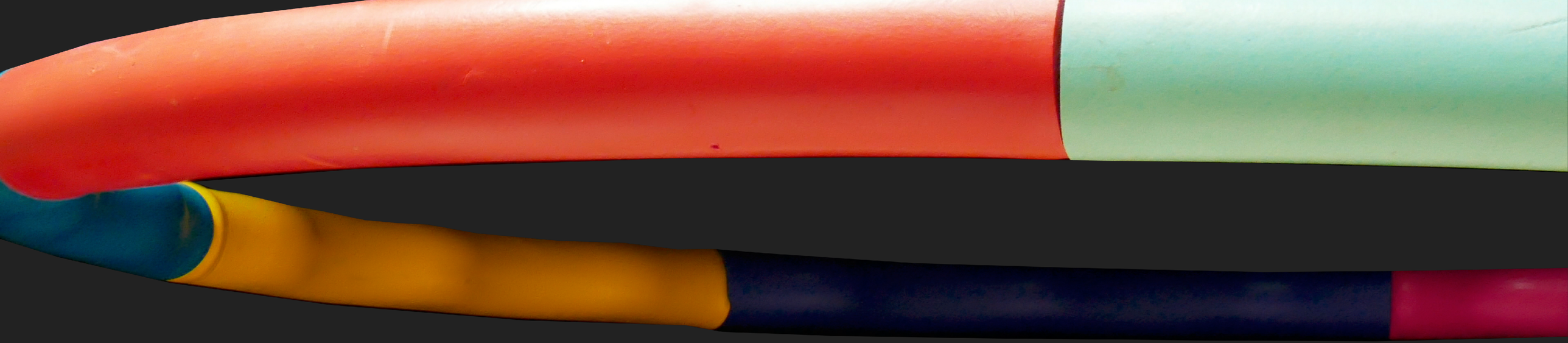


HOOPING WITH xC

TORUS HEAT EQUATIONS TO GET DIZZY FROM

✅

FRINGE PRESENTATION AT IEEE-COPA 2021

ieee-copa.org IEEE COMPUTER SOCIETY

ØYVIND TEIG
DR. LAWRENCE JOHN DICKSON

Last edits: Øyvind: 24. April 2021 at 17.17 Norway time ✅ - Larry: 18. April 2021 at 14:23 California time ☑️

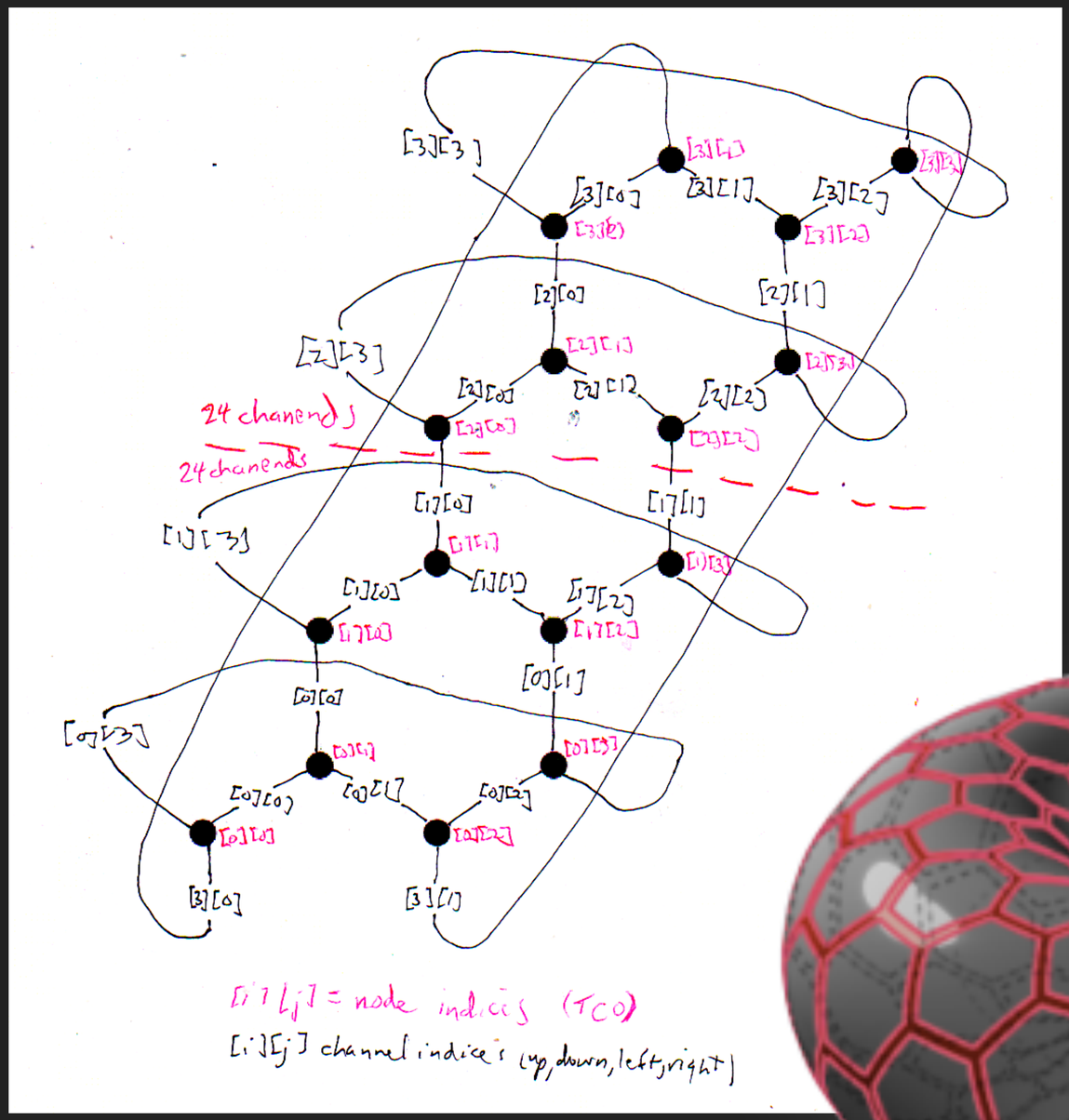
ØYVIND TEIG

- 40(+) years with safety critical systems at
- Autronica Fire and Security AS. HW + SW
- Was *Senior development engineer* with
- Master's degree («Siv.ing») from NTH (now NTNU) in 1975
- Retired since June 2017
- Now blogging: www.teigfam.net/oyvind/home/
  - Disclaimer: no money, no gifts, no ads - only fun and expenses
- I have no association with XMOS
- But I am using xC and XMOS boards in several projects
- Now also NTNU affiliated: www.ntnu.edu/employees/oyvind.teig
- I live in Trondheim, Norway

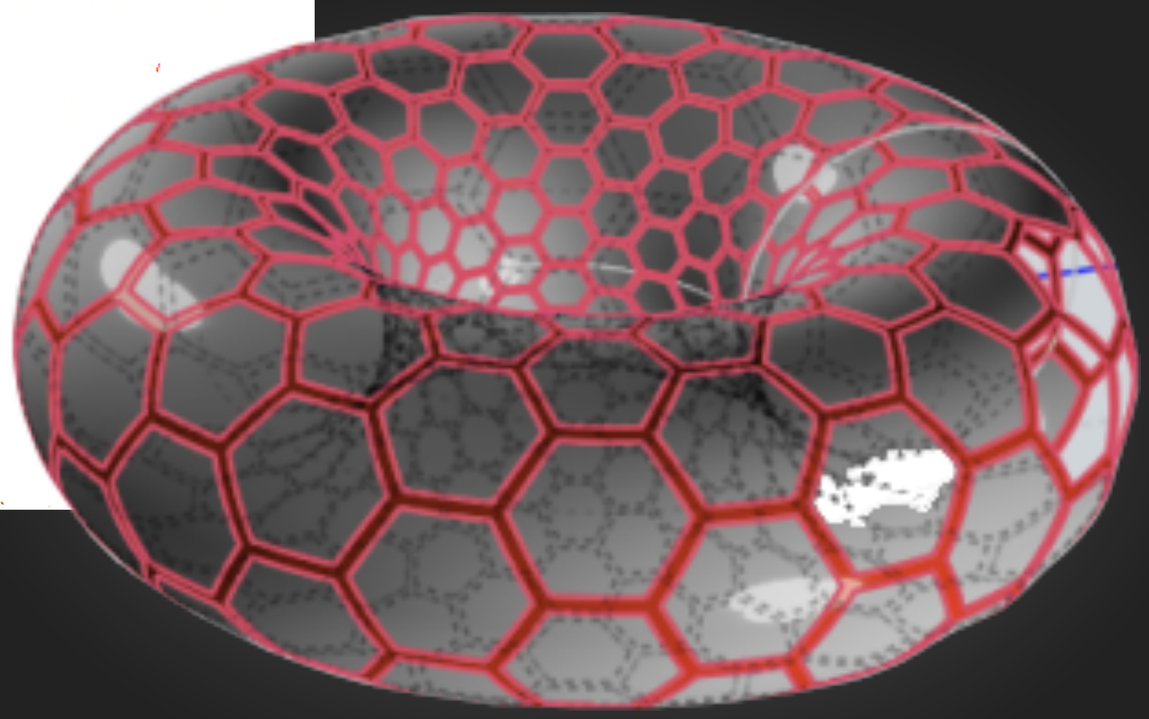
✅

DR. LAWRENCE JOHN DICKSON

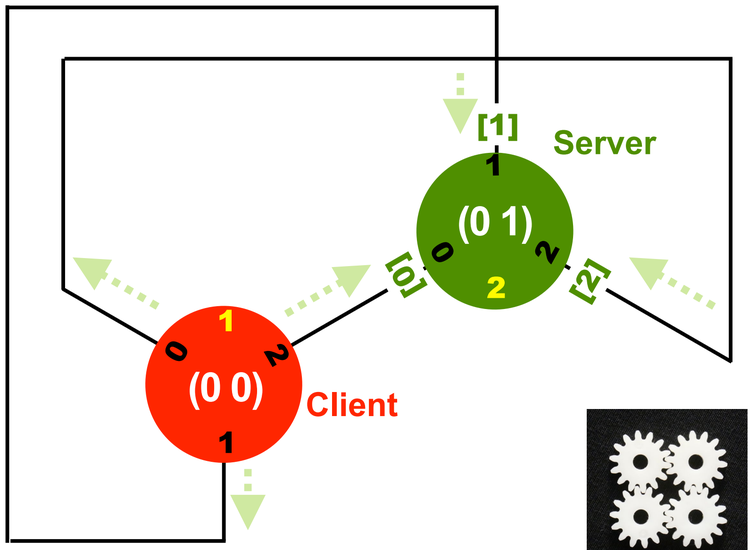
Chief Scientist, Space Sciences Corporation

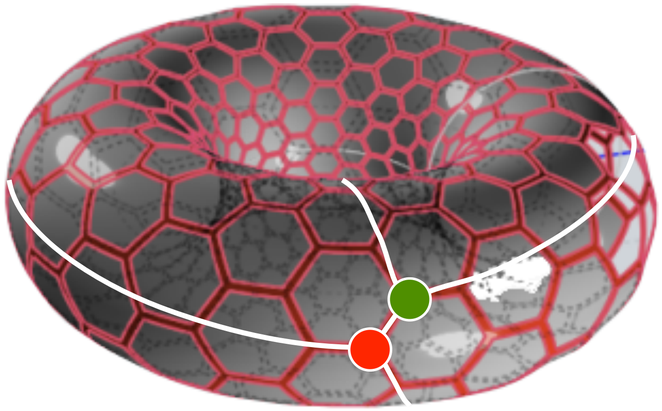


- After the xC/xCORE had been introduced to Larry,
- then these drawings started our joint venture
- Plus some code examples coded in occam (for transputers), and tested in occam and C
- (Larry and Øyvind met at «CPA conferences»)



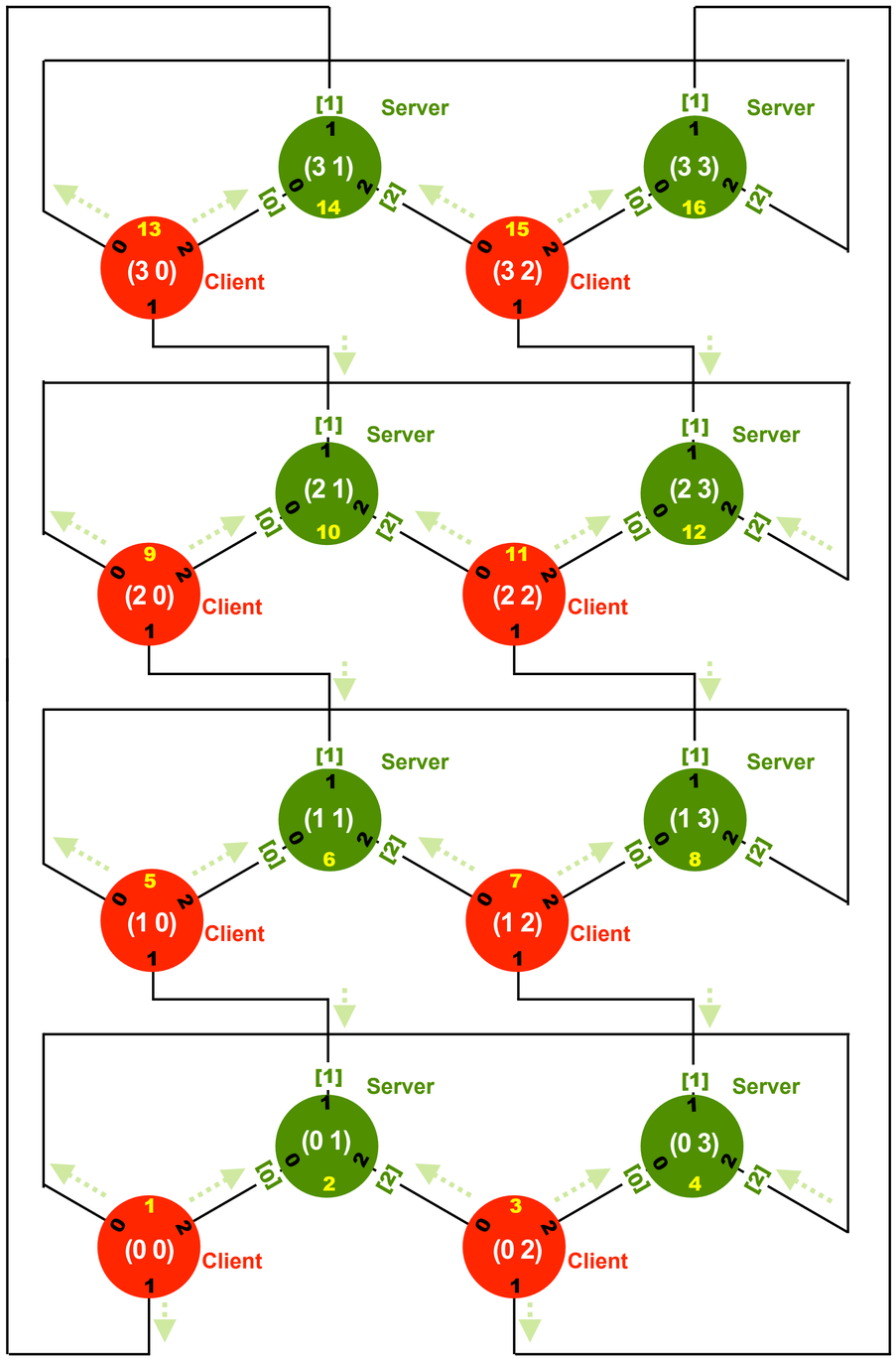
PROBLEM DESCRIPTION (1)





- The story on how points representing temperature on a torus, in the form of these occam PROCs running on one or several transputers,
- exchanging temperature with their neighbours,
- caught the interest of *whether* or *how* this problem may be run as xC tasks on the XMOS xCORE multicore architecture

xC is "C plus multicore extensions" (XMOS)



Drawn as a strict table is not as «torus-intuitive»

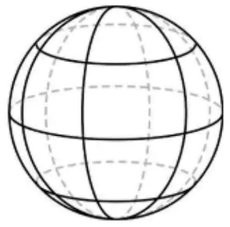
✅

PROBLEM DESCRIPTION (2)

- INMOS, occam, transputer → XMOS, xC, xCORE, both in Bristol, UK
- Both do concurrency with built-in schedulers, for those applications (and users) who enjoy living without any operating system layer
- However, xC looks more like C, the toolset even compiles C and C++ (built using Clang/LLVM),
- plus the xCORE has more concurrency primitives realised as HW units. Now hard deadlines may be guaranteed.
- The xC/xCORE probably was not designed for the «torus kind» of problem. But how far can we stretch it?

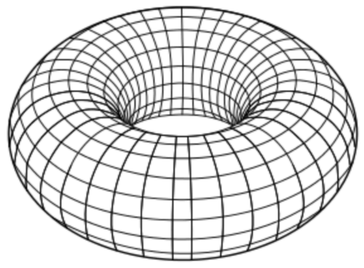
GEOMETRY

Sphere



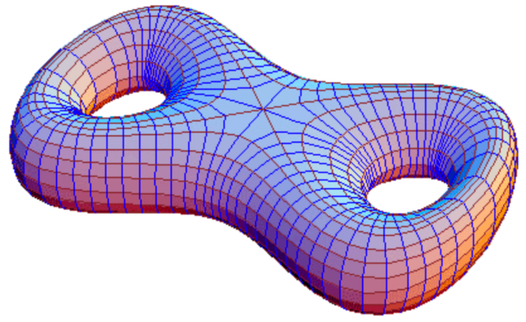
Elliptic

Torus



Euclidean

Double Torus

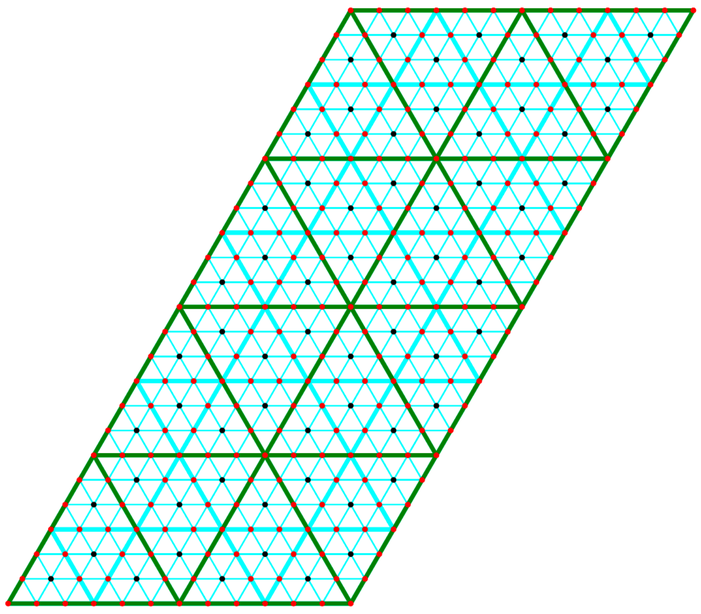


Hyperbolic

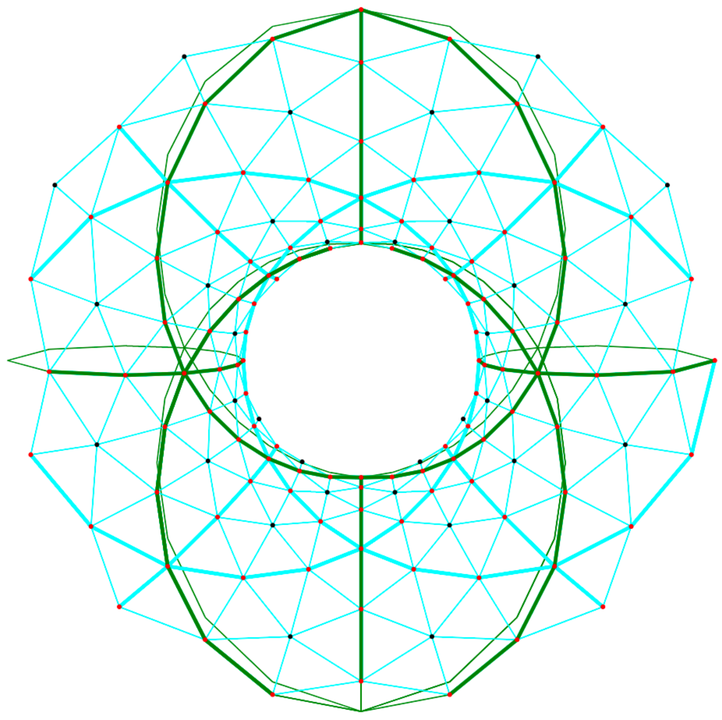
Green is 4 x 4
Thick cyan is 8 x 8
Thin cyan is 24 x 24

Euclidean has no exceptional vertices

Thin green is behind the Torus
All others are in front

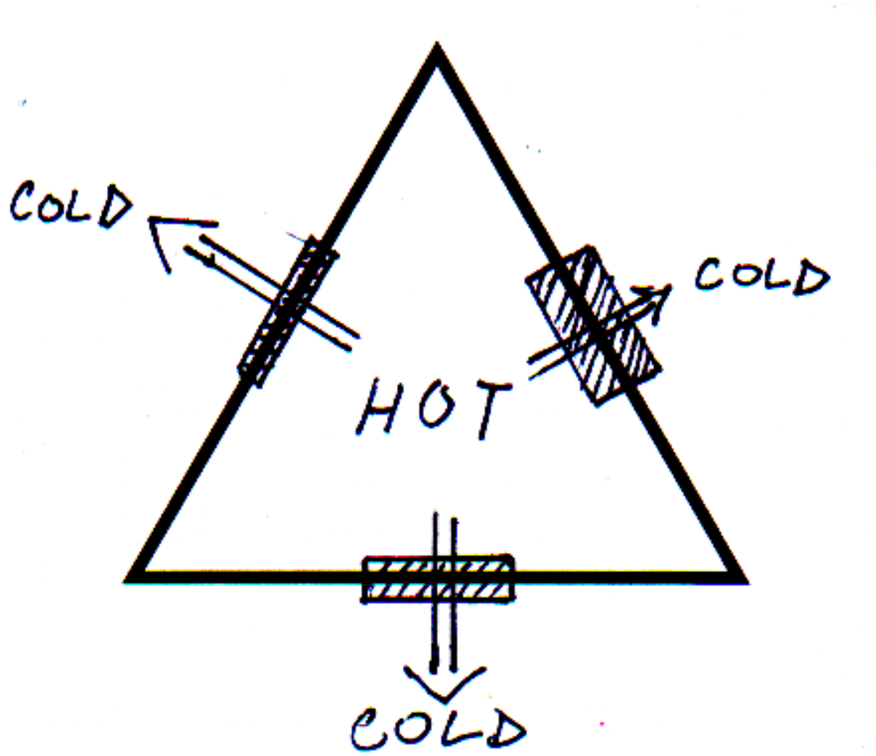


PARALLELOGRAM



TORUS

Linear heat flow




```
Each node (triangle):
  Change in temperature (K) =
    Sum{Energy flow(W)}*Time(sec)/
      (Heat capacity)
Each connection (edge):
  Energy flow(W) =
    Temperature difference(K)*
      (Heat conductance)
```

- Three physical connections per node
- Heat capacities may vary between nodes
- Heat conductances may vary between connections
- Energy transmission is local
- Energy transmission is linear
- Energy is conserved in the system

MATHEMATICS

The parallelogram can be mapped conformally onto the torus.
Left and right edge are identified, and bottom and top edge are identified.

The torus is made by rotating a circle, whose radius is the minor radius r, around an axis that is in the plane of the circle and does not intersect it. The distance from the axis to the center of the circle (called the spine) is the major radius R. Note that R > r.

We may assume the axis is the z-axis and the spine is in the xy-plane. Then the torus can be parametrized by θ and ϕ where $(x, y, z) = ((R + r\cos(\theta))\cos(\phi), (R + r\cos(\theta))\sin(\phi), r\sin(\theta))$ where the variables are everywhere orthogonal. Since the arc length derivatives are $\frac{ds}{d\theta} = r$ and $\frac{ds}{d\phi} = R + r\cos(\theta)$, it follows from the chain rule that if $\theta = \theta(\psi)$ and $r\frac{d\theta}{d\psi} = R + r\cos(\theta)$, then the map from (ψ, ϕ) onto the torus is locally conformal and hence conformal.

This differential equation is easily solved for the inverse function $\psi(\theta)$: $\frac{d\psi}{d\theta} = \frac{r}{R + r\cos(\theta)}$, which by tables of integrals has the solution $\psi = \frac{2r}{\sqrt{R^2 - r^2}} \tan^{-1}\left(\sqrt{\frac{R - r}{R + r}} \tan\frac{\theta}{2}\right)$, whence it follows that $\theta = 2\tan^{-1}\left(\sqrt{\frac{R + r}{R - r}} \tan(\frac{\sqrt{R^2 - r^2}}{2r}\psi)\right) = 2\tan^{-1}\left(\sqrt{\frac{R + r}{R - r}} \tan\frac{\chi}{2}\right)$, where $\chi = \frac{\sqrt{R^2 - r^2}}{r}\psi$. Here, $-\pi \le \chi \le \pi$ and $-\pi \le \phi \le \pi$. So the aspect ratio of the rectangle (or parallelogram) conformally mapped onto the torus is $\frac{r}{\sqrt{R^2 - r^2}}$.

The aspect ratio of parallelogram mapped onto the torus of the Geometry slide is $\frac{1}{\sqrt{3}}$.

This allows the triangles mapped onto the Torus to be equilateral, with same i and j dimensions.
These dimensions have to be even, allowing for Client and Server classification of triangles.

COMPUTATION (CSP/OCCAM MODEL)

The new temperature calculation implied by the physics is shown in C as

```
temp_new = temp_old + (tempin[0]-temp_old)*wt[0] +
                      (tempin[1]-temp_old)*wt[1] +
                      (tempin[2]-temp_old)*wt[2];
```

where a constant wt[i] value is preset:

```
wt[i] = heat_conductance[i]*deltime/heat_capacity;
```

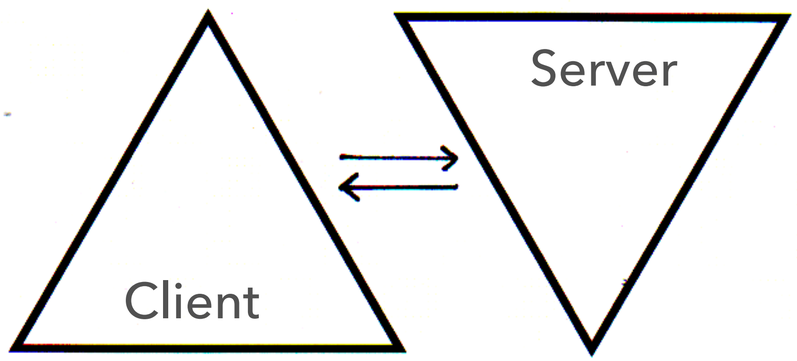
This is easily modeled in CSP communicating processes, one for each node, if neighbors transmit their old temperatures to each other to supply tempin[i]. Thus, each cycle requires each node to do three outputs and three inputs, followed by the above calculation. In occam, this code snippet is

```
WHILE notdone
  SEQ
    PAR
      to.left ! temp.old
      to.vert ! temp.old
      to.right ! temp.old
      from.left ? tempin[0]
      from.vert ? tempin[1]
      from.right ? tempin[2]
    temp.new := temp.old +
                (( ((tempin[0]-temp.old)*wt[0]) +
                   ((tempin[1]-temp.old)*wt[1]) ) +
                   ((tempin[2]-temp.old)*wt[2]))
```

Neither does xC *par*
(even if it is less scalable than the occam PAR)

Note that occam PAR makes no assumptions about which communication fires first.

COMPUTATION (CLIENT/SERVER)



Triangles with base down are clients, those with base up are servers. Clients share edges only with servers, and vice versa. This makes it possible to model this code (8x8) in xC (Øyvind Teig)!

It can still be modeled in CSP/occam, with servers always doing their input first (see below). Surprisingly, this proves to be possible also for the other topologies (sphere, double-torus, etc) - a recent discovery (Larry Dickson).

Client

```
WHILE notdone
  SEQ
    PAR
      SEQ
        to.left ! temp.old
        from.left ? tempin[0]
      SEQ
        to.vert ! temp.old
        from.vert ? tempin[1]
      SEQ
        to.right ! temp.old
        from.right ? tempin[2]
    temp.new := temp.old +
                (( ((tempin[0]-temp.old)*wt[0]) +
                   ((tempin[1]-temp.old)*wt[1]) ) +
                   ((tempin[2]-temp.old)*wt[2]))
```

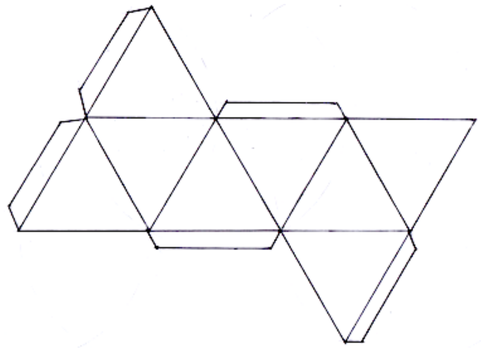
Server

```
WHILE notdone
  SEQ
    PAR
      SEQ
        from.left ? tempin[0]
        to.left ! temp.old
      SEQ
        from.vert ? tempin[1]
        to.vert ! temp.old
      SEQ
        from.right ? tempin[2]
        to.right ! temp.old
    temp.new := temp.old +
                (( ((tempin[0]-temp.old)*wt[0]) +
                   ((tempin[1]-temp.old)*wt[1]) ) +
                   ((tempin[2]-temp.old)*wt[2]))
```

For either a client or a server, there are only 90 possible orders of communications, not 720 or 6! as for the trivial PAR. The server can be modeled by ALTs

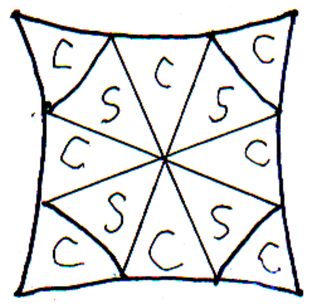
xC: `select()`

OTHER GEOMETRIES (CLIENT/SERVER)

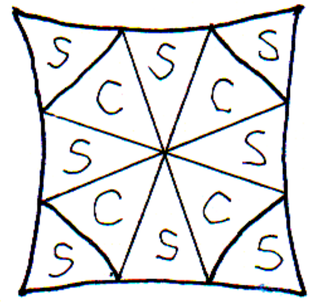


SPHERE: Project a regular octahedron on it (shown unfolded to left) by central projection. The triangles to the left are client if base on bottom and server if base on top, as usual. This proves to be consistent when closed at tabs. For further subdivision, see technique at bottom.

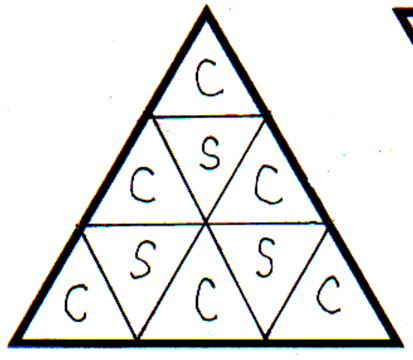
M-FOLD TORUS (M>1): Express as M+1-legged stool with top and bottom seat (shown left for M=2). Cut vertically along axis into 2(M+1) half-legs, unfold each into a hyperbolic quadrilateral, alternating client-type and server-type, subdivided into triangles as at right.

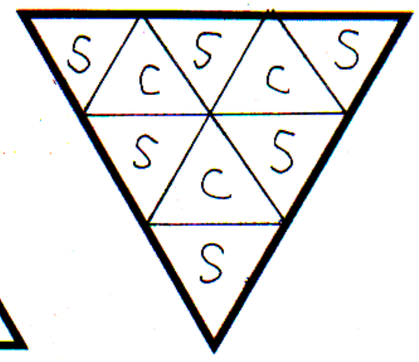


Client-type



Server-type





FURTHER SUBDIVISION: For any n>1, subdivide each large triangle into n^2 small ones, client into client-type, server into server-type, as shown to right (n=3). Entire boundary is same type in each case, so coarse correctness carries over.

MY XCORE-200 EXPLORERKIT BOARDS' PROCESSOR

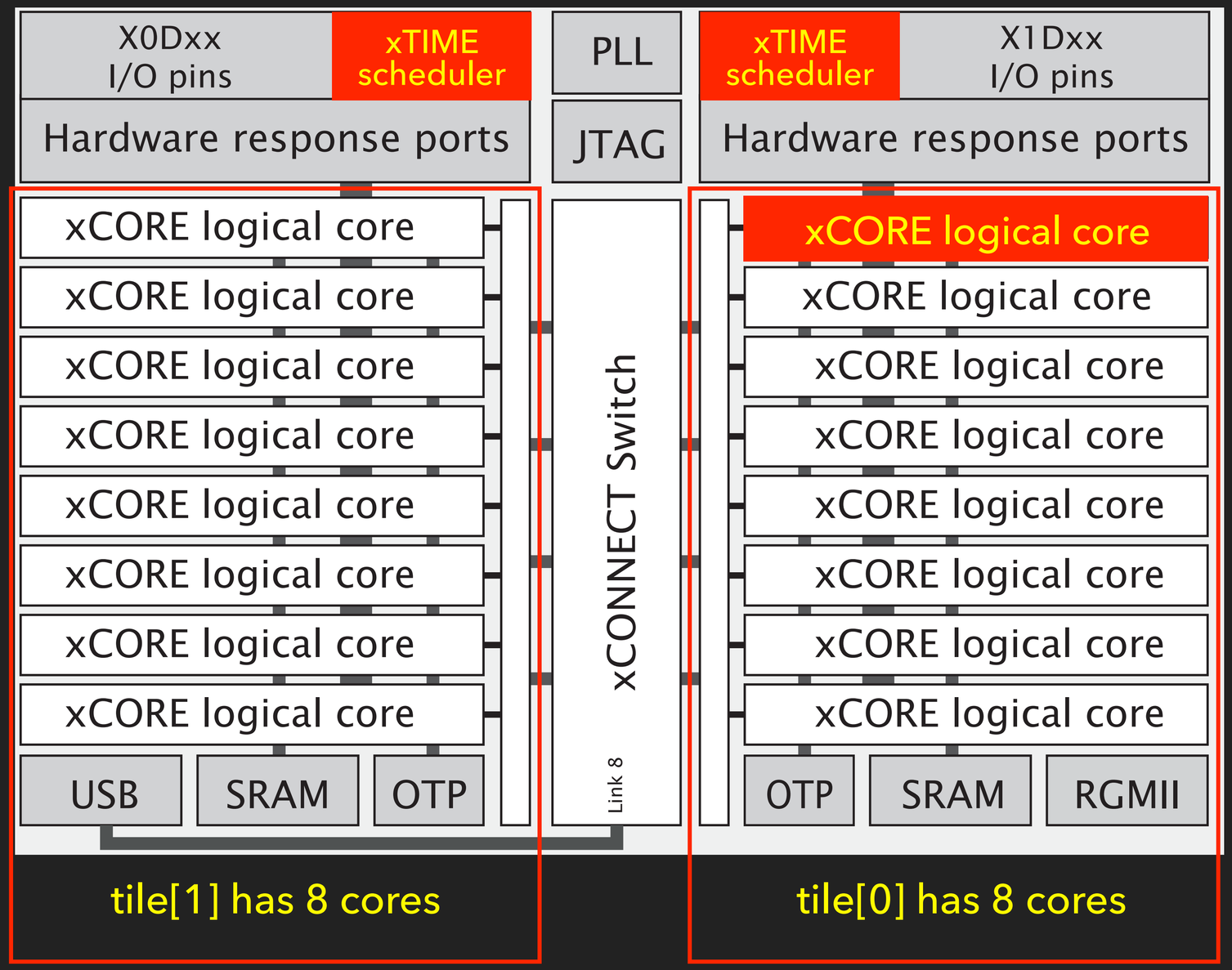
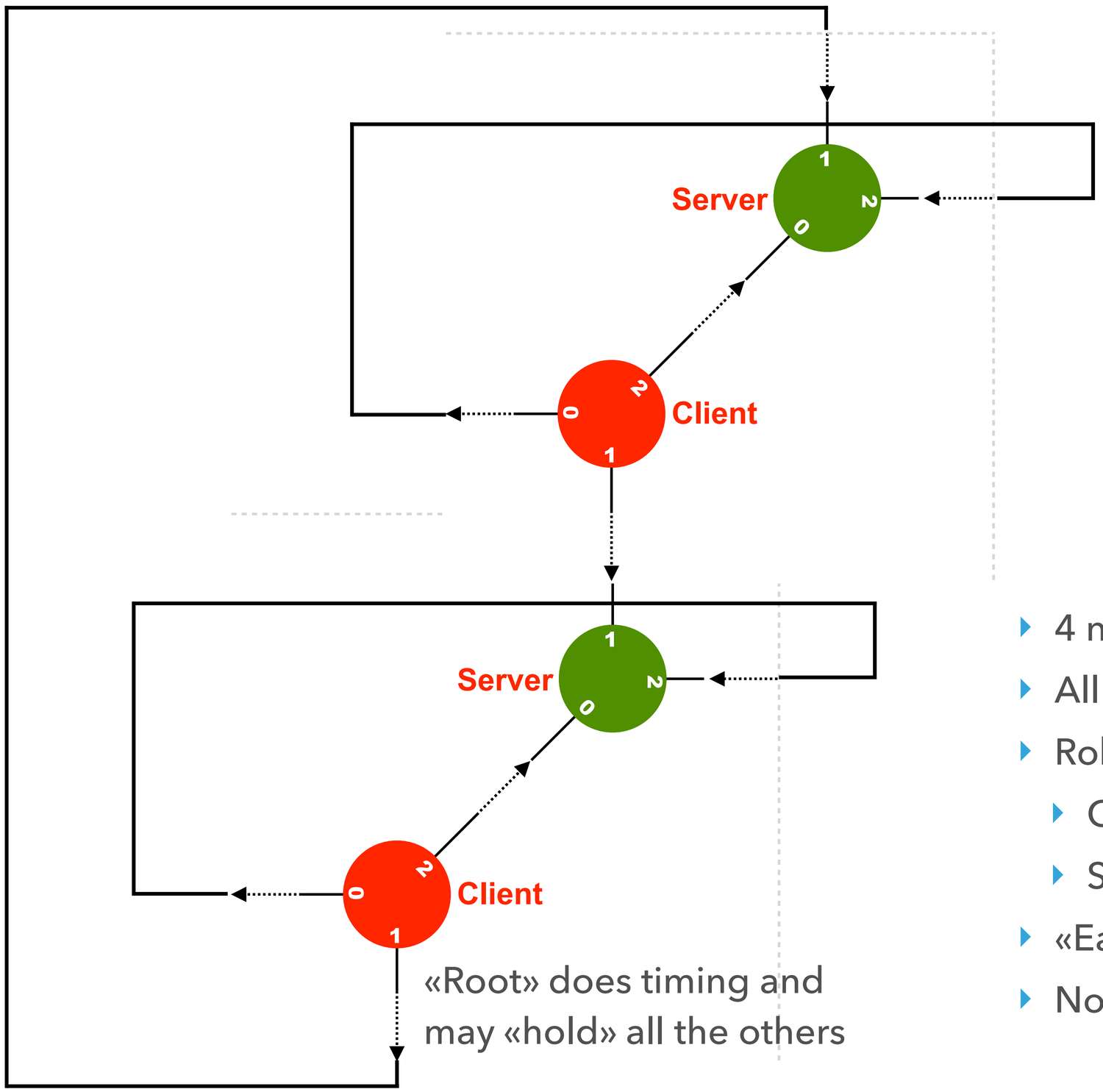


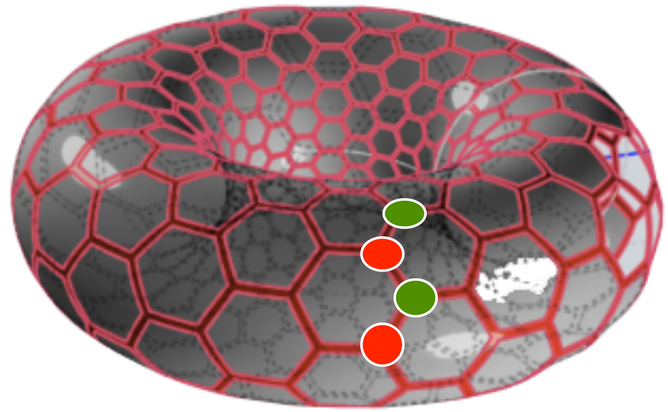
Figure 1: XEF216-512-TQ128 block diagram, from **XEF216-512-TQ128 Datasheet**. 2018/03/23 Document Number: X006990 https://www.xmos.ai/download/XEF216-512-TQ128-Datasheet(1.15).pdf.
As used in the xCORE-200 eXplorerKIT (external flash)

- 2 tiles (500 MIPS per tile (or dual))
- 8 cores per tile (=«Logical cores»)
- xTIME scheduler. If # cores active:
  - 1-5 cores: 1/5 cycles each
  - 6-8 cores: all cycles shared out
  - Deterministic thread execution
- Channels: untyped. Synch or buffered
  - xC `chanends` (32 per tile)
- xC `interface` (typed and role/session)
  - I/O ports
  - Clock blocks (6 per tile)
  - Timers (10 per tile)

✅

TORUS





- 4 nodes = 4 tasks
- All equal as the occam initial example?
- Roles (xC forces us to, as **i * j** increases)
  - Client
  - Server
- «Eager» task not allowed
- No sliding (=no data loss)

Drawn like a parallelogram is more «torus-intuitive»

✅

AND WE PROGRAMMED IN OCCAM. HERE, A ONE-PAGER

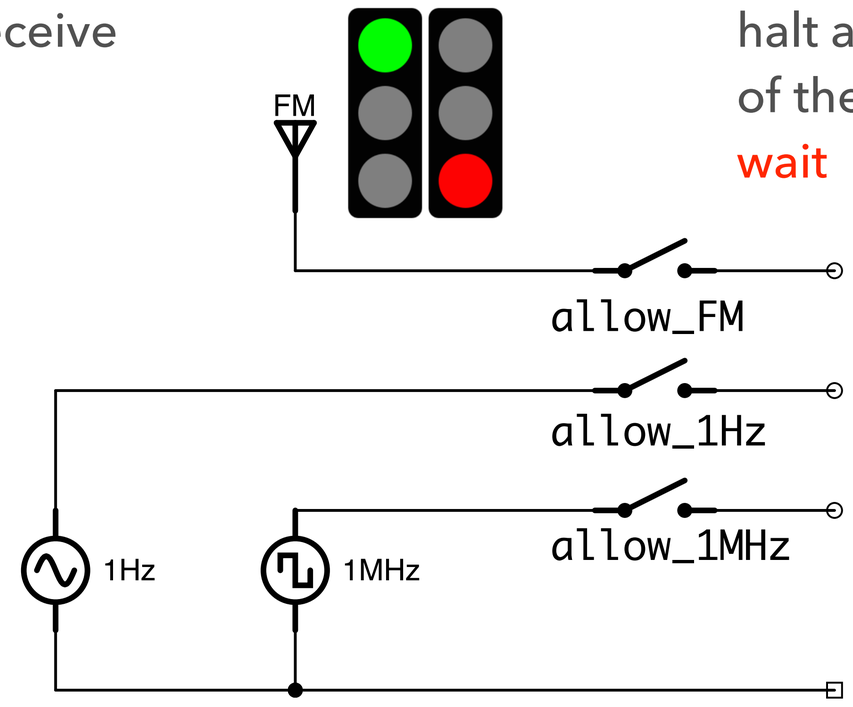
INMOS Limited (UK). Assembly of the two texts done by Øyvind Teig from scans of the respective book covers, Public domain, via Wikimedia Commons

```
PAR
PROC
CHAN OF my_protocol
ALT
```

xC/go: `select`

«ALT specifies a list of guarded commands. The guards are a combination of a boolean condition and an input expression (both optional)» (Wikipedia)

If `allow_FM=FALSE` sender will block until receiver is able to receive



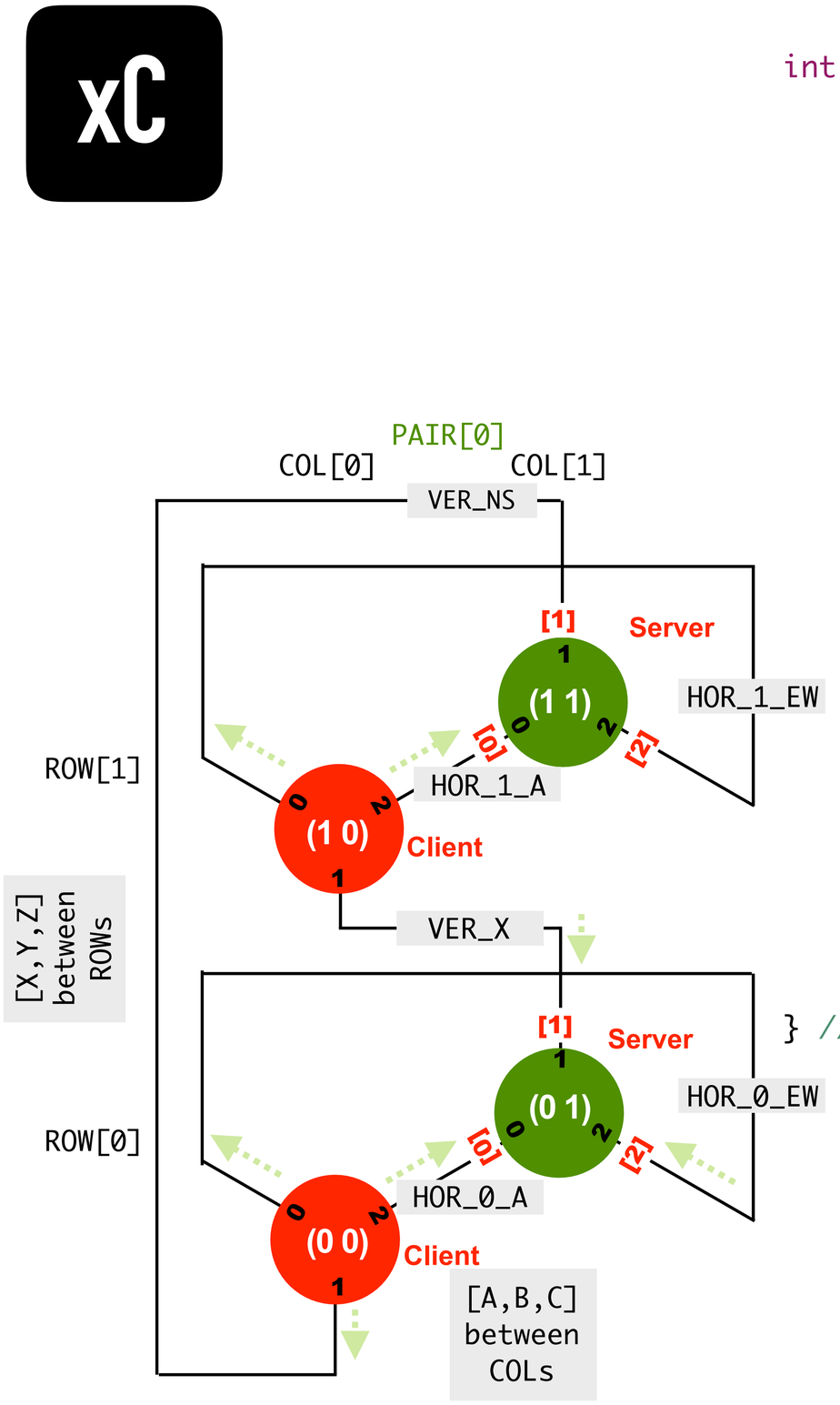
Maybe FM instead is a streamed movie from a server which we may halt and restart and don't loose any of the action? That server will nicely wait

```
WHILE TRUE
  SEQ
    ALT
      (allow_FM=TRUE) & chan_1 ? data_from_FM
        -- handle data_from_FM
      (allow_1Hz=TRUE) & chan_2 ? data_from_1Hz
        -- handle data_from_1Hz
      (allow_1MHz=TRUE) & chan_3 ? data_from_1MHz
        -- handle data_from_1MHz
    Handle_common (..)
```

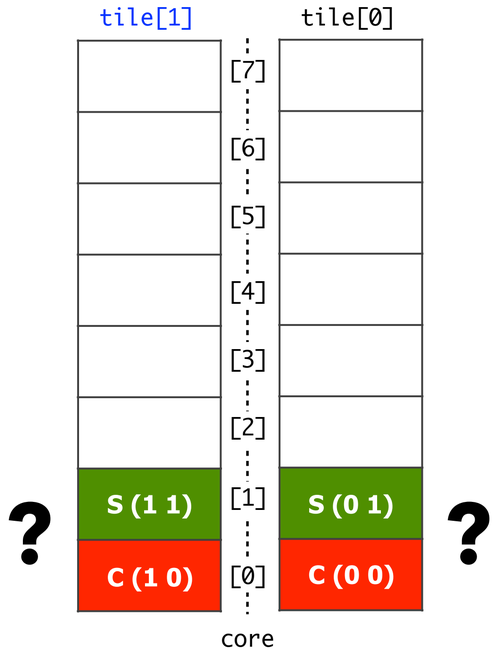
xC/go: `select`

Wanted («good») blocking as waiting since a task (of many - not as single threaded) would have nothing else to do during this waiting. This is CSP. See blog note 092

TEMPERATURE FLOW ON A TORUS. FOUR TASKS?




```
int main (void) {
    #define VER_NS_X [1][0][1] // NS      North-South top-bottom
    #define HOR_1_EW [1][0][2] // EW      EastWest belt side-to-side
    #define HOR_1_A  [1][0][0] // [A,B,C] Between horisontal columns
    #define VER_01_X [0][0][1] // [X,Y,Z] Between vertical rows
    #define HOR_0_EW [0][0][2] // EW      EastWest belt side-to-side
    #define HOR_0_A  [0][0][0] // [A,B,C] Between horisontal columns

    chan chans [NUM_ROWS][NUM_PAIRS_PER_ROW][NUM_CONNS_PER_NODE];
    //         [    2   ][         1       ][        3         ]
    par {
        on tile[0]:
        par {
            Client_node_root (0,0, chans HOR_0_EW, chans VER_NS_X, chans HOR_0_A,
                              outP4_leds);
            Server_node      (0,1, chans HOR_0_A,  chans VER_01_X, chans HOR_0_EW);
        }
        on tile[1]:
        par {
            Client_node (1,0, chans HOR_1_EW, chans VER_01_X, chans HOR_1_A);
            Server_node (1,1, chans HOR_1_A,  chans VER_NS_X, chans HOR_1_EW);
        }
    }
    return 0;
} // main
```



xC: CLIENT AND SERVER ROLES SAVES US TASKS

```
void Client_node (
        const unsigned iof_row,
        const unsigned iof_col,
        chanend        chan_0,
        chanend        chan_1,
        chanend        chan_2) {

    node_context_t context;
    const params_t params    = init_params ();
    temp_degC_t    temp_degC = init_temp (iof_row,iof_col, par

    init_wt      (iof_row, iof_col, context.wt, params);
    init_context (iof_row, iof_col, context);

    while (1) { // Client_node_3_composite_par
        par
        {
            { // subtask 1
                chan_0 <: temp_degC;
                chan_0 :> context.temps_degC[0];
            }
            {  // subtask 2
                chan_1 <: temp_degC;
                chan_1 :> context.temps_degC[1];
            }
            {  // subtask 3
                chan_2 <: temp_degC;
                chan_2 :> context.temps_degC[2];
            }
        }
        context.cycle_cnt++;
        temp_degC = calculate_new_temp_from_flow (context, tem
    }
}
```

Clients send first, always take initiative

Used bidirectionally

```
void Server_node (
        const unsigned iof_row,
        const unsigned iof_col,
        chanend        chan_0,
        chanend        chan_1,
        chanend        chan_2) {

    node_context_t context;
    const params_t params    = init_params ();
    temp_degC_t    temp_degC = init_temp (iof_row,iof_col, params);

    init_wt      (iof_row, iof_col, context.wt, params);
    init_context (iof_row, iof_col, context);

    while (1) {
        par
        {
            { // subtask 1
                chan_0 :> context.temps_degC[0];
                chan_0 <: temp_degC;
            }
            { // subtask 2
                chan_1 :> context.temps_degC[1];
                chan_1 <: temp_degC;
            }
            { // subtask 3
                chan_2 :> context.temps_degC[2];
                chan_2 <: temp_degC;
            }
        }
        context.cycle_cnt++;
        temp_degC = calculate_new_temp_from_flow (context, temp_degC);
    }
}
```

Servers respond to input only

✅

AS xC NORMAL TASK TYPE (3) SHOWING COMMUNICATION PATTERN

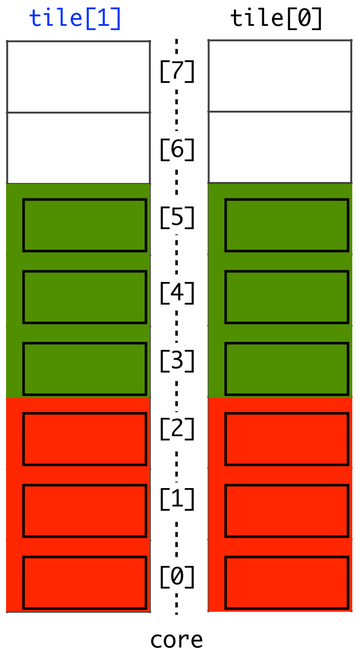
(Names don't match previous page, they are according to blog 217)

```
while (1) { // Client_node_3_composite_par
    par
    {
        { // subtask 1
            chan_0 <: value;
            chan_0 :> rx_val[0];
        }
        {  // subtask 2
            chan_1 <: value;
            chan_1 :> rx_val[1];
        }
        {  // subtask 3
            chan_2 <: value;
            chan_2 :> rx_val[2];
        }
    }
    value = value + rx_val[0] + rx_val[1] + rx_val[2];
    if (value > VALUE_MAX) {
        value = row_col_to_number (iof_row, iof_col);
    } else {}
}

while (1) { // Client_node_3_composite_par_root
    par
    {
        { // subtask 1
            chan_0 <: value;
            chan_0 :> rx_val[0];
        }
        {  // subtask 2
            chan_1 <: value;
            chan_1 :> rx_val[1];
        }
        {  // subtask 3
            chan_2 <: value;
            chan_2 :> rx_val[2];
        }
    }
    value = value + rx_val[0] + rx_val[1] + rx_val[2];
    if (value > VALUE_MAX) {
        value = row_col_to_number (iof_row, iof_col);
    } else {}
    // Delay only here
}
```

Since all in synch!

```
while (1) { // Server_node_3_composite_par
    par
    {
        { // subtask 1
            chan_0 :> rx_val[0];
            chan_0 <: value;
        }
        { // subtask 2
            chan_1 :> rx_val[1];
            chan_1 <: value;
        }
        { // subtask 3
            chan_2 :> rx_val[2];
            chan_2 <: value;
        }
    }
    value = value + rx_val[0] + rx_val[1] + rx_val[2];
    if (value > VALUE_MAX) {
        value = row_col_to_number (iof_row, iof_col);
    } else {}
}

while (1) { // Server_node_3_composite_par
    par
    {
        { // subtask 1
            chan_0 :> rx_val[0];
            chan_0 <: value;
        }
        { // subtask 2
            chan_1 :> rx_val[1];
            chan_1 <: value;
        }
        { // subtask 3
            chan_2 :> rx_val[2];
            chan_2 <: value;
        }
    }
    value = value + rx_val[0] + rx_val[1] + rx_val[2];
    if (value > VALUE_MAX) {
        value = row_col_to_number (iof_row, iof_col);
    } else {}
}
```

4 tasks with
3 subtasks each =
12 tasks on
12 cores on
2 tiles



75% used!

✅

xC HAS TASK TYPES

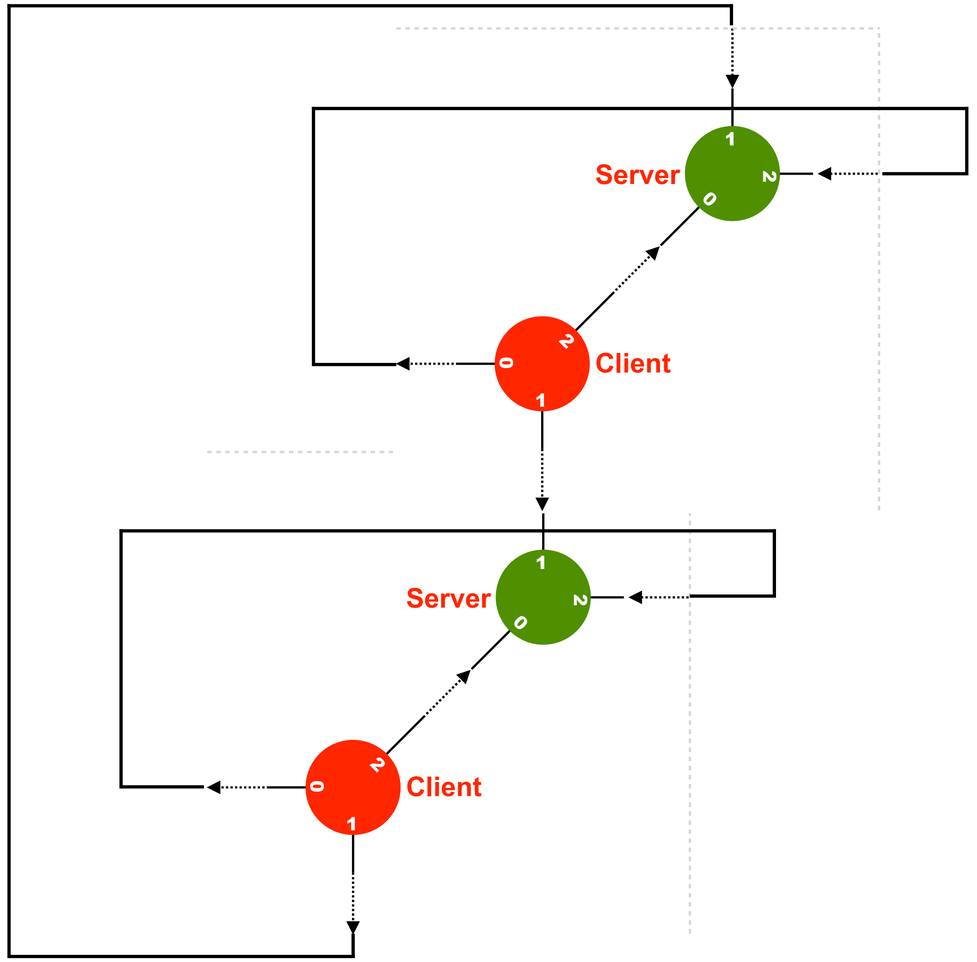
Task type	Usage
Normal No restrictions	Tasks run on a logical core and run independently to other tasks. The tasks have predictable running time and can respond very efficiently to external events.
Combinable Combined selects	Combinable tasks can be combined to have several tasks running on the same logical core. The core swaps context based on cooperative multitasking between the tasks driven by the compiler.
Distributable Called inline	Distributable tasks can run over several cores, running when required by the tasks connected to them.

From the XMOS Programming guide

xC «NORMAL» TASK TYPE

```
Constraint check for tile[0]:
  Cores available:            8,   used:          6 . OKAY
  Timers available:          10,   used:          6 . OKAY
  Chanends available:        32,   used:          7 . OKAY
  Memory available:      262144,   used:       5220 . OKAY
    (Stack: 1076, Code: 3328, Data: 816)
Constraints checks PASSED.
Constraint check for tile[1]:
  Cores available:            8,   used:          6 . OKAY
  Timers available:          10,   used:          6 . OKAY
  Chanends available:        32,   used:          7 . OKAY
  Memory available:      262144,   used:       4908 . OKAY
    (Stack: 1076, Code: 3060, Data: 772)
Constraints checks PASSED.
Build Complete
```

- Many tasks (max 16) and one per logical core is what (I think) the xCORE was originally built for
- Having so much of what's needed for concurrent programs in HW
- But xC tasks were probably supposed to be rather unequal, rather large, and possibly with hard and deterministic timing requirements
- Doing a mesh of a multitude of uniform, smaller tasks therefore challenges the xCORE/xC



✅

COMMENT

- Finding the right way to make this large amount of tasks compilable and then runnable, has been the main challenge
- However, we did «find» several ways to implement it
- But runnability is like throwing with dice: scheduling issues and compiler issues (whichever comes first)…
- But once a configuration has proven to run, it *always* runs

- All done with XMOS xTIMEcomposer 14.4.1 and code running on an xCORE-200 eXplorer board
- This lecture in no way pretends to be a textbook in xC.
  See **XMOS Programming Guide** (2015/9/21 version F, 2015/9/18 in the document),
  see https://www.xmos.ai/file/xmos-programming-guide
- Alternatively, go to the References section of my note 141

✅

For fine text or {fine code;}

View presentation in full resolution at

https://www.teigfam.net/oyvind/home/technology/218-ieee-copa-2021-fringe/

plus: download of code and extras

Normal task

```
#include <platform.h> // core
#include <timer.h>    // delay_milliseconds(..), XS1_TIMER_HZ etc

// The simplest form of interface, no use of
// [[guarded]], [[notification]] slave, or [[clears_notification]]
// used for session type interfaces
//
typedef interface con_if_t {
    unsigned set_get (const unsigned value);
} con_if_t;

// Like occam and go tasks. One per xCORE logical core.
// Expensive, so is most often a much more complex tast or one
// with very hard timing requirements
//
void my_normal_task (
        server  con_if_t    i_con,
        chanend             c_chan_in, // Used unidirectionally here
        in buffered port:32 p_pin) {

    timer    tmr; // one tick every 10 ns
    signed   time_ticks;
    unsigned my_value = 1;
    unsigned c_value_in;
    unsigned pin_val;

    p_pin :> pin_val; // read pin
    tmr :> time_ticks;
    time_ticks += XS1_TIMER_HZ;

    while (1) { // looping not required
        select {
            case tmr when timerafter (time_ticks) :> void : {
                my_value = -my_value;
            } break;
            case p_pin when pinsneq (pin_val) :> pin_val: {
                // change on pin
                my_value = 0;
            } break;
            case c_chan_in :> c_value_in : {
                my_value = c_value_in;
            } break;
            case i_con.set_get (const unsigned value_in) ->
                    unsigned value_return : {
                value_return = my_value;
                my_value = value_in;
            } break;
        }
        my_value++; // As function «calculate» for the others
                    // Shared state may be handled here
    }
}
```

Combinable task

```
// Instead if the common state my_value++ above,
// which is not allowed. For the same semantics,
// one call for each select case
//
unsigned calculate (const unsigned value) {
    return (value+1);
}

 // Several of these may be combined in one select by the compiler
 // First level of restrictions apply
 //
[[combinable]]
void my_combinable_task (
        client  con_if_t i_con_0,
        client  con_if_t i_con_1,
        chanend          c_chan_out, // Used unidirectionally here
        in port          p_pin) {

    timer    tmr; // one tick every 10 ns
    signed   time_ticks;
    unsigned my_value = 1;
    unsigned pin_val;

    p_pin :> pin_val; // read pin
    tmr :> time_ticks;
    time_ticks += XS1_TIMER_HZ;

    while (1) { // looping required
        select {
            case tmr when timerafter (time_ticks) :> void : {
                my_value = -my_value;
                my_value = calculate (my_value);
                my_value = i_con_0.set_get (my_value);
                c_chan_out <: my_value;
            } break;
            case p_pin when pinsneq (pin_val) :> pin_val: {
                // change on pin
                my_value = 0;
                my_value = calculate (my_value);
                my_value = i_con_1.set_get (my_value);
            } break;
            // chanend ok, not used in example
            // Server interface ok, not used in example
        }
    }
}
```

Distributable task

```
// Called as inline function, on the caller's stack
// Third level of resitrictions apply
// When DISTRIBUTABLE only interfaces allowed as communication params
// No side effects with system, like timers, channels or ports
//
[[distributable]]
void my_distributable_task (server con_if_t i_con) {

    unsigned my_value = 1; // Only  interface

    while (1) { // looping required
        select {
            case i_con.set_get (const unsigned value_in) ->
                    unsigned value_return : {
                value_return = my_value;
                my_value = value_in;
                my_value = calculate (my_value);
            } break;
        }
    }
}
```

```
in  buffered port:32 p_pin_b = on tile[0]: XS1_PORT_1M;
in  port             p_pin   = on tile[0]: XS1_PORT_1N;
int main (void) {
    con_if_t i_con_a;
    con_if_t i_con_b;
    chan c_chan;
    par {
        my_normal_task (i_con_a, c_chan, p_pin_b);
        [[combine]]
        par {
            my_combinable_task (
                i_con_a, i_con_b, c_chan, p_pin);
        }
        [[distribute]]
        my_distributable_task (i_con_b);
    }
   return 0;
}
```

Normal tasks can also contain composite par

This example code shows task types,
the code examples are not related to the theme of this presentation

```
Constraint check for tile[0]:
  Cores available:            8,   used:          2 .  OKAY
  Timers available:          10,   used:          2 .  OKAY
  Chanends available:        32,   used:          4 .  OKAY
  Memory available:      262144,   used:       2256 .  OKAY
    (Stack: 476, Code: 1570, Data: 210)
Constraints checks PASSED.
```

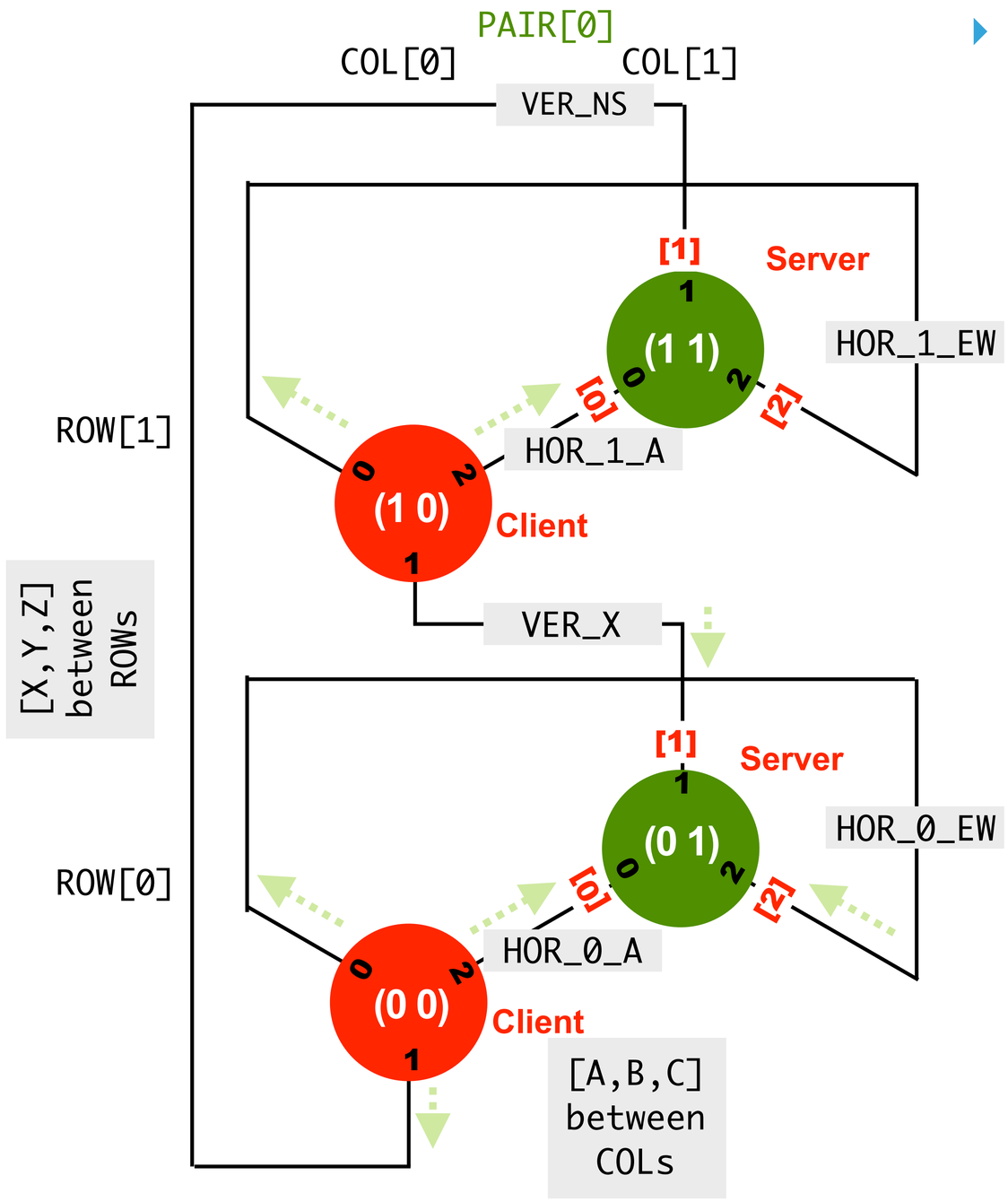
✅

ATTEMPT AT AN ANALYSIS

- Even if XMOS probably introduced these task types to make better use of
- HW cores and and f.ex. HW chanends,
- the user was given several types of task abstractions. Nice, but confusing at first
- It then also probably is a selling point for the xCORE multicore processors
- It’s not «only» 8 logical cores per tile, it’s a useful task abstraction that takes us in the direction of FPGA with VHDL or Verilog, but we instead have these independent hardware modules called «logical cores»
- Plus, it does in fact let us do the toroid multi-node research
- And control my aquarium’s temperature and LED light, etc. (here)
- These xC task types are rather generic and could with advantage have been introduced to any real-time, scheduler or operating system

✅

CHANNELS ARE OBVIOUSLY TOO EXPENSIVE FOR OUR CASE



- We are now leaving chan and chanend since they are too expensive. We can express 4*4 nodes, but not larger topologies

- In order to cope with this limitation, xC has
  - The *combinable* and *distributable* tasks
  - Use of *interfaces* as protocol / session primitives, as already seen examles of
  - Often *both channels and interface* usage needed
- Use of boolean expressions in the `select()`-guarded commands

✅

NO SLIDING BEHAVIOUR ACCEPTED: REPEATABLE CONVERGING VALUE!

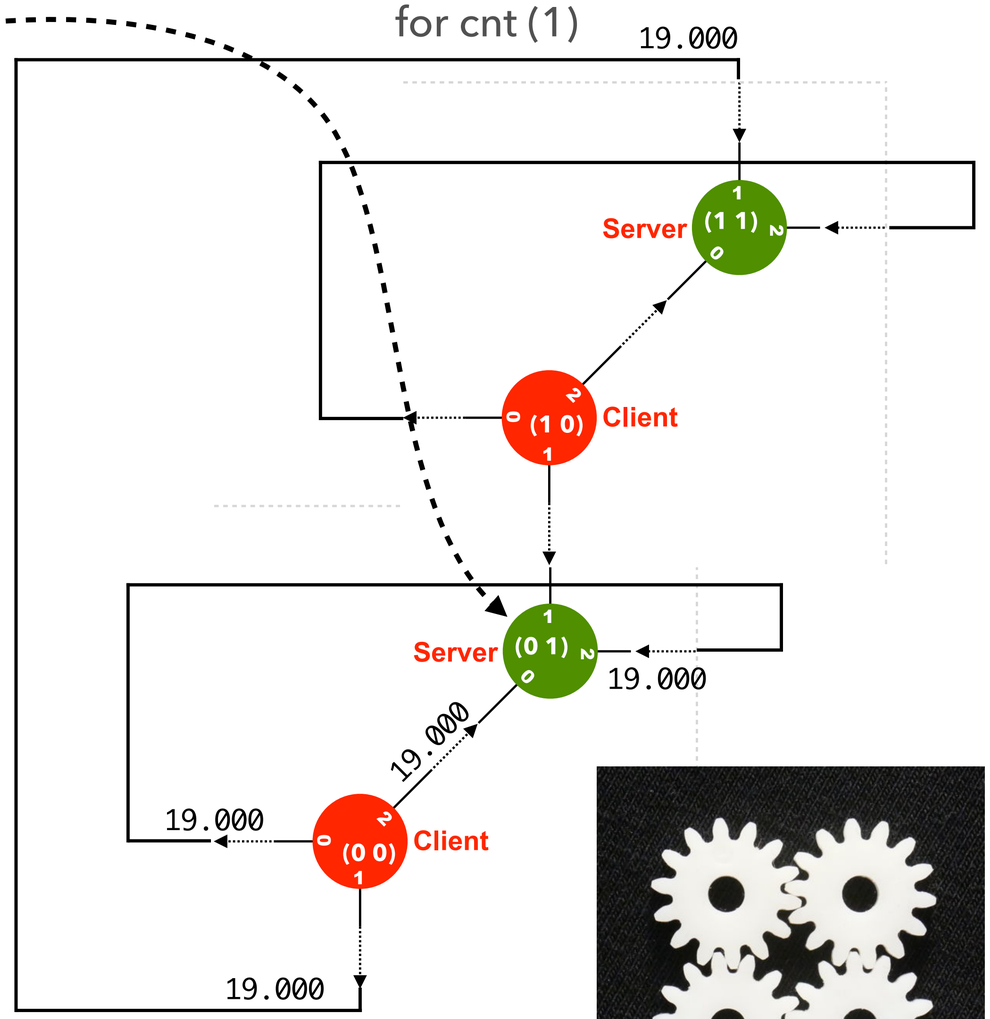
```
(0 1) S(0) [0.0000] "Server_node_task"
(1 1) S(0) [0.0000] "Server_node_task"
(1 0) C(0) [0.0000] "Client_node_task"
(0 0) C(0) [19.0000] "Client_node_task root" as "Top_26 2*2" on Mar 24 2021 18:14:51
```

All values must pass from cycle_cnt (0) to (1) to (n+1), no sliding
No sliding, but internal natural «strectching» may happen

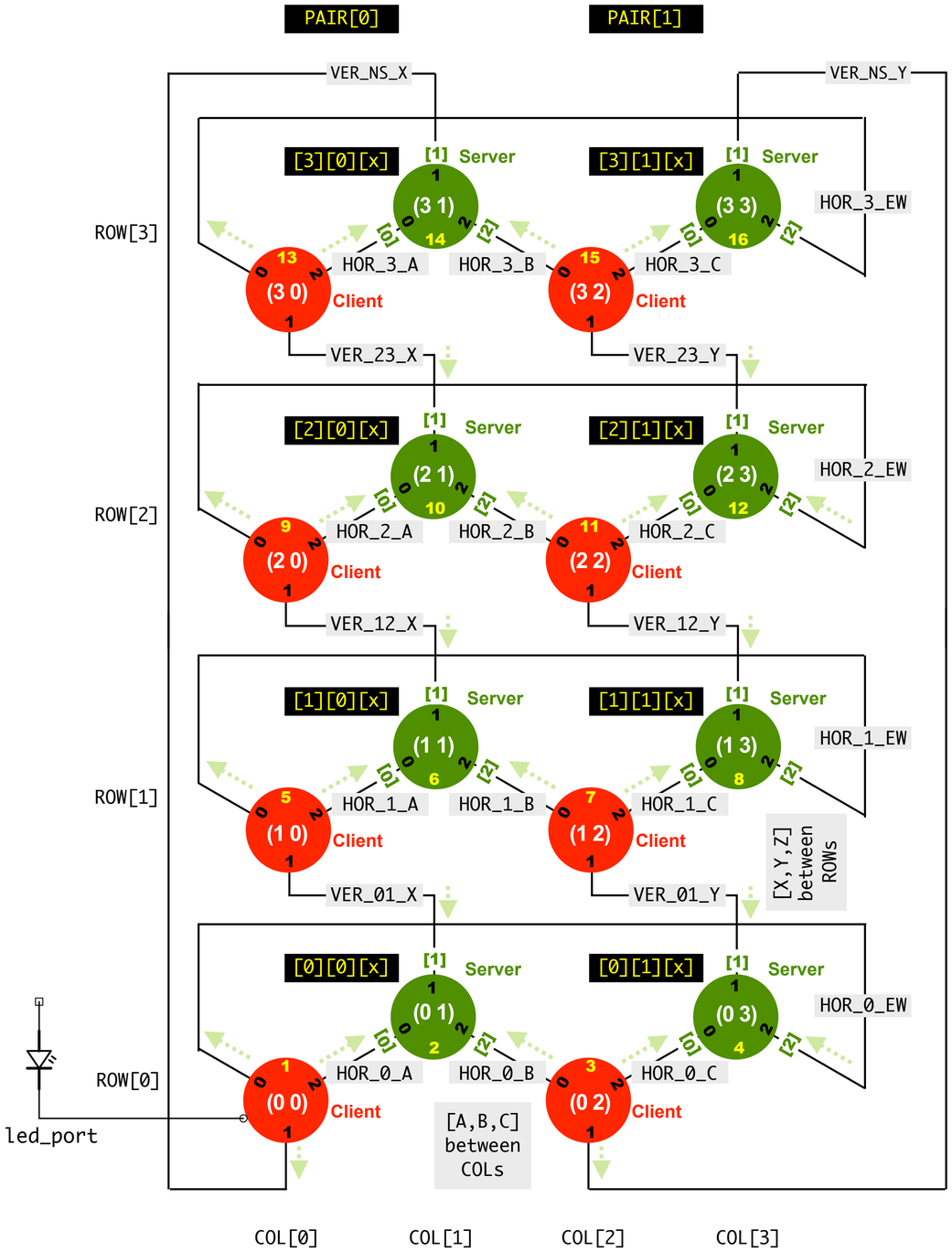
```
(1 0) C(1) [0.00000000] from [0.0000][0.0000][0.0000]
(1 1) S(1) [0.47499999] from [0.0000][19.0000][0.0000]
(0 1) S(1) [2.37500000] from [19.0000][0.0000][19.0000]
(0 0) C(1) [18.54399872] from [0.0000][0.0000][0.0000] at 804 ms

(1 0):C(2) [0.09500000] from [0.4750][2.3750][0.4750]
(1 1):S(2) [0.86734998] from [0.0000][18.5440][0.0000]
(0 1):S(2) [4.33675003] from [18.5440][0.0000][18.5440]
(0 0):C(2) [18.14834213] from [2.3750][0.4750][2.3750] at 1905 ms

(1 0):C(3) [0.25707000] from [0.8673][4.3368][0.8673]
(1 1):S(3) [1.20283103] from [0.0950][18.1483][0.0950]
(0 1):S(3) [5.95715523] from [18.1483][0.0950][18.1483]
(0 0):C(3) [17.80298615] from [4.3368][0.8673][4.3368] at 3006 ms
. . .
(0 0):C(464) [12.50002575] from [12.5000][12.5000][12.5000] at 380
(1 0):C(465) [12.49995041] from [12.5000][12.5000][12.5000]
(1 1):S(465) [12.49995708] from [12.4999][12.5000][12.4999]
(0 1):S(465) [12.50000763] from [12.5000][12.4999][12.5000]

(0 0) C(465) [12.50002575] from [12.5000][12.5000][12.5000] at 391
(1 0) C(466) [12.49995136] from [12.5000][12.5000][12.5000]
(1 1) S(466) [12.49995804] from [12.5000][12.5000][12.5000]
(0 1) S(466) [12.50000858] from [12.5000][12.5000][12.5000]
```

Converging value 12.5000

19.000 on link 0 and 2 in cnt (0)
for cnt (1)



Every time!



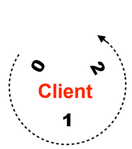
connections [ROW][PAIR][PARAM]

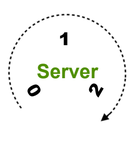
NUM 1 -16

COORD (0 0) - (3 3)

PARAM 0, 1, 2

Param 0 LEFT
Param 1 VERTICAL
Param 2 RIGHT





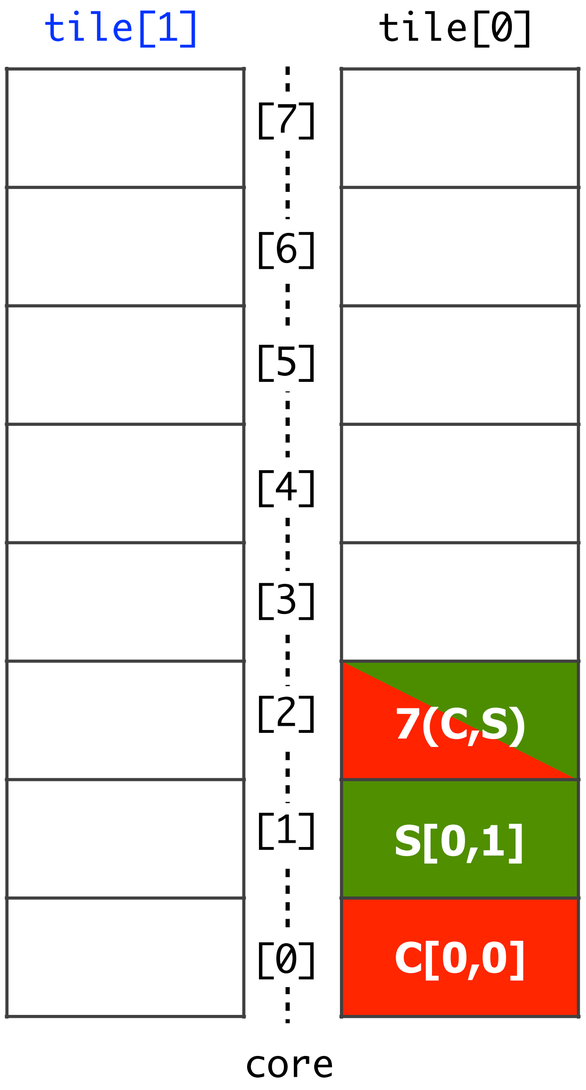
Client (params 0,1,2) **Server (params 0,1,2)**

TOPOLOGY 4 * 4 nodes
"2021 03 17 TOP_25.pages"

```
interface conn_if_t conn [NUM_ROWS][NUM_PAIRS_PER_ROW][NUM_CONNS_PER_NODE];
//                       [    4   ][         2       ][        3         ]

#define VER_NS_Y [3][1][1] // NS      North-South top-bottom
#define VER_NS_X [3][0][1] // NS      North-South top-bottom

#define HOR_3_EW [3][1][2] // EW      EastWest belt side-to-side
#define HOR_3_C  [3][1][0] // [A,B,C] Between horisontal columns
#define HOR_3_B  [3][0][2] // [A,B,C] Between horisontal columns
#define HOR_3_A  [3][0][0] // [A,B,C] Between horisontal columns

#define VER_23_Y [2][1][1] // [X,Y,Z] Between vertical rows
#define VER_23_X [2][0][1] // [X,Y,Z] Between vertical rows
#define HOR_2_EW [2][1][2] // EW      EastWest belt side-to-side
#define HOR_2_C  [2][1][0] // [A,B,C] Between horisontal columns
#define HOR_2_B  [2][0][2] // [A,B,C] Between horisontal columns
#define HOR_2_A  [2][0][0] // [A,B,C] Between horisontal columns

#define VER_12_Y [1][1][1] // [X,Y,Z] Between vertical rows
#define VER_12_X [1][0][1] // [X,Y,Z] Between vertical rows
#define HOR_1_EW [1][1][2] // EW      EastWest belt side-to-side
#define HOR_1_C  [1][1][0] // [A,B,C] Between horisontal columns
#define HOR_1_B  [1][0][2] // [A,B,C] Between horisontal columns
#define HOR_1_A  [1][0][0] // [A,B,C] Between horisontal columns

#define VER_01_Y [0][1][1] // [X,Y,Z] Between vertical rows
#define VER_01_X [0][0][1] // [X,Y,Z] Between vertical rows
#define HOR_0_EW [0][1][2] // EW      EastWest belt side-to-side
#define HOR_0_C  [0][1][0] // [A,B,C] Between horisontal columns
#define HOR_0_B  [0][0][2] // [A,B,C] Between horisontal columns
#define HOR_0_A  [0][0][0] // [A,B,C] Between horisontal columns

par {
    on tile[0]:
    par {
        Client_node_task (0,0, conn HOR_0_EW, conn VER_NS_X, conn HOR_0_A, led_port );
        Server_node_task (0,1, conn HOR_0_A,  conn VER_01_X, conn HOR_0_B);
    }
    on tile[0]:
    [[combine]]
    par {
        Client_node_task (0,2, conn HOR_0_B,  conn VER_NS_Y, conn HOR_0_C, null);
        Server_node_task (0,3, conn HOR_0_C,  conn VER_01_Y, conn HOR_0_EW);

        Client_node_task (1,0, conn HOR_1_EW, conn VER_01_X, conn HOR_1_A, null);
        Server_node_task (1,1, conn HOR_1_A,  conn VER_12_X, conn HOR_1_B);
        Client_node_task (1,2, conn HOR_1_B,  conn VER_01_Y, conn HOR_1_C, null);
        Server_node_task (1,3, conn HOR_1_C,  conn VER_12_Y, conn HOR_1_EW);

        Client_node_task (2,0, conn HOR_2_EW, conn VER_12_X, conn HOR_2_A, null);
        Server_node_task (2,1, conn HOR_2_A,  conn VER_23_X, conn HOR_2_B);
        Client_node_task (2,2, conn HOR_2_B,  conn VER_12_Y, conn HOR_2_C, null);
        Server_node_task (2,3, conn HOR_2_C,  conn VER_23_Y, conn HOR_2_EW);

        Client_node_task (3,0, conn HOR_3_EW, conn VER_23_X, conn HOR_3_A, null);
        Server_node_task (3,1, conn HOR_3_A,  conn VER_NS_X, conn HOR_3_B);
        Client_node_task (3,2, conn HOR_3_B,  conn VER_23_Y, conn HOR_3_C, null);
        Server_node_task (3,3, conn HOR_3_C,  conn VER_NS_Y, conn HOR_3_EW);
    }
}
```

par (unsigned ix = 0; ix < 14; ix++) is another construct

THAT'S BETTER!




```
par
    on tile[0]:
    par 2N
        Client core[0]
        Server core[1]
    [[combine]]
    on tile[0]:
    par 14CN core[2]
        Client
        Server
        Client
        Server
        Client
        Server
        Client
        Server
        Client
        Server
        Client
        Server
        Client
        Server
```

Fastest round trip, with all communications and calculations is 710 µs (where waiting in root must be 300 µs)

Done on each node after three comms:

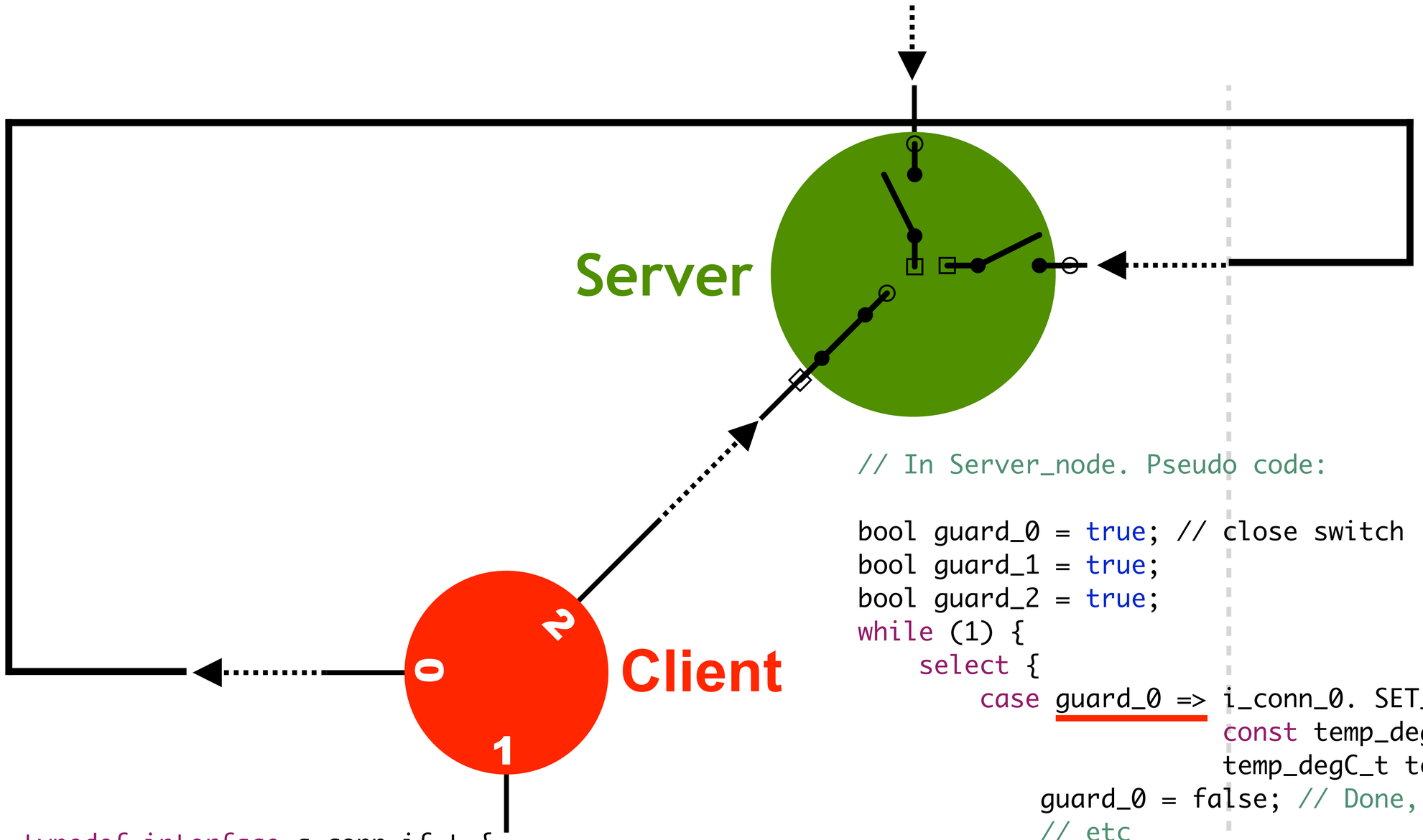
```
typedef float temp_degC_t;

temp_degC_t calculate_new_temp_from_flow (
        const node_context_t context,
        const temp_degC_t    tempold) {

    const temp_degC_t term1 = context.wt[0] *
          (context.temps_degC[0]-tempold);
    const temp_degC_t term2 = context.wt[1] *
          (context.temps_degC[1]-tempold);
    const temp_degC_t term3 = context.wt[2] *
          (context.temps_degC[2]-tempold);

    return (tempold + term1 + term2 + term3;);
}
```

PS. float handling is through a library

If no use of combinable and placing on core 0-15 the fastest round trip is 28 µs with zero waiting in the root. Possible, since 16 nodes/tasks

✅

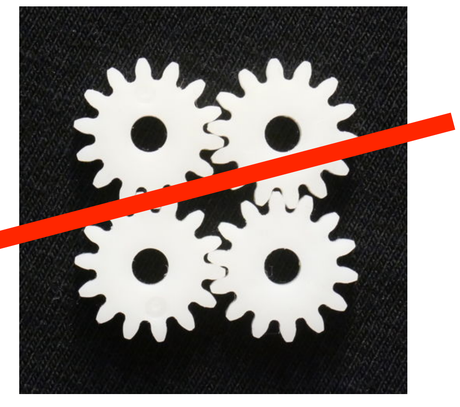
EXPANDING BEYOND 4*4 WITH GUARDED INTERFACE PROTOCOL FAILS



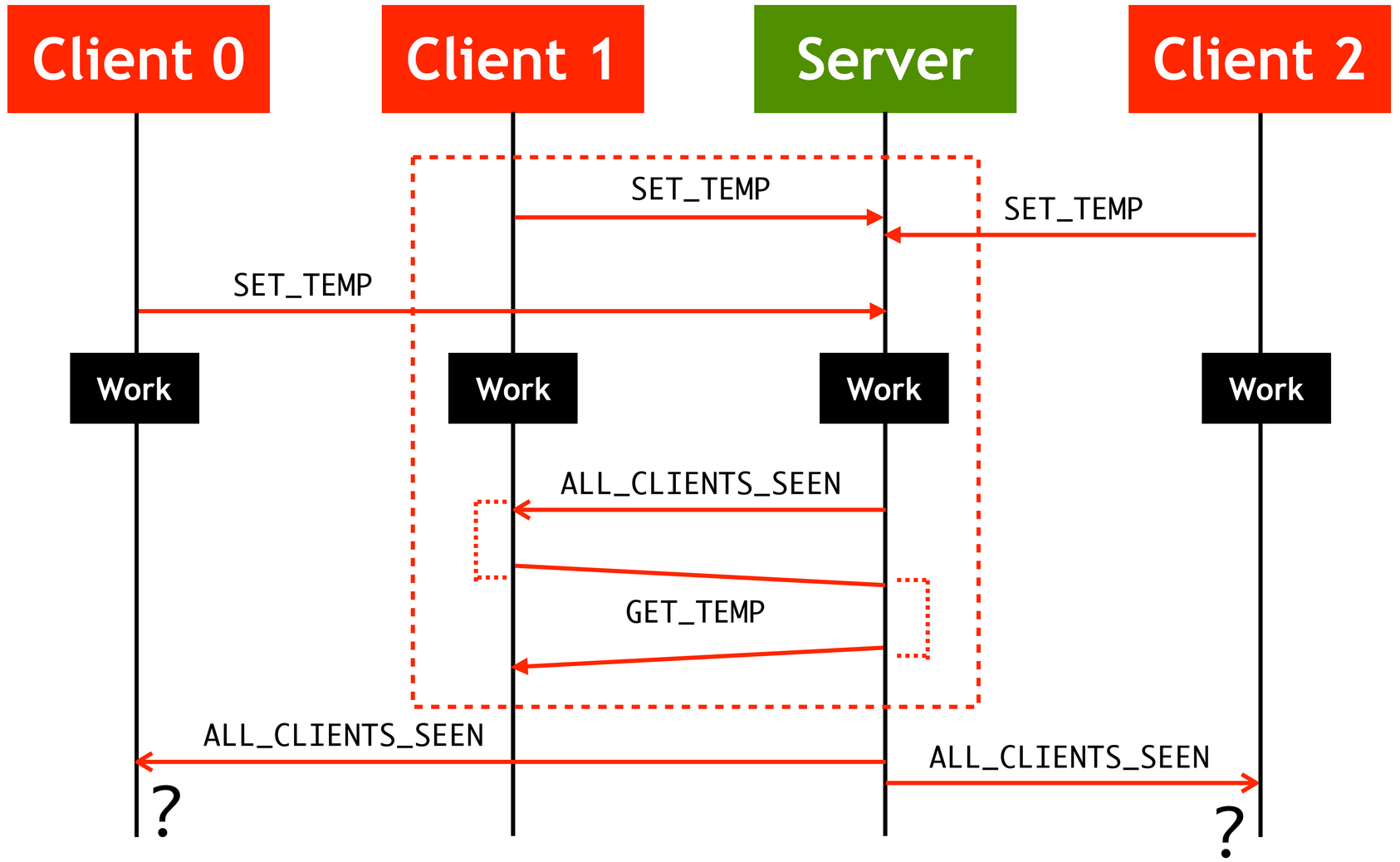

```
typedef interface g_conn_if_t {
    [[guarded]] temp_degC_t SET_GET_TEMP (
                const temp_degC_t temp_degC);
} g_con_if_t;
```

```
// In Server_node. Pseudo code:

bool guard_0 = true; // close switch
bool guard_1 = true;
bool guard_2 = true;
while (1) {
    select {
        case guard_0 => i_conn_0. SET_GET_TEMP (
                        const temp_degC_t temp_degC) ->
                        temp_degC_t temp_degC_ret : {
            guard_0 = false; // Done, open switch
            // etc
        } break;
        // case 1 and 2
    }
}
```

```
xcc1: internal compiler error
Failed in /jenkins/RELEASE_14_4/sb/tools_xcc1_c_llvm/FrontEnd/Lowering/
  lower_combined_pars.cpp, line 183
  info->stateObj
For bug reporting instructions, please see:
http://www.xmos.com/support
```

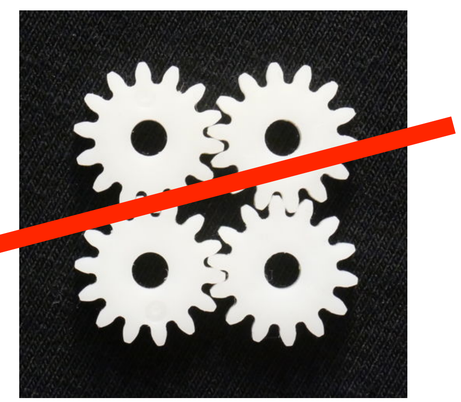


EXPANDING BEYOND 4*4 WITH ASYNCH INTERFACE PROTOCOL FAILS




```
typedef float temp_degC_t;
typedef interface a_con_if_t {
                              void        SET_TEMP          (const temp_degC_t temp_degC_in);
    [[notification]] slave    void        ALL_CLIENTS_SEEN  (void);
    [[clears_notification]]   temp_degC_t GET_TEMP          (void);
} a_con_if_t;
```

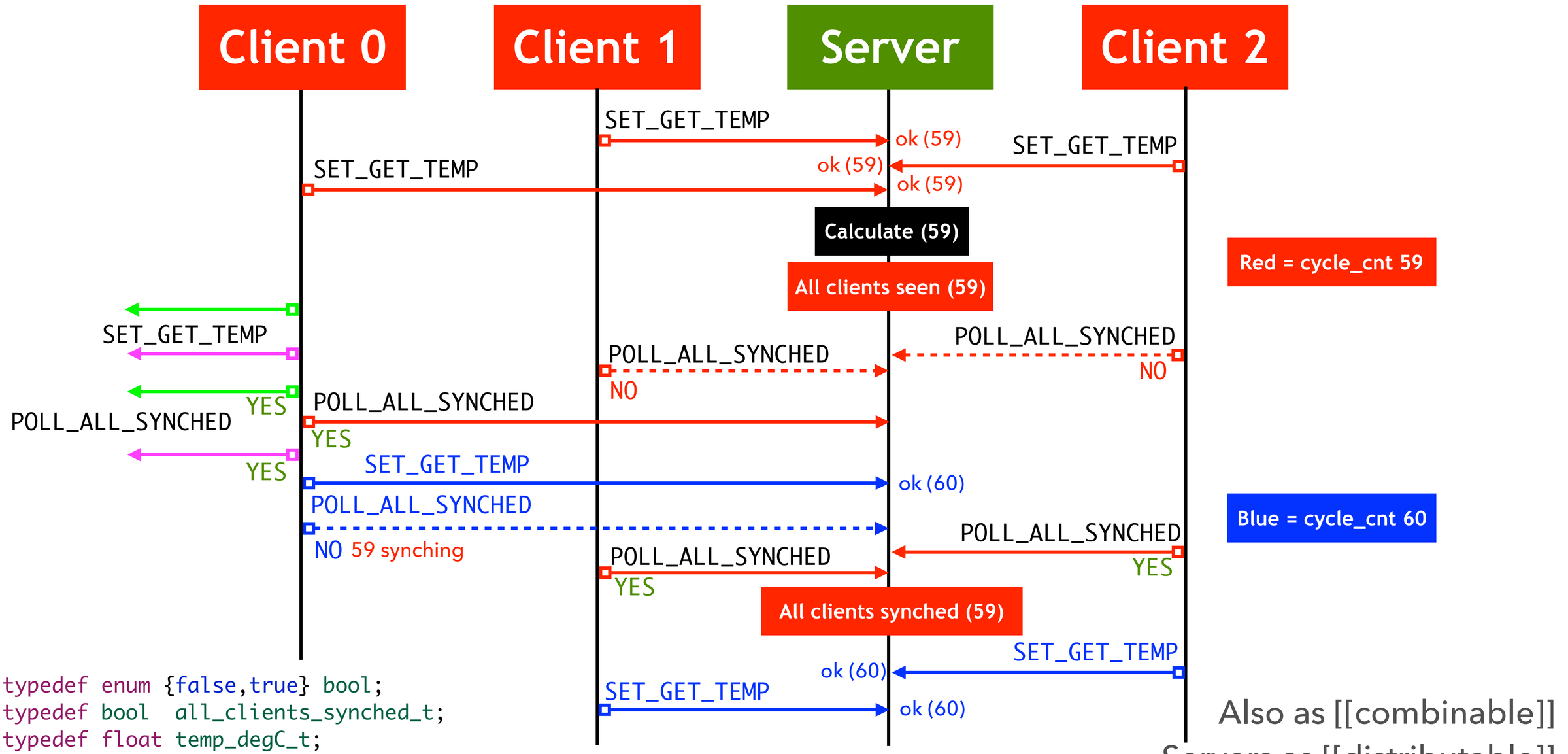
See blog 217



Builds, but does not seem to run correctly

I assume the scheme is made for a client at a time in server

EXPANDING BEYOND 4*4 WITH POLLING SYNCH PHASE SUCCEEDS




```
typedef enum {false,true} bool;
typedef bool  all_clients_synched_t;
typedef float temp_degC_t;

typedef interface conn_if_t {
    temp_degC_t            SET_GET_TEMP     (const temp_degC_t temp_degC);
    all_clients_synched_t POLL_ALL_SYNCHED (void);
} conn_if_t;
```

Also as [[combinable]]
Servers as [[distributable]]

Builds and runs, but polling is often
a waste of cycles, not always as ok as here,
since the root is supposed to «wait» anyhow,
plus the latest finished node decides the speed

✅

The code is not as bad as the message sequence diagram (MSD)

Client

```
while (1) { // In Client_node_task
    select {
        case tmr when timerafter (time_ticks) :> void : {
            if (client_state == CS_SEND_DO_IO_SERVER) {
                switch (iof_client) { // First this..
                    case 0: {
                        context.temps_degC[0] = i_conn_0.SET_GET_TEMP (temp_degC);
                    } break;
                    case 1: {
                        context.temps_degC[1] = i_conn_1.SET_GET_TEMP (temp_degC);
                    } break;
                    case 2: {
                        context.temps_degC[2] = i_conn_2.SET_GET_TEMP (temp_degC);
                    } break;
                }
                iof_client++; // ..then this
                if (iof_client == NUM_CONNS_PER_NODE) {
                    iof_client = 0;
                    temp_degC = calculate_new_temp_from_flow (context, temp_degC);
                    context.cycle_cnt++; // at client NUM_CONNS_PER_NODE seen
                    client_state = CS_SYNCHRONIZE;
                    tmr :> time_ticks;
                    time_ticks += CLIENT_DELAY_UNTIL_NEXT_SYNCH_POLL_US * XS1_TIMER_MHZ;
                } else { //doing next SET_GET_TEMP
                    tmr :> time_ticks;
                    time_ticks += CLIENT_DELAY_UNTIL_NEXT_THIRD_US * XS1_TIMER_MHZ;
                }
            } else if (client_state == CS_SYNCHRONIZE) {
                if (not all_clients_seen[0]) {
                    all_clients_seen[0] = i_conn_0.POLL_ALL_SYNCHED ();
                } else {}
                if (not all_clients_seen[1]) {
                    all_clients_seen[1] = i_conn_1.POLL_ALL_SYNCHED ();
                } else {}
                if (not all_clients_seen[2]) {
                     all_clients_seen[2] = i_conn_2.POLL_ALL_SYNCHED ();
                } else {}
                if (all_clients_seen[0] and all_clients_seen[1] and all_clients_seen[2]) {
                    for (unsigned ix=0; ix<NUM_CONNS_PER_NODE; ix++) {
                        all_clients_seen[ix] = false;
                    }
                    iof_client = 0;
                    client_state = CS_SEND_DO_IO_SERVER;
                    tmr :> time_ticks;
                    time_ticks += (ROOT_WAIT_FOR_NEXT_ROUND_US * (unsigned) root);
                } else {
                    tmr :> time_ticks;
                    time_ticks += (CLIENT_DELAY_UNTIL_NEXT_SYNCH_POLL_US * XS1_TIMER_MHZ);
                }
            }
        } break;
    }
}
```

DATA

Synchronize

Server

```
while (1) { // In Server_node_task
    select {
        case i_conn_0.SET_GET_TEMP (const temp_degC_t temp_degC_in) -> temp_degC_t temp_degC_return : {
            temp_degC_return = temp_degC; // always return old value
            synch.client_tagged_key[0] = context.cycle_cnt; // not from synch.key
            context.temps_degC[0] = temp_degC_in;
            temp_degC = Handle_server_set_get (context, synch, temp_degC);
        } break;
        case i_conn_1.SET_GET_TEMP (const temp_degC_t temp_degC_in) -> temp_degC_t temp_degC_return : {
            temp_degC_return = temp_degC; // always return old value
            synch.client_tagged_key[1] = context.cycle_cnt; // not from synch.key
            context.temps_degC[1] = temp_degC_in;
            temp_degC = Handle_server_set_get (context, synch, temp_degC);
        } break;
        case i_conn_2.SET_GET_TEMP (const temp_degC_t temp_degC_in) -> temp_degC_t temp_degC_return : {
            temp_degC_return = temp_degC; // always return old value
            synch.client_tagged_key[2] = context.cycle_cnt; // not from synch.key
            context.temps_degC[2] = temp_degC_in;
            temp_degC = Handle_server_set_get (context, synch, temp_degC);
        } break;
        case i_conn_0.POLL_ALL_SYNCHED () ->
            all_clients_synched_t all_clients_seen_return  : {
                all_clients_seen_return =
                    Handle_server_poll_all_clients_seen (0, context, synch);
        } break;
        case i_conn_1.POLL_ALL_SYNCHED () ->
            all_clients_synched_t all_clients_seen_return  : {
                all_clients_seen_return =
                    Handle_server_poll_all_clients_seen (1, context, synch);
        } break;
        case i_conn_2.POLL_ALL_SYNCHED () ->
            all_clients_synched_t all_clients_seen_return  : {
                all_clients_seen_return =
                    Handle_server_poll_all_clients_seen (2, context, synch);
        } break;
    }
}
```

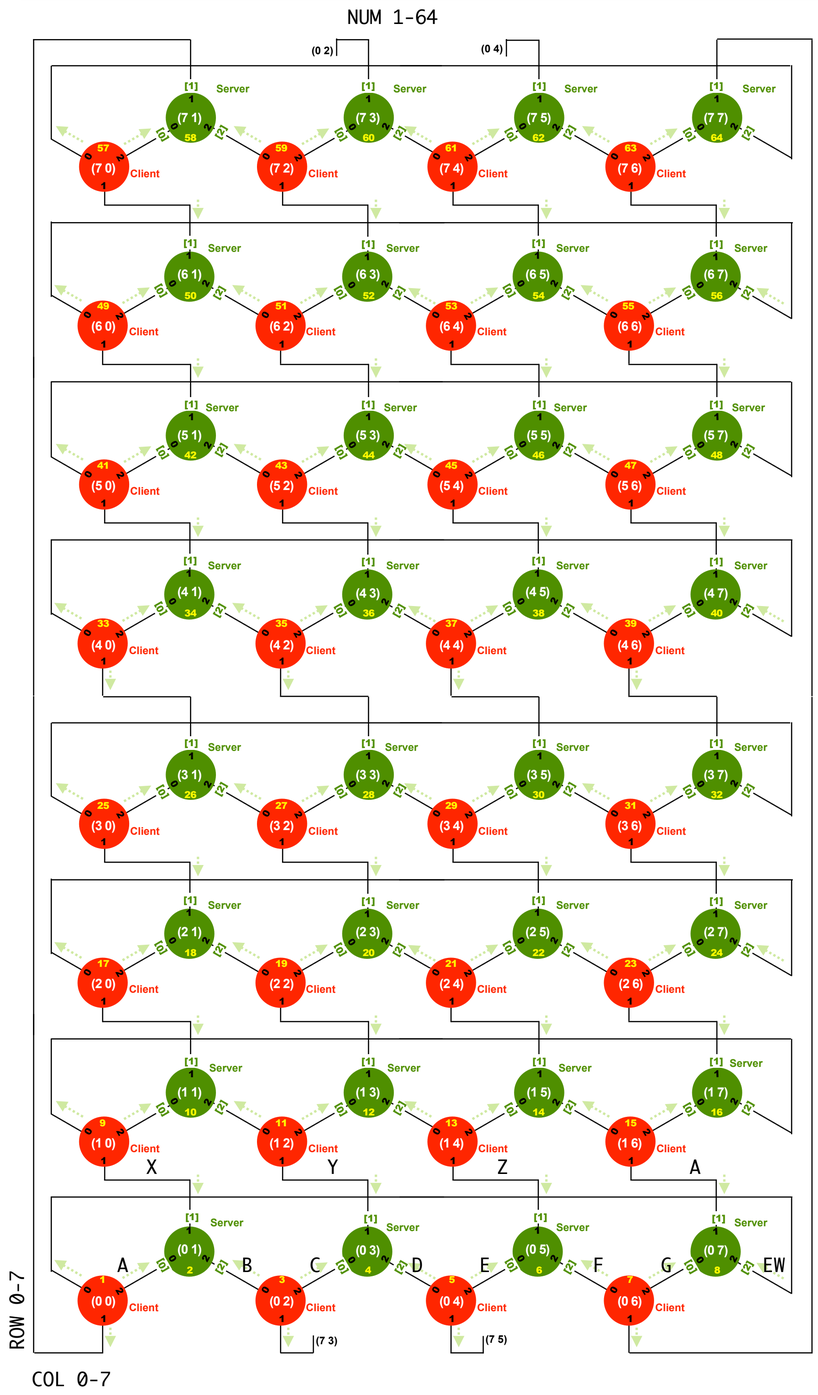
Client with its three servers | Server with its three clients

No operating system!

Just xC and the HW on the xCORE

✅

8*8 NODES AS DISTRIBUTABLE (WITH THE SAME CODE AS JUST SHOWN)




```
Constraint check for tile[0]:
  Cores available:            8,   used:          3 .  OKAY
  Timers available:          10,   used:          3 .  OKAY
  Chanends available:        32,   used:          1 .  OKAY
  Memory available:      262144,   used:      94120 .  OKAY
    (Stack: 49796, Code: 40084, Data: 4240)
Constraints checks PASSED.
```

```
(2 2):C(1002) [50.00000000] from [18.0000][12.0000][20.0000]
(5 7):S(1002) [90316.00000000] from [407.0000][89524.0000][385.0000]
(4 2):C(1003) [91124.00000000] from [891.0000][406.0000][89827.0000]
(6 6):C(1002) [87689.00000000] from [54.0000][87579.0000][56.0000]
(4 4):C(1003) [37.00000000] from [89827.0000][432.0000][89865.0000]
(4 6):C(1003) [91232.00000000] from [89865.0000][434.0000][933.0000]
(0 7):S(1000) [88248.00000000] from [88232.0000][15.0000][1.0000]
(7 1):S(1000) [88522.00000000] from [88462.0000][1.0000][59.0000]
(0 1):S(1000) [13.00000000] from [1.0000][9.0000][3.0000]

  (0 7):S(999) [88882.00000000] from [7.0000][88874.0000][1.0000]
  (7 1):S(999) [6155.00000000] from [3090.0000][1.0000][3064.0000]
  (0 1):S(999) [83444.00000000] from [1.0000][83440.0000][3.0000]

(0 0):C(1000) [1.00000000] from [88882.0000][6155.0000][83444.0000] at 37849 ms
```

Values always pass from `cycle_cnt` (n) to (n+1) = no sliding

```
int main (void) {

interface conn_if_t conn [NUM_ROWS][NUM_PAIRS_PER_ROW][NUM_CONNS_PER_NODE];
//                       [    8   ][         4       ][        3         ]

#define VER_R_A(row)  [row][3][1] // [X,Y,Z,A] Between vertical rows
#define VER_R_Z(row)  [row][2][1] // [X,Y,Z,A] Between vertical rows
#define VER_R_Y(row)  [row][1][1] // [X,Y,Z,A] Between vertical rows
#define VER_R_X(row)  [row][0][1] // [X,Y,Z,A] Between vertical rows

#define HOR_R_EW(row) [row][3][2] // EW      EastWest belt side-to-side
#define HOR_R_G(row)  [row][3][0] // [A,B,C] Between horisontal columns
#define HOR_R_F(row)  [row][2][2] // [A,B,C] Between horisontal columns
#define HOR_R_E(row)  [row][2][0] // [A,B,C] Between horisontal columns
#define HOR_R_D(row)  [row][1][2] // [A,B,C] Between horisontal columns
#define HOR_R_C(row)  [row][1][0] // [A,B,C] Between horisontal columns
#define HOR_R_B(row)  [row][0][2] // [A,B,C] Between horisontal columns
#define HOR_R_A(row)  [row][0][0] // [A,B,C] Between horisontal columns

par {
    on tile[0]:
    par {
        Client_node_task (0,0, conn HOR_R_EW(0), conn VER_R_X(7), conn HOR_R_A(0), outP4_leds);
        Client_node_task (0,2, conn HOR_R_B(0),  conn VER_R_Y(7), conn HOR_R_C(0), null);
    }
    on tile[0]:
    [[combine]]
    par {
        Client_node_task (0,4, conn HOR_R_D(0),  conn VER_R_Z(7), conn HOR_R_E(0), null);
        Client_node_task (0,6, conn HOR_R_F(0),  conn VER_R_A(7), conn HOR_R_G(0), null);
        Client_node_task (1,0, conn HOR_R_EW(1), conn VER_R_X(0), conn HOR_R_A(1), null);
        Client_node_task (1,2, conn HOR_R_B(1),  conn VER_R_Y(0), conn HOR_R_C(1), null);
        Client_node_task (1,4, conn HOR_R_D(1),  conn VER_R_Z(0), conn HOR_R_E(1), null);
        // 24 clients removed here
        Client_node_task (7,6, conn HOR_R_F(7),  conn VER_R_A(6), conn HOR_R_G(7), null);
    }
    on tile[0]: [[distribute]] Server_node_task (0,1, conn HOR_R_A(0),  conn VER_R_X(0), conn HOR_R_B(0));
    on tile[0]: [[distribute]] Server_node_task (0,3, conn HOR_R_C(0),  conn VER_R_Y(0), conn HOR_R_D(0));
    on tile[0]: [[distribute]] Server_node_task (0,5, conn HOR_R_E(0),  conn VER_R_Z(0), conn HOR_R_F(0));
    on tile[0]: [[distribute]] Server_node_task (0,7, conn HOR_R_G(0),  conn VER_R_A(0), conn HOR_R_EW(0));
    // 26 servers removed here
    on tile[0]: [[distribute]] Server_node_task (7,5, conn HOR_R_E(7),  conn VER_R_Z(7), conn HOR_R_F(7));
    on tile[0]: [[distribute]] Server_node_task (7,7, conn HOR_R_G(7),  conn VER_R_A(7), conn HOR_R_EW(7));
}
```

Tasks may be 3 «ahead» or 3 «behind»:

- In the window between the first `cycle_cnt` (1000) and last (1000) (which is the root)
- we see nodes lagging behind in their calculations (997), (998) and (999)
- but also some being ahead (1001), (1002) and (1003)
- This reflects the 8*8 node's internal natural «stretch» of +/- 3
- However, no sliding of data!

EXPANDING BEYOND 8*8 WITH A MASTER «NEXT PHASE» TICKS?

- Since the simplest protocol `g_con_if_t` does not compile on the present xTIMEcomposer
- and thus may be (part of) the reason why xC `par` ~~seems~~/is "less scalable" than the occam `PAR` ..
  - ..communication "in par" in *composed* task with interface and `[[combine]] par` is not tried
- Maybe we could still do *without* the polling synch phase
- and have each *client* just "wait" for a «next» tick from a master timing node (ie. the root)?
- Is this a feasible way to *simulate* a *barrier*?

- What would the pros and cons be?
- I would have started with these interfaces…
- …the blog note contains a test of this, including some result

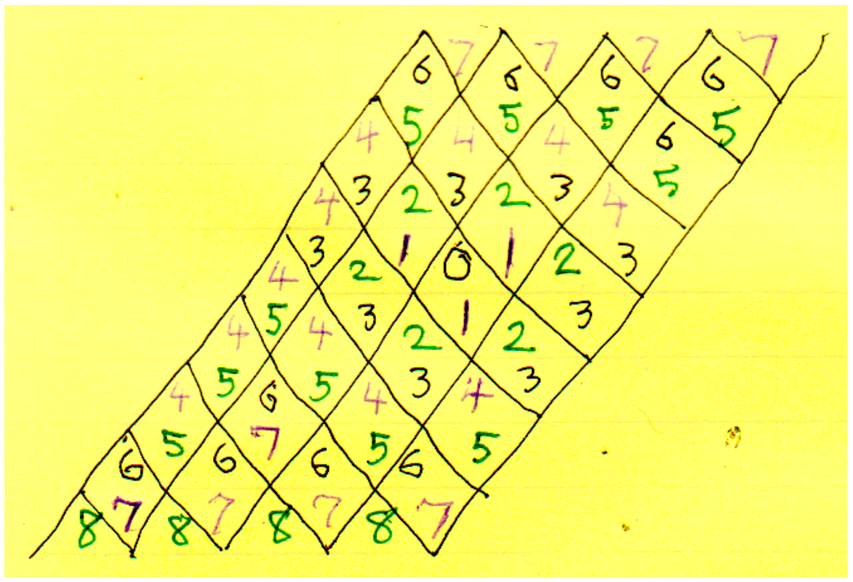
```
// SIMPLER BETWEEN ALL (no polling synch phase)
typedef interface conn_if_t {
    temp_degC_t SET_GET_TEMP (const temp_degC_t temp_degC);
} conn_if_t;

// FROM CLIENTS TO ROOT or SYNCH task:
typedef interface wait_if_t {
    void WAIT (const unsigned cycle_cnt); // cnt needed?
} wait_if_t;

// BACK TO CLIENTS AFTER SAME cycle_cnt FROM ALL:
typedef interface next_if_t {
    void NEXT (void);
} next_if_t;
```

SUMMARY

- We did a lot of hooping, and the hoop often fell to the ground
  - Compiler internal error. Crash in some «line 183» of an LLVM code piece. Waiting for a new xTIMEcomposer version (using 14.4.1)
  - Code that compiles but will not run (if all tasks are combined then the code may run to a point when progress fails.) I have not discovered how to stop this on the code. But if it stops, it stops fast. See blog note 217
  - Lots of solutions that we failed on (two shown here). Sadly, the simplest.. failed («guarded»)
- But xC is so multifaceted that we had some degree of success
  - Topology as 8 by 8 = 64 nodes was possible
  - «Lucky» that the servers could be distributable!
- Observe that xTIMEcomposer should be able to simulate all of this code (with the same timing and deterministic properties as the real HW), without any xCORE-200 eXplorer board. xTIMEcomposer is free to download
- Future work: other topologies, other sizes
- This was my last page, so thanks for listening!
- (As mentioned: code and presentation available at https://www.teigfam.net/oyvind/home/technology/218-ieee-copa-2021-fringe/)

✅

AVOIDING BARRIERS BUT STAYING VALID



Every communication must be between two nodes in the same cycle. (Proof by induction.) This is important - violates the conservation law otherwise - but surprisingly hard to enforce. Barriers would do it, but we deliberately want to avoid barriers, letting the nodes get a little, but not much, out of sync at the same time. The table on the left shows how many cycles it takes for synchrony to transmit.

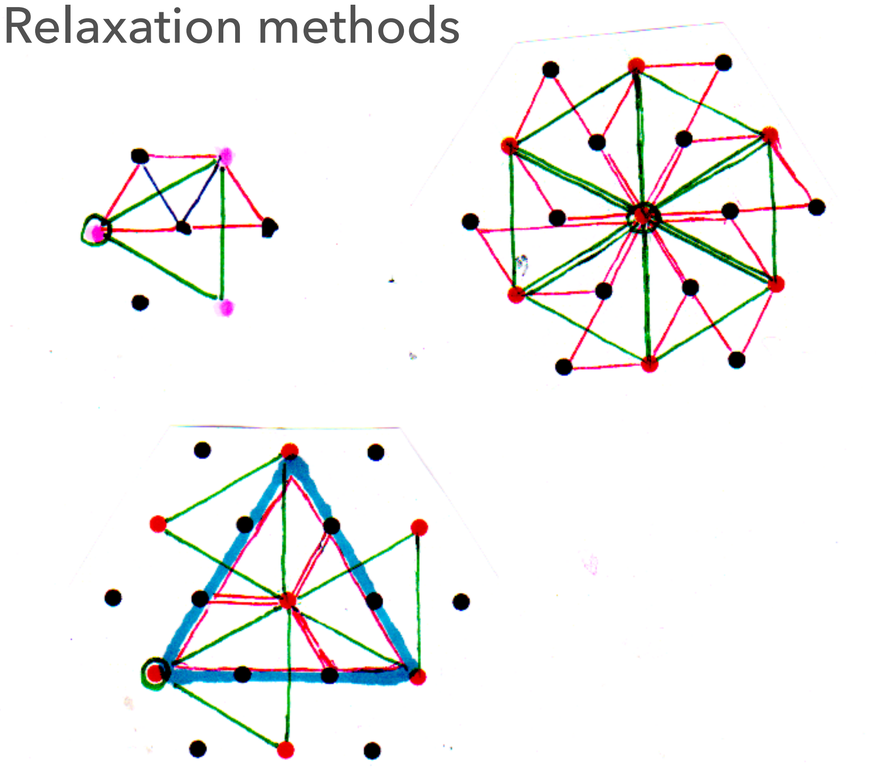
What is the advantage of the avoidance of barriers?

In massively parallel operations (like climate modeling), individual nodes sometimes become much slower than average. If barriers are used, this immediately slows down the entire program, perhaps thousands of nodes. In many cases, if the slowdowns are sporadic and happen at different nodes at different times, the occam-style loose linkage between different distant nodes will cause this effect to be vastly mitigated.

FUTURE WORK

(A) This approach is meant to emulate massively parallel hardware, of the GPU or AI type, that will be used on physically challenging problems like climate modeling.
We wish to extend it to:

- Nonlinear equations
- Non-euclidean geometries
- Relaxation methods



(B) We need to design data flow in and out while modeling is underway, while preserving massive parallelism. One of us (Larry) devised the “shuriken blade” approach to move data on a 9 - 3 - 1 tree (thus with a fanout of 4) while preserving its locality.
We need to:

- Actually implement this
- Use it for radiation and IO emulation

Important note: climate on a sphere does not need non-local radiation feedback. Torus is worse in this case (inner part radiates to distant nodes)

(C) We need to reach to vastly greater node count (e.g. from 64 to 576, from 16 to 144 or 1296, and beyond that to emulate 100,000 or more nodes, which we foresee in hardware). It does not matter if the emulation is slow.

- Improve number of soft nodes per hard node on the xC
- Connect multiple xC while retaining CSP purity, like Transputer links

FUTURE POSSIBILITIES

The items here are from a less urgent wish list: perhaps we can get along without them.

(I) One of us (Larry) would particularly like an occam 2 compiler, or an occam 2 translator to xC, so as to be able to simplify the steps to emulations that are not naturally client-server

- If local workspace for soft processes can be managed, it should be easy to simulate the Transputer model, which is of trivial complexity by modern standards
- Process queue delays can be kept predictable by the nature of emulations

(II) It would really be good if the occam compiler/translator could take advantage of the hardware advances of the xCore, such as its subdivision of tiles by staggered cycles

(III) As mentioned above, external links on the CSP model would open the door to really big emulations and other uses that stay strict in their adherence to CSP

(IV) We could branch out to hardware design and mapping to GPU/AI chips, and so apply the results of the emulations to real hardware