# Mobile Interactive Group, UK

## Erlang runs TV votes on Facebook

Erlang Factory 2011, London

David Dawson, Head of Technology
Marcus Kern, CTO

Integrated mobile and digital communications



MIG

C A P dilemma

Cloudbusting

Consistency (eventually)

Summary

# Who we are

- The UK ’s most integrated mobile and digital communications business serving the entire mobile services value chain.
- Fastest growing privately owned technology business in the UK
  (Sunday Times, Tech Track No.1, 2008)
  (Deloitte Technology Fast 50, No.1, 2010)
- Billing and messaging, mobile operator platforms, mobile marketing, mobile advertising, experiential marketing, CRM,  digital creative, on device applications and more
- Use Open source Technology
- Balance of Cutting edge / proven Technology
- Happy Developers
- 180 Staff based in the UK, US, NL, AU

mobile interactive group





ERLANG

# Who we work with

- Over 350 customers through multiple sectors across MIG
- New technologies and products being developed all the time
- We support our customers internationally
- Totally integrated approach across the group companies

# The Challenge

- Paid for mass-participation vote via SMS. The L-word
- Payment vs. Interaction events
- Porting the experience to Facebook
- Use Facebook credits
- Expanding interaction through social features
- MIG is a





ERLANG

# The Requirements

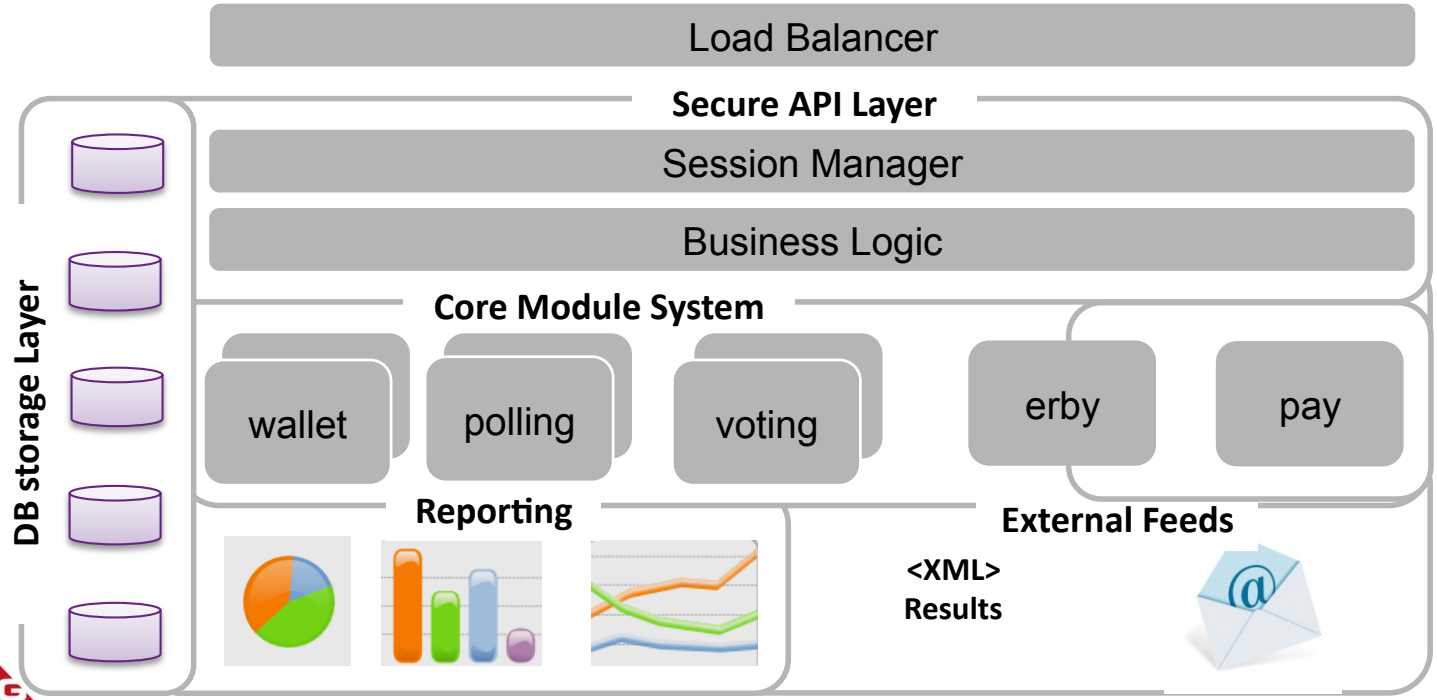
- Build an interactive & payment platform
- Voting / Polls / Payments
- A pluggable architecture
- 1000s of paid votes/s
- A globally scalable platform which
  - Has **consistent** storage
  - Is highly **available**
  - And can deal with network **partitioning**




ERLANG

# Architecture approach

- Frontend app interacts with our 4.5 layered stack:
  - Secure API
  - Session Manager
  - Easily configurable business logic
- erby - erlang to ruby bridge
- Pluggable modules
- DB Storage layer







ERLANG

MIG

C A P dilemma

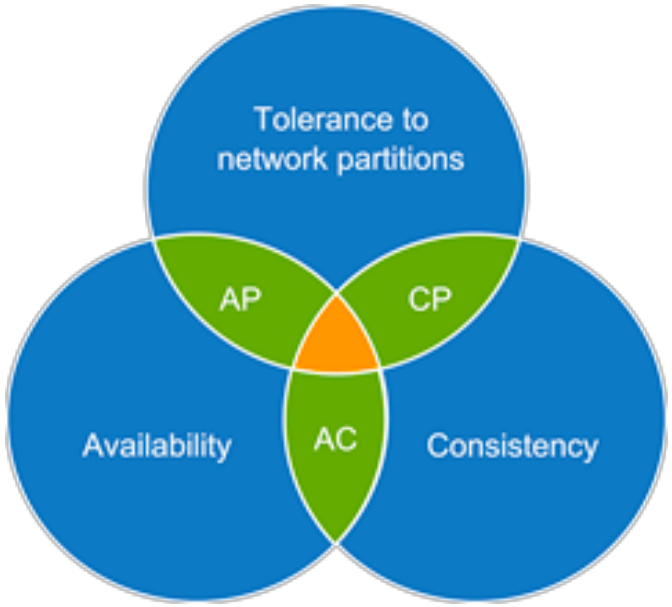
Cloudbusting

Consistency (eventually)

Summary

# Can you spot the problem?

- We want a DB that give us:
  - Consistency
  - Availability
  - Partition Tolerance
- But CAP theorem dictates we can only have 2 of the 3






# Three options - can't do all three at once

- So we had to opt for 2 of the 3
  - Availability
    - Not being able to vote is bad
  - Partition Tolerance
    - Network splits happen, fact of life!
- At the cost of Consistency
  - As long as we can be eventually consistent
  - We will solve this problem later if needs be



# Our choice

- Apart from satisfying the AP of CAP we wanted something that was:
  - Built in erlang
  - We already had used
  - Had a great community
  - And we could contribute back to
- Hence




# Distributed Database

- Other reasons we choose RIAK for
  - It has tuneable CAP settings
  - Eventual consistency is in milliseconds
  - It is stable as hell ( we run our gateway off it )
  - It scales - doubling our machines doubled our throughput
  - The code is clean and easy to follow
  - The Basho guys are smart and helpful



MIG

C A P dilemma

Cloudbusting

Consistency (eventually)

Summary

# Cloudbusting – The promise

- Rapid scaling
- Flexible costs for TV events
- Serve local
- Redundant storage
- Run both websv & DB



# Cloudbusting – Evaluation

- So we picked Amazon EC2
- Main focus: DB => we used Basho Bench for load testing
- Rent reasonable fuel: 5 x Cluster Compute Instances (each with 23GB memory, Dual Xeon X5570 quad-core 64-bit, 1690 GB of instance storage, 10GBit/s Ethernet)
- That's 1690 GB Ephemeral! (*e·phem·er·al = Lasting for a markedly brief time*)
- Use EBS



# Cloudbusting – Evaluation

Use EBS:

- "Highly available" - redundancy and a lower disk failure rate
- Portable - volume can be connected to any instance
- Backups - can easily create snapshots

**BUT:**

- Extremely variable performance – (seek times 0.5ms to 10ms+)
- We observed variation of between 4k to 16k RIAK ops/s
- Maximum throughput of 1Gbit/s
- ££,£££.££ !?!?



# Cloudbusting – Conclusion

- Our immediate conclusion is to
  - Buy and co-locate our own RIAK nodes
  - Streamline scaling process (if we need it)
  - Due to strong DB dependency, no gain to separate app nodes

- Keeping a close eye on EBS/alternatives

- Also investigating Joyent (promising for US services)

- *However, if the ROI of owning is less then 18 months, we'll stay with owning.*



MIG

C A P dilemma

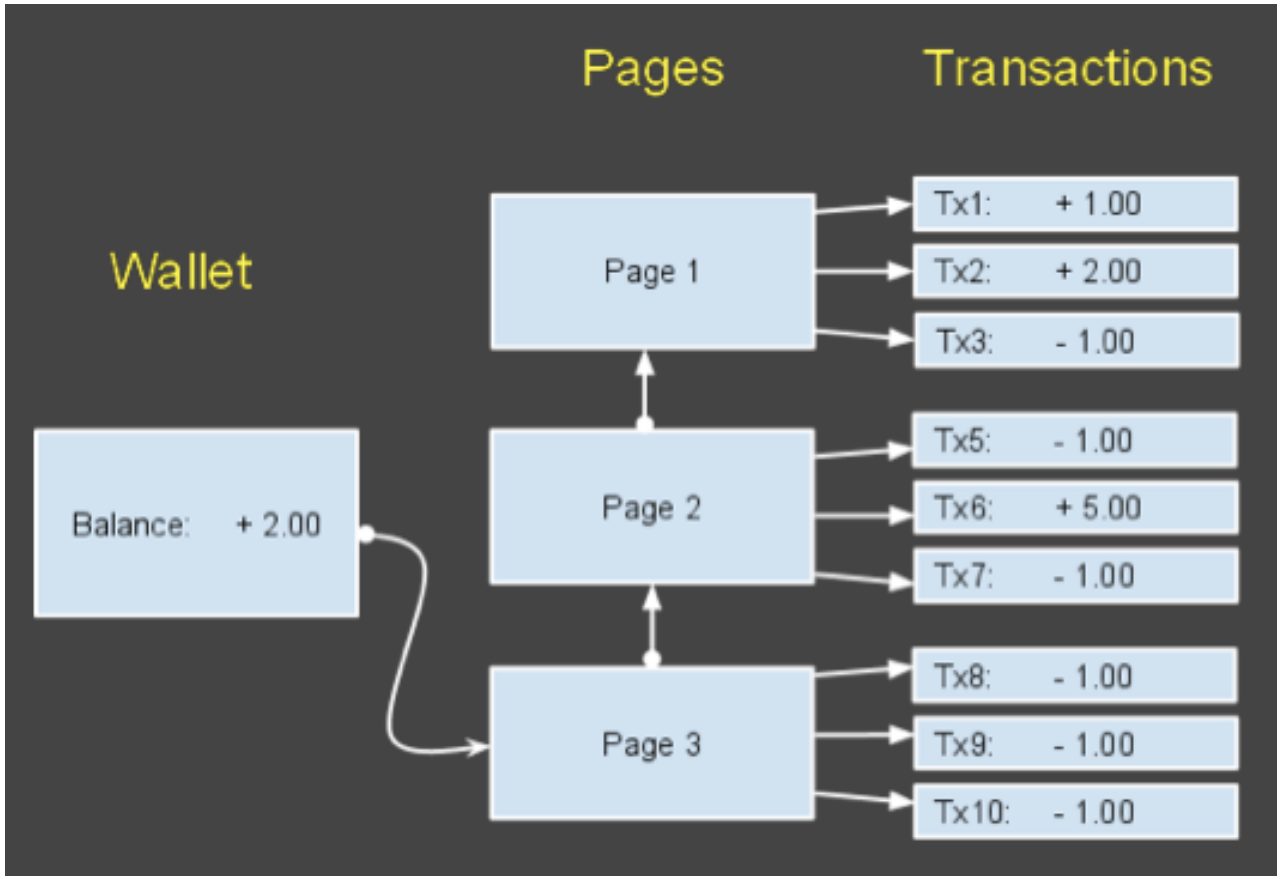
Cloudbusting

Consistency (eventually)

Summary

# Consistency (eventually)

- Attempt #1 with the payment engine

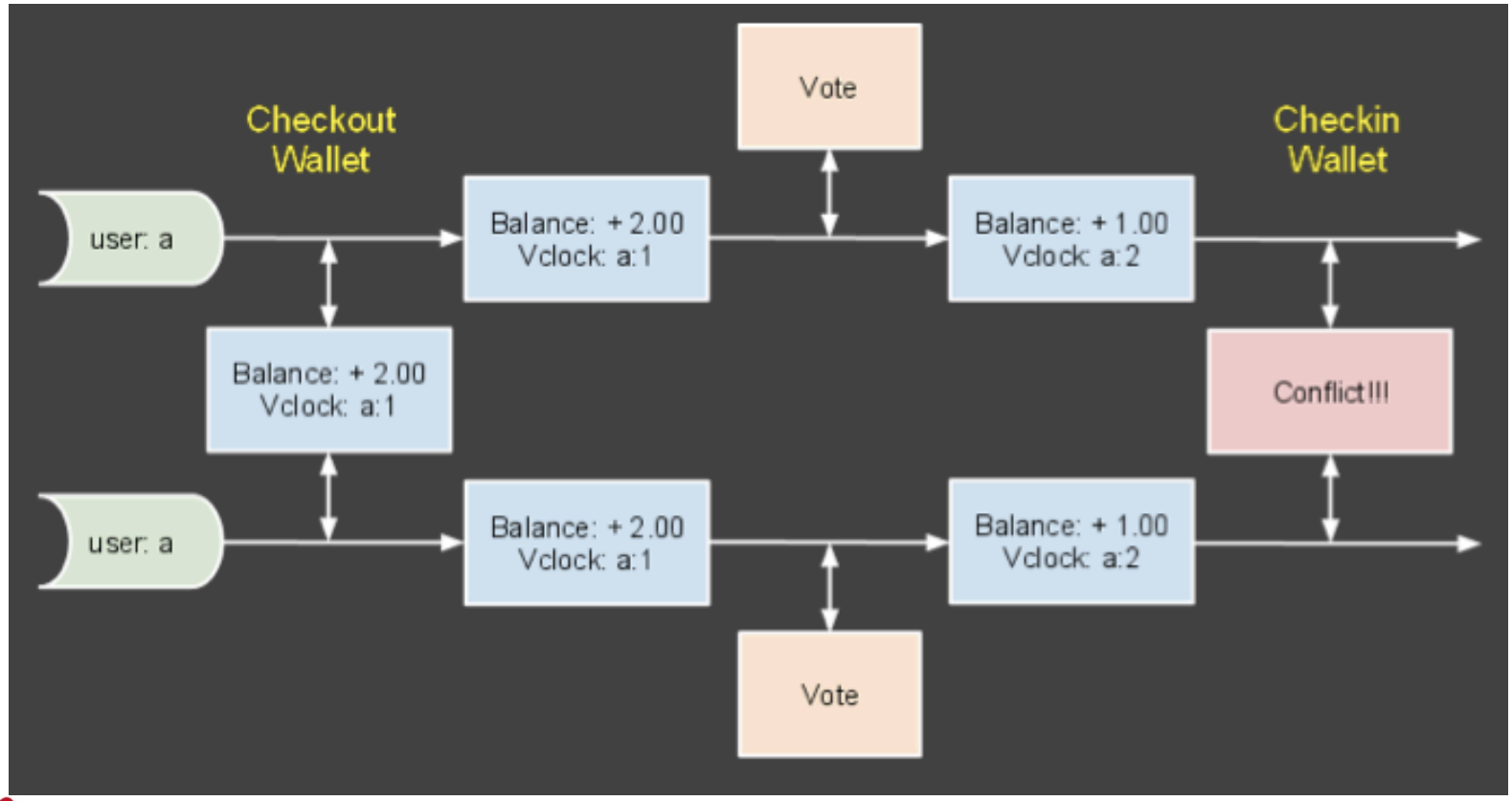







# Consistency (eventually)

Problem!!! - Two concurrent interactions for the same user (conflict!)



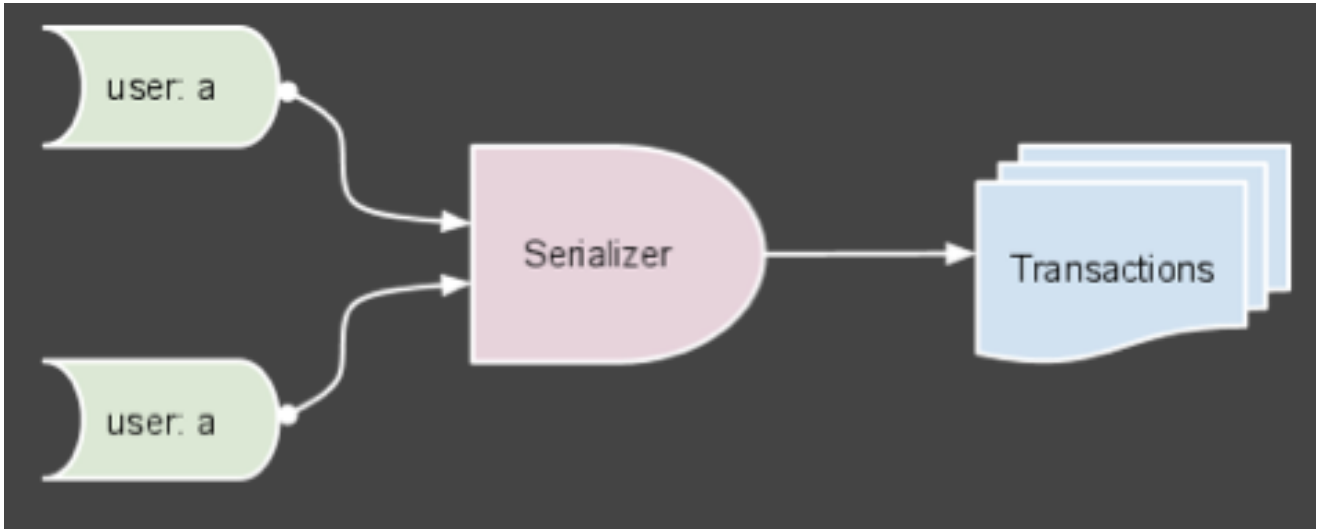





# Serialiser

Hence we came up with the idea of a serializer

- One Erlang process per unique user
- Serializes their interactions
- Risk of conflict massively reduced
- Version 1: global:register
- Version 2: riak_core








# Inconsistency - edge case

Although a small risk of conflicts exists:

- Adding / Removing Riak Nodes
- Network partitioning

We need to deal with these conflicts

- In a deterministic manner
- Borrow some ideas from Statebox
- Although very small risk of going overdrawn
  - e.g. Oyster card approach



# Consistency (eventually #2 )

Attempt #2 with the payment engine

- Merge transactions and wallet into 1 document
- Append to the document
- Move Current to Archived after show
- What about wallet size?
  - Comparable performance 1 - 1000 transactions
  - Acceptable performance 1000 - 5000 transactions

**Archived**

| TransactionID | Amount | Balance |
|---|---|---|
| Tx1: | + 1.00 | + 1.00 |
| Tx2: | + 2.00 | + 3.00 |
| Tx3: | - 1.00 | + 2.00 |

**Current**

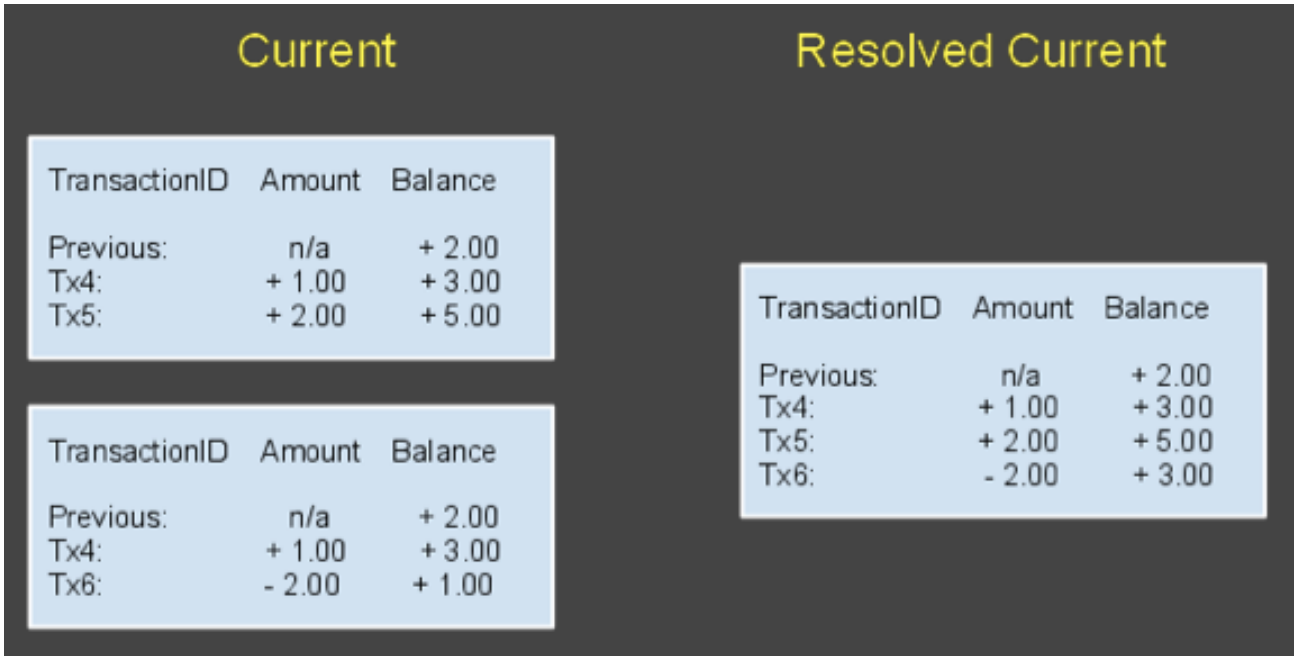
| TransactionID | Amount | Balance |
|---|---|---|
| Previous: | n/a | + 2.00 |
| Tx4: | + 1.00 | + 3.00 |
| Tx5: | + 2.00 | + 5.00 |



# Conflict Resolution

In the event of a conflict with multiple wallets

- Union the wallets ( TransactionID is unique )
- Calculate new balance
- Write Resolved wallet







ERLANG

MIG

C A P dilemma

Cloudbusting

Consistency (eventually)

Summary

# Summary

1. Stick with choice of *Availability/Partition Tolerance*
2. Continue to investigate Cloud options
3. Working closely with Basho

4. Minimize conflicts through code
5. Deal with them in a deterministic way

6. Following / sharing case studies



# Questions