Star Trek: revisited in Lua and Erlang/OTP

Kenji Rikitake

7-MAR-2014

for Erlang Factory SF Bay 2014

@jj1bdx

Topics

- Star Trek game and the BSD trek
- Porting to Lua and remaking for Erlang
- Pitfalls and things I've learned
- Demo
- Future plans and references

```
100 Klingons
2 starbases at 6,7, 0,4
It takes 850 units to kill a Klingon
Condition RED

Short range sensor scan
  0 1 2 3 4 5 6 7 8 9
0 . . K . * . . . . . 0   stardate       3600.00
1 . . . K . . . . . . 1   condition      RED
2 . . K . . . . . . . 2   position       6,7/9,7
3 . . . . . K K . . . 3   warp factor    5.0
4 . . . . . . . . . . 4   total energy   5000
5 . . . . . K . . . . 5   torpedoes      10
6 . . . . . . . . . . 6   shields        up, 100%
7 . * K . . . . . . . 7   Klingons left  100
8 . . . . . . . . . . 8   time left      26.00
9 . . . . . . # E . . 9   life support   active
  0 1 2 3 4 5 6 7 8 9
Klingon at 5,5 moves to 9,9
Klingon at 3,5 escapes to quadrant 5,7

Stardate 3600.00: Klingon attack:
HIT: 306 units from 9,9, shields absorb 100%, effective hit 0
HIT: 247 units from 7,2, shields absorb 79%, effective hit 51
HIT: 231 units from 3,6, shields absorb 66%, effective hit 78
HIT: 188 units from 2,2, shields absorb 55%, effective hit 83
HIT: 181 units from 1,3, shields absorb 49%, effective hit 92
HIT: 158 units from 0,2, shields absorb 43%, effective hit 90
Klingon at 7,2 escapes to quadrant 7,7
Klingon at 2,2 moves to 8,8
Klingon at 1,3 escapes to quadrant 5,7
```

This is BSD trek:
a Star Trek text game
by Eric Allman
and UCB BSD team
last updated in 1993

Why Star Trek?

- Small and easy to understand
- Well-known at least since 1970s
- Many implementations: FORTRAN/BASIC, Netrek, a multiplayer game, Android app
- I couldn’t find an Erlang implementation

Then why BSD trek?

- Written in ANSI C / code frozen in 1993
- Still playable on FreeBSD and OS X
- New features (event queues)
- (Obviously) BSD licensed
- And I love BSD. Period.

What I have done

- Reading the BSD Trek code in C
- Porting the BSD Trek to Lua, to understand the data structures and the code flow
- Redesigning a new trek in Erlang/OTP

Why Lua first?

- Small and easy to understand
- Syntax and semantics are like C but better (too bad it's based on a shared-memory programming model)
- Popular among game programmers
- Simple data structure: based solely on key-value tables, no pointer, garbage collected
- <u>Even an Erlang implementation exists!</u>

2. lua

```
compkldist: klingon 2: x = 4, y = 1
compkldist: klingon 3: x = 9, y = 1

Stardate 2600.00: Klingon attack:
*** HIT: 154 units from 7,8, shields absorb 99%, effective hit 0
*** HIT: 117 units from 4,1, shields absorb 89%, effective hit 12
*** HIT: 91 units from 9,1, shields absorb 81%, effective hit 16
klmove: fl = AFTER, Etc.nkling = 3
klmove: klingon number 1 did not move
Klingon at 4,1 escapes to quadrant 2,0
Klingon at 9,1 moves to 3,7
klmove: new Etc.klingon[1] x, y = 7, 8
klmove: skip escaped klingon number 2
klmove: copied Etc.klingon element 3 to 2
klmove: new Etc.klingon[2] x, y = 3, 7
klmove: new klingons: 2
compkldist: klingon 1: x = 3, y = 7
compkldist: klingon 2: x = 7, y = 8
Command: s

Short range sensor scan
  0 1 2 3 4 5 6 7 8 9
0 . . . . . . . . . . 0  stardate      2600.00
1 . . . . . . . # .   E 1  condition     RED
2 . . . . . . . . . . 2  position      2,1/1,9
3 . * . . . . . . K . . 3  warp factor   5.0
4 . . . . . . . . . . 4  total energy  4970
```

Luatrek is for learning how BSD trek works; so lots of trace messages!

Good things about lua

- The table structure is versatile — the module system, sparse arrays, hash tables, etc.
- Less **“goto fail;”** — Lua only allows strict if-then-else-end blocks (on the other hand, no C **continue** statement so **goto is essential)**
- Semantically simple — no malloc()/free(), table constructor built-in ({1,2,3})

An excerpt from luatrek source code

2. vim

```lua
function M.compkldist (f)
    if Etc.nkling == 0 then
        return
    end
    for i = 1, Etc.nkling do
        local dx = Ship.sectx - Etc.klingon[i].x
        local dy = Ship.secty - Etc.klingon[i].y
        local d = math.sqrt((dx * dx) + (dy * dy))
        -- compute average of new and old distances to Klingon
        if not f then
            Etc.klingon[i].avgdist = 0.5 * (Etc.klingon[i].dist + d)
        else
            -- new quadrant: average is current
            Etc.klingon[i].avgdist = d
        end
        Etc.klingon[i].dist = d
    end
    -- leave them sorted
    sortkl()
    -- trace code to dump klingons in the sector
    if V.Trace then
        for i = 1, Etc.nkling do
            printf("compkldist: klingon %d: x = %d, y = %d\n",
                i, Etc.klingon[i].x, Etc.klingon[i].y)
        end
```

klingon.lua [RO] 155,1 27%

A bad thing about Lua

```
kenji@bigmac[1019]% lua
Lua 5.2.2  Copyright (C) 1994-2013 Lua.org, PUC-Rio
> A = {1,2,3}
> print (A == {1,2,3})
false << WHAT? Are you serious?
>
```

- Every table constructor ("{}") makes a different table
- Lua variable only holds the reference (or address of the table (therefore you cannot print a table)
- So the one in A and the latter one are not the same!
- **Comparing two tables** needs **comparing all elements** specifically in a dedicated piece of code

It’s not that bad in Erlang

```
2. beam.smp
Eshell V5.10.4  (abort with ^G)
1> A = {1,2,3}.
{1,2,3}
2> A == {1,2,3}.
true   << Yes!
3>
```

- In Erlang, {1,2,3} is a tuple, nothing else
- Erlang creates a new tuple every time
- And Erlang compares the one in the variable A and the constant tuple in the full complete details
- So the two tuples are considered the same
- This is how it’s supposed to be, isn’t it?

Lua port caveats

- A sequential game — the game program will wait for your input forever
- Very hard to debug — the game state is manipulated in so many different places
- Everything is in the shared tables — virtually impossible to make it asynchronous

Redesign the game in Erlang/OTP: implementation goals of **erltrek**

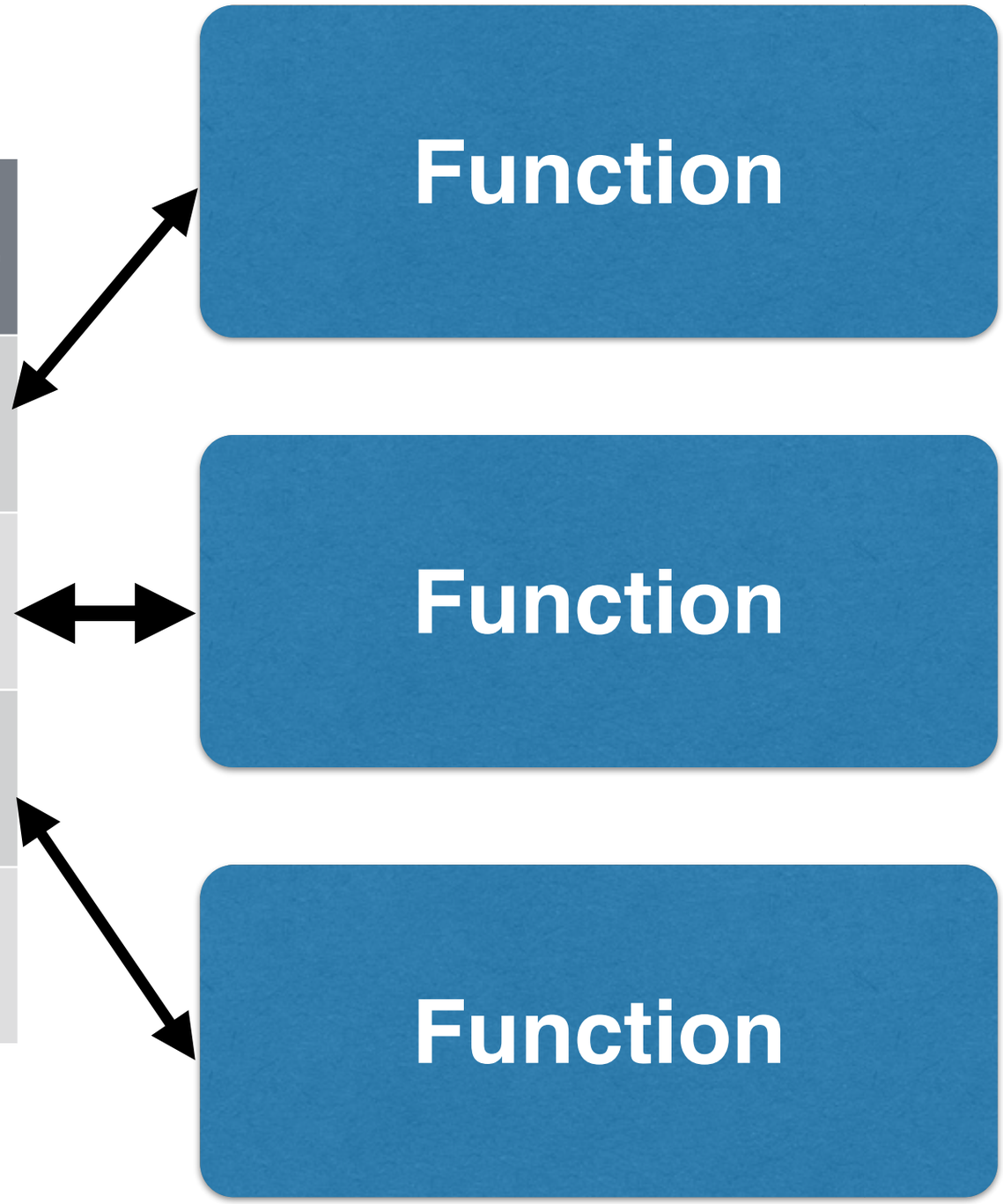
- Make it real-time — the game events happen even **without the user input**
- Get the most out of OTP library modules (no maps yet — it’s built upon R16B03-1)
- Keep the original look and feel as they are, but make them more logical

erltrek data structure

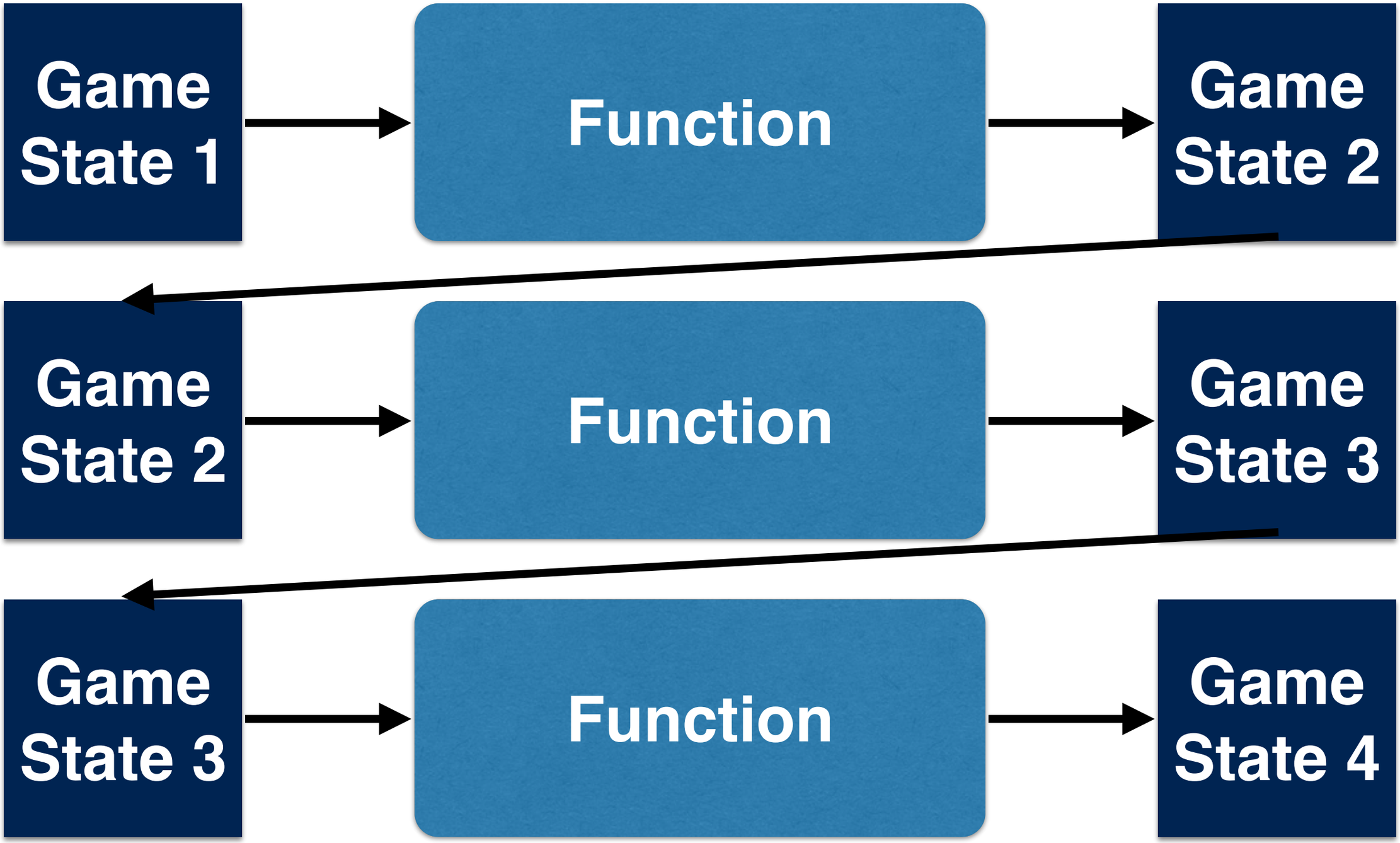
- OTP has a plenty of nice libraries — (sparse) **array**, **orddict**, and especially **dict** (=table)
- Dicts with the Quadrant positions as the keys: stars, inhabited systems, black holes, starbases, and the number of Klingons (maybe maps soon)
- For the current sector: (pseudo-2D) array of the entities, and the dict of Klingons in the sector
- Records for structured data definition

shared-memory data structures

quadrant	Sector position of the stars
{1,3}	[{1,3}, {5,6}, {0,9}]
{2,4}	[{3,7}, {7,1}, {9,2},{4,0}
{0,5}	[{2,3}]
{7,1}	[{2,2}, {0,0}]



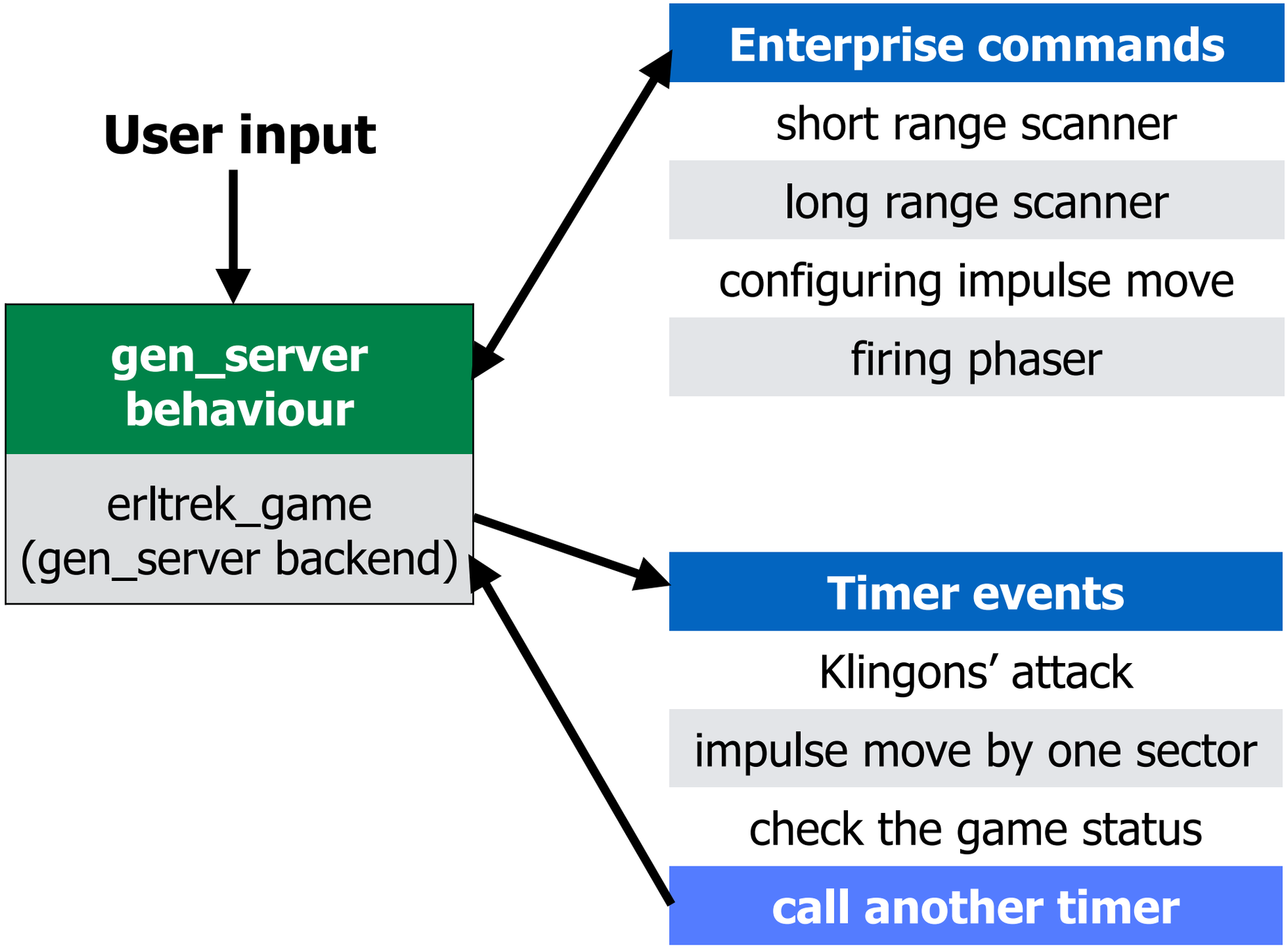
Data flow in Erlang/OTP gen_server behaviour



What you can do in gen_server

- Handle timer events as an unformatted info: erlang:start_timer/3 works the best
- Process call requests with replies — starting the game, entering the game commands
- Process cast requests without replies — ending the game without waiting for input
- All in the same server

erltrek control flow summary



Tips: writing functions for Erlang/OTP gen_server

- function(InputState) -> OutputState
- Erlang expressions always return values, even when the conditional execution is assumed
- Intermediate variables are often redundant
- Use lists instead of loops
- Make a loop or conditional branch as another function, especially to reduce nesting levels

2. vim

```
-spec prepare_phaser(integer(), integer(), integer(), game_state()) -> game
_state().

prepare_phaser(SX, SY, ENERGY, GameState) ->
    {Tick, SHIP, NK, DS, DI, DB, DH, DKQ, SECT, DKS} = GameState,
    % Deplete energy from Enterprise
    SHIPENERGY = SHIP#enterprise_status.energy,
    SHIP2 = SHIP#enterprise_status{energy = SHIPENERGY - ENERGY},
    ShipSC = SHIP#enterprise_status.sectxy,
    % Calculate course for each Klingon
    COURSE = erltrek_calc:sector_course(ShipSC, #sectxy{x = SX, y = SY}),
    LK = dict:fetch_keys(DKS),
    LDIST = [erltrek_calc:sector_distance(ShipSC, SC) || SC <- LK],
    LCOURSE = [erltrek_calc:sector_course(ShipSC, SC) || SC <- LK],
    NewGameState = {Tick, SHIP2, NK, DS, DI, DB, DH, DKQ, SECT, DKS},
    hit_phaser(LK, LDIST, LCOURSE, ENERGY, COURSE, NewGameState).

%% Calculate phaser hit for each klingon

-spec hit_phaser([#sectxy{}], [float()], [float()],
    integer(), float(), game_state()) -> game_state().

hit_phaser([], _, _, _, _, GameState) ->
    GameState; % do nothing if klingon list is empty
erltrek_phaser.erl [RO]                                  134,13          77%
```

2. vim

```
-spec hit_phaser([#sectxy{}], [float()], [float()],
    integer(), float(), game_state()) -> game_state().

hit_phaser([], _, _, _, _, GameState) ->
    GameState; % do nothing if klingon list is empty
hit_phaser(LK, LDIST, LCOURSE, ENERGY, COURSE, GameState) ->
    {Tick, SHIP, NK, DS, DI, DB, DH, DKQ, SECT, DKS} = GameState,
    [SK|LK2] = LK,
    [SDIST|LDIST2] = LDIST,
    [SCOURSE|LCOURSE2] = LCOURSE,
    [K] = dict:fetch(SK, DKS),
    KE = K#klingon_status.energy,
    % Calculate hitting level
    io:format("ENERGY = ~b COURSE = ~.1f SDIST = ~.1f SCOURSE = ~.1f~n",
              [ENERGY, COURSE, SDIST, SCOURSE]),
    HIT = trunc(float(ENERGY) * math:pow(0.9, float(SDIST)) *
              math:exp(-0.7 * abs((SCOURSE - COURSE)/2.0))),
    % Deplete energy from Klingon and update the dict
    io:format("Phaser hit to Klingon at sector ~b,~b level ~b~n",
              [SK#sectxy.x, SK#sectxy.y, HIT]),
    NKE = KE - HIT,
    case NKE > 0 of
        true -> % klingon is alive
            K2 = K#klingon_status{energy = NKE},
            DKS2 = dict:append(SK, K2, dict:erase(SK, DKS)),
erltrek_phaser.erl [RO]                                  136,13        91%
```

2. vim

```
    HIT = trunc(float(ENERGY) * math:pow(0.9, float(SDIST)) *
                math:exp(-0.7 * abs((SCOURSE - COURSE)/2.0))),
    % Deplete energy from Klingon and update the dict
    io:format("Phaser hit to Klingon at sector ~b,~b level ~b~n",
                [SK#sectxy.x, SK#sectxy.y, HIT]),
    NKE = KE - HIT,
    case NKE > 0 of
        true -> % klingon is alive
            K2 = K#klingon_status{energy = NKE},
            DKS2 = dict:append(SK, K2, dict:erase(SK, DKS)),
            GameState2 = {Tick, SHIP, NK, DS, DI, DB, DH, DKQ, SECT, DKS2},
            hit_phaser(LK2, LDIST2, LCOURSE2, ENERGY, COURSE, GameState2);
        false -> % klingon is killed
            io:format("Klingon at sector ~b,~b killed~n",
                [SK#sectxy.x, SK#sectxy.y]),
            QC = SHIP#enterprise_status.quadxy,
            DKQ2 = dict:store(QC, dict:fetch(QC, DKQ) - 1, DKQ),
            DKS3 = dict:erase(SK, DKS),
            NK2 = NK - 1,
            SECT2 = array:set(erltrek_setup:sectxy_index(SK), s_empty, SECT
),
            GameState3 = {Tick, SHIP, NK2, DS, DI, DB, DH, DKQ2, SECT2, DKS
3},
            hit_phaser(LK2, LDIST2, LCOURSE2, ENERGY, COURSE, GameState3)
    end.
```

erltrek_phaser.erl [RO] 173,8 Bot

erltrek implementation status as of 2-MAR-2014

- Timer-based tick — game proceeds without input!
- Short/long range scanner
- Impulse engine for Enterprise
- Attacks from Klingons
- Phaser attack from Enterprise
- (Yes, we need more)

Future goals

- Implement more of the basic functionality
- Fix bugs and refactor code
- Rewrite using maps once 17.0 is available
- Put more asynchronously-active entities
- Build a generic API and GUI client

Related GitHub repos

- erltrek
- luatrek
- bsdtrek: a snapshot of FreeBSD games Port
- tinymt-erlang PRNG (used in erltrek)

Erlang tools

- Vim (or Emacs) (I use both) (no editor wars)
- Erlang documentation (local copy preferred)
- Books: Joe's, Francesco/Simon's, LYSE, OTP in Action, Introducing Erlang, etc.
- dialyzer, type spec, observer, gen_server (OTP Design Principles)

Lua tools

- Vim (or Emacs) (I use both) (no editor wars)
- Book: Programming In Lua
- Luarocks package manager and repositories
- Libraries: Penlight and Ldoc

Acknowledgments

- Eric Allman, father of BSD Trek
- Francesco Cesarini, Erlang Solutions boss
- Robert Virding, author of luerl
- Fréd Hébert, author of LYSE
- Kyoko Rikitake, partner of my life

Thanks

Questions?