

Spatial Approaches to Pervasive Computing

Tutorial at IEEE SASO 2008



Jonathan Bachrach (MIT)
jrb@csail.mit.edu



Jacob Beal (BBN Technologies)
jakebeal@bbn.com



Cristian Borcea (NJIT)
borcea@njit.edu



Marco Mamei (Univ. Modena Reggio Emilia)
marco.mamei@unimore.it

Syllabus

- Pervasive Computing, Spatial Computing
 - Example Scenarios
- Overview of Spatial Approaches
- Three spatial programming models:
 - Spatial Programming / Smart Messages
 - TOTA / Field-based Coordination
 - Proto / Amorphous Medium abstraction

Wireless devices are filling our environment...



Wireless devices are filling our environment...



Wireless-enabled embedded systems

- >3.3B cell phones vs. 600M Internet-connected PC's in 2007
 - >600M cell phones with Internet capability, rising rapidly
- New cars come equipped with navigation systems and will soon have wireless interfaces (WiFi/DSRC, cellular, WiMax)
- Sensor deployment just starting, but some estimates ~5-10B units by 2015
- Military/emergency response wireless robots, unmanned vehicles, unmanned aircraft

Networked wireless devices

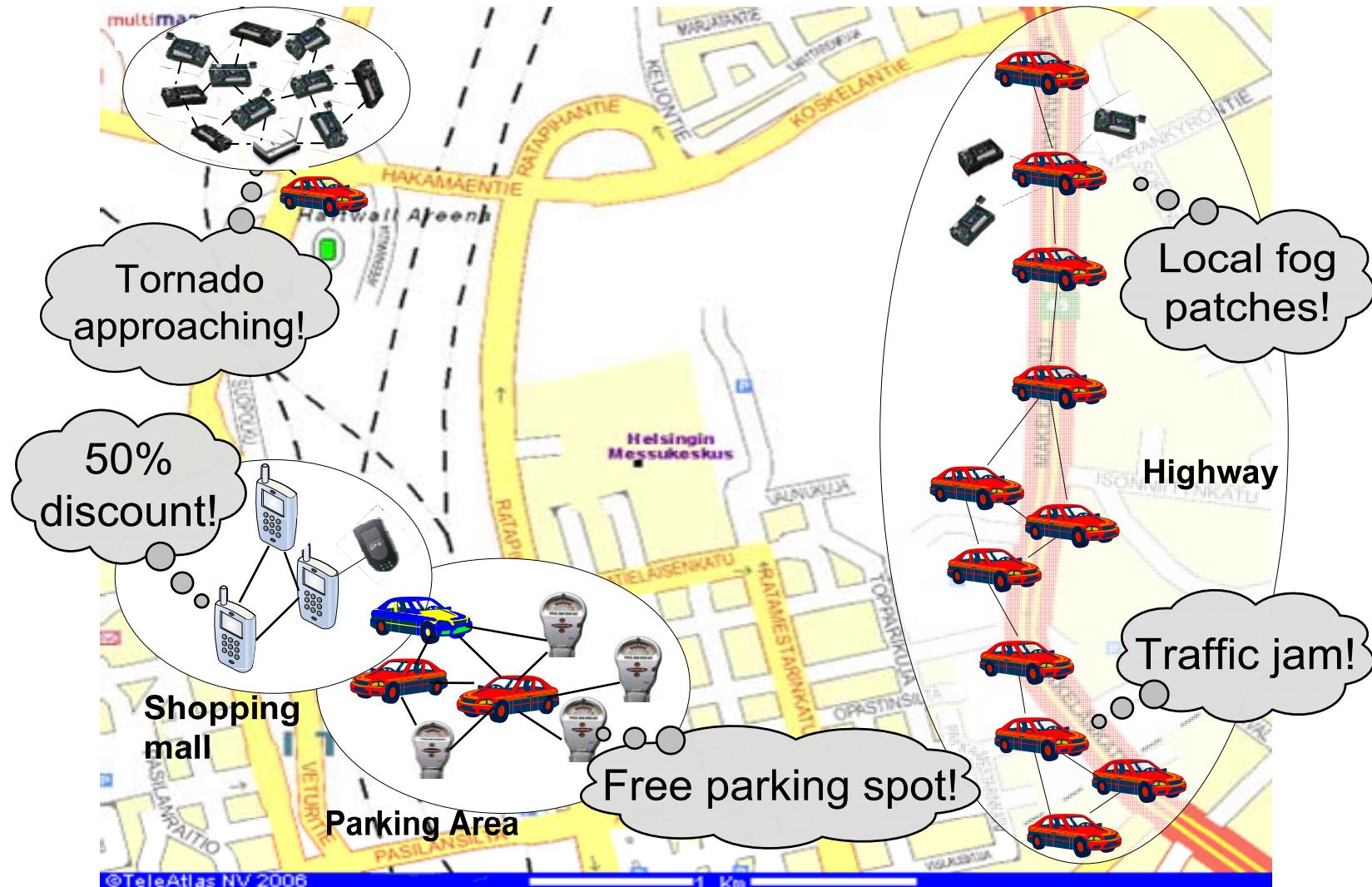


Pervasive computing vision



- Computing, communication, and sensing anytime, anywhere
- Wireless embedded systems cooperate to achieve global tasks

Outdoor distributed applications

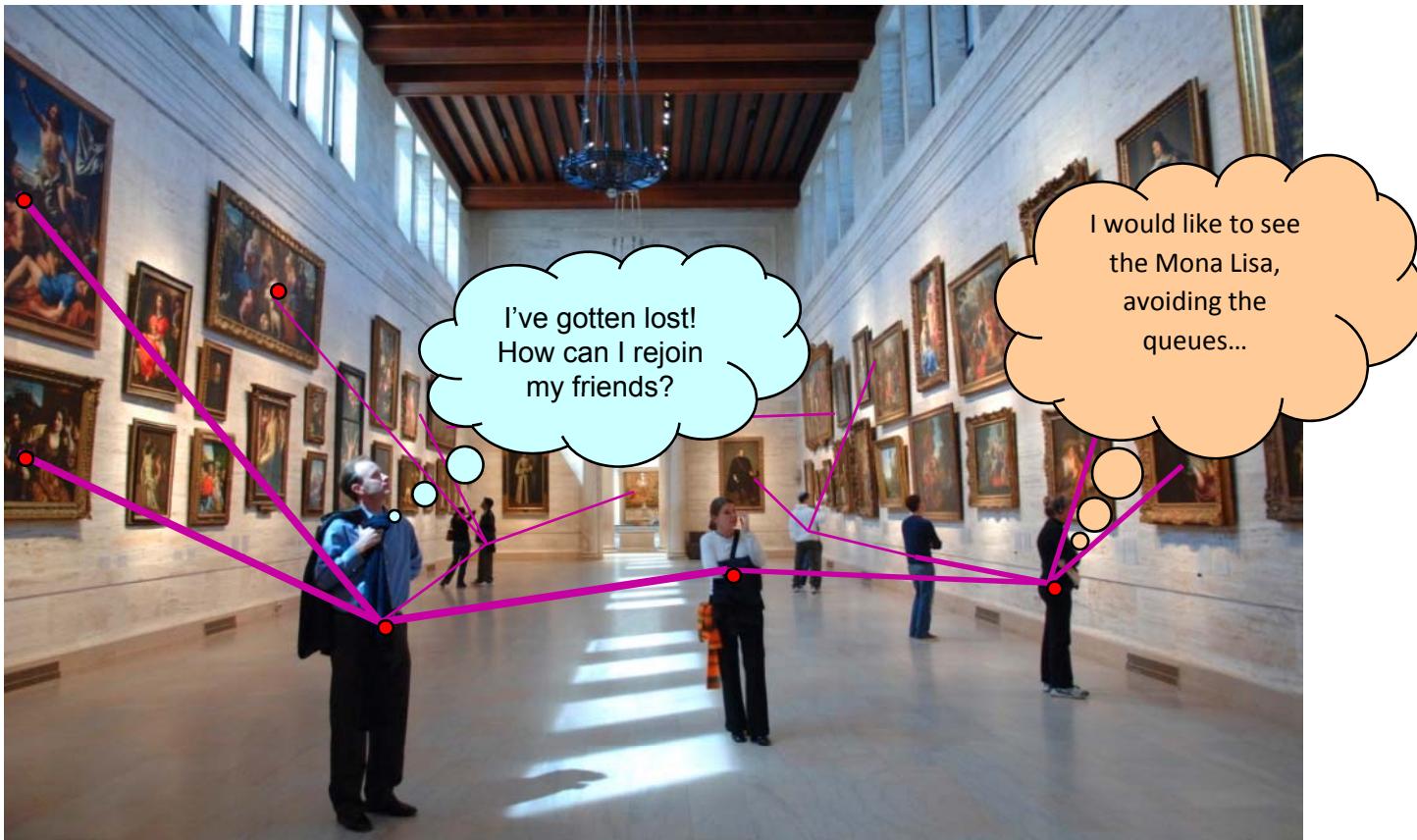


Example: intrusion detection

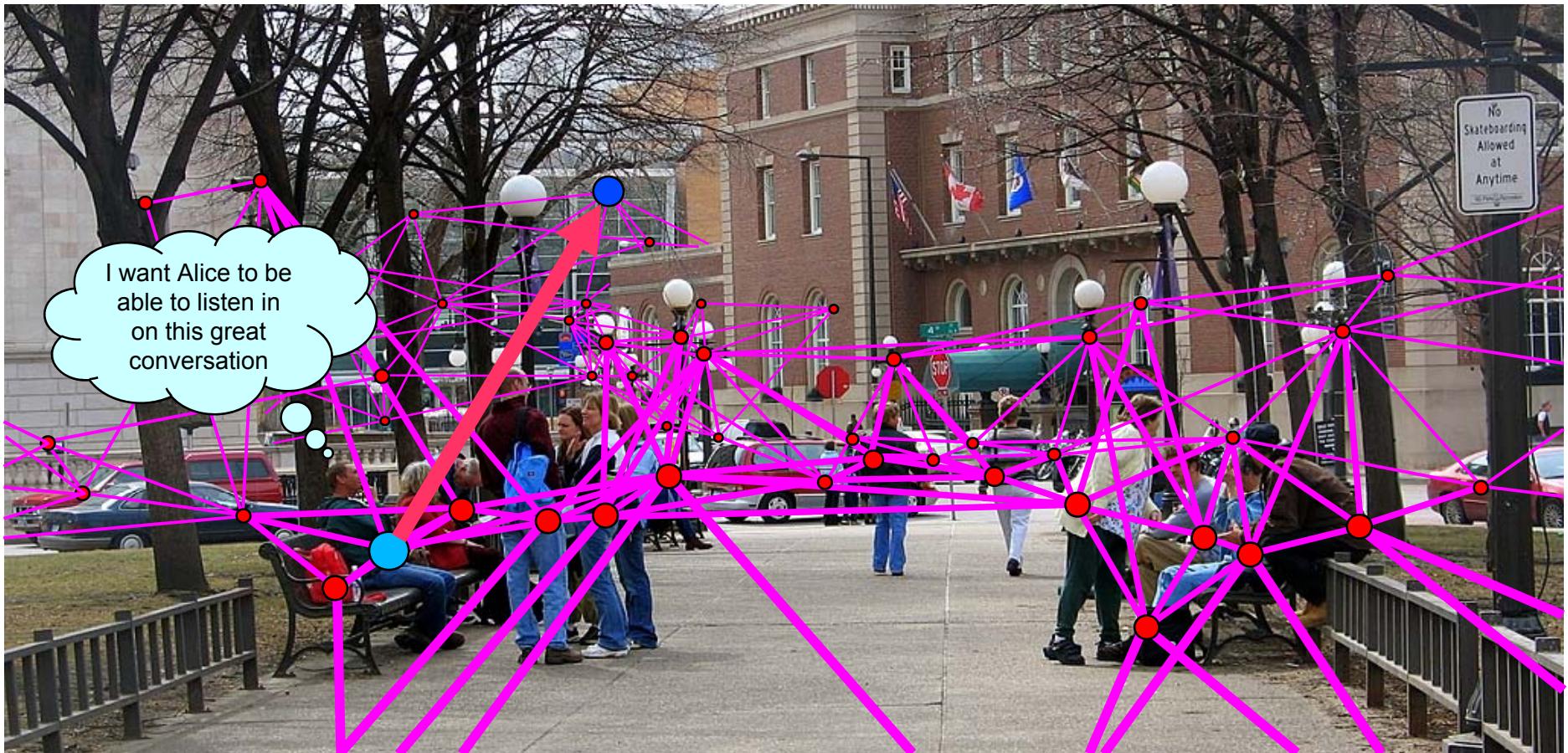


- Intrusion detection across hills using motion sensors and autonomous robots with cameras
- Number and location of systems are unknown
- Configuration is not stable over time
 - Intrusions can appear randomly
 - Robots can move

Example: museum guide



Example: mobile streaming



Not Very Close to This Vision Yet

- Nomadic computing
 - Devices: laptops
 - Internet: intermittent connectivity
 - Work: typical desktop applications
- Mobile communication
 - Devices: PDAs, mobile phones, Blackberries
 - Internet: continuous connectivity
 - Work: read email, potentially web
- Experimental sensor networks
 - Devices: Berkeley/Crossbow motes
 - Internet: Possible through base station
 - Work: Monitor environment, wildlife



Why?

- **Hard to program distributed applications over collections of wireless systems**
 - Systems: distributed across physical space, mobile, heterogeneous hardware and software, resource-constrained (battery, bandwidth, memory)
 - Networks: large scale volatile (ad hoc topologies, dynamic resources), less secure than wired networks

Traditional distributed computing does not work well outdoors

- End-to-end data transfers may rarely complete
- Fixed address naming and routing (e.g., IP) are too rigid
- Difficult to deploy new applications in existing networks

**Outdoor distributed computing requires novel programming
models and system architectures!**

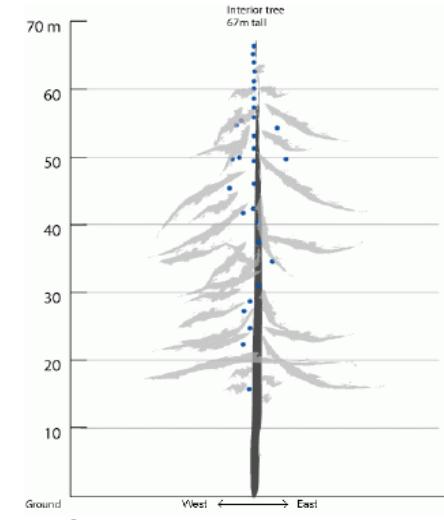
Spatial computers



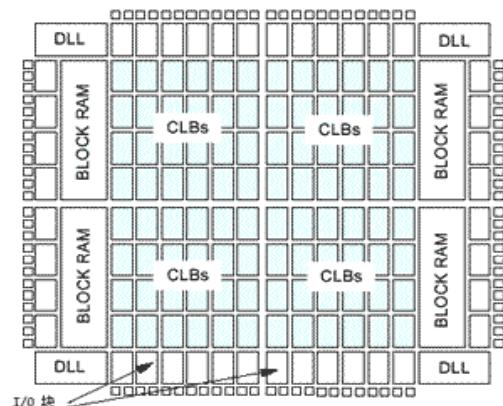
Robot Swarms



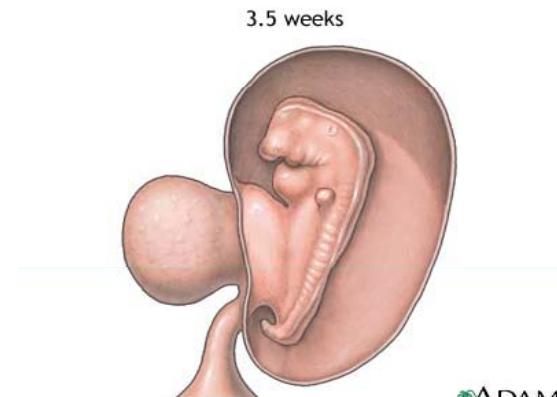
Biological Computing



Sensor Networks



Reconfigurable Computing



Cells during Morphogenesis



Modular Robotics

More formally...

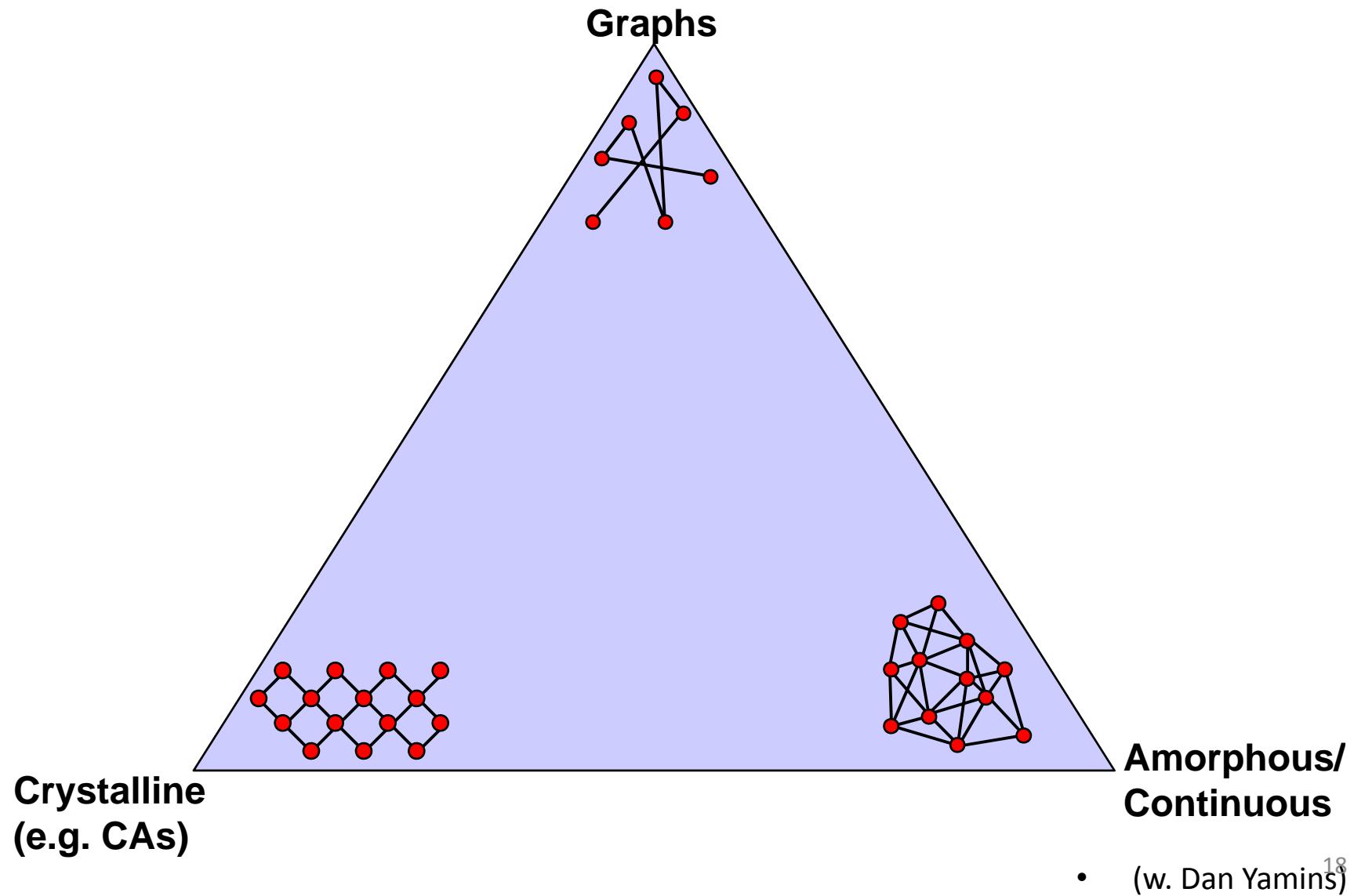
- A spatial computer is a collection of computational devices distributed through a physical space in which:
 - the difficulty of moving information between any two devices is strongly dependent on the distance between them, and
 - the “functional goals” of the system are generally defined in terms of the system's spatial structure

More formally...

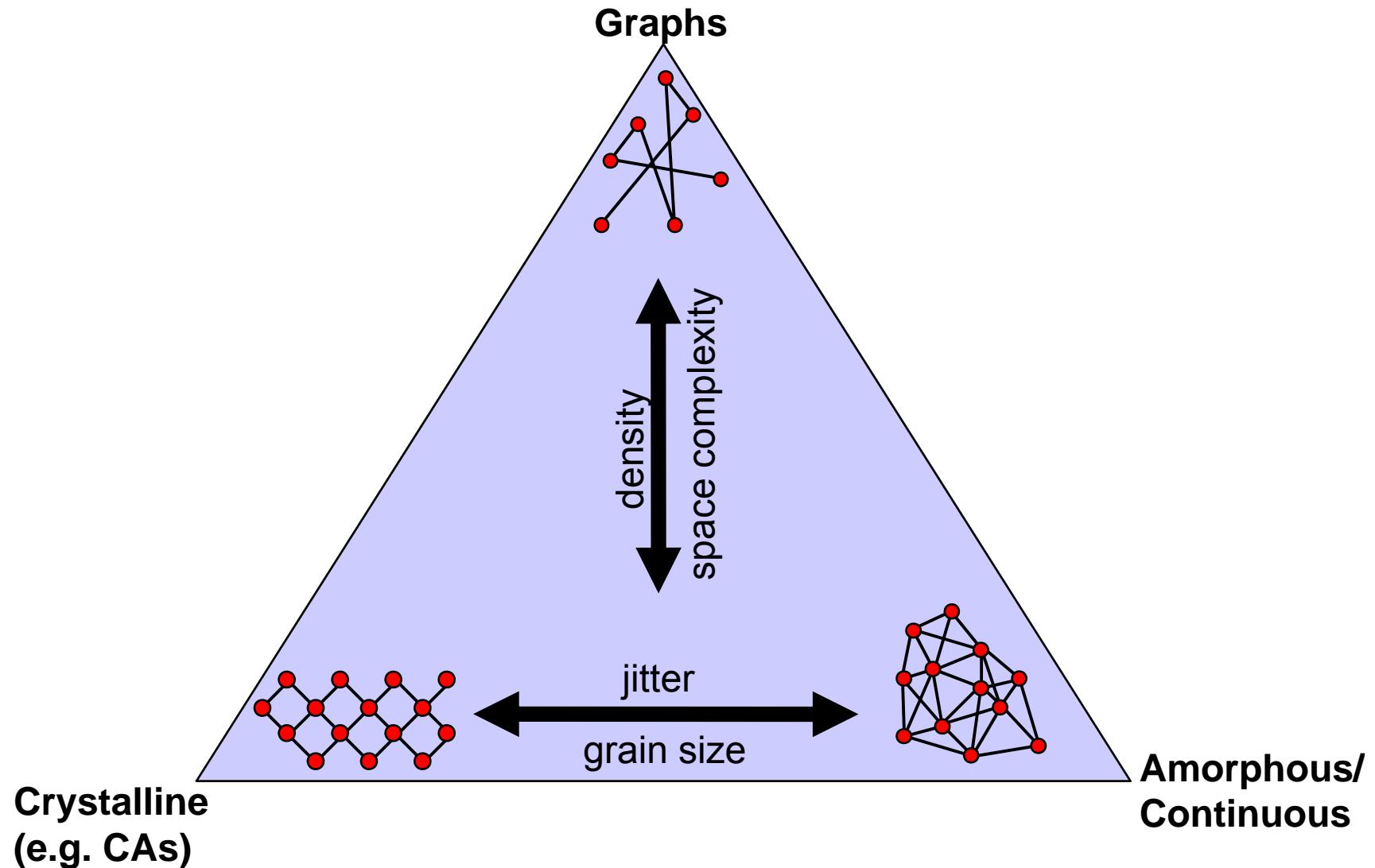
- A spatial computer is a collection of computational devices **distributed through** a physical space in which:
 - the difficulty of moving information between any two devices is **strongly dependent on the distance** between them, and
 - the “functional goals” of the system are **generally defined** in terms of the system's spatial structure

Notice the ambiguities in the definition

Promising Solution: Spatial Computing Approaches

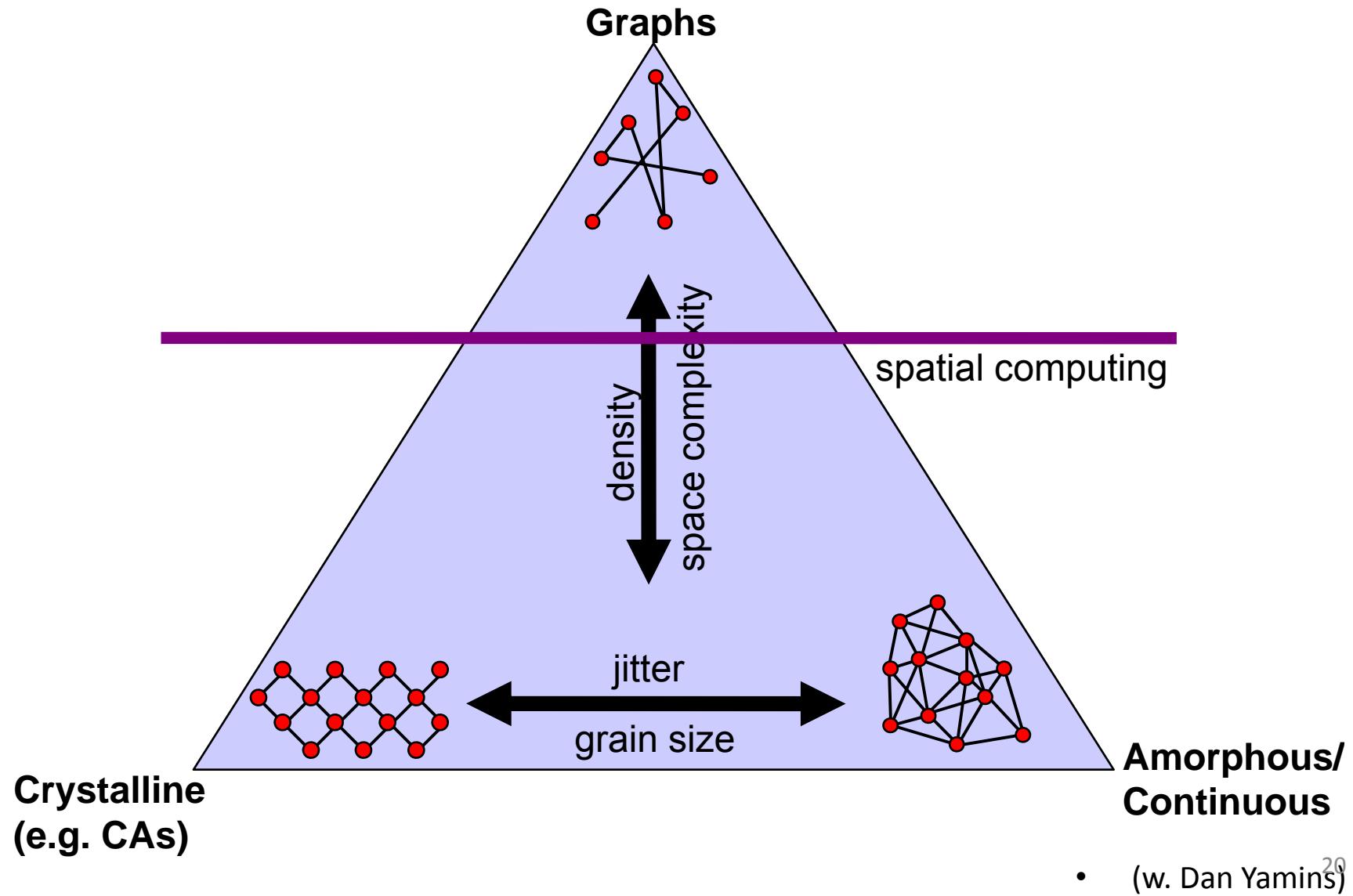


Promising Solution: Spatial Computing Approaches



• (w. Dan Yamins)¹⁹

Promising Solution: Spatial Computing Approaches



• (w. Dan Yamins)²⁰

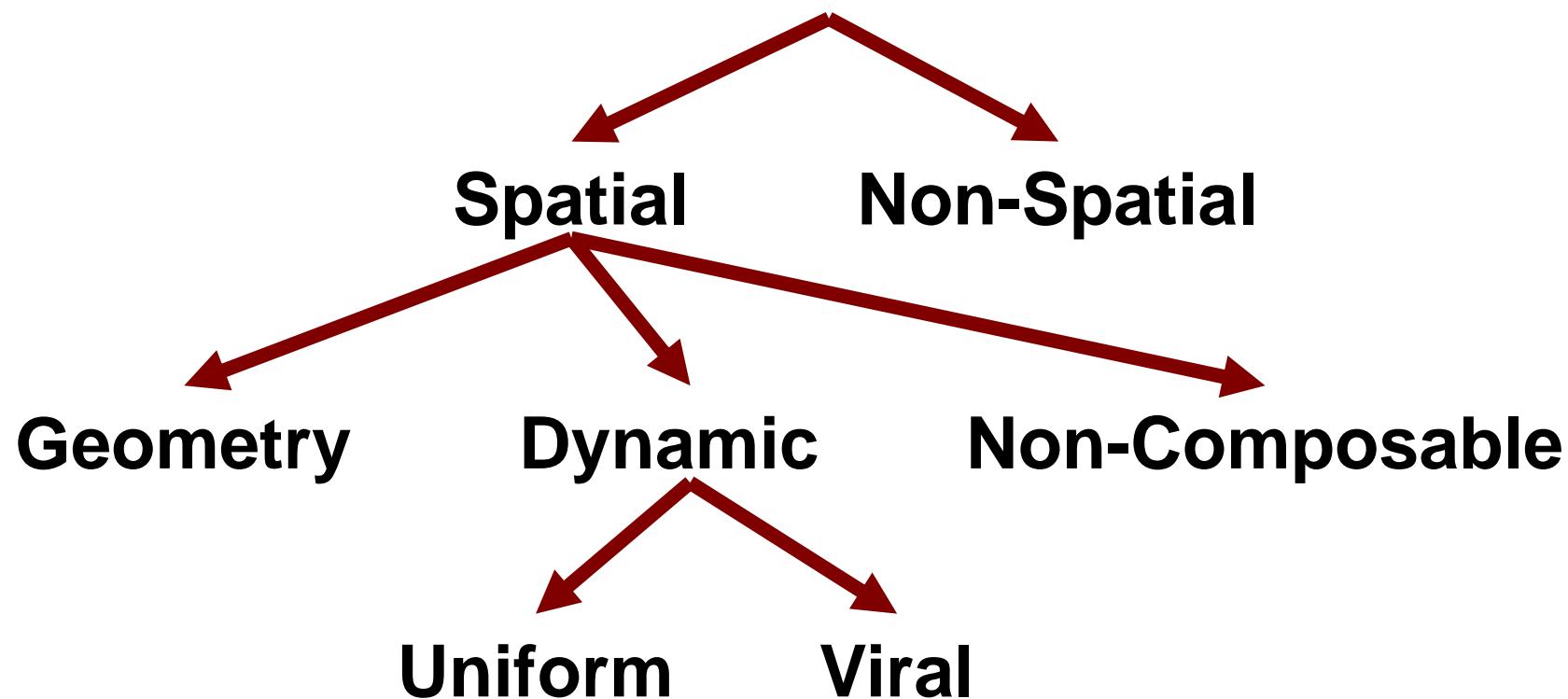
Desiderata for spatial computing approaches

- Take advantage of spatial nature of problems
- Simple, easy to understand code
- Scalable to potentially vast numbers of devices
- Robust to errors, adapt to changing environment

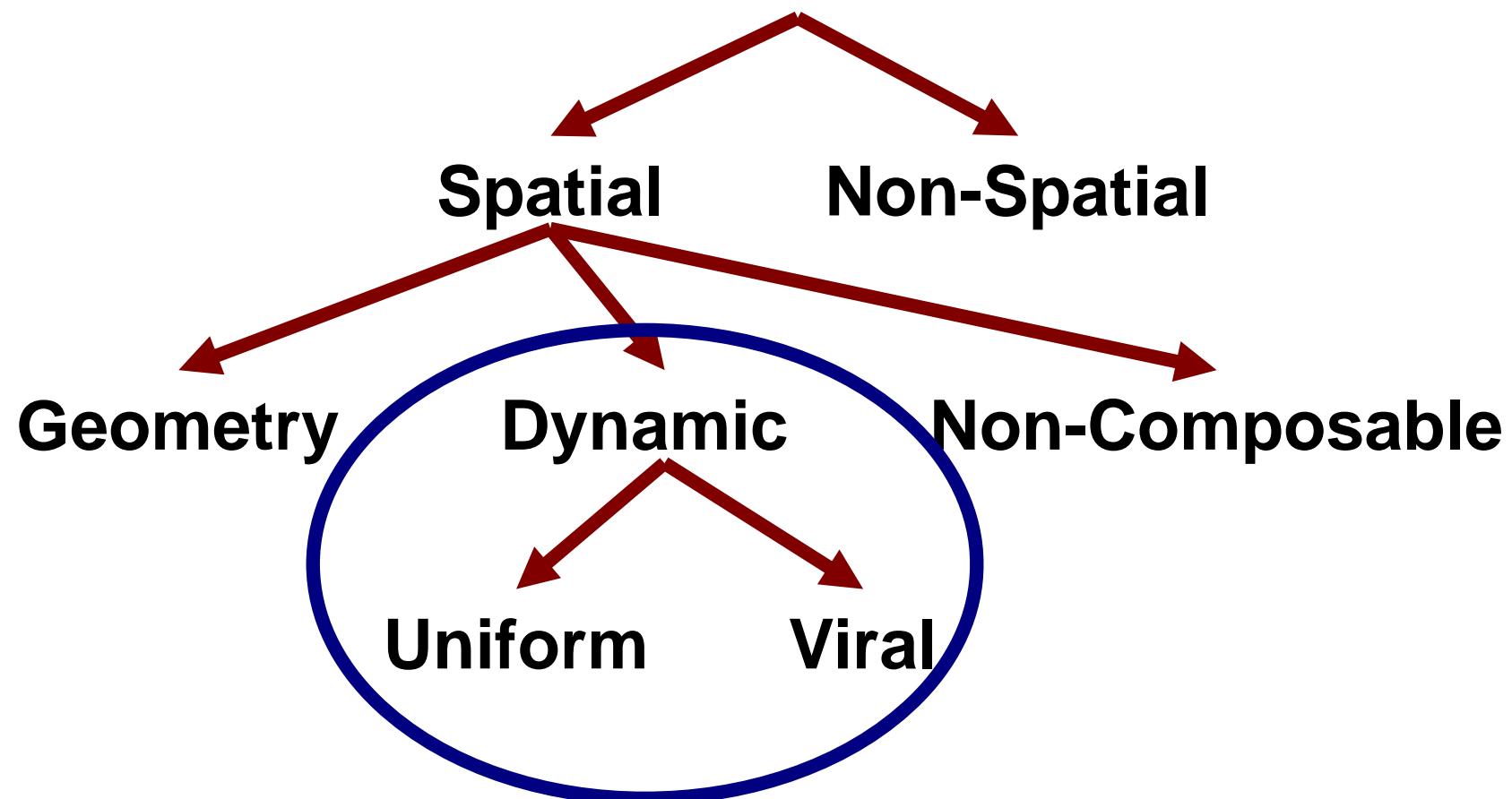
Syllabus

- Pervasive Computing, Spatial Computing
 - Example Scenarios
- **Overview of Spatial Approaches**
- Three spatial programming models:
 - Spatial Programming / Smart Messages
 - TOTA / Field-based Coordination
 - Proto / Amorphous Medium abstraction

A taxonomy of approaches



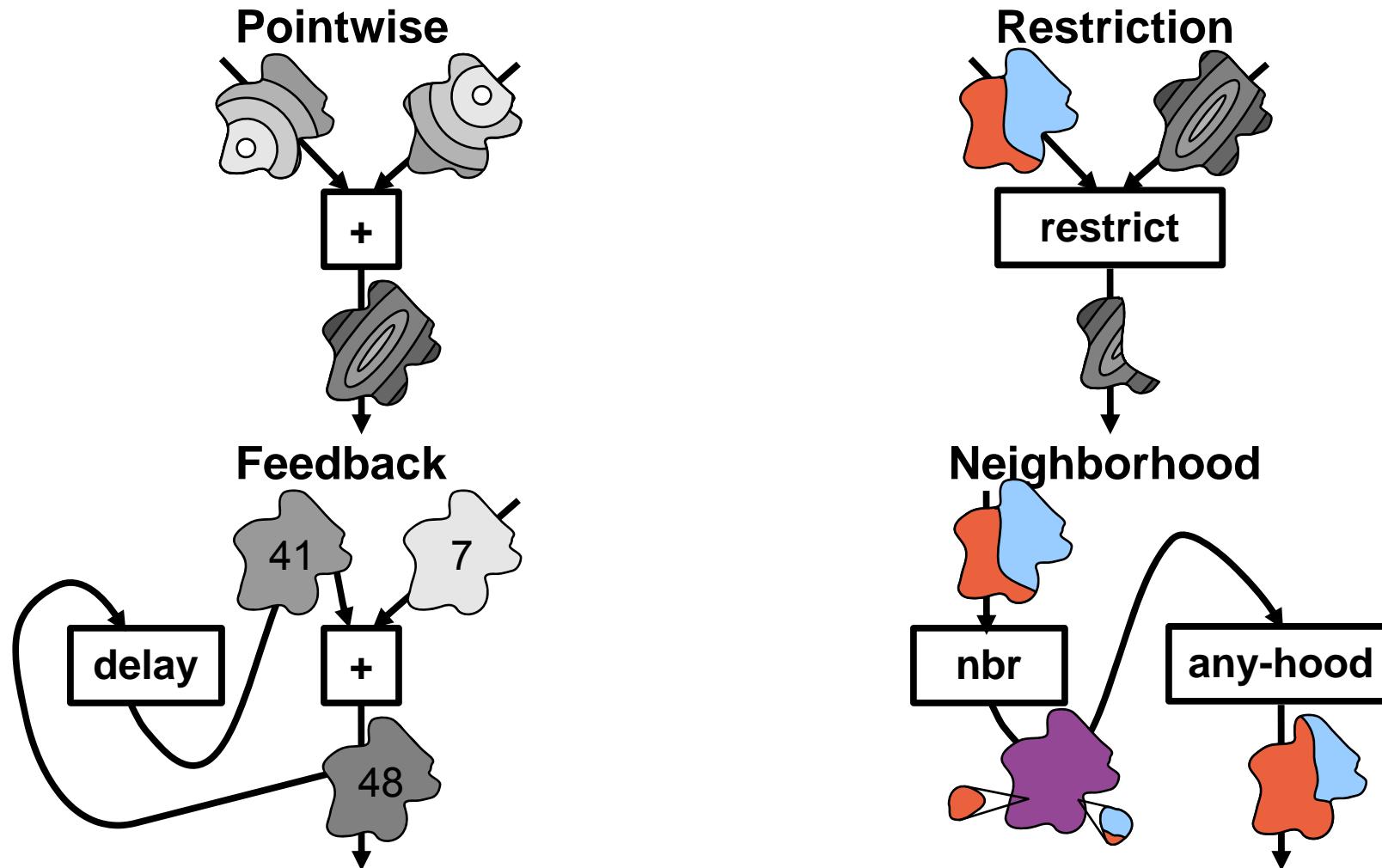
A taxonomy of approaches



Approaches from local dynamics

- Primitives describe only actions between devices and the neighbors they communicate with.
- Advantages: coherent and correct semantics
- Disadvantages: programmer must figure out how to marshal local dynamics to produce coherent large-area programs

Proto: computing with fields



TOTA: viral tuples

- Middleware to create distributed data structures like **gradients** and **fields**.
- Data structures made of **tuples injected from a node** and virally spread across the network.
- $T = (C, P, M)$
- **API** to inject/read tuples
- **API** to define (C, P, M)

C = value;
P = propagate breadth-first
increasing value by 1 at every hop;
M = maintain gradient despite
network reconfiguration;

Other viral approaches

- Smart Messages (Borcea)
 - Execution migration to nodes of interest
 - Nodes of interest discovered using self-routing
- Paintable Computing (Butera)
 - Consistent transfer, view of neighbor data
 - Code for install, uninstall, transfer control and update
- RGLL (Sutherland)
 - Code for arrival, tick, collision, departure
 - Communication via collision

Approaches from geometry

- Primitives describe large-scale geometric regions (e.g. “all devices on the left hill”)
- Advantages: coherent, easy to specify large-scale programs
- Disadvantages: generally easy to accidentally specify programs that cannot be executed correctly

Regiment

- Provide a rich set of operators to work with data distributed over time and space.
- Simple Regions, created from geometric or radio relationships:
 - K-hop-neighborhood
 - K-nearest nodes
 - All nodes within circle (square, etc.)
- Derived Regions, built from other regions
 - Union, Intersection
 - Map, Filter

Spatial programming



- Virtual name space over outdoor networks of embedded systems

- Systems named by **spatial references** using their locations and properties
- Applications are sequential programs that read/write spatial references (similar to regular variables)
- Read/write trigger transparent program migrations on each referenced system

Other geometric approaches

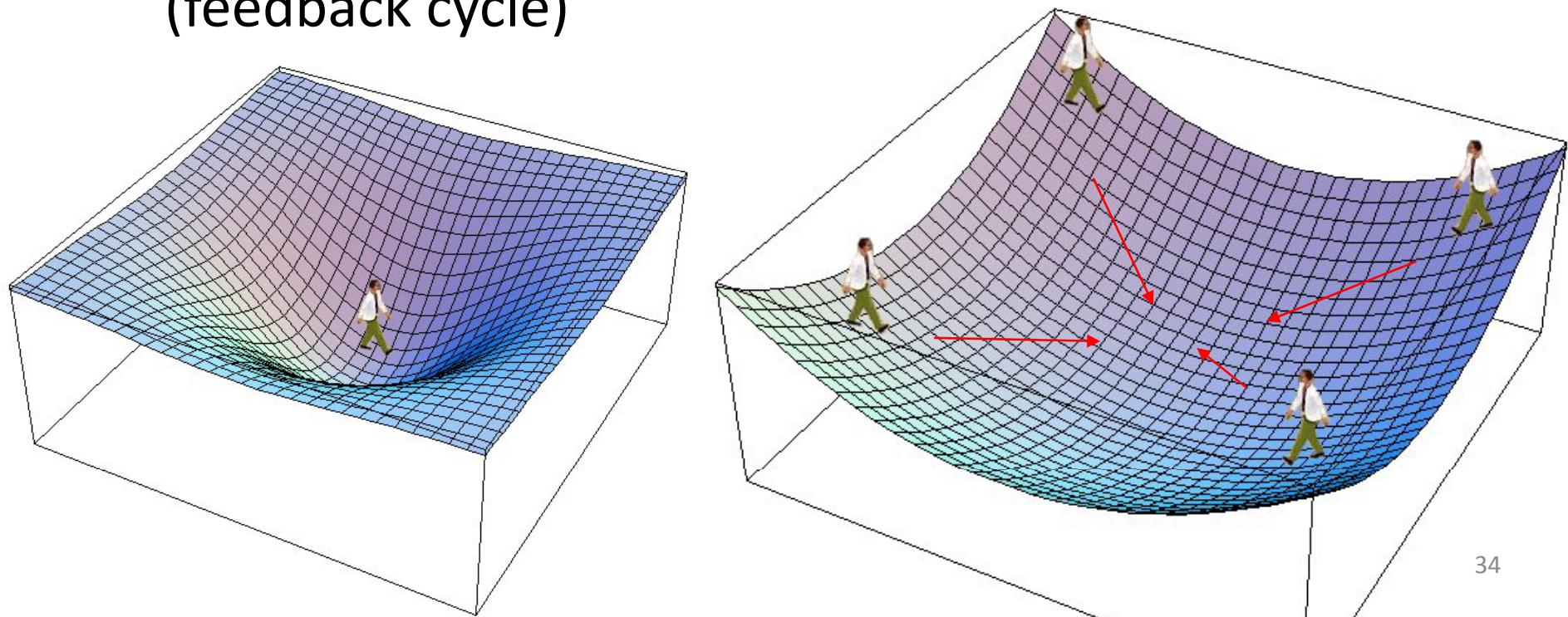
- EgoSpaces
- SpatialViews
- Spidey
- Abstract Regions

Non-composable approaches

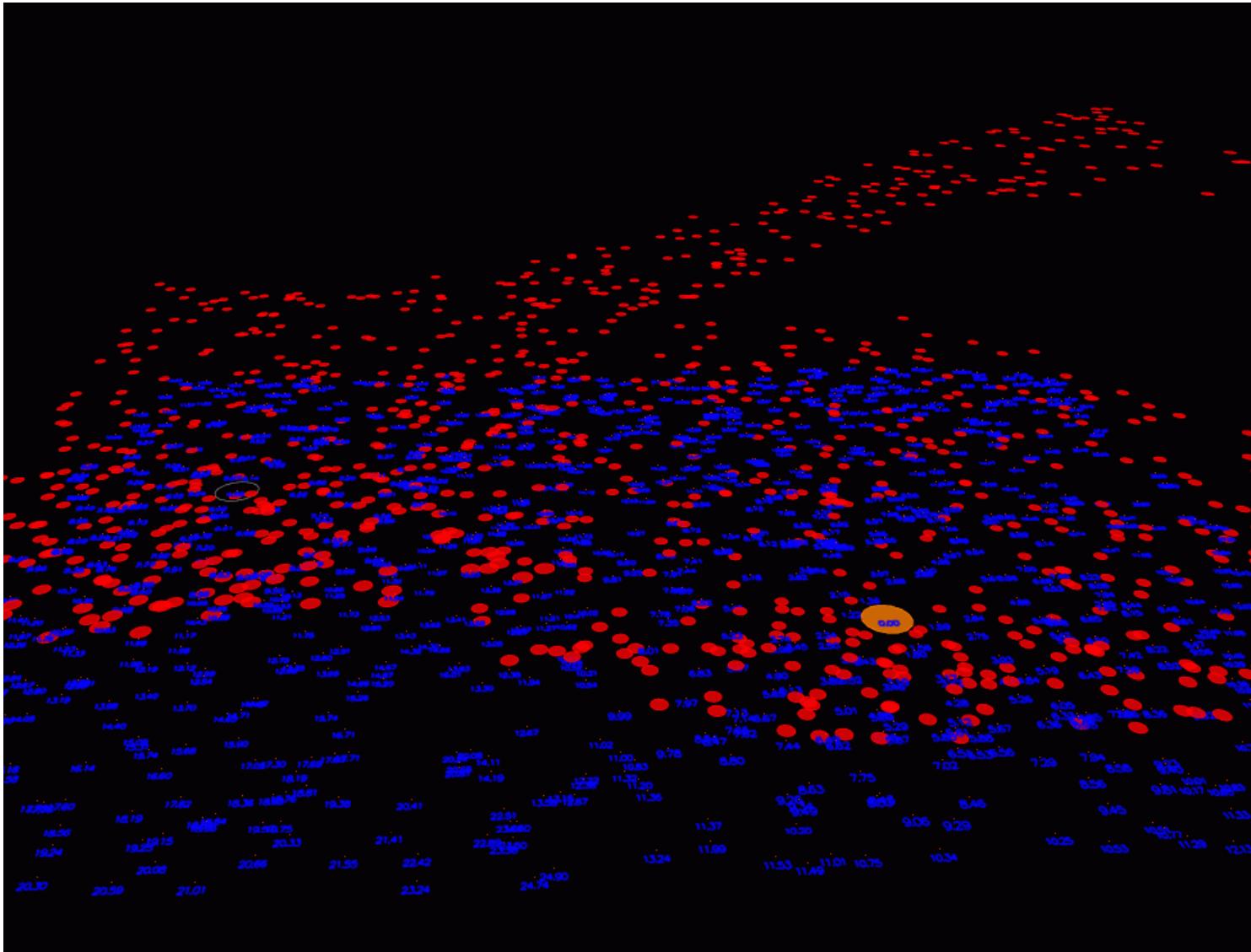
- Algorithms and techniques, generally based on geometry, but not part of a system of composable parts
- Advantages: powerful spatial ideas for that are good for inclusion in code libraries
- Disadvantages: developed as stand-alone ideas, and may have limited composability

Field-based coordination

- Contextual information is expressed by means of **distributed data-structures** (i.e. fields) spread by agents.
- Agent move and act being driven by these fields (feedback cycle)



Self-healing gradients



Other non-composable approaches

- Yamins' locally checkable patterns
 - Family of self-stabilizing CA patterns
- hood (Whitehouse, et. al.)
 - nesC library for interacting with neighbors
- McLurkin's “Stupid Robot Tricks”
 - Swarm behaviors intended mainly for time-wise multiplexing.
- *Countless one-shot systems...*

Significant non-spatial approaches

- “roll-your-own” (e.g. C/C++)
- TinyDB
 - Distributed database queries for sensor networks
- Kairos
 - Distributed graph algorithms
- WaveScript
 - Distributed streaming language
 - Follow-on to Regiment w/o the spatial primitives

Syllabus

- Pervasive Computing, Spatial Computing
 - Example Scenarios
- Overview of Spatial Approaches
- Three spatial programming models:
 - **Spatial Programming / Smart Messages**
 - TOTA / Field-based Coordination
 - Proto / Amorphous Medium abstraction

Outdoor programming example



- Intrusion detection across the hills using motion sensors and autonomous mobile robots with cameras
- Number and location of systems are unknown
- Configuration is not stable over time
 - Intrusions can appear randomly
 - Robots can move

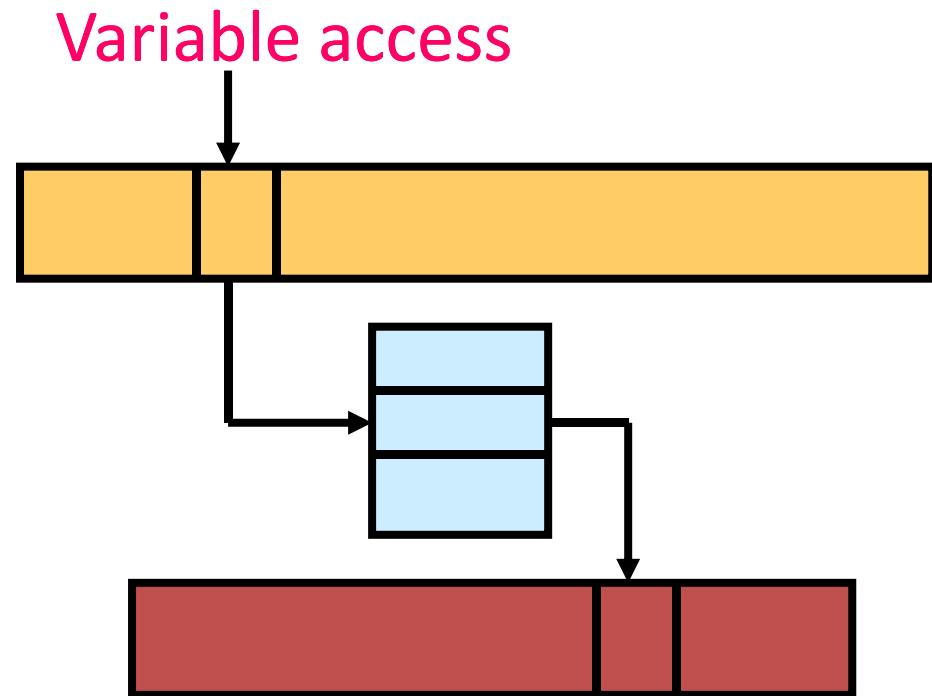
Traditional (indoor) programming

Program

Virtual Address Space

Page Table + OS

Physical Memory



- Programs access data through variables
- Variables mapped to physical memory locations
- Page Table and OS guarantee reference consistency
- Access time has an (acceptable) upper bound

Software distributed shared memory

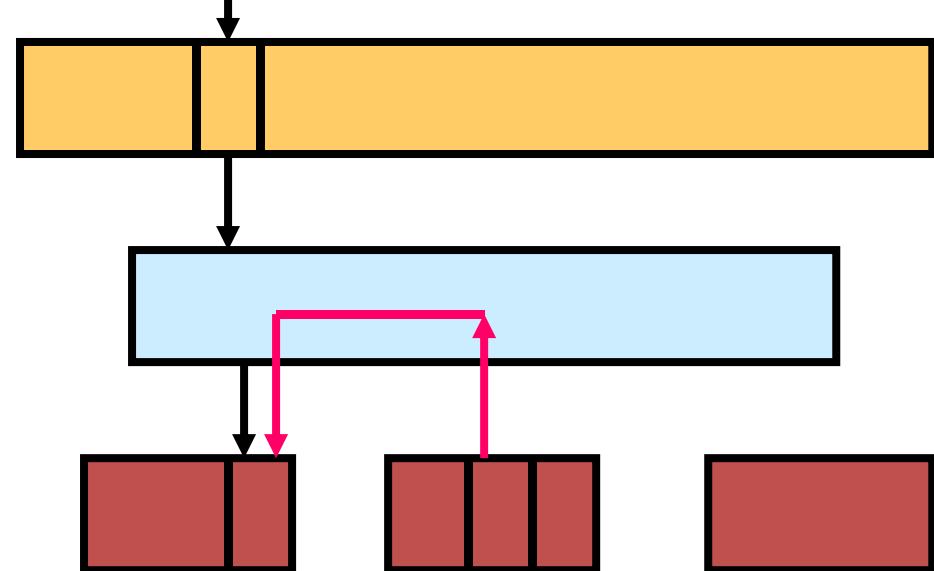
Distributed Application

Shared virtual
address space

Page Table +
Message Passing

Distributed
Physical Memory

Variable accesses



- Applications access distributed data through shared variables
- Runtime system translates variable accesses into message passing (when necessary)

From indoor to outdoor computing

Virtual Address Space	Space Region
Variables	Spatial References
Variables mapped to physical memory	Spatial references mapped to systems embedded in the physical space
Reference consistency	?
Bounded access time	?

Spatial Programming (SP) at a glance

- Provides a **virtual name space** over outdoor networks of embedded systems
- Embedded systems named by their locations and properties
- Runtime system takes care of name resolution, reference consistency, and networking aspects
- Implementation on top of Smart Messages: SP applications execute, sequentially, on each system referenced in their code

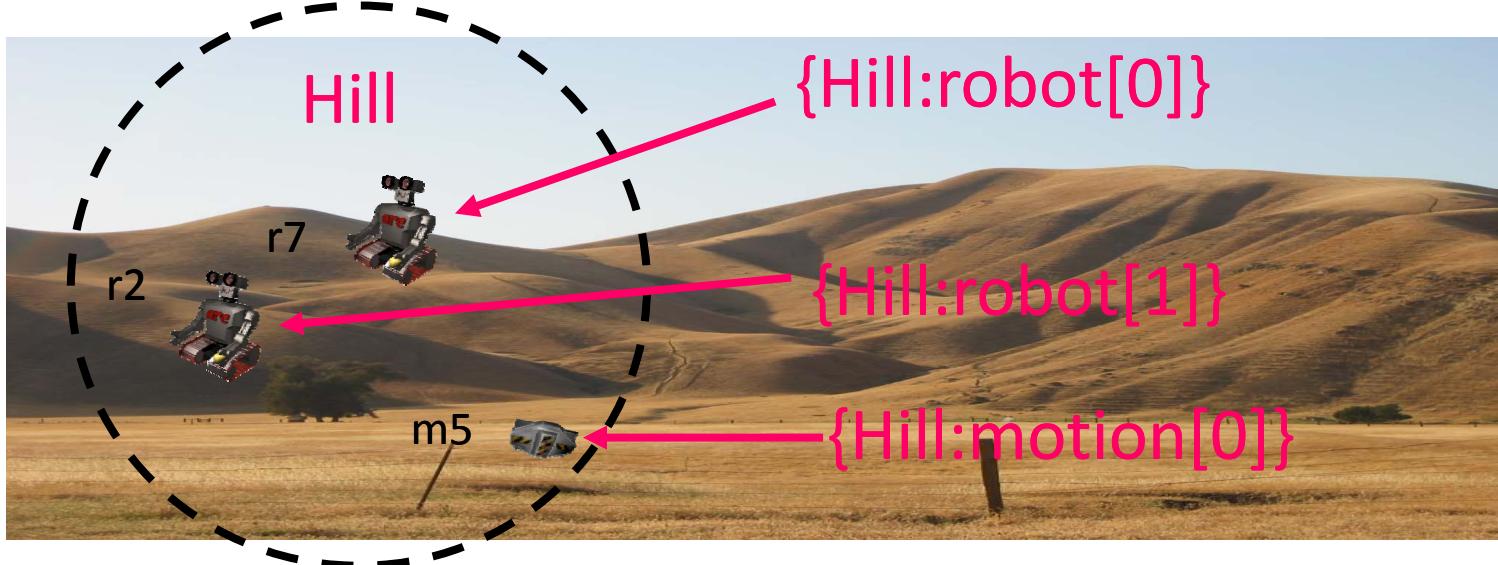
Space regions

Hill = new Space({lat, long}, radius);



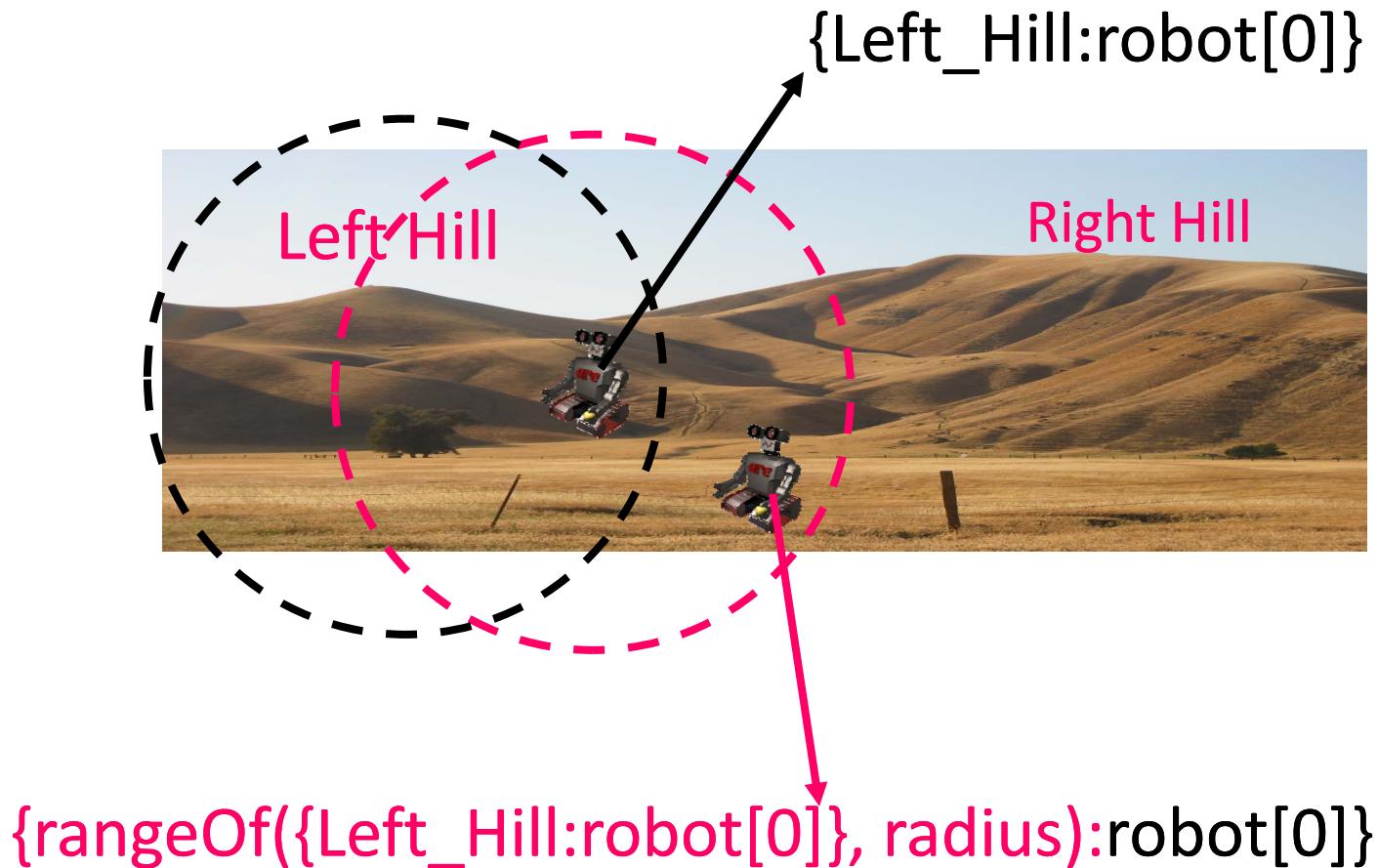
- Virtual representation of a physical space
- Similar to a virtual address space in a conventional computer system
- Defined statically or dynamically

Spatial references



- Defined as **{space:property}** pairs
- Virtual names for embedded systems
- Similar to variables in conventional programming
- Indexes used to distinguish among similar systems in the same space region

Relative space regions



From indoor to outdoor computing

Virtual Address Space	Space Region
Variables	Spatial References
Variables mapped to physical memory	Spatial references mapped to systems embedded in the physical space
Reference consistency	?
Bounded access time	?

Reference consistency

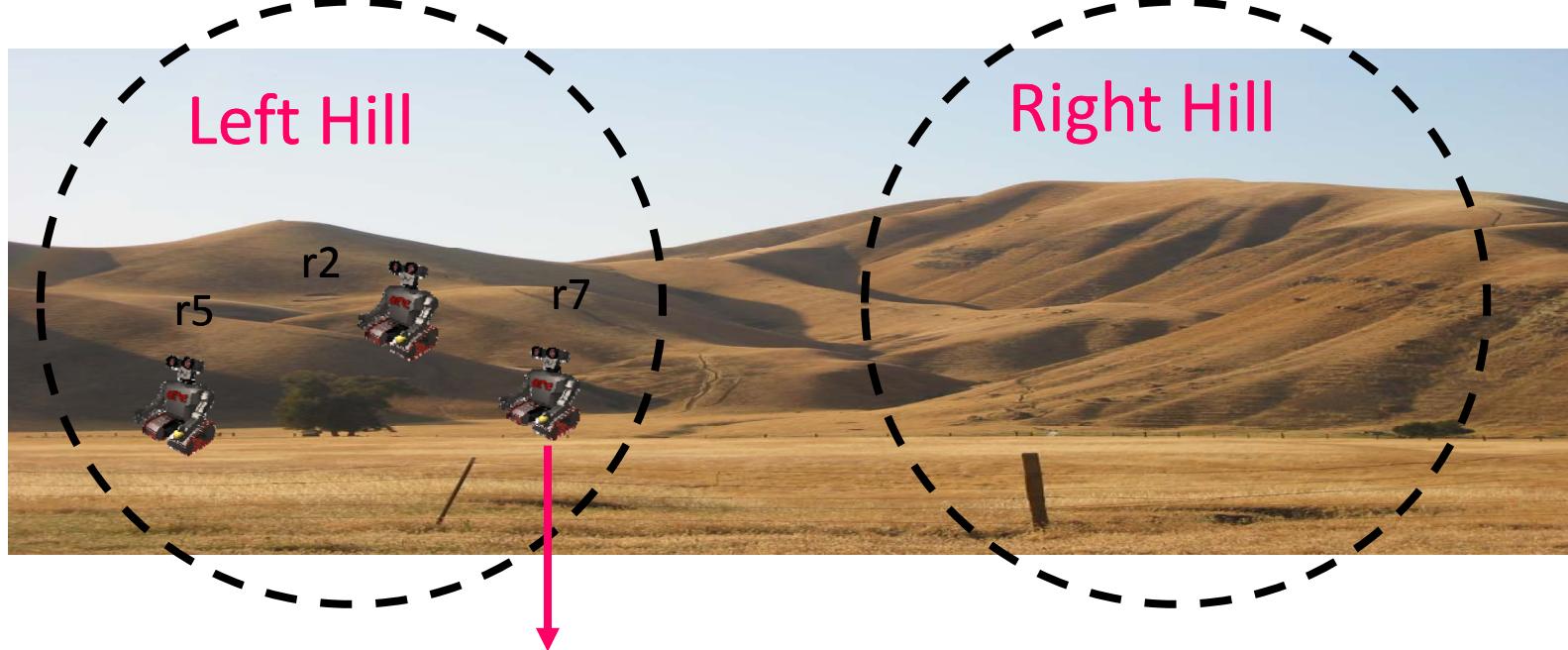
- At first access, a spatial reference is mapped to an embedded system located in the specified space
- Mappings maintained in per-application Mapping Table (MT) – similar to a page table

$\{\text{space, property, index}\} \rightarrow \{\text{unique_address, location}\}$

- Subsequent accesses to the same spatial reference will reach the same system (using MT) as long as it is located in the same space region

Reference consistency example

(1)



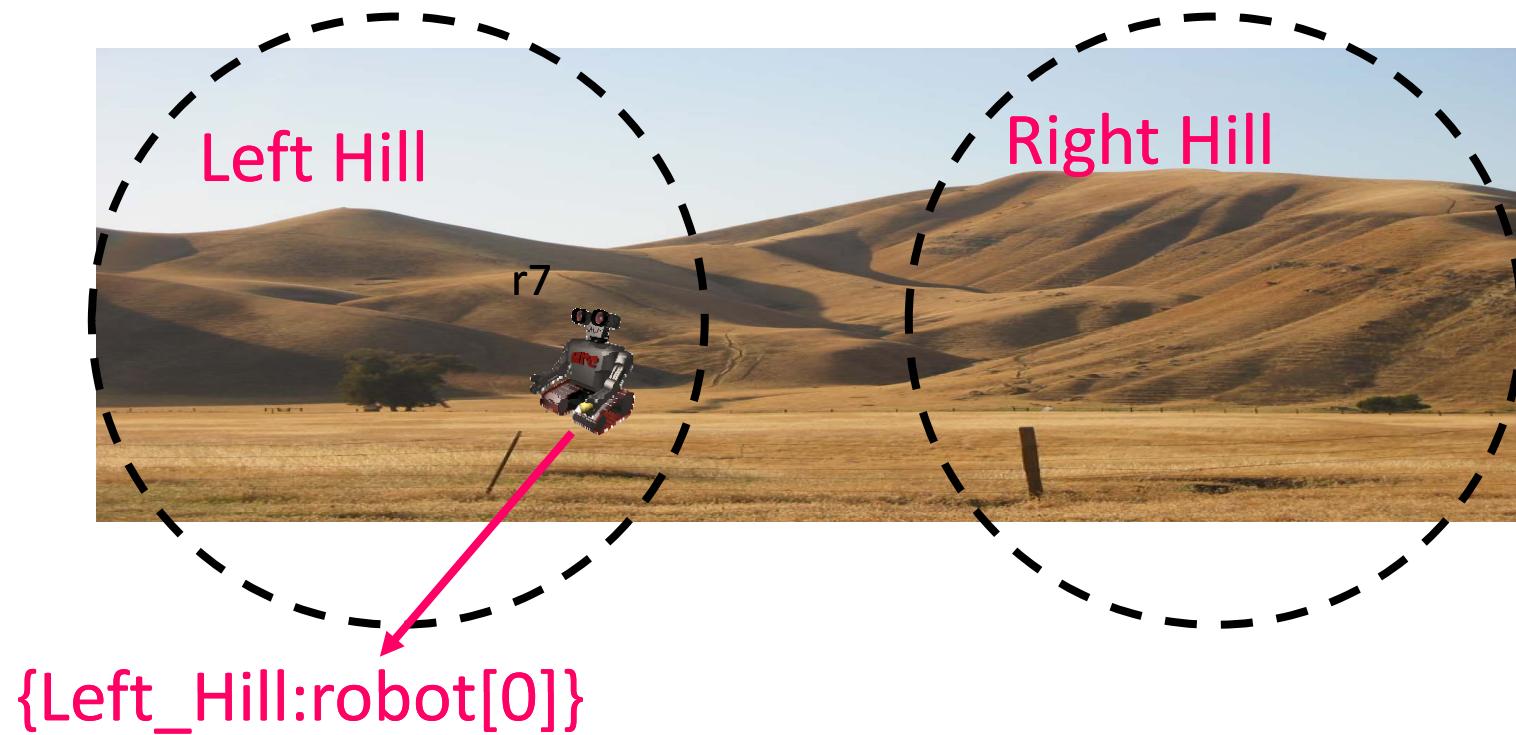
{Left_Hill:robot[0]}.move = ON;

Reference consistency example (2)

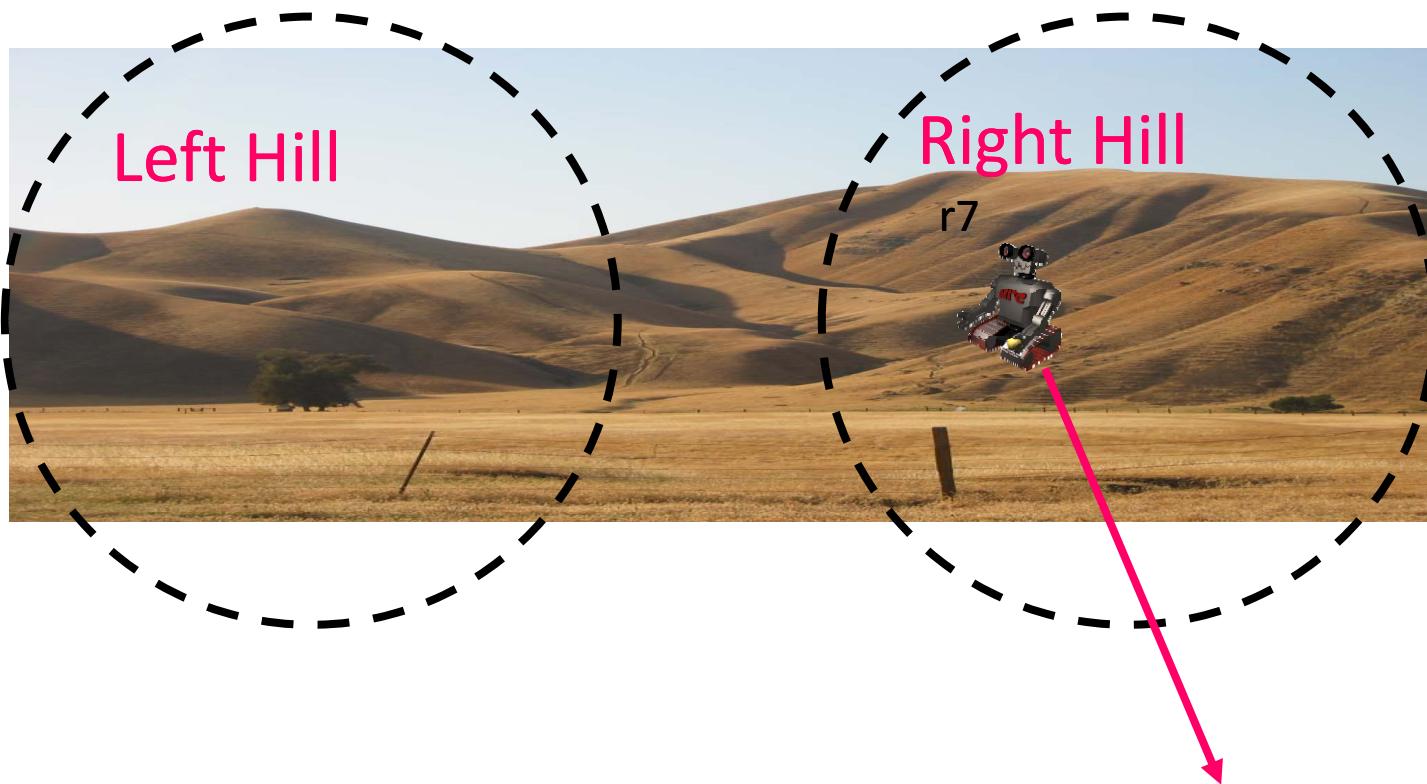


{Left_Hill:robot[0]}.move = OFF;

Space casting (1)



Space casting (2)



From indoor to outdoor computing

Virtual Address Space	Space Region
Variables	Spatial References
Variables mapped to physical memory	Spatial references mapped to systems embedded in the physical space
Reference consistency	Mapping Table
Bounded access time	?

Bounding the access time

- How to bound the time to access a spatial reference?
 - Systems may move, go out of space, or disappear
- Solution: associate an explicit timeout with each spatial reference access

```
try{
    {Hill:robot[0], timeout}.move = ON;
}catch(TimeoutException e){
    // the programmer decides the next action
}
```

Spatial programming example



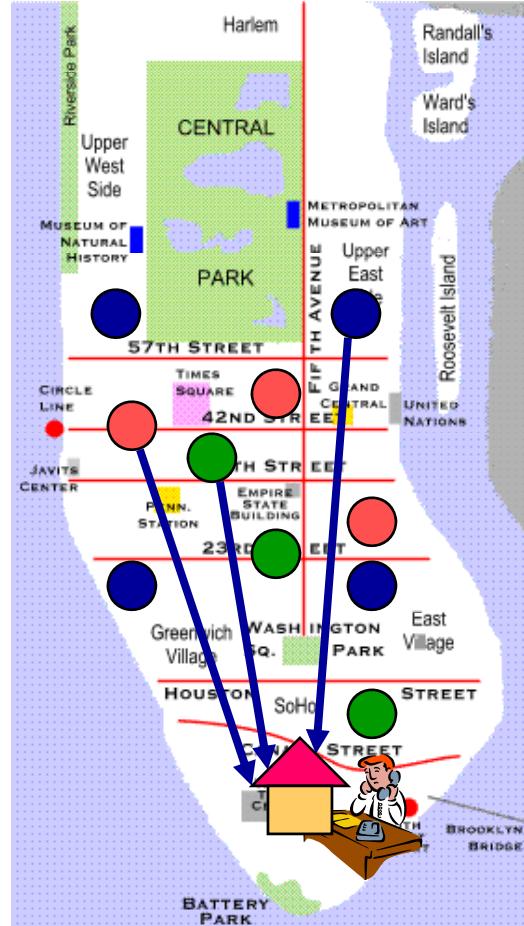
- Find the sensor that detected the “strongest” motion on Left Hill
- Turn on a camera in the proximity of this sensor

```
for(i=0; i<1000; i++)
try{
    if ({Left_Hill:motion[i], timeout}.detect > Max_motion)
        Max_motion = {Left_Hill:motion[i], timeout}.detect;
        Max_id = i;
}catch(TimeoutException e)
    break;
intrusionSpace = rangeOf({Left_Hill:motion[Max_id].location}, Range);
{intrusionSpace:robot[0]}.camera = ON;
{intrusionSpace:robot[0]}.focus = {Left_Hill:motion[Max_id].location};
```

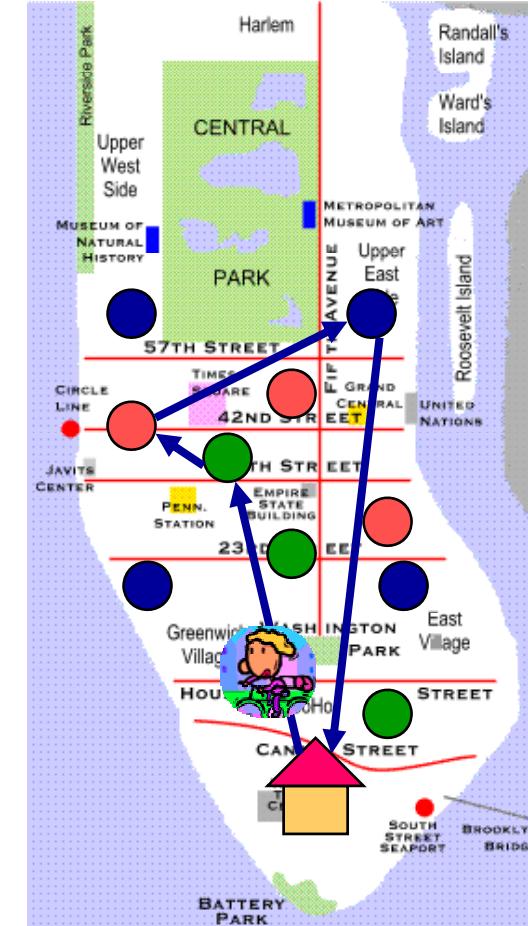
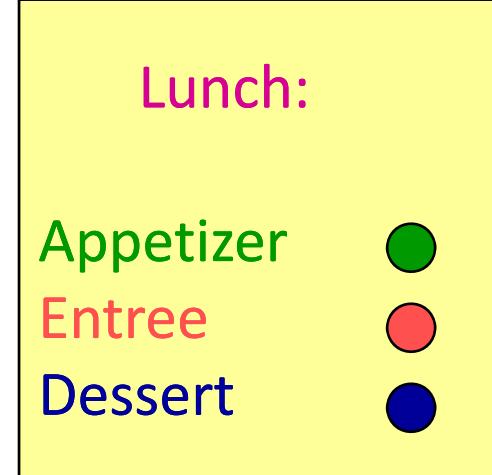
Spatial Programming implemented on top of Smart Messages

- Next
 - Overview of Smart Messages
 - Few implementation details

“Dumb” messages vs. “smart” messages



Data migration

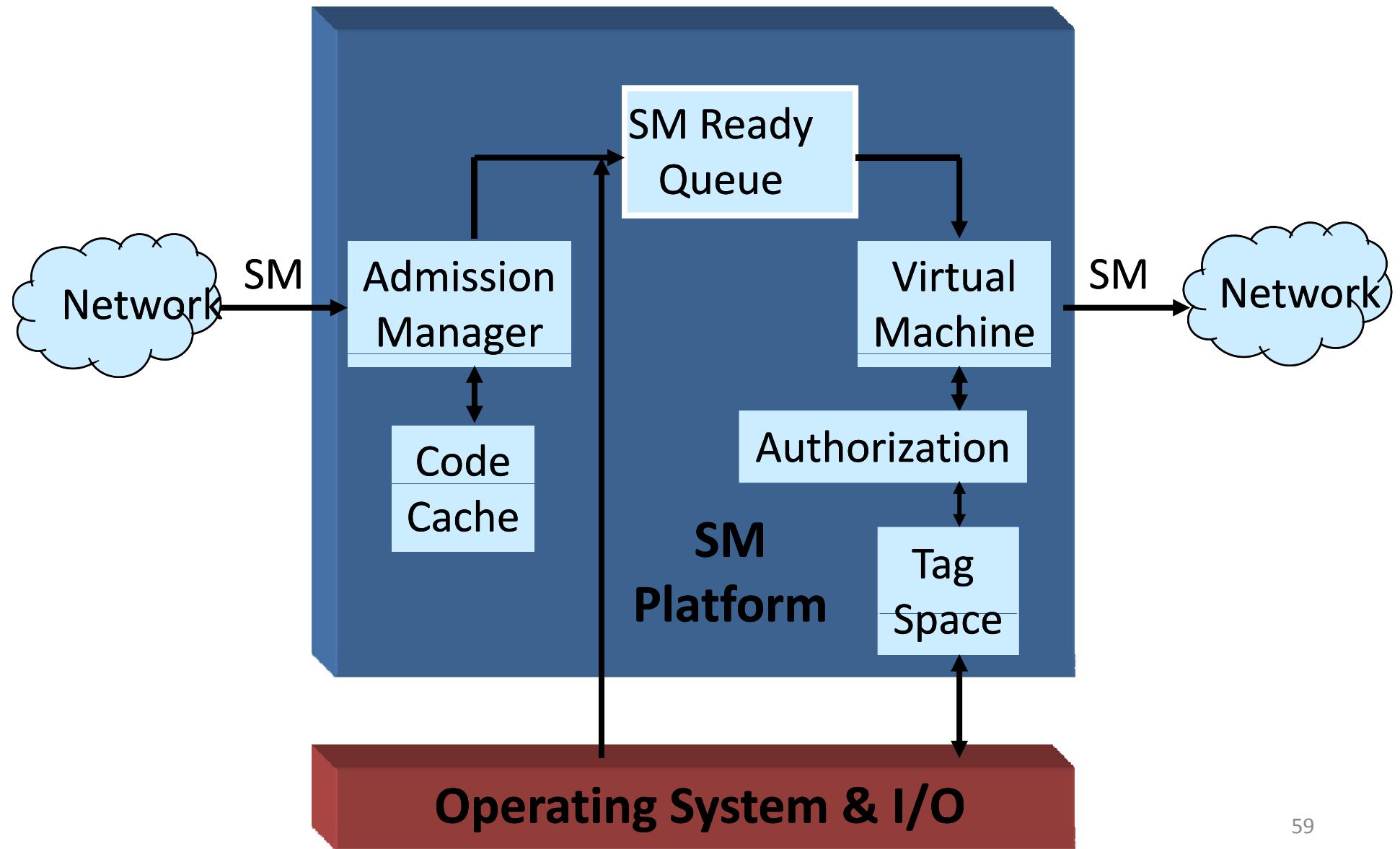


Execution migration

Smart Messages at a glance

- User-defined distributed applications
- Composed of code bricks, data bricks, and execution control state
- Execute on nodes of interest named by properties
- Migrate between nodes of interest
- Self-route at every node in the path during migrations

Cooperative node architecture



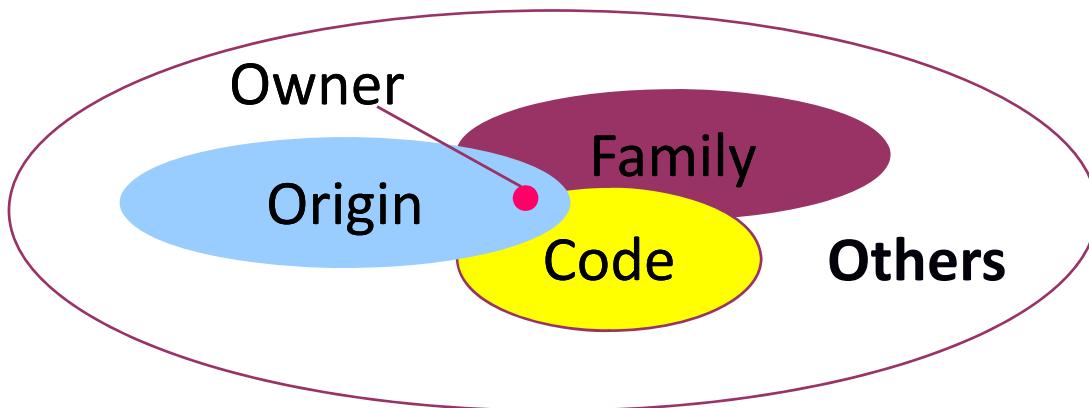
SM execution at a node

- Takes place over a virtual machine
- Non-preemptive, but time bounded
- Ends with a migration, or terminates
- During execution, SMs can
 - Spawn new SMs
 - Create smaller SMs out of their code and data bricks
 - Access the tag space
 - Block on a tag to be updated (update-based synchronization)

Tag space

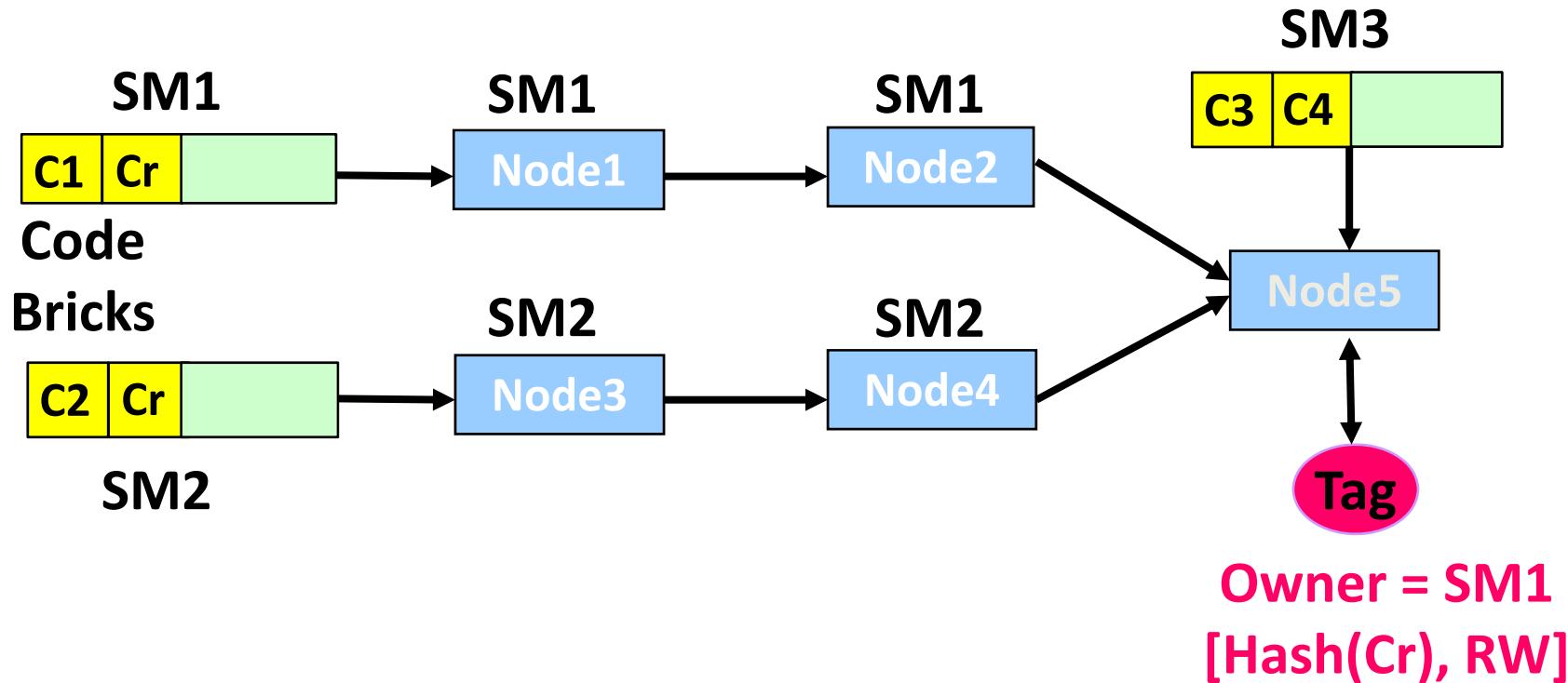
- Collection of application tags and I/O tags
 - Essentially, tags are *(name, value)* pairs
- Application tags: persistent memory across SM executions
- I/O tags: access to operating system and I/O subsystem
- Tags used for
 - Content-based naming migrate(tag)
 - Inter-SM communication write(tag, data), read(tag)
 - Synchronization block(tag, timeout)
 - I/O access read(temperature)

Protection domains for tag space



- Owner: SM that creates the tag
- Family: all SMs having a common ancestor with the SM that owns the tag
- Origin: all SMs created on the same node as the family originator of the tag owner
- Code: all SMs carrying a specific code brick
- Others: all the other SMs

Access control example (code-based protection domain)

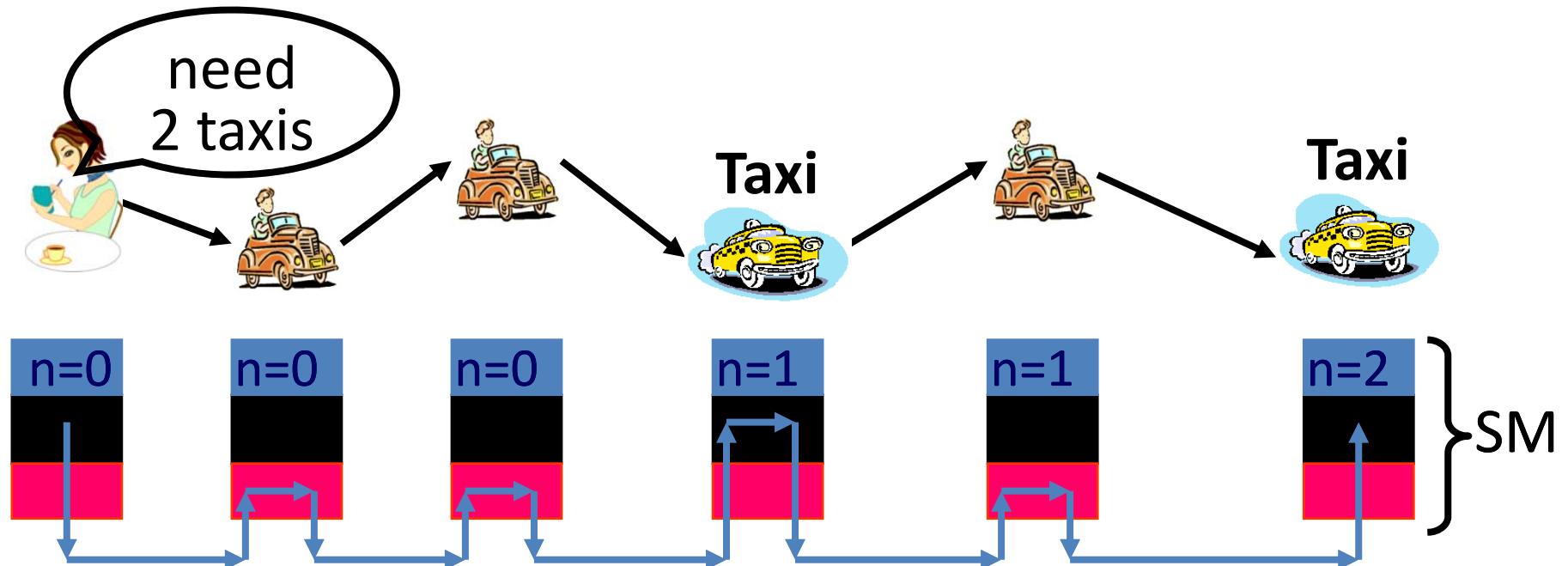


Cr = Same routing used by SM1 and SM2
Access permission granted for SM2
Access permission denied for SM3

SM admission

- Ensures progress for all SMs in the network
- Prevents SMs from migrating to nodes that cannot provide enough resources
- SMs specify lower bounds for resource requirements (e.g., memory, bandwidth)
- SMs accepted if the node can satisfy these requirements
- More resources can be granted according to admission policy
 - If not granted, SMs are allowed to migrate

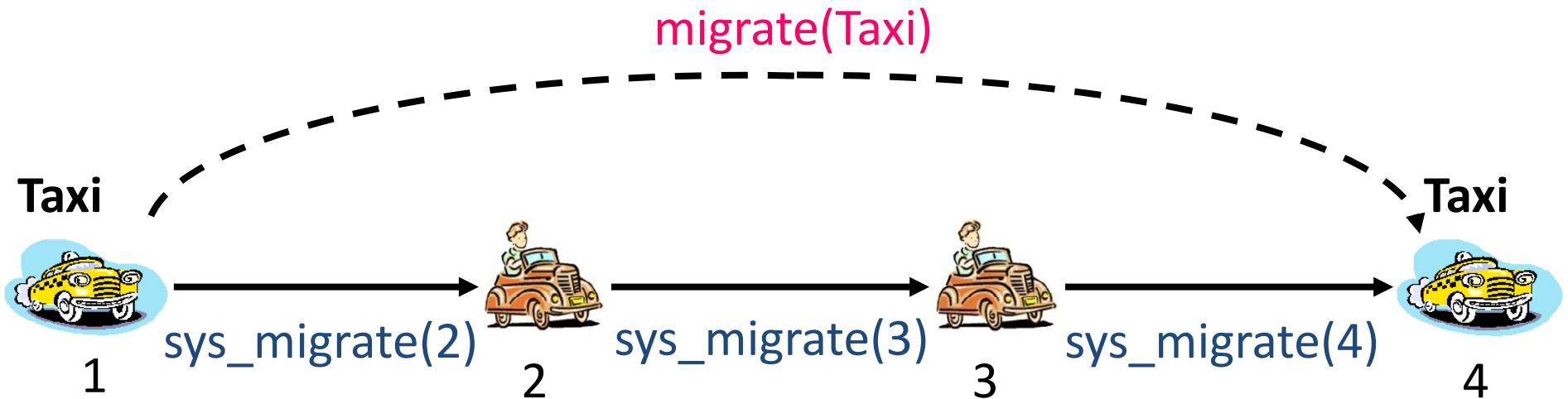
Application example



```
n=0  
while (n<NumTaxis)  
    migrate(Taxi);  
    if (readTag(Available))  
        writeTag(Available, false);  
        writeTag(Location, myLocation);  
    n++;
```

■ data brick
■ application code brick
■ routing code brick

SM migration



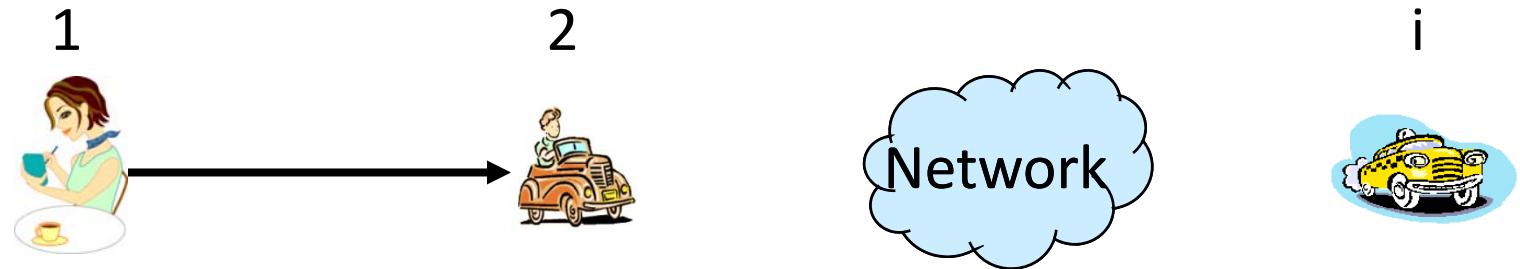
■ **migrate()**

- migrates application to next node of interest
- names nodes by tags
- implements routing algorithm

■ **sys_migrate()**

- one hop migration
- used by migrate to implement routing

Routing example



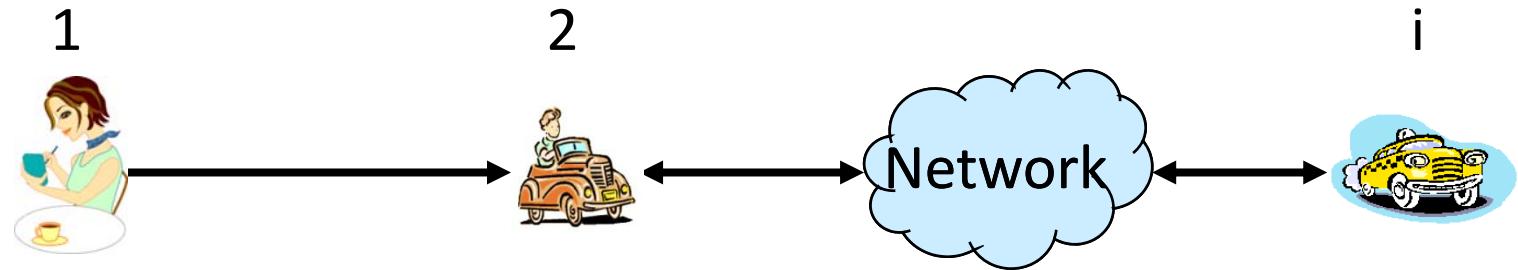
RouteToTaxi = 2

RouteToTaxi = ?

Taxi

```
migrate(Taxi){  
    while(readTag(Taxi) == null)  
        if (readTag(RouteToTaxi))  
            sys_migrate(readTag(RouteToTaxi));  
        else  
            create_SM(DiscoverySM, Taxi);  
            createTag(RouteToTaxi, lifetime, null);  
            block_SM(RouteToTaxi, timeout);  
}
```

Routing example



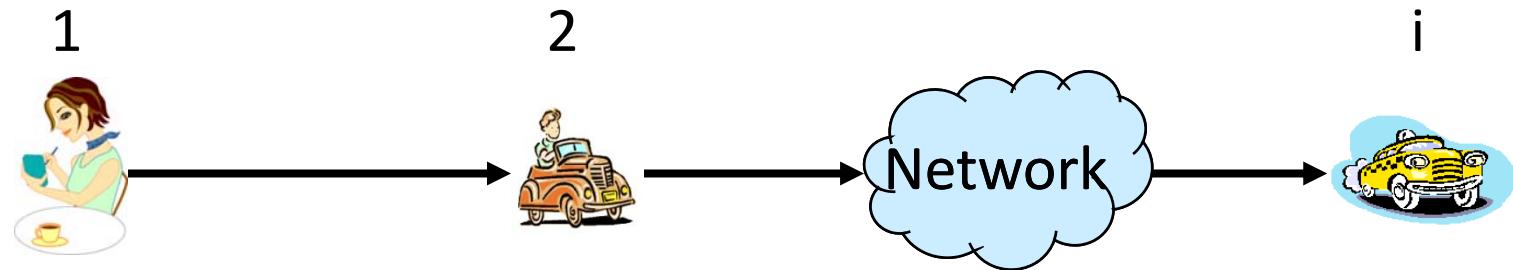
RouteToTaxi = 2

RouteToTaxi = j

Taxi

```
migrate(Taxi){  
    while(readTag(Taxi) == null)  
        if (readTag(RouteToTaxi))  
            sys_migrate(readTag(RouteToTaxi));  
        else  
            create_SM(DiscoverySM, Taxi);  
            createTag(RouteToTaxi, lifetime, null);  
            block_SM(RouteToTaxi, timeout);  
    }  
}
```

Routing example



RouteToTaxi = 2

RouteToTaxi = j

Taxi

```
migrate(Taxi){  
    while(readTag(Taxi) == null)  
        if (readTag(RouteToTaxi))  
            sys_migrate(readTag(RouteToTaxi));  
        else  
            create_SM(DiscoverySM, Taxi);  
            createTag(RouteToTaxi, lifetime, null);  
            block_SM(RouteToTaxi, timeout);  
    }  
}
```

SM self-routing

- SMs carry the routing and execute it at each node
- Routing information stored in tag space
- SMs control their routing
 - Select routing algorithm (migrate primitive)
 - Multiple library implementations
 - Implement a new one
 - Change routing algorithm during execution in response to
 - Adverse network conditions
 - Application's requirements

Spatial programming using Smart Messages

- SP application translates into an SM
 - Spatial reference access translates into an SM migration to the mapped node
 - Embedded system properties: Tags
- SM self-routes using geographical routing and content-based routing
- Reference consistency
 - Unique addresses (stored in mapping table) are unique tags created at nodes
 - SM carries the mapping table

SP to SM translation: example



Spatial Reference Access



Smart Message

Mapping Table
Code Brick

```
Max_motion = {Left_Hill:motion[1], timeout}.detect;
```

```
{Left_Hill,motion,1} →{yU78GH5,location}
```

```
ret = migrate_geo(location, timeout);
if (ret == LocationUnreachable)
    ret = migrate_tag(yU78GH5, timeout);
if ((ret == OK) && (location == Left_Hill))
    return readTag(det);
else throw TimeoutException
```

Prototype implementation

- SM implemented over modified version of Sun's Java K Virtual Machine
 - Small memory footprint (160KB)
- SM and tag space primitives implemented inside virtual machine as native methods (efficiency)
- I/O tags: GPS location, neighbor discovery, image capture, light sensor, system status



Ad hoc networks of
HP iPAQ PDAs running Linux

Lightweight migration

- Traditional process migration difficult
 - Strong coupling between execution entity and host
 - Needs to take care of operating system state (e.g., open sockets, file descriptors)
- Tag space decouples the SM execution state from the operating system state
- SM migration transfers only
 - Data bricks explicitly specified by programmer
 - Minimal execution control state required to resume the SM at the next instruction (e.g., instruction pointer, operand stack pointer)

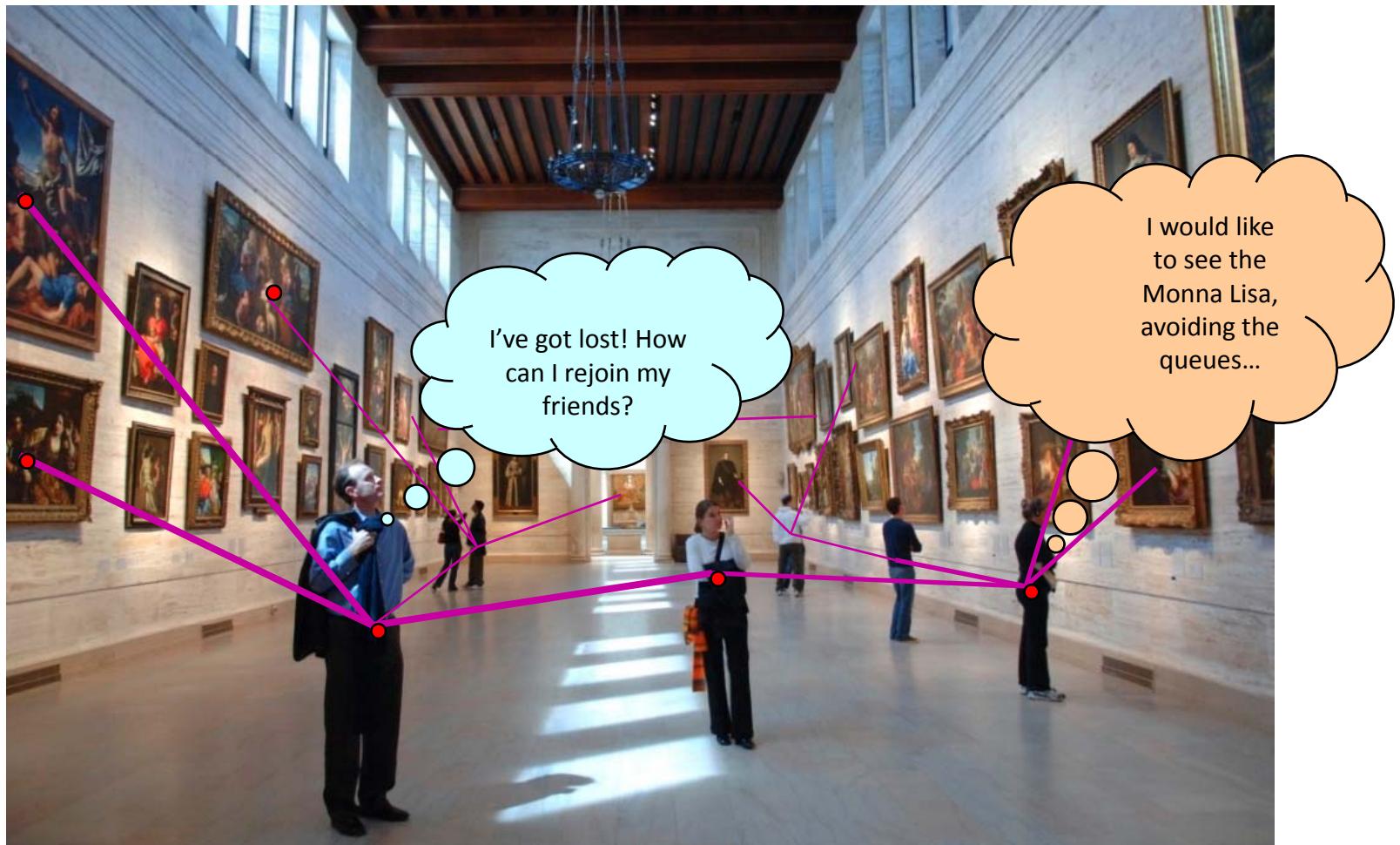
Summary

- Spatial Programming makes outdoor distributed computing simple
 - Volatility, mobility, configuration dynamics, ad hoc networking are hidden from programmer
- Implementation on top of Smart Messages
 - Easy to deploy new applications in the network
 - Quick adaptation to highly dynamic network configurations
- Acknowledgments: Liviu Iftode, Porlin Kang

Syllabus

- Pervasive Computing, Spatial Computing
 - Example Scenarios
- Overview of Spatial Approaches
- Three spatial programming models:
 - Spatial Programming / Smart Messages
 - TOTA / Field-based Coordination
 - Proto / Amorphous Medium abstraction

Museum case study application

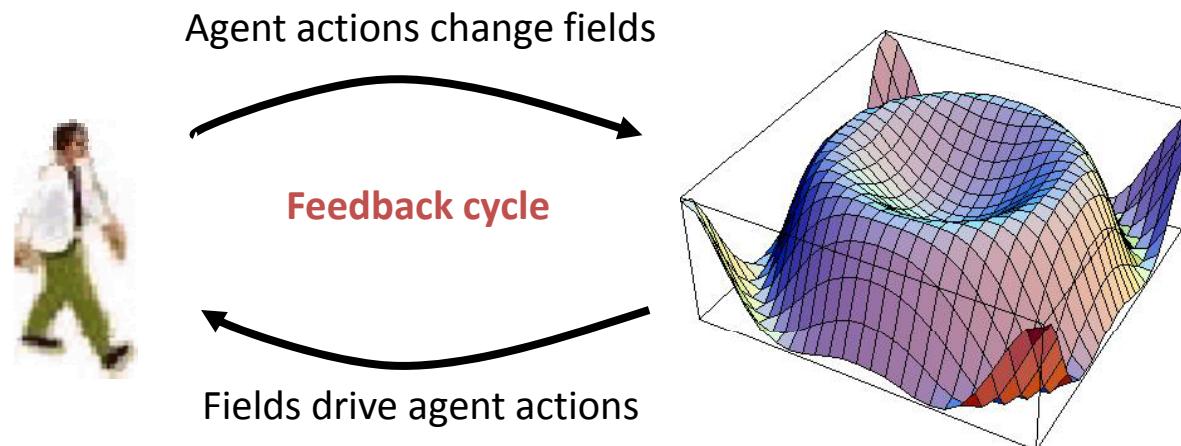


The meeting task

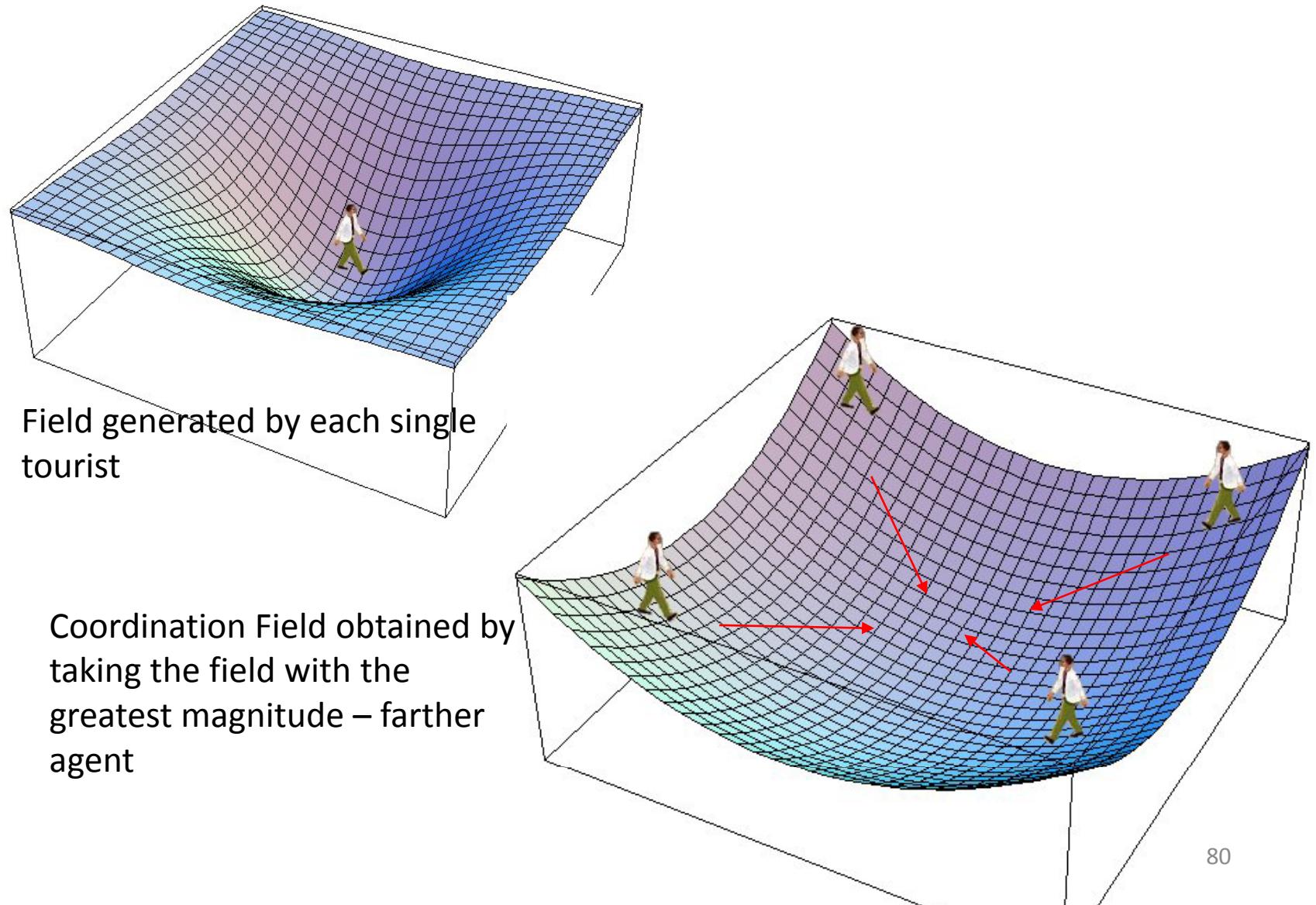
- A Group a tourist want to meet somewhere within the building.
- The “**classic**” solution – general context information
 - Involve **rather complex algorithms** to decide what to do
 - If something changes (e.g. a corridor is closed) the solution has to be recomputed
- Our proposal
 - Express context by means of a “**red carpet**” leading to the meeting room.
 - How to represent the “red carpet”?

The Co-Fields model

- Contextual information is expressed by means of **distributed data-structures spread in the environment** (i.e. fields). Agent move and act being driven by these fields
- Agents' combine perceived fields in a, so called, **coordination field** that encodes their actual application task. Then, they act being driven by the resulting values and gradient.
- The coordination policy is encoded into fields' waveform, in the way in which agents combine perceived fields and in the way in which they react to the fields.



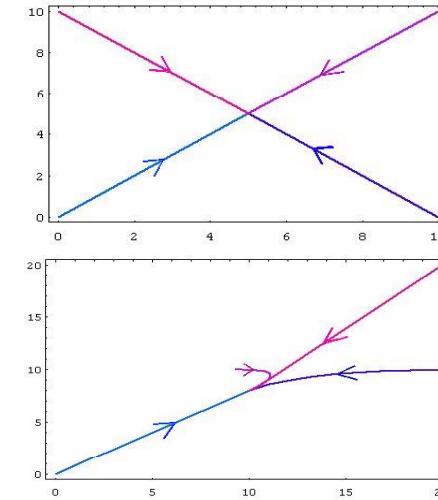
Meeting in Co-Fields



Benefits

- Designing the application becomes trivial. Just compute the coordination field and move following the gradient downhill.

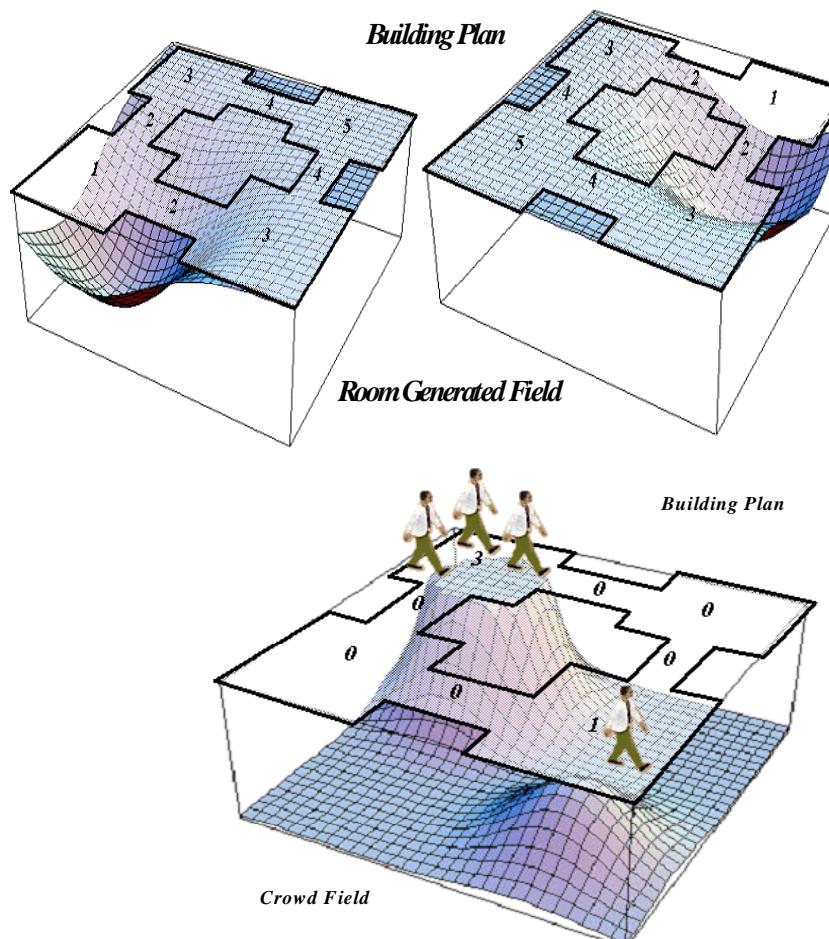
$$\frac{dx_j}{dt} = \pm v \cdot \frac{\partial \text{coord}_i(X_1, X_2, \dots, X_n, t)}{\partial X_j}$$



- The solution is adaptive. If a tourist gets trapped somewhere, then the meeting room automatically get closer to the unlucky tourist.
- Supposing that fields distributed data structures are maintained coherent despite environment dynamism, then the application is robust to closed corridors, etc.

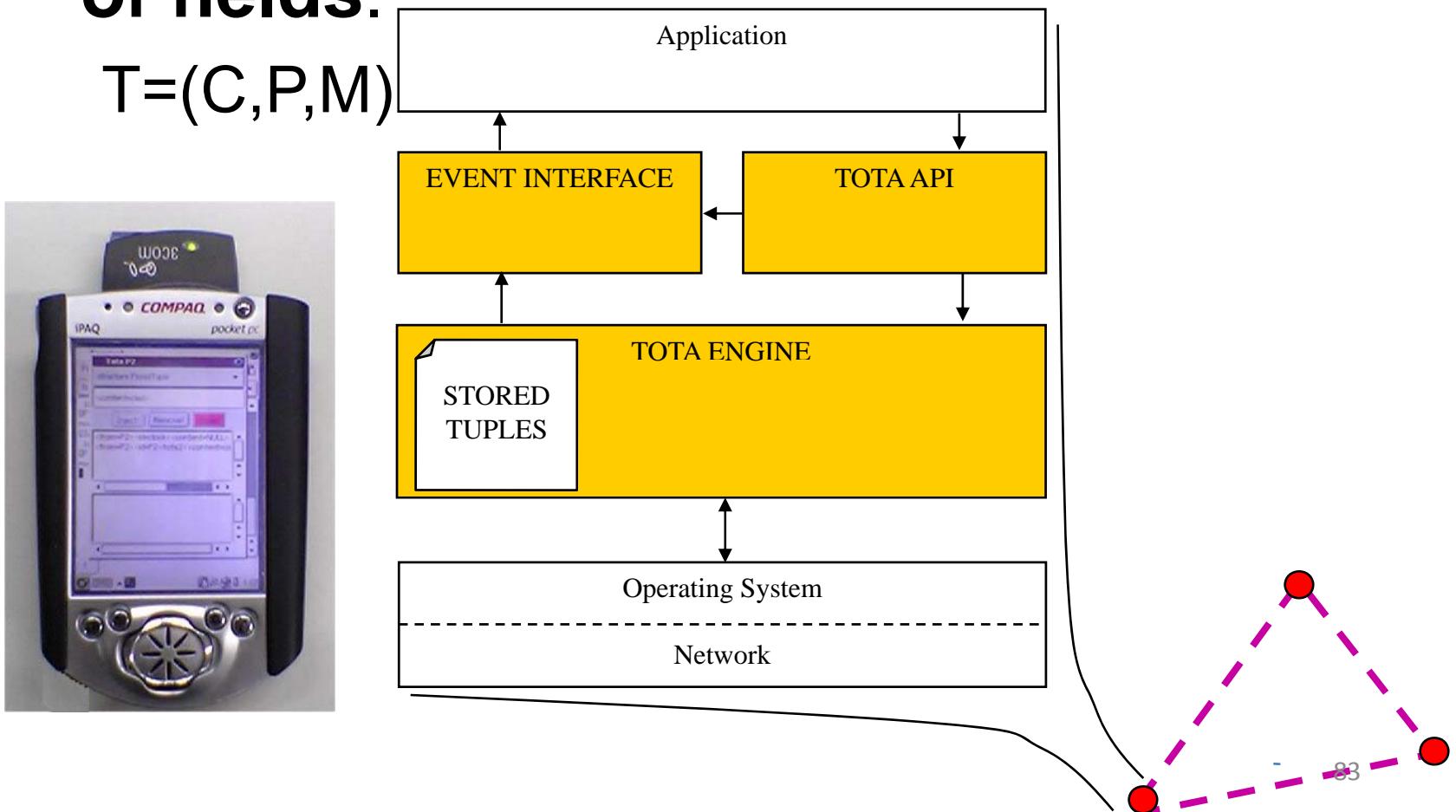
Other applications

- Avoid queues



The TOTA middleware

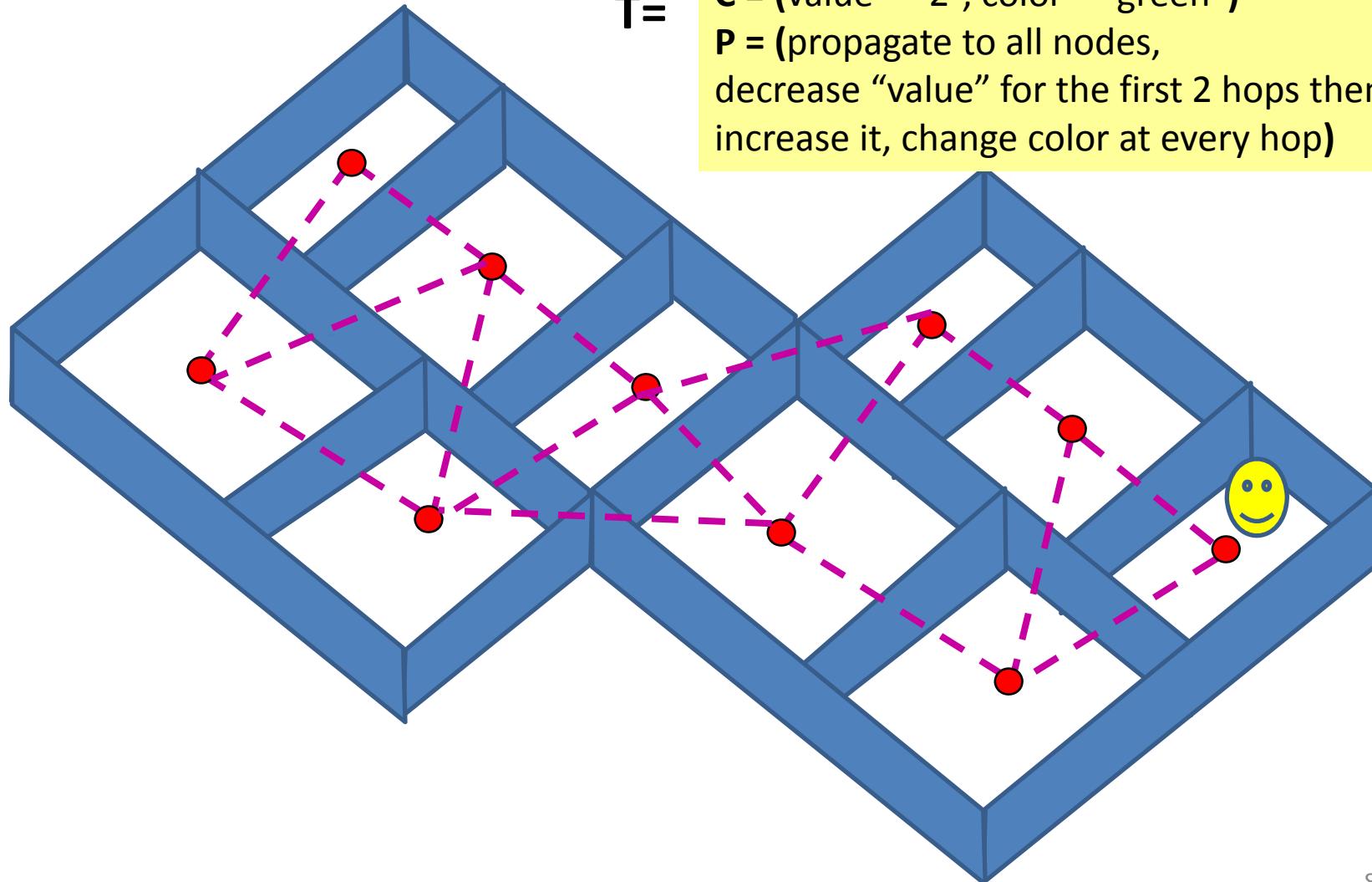
- Distributed tuples injected and spread in the network **implementing the concept of fields.**



The TOTA scenario

T =

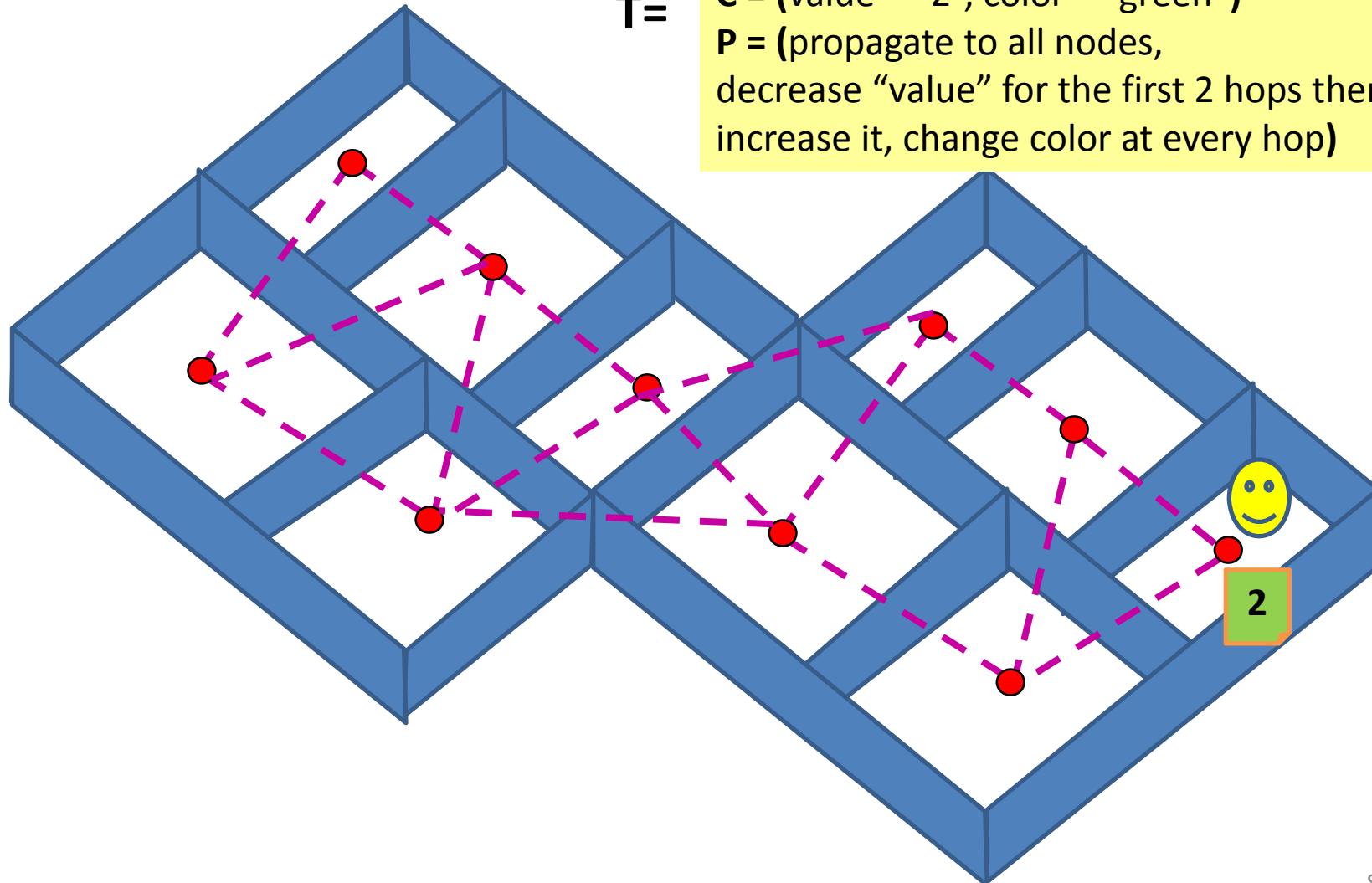
C = (value = "2", color = "green")
P = (propagate to all nodes,
decrease "value" for the first 2 hops then
increase it, change color at every hop)



The TOTA scenario

T =

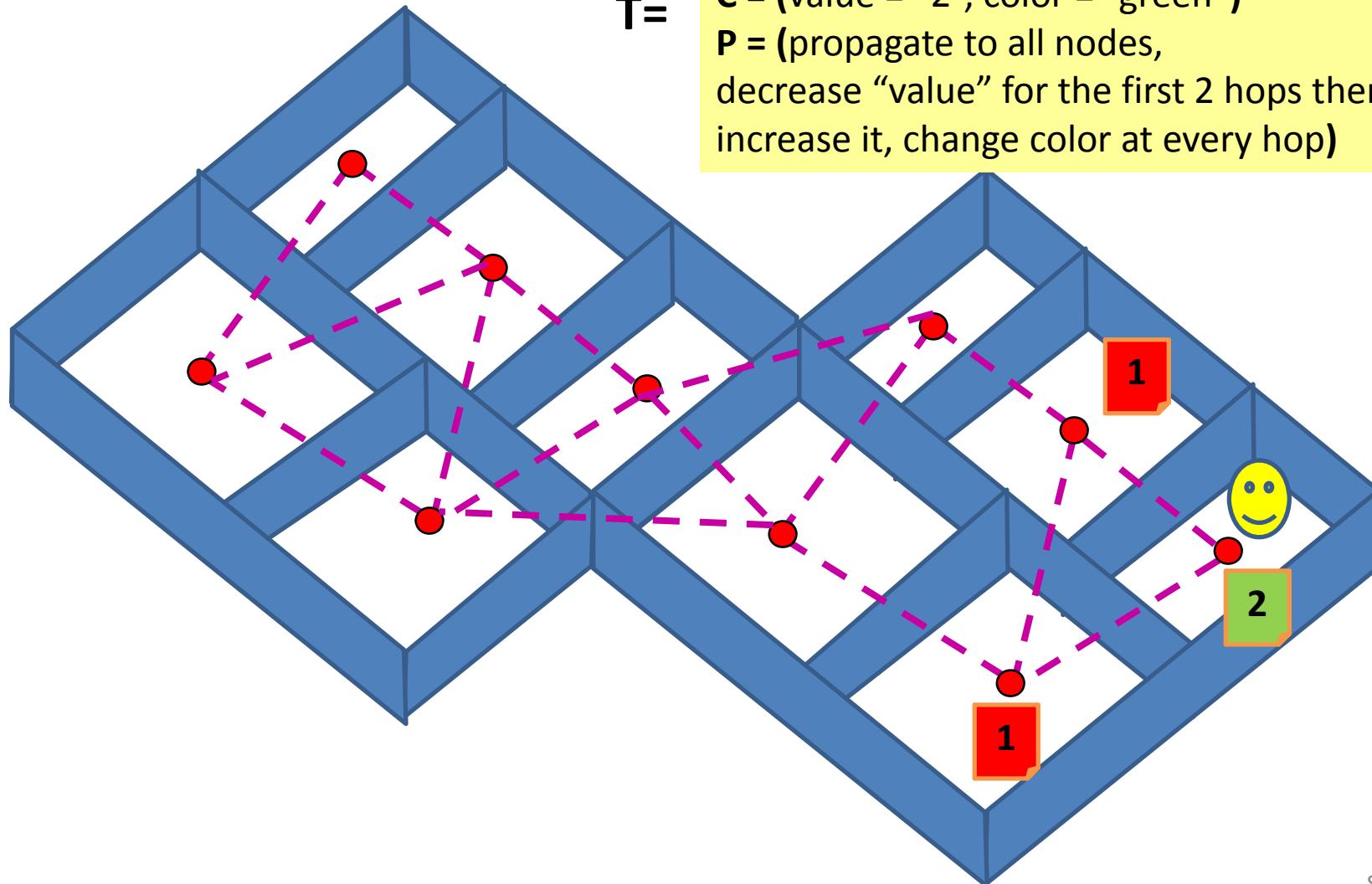
C = (value = "2", color = "green")
P = (propagate to all nodes,
decrease "value" for the first 2 hops then
increase it, change color at every hop)



The TOTA scenario

T=

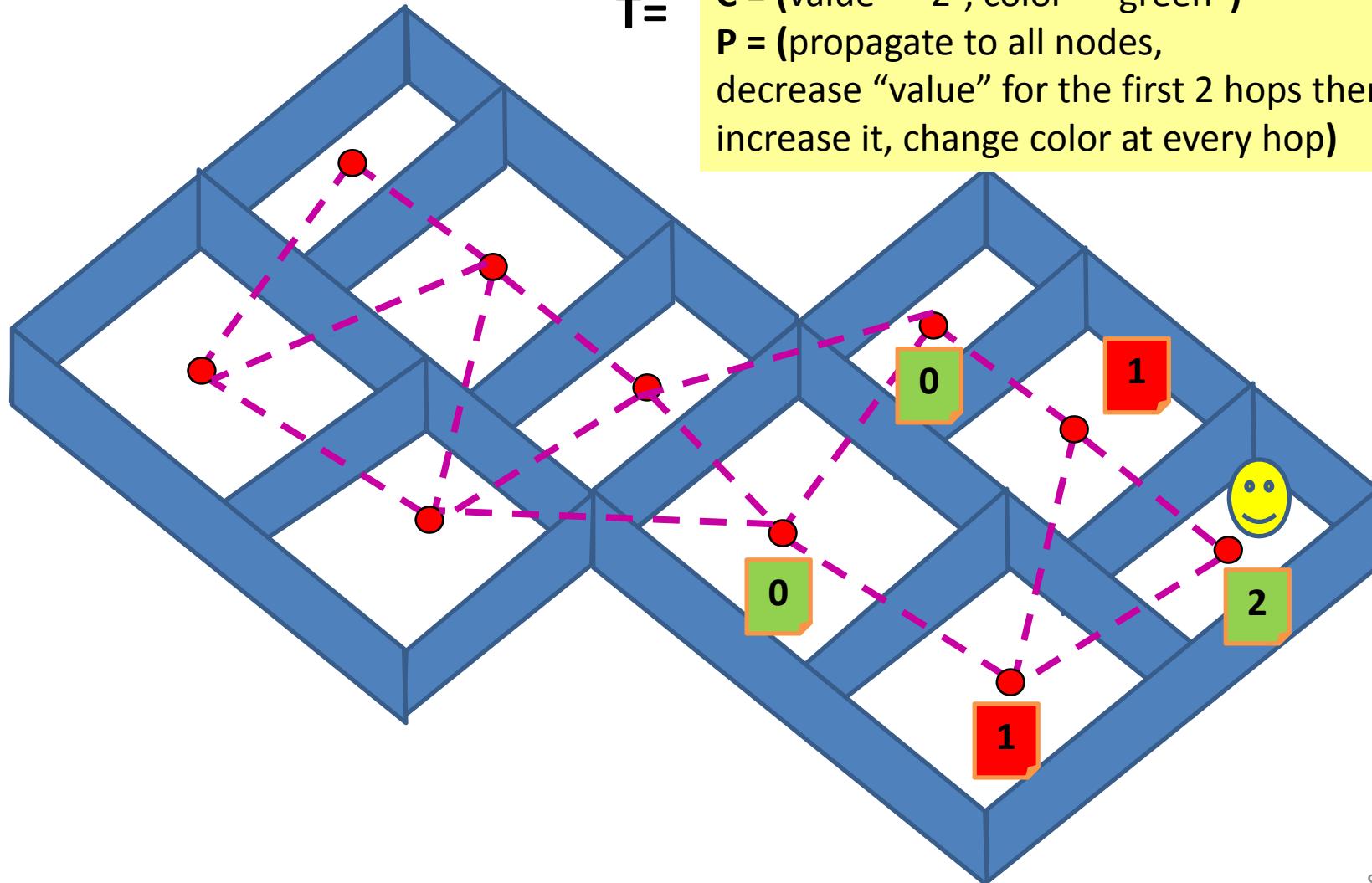
C = (value = "2", color = "green")
P = (propagate to all nodes,
decrease "value" for the first 2 hops then
increase it, change color at every hop)



The TOTA scenario

T =

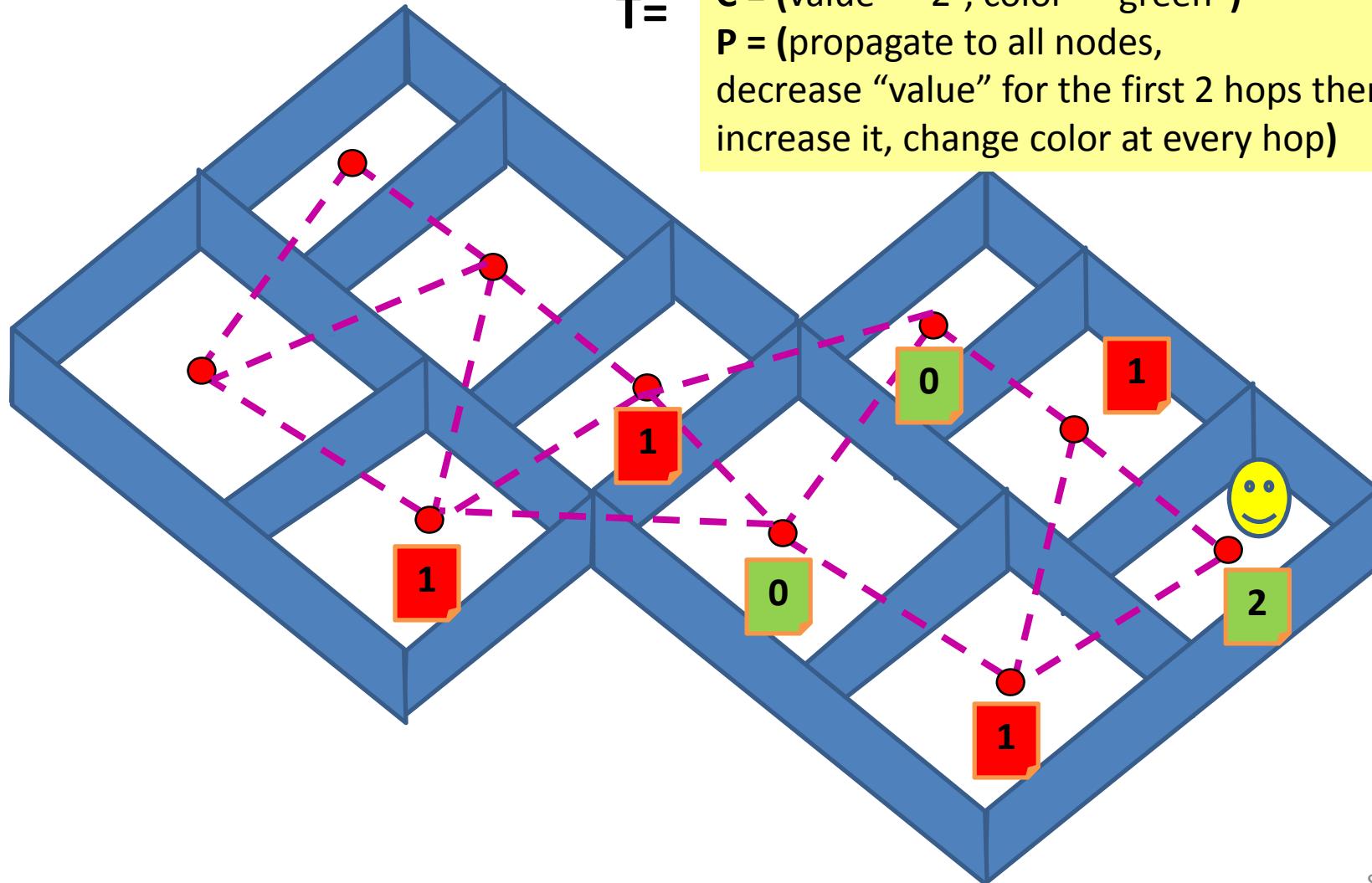
C = (value = "2", color = "green")
P = (propagate to all nodes,
decrease "value" for the first 2 hops then
increase it, change color at every hop)



The TOTA scenario

T=

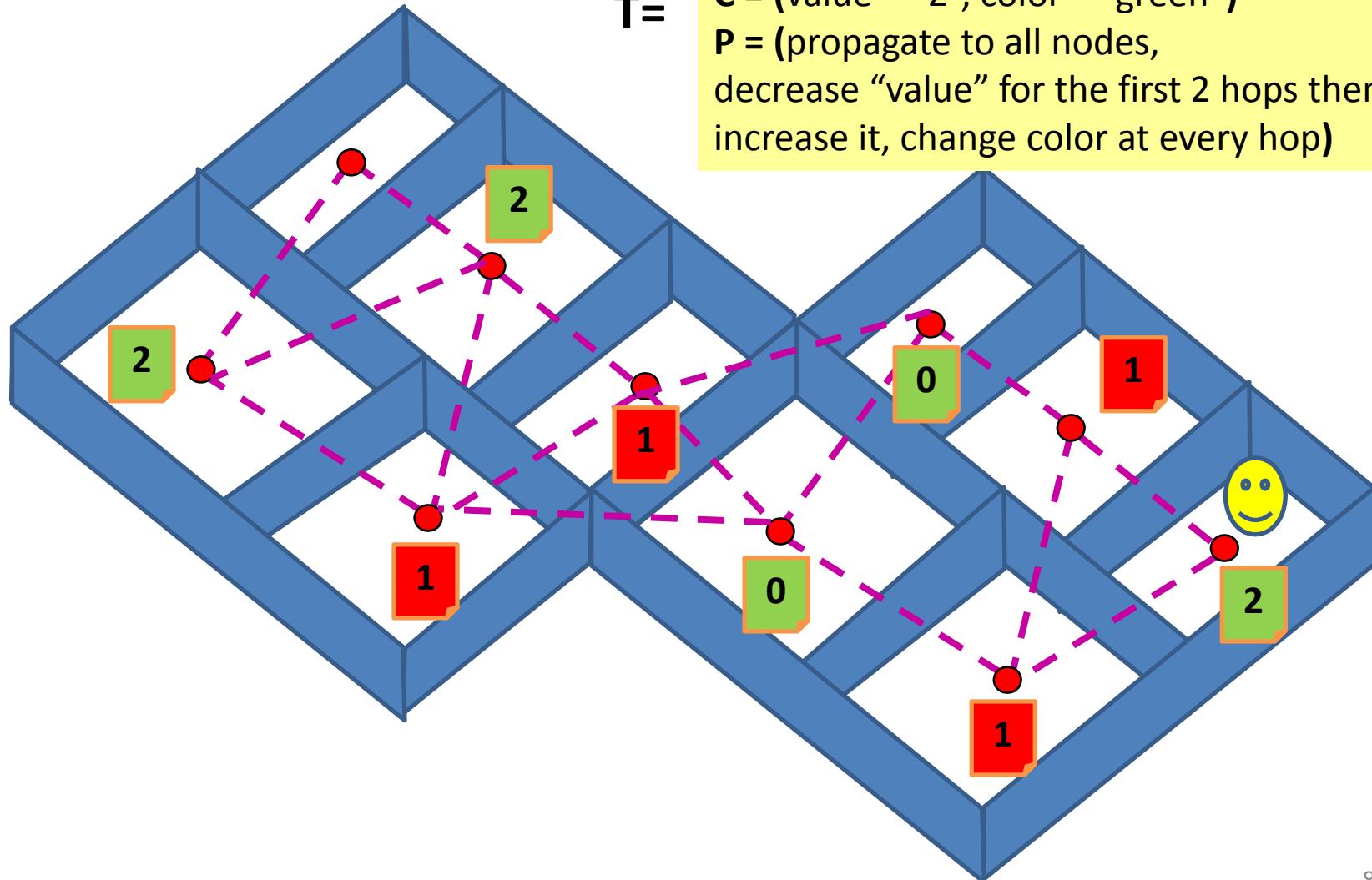
C = (value = "2", color = "green")
P = (propagate to all nodes,
decrease "value" for the first 2 hops then
increase it, change color at every hop)



The TOTA scenario

T=

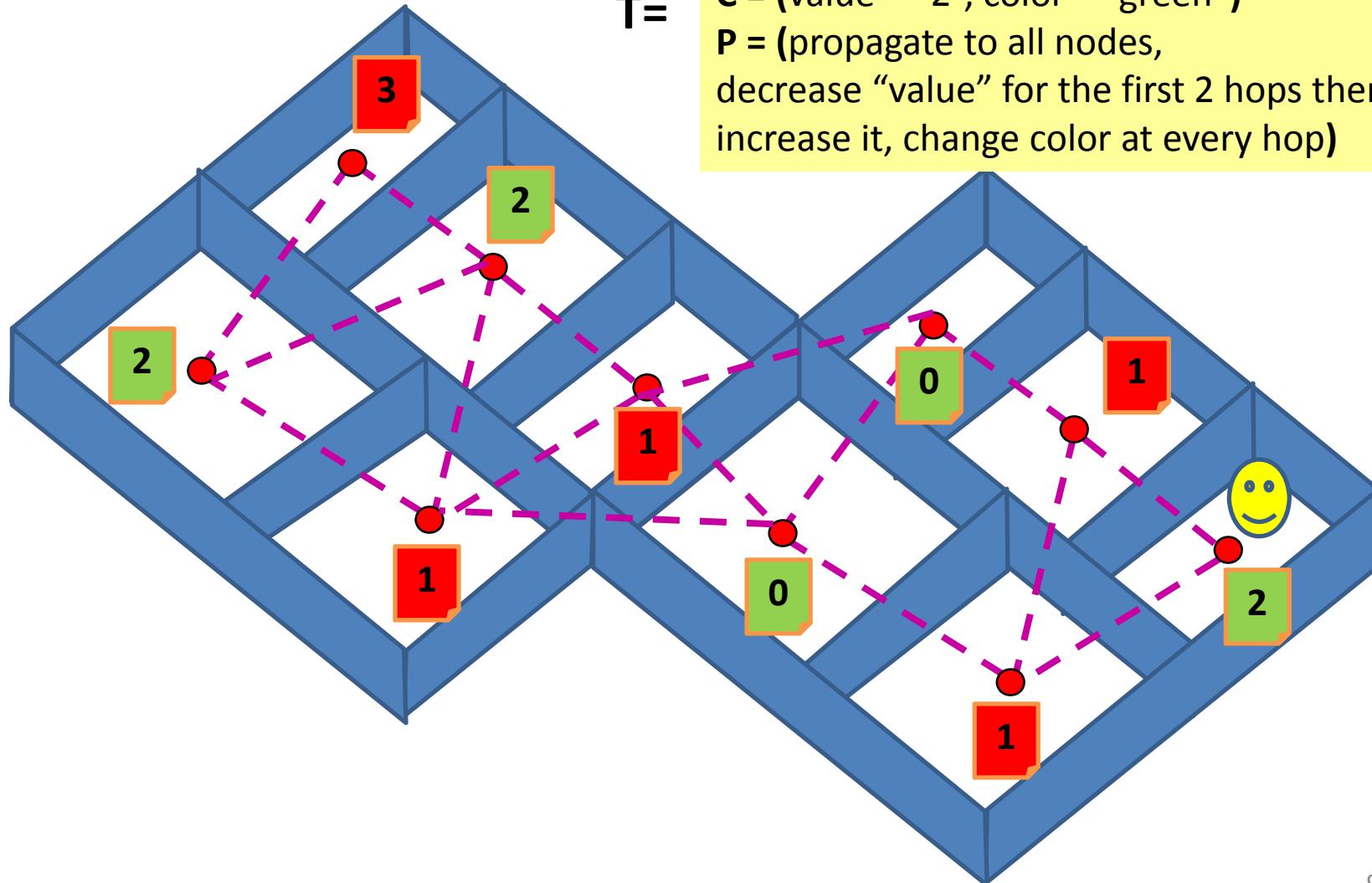
C = (value = "2", color = "green")
P = (propagate to all nodes,
decrease "value" for the first 2 hops then
increase it, change color at every hop)



The TOTA scenario

T=

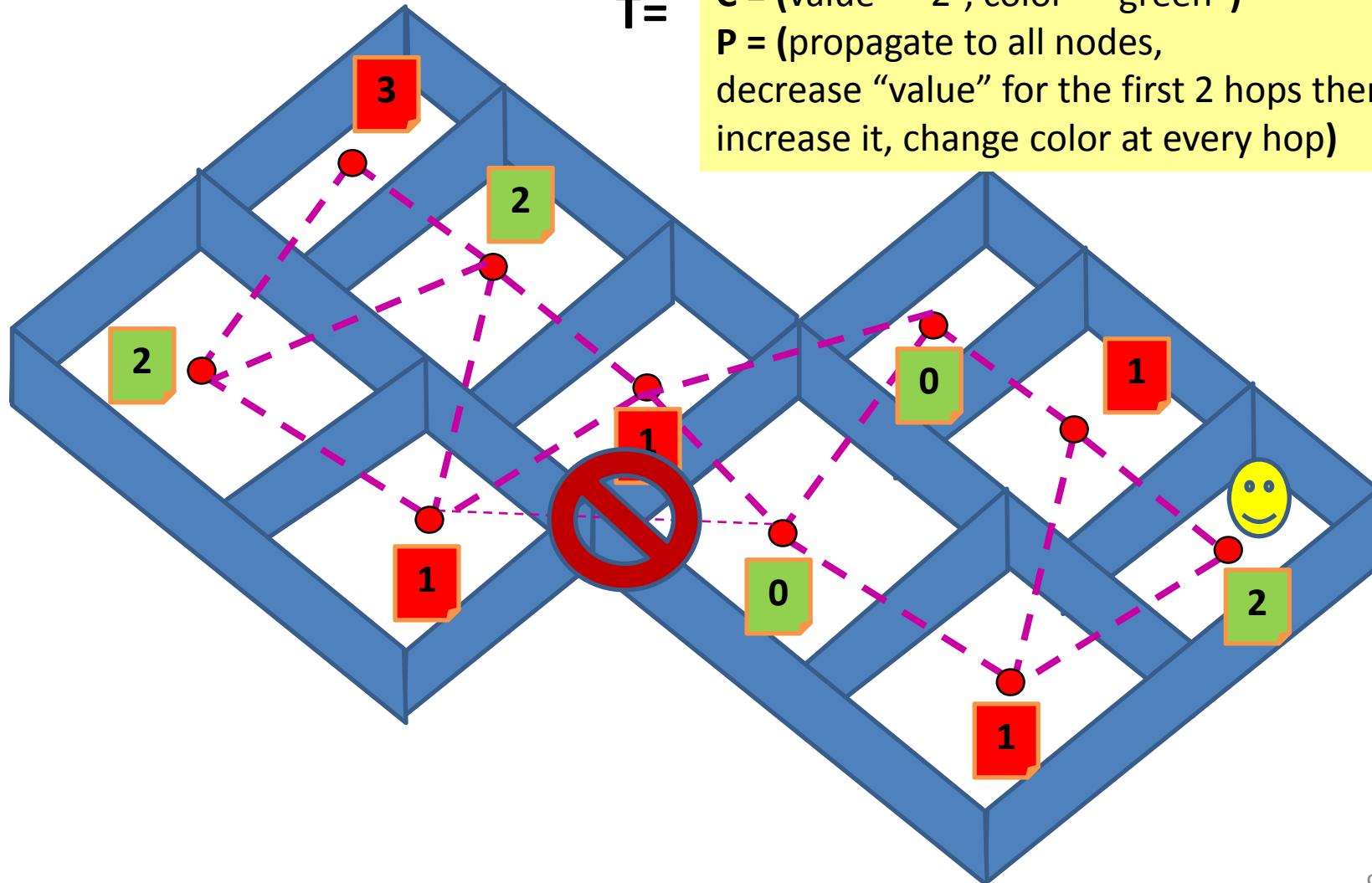
C = (value = "2", color = "green")
P = (propagate to all nodes,
decrease "value" for the first 2 hops then
increase it, change color at every hop)



The TOTA scenario

T=

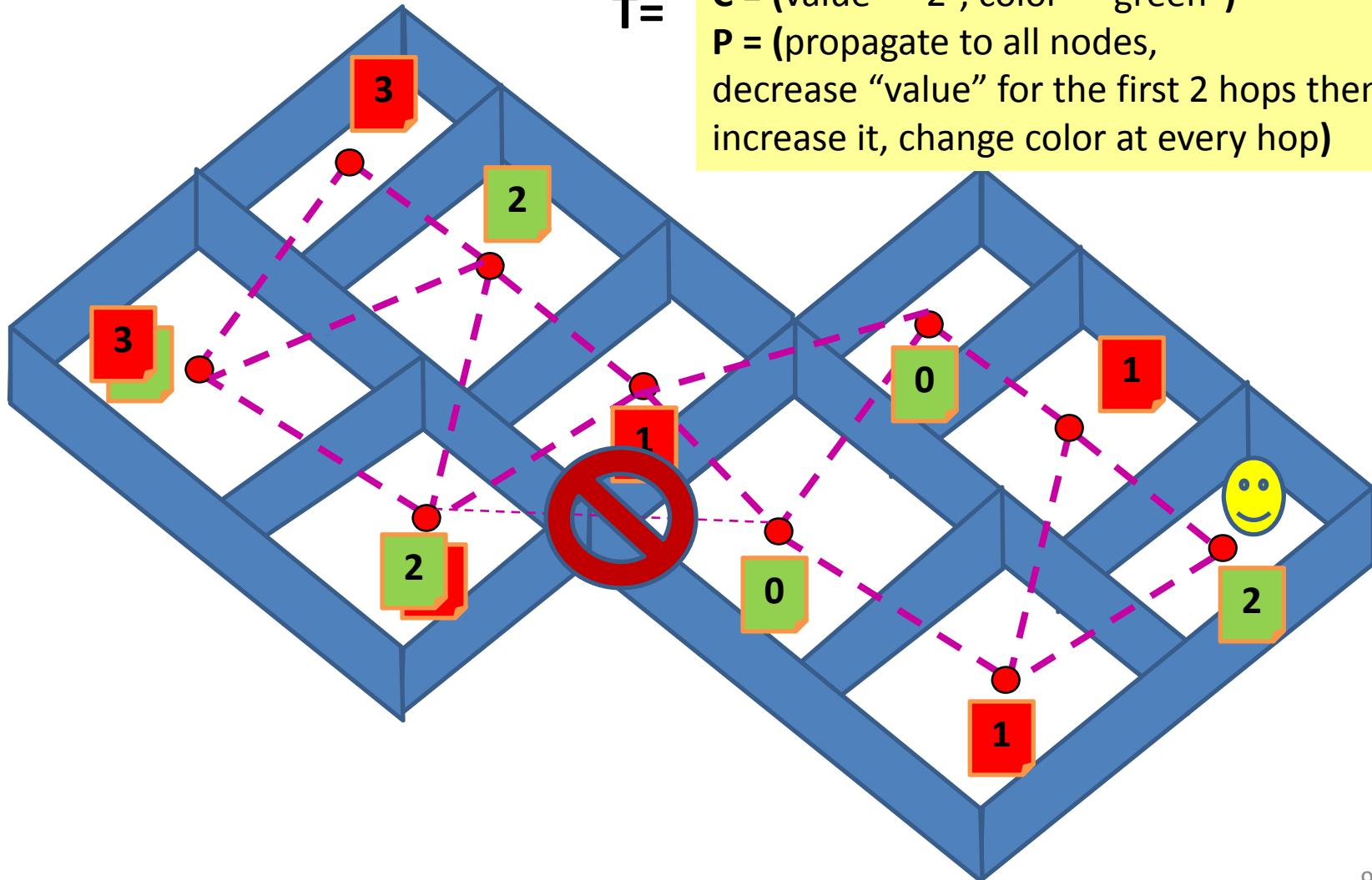
C = (value = "2", color = "green")
P = (propagate to all nodes,
decrease "value" for the first 2 hops then
increase it, change color at every hop)



The TOTA scenario

T =

C = (value = "2", color = "green")
P = (propagate to all nodes,
decrease "value" for the first 2 hops then
increase it, change color at every hop)



TOTA main algorithm

- Propagation is **easy**
 - Breadth-first, avoiding backward propagation
- Maintenance is **difficult**
 - We do not want to periodically re-propagate
 - We wish maintenance is localized near the point where the network changes

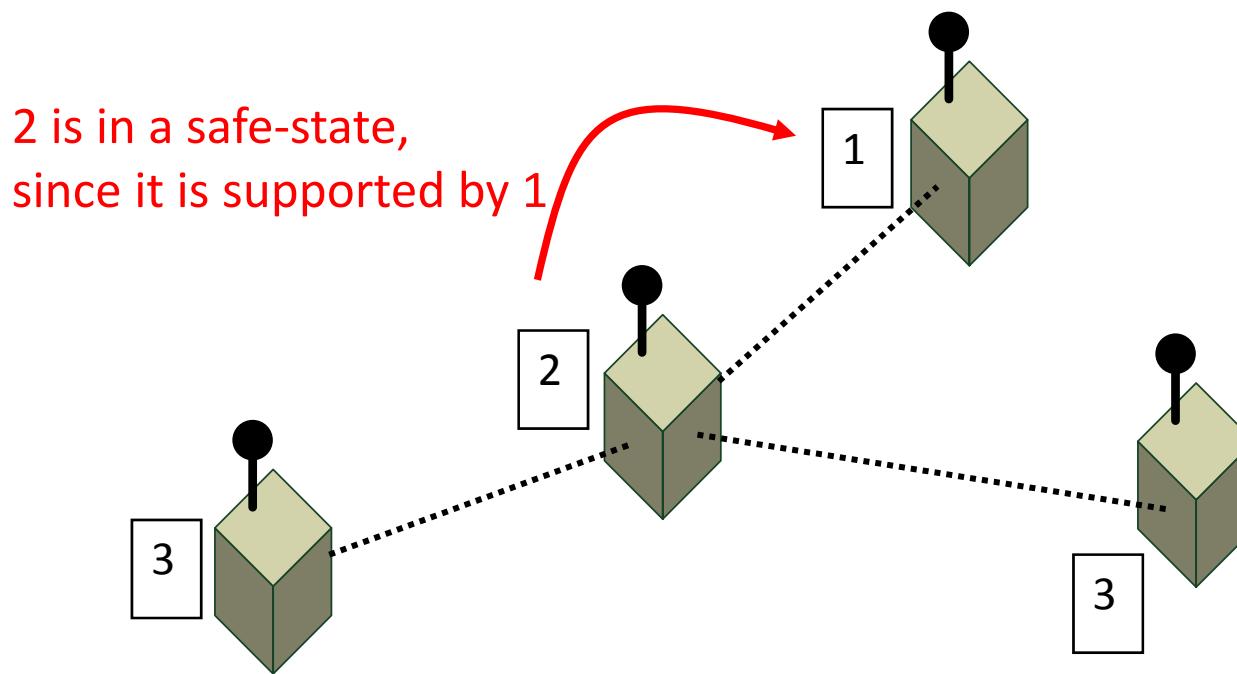
Self-Maintenance

EXAMPLE 1

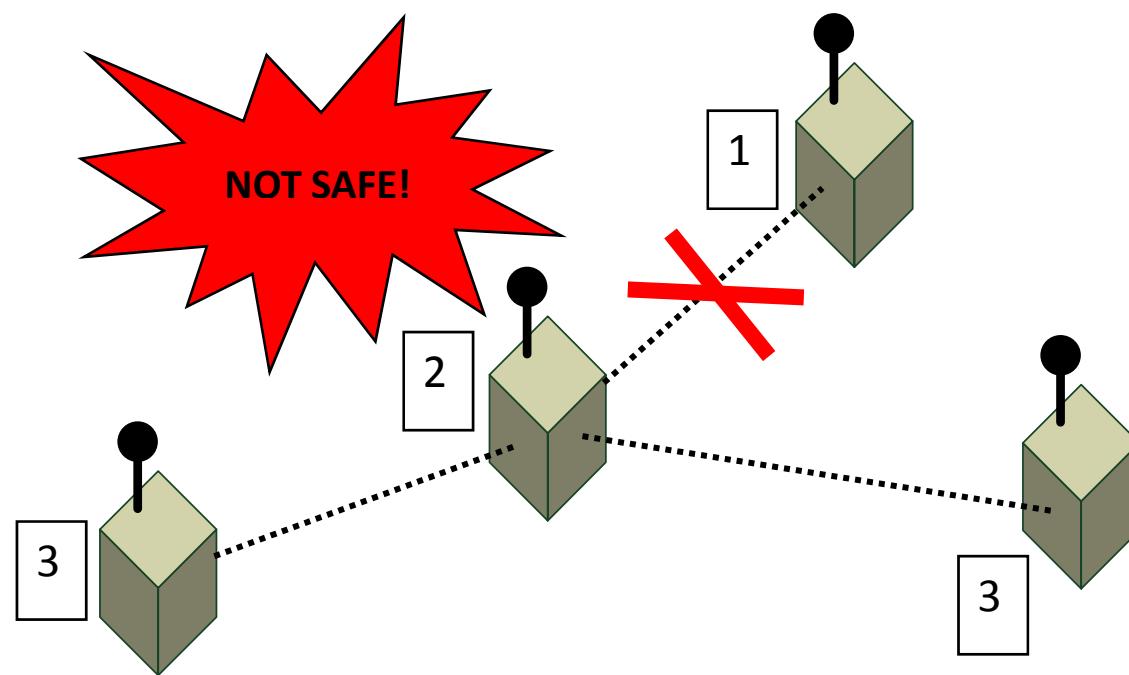
Given a Hop tuple X, we will call another tuple Y a **supporting tuple** of X if:

1. Y belongs to the same distributed tuple as X
2. Y is one-hop distant from X
3. the value of Y is equal to the value of X minus one

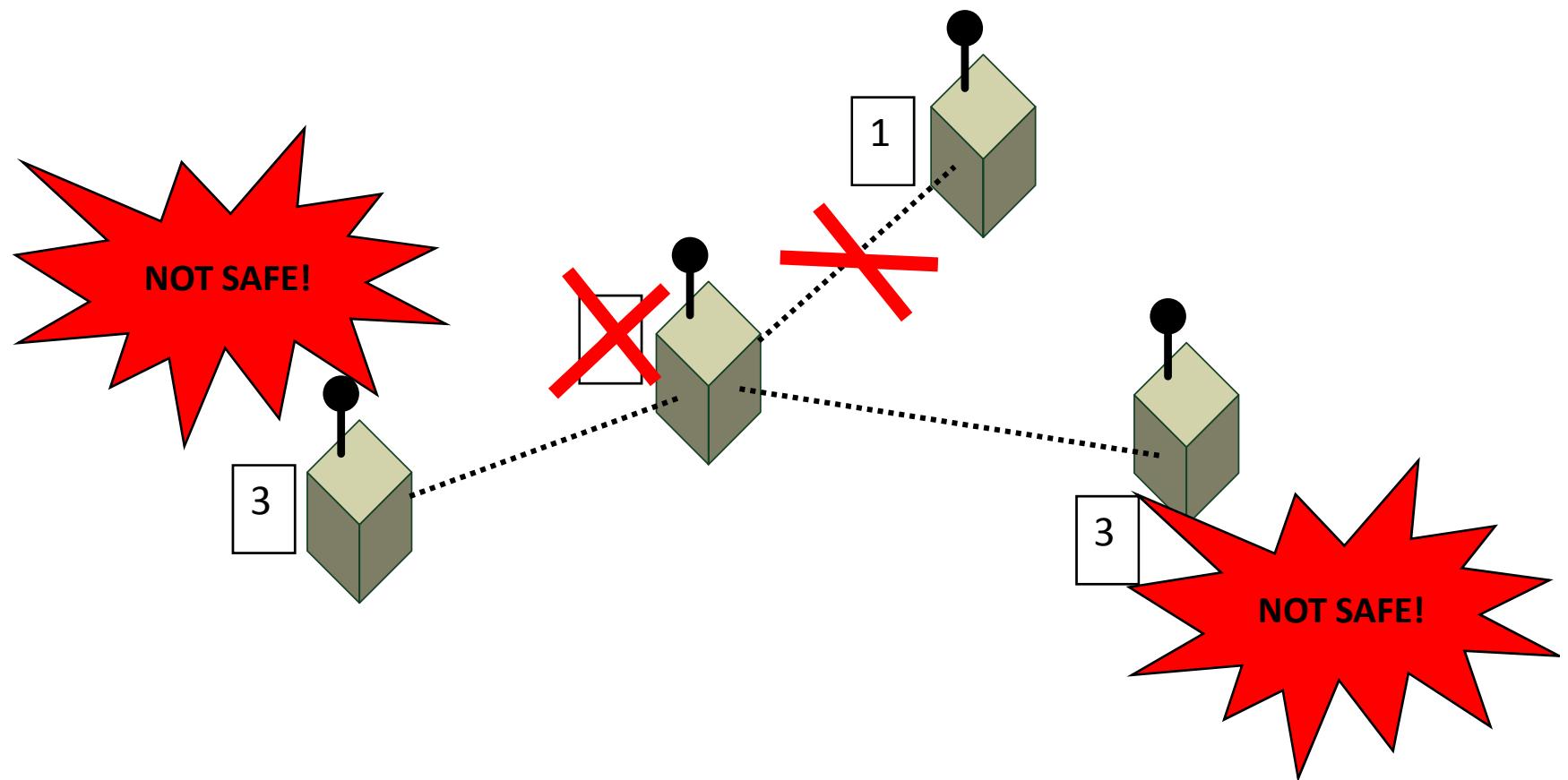
X is in a **safe-state** if it has at least a supporting tuple



Self-Maintenance EXAMPLE 1

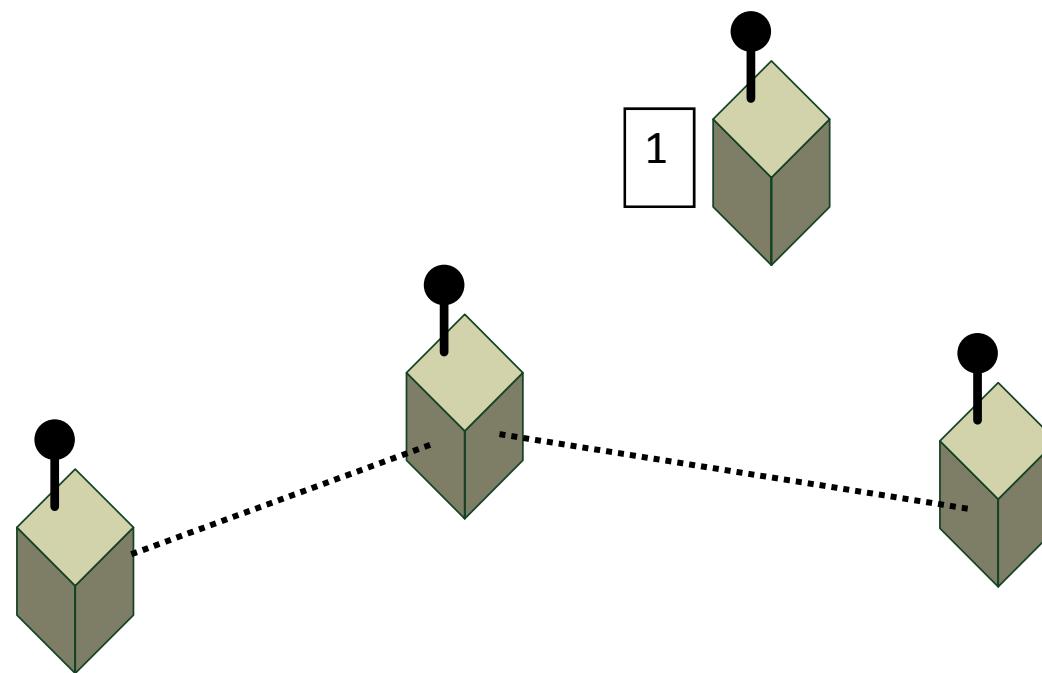


Self-Maintenance EXAMPLE 1

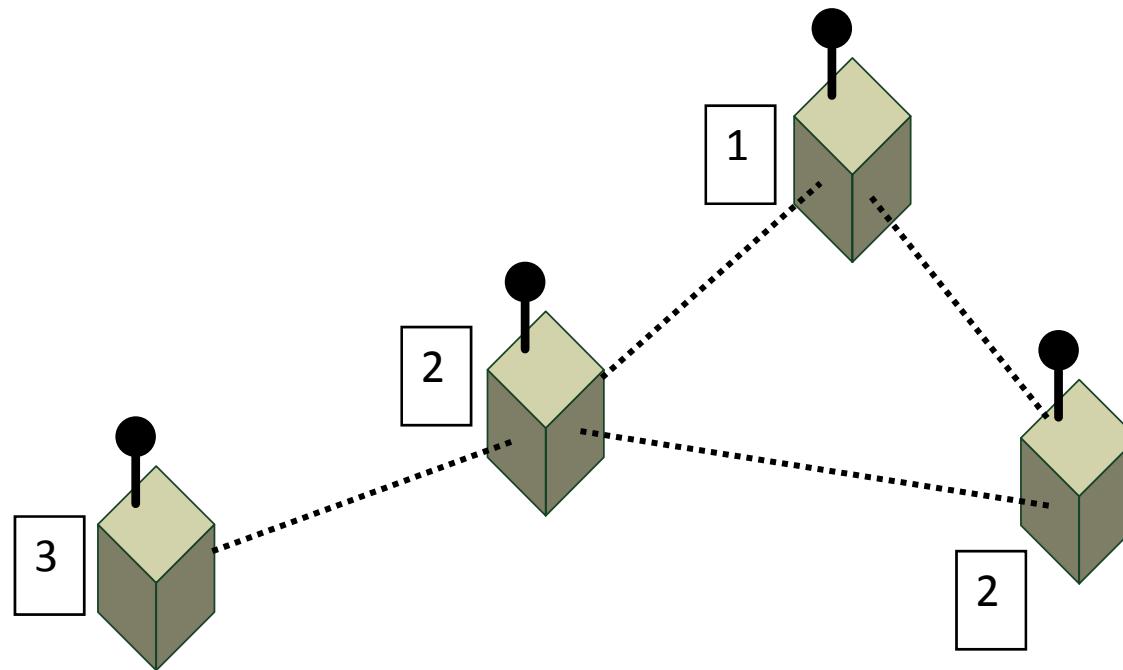


Self-Maintenance

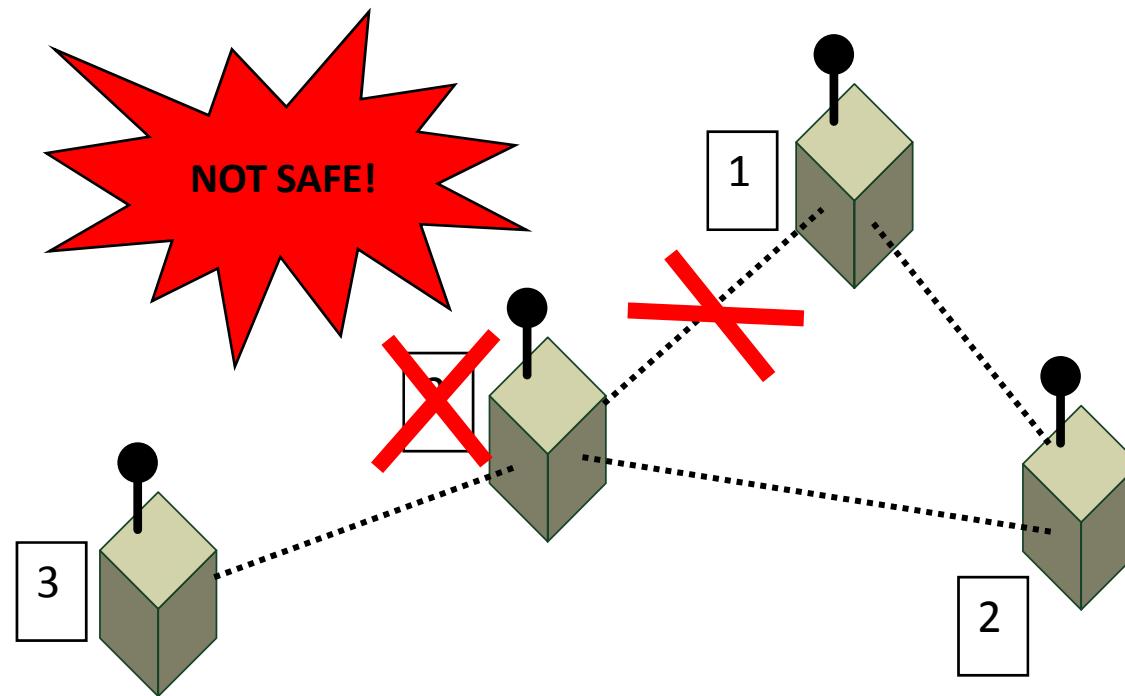
EXAMPLE 1



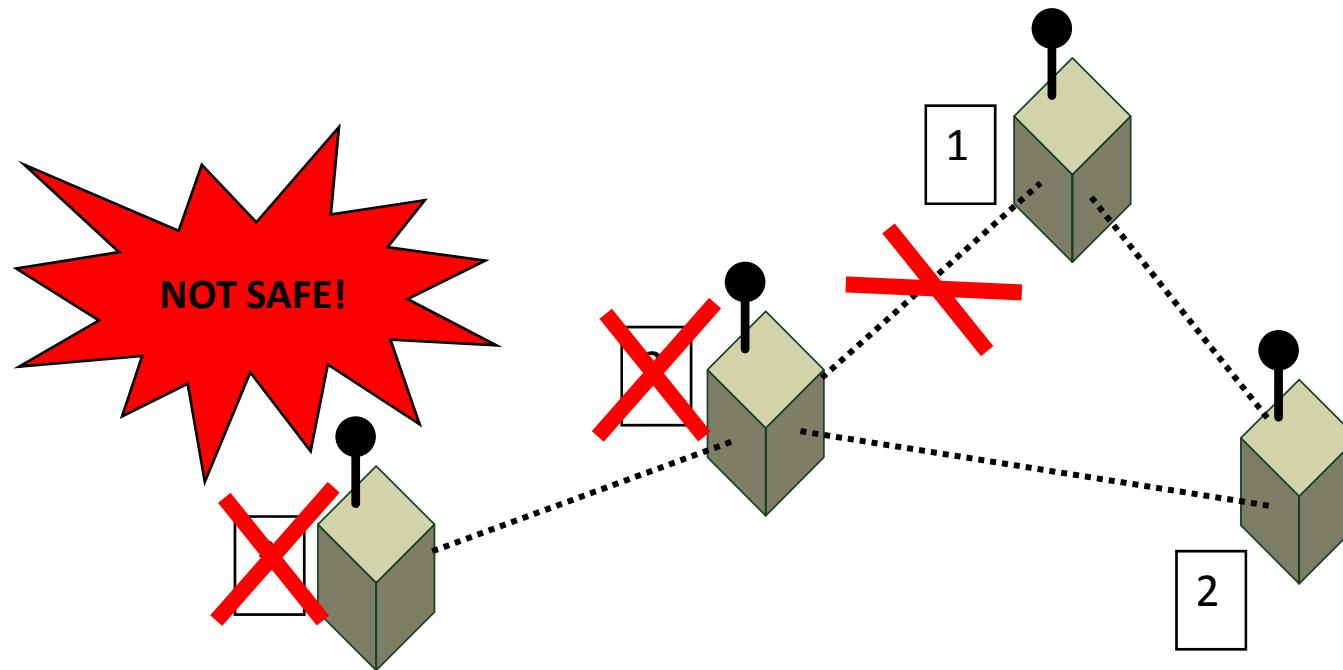
Self-Maintenance EXAMPLE 2



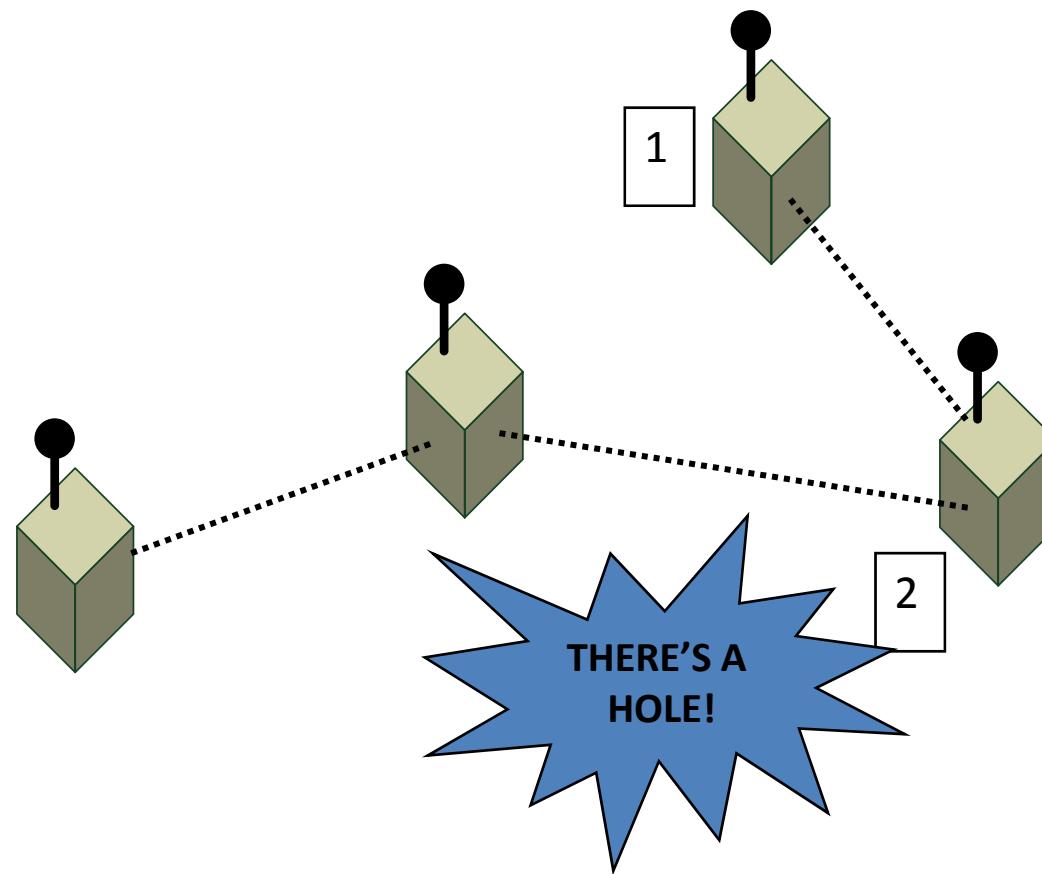
Self-Maintenance EXAMPLE 2



Self-Maintenance EXAMPLE 2

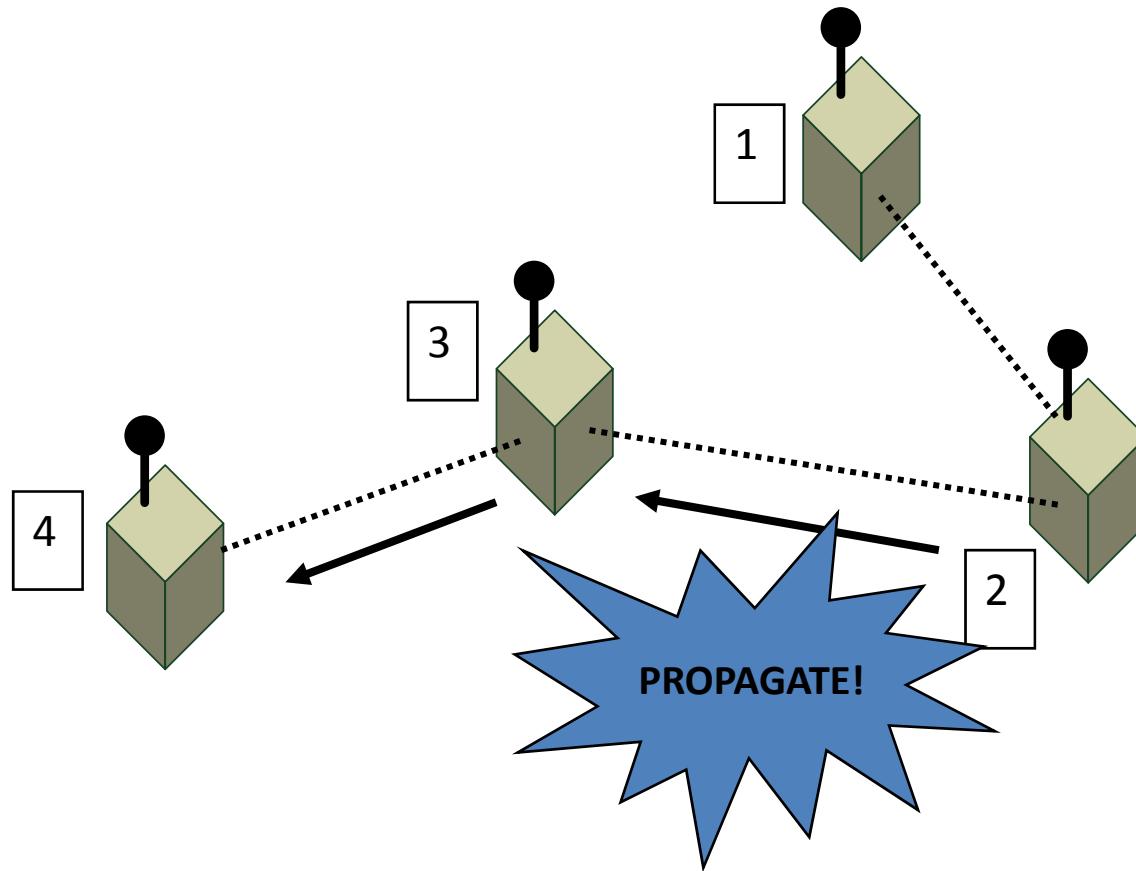


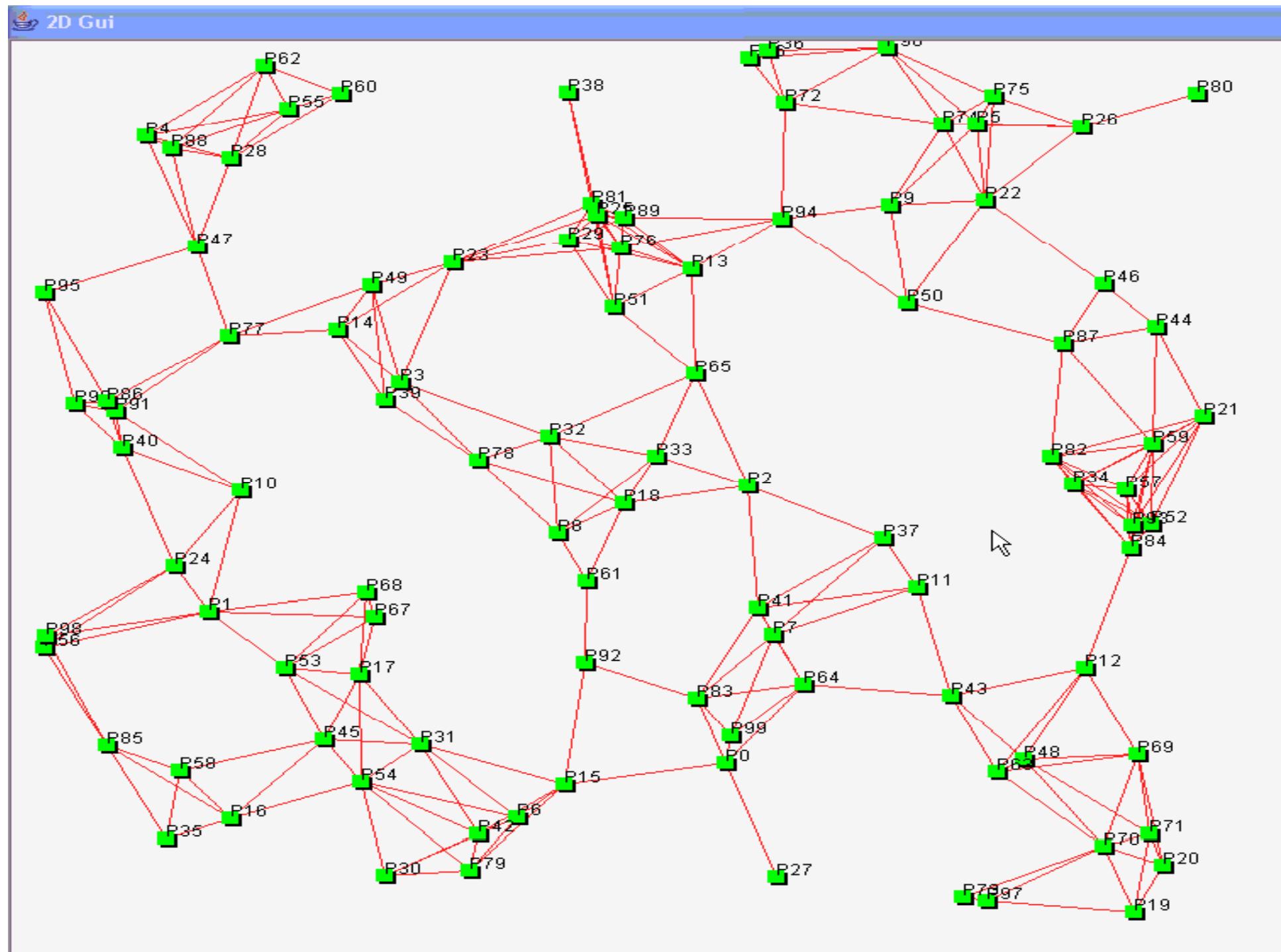
Self-Maintenance EXAMPLE 2



Self-Maintenance

EXAMPLE 2





Coding the meeting example 1/4

```
public class MeetAgent extends AbstractAgent {  
  
    private static final int speed = 30;  
    private PeerInterface_ApplicationSide peer;  
    private TotaMiddleware tota;  
  
    public MeetAgent(PeerInterface_ApplicationSide peer) {  
        this.peer = peer;  
        tota = new TotaMiddleware(peer);  
    }  
}
```

Coding the meeting example 2/4

```
public void step(int time) {  
    tota.step(time);  
    // inject the meeting tuple  
    if(time == 0) {  
        TotaTuple t = new GradientTuple();  
        t.setContent("<content=meet>"); // this notation is mandatory  
        tota.inject(t);  
    }  
  
    if(time > 0) {  
        int[] dir = getDirection();  
        if(dir != null)  
            move(dir[0],dir[1]);  
    }  
}
```

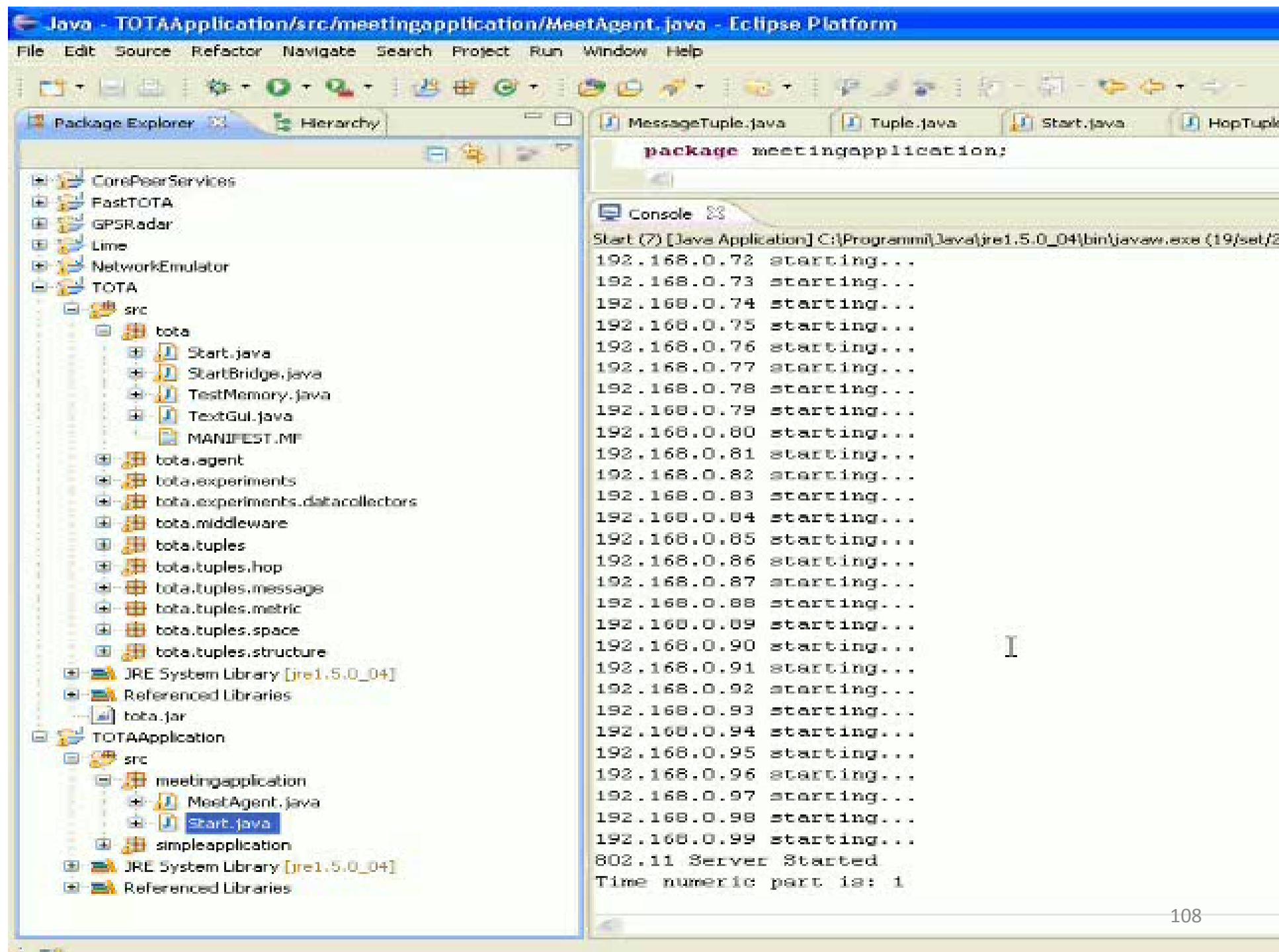
Coding the meeting example 3/4

```
private int[] getDirection() {
    // read the local tuple space
    GradientTuple mt = new GradientTuple();
    mt.setContent("<content=meet>");
    // read the local tuple space
    Vector local = tota.read(mt);

    // look in the local tuple space, the meet tuple with the highest hop counter
    int maxi = 0;
    GradientTuple maxt = (GradientTuple)local.get(0);
    for(int i=1; i<local.size(); i++) {
        GradientTuple t = (GradientTuple)local.get(i);
        if(maxt.hop < t.hop)
            maxt = t;
    }
}
```

Coding the meeting example 4/4

```
// look in the neighbor tuple spaces for neighbor
// having a lower value of the max tuple
GradientTuple tofollow = null;
Vector remote = tota.readOneHop(mt);
for(int i=0; i<remote.size(); i++) {
    GradientTuple t = (GradientTuple)remote.get(i);
    if(t.id.equals(maxt.id) && t.hop < maxt.hop) {
        if(tofollow==null || (t.ofollow.hop > t.hop))
            tofollow = t;
    }
}
return getDirection(tofollow.from);
...
}
```



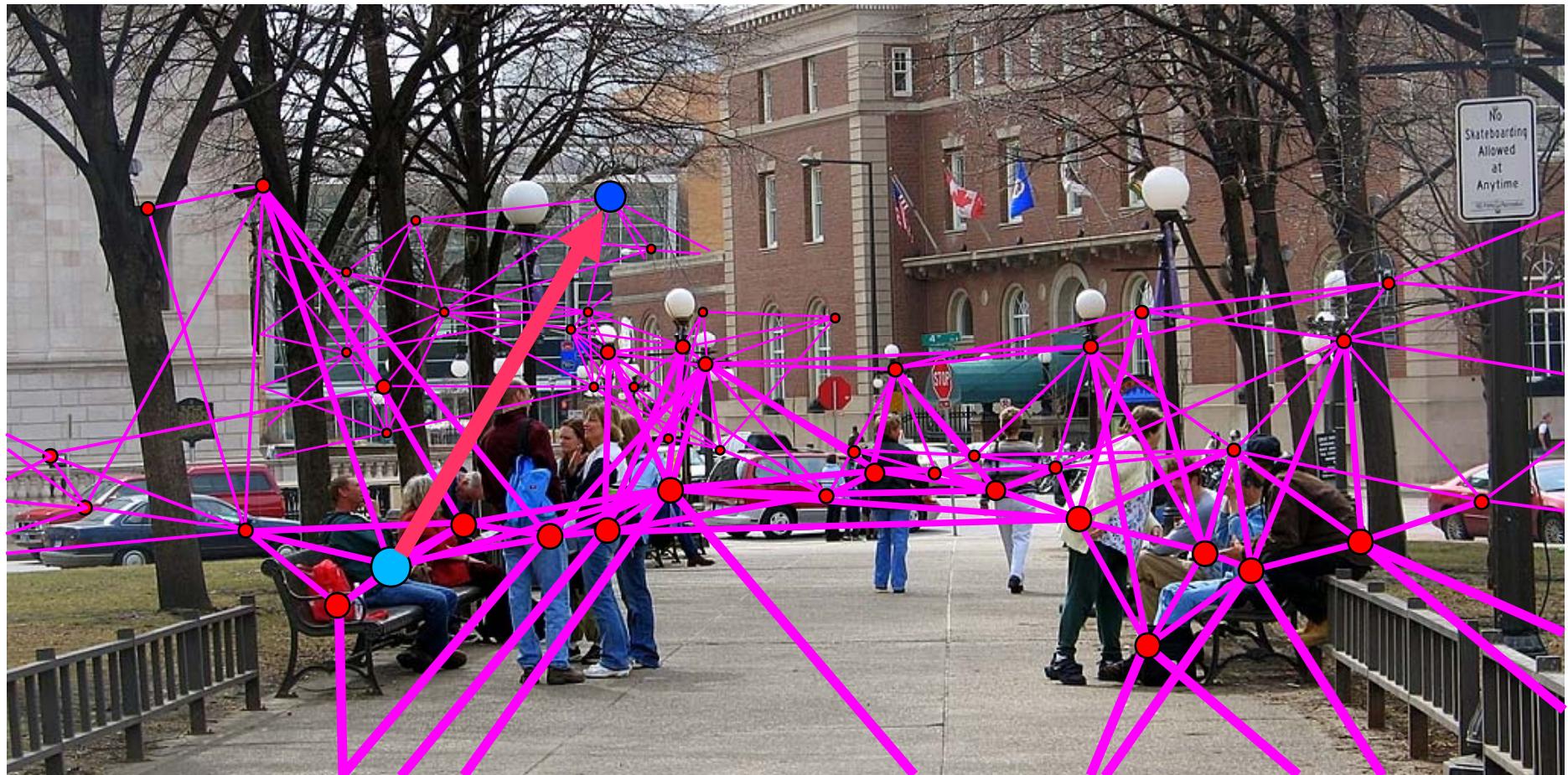
Summary

- Field-based data structures are useful to represent context:
 - in a wide range of scenarios
 - so that it is easily usable by services
- TOTA tuples are robust to dynamic spatial computer and can be easily programmed
- Download and play with TOTA
 - <http://polaris.ing.unimo.it/tota/download.html>

Syllabus

- Pervasive Computing, Spatial Computing
 - Example Scenarios
- Overview of Spatial Approaches
- Three spatial programming models:
 - Spatial Programming / Smart Messages
 - TOTA / Field-based Coordination
 - Proto / Amorphous Medium abstraction

Example: mobile streaming

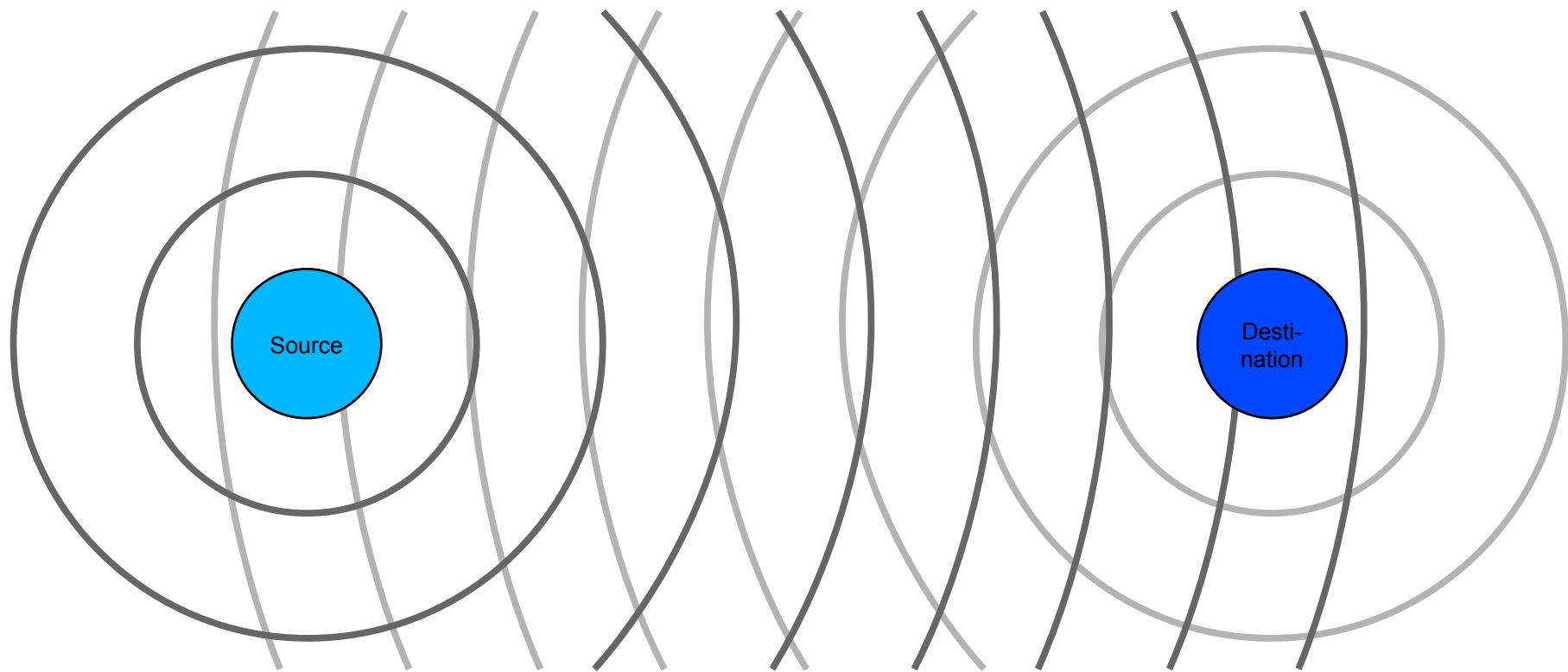


Geometric program: channel



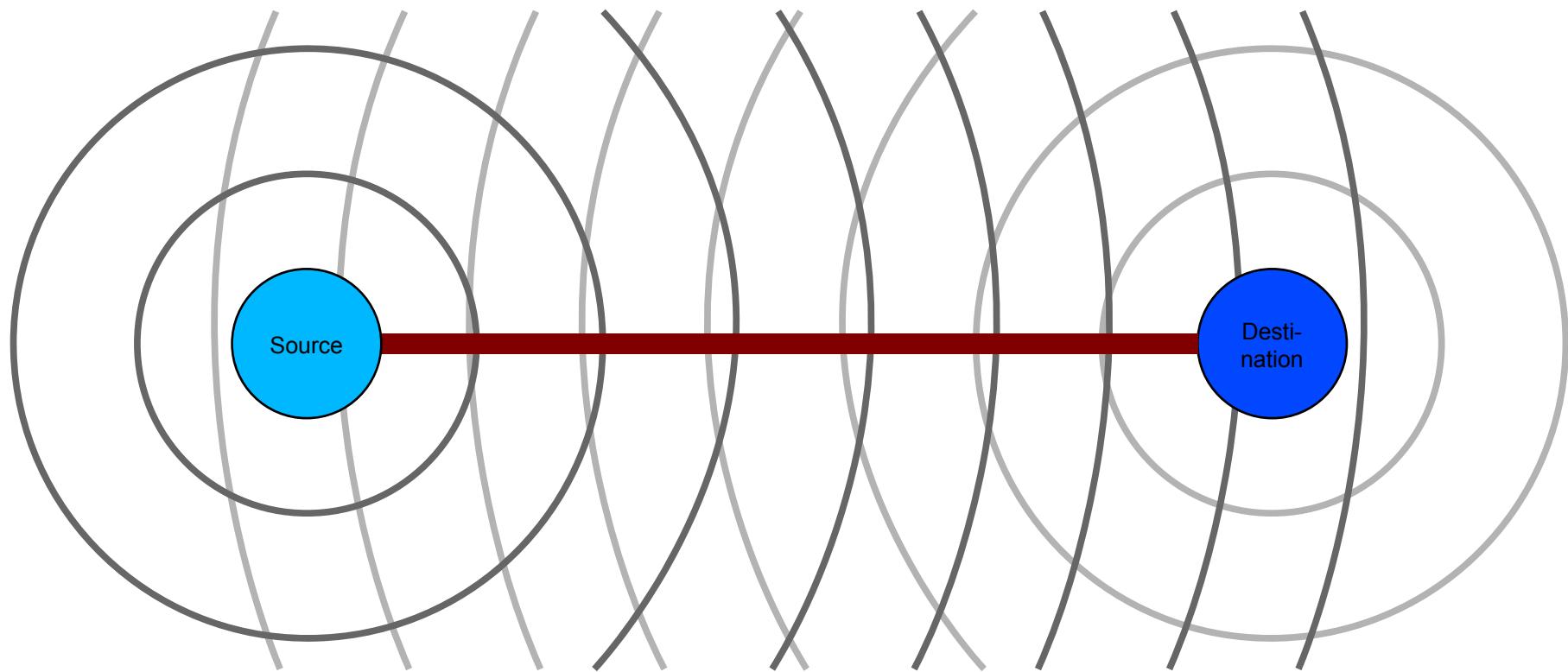
(cf. Butera)

Geometric program: channel



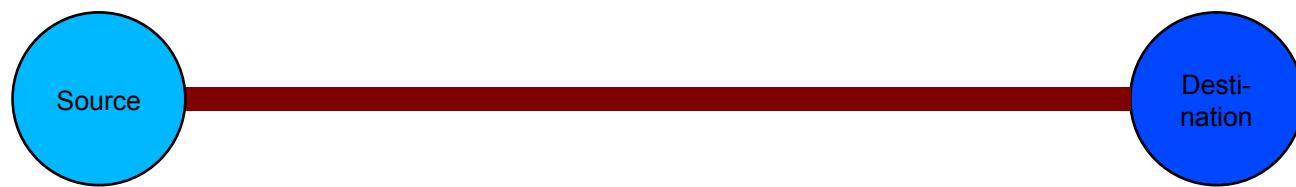
(cf. Butera)

Geometric program: channel



(cf. Butera)

Geometric program: channel



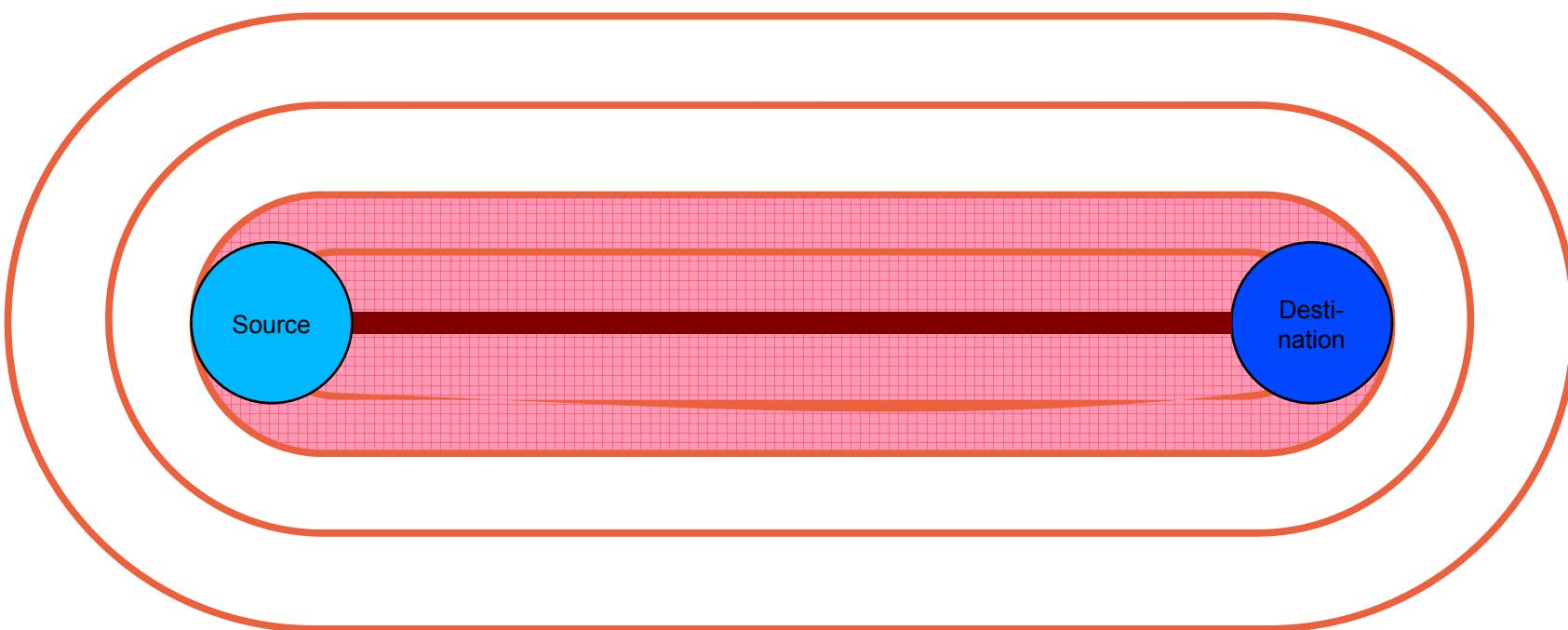
(cf. Butera)

Geometric program: channel



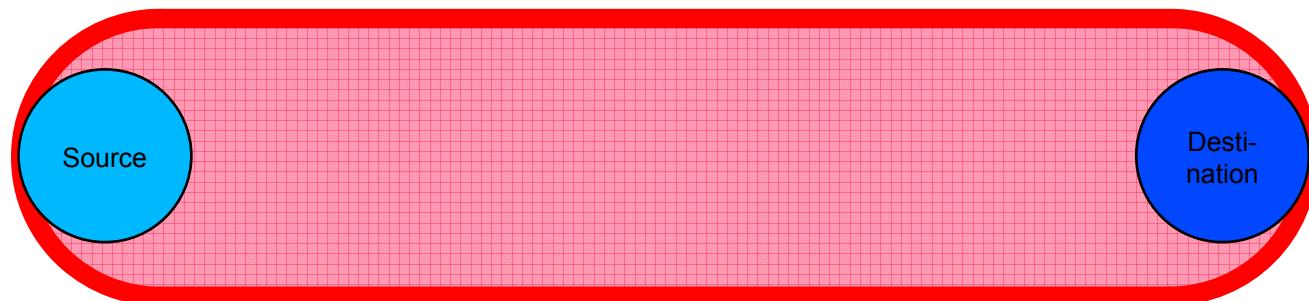
(cf. Butera)

Geometric program: channel



(cf. Butera)

Geometric program: channel



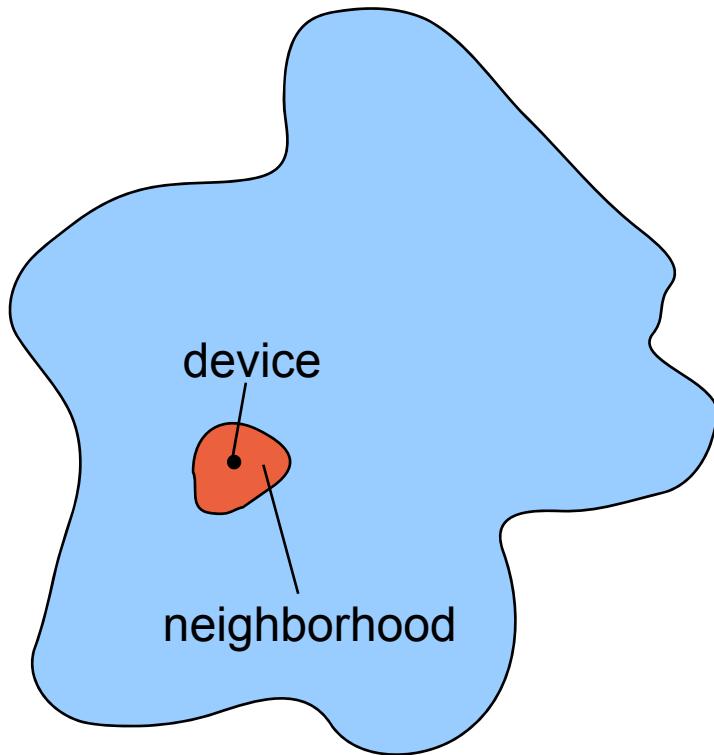
(cf. Butera)

Why use continuous space?

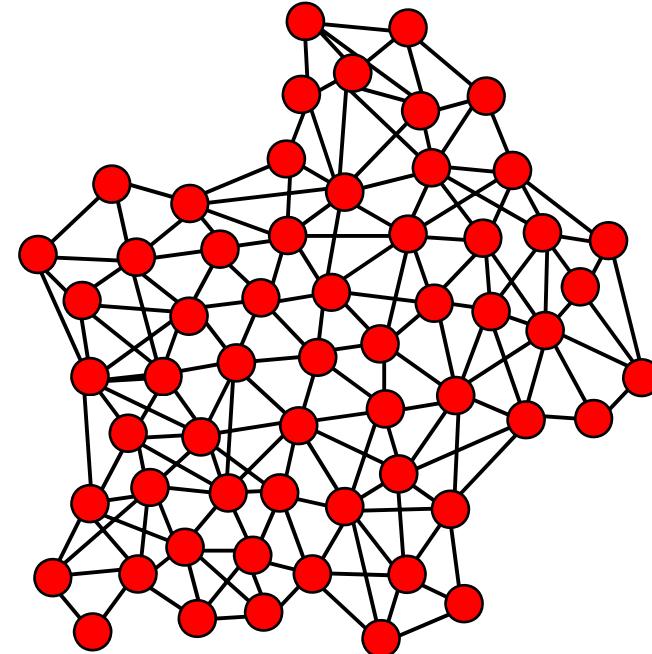
- Simplicity
- Scaling & Portability
- Robustness

(we'll come back to this in a bit...)

Amorphous medium

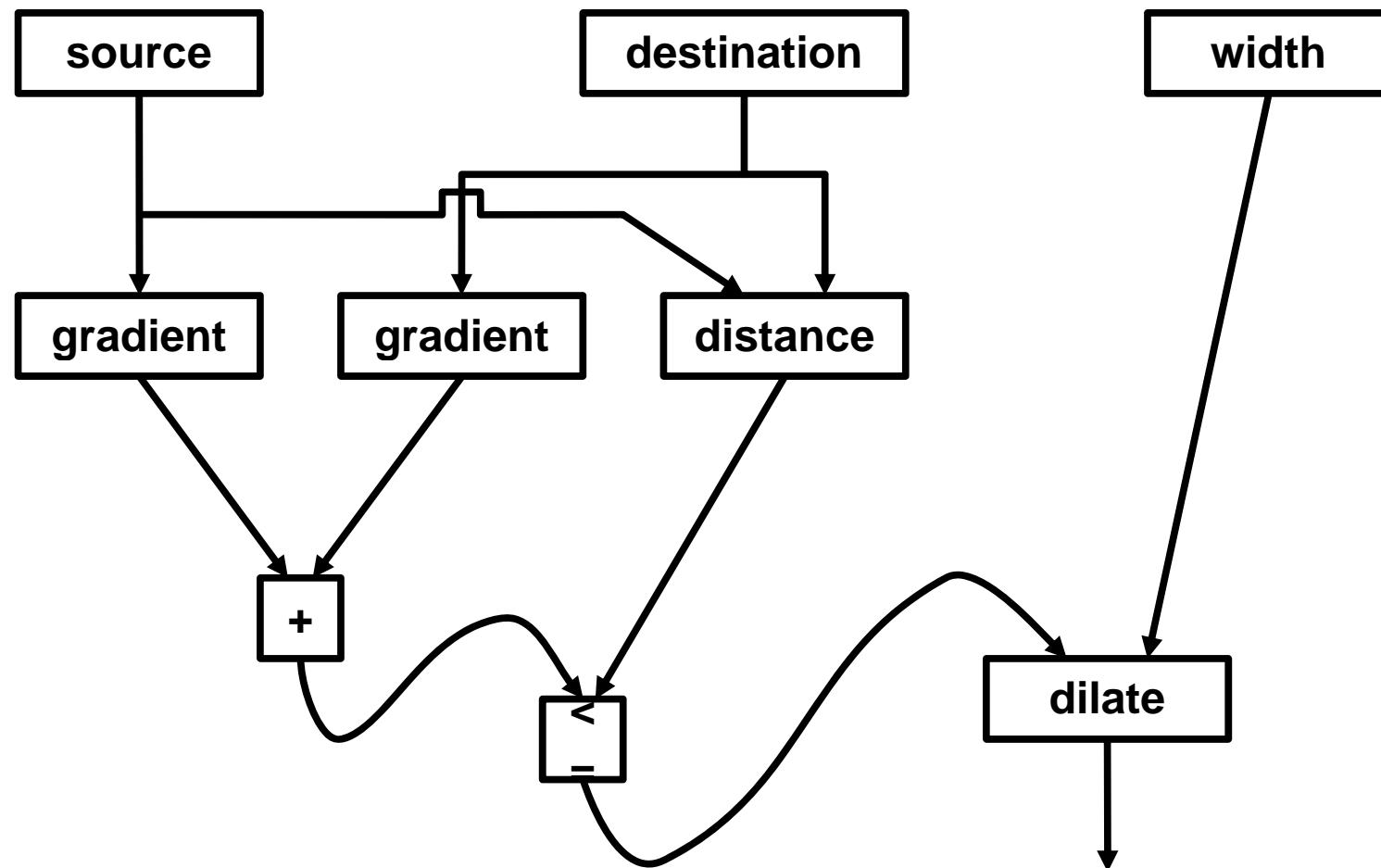


- Continuous space & time
- Infinite number of devices
- See neighbors' past state

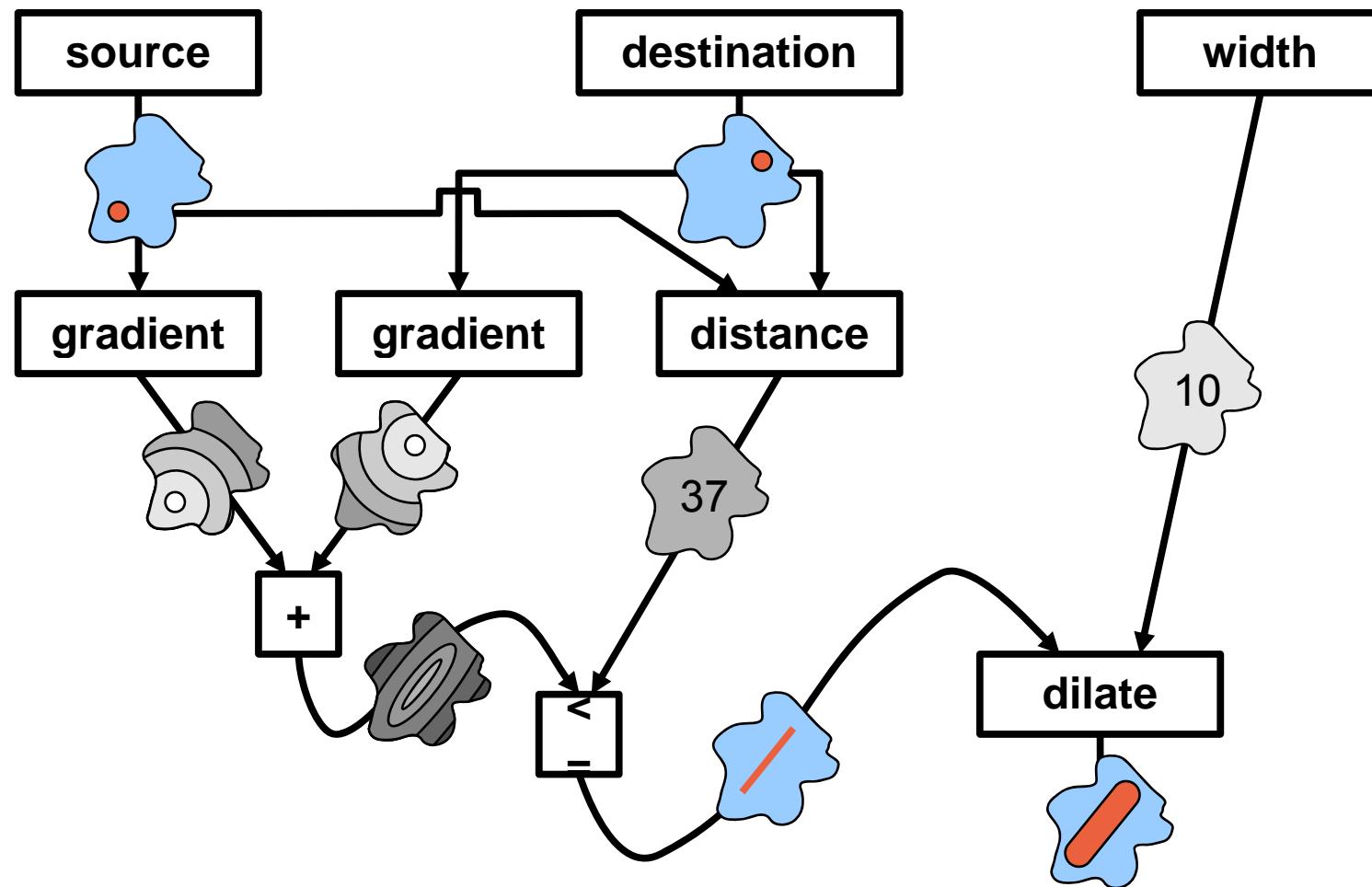


- Approximate with:
- Discrete network of devices
 - Signals transmit state

Computing with fields



Computing with fields



Diving into the details

- Let's build this up using the Proto simulator,
 - one piece at a time...
 - *(break to work w. simulator)*

Proto

```
(def gradient (src ...))  
(def distance (src dst ...) ...)  
(def dilate (src n)  
  (<= (gradient src) n))  
(def channel (src dst width))  
(let* ((d (distance src dst))  
       (trail (<= (+ (gradient src)  
                     (gradient dst))  
                  d)))  
  (dilate trail width)))
```

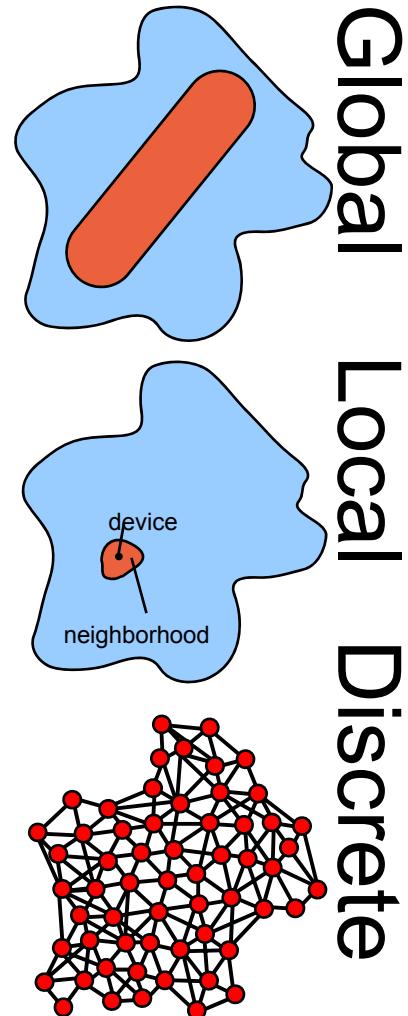
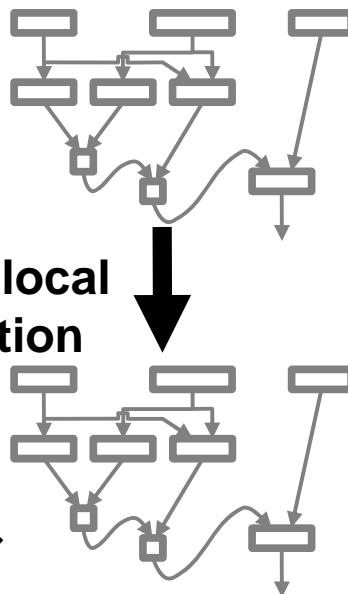
platform
specificity &
optimization

evaluation

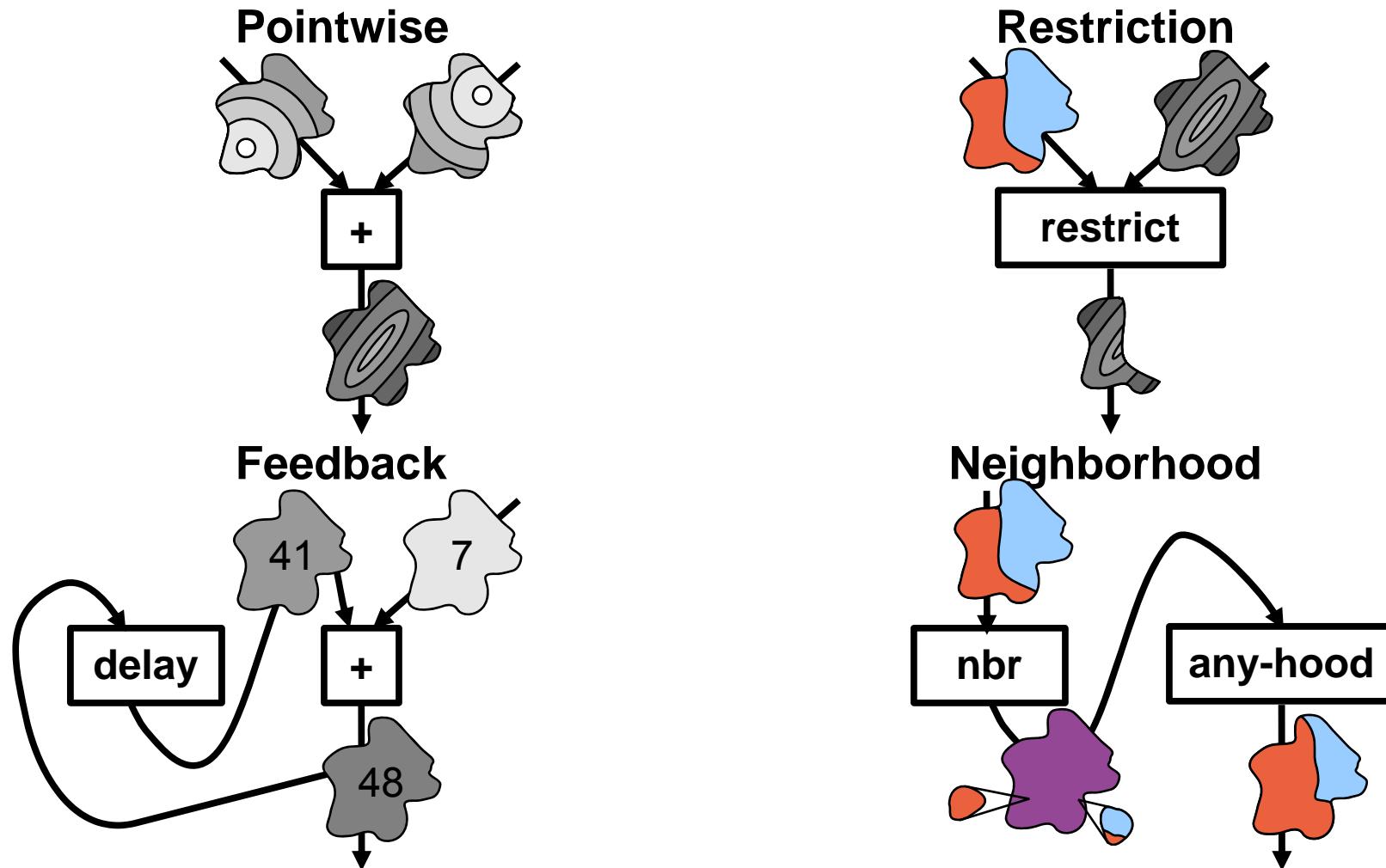
global to local
compilation

discrete
approximation

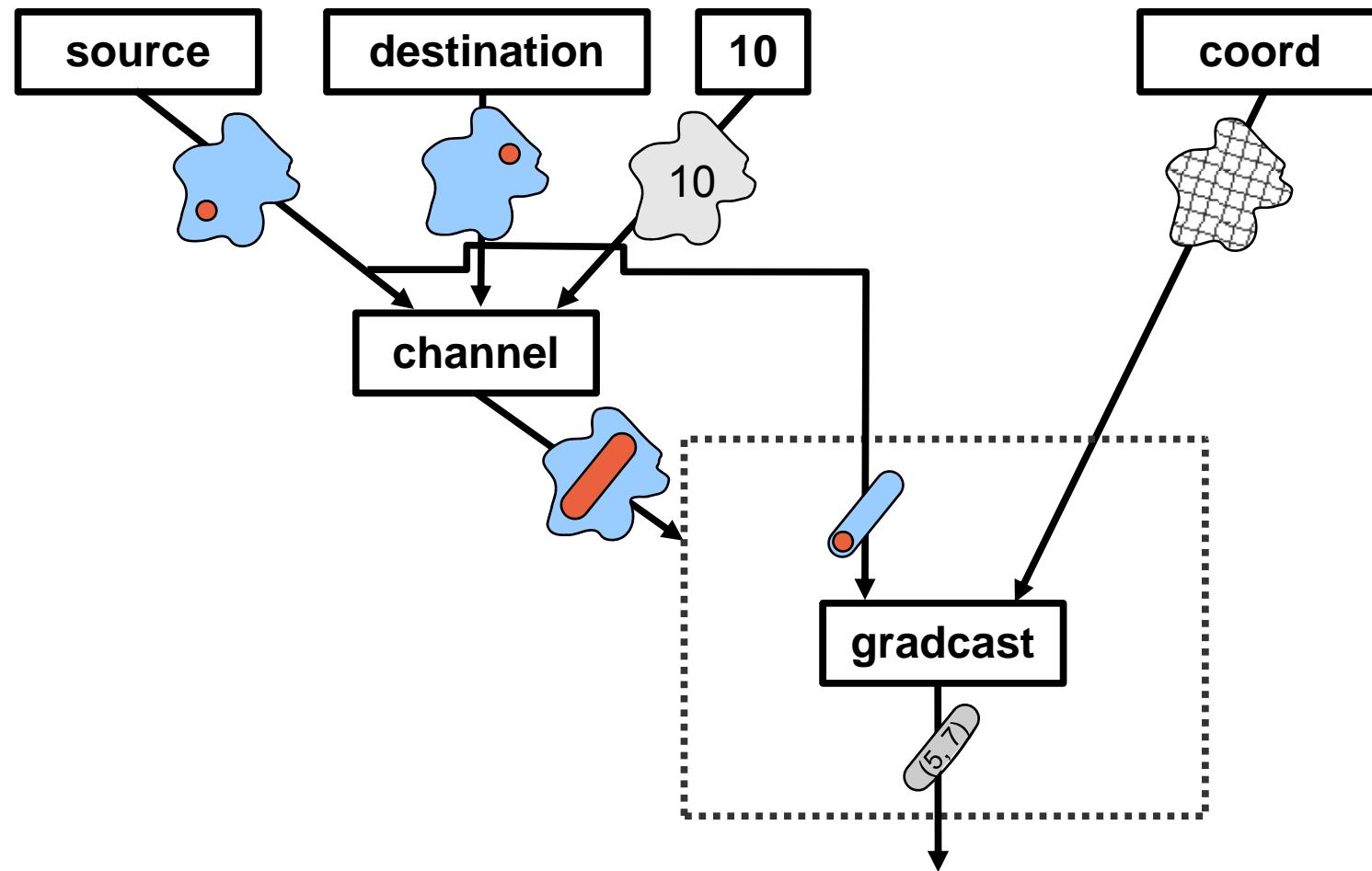
Device
Kernel



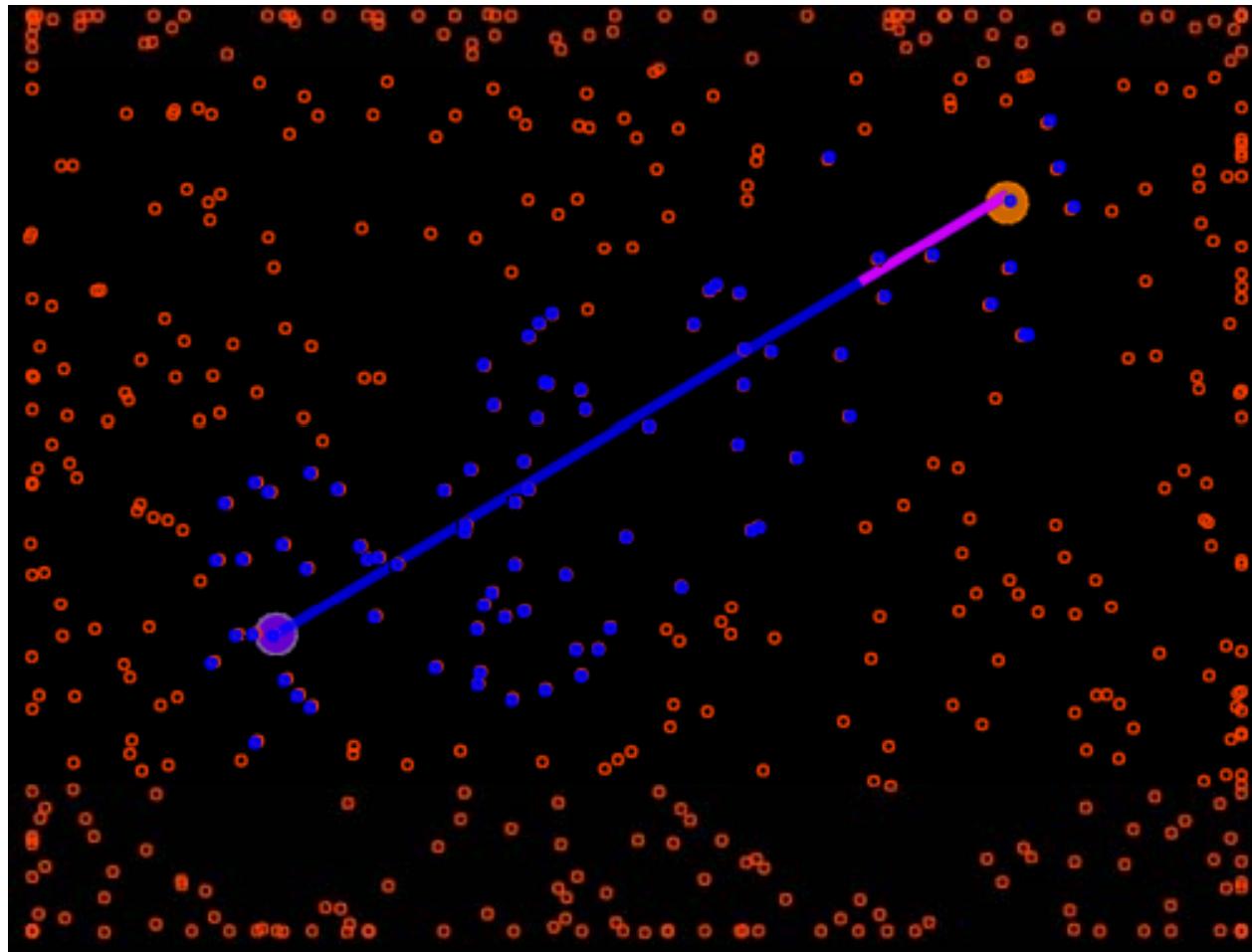
Proto's families of primitives



Modulation by restriction

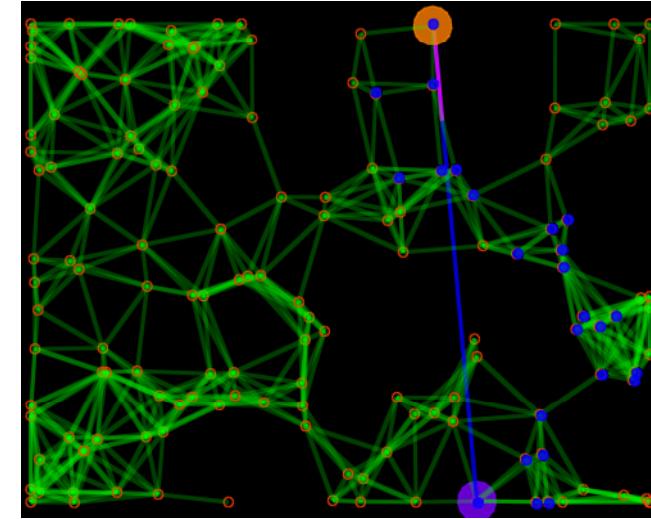
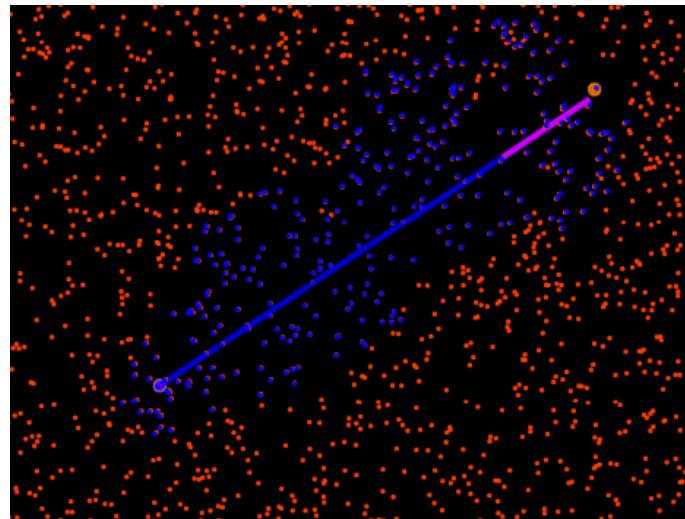


In simulation...



Why use continuous space?

- Simplicity
- Scaling & Portability
- Robustness

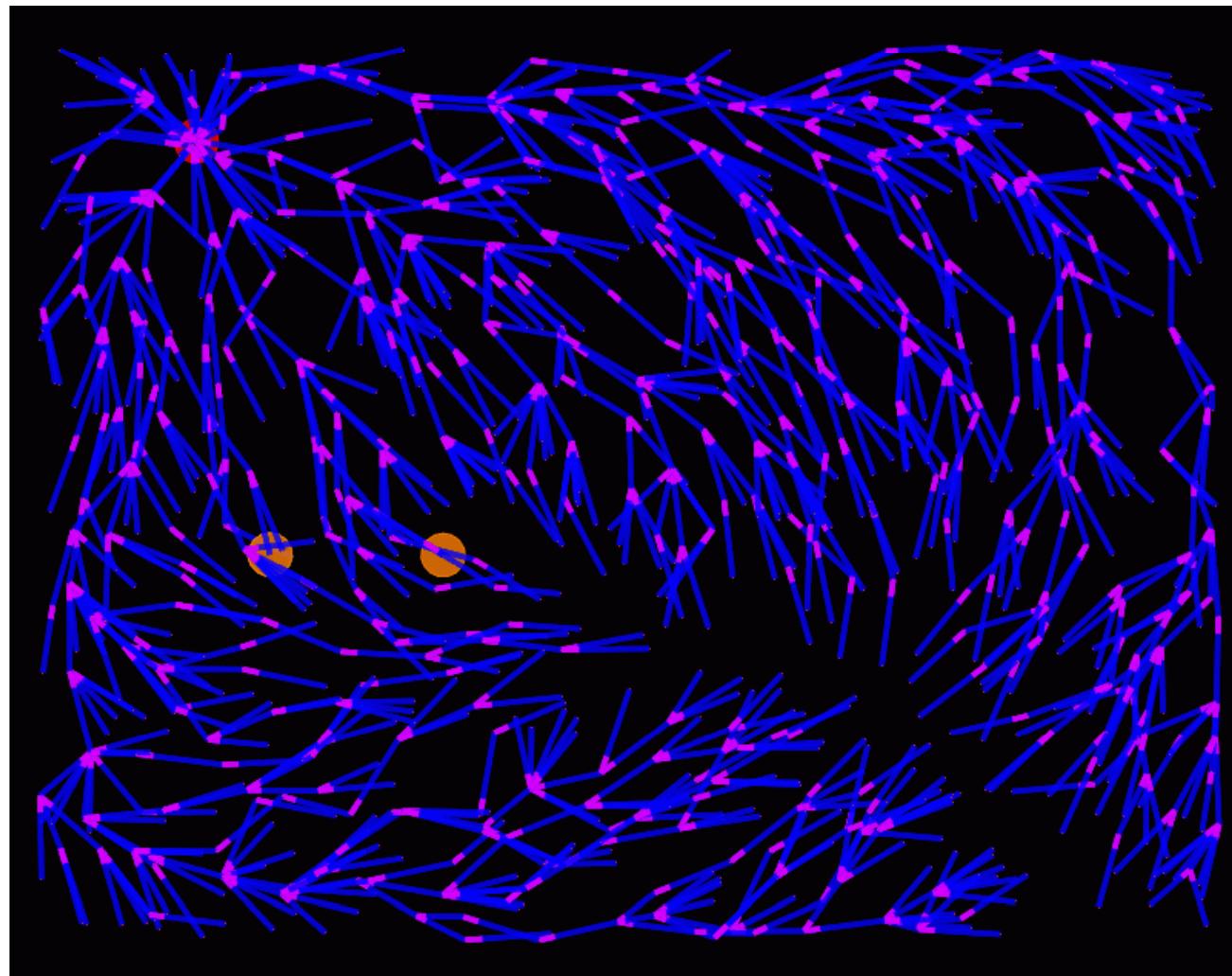


Example: intrusion detection



- Use channel to stream intruder information

Example: museum traffic control



Weaknesses

- Functional programming scares people
- Programmers can break the abstraction
- No dynamic allocation of processes
- No formal proofs available for quality of approximation in a composed program

(active research on last two)

Summary

- Amorphous Medium abstraction simplifies programming of space-filling networks
- Proto has four families of space and time operations, compiles global descriptions into local actions that approximate the global
- Geometric metaphors allow complex spatial computing problems to be solved with very short programs.

Conclusions

- New and exciting research area!
- Many pervasive computing scenarios reflect spatial computers concepts
- Many research approaches exist to programming pervasive application for spatial computing
- Still there is a lot of open issues to be addressed:
 - More software engineering
 - More real-world applications
 - Better definition and management of “emergent” behaviors and properties

Some References

- C. Borcea and D. Iyer and P. Kang and A. Saxena and L. Iftode, **Spatial programming using smart messages: Design and implementation**, IEEE International Conference on Distributed Computing Systems. Tokio, Japan, 2004.
- O. Riva, T. Nadeem, C. Borcea, L. Iftode, **Context-aware Migratory Services in Ad Hoc Networks**, IEEE Transactions on Mobile Computing 6(12):34-33, 2007.
- M. Mamei, F. Zambonelli, **Programming Pervasive and Mobile Computing Applications: the TOTA Approach**, ACM Transaction on Software Engineering and Methodology, 2008, to appear
- M. Mamei, F. Zambonelli, L. Leonardi, **Co-Fields: A Physically Inspired Approach to Distributed Motion Coordination**, IEEE Pervasive Computing, 3(2):52-61, 2004.
- J. Bachrach, J. Beal, T. Fujiwara, **Continuous Space-Time Semantics Allow Adaptive Program Execution**. IEEE SASO 2007, Cambridge (MA) , USA, 2007.
- J. Beal, J. Bachrach, D. Vickery, M. Tobenkin, **Fast Self-Healing Gradients**, ACM Symposium on Applied Computing, Fortaleza, Brazil, 2008.

Thanks for your attention!