FEUERLABS

KVDB

A Database System for Configuration Data and Connected Devices

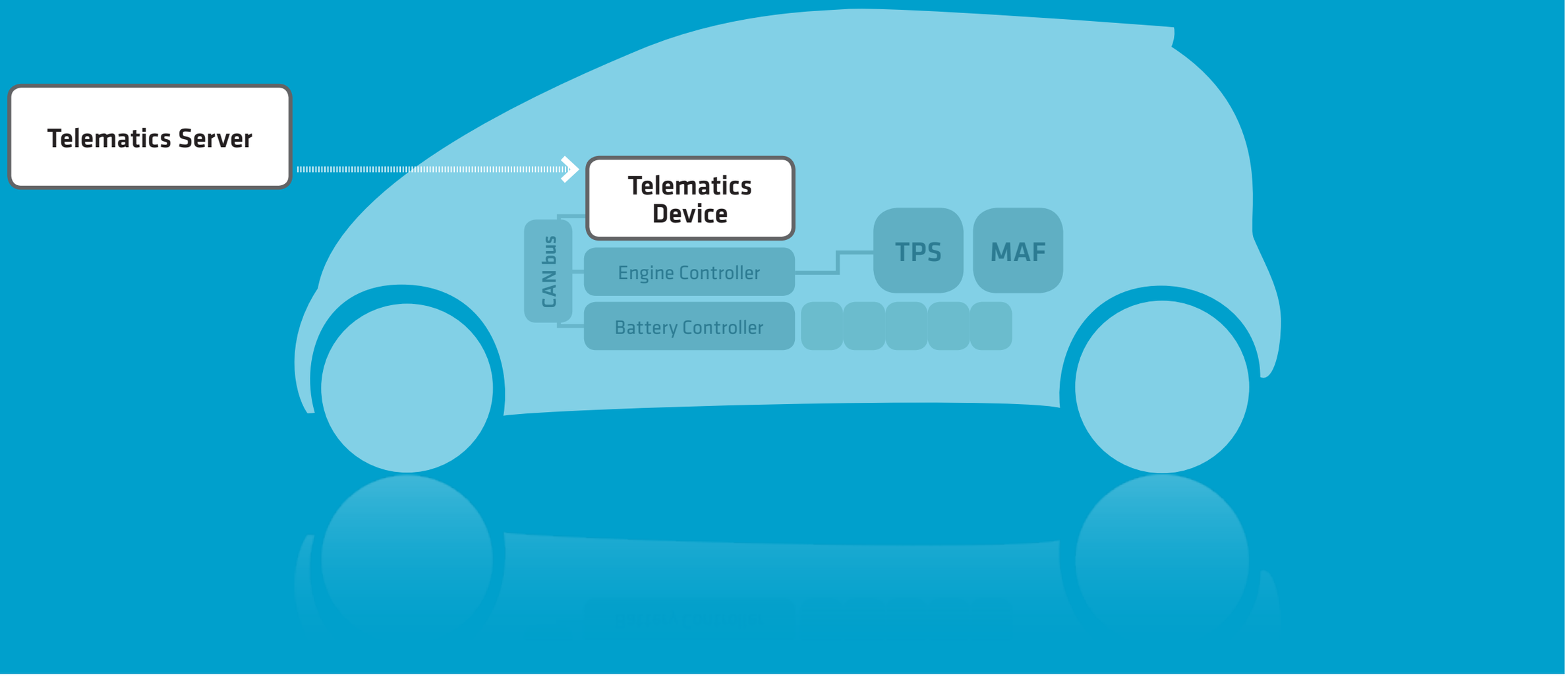
by Ulf Wiger, Co-Founder, Feuerlabs

What do we do?

- **Feuerlabs** provides device management servers, data communication, and embedded development frameworks to implement industrial-level M2M solutions
- Embedded stack, Exosense Device
- Server side, Exosense Server
- Communication - Provide roaming data plans

FEUERLABS

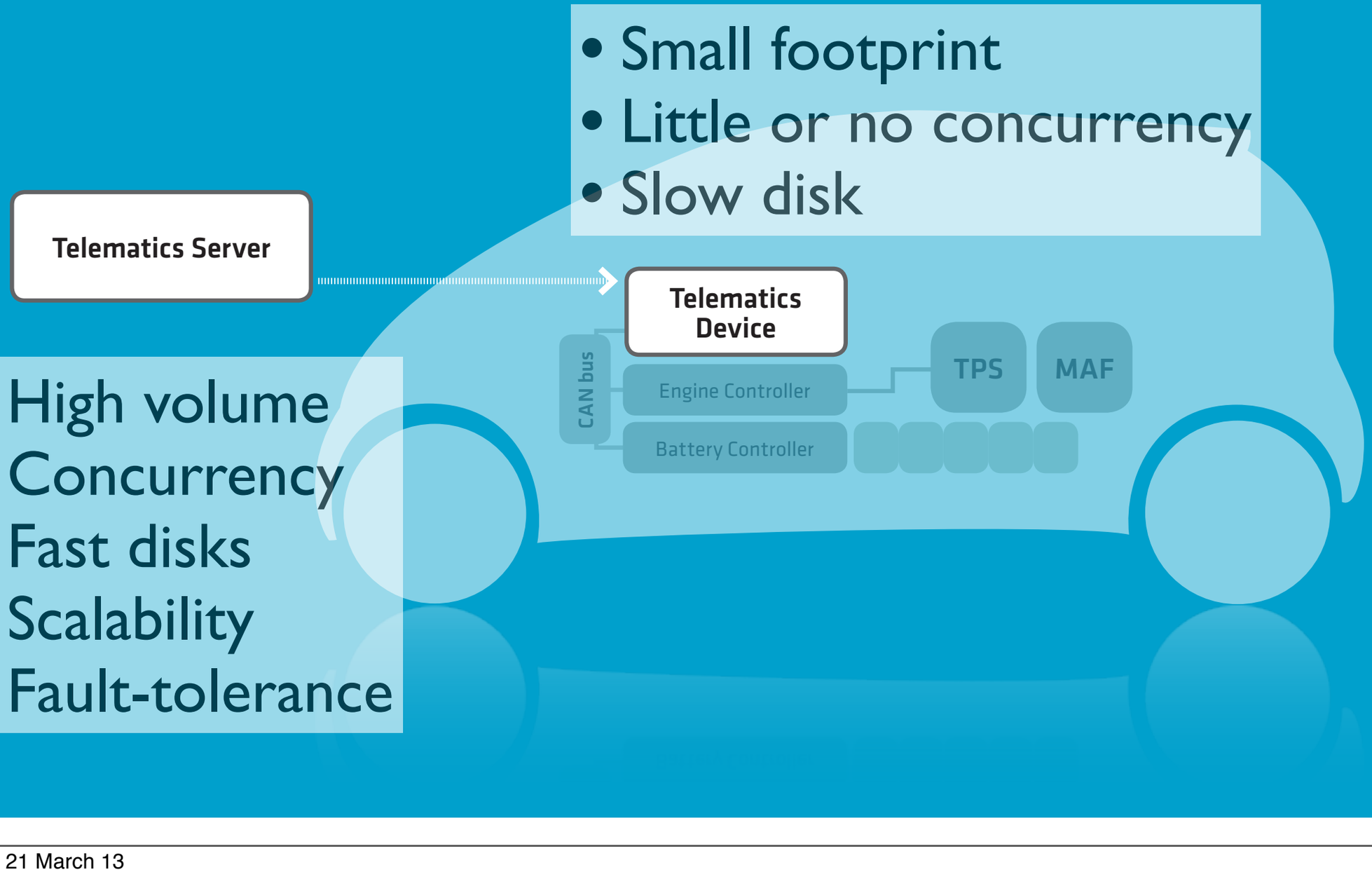
Feuerlabs' domain



FEUERLABS

Database Requirements

- Small footprint
- Little or no concurrency
- Slow disk



- High volume
- Concurrency
- Fast disks
- Scalability
- Fault-tolerance

FEUERLABS

KVDB Features

1. Ordered-set semantics
2. Choice of backends: Leveldb, SQLite3, ETS
3. Transactions
4. kvdb_conf – configuration data API
5. Persistent queues
6. CRON-like persistent timers
7. Extensible indexing
8. Schema callback module

FEUERLABS

Ordered-Set Semantics

- Hard-wired assumption
- Supported by all backends
- Sext encoding or ‘raw’ binaries
- {Key, Value} or {Key, Attrs, Value} representation

```
kvdb:add_table(table1,
               [{encoding, {sext, raw}}]),
kvdb:add_table(table2,
               [{encoding, {raw, term, term}}])
```

Choice of backends

- Leveldb – best choice for server-side
  - Fast, robust, creates lots of files
- SQLite3 – not used by Feuerlabs at the moment
  - Slooow (uses SQL interface)
- ETS – used for device-side (with transaction log) and as transaction store

FEUERLABS

Transactions

- Work similarly to Mnesia transactions
- Rollback semantics
- Transaction store is a KVDB ETS backend
  - Can be delegated to another process

```
kvdb:in_transaction(
    Db,
    fun(Db1) ->
            {ok, {_, As, Amt}} = kvdb:get(Db1, K={Name, salary}),
            kvdb:put(Db1, {K, As, Amt * 1.5})
    end).
```

FEUERLABS

Passing a Transaction Context

```
login(Aid, User, Db) ->
    case is_active(User) of
        false ->
            gen_server:call(
               ?MODULE, {make_user_active, Aid, User, Db});
        true ->
            ...
    end.

handle_call({make_user_active, A, U, Db}, _, St) ->
    case ets:lookup(?TAB, U) of
        [] ->
            kvdb:in_transaction(
              Db, fun(_) ->
                      ...
                  end);
        ...
    end;
```

kvdb_conf

- Structured (binary) keys: <<“person*Joe*age”>>
- API for
  - traversing and skipping subtrees
  - reading and writing subtrees
  - splitting and joining key parts
  - ...

FEUERLABS

kvdb_conf Example

```
Eshell V5.9.2  (abort with ^G)
1> rr(code:which(kvdb_conf)), kvdb:start().
ok
2> kvdb_conf:open(nofile, [{backend,ets}]).
{ok,<0.46.0>}
3> [kvdb_conf:write({K,[],V}) ||
    {K,V} <- [{<<"a*1*x[1]">>, <<1>>}, {<<"a*1*x[2]">>,<<2>>}]].
[ok,ok]
4> kvdb_conf:read(<<"a*1*x[1]">>).
{ok,{<<"a*1*x[00000001]">>,[],<<1>>}}
5> kvdb_conf:read_tree(<<"a">>).
#conf_tree{root = <<"a">>,
           tree = [{<<"1">>,[{<<"x">>,[{1,[],<<1>>},
                                       {2,[],<<2>>}]}]}]}
6> kvdb_conf:shift_root(down, v(5)).
#conf_tree{root = <<"a*1">>,
           tree = [{<<"x">>,[{1,[],<<1>>},{2,[],<<2>>}]}]}
7> kvdb_conf:write_tree(<<"b">>, v(5)).
ok
8> kvdb_conf:read(<<"b*1*x[1]">>).
{ok,{<<"b*1*x[00000001]">>,[],<<1>>}}
```

Persistent Queues

- FIFO, LIFO, ‘keyed’ FIFO/LIFO

```
3> kvdb:add_table(db, qtab, [{type, fifo}]).
ok
4> kvdb:push(db, qtab, q1, {a, 1}), kvdb:push(db, qtab, q1, {b, 2}).
{ok,{q_key,q1,105669048607563,b}}
6> kvdb:pop(db, qtab, q1).
{ok,{b,2}}
7> kvdb:pop(db, qtab, q1).
{ok,{a,1}}

8> kvdb:add_table(db, prio, [{type, {keyed,fifo}}]).
9> kvdb:push(db, prio, q1, {a, 1}), kvdb:push(db, prio, q1, {b, 2}).
10> kvdb:pop(db, prio, q1).
{ok,{a,1}}
11> kvdb:pop(db, prio, q1).
{ok,{b,2}}
```

CRON-like Timers

- Based on ‘keyed’ FIFO queues

```
12> kvdb_cron:create_crontab(db, timers).
ok
13> kvdb_cron:add(
        db,timers,"{in 3 secs; 3 times}",[],kvdb_cron,testf,[]).
{ok,{q_key,<<>>,105729091563262,105729094536167}}
14>
CRON!! {{{2013,3,19},{21,38,14}},658320}

CRON!! {{{2013,3,19},{21,38,17}},655700}

CRON!! {{{2013,3,19},{21,38,20}},642523}
```

Timer Examples

- “{ TimeSpec; Repeat; Until }”
- “{ in 2 hrs, 3 mins, 2 secs; repeat; 18:45:00 }”
- ”{ at 7:00:00; daily; 2014-12-31 }” % inclusive
- “{ in 1000 msec; 100 times }”
- “5000” % milliseconds, one-shot timer

FEUERLABS

Extensible Indexing

- Index on attribute values,
  or whole object (using callback function)
- {index, [name, ← on attribute value
  {skill, each, skills}, ← each value in list
  {tag, words, tags}]} ← each word in string

Indexing Example

```
5> kvdb:put(db,t1,{ulf, [{a,17},{skills,[erlang,opera]},
                         {tags,<<"swede,tall,handsome">>}], wiger}).
ok
6> kvdb:index_get(db,t1,a,17).
[{ulf,[{a,17},
       {skills,[erlang,opera]},
       {tags,"swede,tall,handsome"}],
      wiger}]
7> kvdb:index_get(db,t1,skill,erlang).
[{ulf,[{a,17},
       {skills,[erlang,opera]},
       {tags,"swede,tall,handsome"}],
      wiger}]
9> kvdb:index_get(db,t1,tag,<<"handsome">>).
[{ulf,[{a,17},
       {skills,[erlang,opera]},
       {tags,"swede,tall,handsome"}],
      wiger}]
```

Schema Callback Module

```
%% kvdb_schema callbacks
-export([validate/3,
         on_update/4,
         pre_commit/2,
         post_commit/2]).

Ex kvdb_schema_events.erl:

%% Send gproc_ps event if queue transitions from/to empty
%%
on_update({q_op,_,Q,true}, DB, Table, _) ->
    notify_queue_status(DB, Table, Q, empty),
    ok;
on_update({q_op,_,Q,false}, DB, Table, _) ->
    notify_queue_status(DB, Table, Q, not_empty),
    ok;
on_update(_, _, _, _) ->
    ok.
```

FEUERLABS

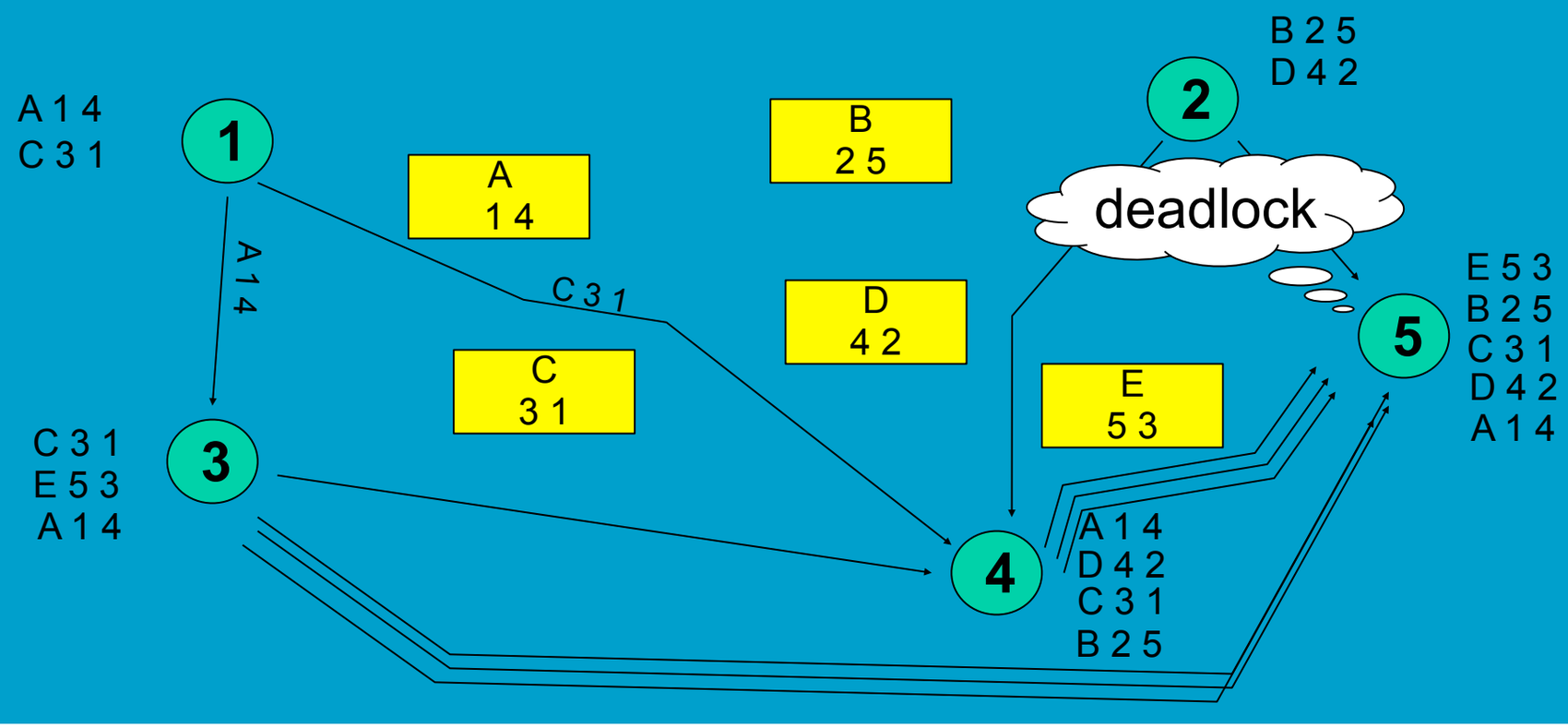
TODO

- Hehe… lots
- Major: Integrate locking (in progress)
- Log-based table merge support
- Distribution patterns

FEUERLABS

Scalable Transactions

- Message-based deadlock detection algorithm
- Model-checked by Thomas Arts (long time ago)
- Cooperative – no central dependency graph



FEUERLABS

Locker Library: ddd

- Standalone locking and deadlock detection
- Hierarchical read/write locks – [L1, L2, ...]
- Built-in quorum support
- Work in progress

Check it out...

- http://github.com/Feuerlabs/kvdb.git
- MPL 2.0 License