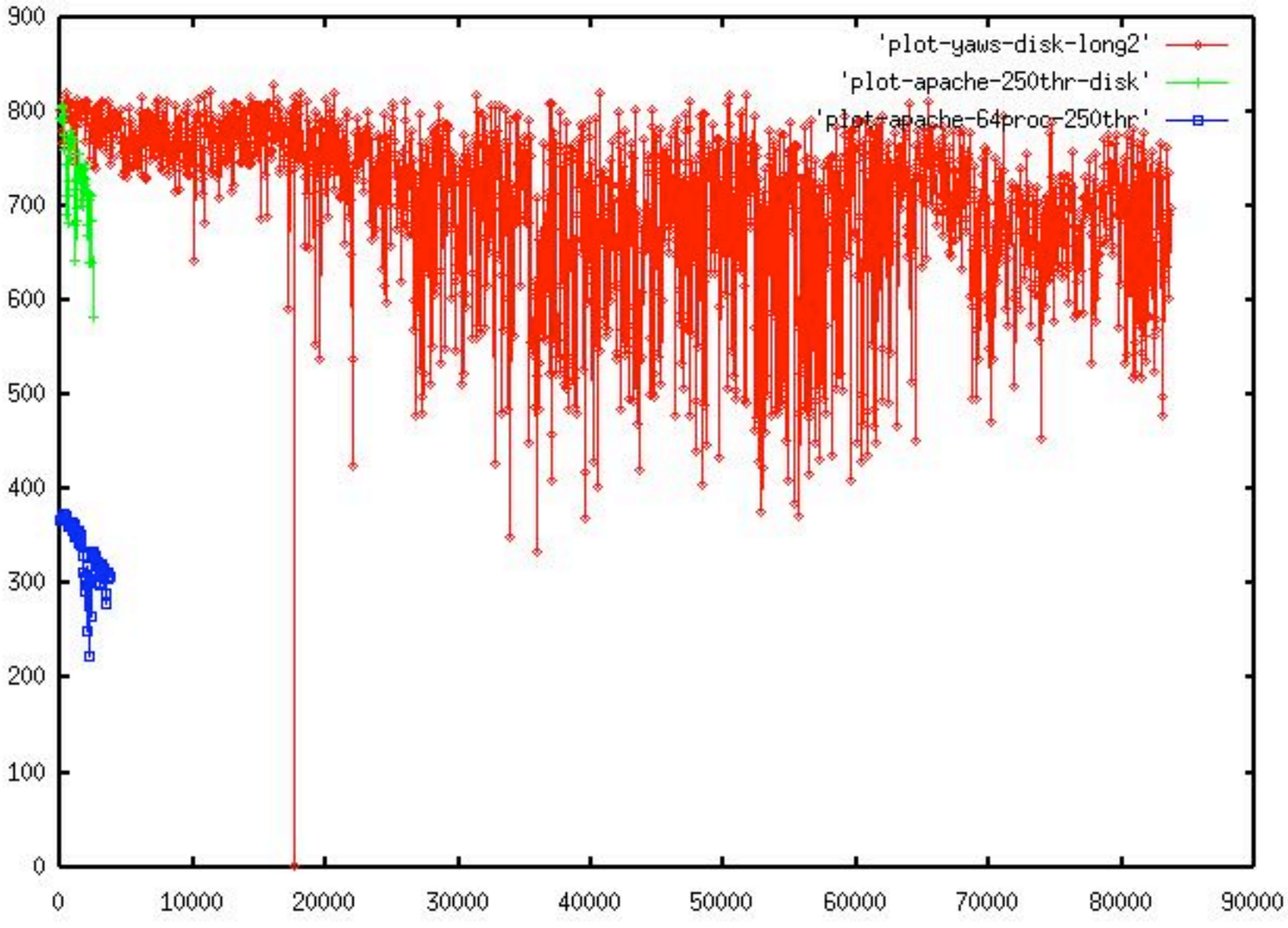
Erlang Concepts

Samuel Tesla
samuel.tesla@gmail.com

YAWS vs. Apache



http://www.sics.se/~joe/apachevsyaws.html

Matching & Variables

Matching

- `A = 42.`
- `{A, B} = {42, b}.`
- `[Head | Tail] = [a, b, A].`
- `{a, [b, 42]} = {Head, Tail}.`

Single Assignment

- Once a variable is bound in a scope, it cannot be rebound.
- This behavior is required for matching to work right.
- Variables **cannot** change once bound.
- Much easier to reason about and debug.

Function Signatures

- `value({tag, Value}) -> Value.`
- `first([First | _Rest]) -> First.`
- `div(_X, 0) -> undefined;`

  `div(X, Y) -> X div Y.`
- `is_even(X) when X div 2 =:= 0 ->`

  `  true;`

  `is_even(_) -> false.`

Recursion

- In order to make variables "mutable" make them parameters.
- Recurse with new values.
- Tail-Call Optimization prevents stack overflow.
- Many algorithms have to be redesigned to work with tail-recursion.

Example

```
sum(List) -> sum(List, 0).
```

```
sum([], Sum) -> Sum;
sum([First | Rest], Sum) ->
  sum(Rest, First + Sum).
```

Higher-Order Functions

- Functions as first-class values.
- Functions that return other functions, or take other functions as parameters.
- Functions can capture the state when they were created to produce closures.
- `fun(X,Y) -> X + Y end.`

Example

```
sum(List) ->
    lists:foldl(fun(X, Sum) ->
                        X + Sum
                end, 0, List).
```

Concurrency & Distribution

Processes

- User-space “green” threads
- VM manages processes across kernel threads to maximize CPU utilization across all available cores.
- Because processes are the basic modeling tool, Erlang programs tend to scale linearly as cores are added.

Processes

- Processes are self-contained.
- Each process has its own stack.
- GC is per-process.
- If a process crashes, it does not affect any other process.

Nodes

- Each OS process running the VM is a node.
- Nodes are completely separate from each other.
- Nodes can connect to form a cluster.
- Security bewteen nodes is all or nothing.
- Processes can be started on any node from any other node in a cluster.

Messages

- Messages are sent from any process to any other process in the cluster.
- The VM guarantees that messages will be delivered in order.
- The VM does **not** guarantee that messages will be delivered.
- `Pid ! {tag, Value}.`

Links

- Any two processes can be linked.
- If a process exits with an abnormal status, all linked processes will exit with the same status.
- System processes trap exits. Instead of exiting, they receive a message with the Pid and exit status of linked processes.

Monitors

- A process may request notification when another process exits without becoming a system process by monitoring it.
- The monitoring process will receive a notification if the process exits, the node that process is running on disconnects, or if either is already the case when the monitor is requested.

Fault Tolerance & Reliability

Joe's Rule

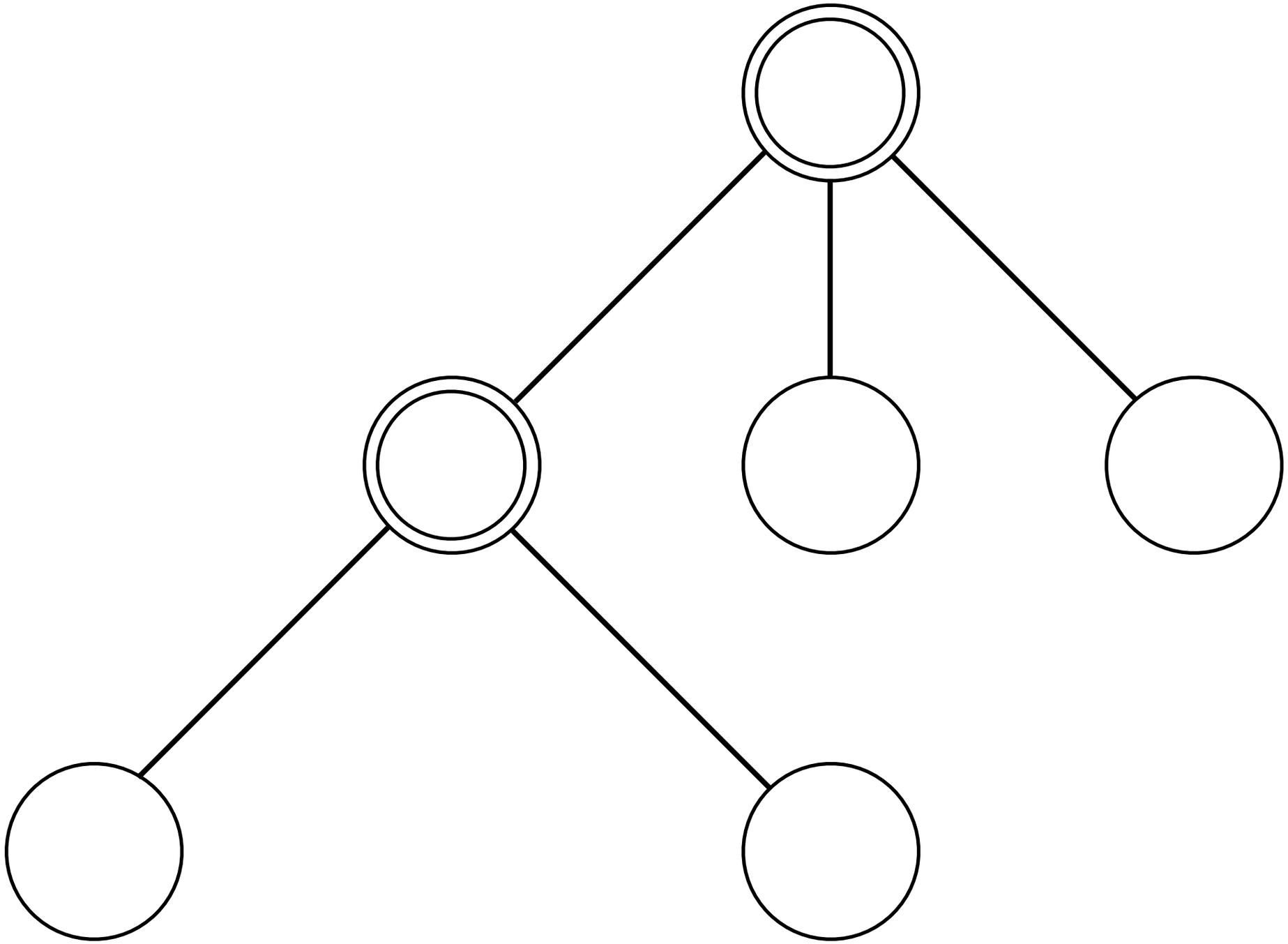
"To make something fault tolerant, we need at least two computers. One computer does the job, and another computer watches the first computer and must be ready to take over at a moment's notice if the first computer fails."

– Joe Armstrong, *Programming Erlang*

Supervision Trees

- Worker processes do the real work.
- Supervisor processes monitor workers and restart them as needed.
- Supervisors can also monitor other supervisors.
- This sort of structure is called a **supervision tree**.

Supervision Trees



Live Code Reloading

- The VM can reload a module from disk.
- In-progress function calls will complete using the code that was current when they started.
- New function calls will use the most current loaded code.

Questions?