Erlang Solutions Ltd.

Mnesia for the CAPper

Ulf Wiger
Erlang Solutions Ltd

Erlang Factory, London, 11 June 2010

The Mnesia DBMS

- A fast ACID in-memory DBMS
- Tightly integrated with the language
- Old, tried and true

ACID:

- Atomicity
- Consistency
- Isolation
- Durability

- Is it still competitive?
- Is it suitable for CAP-style database apps?

The CAP Theorem:

- “Consistency
- Atomicity
- Partition Tolerance

– pick any two.” (Eric Brewer)

Outline

- Cool things about Mnesia
- Some not-so-cool things
- Some recent developments

The convenient database

- Fully integrated into OTP
- Create schema and tables on the fly
- No language impedance mismatch
- Client and data in the same memory space
  - without sacrificing safety

```
1> mnesia:create_schema([node()]).
ok

2> mnesia:start().
ok

3> mnesia:create_table(t,
      [{disc_copies,[node()]}]).
{atomic,ok}

4> [mnesia:dirty_write(
      {t,N,N*1000}) ||
     N <- lists:seq(1,30)].
[ok,ok,ok,...]

5> Q = qlc:q([S ||
             {t,N,S} <- mnesia:table(t),
                        3 < N, N < 7]).
{qlc_handle...}

6> mnesia:transaction(fun() -> qlc:e(Q) end).
{atomic,[4000,5000,6000]}
```

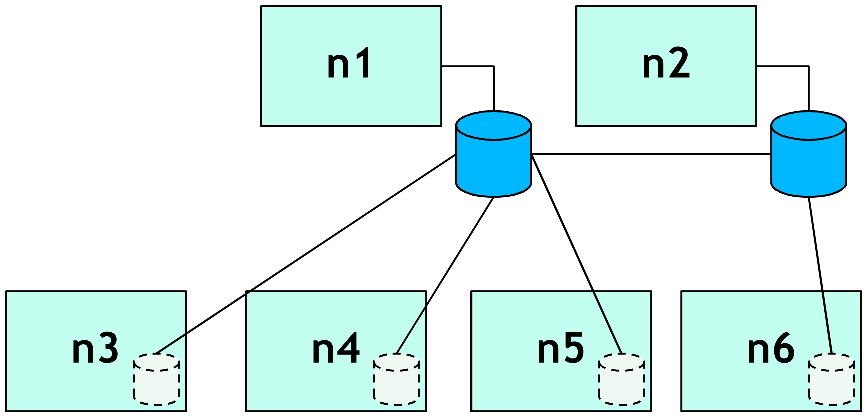
...and fast, too

- In benchmarks done at Ericsson a few years ago
  - Mnesia tied the best commercially available cluster DBMS (Clustra) for transaction throughput and scalability
  - Two in-house products were faster – one became MySQL Cluster (NDB)
  - Mnesia beat them all on response times
- Linear scalability up to at least 50 nodes
  - If the data model is ideal for fragmentation
- A "dirty read" in Mnesia takes ~5-50 µsec (for relatively small objects)
  - Not possible to match when crossing memory protection boundaries

Rugged

- The “D” in “ACID” stands for Durability
- Committed transactions can be rolled forward
  - Nodes may crash during two-phase commit
- Disk-based tables are repaired automatically

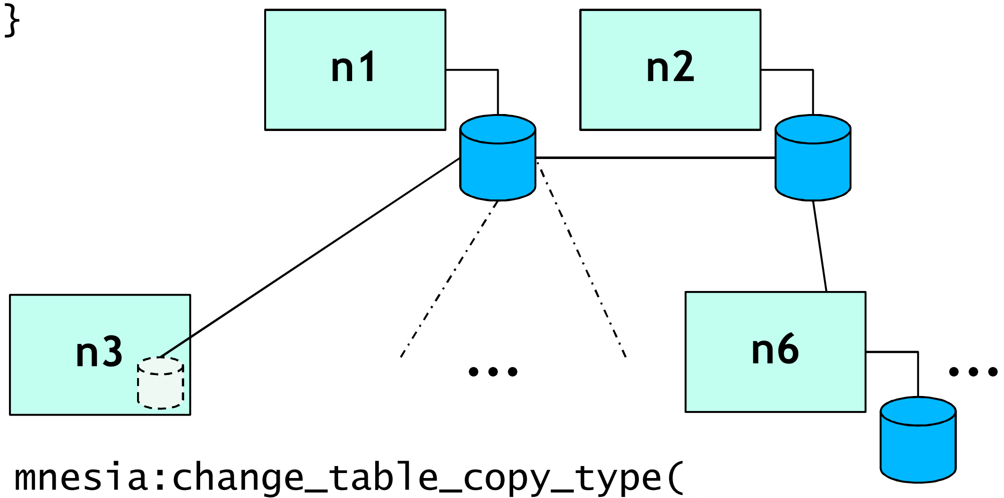
Rugged



- Diskless nodes can be added ad-hoc...

...

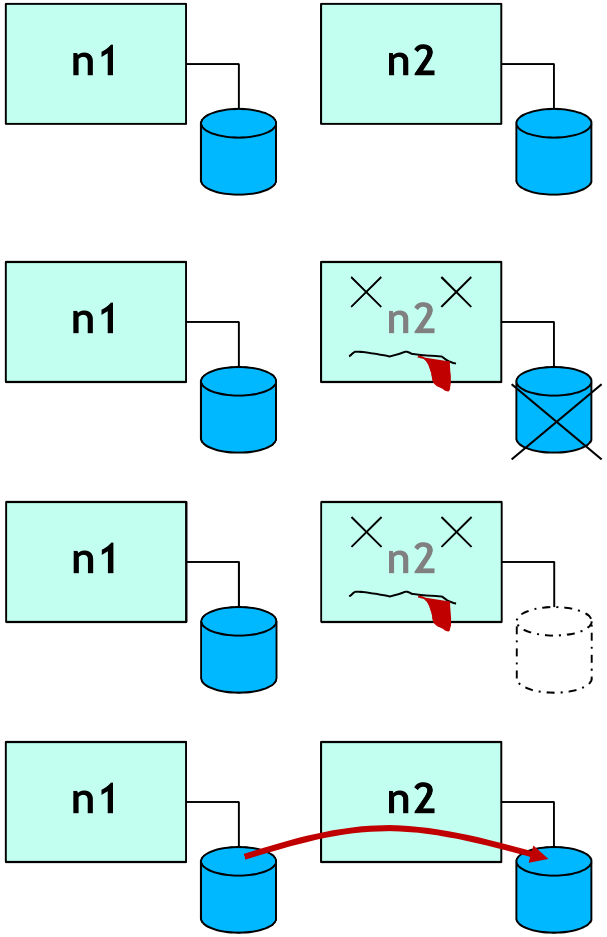
```
{extra_db_nodes, [n1@host1, n2@host2]}
```



- ...and easily converted to disk-based nodes

```
mnesia:change_table_copy_type(
    schema, n6@host6, disc_copies).
```

Rugged



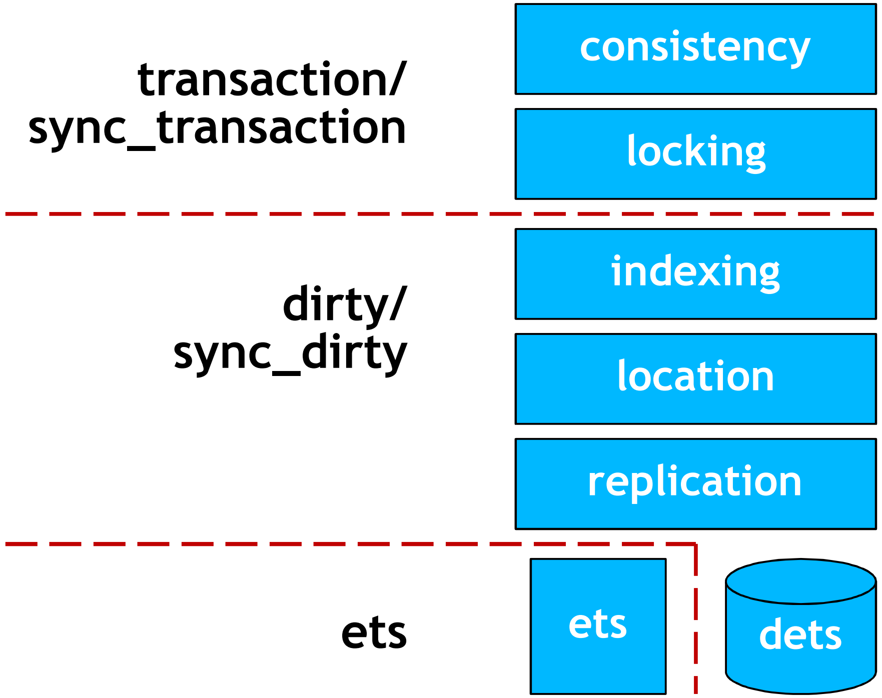
- If a disk copy becomes corrupt...

```
rm -r $MNESIA_DIR/*
```

- It can be automatically rebuilt from the cluster
  - Start with extra_db_nodes

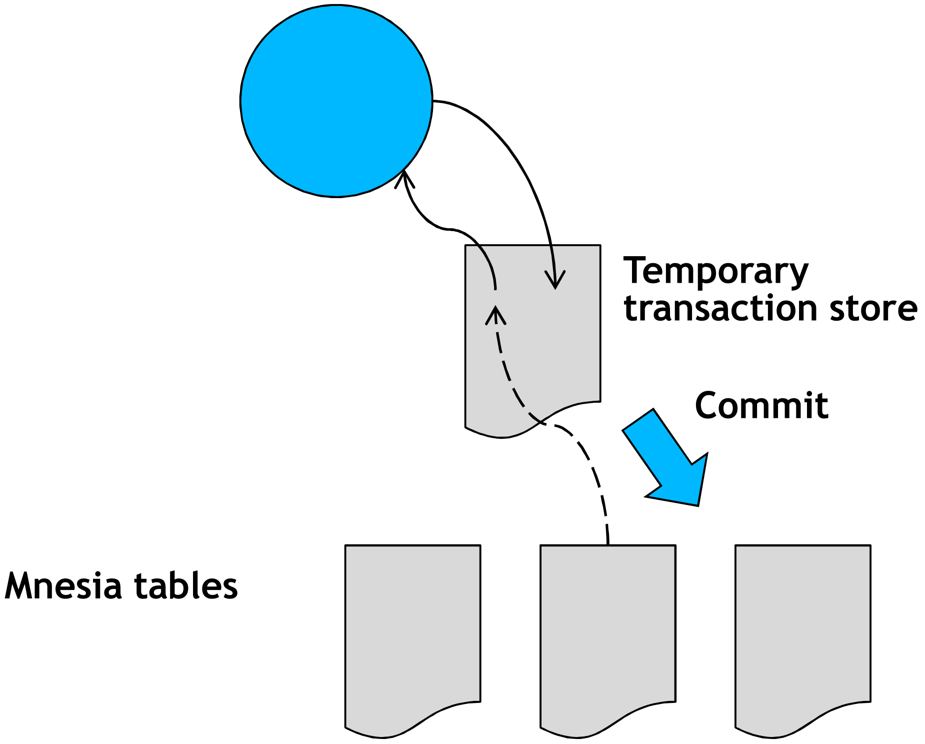
Naughty database...

- Subversion is optional



- Transaction commits with roll-forward
- Note: dirty writes give no consistency guarantees
  - best-effort replication
- Dirty deeds from within transactions can yield some nasty surprises

The Isolation Property

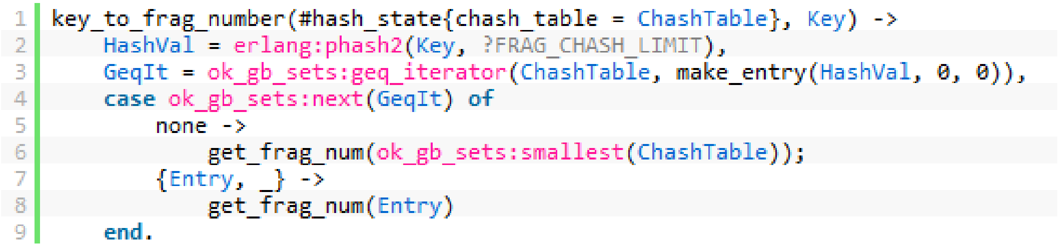


- Nested transactions:
  - A new transaction store is created
  - All data copied from store A to store B
  - On commit, all data is copied back
- Dirty reads know nothing of the transaction store

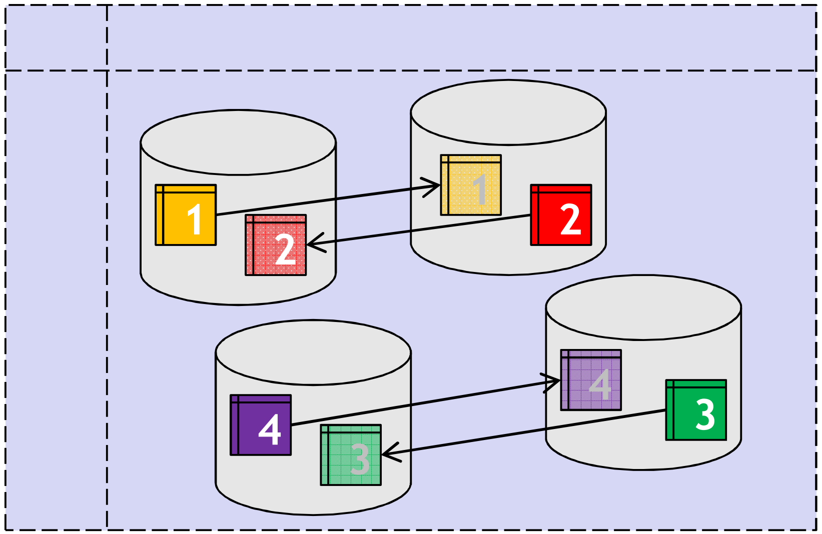
Fragmented Databases (sharding)

- (Almost) transparent to the user
- Semi-automatic or manual configuration
- Each fragment a first-class table
  - Can be indexed, replicated, etc.
- All fragments must be available at all times
  - Number of fragment replicas rather than R/W parameters

Custom key distribution



```
key_to_frag_number(#hash_state{chash_table = ChashTable}, Key) ->
    HashVal = erlang:phash2(Key, ?FRAG_CHASH_LIMIT),
    GeqIt = ok_gb_sets:geq_iterator(ChashTable, make_entry(HashVal, 0, 0)),
    case ok_gb_sets:next(GeqIt) of
        none ->
            get_frag_num(ok_gb_sets:smallest(ChashTable));
        {Entry, _} ->
            get_frag_num(Entry)
    end.
```



http://igorrs.blogspot.com/2009/11/consistent-hashing-for-mnesia-fragments.html

Extensible

- Activity callback modules
  - Extend or modify Mnesia's semantics
  - Per-transaction or as a global default
- Fragmented tables implemented as an activity callback
  - (but using some ugly hacks)

rdbms.erl:

```
write(Tid, Ts, Tab, Rec, Lock) ->
    VMod = ?vmod,
    validate_rec(Tab, Rec, VMod),
    do_write(Tid, Ts, Tab, Rec, Lock, VMod),
    check_references(Tab, Rec, write, VMod).

do_write(Tid, Ts, WTab, WRec, Lock, VMod) ->
    AMod = module(WTab, VMod),
    AMod:write(Tid, Ts, WTab, WRec, Lock),
    rdbms_index:update_index(
        Tid Ts, WTab, write,
        WRec, LockKind, VMod).
```

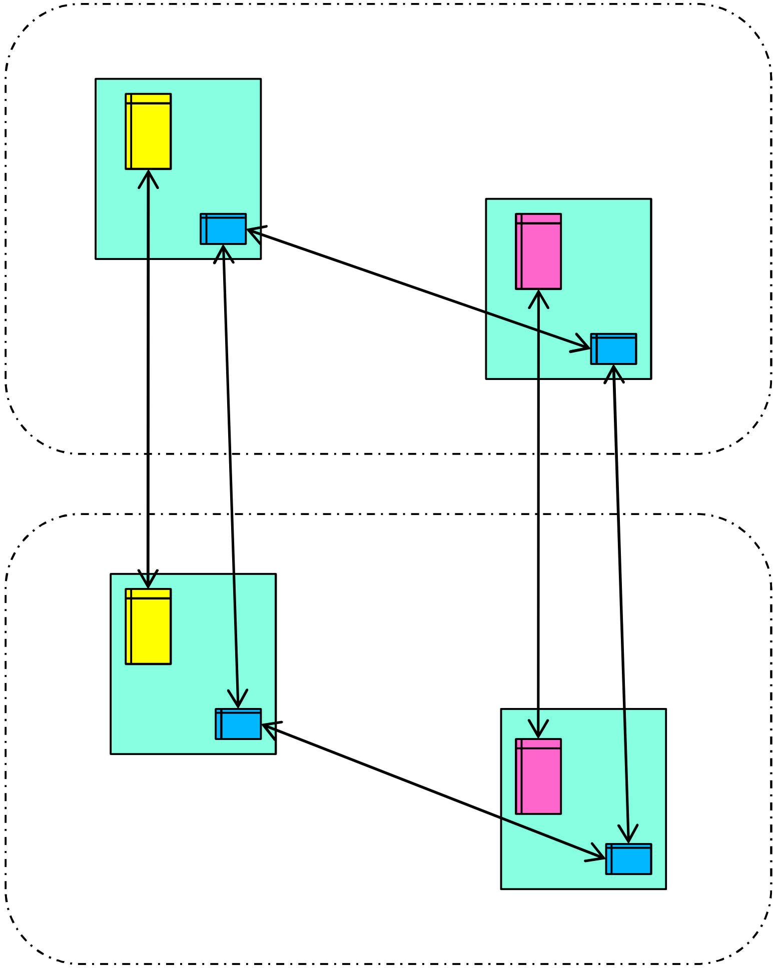
```
1> mnesia:activity(
      transaction,
      fun() ->
         [Old#person{age = Age}] =
             mnesia:read({person, Id}),
         Older = Old#person{age = Age+1},
         mnesia:write(Older)
      end, rdbms).
ok
```

Some nifty bits

- Asymmetric locking (“sticky locks”)
  - If all locking is done from one node, no network activity needed
- Incremental backup
  - Supported, but practically undocumented
- Automatic SNMP hooks
  - Declare a MIB as a mnesia table, instrumentation for free
- Automatic repair + checkpointing
  - Presumably brings up a consistent database each time
- ‘Install fallback’ for automatic recovery from backup
  - Used during in-service upgrade

Geographic redundancy?

- Not really
- However, mnesia is tolerant to network delays
- Replicas can be distributed explicitly
  - Possibly across data centers
- Schema must be fully replicated



Not-so-hot stuff

- Disk-only copies limited to 2 GB/table instance
  - Silently fails if you exceed the limit
- No concurrent versions of the schema
  - Redundancy upgrade becomes extremely difficult
- Deadlock prevention scales poorly to massive concurrency
  - ...but possibly better than other known techniques
- Imperative data definition API
- Partitioned network handling (more later)
- Overload handling (more later)

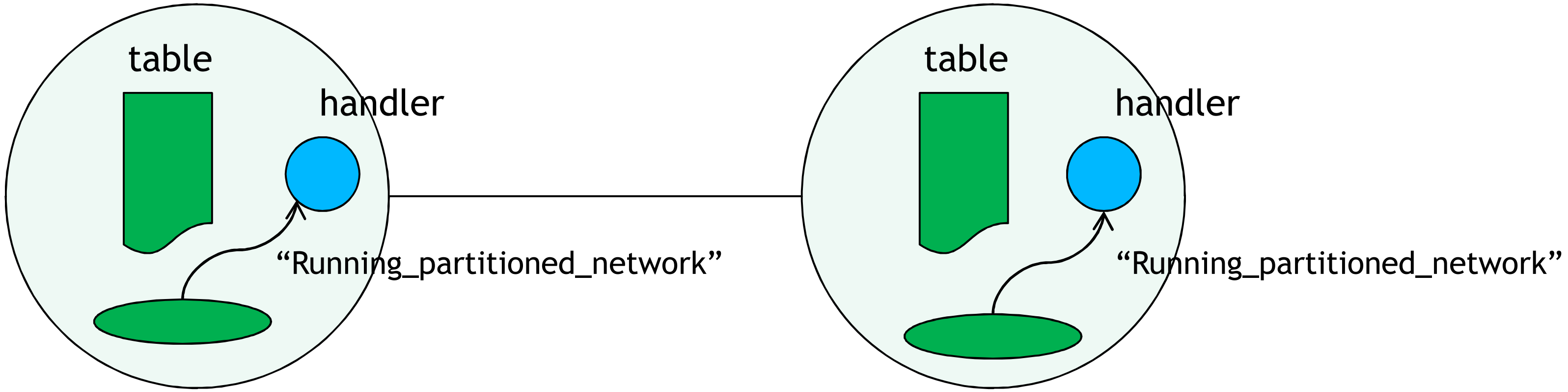
Split-brain (partitioned network)

- Network failure is indistinguishable from normal node-downs
- When nodes are reconnected, database can be inconsistent
- Pathological problem in general
- Mnesia detects the condition
  - Issues a “running partitioned network” event
  - Refuses to merge the tables

Split-brain (partitioned network)

- Only remedy offered:
  - Call mnesia:set_master_nodes([N]) on one side
  - Restart other side; unconditionally load data from N
  - Data loss is very likely
- You have to write code to manage it!
- Smooth recovery has not been possible

The ‘unsplit’ Application



- Install an event handler on each node (automatic)
- When triggered, grab a global lock (global:trans/2)
  - The one who wins, resolves the conflict
- Merge the schema, lock tables, and merge in one operation
  - Requires a mnesia patch (or OTP R14B, released 16 June)

How to merge

```
mnesia:create_table(test,[{ram_copies,[n1@debian,n2@debian]},
                          {attributes,record_info(fields, test)},
                          {user_properties,
                           [{unsplit_method,{unsplit_lib,vclock,
                                             [#test.vclock]}}]}
                         ]).
```

Position of
vclock attr

- Vector clock implementation borrowed from Riak
- Other methods possible
  - Predefined methods: last_modified, bag, ...
- The unsplit_reporter module can be used to report inconsistencies
  - Sends "summary alarm" to alarm_handler in SASL
  - Collects conflicting records in a temp table for inspection

Automatic updating of Vector Clocks

- mnesia:activity(transaction, Fun, my_mnesia_cb)
- Make a hook function for write(Tid,Ts,Tab,Rec,LockKind)
- Suggestion: exprecs for generic record attribute access:

```
-module(my_mnesia_cb).
...
-include(“table_defs.hrl”).
-include(“exprecs.hrl”).
-export_records([....]).

write(Tid, Ts, Tab, Rec, LockKind) ->
   Rec1 = try Old = ‘#get-’(Rec, [vclock]),
              ‘#set-’(Rec, [{vclock, unsplit_vclock:increment(node(), Old)}])
          catch
             error:badarg ->
               Rec
          end,
   mnesia:write(Tid,Ts,Tab,Rec1,LockKind).
```

Making use of the ‘exprecs’ utility
http://github.com/esl/parse_trans

Dealing with conflicts

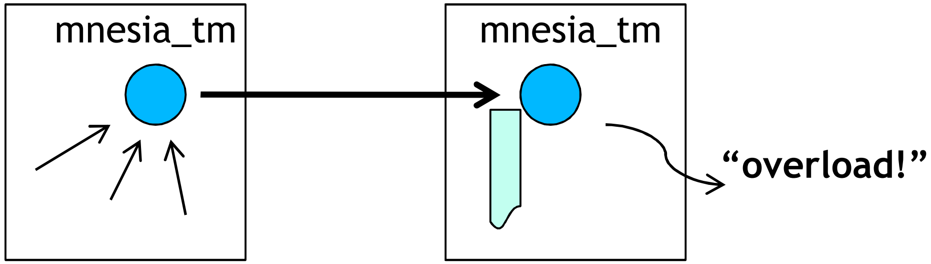
- Riak keeps a set of values for each key
  - Normally only one value
  - Multiple values if automatic conflict resolution impossible
- Mnesia could too
  - #record{key = K, values = [V]}
- This is not how people usually design their data model
- Does not work together with mnesia's indexing...

Work in progress

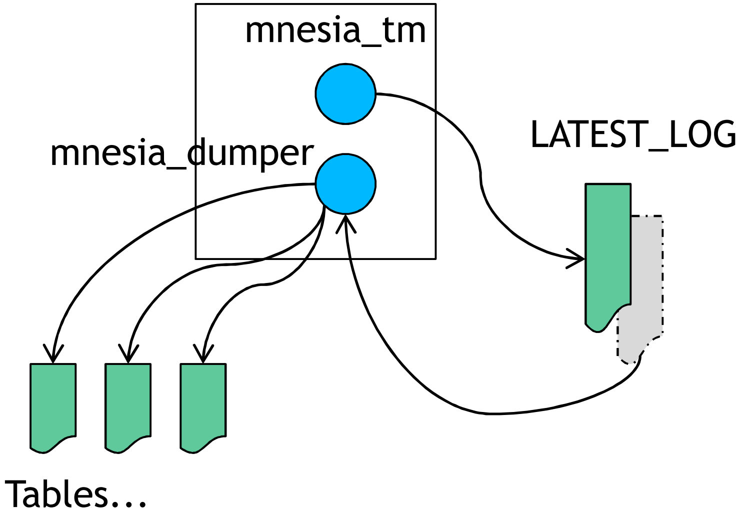
- http://github.com/esl/unsplit
  http://github.com/esl/parse_trans (for exprecs)
  http://github.com/uwiger/otp/tree/schema_merge
  (the mnesia patch)
- Possibly vie for inclusion into OTP
- NOTE! Problem is still very hard
  - You need to plan your data model
  - Prepare for inconsistencies
- Split happens – this might at least give you a chance to cope

Overload

- Swamping the message queue of a remote mnesia_tm
  - Mnesia sends an event if it happens
  - Very difficult to be pro-active



- Overlapping transaction log dump intervals
  - Mnesia sends an event...



- It does *not* tell you when it's no longer overloaded!

Slight remedy

- A new (undocumented) API to sample overload
- Intended to be called from a load regulation component
- Will be part of OTP R14B
- http://github.com/uwiger/otp/tree/mnesia_overload (patch on R13B)

```
mnesia_lib:overload_read() -> [{Type,boolean()]
   Type = mnesia_tm | mnesia_dump_log
```

Summary

- Mnesia has a few miles in it yet
- Biggest wart: lack of a scalable disk back-end
  - Fixable problem
  - Has been done a few times already
  - Replica sync logic might need to be revisited
- Medium wart: Dirty write unsafe on replicated data
  - Not as easily fixed as one might think
  - ‘dirty_transaction’? Like a transaction but without locks...
- True geographic redundancy would be nice