http://www.erlang-factory.com

Erlang Extreme

Erlang Factory Lite – Erlang Day in Kraków

Piotr Kaleta, Michał Ptaszek, Michał Zajda

25 November 2009

Agenda

- Multicore support
- Distribution
- Fault tolerance
- OTP behaviors

Erlang

Multicore support

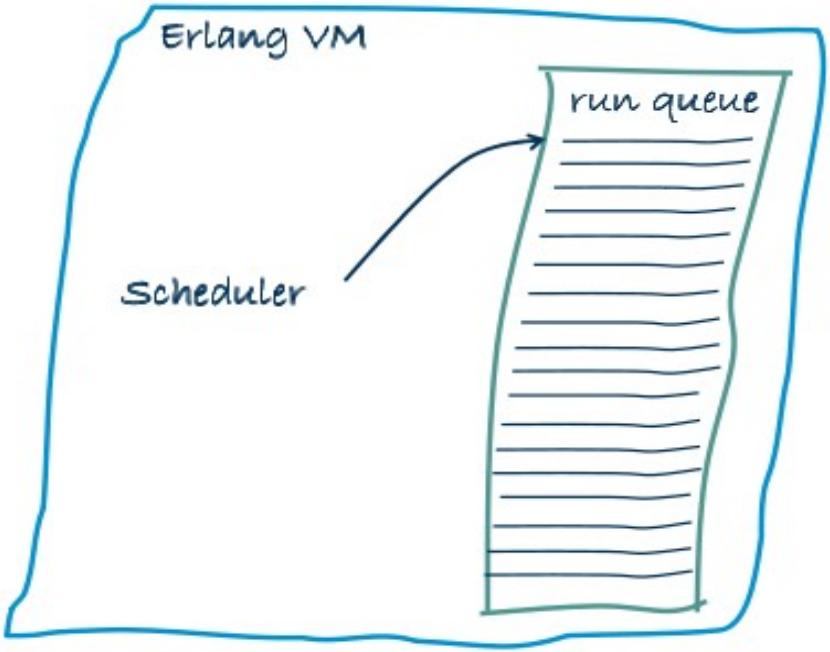
- Erlang designed for:
  - share-nothing architecture
  - asynchronous message passing
  - distribution transparency
- So:
  - no mutexes
  - no transaction memory
  - race conditions only on the architecture level (no programming traps)
  - apart from the Erlang VM itself

Erlang

Multicore support

- Evolving from...

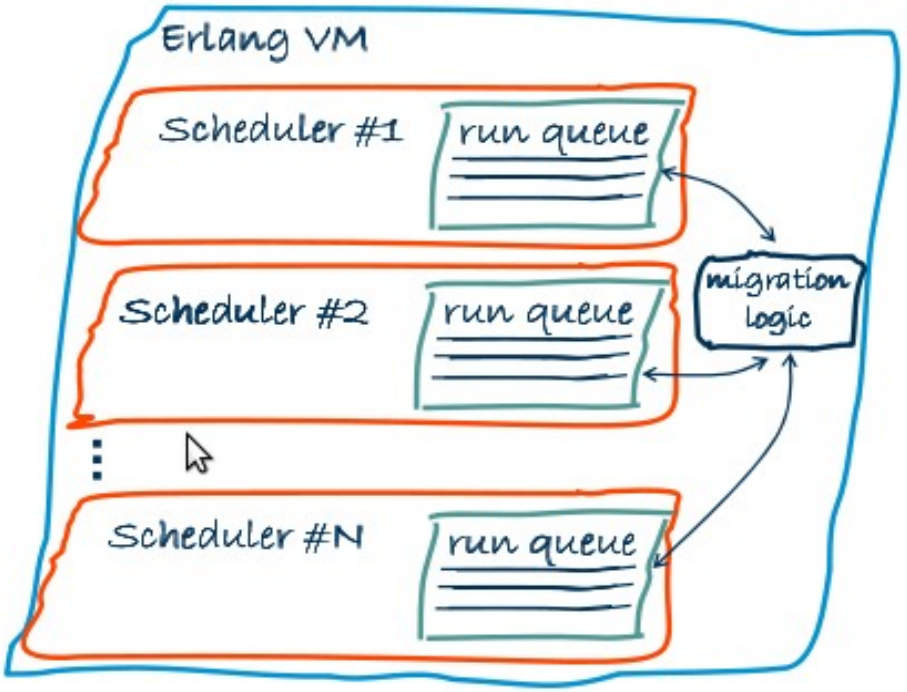
non-SMP VM



Erlang

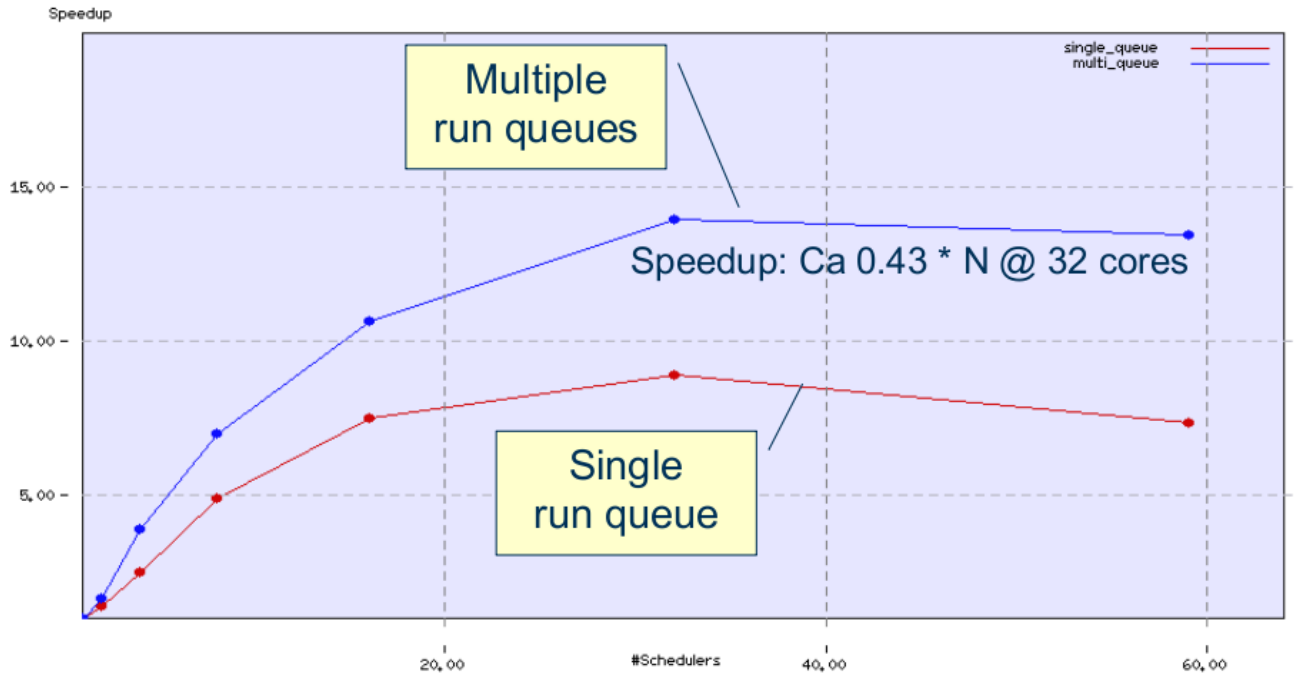
Multicore support

- To...



Multicore support

- Some benchmarks
  - Speedup of “Big Bang” on Tilera Tile64 chip
    - **1k processes, talking to each other**



Multicore support

- Demo
  - *pmap* on one core
  - *pmap* on two cores

Erlang

Distribution

- Light-weighted processes
  - 20M processes benchmark has been performed
    - **By Ulf Wiger in 2005**
  - 300 bytes of overhead for each
  - no shared state between them
- Transparent communication
  - Sending message to the local process
    - ***LocalPid ! Msg***
  - Sending message to the remote process
    - ***RemotePid ! Msg***

Erlang

Distribution

- *epmd* – Erlang Port Mapper Daemon
  - maps symbolic node names to machine addresses
    - **NodeName@Host → 192.168.1.10:12345**
- Example:
  - Mnesia: distributed DBMS for highly scalable apps
    - **Fast real-time key/value lookup.**
    - **Complicated non real-time queries mainly for operation and maintenance.**
    - **Distributed data due to distributed applications.**
    - **High fault tolerance.**
    - **Dynamic re-configuration.**
    - **Complex objects.**

Erlang

Distribution

- Demo
  - manager node sends to remote node task to execute

Erlang

Fault tolerance

- Isolation
  - When something is going to crash, let it crash
  - Crash is a regular way of handling errors
- Share nothing architecture
- No implicit synchronization
  - Spawn always succeed
  - Sending always succeed
  - Fire and forget strategy
- Nodes and processes monitoring
- Supervision trees
- Distributed applications

Erlang

Erlang/OTP behaviors

- Formalizations of most common patterns
- Divide the code into two chunks:
  - generic – provided by distribution – behavior module
  - specific – provided by developer – callback module
- Including
  - gen_server
  - gen_fsm
  - gen_event
  - supervisor
  - application
  - release
  - defined by user

Erlang

Erlang/OTP behaviors - gen_server

- Central server, arbitrary number of clients
- A way to make calls sequent
- Synchronous/asynchronous API
- Implementing callback functions for:
  - initialization of the state
  - handling calls
  - handling casts
  - handling other messages
  - termination

Erlang/OTP behaviors - gen_fsm

- Finite state machine
  - If we are in state *S* and the event *E* occurs, we should perform action *A* and make a transition to state *S'*
- Function per state
- Synchronous/asynchronous API
- All state events

Erlang

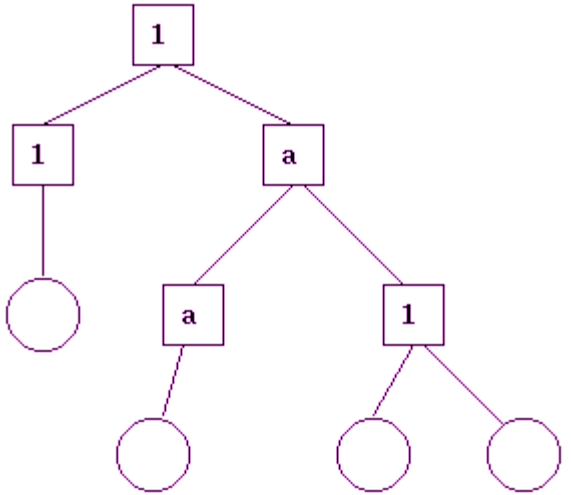
Erlang/OTP behaviors - gen_event

- Generic event manager
  - event manager – an Erlang process
  - event handler – callback module
- Publisher/subscriber pattern
- Dynamic list of handlers

Erlang

Erlang/OTP behaviors - supervisor

- Process responsible for starting, monitoring and stopping its child processes
- Tree structure
- Various restart strategies
  - *one_for_one*
  - *one_for_all*
  - *rest_for_one*
- Maximum restart frequency
- Fault tolerant approach



Erlang

Erlang/OTP behaviors - application

- Reusable component that can be started and stopped as a unit
- Described by application resource file containing
  - Name
  - Version
  - Modules
  - Dependencies
  - Environment variables
  - etc
- Can be started using application callback module

Erlang

Erlang/OTP behaviors - release

- Ties several applications into one system
- Described by release resource file containing
  - release name and version
  - ERTS version
  - list of applications (with their versions)
- Easily preparing target systems with boot scripts and release packages

Erlang

Erlang/OTP behaviors

- Demo
  - application structure example

```
{application, eptic, [
        {description, "Eptic Erlang Web application"},
        {vsn, "1.3"},
        {modules, [e_cache,e_cluster,e_conf,e_error,
                e_db_couchdb,e_db,e_db_mnesia,
                e_dict,e_dispatcher,e_file,e_json,e_lang,
                e_mod_gen,e_mod_inets,e_mod_yaws,e_multipart_inets,e_multip
aws,
                eptic,e_session,e_validator,e_component,e_cache_ets,e_cache
,e_annotation,e_user_annotation,
                e_logger, e_logger_viewer,
                e_start]},
        {applications, [kernel, stdlib]},
        {registered, []},
        {env, [
                {upload_dir, "/tmp"},
                {template_expander, wpart_xs},
                {template_root, "templates"},
                {node_type, single_node}
        ]},
        {mod, {eptic, []}}
]}.
```

```
.
|-- Emakefile
|-- LICENSE
|-- Makefile
|-- Mnesia.nonode@nohost
|-- bin
|-- erts-5.6.5
|-- lib
|-- lib
|   |-- compiler-4.5.5
|   |-- crypto-1.5.3
|   |-- edoc-0.7.6.2
|   |-- inets-5.0.12
|   |-- kernel-2.12.5
|   |-- mnesia-4.4.7
|   |-- runtime_tools-1.7.3
|   |-- sasl-2.1.5.4
|   |-- ssl-3.10
|   |-- stdlib-1.15.5
|   |-- tools-2.6.2
|   |-- xmerl-1.1.10
|   |-- yaws-1.73
|   `-- yaws-1.80
|-- log
|-- pipes
`-- releases
```


Erlang

Questions