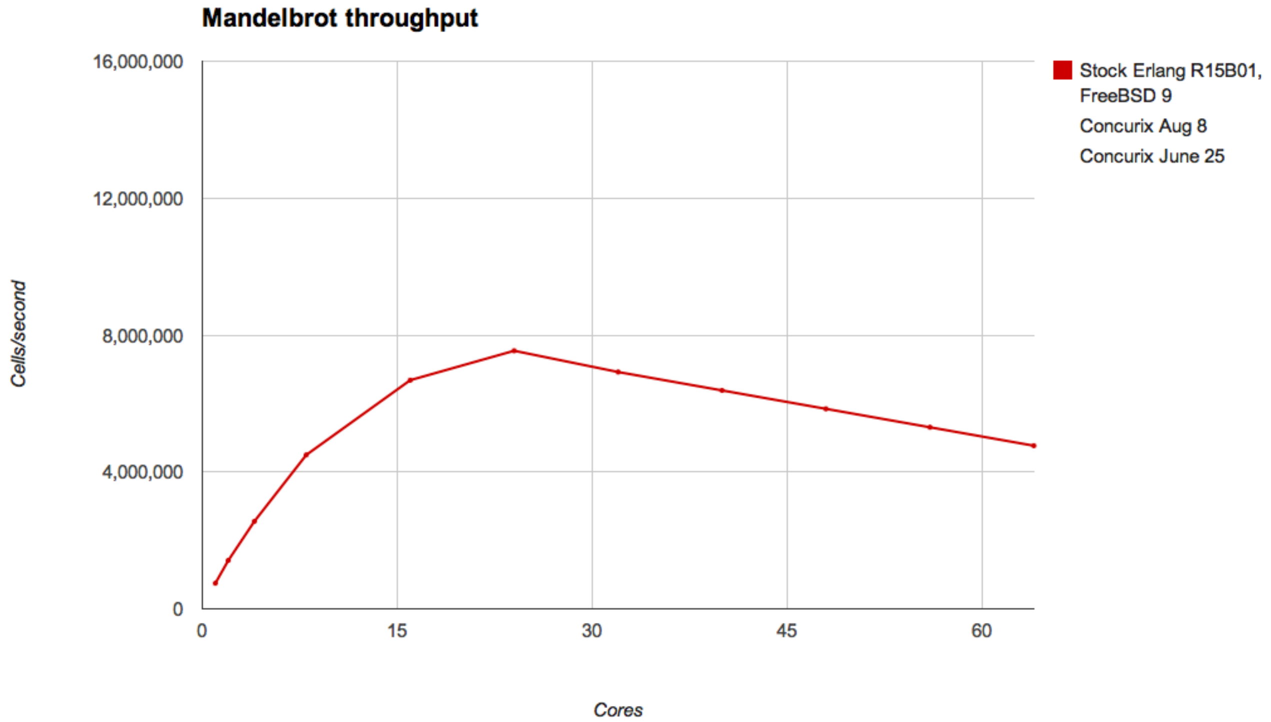
More Scenarios: cyber-security detection?

ANONYMOUS
We are Legion. We do not Forgive.
We do not Forget. Expect us.

Q&A

The manycore era is here.

Almost…even very parallelizable workloads had trouble scaling



The manycore era is here.

After a bit of work….