Intro to Erlang

Concurrent real-time software for a concurrent real-time world

keyfunda.

Outline

0. Installing Erlang (offline)
1. Executive summary of Erlang and applications
2. Basics
   - Shell and expressions
   - Modules, forms, funs
   - Tuples and Lists, Recursion
   - Code: Simple prime sieve
3. Processes and messages
   - Spawning processes
   - Code: Parallel prime sieve
   - Code: Comparison with node.js and vert.x
4. Distributed Erlang
5. Data storage: ETS and Mnesia
6. A simple server in Erlang
7. A roadmap for learning OTP
8. A sip of Elixir
9. Homework: Modeling an epidemic with Erlang

TL;DR

☞ http://www.fastcolabs.com/3026758/inside-erlang-the-rare-programming-language-behind-whatsapps-success

FAST COMPANY

Inside Erlang, The Rare Programming Language Behind WhatsApp's Success

Facebook's $19 billion acquisition is winning the messaging wars thanks to an unusual programming language.

By Ainsley O'Connell 2 Notes / 540 Tweet / 627 Like /

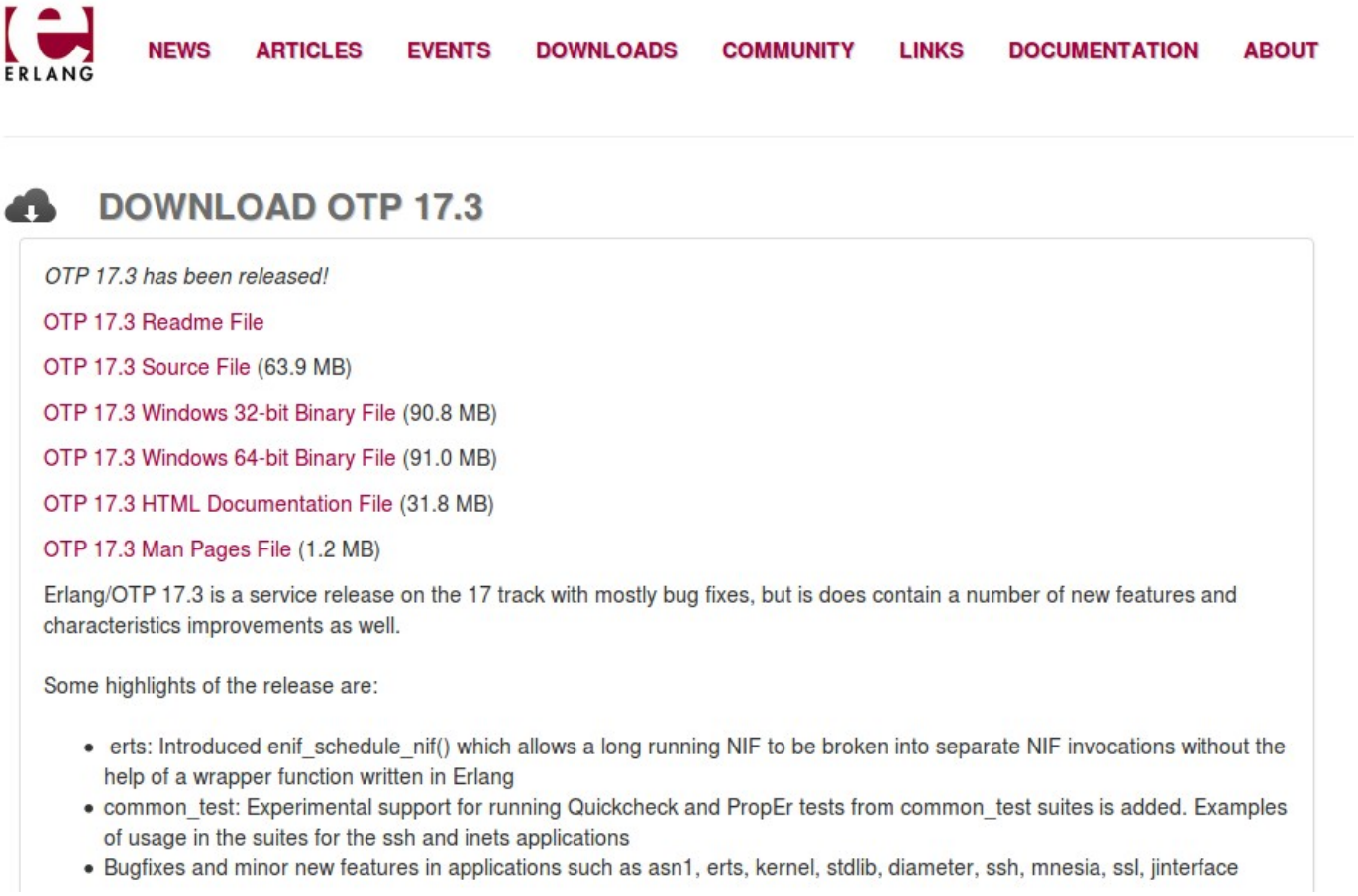
How do you support 450 million users with only 32 engineers? For WhatsApp, acquired earlier this week by Facebook, the answer is Erlang, a programming language developed in the '80s that is finally having its moment in the spotlight.

0. Installing Erlang

Installing Erlang

☞ http://www.erlang.org/download.html

CAUTION:
We are not responsible for what this may do to your computer.



1. Executive summary of Erlang and applications

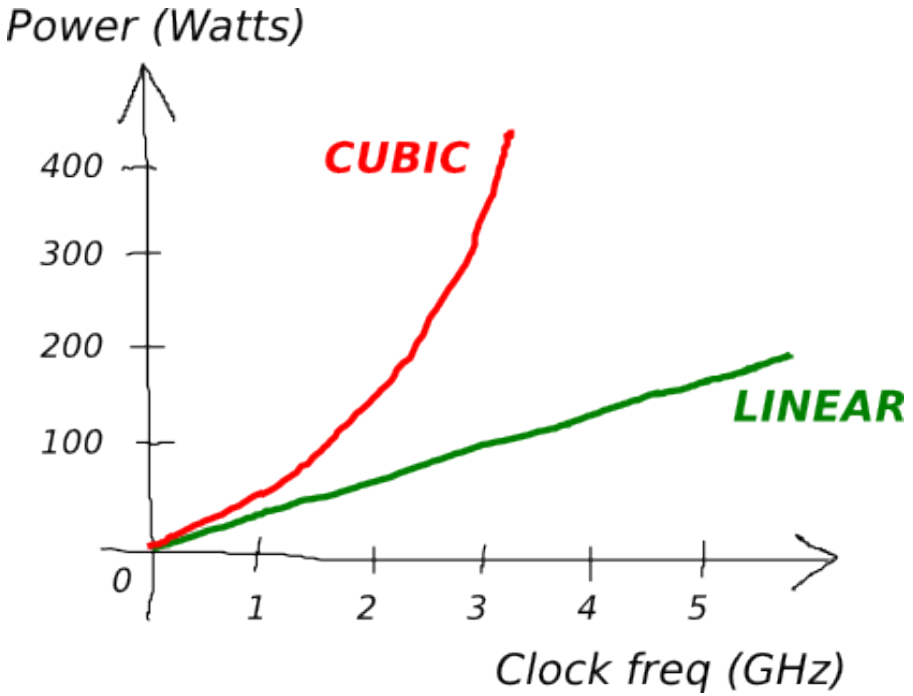
Key features of Erlang

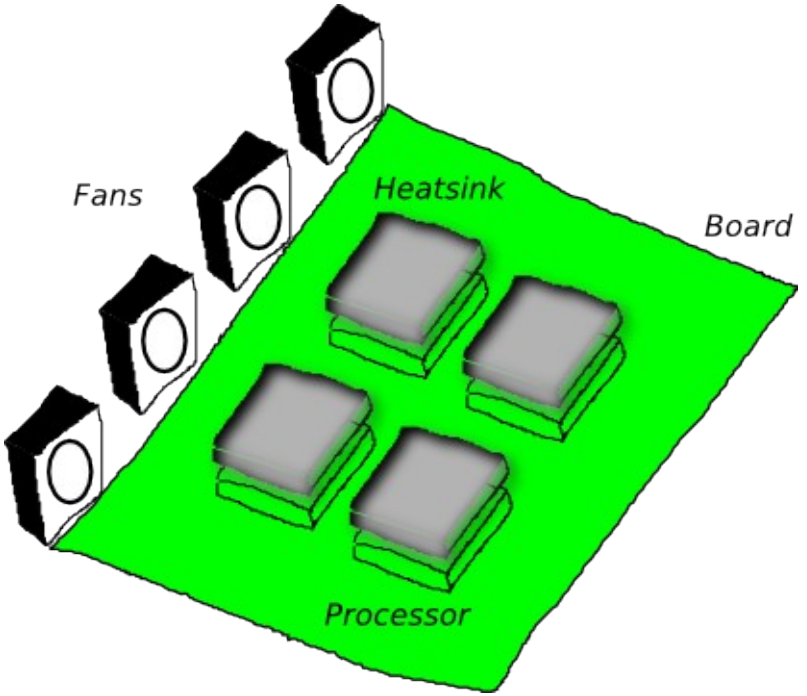
1. Scalable concurrency for multi-core
2. Functional programming (mostly) free of side-effects
3. Built-in distribution – well suited for clusters
4. **Erlang virtual machine – BEAM**
5. **OTP – Open Telecom Platform**

Power wall & Multi-core processors

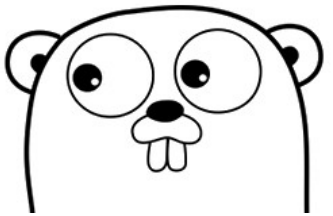
http://www.keyfunda.com/blog/2014/9/17/semiconductor-scaling-and-concurrent-clouds-part-i

- Transistor scaling – Power as ***CUBE*** of clock freq
- Reduce clock, increase cores (within same TDP)





Concurrency vs Parallelism



Rob Pike, Unix guru & a creator of Go language

http://blog.golang.org/concurrency-is-not-parallelism

Concurrency

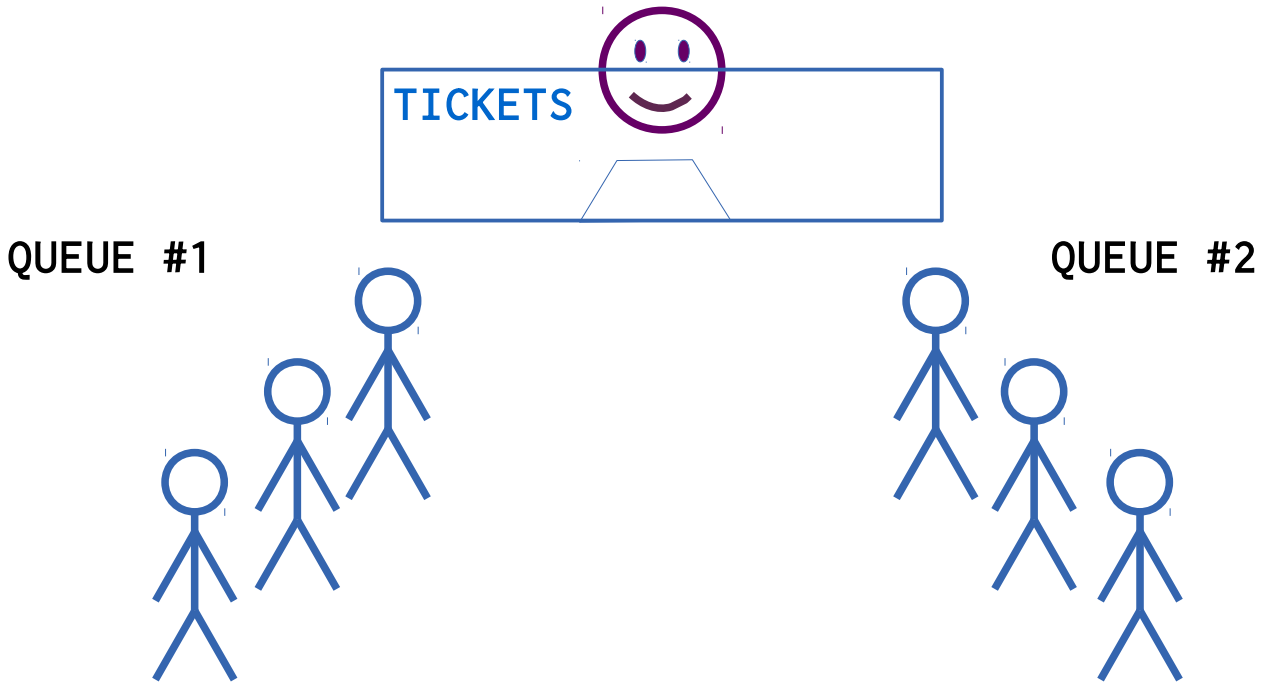
Programming as the composition of independently executing processes.

(Processes in the general sense, not Linux processes. Famously hard to define.)

Parallelism

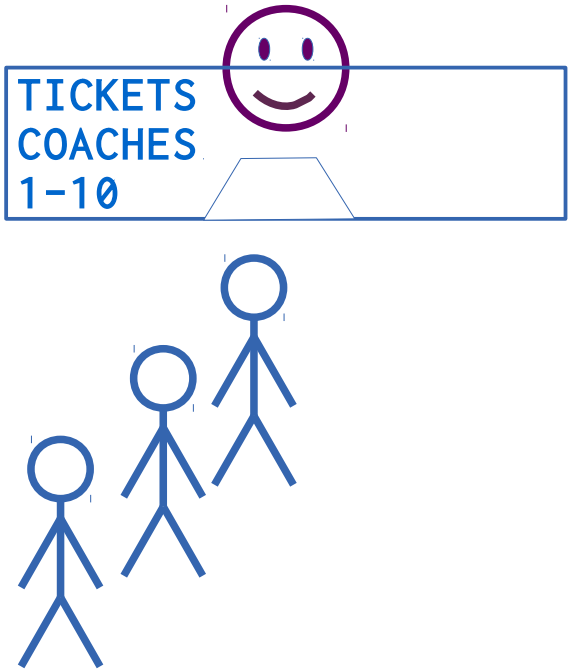
Programming as the simultaneous execution of (possibly related) computations.

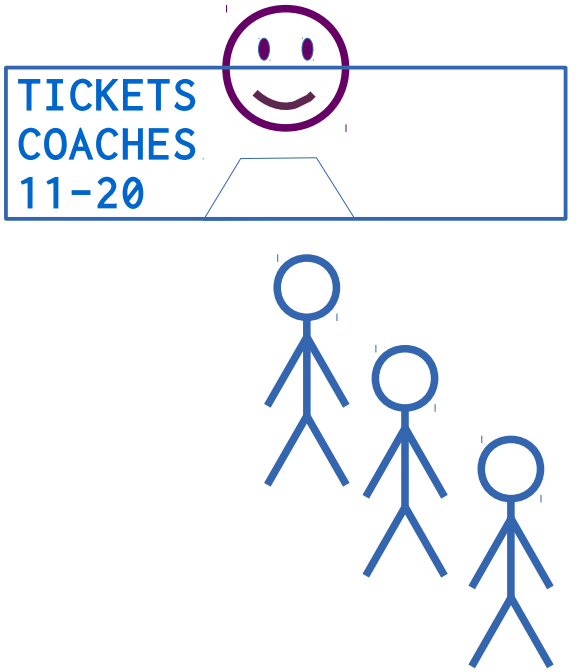
Single Core Concurrency



- Clerk is smart enough to **compose** work into indep. pieces
  - *While credit card approval for Q#1 is being processed*
  - *Clerk keeps busy entering details for passenger in Q#2*
- At an instant of time only one operation is performed
- But both queues are moving, on a larger time scale

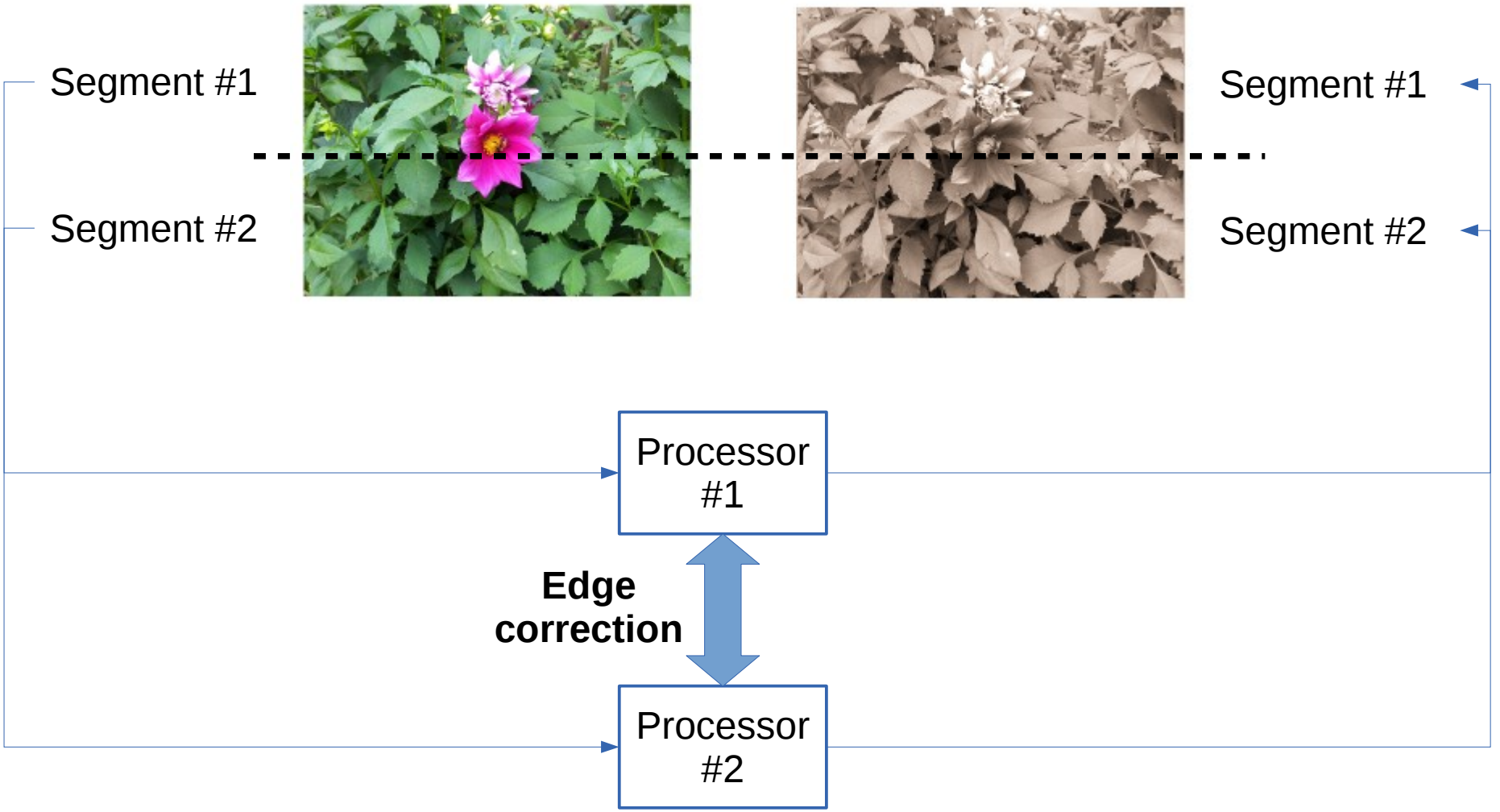
Multi Core Parallelism





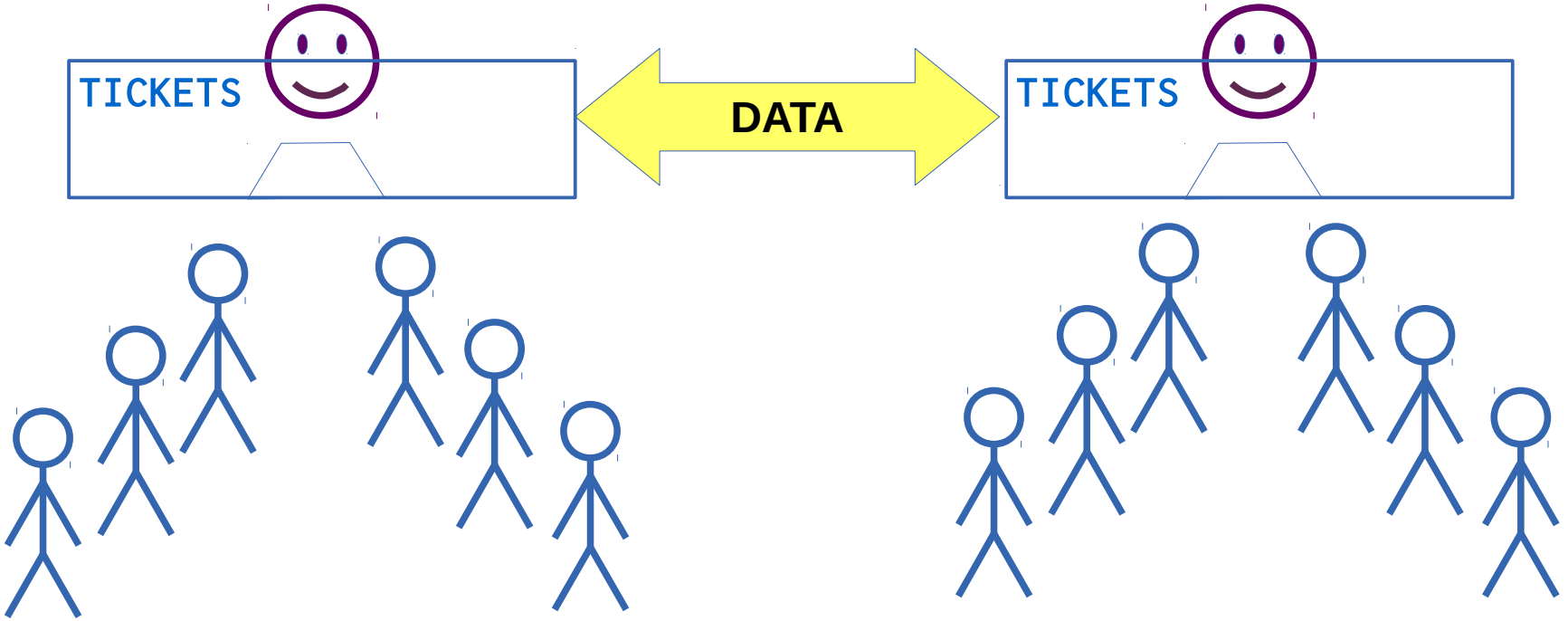
- Two separate train sections handled by two separate clerks
- ***Don't need to exchange fine-grained data***
- Some “edge” data shared periodically e.g; family in same coach

Multi Core Parallelism – Image filter



Data exchange is structured, not fine-grained (only edge)

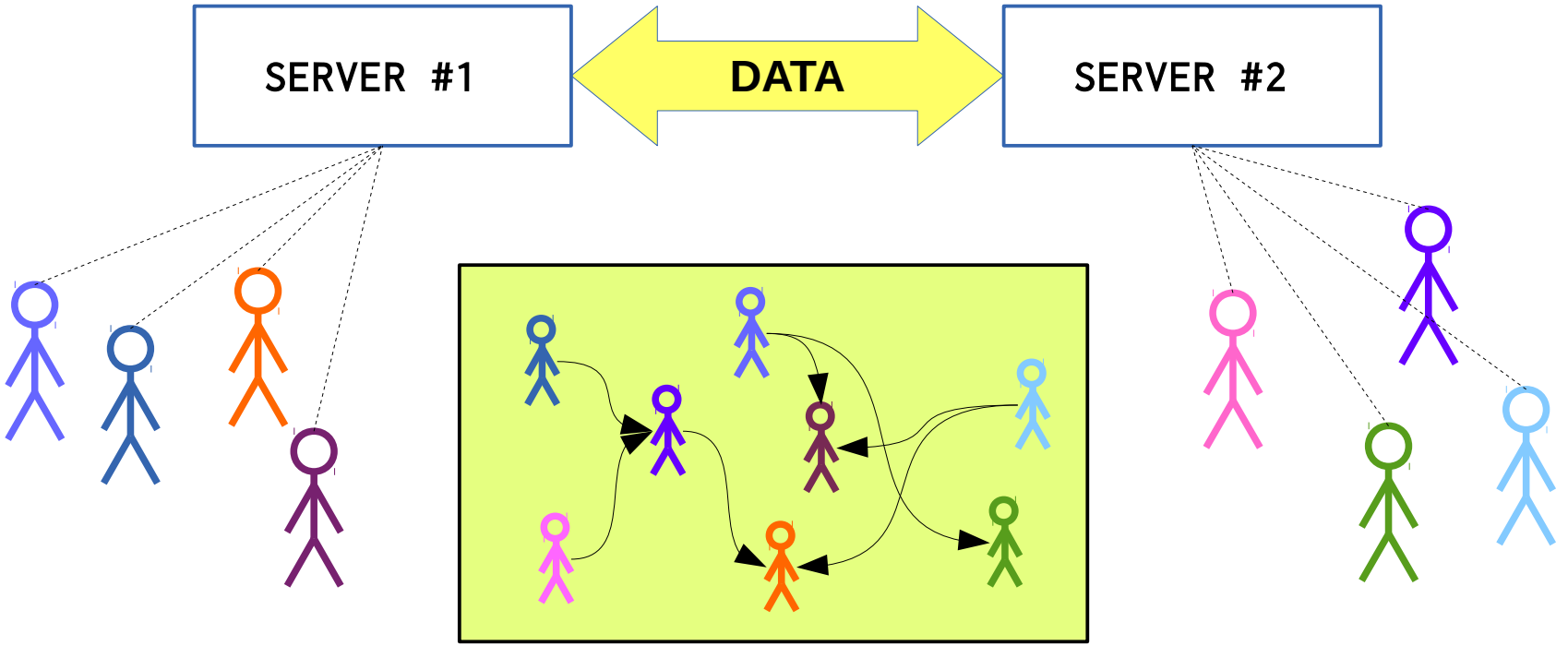
Multi Core Concurrency



- ***ALL*** tickets handled by two separate clerks
- ***NEED to exchange fine-grained data***
- Can't sell the same seat to two people!

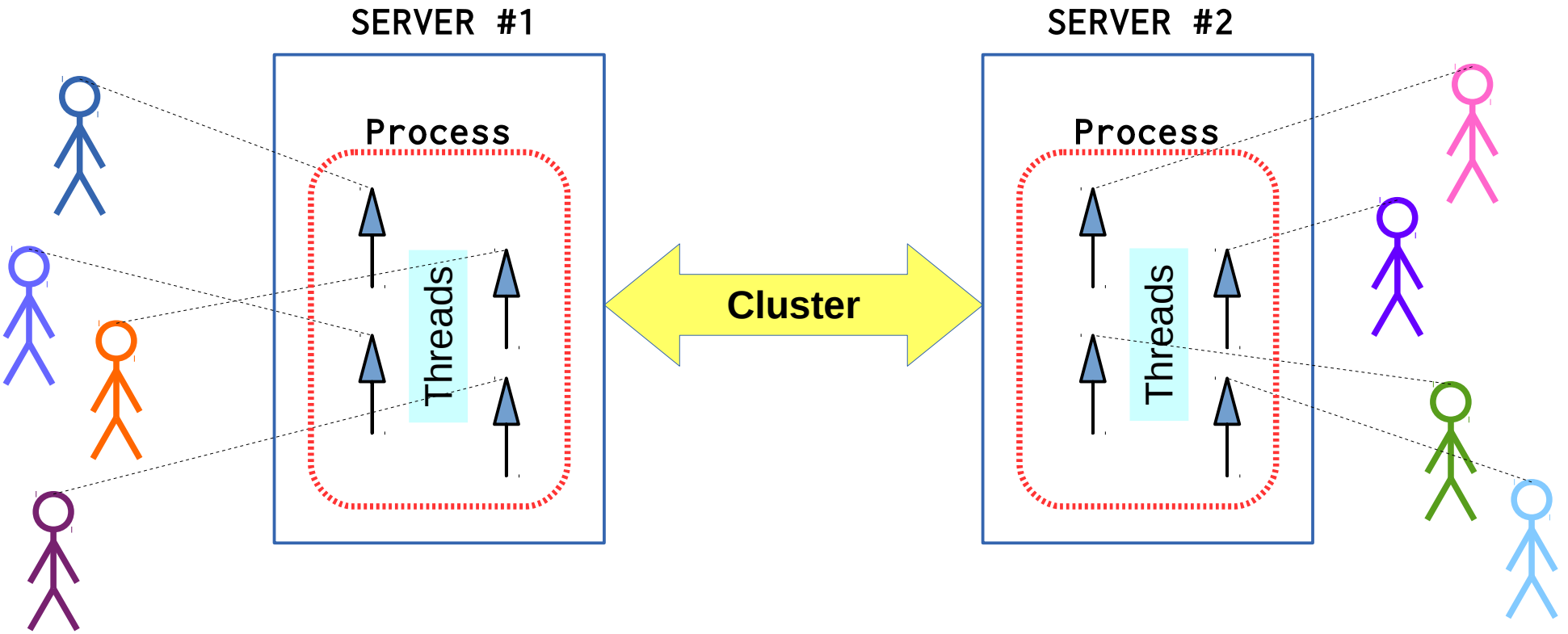
Multi Core Concurrency – MMORPG

http://www.keyfunda.com/blog/2014/9/23/semiconductor-scaling-and-concurrent-clouds-part-ii



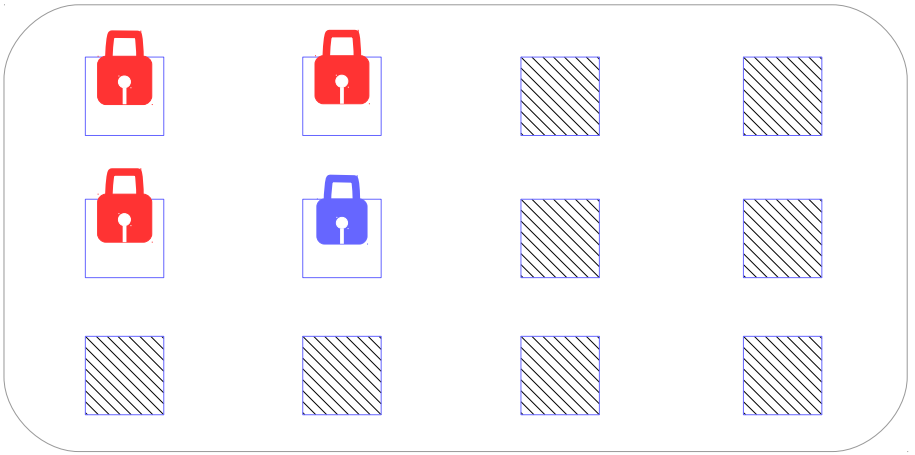
- MMORPG = Massively Multiplayer Online Role-Playing Game
- Scenes and Interactions need to be co-ordinated
- Fine-grained data exchange required

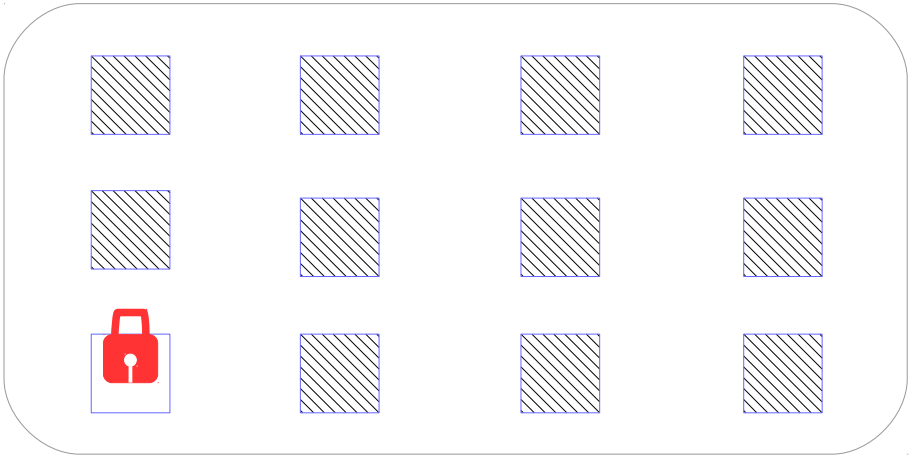
Thread-based multi core concurrency

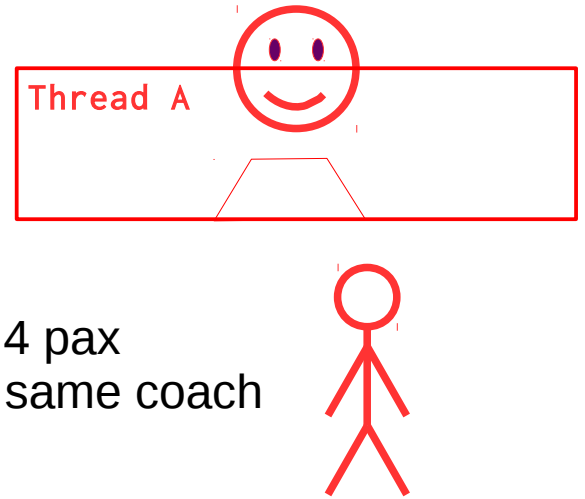


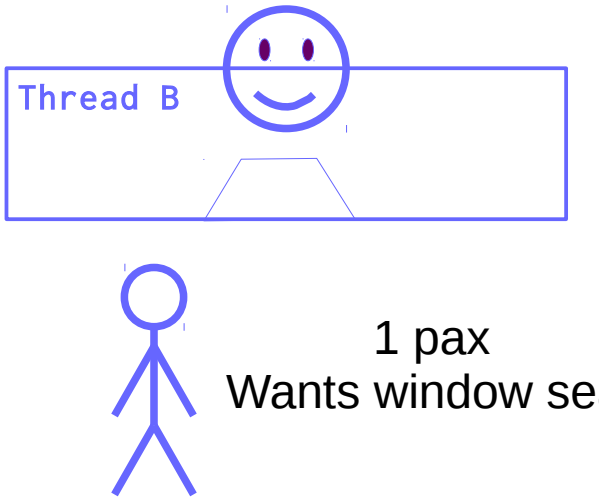
- Threads poll client sockets, frequent context switches
  - *Evented single-threaded server process:* ***nginx, Netty, node.js***
- Need to share memory between threads on different cores
  - *Lock synchronization can be done in C++, Java – but* ***TOUGH***
- Distributed multi-core also needs data exchange over cluster

“Simple” lock problem – Deadlock









4 pax
Prefer same coach

1 pax
Wants window seat

Deadlock must be broken by a SYNCHRONIZATION process

ONE DOES NOT SIMPLY WRITE CODE WITH LOCKS

Functional programming

- Functional programming **ideally** involves only pure functions

  - ***Pure function:*** *Same input gives same the same output, always*
  - *No side-effects, no mutable state*
  - ***Benefits:*** *No locks, Code correctness, Referential transparency*

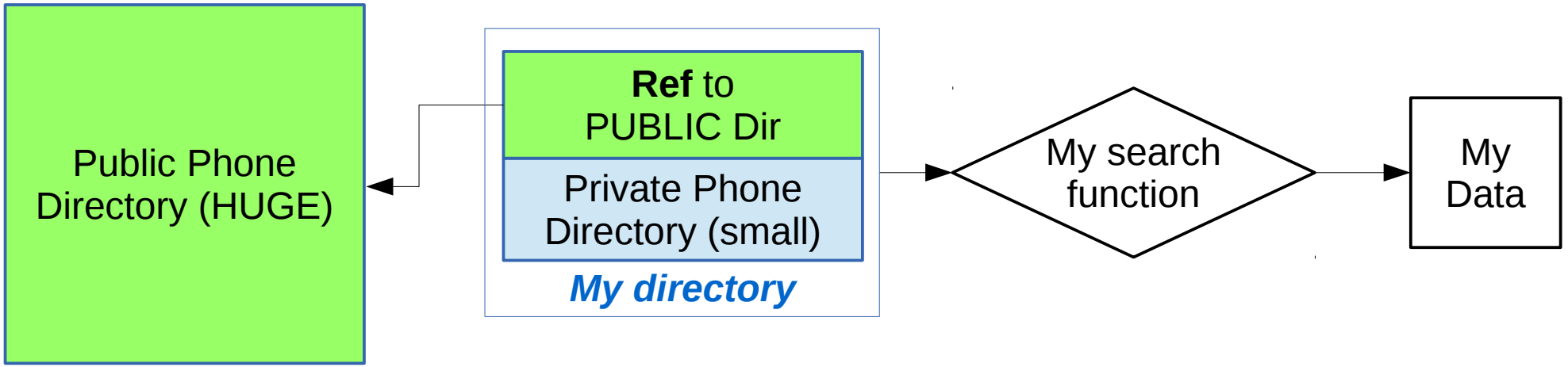
- But how can an ATM dispense cash without mutable state?

  - *FP languages keep "pure function" code separate from the "stateful" code*
  - *Manage shared state using **Transactions** (similar to a SQL database)*

- Erlang is mostly functional, with some stateful parts
- Functional part needs immutable values

Why immutable values?

> "... a value is something that doesn't change. 42 doesn't change. June 29th 2008 doesn't change. .... Even aggregates are values..."
>
> *http://clojure.org/state*

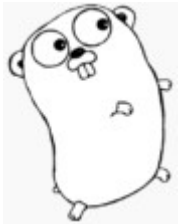


Mutating the Public Directory invalidates MyData.

If you want to make a change, make it on a **COPY.**

The original PublicDir **VALUE** should be **immutable.**

Concurrency oriented languages

Language	Runtime	Concurrency model
Clojure 	Java VM	• Refs (synchronous world) • Agents (asynchronous world) • Functional, with Software Transactional Memory (STM) • Distributed concurrency not built-in (by choice)
Go 	Native	• Lightweight goroutines • “Share memory by communicating” philosophy • Type-safe synchronized channels between channels • Distributed concurrency not built-in (libraries: gocircuit)
Erlang 	BEAM VM	• Lightweight processes managed by runtime • Processes communicate by message passing • Mix of functional and stateful programming • Distributed concurrency built in (easy to do)