The *Regiment* *Macroprogramming* System

presented by Gidon Ernst

Ad-Hoc Networks Seminar

Index

- Introduction
- *Regiment*'s language
- Average temperature example
- Advanced Features of *Regions*
- *Regiment*'s compiler
- Evaluation

Introduction

- *Regiment* is a programming environment for sensor networks (like TinyOS)
- It consists of:
  - A specialized programming **language**
  - A **compiler** (and optimizer)

What was *Macroprogramming* again?

- Traditionally, you write node-local programs
- In *Macroprogramming*, you directly control **the network itself**
  - One global program
  - Nodes are represented as objects

Regiment

- The **language**
  - Signals & Regions
  - Transformations
- The **compiler**
  - Generates the local programs
  - *Reduction* and *normalization*
  - Intermediate representations

Language concept

- Signal: a sensor reading
  (or some other scalar value)

- Region: collection of signals (or regions)

- Node-level functions
- Network-level functions

Language overview

- Seen so far: ☎
  - `rmap`
  - `rfilter`
- Now introducing:
  - `smap`
  - `rfold`

Language: smap

- `smap(function, signal):`
  - Identical to: `function(value)`
  - Advantage: function can be factored out

```
fun to_celsius(t) { t*SCALING_FACTOR }

fun read_celsius(n) {
  smap(to_celsius, sense("temp", n))
}
```

Language: rfold

- `rfold(function, initial, region):`
  - Combine the region values to a single Signal
  - `initial` must be neutral wrt. `function`

    (might be used multiple times)

```
rfold( add, 0, temps )
```

temps ={ t1, t2, t3 }
→ add(0, add(t1, add(t2, add(t3)))
→ 0 + t1 + t2 + t3

Language: rfold

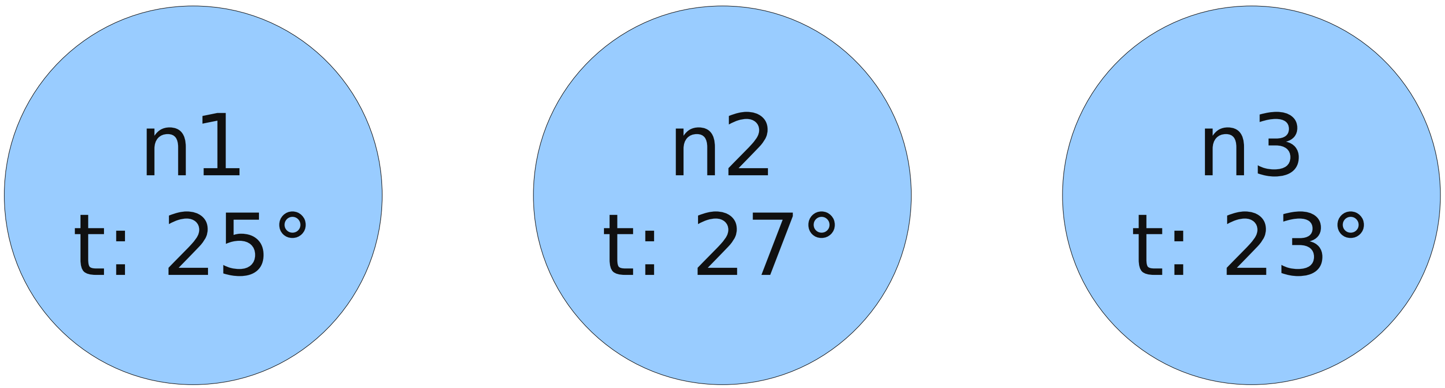
- `rfold(function, initial, region):`
  - Combine the region values to a single Signal
  - `initial` must be neutral wrt. `function`
    (might be used multiple times)

```
rfold( add, 0, temps )
```

temps ={ t1, t2, t3 }

$\rightarrow 0 + t1 + 0 + t2 + t3$

Example: average temperature



$$avg = \left(\sum_{i=1}^{N} sense\left(\text{"temp"}, n_i\right)\right)/N \qquad N=3$$

Example: average temperature

$$avg = \left(\sum_{i=1}^{N} sense(\text{"temp"}, n_i)\right) / N$$

1. Read all temperatures
2. Sum them up and count the nodes
3. Calculate the average
4. Transfer to base

Example: average temperature

$$avg = \left(\sum_{i=1}^{N} sense(\text{"temp"}, n_i)\right) / N$$

1. Read all temperatures: ☎
2. Sum them up and count the nodes
3. Calculate the average
4. Transfer to base

Example: average temperature

$$avg = \left(\sum_{i=1}^{N} sense(\text{"temp"}, n_i)\right) / N$$

1. Read all temperatures: `rmap`
2. Sum them up and count the nodes: ☎
3. Calculate the average
4. Transfer to base

Example: average temperature

$$avg = \left(\sum_{i=1}^{N} sense\left(\text{"temp"}, n_i\right)\right) / N$$

1. Read all temperatures: `rmap`
2. Sum them up: `rfold`
3. Calculate the average: ☎
4. Transfer to base

Example: average temperature

$$avg = (\sum_{i=1}^{N} sense(\text{"temp"}, n_i))/N$$

1. Read all temperatures: `rmap`
2. Sum them up: `rfold`
3. Calculate the average: `smap`
4. Transfer to base: ☏

Example: average temperature

1. Read all temperatures

```
fun read(n) { sense("temp", n) }

temps = rmap(read, world)
```

world={n1, n2, n3} → temps={25, 27, 23}

Example: average temperature

2. Sum them up and count the nodes

```
fun dosum(temp, (sum, cnt)) {
    (sum+temp, cnt+1)
}

sum_cnt = rfold(dosum, (0,0), temps)
```

temps = {25, 27, 23} → sum_cnt = (75, 3)

Example: average temperature

3. Calculate the average

```
fun doavg((sum, cnt)) {
    sum / cnt
}

avg = smap( doavg, sum_cnt )
```

sum_cnt = (75, 3) → avg = 25

Example: average temperature

4. Transfer the temperature to the base

```
BASE ← avg
```

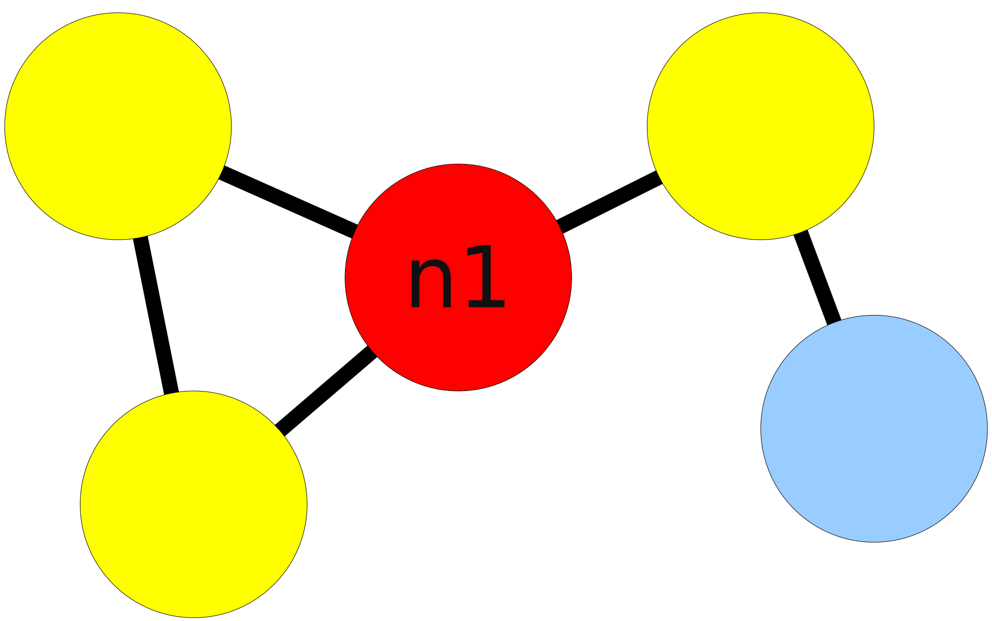
Advanced Region Features

- Seen so far:
  - `smap, rmap, rfilter, rfold`
- Now introducing:
  - `khood`
  - `table_gossip`

Language: khood

- `khood(d, n)`: d-hop neighbourhood of n

```
nodes = khood(1, n1)
```



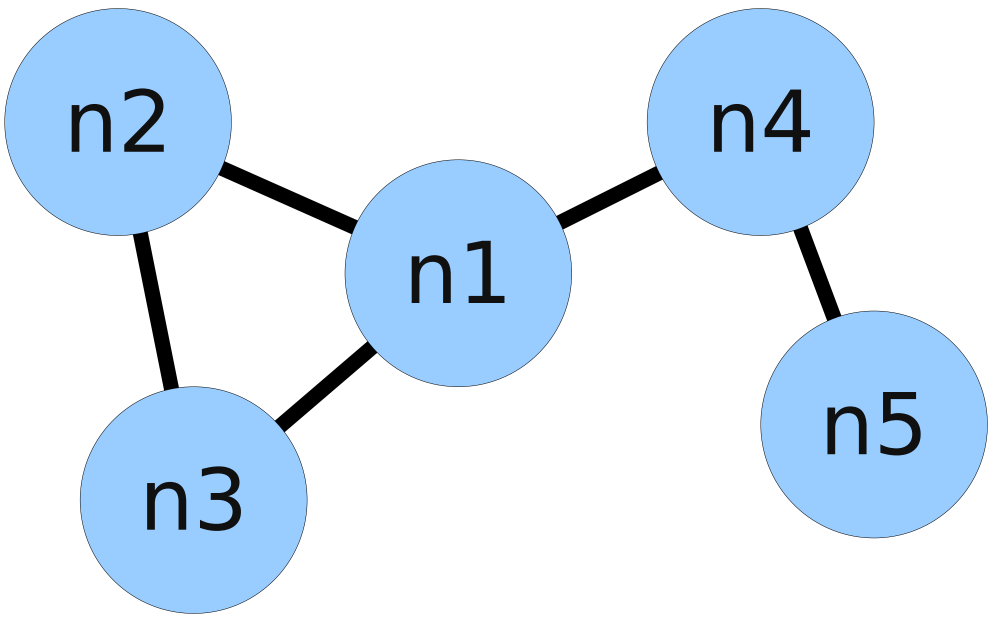
Uses direct communication, no broadcast

Language: table_gossip

- `table_gossip(region):`
  - Broadcast values in the region
  - Each nodes collects overheared values
  - Usage: effective sharing of values

Language: table_gossip

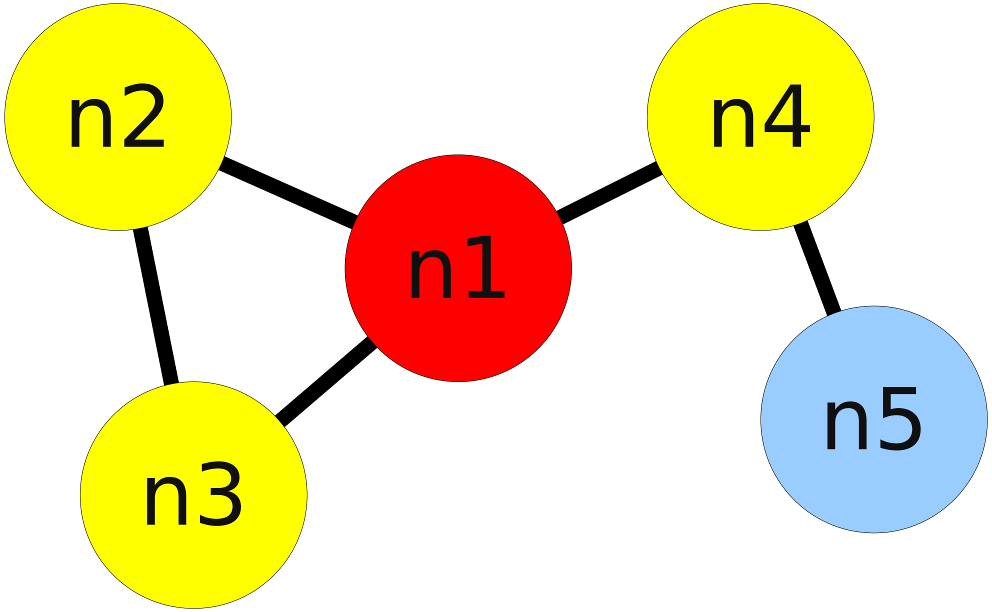
```
regions = table_gossip(world)
```



This will form 5 regions, one for each node.

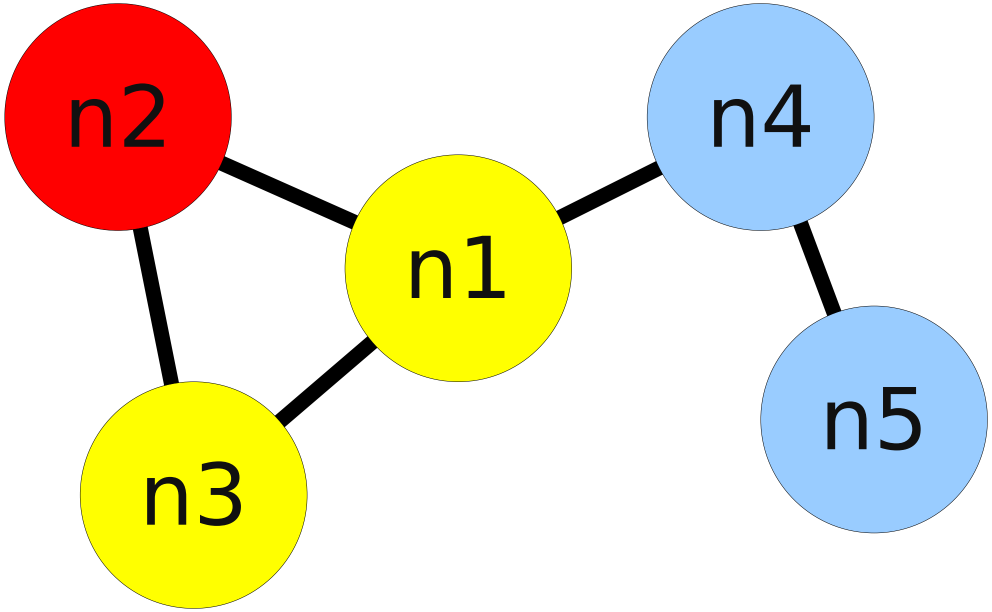
Language: table_gossip (1)

```
regions = table_gossip(world)
```



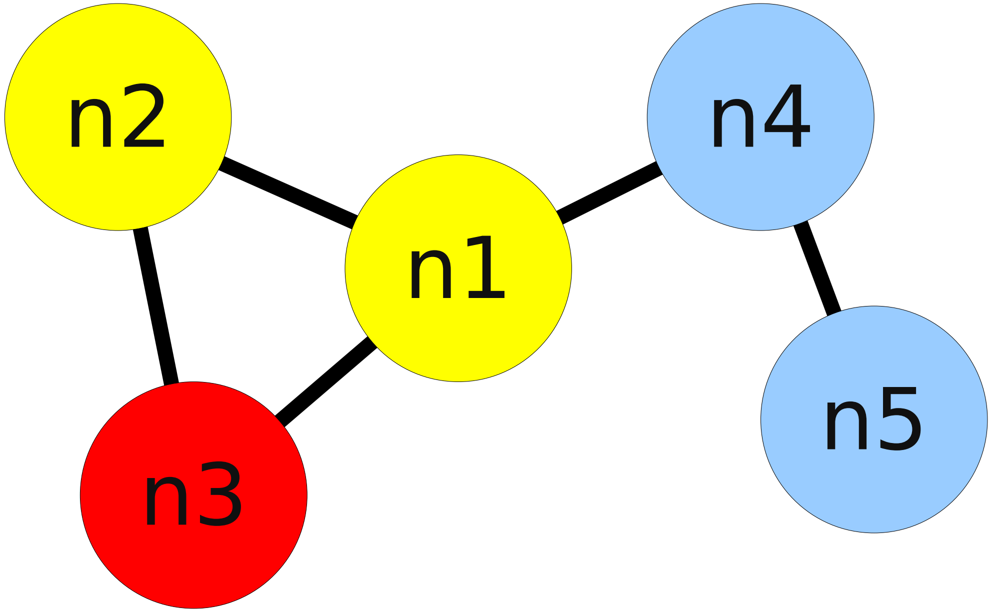
Language: table_gossip (2)

```
regions = table_gossip(world)
```



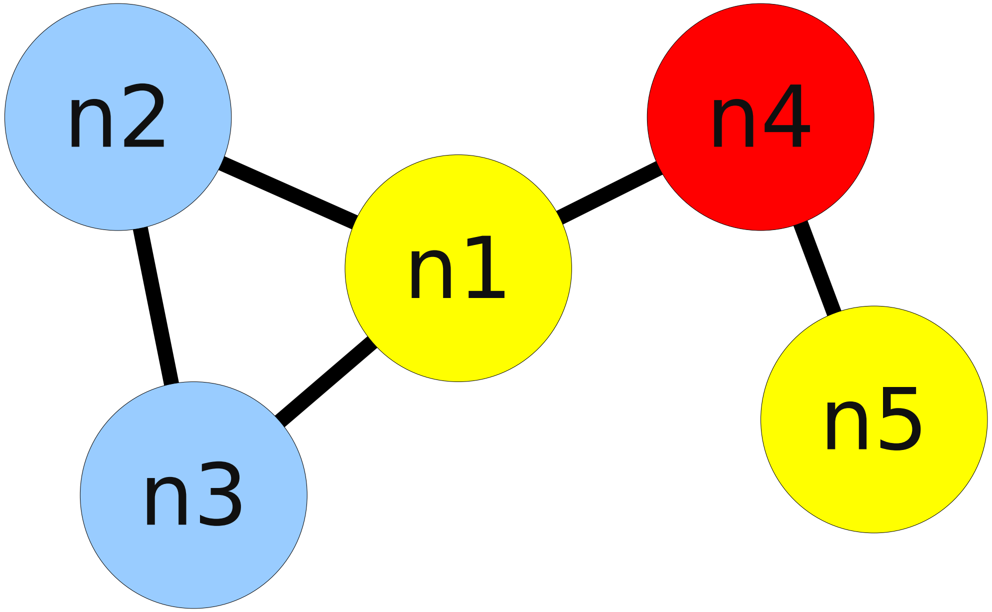
Language: table_gossip (3)

```
regions = table_gossip(world)
```



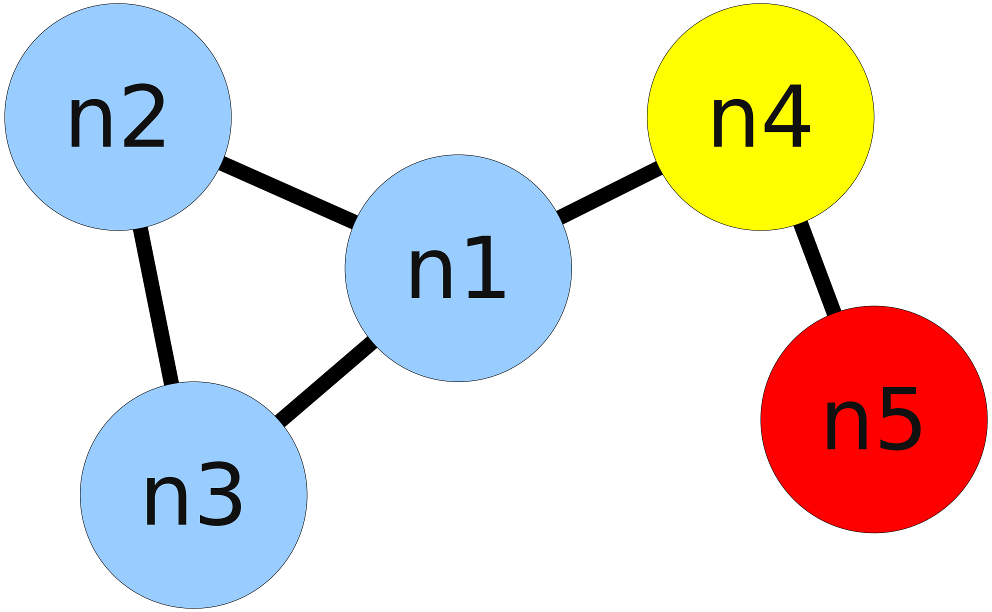
Language: table_gossip (4)

```
regions = table_gossip(world)
```



Language: table_gossip (5)

```
regions = table_gossip(world)
```

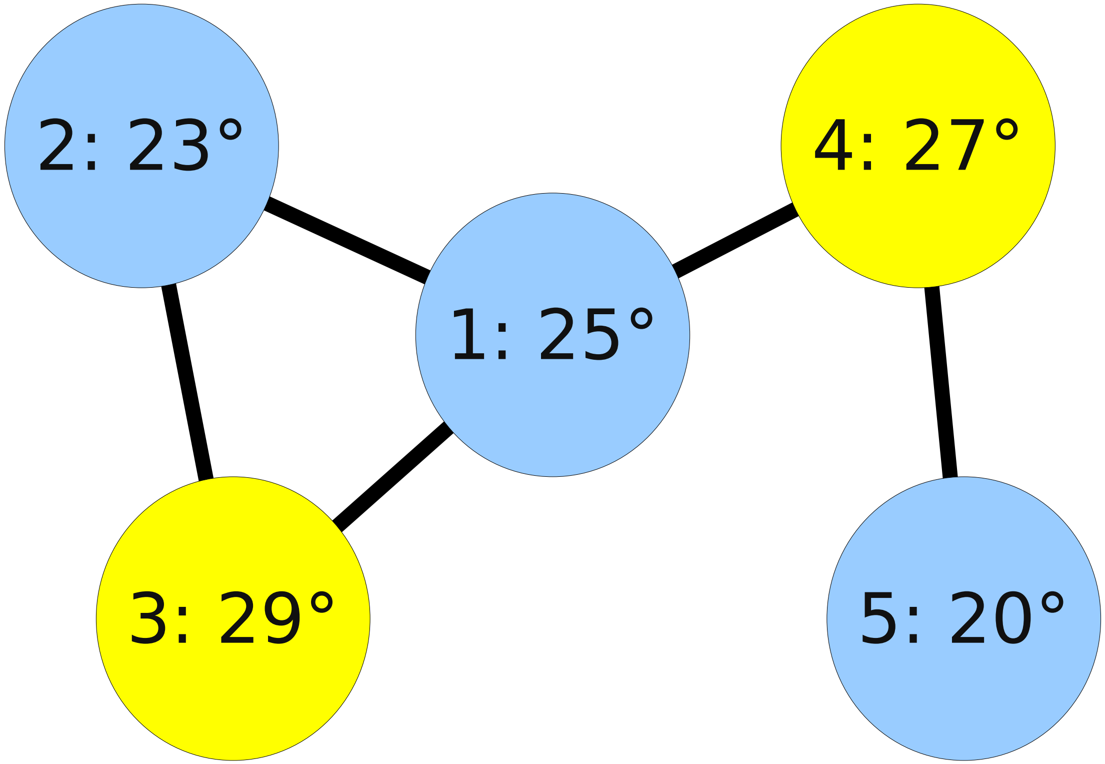


Example: hot spots (task)

- Task: find all nodes which have higher temperature than their neighbours

- On each node: ☎
  - Collect all neighbour's temperatures
  - Check if own temperature is the maximum
  - If so: send to network

Example: hot spots (topology)



Want to find nodes 3 and 4

Example: hot spots (code)

```
temps = rmap(read, world)

tables = table_gossip(temps)
```

tables = { { (1,25), (2,23), (3,29), (4,27) },
{ (1,25), (2,23), (3,29) },
{ (1,25), (2,23), (3,29) },
{ (3,29), (4,27), (5,20) },
{ (4,27), (5,20) } }

Example: hot spots (code)

```
fun max((n0,t0), (n1, t1))
  { (t0>t1) ? (n0,t0) : (n1,t1) }

fun selftest((node, temp))
  { node == nodeid }

maxima = rfold(max, (0,0), tables)
BASE ← rfilter(selftest, maxima)
```

result = { (3,29), (4,27) }

The compiler

- Normalize
- Switch point of view
- Convert to an event-driven program

Compiler: Normalization

- Function inlining
  - `rmap(read, world) → rmap({..}, world)`
- Collapse double rmap
  - `rmap(f, rmap(g, e)) → rmap(f·g, e)`
- Inline rmap into rfold
  - `rfold(f, u, rmap(g,e)`
    `→ rfold( fun(v,a) f(g(v)), u, e )`
- *RQuery* normal form

Compiler: Switch POV

- Flatten a nested query to a stream of operations

```
rfilter( selftest,
          rmap(max, (0,0),
            table_gossip(
              rmap(read, world) )))
```

world → rmap → table_gossip

→ rmap → rfilter

Compiler: event-triggered code

- Nodes represent functionality
- In-edges: event triggers
- Out-edges: handler output

```
world → rmap → table_gossip
      → rmap → rfilter
```

Evaluation

- No comparison to other concepts
- Results come from simulation

- Only bounded recursion ☎
- No persistent variables
  - → Missing integration into general purpose systems

BASE ← beer

Thank you for listening!

```
answers = rmap(me, questions)
```

- *References:*
  - R. Newton, G. Morrisett, and M. Welsh:

    The Regiment Macroprogramming System
  - http://people.csail.mit.edu/newton/IPSN07_ver13.ppt