ERLANG

The Language from the Future?

Damien Katz
Couchbase CTO, Creator of Apache CouchDB

AGENDA

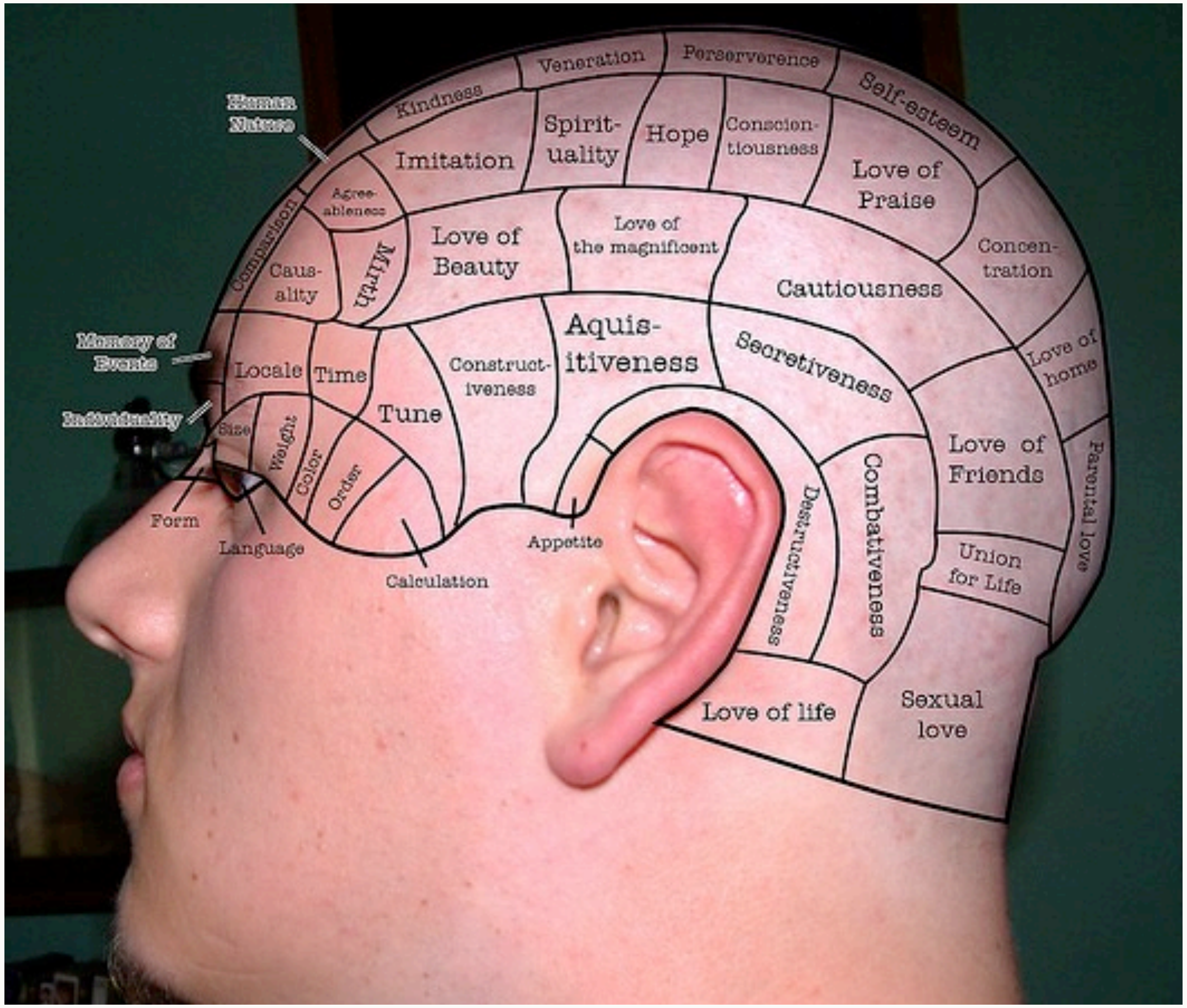
- What is Erlang
    - From a High Level, Why is Erlang Cool?
- Why is Erlang from the future?
    - Accidentally designed for multi-core
- Erlang and CouchDB
    - Great fit, but some problems.
- So when would **You** use Erlang?

WHY IS ERLANG SO COOL?

A LITTLE BACKGROUND

- Functional Concurrent Programming Language
- Built by Ericsson in 80s and 90s
- Designed for Reliability and Concurrency
- Used in commercial telecom switches with great success and reliability

WHAT IS ERLANG LIKE?



WHAT IS ERLANG LIKE?

- It’s weird.
- It’s simple.
- It’s extremely productive
- It’s extremely reliable
- The design of the VM is beautiful
- It’s kinda slow

IT'S WEIRD

IT’S WEIRD

- The syntax looks like nothing you’ve used (unless you’ve used Prolog (you haven’t))
- No looping
- No destructive updates.
- The `if`expression is almost useless

IT'S ODDLY SIMPLE

IT'S ODDLY SIMPLE

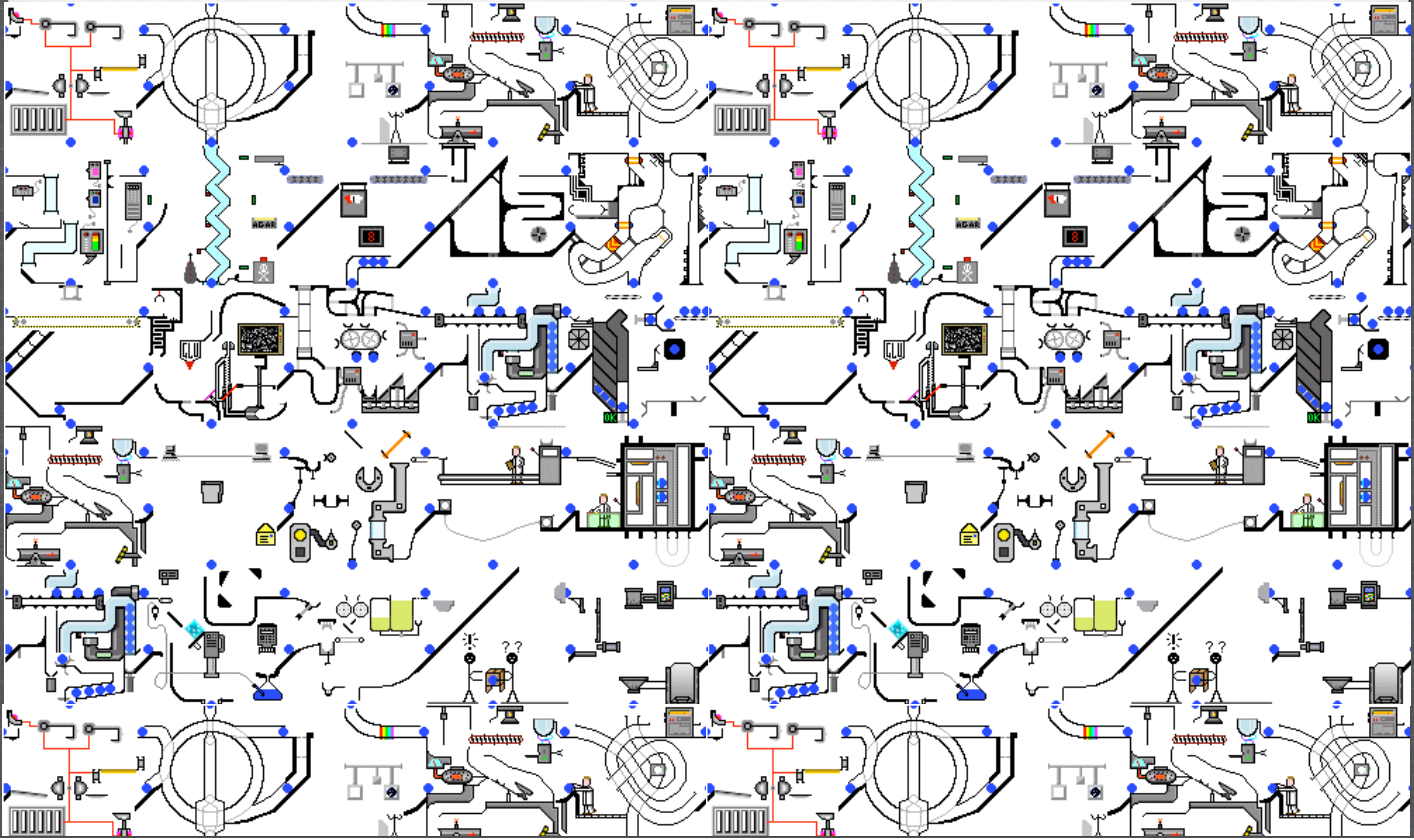
- You get lists, tuples, numbers, floats, atoms, binaries, and some Erlang specific types (pids, unique refs).
- No classes, no OO, no user defined types.
- It's functional, so you can create closures and pass around functions.
- It's like LISP, without all the Macro Power!

WTF? HOW DO YOU GET ANYTHING DONE?

WTF? HOW DO YOU GET ANYTHING DONE?

- I dunno. You just do. The weirdness wears off, enlightenment comes and suddenly you produce small, reliable code.
- I still miss classes, structs, better ways to organize code.
- The real magic is concurrency and error handling

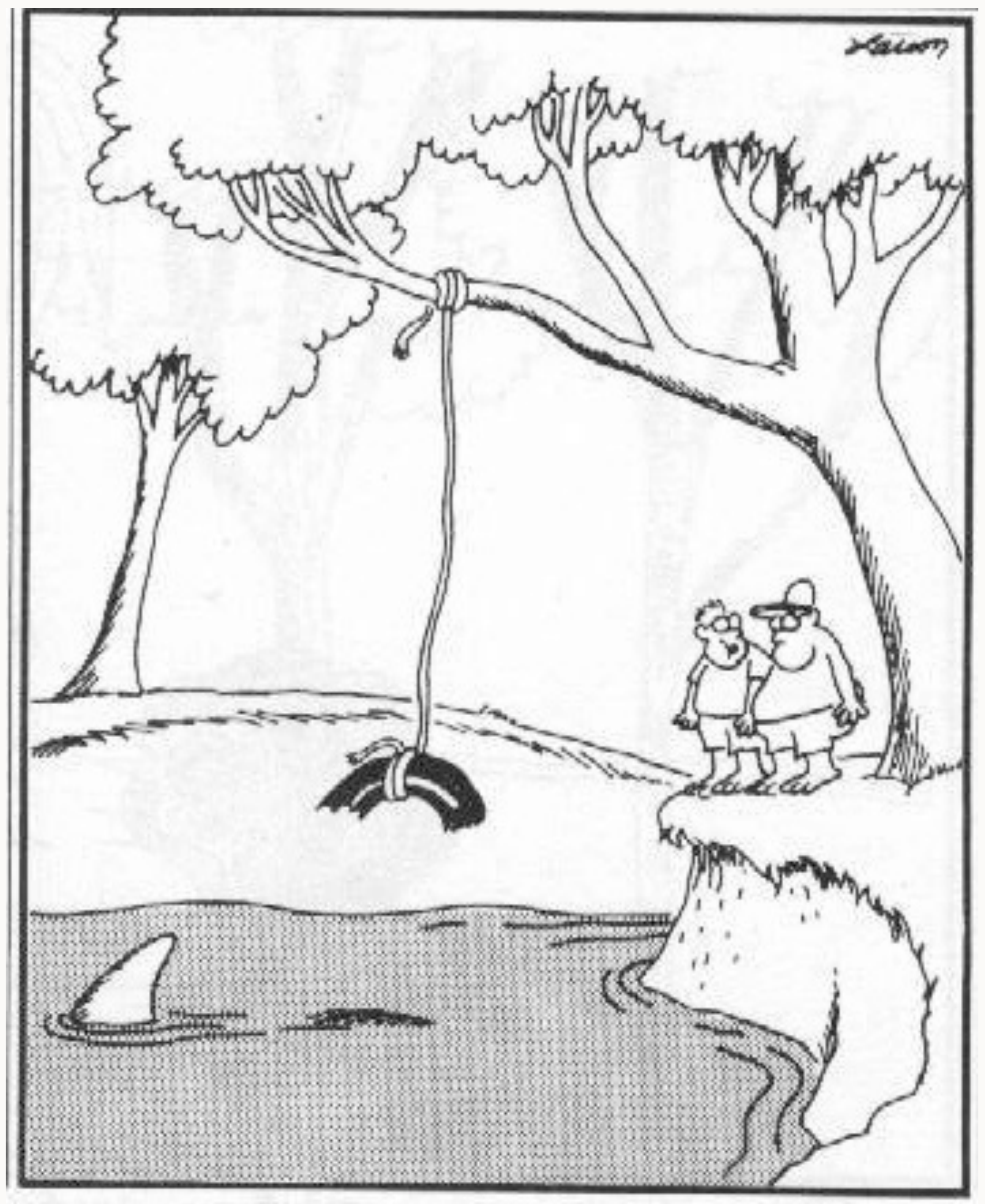
CONCURRENCY



CONCURRENCY

- Erlang “processes” are very lightweight. Lighter than threads. Almost like an object in other languages. But like OS processes in concept.
- Can scales to millions of processes per VM.
- Processes “protect” their state. Send a message to ask for some state, or to modify state.
- Each process has it’s own heap. Makes concurrent GC possible without long VM pauses.

ERROR HANDLING

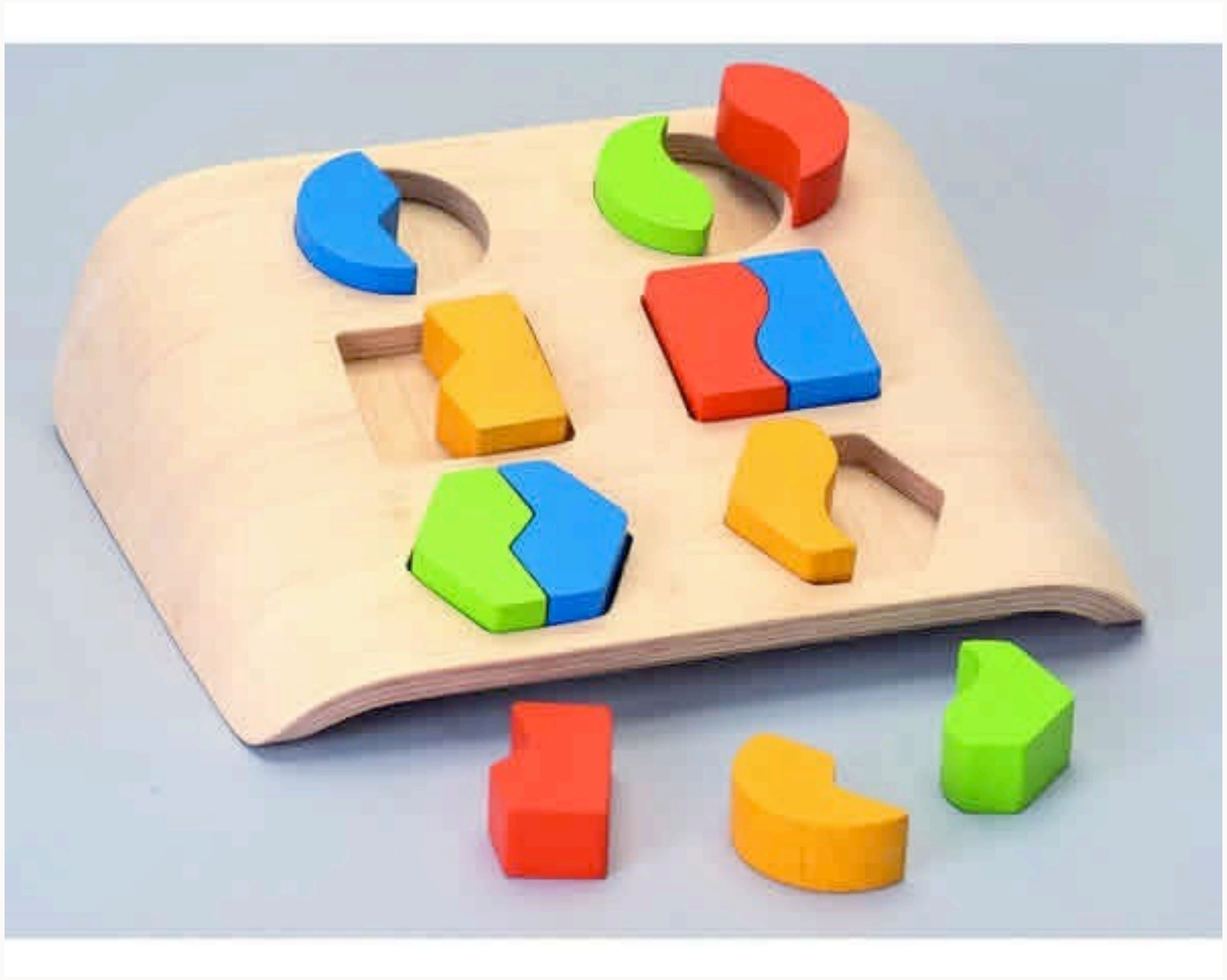


"Listen . . . You go tell Billy's mother, and I'll start looking for another old tire."

ERROR HANDLING

- Concurrency and Error handling go hand in hand.
- Allows application code and error handling code to be separated.
- Generally, don't worry about robustness and error handling. Let it crash and another process will deal with the error.
- Simplifies code, makes it MORE reliable.

PATTERN MATCHING



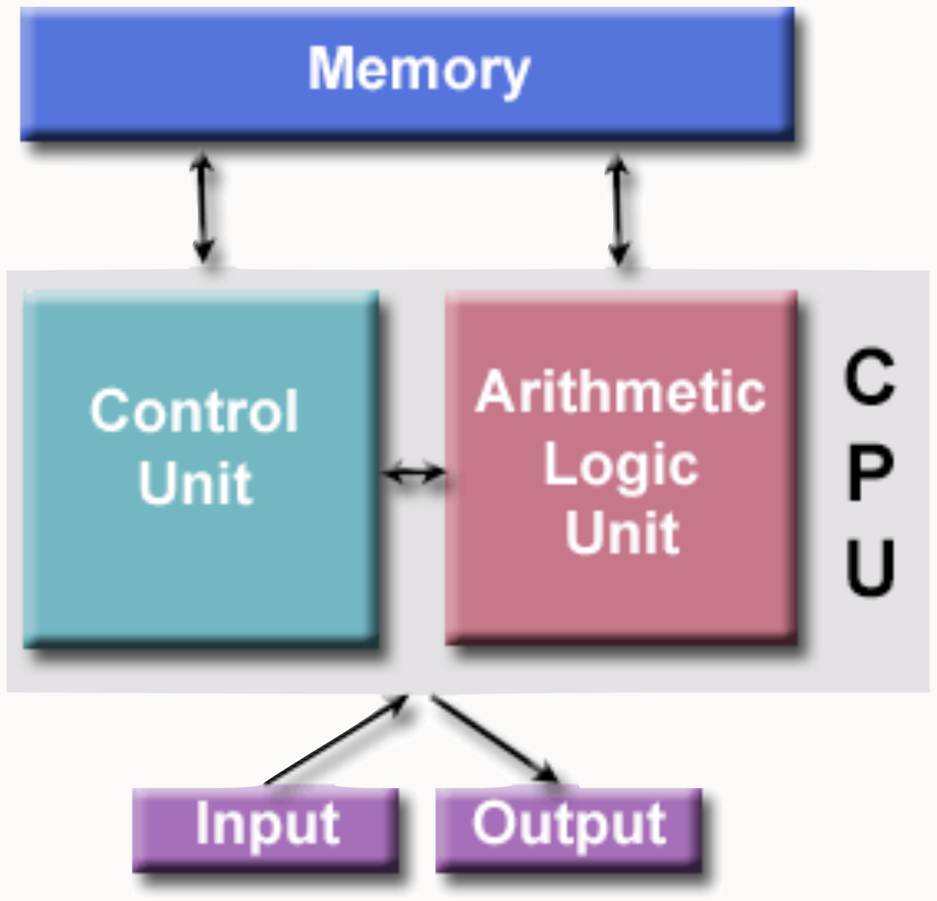
PATTERN MATCHING

- I wish every language had this.
- Key to making Erlang code concise and selective message receiving easy.
- Makes up for busted `if` expressions.

SO, HOW IS IT FROM THE FUTURE?

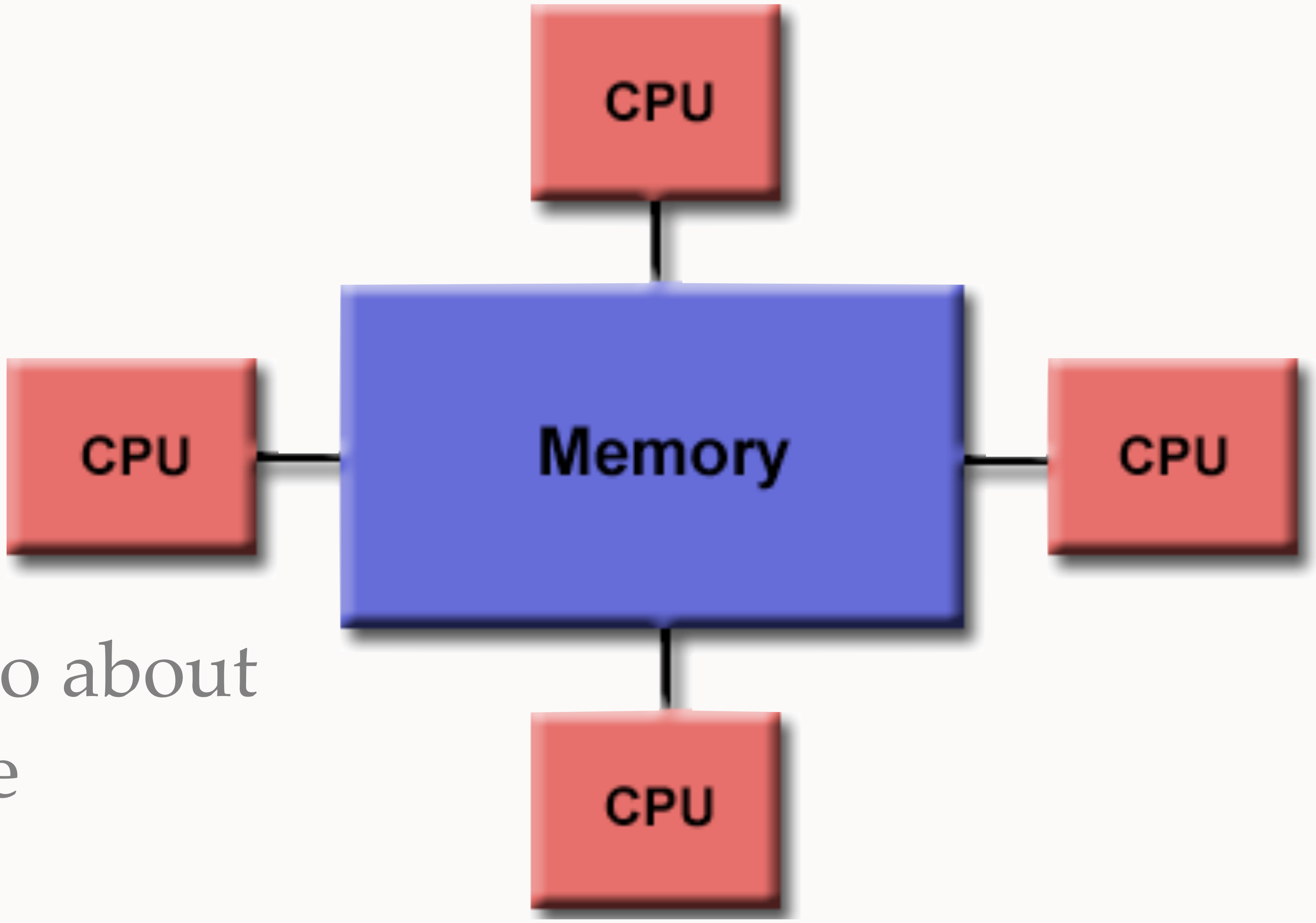
THE PAST

- One CPU
- Uniform Main Memory
- Concurrency was about making a single processor do many things



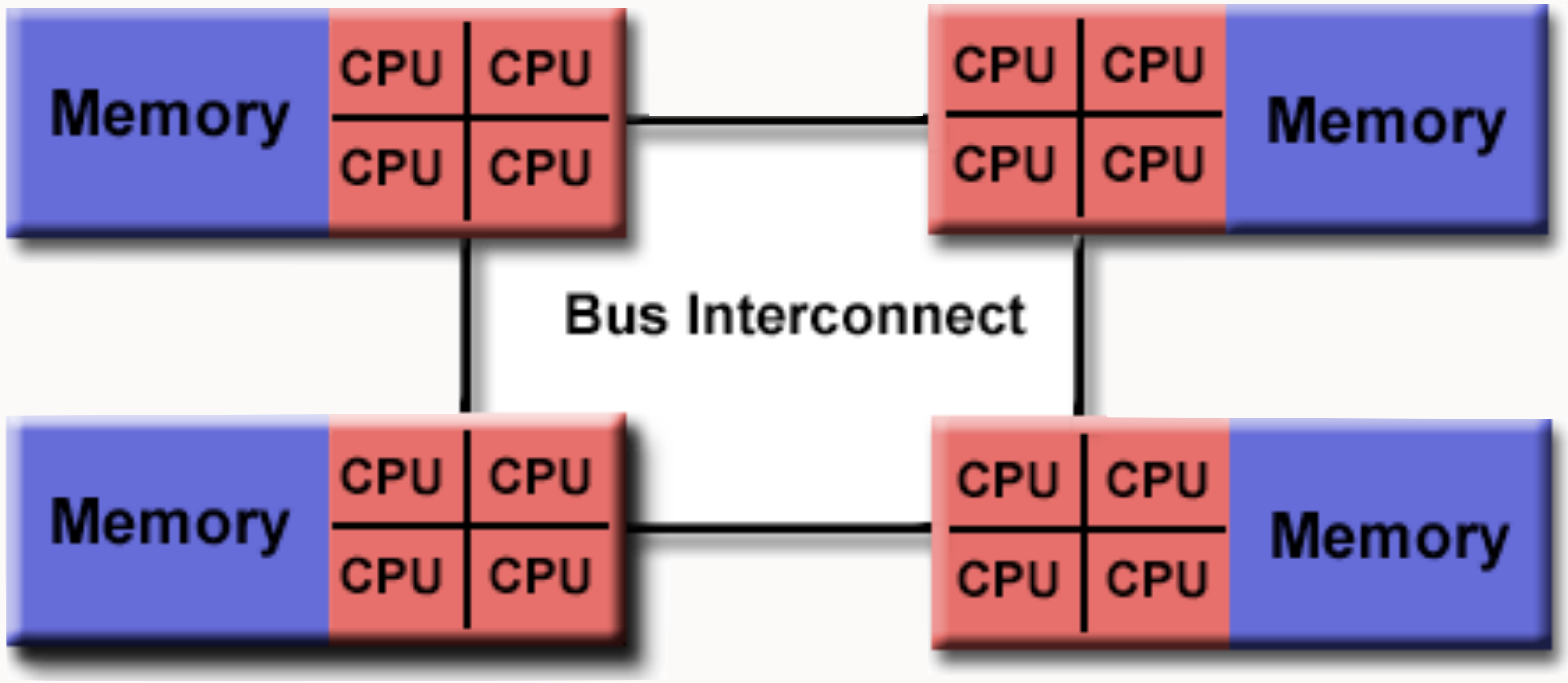
MULTICORE PRESENT: HOW WE THINK

- Lots of CPUS
- Lots of Memory
- Concurrency is also about exploiting all those processors



MULTICORE PRESENT: REALITY

- Lots of Cores, each with levels of cache (L1, L2, L3)
- NUMA - Non-uniform Memory Access
- Physics!

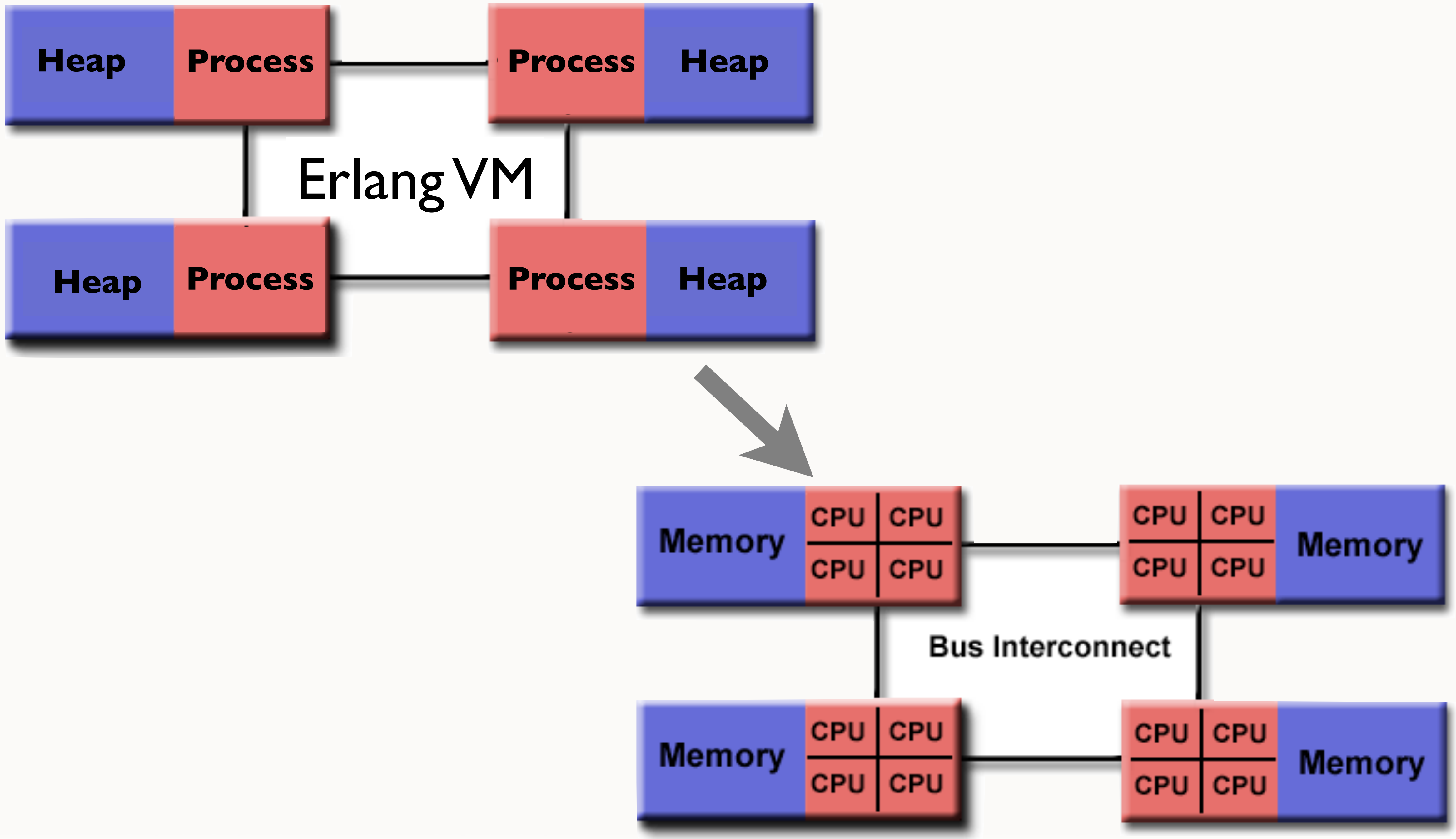


MOST LANGUAGES DON'T MODEL REALITY

MOST LANGUAGES DON'T MODEL REALITY

- Shared memory modeled as one big space
- Lots of memory, yet processing small amounts of tightly packed memory is ***much*** faster.
- No way to tell the language that some things should be close to our processor
- Inter-Process Communication maps more closely to Inter-*Processor* Communication

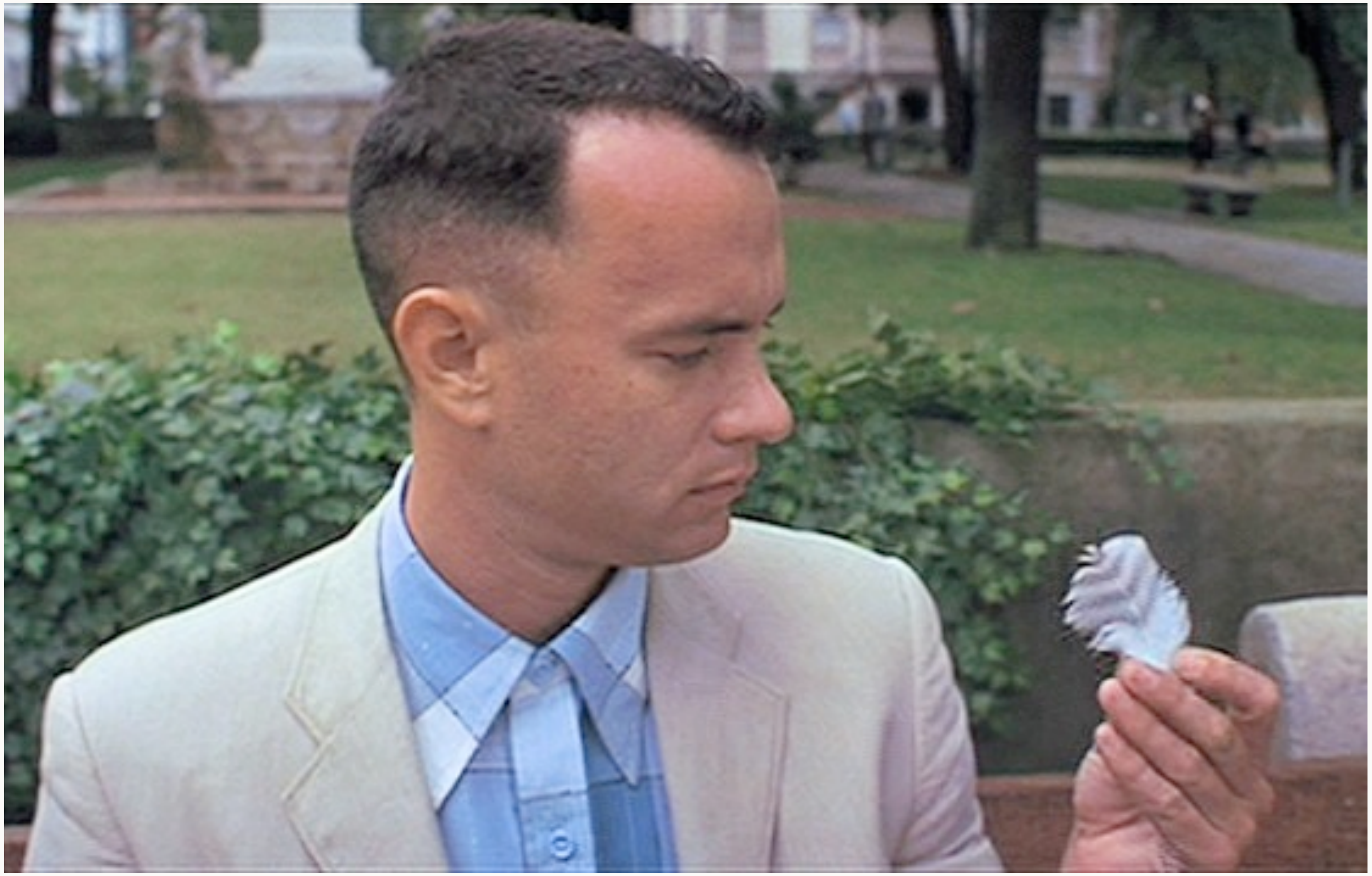
ERLANG MAPS WELL TO MODERN MULTICORE



ERLANG MAPS WELL TO MODERN MULTICORE

- Erlang makes inter-process communication easy.
- Each Erlang process has it's own heap.
- Erlang process heaps tend to be much smaller than shared memory heaps, fitting more relevant data into processor cache.
- Getting/setting data from another process is an explicit message send, similar to memory bus communication of processors.

ACCIDENTAL MULTI-CORE AWESOMENESS



ACCIDENTAL MULTI-CORE AWESOMENESS

- Designed originally to run on small systems, like network switches.
- Only got multi-core aware a few years ago.
- Could always take advantage of multi-core, but had to use less efficient OS level IPC.
- But the model is right, implementation getting better and better...

ERLANG AND COUCHDB

DATABASES NEED:

- Concurrency - CHECK!
- Reliability - CHECK!
- Monitoring - CHECK!
- Rapid Recovery - CHECK!
- Distributed Computing - CHECK!
- High Performance - Ummmm

ERLANG IS ALMOST PERFECT!

- It ticks most of checkboxes!
- It’s the Language of Future!
- What’s the problem?

ERLANG'S SYNTAX IS THE BIG PROBLEM

ERLANG’S SYNTAX IS THE BIG PROBLEM

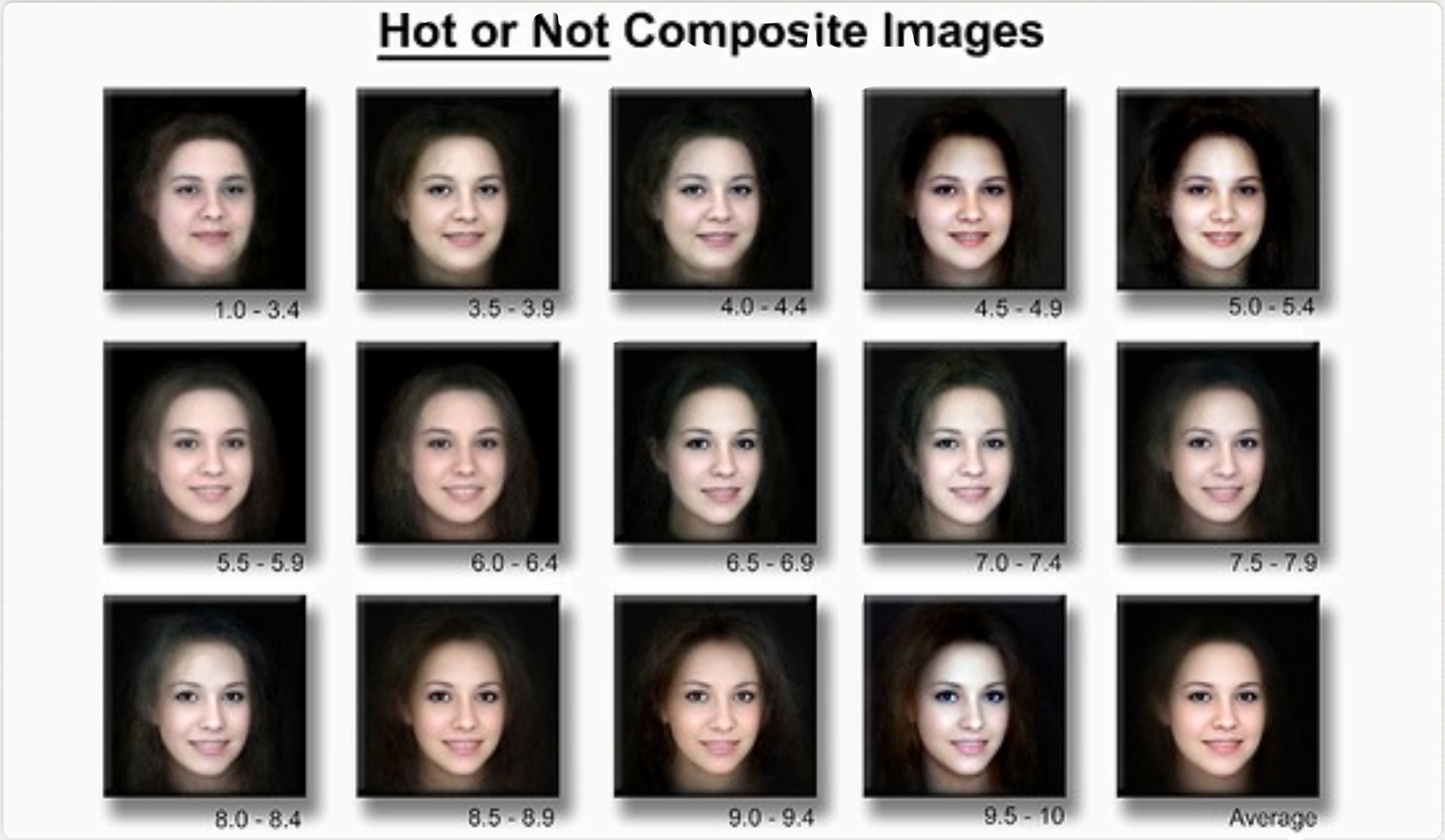
- I love me some Erlang.
- I see past the weirdness.
- I’m not the average developer.

WEIRD SYNTAX MAKES ERLANG SLOW!

WEIRD SYNTAX MAKES ERLANG SLOW!

- It’s weird!
- Weird syntax prevents massive adoption.
- Massive adoption leads to massive investment
- Massive investment leads to better tools, better and faster VMs.

JAVA WAS SLOW TOO, BUT IT WAS FAMILIAR



JAVA WAS SLOW TOO, BUT IT WAS FAMILIAR

- But it had a syntax that was familiar C and C++ developers.
- Java was easier and simpler in lots of ways developers cared about.
- Java got massively popular.
- Then Java got massive investment.
- Then Java got pretty damn fast.

Erlang syntax was just a little more normal!

WHEN SHOULD YOU USE ERLANG?

BACKEND, HEAVY LIFTING SYSTEMS

ROBUST, RELIABLE, LONG LIVED

DISTRIBUTED SYSTEMS

YOUR ENGINEERS CAN FILL IN GAPS

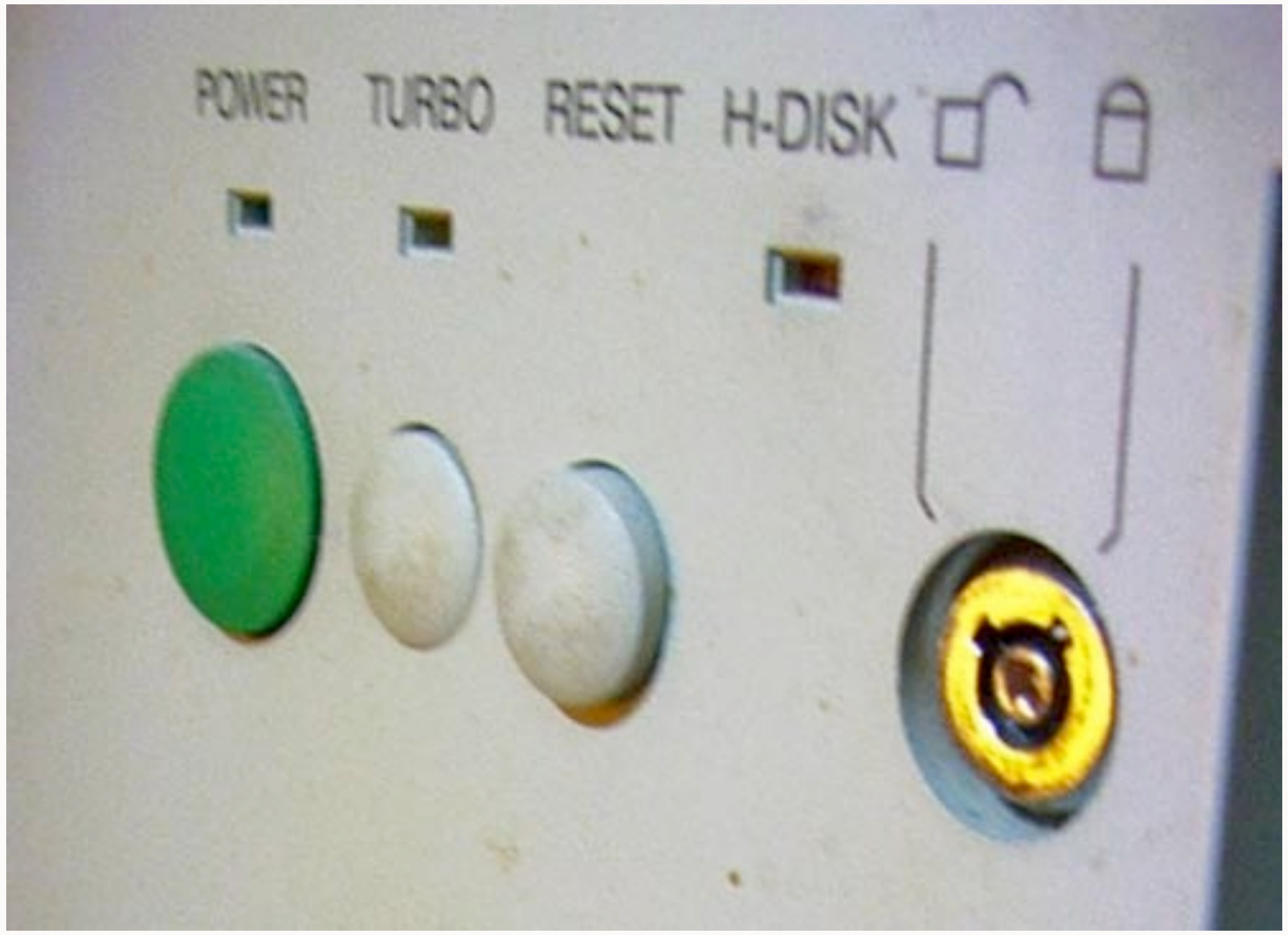
- Less tools, less libraries, less shared knowledge.
- Erlang VM sometimes does weird things. (true of most VMs though)

WHEN SHOULD YOU *NOT* USE ERLANG?

MANIPULATING TEXT OR HTML (CUMBERSOME)

CPU INTENSIVE TASKS

- CPU is cheap but...
- Faster is better
- Customers want low cost cloud solution (CPU still costs $$, and lower CPU is an advantage)



ENGINEER TURNOVER IS A PROBLEM

- Good programmers are hard to find. Erlangers even harder.
- Some might argue that letting them program in Erlang will bring better engineers and reduce turnover. Could be!

CAN I GET ERLANG BENEFITS IN OTHER LANGUAGES?

ERLANG BENEFITS IN GC'D LANGUAGES

ERLANG BENEFITS IN GC'D LANGUAGES

- You can get the distributed features but..
- Process isolation and error handling and concurrent garbage collection... FORGET IT.
- Most garbage collected languages can fake it, but you need both language and VM support to do it right.
- Leaky abstractions are leaky.

ERLANG BENEFITS IN C/C++

We're Old Enough to Choose
www.ABetterPutnam.com
We Want our Cake and Eat It Too!!
www.ABetterPutnam.com
We're Old Enough to Choose

ERLANG BENEFITS IN C/C++

- The Erlang VM is written in C.
- C/C++ let's you control memory and concurrency. So....
  - Use as little shared state as possible
  - Break down tasks, use memory pools and finite state machines when possible.
  - Avoid locks, use messages instead.
  - ~10x the code size of Erlang. But 5-10x CPU performance.. And coding and debugging will take 5-10x as long.

OR JUST USE ERLANG!

- Unless you sell a commercial product, or compete on absolute performance, your biggest concern is engineer productivity and maintenance.
- Save money on coding and debugging, spend it on more CPUs
- More reliable in production too.

COUCHBASE CUSTOMERS:

COUCHBASE: A HYBRID OF C/C++ AND ERLANG

COUCHBASE: A HYBRID OF C/C++ AND ERLANG

- We use Erlang for distributed stuff. Cluster management and replication.
- Migrating performance sensitive code away from Erlang. Increasing performance and reducing CPU. It’s more code, and longer debugging.
- But our customers care not. They don’t want compromises. They want fast, reliable AND cheap to run.

THANK YOU!

couchbase.com

damienkatz.net

damien@couchbase.com