# Introducing Erlang into a large company

## Tactics

## A Test Drive

- Choose a nasty problem that needs to solved that nobody wants to touch
- Bonus if there has already been at least one failure
- Even better, a problem that doesn’t imply a long term commitment

## What You Get

- Nobody is going to argue about your choice of tool
- Failure will be no worse than the last guy
- You will have a solid sense of language capabilities
- You will  have a   convincing  case if you succeed.

## Merging Onto the Highway

- Choose a problem that you have solved before
- Which presents known difficulties for your usual tool chain
- Take advantage of current tools – don't reinvent wheels
- Focus on time-to-market
- Management is always looking for a reason ;)

## What You Get

- Protection - it is harder to cancel a performant system in production than an experiment
- Writing a fault-tolerant non-stop systems is not good for job security
- Good reputation
- Opposition
- Blame
- Disregard

## Reaching 60 MPH

- Choose a nasty problem – a critical one
- Use Erlang for the a central component
- Bonus if you can demonstrate interop with standard libraries and other languages

## What You Get

- Your life will be easier as the system will stabilize faster
- You will end up integrating the standard OTP with your internal build tools
- You will write interfaces for common internal libraries
- This will make it easier for coworkers to bootstrap Erlang into their usual workflow

## Some Successes

- Reduced migration time ( delicious )
- Scaling serving infrastructure ( BOSS )
- Indexing pipeline control (Vertical Search Platform)

## Obstacles Encountered On the Way

## Bumps Along the Road . . .

- Describing distributed asynchronous architectures to most engineers is HARD
- When you remove the usual boot/restart cycles you uncover memory management problems
- Doing this doesn't make you popular
- Pushing engineers out of their comfort zone is risky

## . . . Lots

- For/if/while loops are hard imperative programmers to let go of
- Just because Erlang scales out of the box to ~100 nodes doesn't mean it is a “real distributed language “
- Java which scales to 1 out of the box is ☺
- Security concerns

## What Erlang Has Taught Us

## Some Surprising Things

- C/C++/Java programmers get really annoyed when you call Erlang a systems language ;)
- Single assignment is a great simplifier
- Invariant variables don’t slow things down as much as supposed
- We haven’t missed objects very much
- Erlang seems to be what Alan Kay refers to as the essence of OOP

## . . .

- How easy and directly enlightening reading a stack trace can be
- The satisfaction that comes from debugging a running system with precision and flexibility
- Functional decomposition makes system tracing so much easier
- A lot of features that people are trying to develop for large scale systems are already solved in Erlang

## . . .

- Being a fan-boy is fun - but there is room for other languages and Erlang doesn't have to be tops for everything

## Some New Approaches

- More emphasis on specification
- Think more type less
- ‘!’ and ‘receive’ are more useful than ‘for’ and ‘extends’ for many of the problems we have to solve
- Program for what you expect rather than what you fear
- Erlang allows you to focus on the harder problems in distributed computing

## It Has Been A lot of Fun

## Thanks to . . .

- Joe, Robert, Mike, Claes . . . .
- The OTP team
- Erlang Consulting
- The Erlang Community