```
infrastructure infr // refers to infr.ginf

// Cluster
process Cluster = A | R | D

process A {
    send {"arrived", comp.rating} @ (true);
}

process D {
    loop {
        receive (proc.x == "dissolve" && proc.y == comp.partner) {x, y} [
            comp.rank := 2,
            comp.exPartner := comp.partner,
            comp.partner := -1
        ];
    }
}

process R {
    loop {
        receive (proc.x == "propose") {x, y, l, n};
        spawn(Rprime)
    }
}

process Rprime {
    if (comp.lock == 0 && rankDemand(proc.y) < comp.rank){
        send{"accept", comp.idr, proc.n} @ (receiver.idi == proc.l)
[comp.lock := 1];
        receive {
            case (comp.partner != -1 && proc.a == "ack" && proc.b == comp.idr
&& proc.c == proc.l){a, b, c}[
                comp.exPartner := comp.partner, comp.partner := proc.l
            ]:{
                send{"dissolve", comp.idr} @ (receiver.idi == comp.exPartner)
[
                    comp.rank := rankDemand(proc.y),
                    comp.lock := 0
                ] print("U $comp.partner$ with C $comp.idr$ !");
            }

            case (comp.partner == -1 && proc.a == "ack" && proc.b == comp.idr
&& proc.c == proc.l){a, b, c}[
                comp.lock := 0, comp.partner := proc.l, comp.rank :=
rankDemand(proc.y)
            ] print("U $comp.partner$ with C $comp.idr$ !"):{
            }

            case (proc.a == "ack" && proc.b != comp.idr && proc.c == proc.l)
{a, b, c}[comp.lock := 0]:{
            }
        }
```

```
    } else {
        set;
    }
}

// Unit
process Unit = I | T | M | N

process I {
    send{"propose", comp.demand, comp.idi, comp.ref} @ (receiver.rating ==
ratingForRank(comp.ref)) [comp.timer := 0];

    loop {
        if (comp.lock == 0 && !comp.success && comp.timer == comp.timeout){
            send{"propose", comp.demand, comp.idi, next(comp.ref)} @
(receiver.rating == ratingForRank(next(comp.ref))) [
                comp.timer := 0,
                comp.ref := next(comp.ref)
            ];
        } else if (comp.lock == 0 && comp.dissolve) {
            send{"propose", comp.demand, comp.idi, 0} @ (receiver.rating ==
ratingForRank(0)) [
                comp.timer := 0,
                comp.ref := 0,
                comp.dissolve := false,
                comp.success := false,
                comp.rank := noRank(),
                comp.partner := -1
            ];
        } else if (comp.lock == 0 && comp.arrival && comp.bof <= comp.rank - 1
&& comp.partner != -1) {
            send{"dissolve", comp.idi} @ (receiver.idr == comp.partner) [
                comp.success := false,
                comp.ref := comp.bof,
                comp.bof := noRank(),
                comp.rank := noRank(),
                comp.exPartner := comp.partner,
                comp.partner := -1
            ];

            send{"propose", comp.demand, comp.idi, comp.ref} @
(receiver.rating == ratingForRank(comp.ref)) [comp.timer := 0];
        }
    }
}

process T {
    loop {
        waitfor(comp.timer < comp.timeout)[comp.timer := comp.timer + 1];
        waitfor(100);
    }
}
```

```
process N {
    loop {
        receive(proc.x == "accept" && (proc.z != comp.ref || comp.success))
{x, y, z};
        spawn(Nprime)
    }
}

process Nprime {
    send{"ack", -1, comp.idi} @ (receiver.idr == proc.y);
}

process M {
    loop {
        receive {
            case (!comp.success && proc.x == "accept" && proc.z == comp.ref)
{x, y, z}[
                comp.lock := 1,
                comp.success := true,
                comp.rank := comp.ref,
                comp.exPartner := comp.partner
            ]:{
                send{"ack", proc.y, comp.idi} @ (true) [
                    comp.lock := 0,
                    comp.partner := proc.y
                ];
            }

            case (proc.x == "dissolve" && proc.y == comp.partner){x, y}
[comp.dissolve := true]:{
            }

            case (proc.x == "arrived" && comp.bof >= rankRating(proc.y)){x,y}[
                comp.arrival := true,
                comp.bof := rankRating(proc.y)
            ]:{
            }
        }
    }
}

// Functions

function int next(int rank){
    return (rank + 1) % noRank()
}

function int noRank(){
    return 2
}

function string ratingForRank(int rank){
    if (rank == 0){
```

```
        return "H"
    } else {
        return "L"
    }
}

function int rankRating(string rating){
    if (rating == "H"){
        return 0
    } else {
        return 1
    }
}

function int rankDemand(string demand){
    if (demand == "L"){
        return 0
    } else {
        return 1
    }
}

//////////////////// COMPONENTS ////////////////////

// Units
component {
    demand := "L", // L = low, H = high
    idi := 0,
    partner := -1,
    exPartner := -1,
    ref := 0,
    success := false,
    arrival := false,
    dissolve := false,
    rank := 2,
    bof := 2,
    lock := 0,
    timer := 0,
    timeout := 20
} : Unit

component {
    demand := "H", // L = low, H = high
    idi := 1,
    partner := -1,
    exPartner := -1,
    ref := 0,
    success := false,
    arrival := false,
    dissolve := false,
    rank := 2,
    bof := 2,
    lock := 0,
```

```
    timer := 0,
    timeout := 20
} : Unit

component {
    demand := "L", // L = low, H = high
    idi := 2,
    partner := -1,
    exPartner := -1,
    ref := 0,
    success := false,
    arrival := false,
    dissolve := false,
    rank := 2,
    bof := 2,
    lock := 0,
    timer := 0,
    timeout := 20
} : Unit

component {
    demand := "H", // L = low, H = high
    idi := 3,
    partner := -1,
    exPartner := -1,
    ref := 0,
    success := false,
    arrival := false,
    dissolve := false,
    rank := 2,
    bof := 2,
    lock := 0,
    timer := 0,
    timeout := 20
} : Unit

component {
    demand := "H", // L = low, H = high
    idi := 4,
    partner := -1,
    exPartner := -1,
    ref := 0,
    success := false,
    arrival := false,
    dissolve := false,
    rank := 2,
    bof := 2,
    lock := 0,
    timer := 0,
    timeout := 20
} : Unit

// Clusters
```

```
component {
    rating := "H", // L = low, H = high
    idr := 0,
    partner := -1,
    exPartner := -1,
    rank := 2,
    lock := 0
} : Cluster

component {
    rating := "H", // L = low, H = high
    idr := 1,
    partner := -1,
    exPartner := -1,
    rank := 2,
    lock := 0
} : Cluster

component {
    rating := "H", // L = low, H = high
    idr := 2,
    partner := -1,
    exPartner := -1,
    rank := 2,
    lock := 0
} : Cluster

component {
    rating := "L", // L = low, H = high
    idr := 3,
    partner := -1,
    exPartner := -1,
    rank := 2,
    lock := 0
} : Cluster

component {
    rating := "L", // L = low, H = high
    idr := 4,
    partner := -1,
    exPartner := -1,
    rank := 2,
    lock := 0
} : Cluster
```