Aggregate Computing and Field Calculus

Ferruccio Damiani

University of Turin - Department of Computer Science

www.di.unito.it/~damiani

Mobile Device Programming

(Laurea Magistrale in Informatica, a.a. 2018-2019)

Challenges

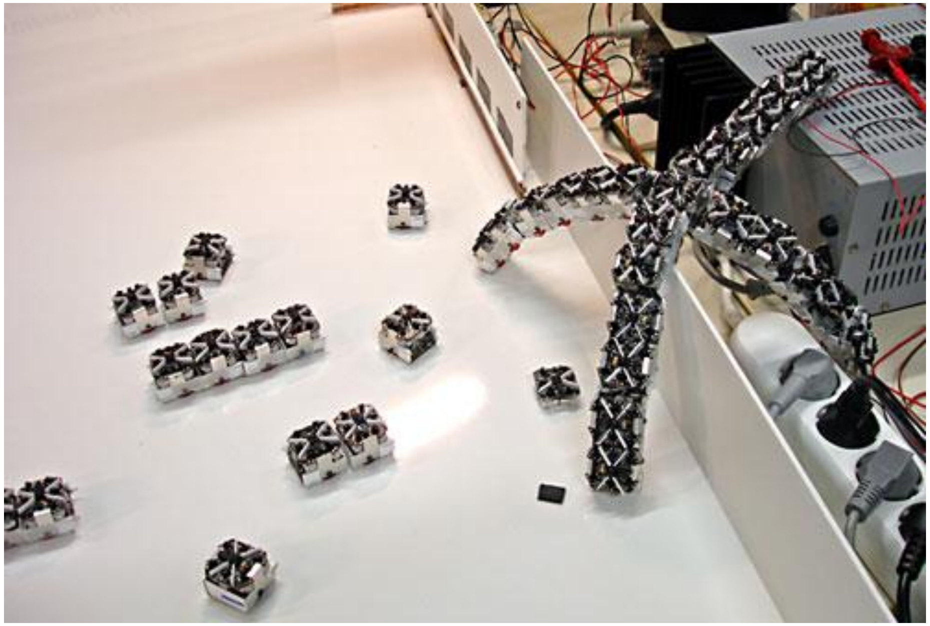
Programming a distributed system poses several challenges:

- diverse heterogeneous entities → device abstraction?
- collaboration vs selfishness → centralization? aggregation?
- dynamic goals and environment → adaptive algorithms?
- data security and privacy → cryptography? localised aggregation?

Challenges

Programming a distributed system poses several challenges:

- diverse heterogeneous entities → device abstraction?
- collaboration vs selfishness → centralization? aggregation?
- dynamic goals and environment → adaptive algorithms?
- data security and privacy → cryptography? localised aggregation?



Challenges

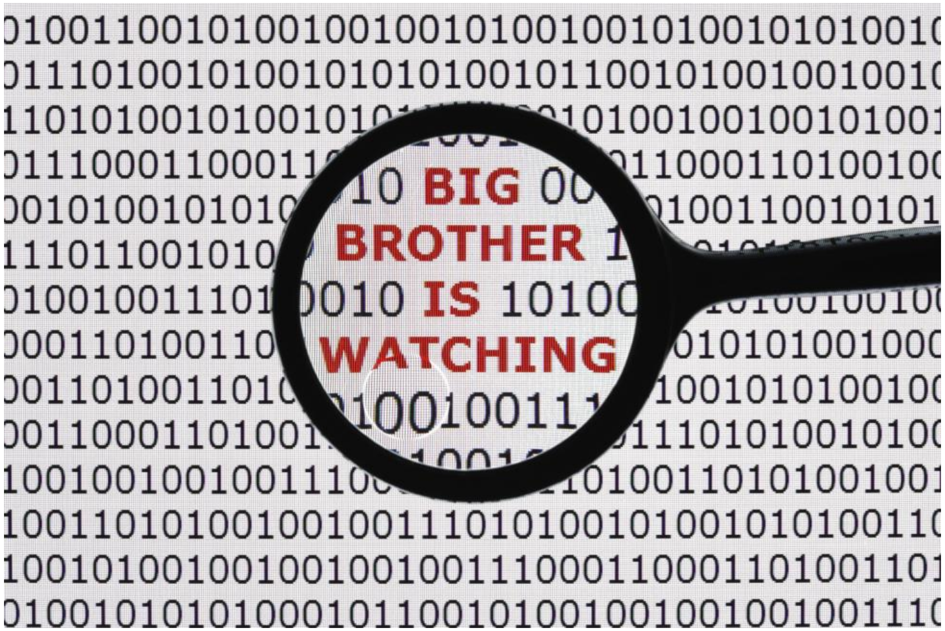
Programming a distributed system poses several challenges:

- diverse heterogeneous entities → device abstraction?
- collaboration vs selfishness → centralization? aggregation?
- dynamic goals and environment → adaptive algorithms?
- data security and privacy → cryptography? localised aggregation?

Challenges

Programming a distributed system poses several challenges:

- diverse heterogeneous entities → device abstraction?
- collaboration vs selfishness → centralization? aggregation?
- dynamic goals and environment → adaptive algorithms?
- data security and privacy → cryptography? localised aggregation?



Challenges

Programming a distributed system poses several challenges:

- diverse heterogeneous entities → device abstraction?
- collaboration vs selfishness → centralization? aggregation?
- dynamic goals and environment → adaptive algorithms?
- data security and privacy → cryptography? localised aggregation?

Classical paradigms, algorithms and languages hardly deal with these expectations

Aggregate Computing: the underlying idea

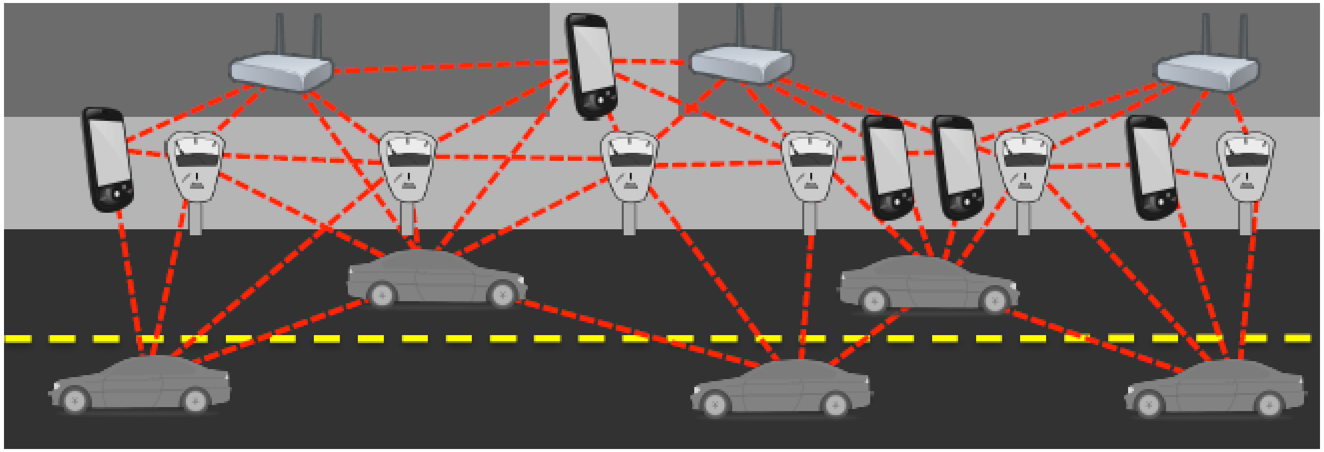
Shifting

- From single-device focus and query-based system programming
- To data-based aggregate viewpoint:
  - overall set of devices spread in a pervasive computing environment seen as a single aggegate machine
  - overall dispersed localised data as a single entity: computational field
  - aggregate specifications as global plans, locally interpreted by agents

Aggregate Computing: the underlying idea

Shifting

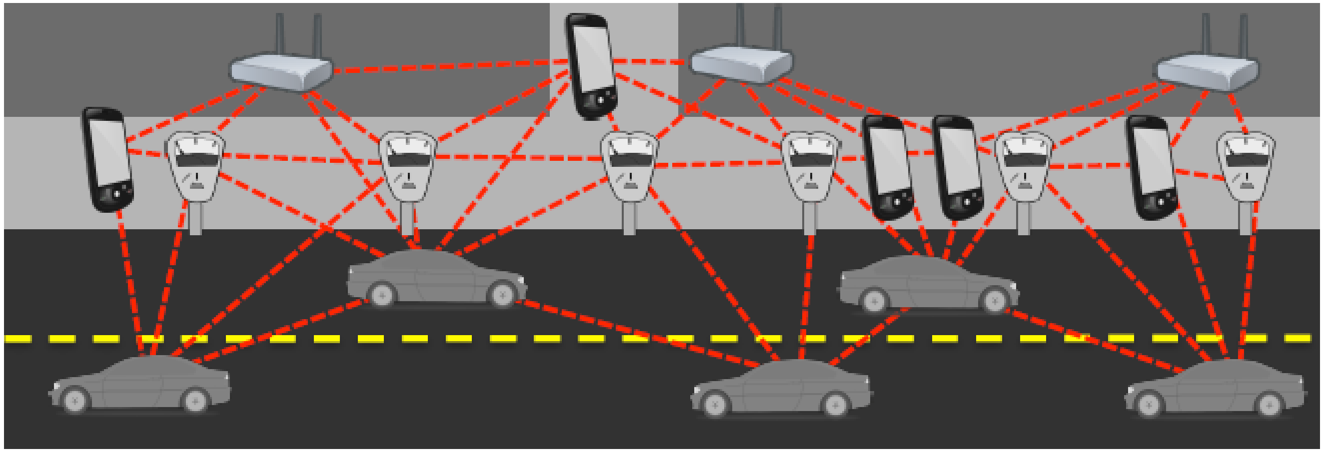
- From single-device focus and query-based system programming
- To data-based aggregate viewpoint:
  - overall set of devices spread in a pervasive computing environment seen as a single aggegate machine
  - overall dispersed localised data as a single entity: computational field
  - aggregate specifications as global plans, locally interpreted by agents



Aggregate Computing: the underlying idea

Shifting

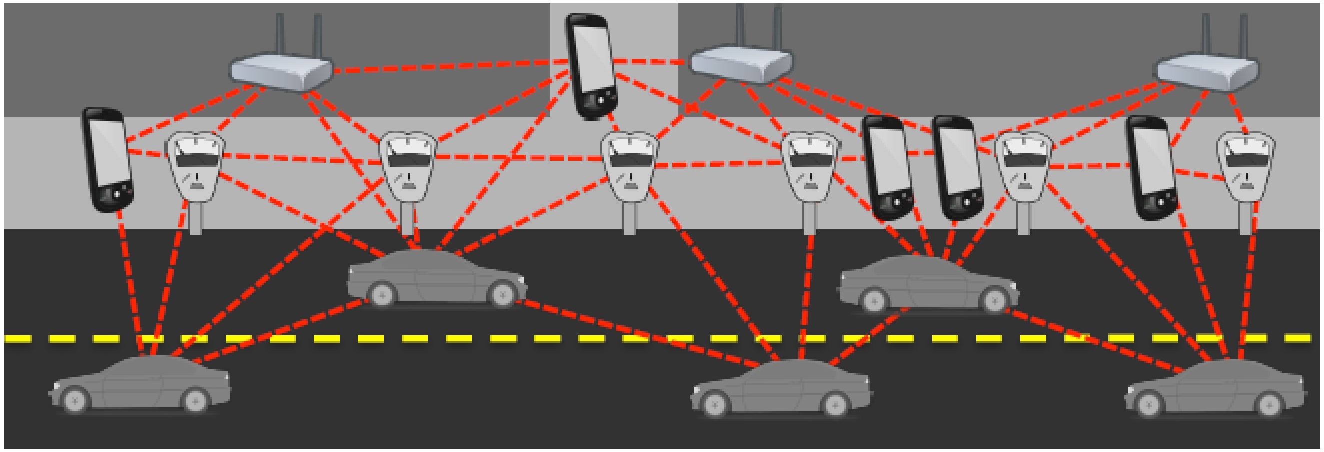
- From single-device focus and query-based system programming
- To data-based aggregate viewpoint:
  - overall set of devices spread in a pervasive computing environment seen as a single aggegate machine
  - overall dispersed localised data as a single entity: computational field
  - aggregate specifications as global plans, locally interpreted by agents



Aggregate Computing: the underlying idea

Shifting

- From single-device focus and query-based system programming
- To data-based aggregate viewpoint:
  - overall set of devices spread in a pervasive computing environment seen as a single aggegate machine
  - overall dispersed localised data as a single entity: computational field
  - aggregate specifications as global plans, locally interpreted by agents



Aggregate Computing: the underlying idea

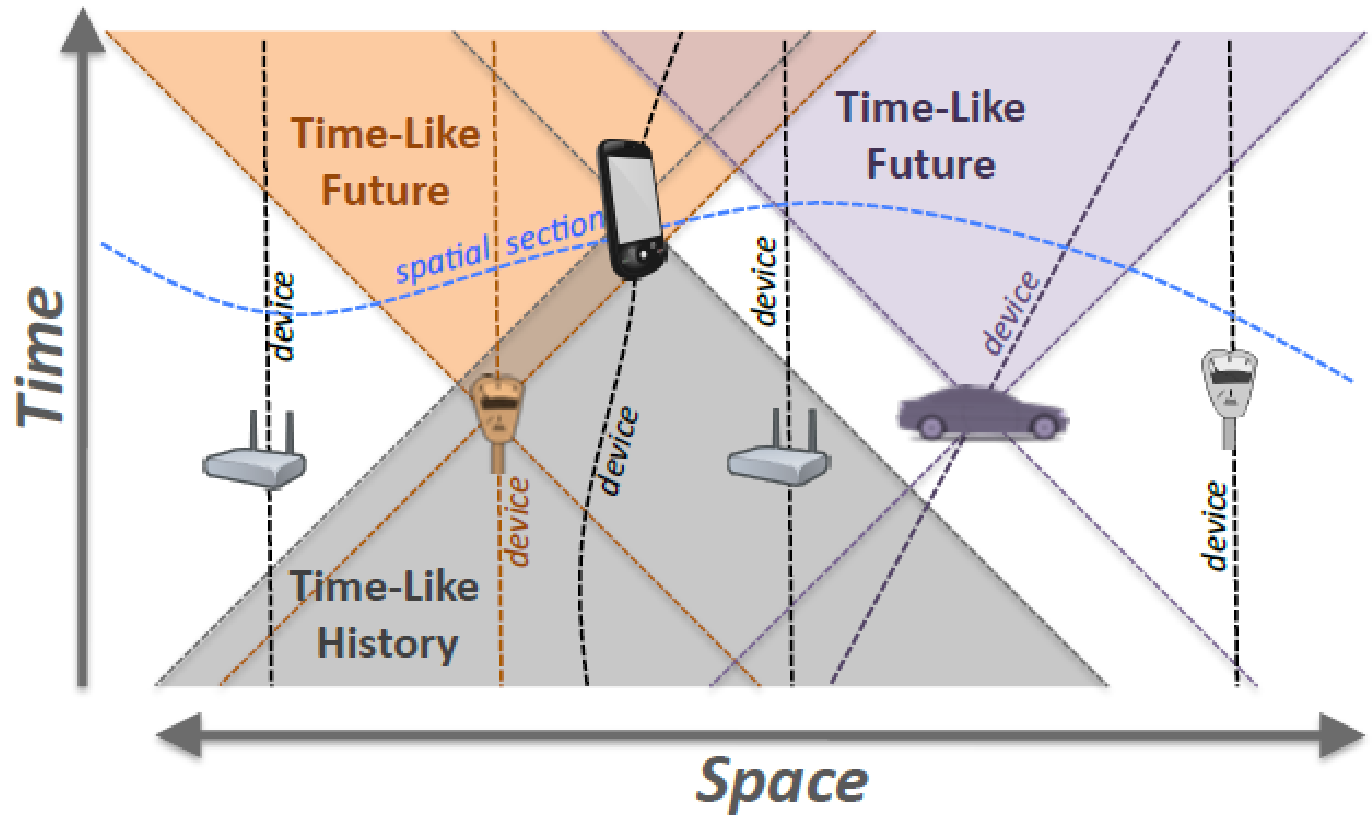
Shifting

- From single-device focus and query-based system programming
- To data-based aggregate viewpoint:
  - overall set of devices spread in a pervasive computing environment seen as a single aggegate machine
  - overall dispersed localised data as a single entity: computational field
  - aggregate specifications as global plans, locally interpreted by agents

adaptivity, resiliency, robustness, simplicity

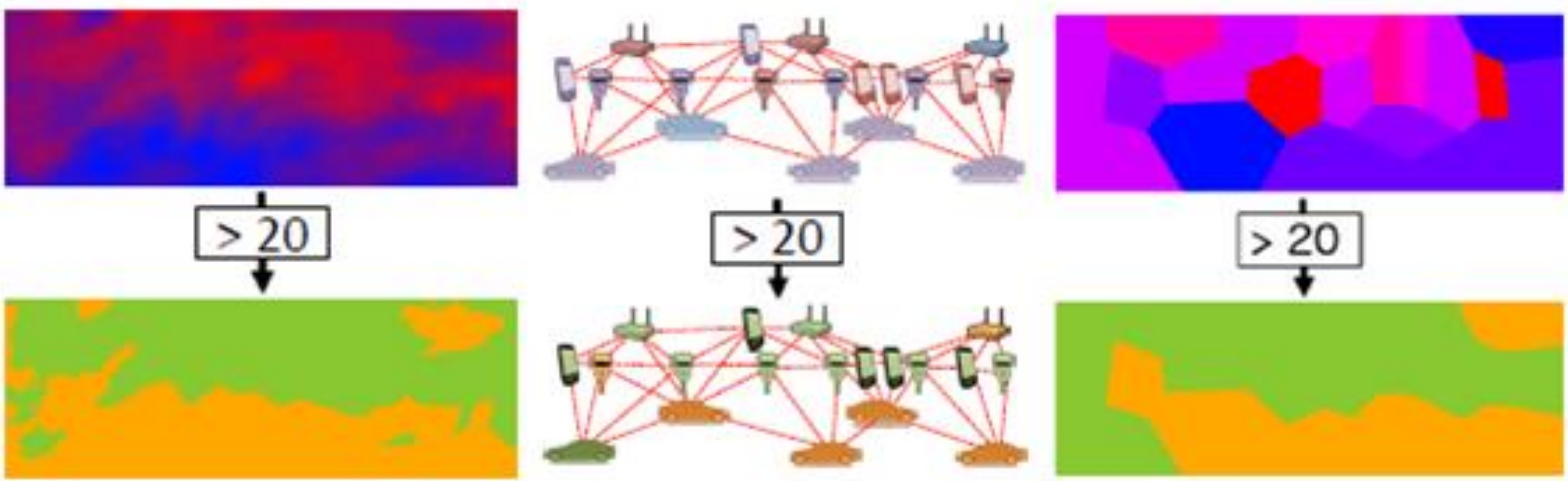
Field Calculus: modelling complex systems

Example of continuous space-time relations between six devices distributed along a street: wireless access points and meters are stationary, while the car moves steadily and the phone stops and starts twice



Field Calculus: an example

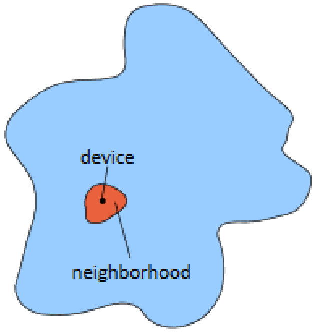
- continuous computation of a temperature threshold (a)
- approximated by the discrete network of devices (b)
- producing an approximation (c)

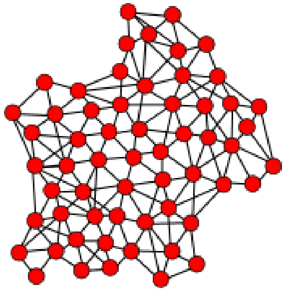


(a) Continuous (b) Discrete (c) approximation

Field Calculus: overview

- Programs executed by a network of devices (a dynamic neighboring relation represents physical or logical proximity)
- Every device is given the same (terminating) program, iterated in asynchronous computation rounds
- Each program iteration comprises:
  1. gathering of messages from other related/nearby devices
  2. perception of contextual information through sensors
  3. storing local state of computation
  4. computing a new local state
  5. dispatching messages to neighbors
  6. executing some form of actuation





Field Calculus: syntax

$\mathrm{P} ::= \bar{\mathrm{F}}\ \mathrm{e}$	*program*
$\mathrm{F} ::= \textbf{def}\ \mathrm{d}(\bar{\mathrm{x}})\ \{\ \mathrm{e}\ \}$	*function declaration*
$\mathrm{e} ::=$	*expression*
$\quad \mathrm{x} \mid \mathrm{v} \mid \mathrm{f}(\bar{\mathrm{e}})$	*variable, value, function application*
$\quad \mid\ \textbf{rep}(\mathrm{e})\{(\mathrm{x}) \Rightarrow \mathrm{e}\}$	*time evolution (feedback)*
$\quad \mid\ \textbf{nbr}\{\mathrm{e}\}$	neighbourhood field *construction (includes device itself)*
$\quad \mid\ \textbf{if}(\mathrm{e})\{\mathrm{e}\}\{\mathrm{e}\}$	space restriction
$\mathrm{v} ::= \ell \mid \phi$	*value*
$\ell ::= \mathrm{c}(\bar{\ell})$	*local value*
$\phi ::= \bar{\delta} \to \bar{\ell}$	*neighbouring field value (arising at runtime)*
$\mathrm{f} ::= \mathrm{d} \mid \mathrm{b}$	*function (**d**efined or **b**uilt-in) name*

NOTATION: $\bar{\mathrm{F}}$ denotes a sequence "$F_1\ F_2 \ldots F_n$" ($n \geq 0$); $\bar{x}$ denotes a sequence "$x_1, x_2, \ldots x_n$" ($n \geq 0$); ...

Field Calculus: semantics

- b(e_1,...,e_n): applies built-in function b to arguments e_1,...,e_n. Built-in functions are stateless mathematical, logical, or algorithmic functions, sensors or actuators
- **rep**(e_1){(x)=>e_2}: defines a local state variable x initialized with value v. Updated at each round with the result of executing its body e, thereby defining a field that evolves over time
- **nbr**{e}: gathers a map at each device (actually, a field) from all neighbors (including itself) to their latest value of s. Built-in"hood" functions then summarize such maps, e.g., min-hood(m) finds the minimum value in map m (excluding the value associated to the device itself)
- **if**(e_0){e_1}{e_2}: partitions the network into two regions: where e_0 is true e_1 is computed, elsewhere e_2 is computed instead. Importantly, partition implies branches are encapsulated and cannot have effects outside their subspace

Field Calculus: examples

- A program that computes whether temperature is high:

```
temperature() > 20                                  //  bool
```

- A program that ***shows*** whether temperature is high:

```
if (temperature() > 20)  {set-led("green")}  {set-led("orange")}  // unit
```

- A program that counts the number of rounds in each device:

```
rep 0 { (x)  =>  x + 1 }
```

- A program that computes whether any neighbor has a high temperature:

```
any-hood (nbr { temperature() > 20 })               //  bool
```

// any-hood: maps each device to ``whether any of its neighbor (excluding itself) has value true"

- A program that computes whether any location has ever experienced a high temperature:

```
def gossip-ever (value) {                          // (bool) ⟶ bool
    rep (false) { (ever) =>
                    ever or value or (any-hood (nbr ever))
    }
}
gossip-ever (temperature() > 20)
```

- A program that computes the distance to a region with a high temperature:

```
def distance-to (source) {                        //  (bool) ⟶ num
    rep (infinity) {  (d)  =>
            mux ( source, 0, min-hood(nbr{d} + nbr-range()) )
    }
}

distance-to (temperature() > 20)
```

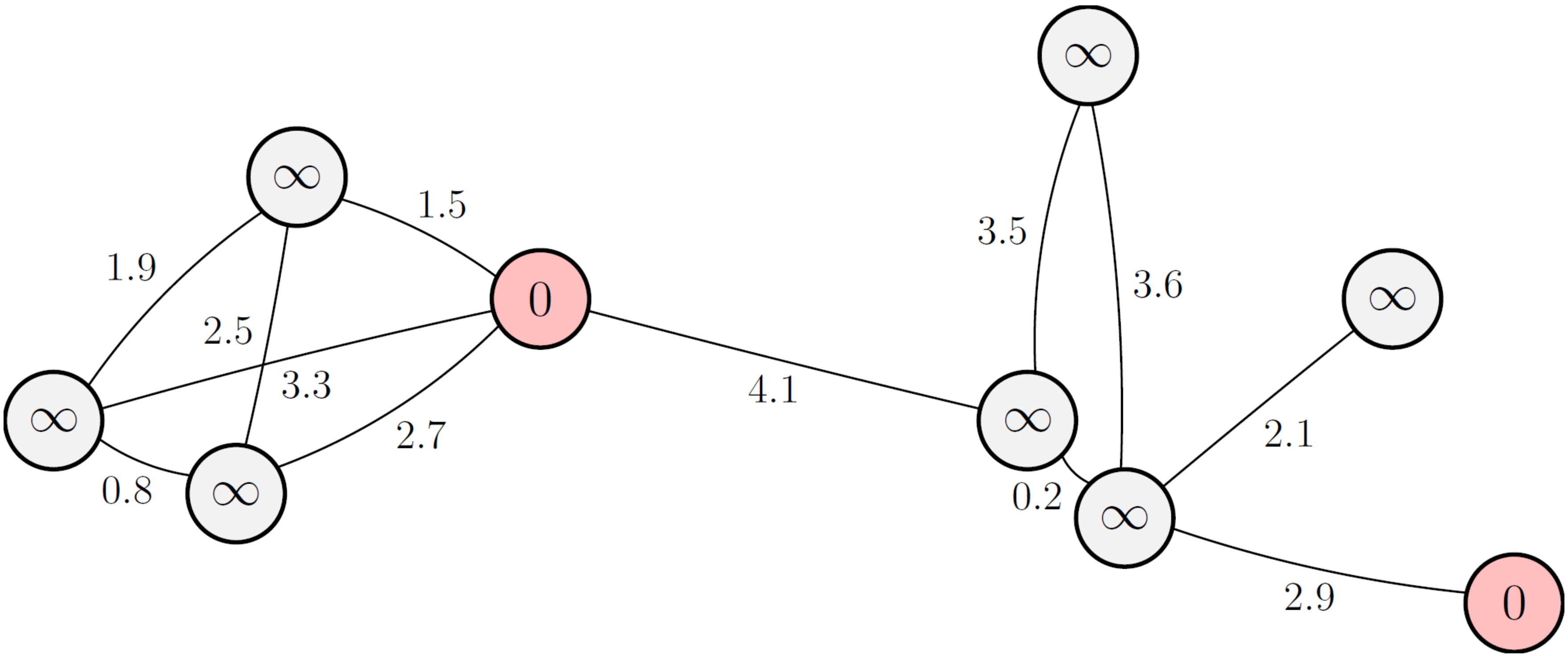
// mux: maps each device to the value of its second argument, if the value first argument is true,

//   and to the value of its third argument, otherwise

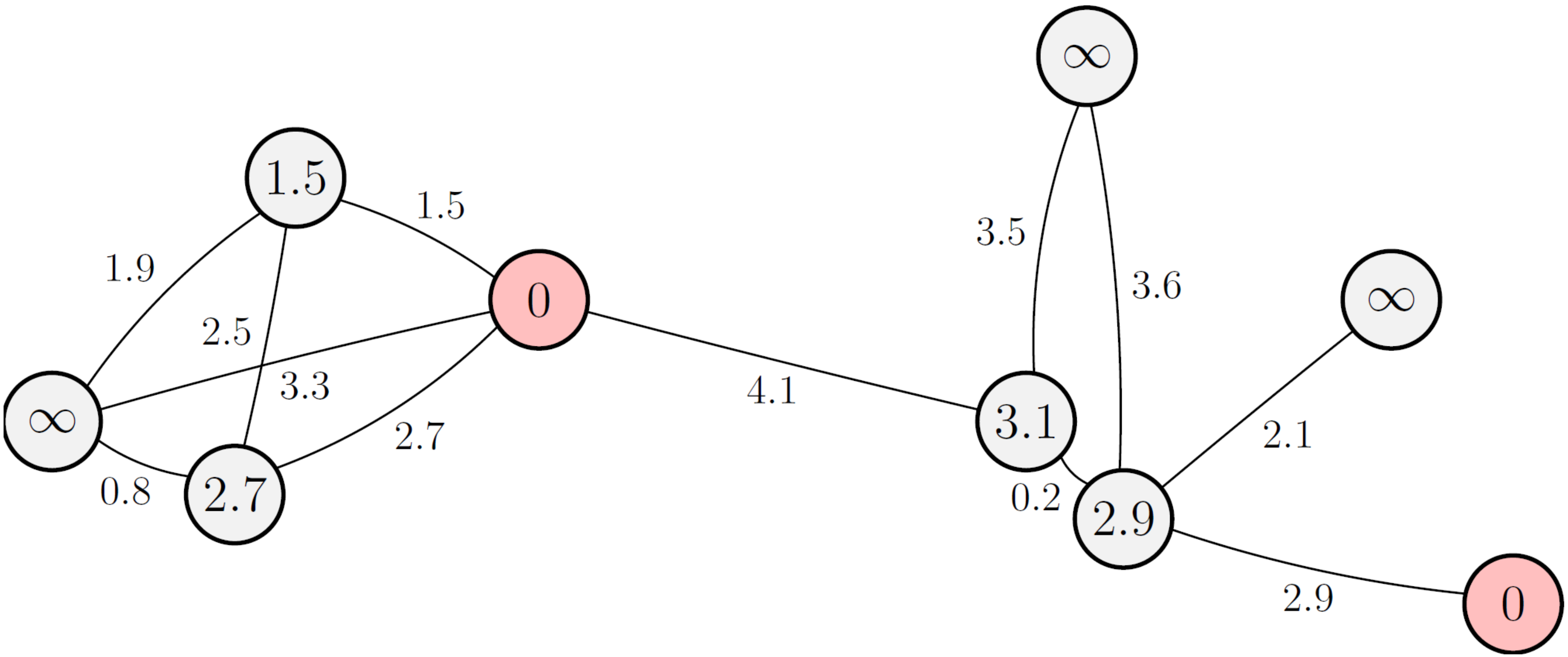
// min-hood: maps each device to the minimum value of its neighbor (excluding itself)

// nbr-range: maps each device to field of distances to its neighbors

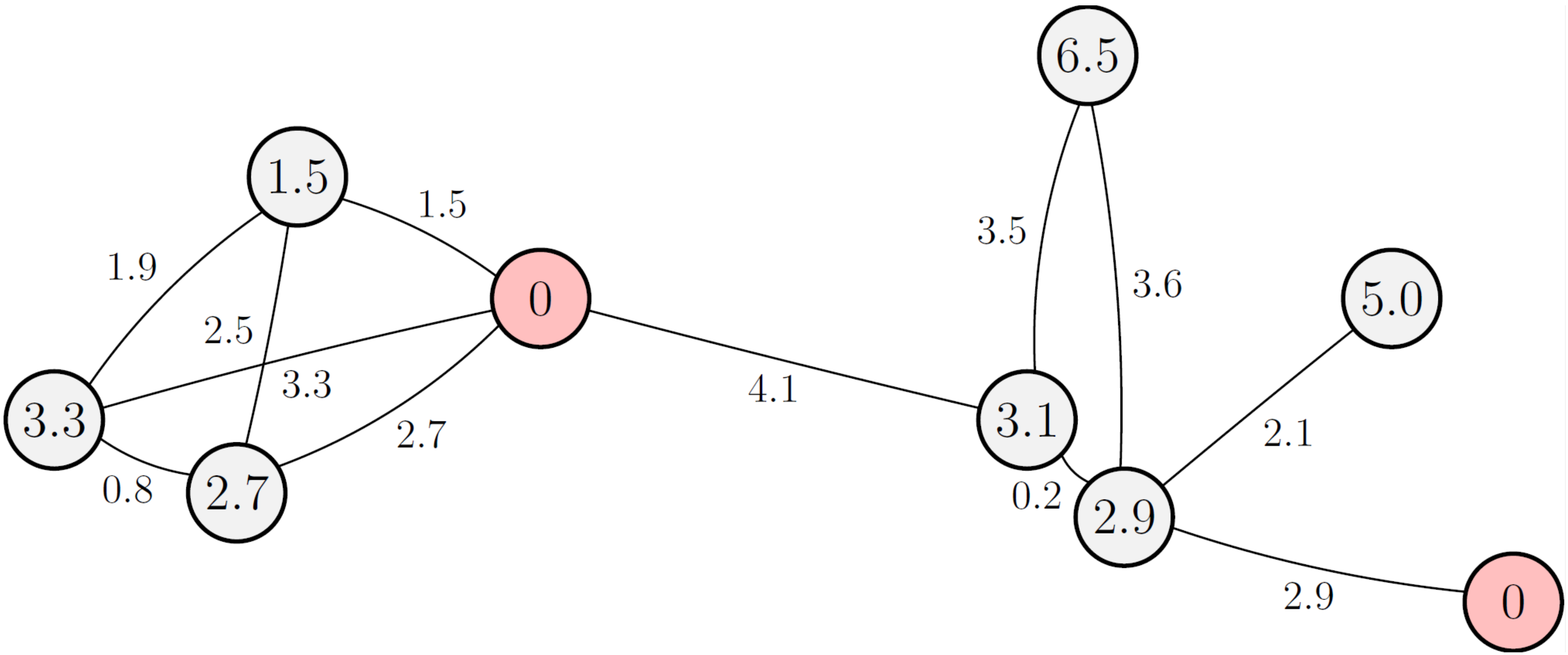
```
def  distance-to (source) {                          //  (bool) ⟶ num
     rep (infinity) {  (d)  =>
            mux ( source, 0, min-hood(nbr{d} + nbr-range()) )
     }
}
```



```
def  distance-to (source) {                        //  (bool) ⟶ num
     rep (infinity) {  (d)  =>
              mux ( source, 0, min-hood(nbr{d} + nbr-range()) )
     }
}
```



```
def distance-to (source) {                     // (bool) ⟶ num
    rep (infinity) { (d) =>
        mux ( source, 0, min-hood(nbr{d} + nbr-range()) )
    }
}
```



- A function that broadcasts a value from a source:

```
def  broadcast (source, value) {                          //  (bool, num) ⟶ num
        snd (rep (pair(infinity, value)) { (old) =>
                mux(source, pair(0, value), min-hood(nbr{pair(fst(old) + 1, snd(old))}))
                })
}
```

- A function that computes the distance:

```
def  distance (source, destination) {                     //  (bool, bool) ⟶ num
        broadcast(source, distance-to(destination))
  }
```

- A function that computes a channel:

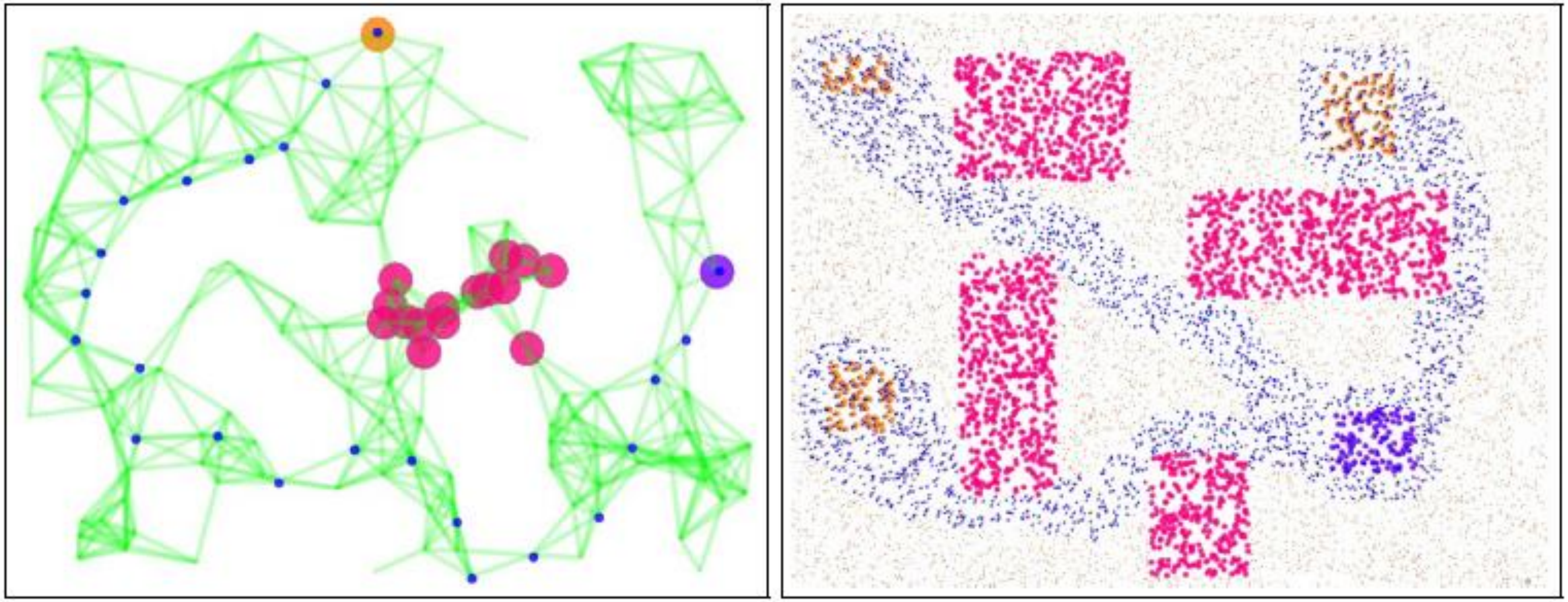
```
def  channel (source, destination, width) {      //  (bool, bool, num) ⟶ bool
   distance-to(source) + distance-to(destination)
   <
   width + distance(source, destination)
}
```

- A program that computes a channel avoiding obstacles:

```
def  channel-avoiding-obstacles (o, s, d, w) {
                                   //  (bool, bool, bool, num) → bool
   if (o) { false } { channel(s,d,w) }
}
```

channel-avoiding-obstacles ($\boldsymbol{e}_{\boldsymbol{obstacle}}$, $\boldsymbol{e}_{\boldsymbol{source}}$, $\boldsymbol{e}_{\boldsymbol{destination}}$, $\boldsymbol{e}_{\boldsymbol{width}}$)

Channels (blue) route between source (orange) and destination (purple) around obstacles (pink), deployed in a low-density network with topology (green) in evidence (left), and in a high-density environment of 10,000 nodes (right).



- A wrong program (mux does not interfere with channel computation):

```
def  wrong-channel-avoiding-obstacles (o, s, d, w) {
                                   //  (bool, bool, bool, num) ⟶ bool
   mux (o) { false } { channel(s,d,w) }
}
```

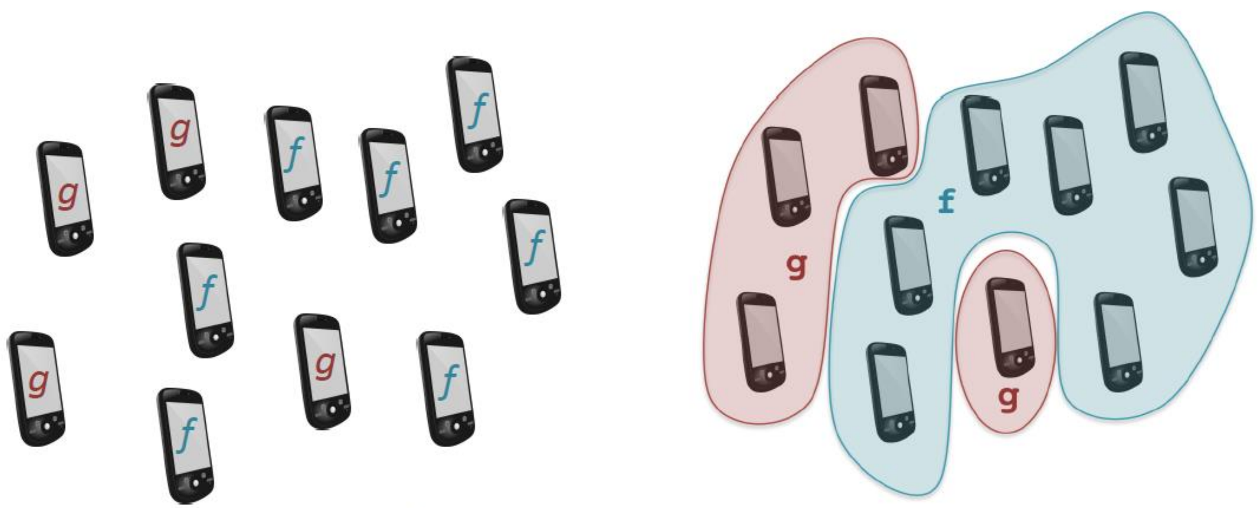
wrong-channel-avoiding-obstacles ($\boldsymbol{e}_{\boldsymbol{obstacle}}$, $\boldsymbol{e}_{\boldsymbol{source}}$, $\boldsymbol{e}_{\boldsymbol{destination}}$, $\boldsymbol{e}_{\boldsymbol{width}}$)

Field Calculus: higher-order functions

Higher order functions support:

- Devices running different programs
- Runtime updates

✓ Damiani, F., Viroli, M., Pianini, D., Beal, J. 2015. **Code Mobility Meets Self-organisation: A Higher-Order Calculus of Computational Fields**. In Proceedings of FORTE 2015. LNCS, Vol. 9039, pp 113–128, Springer. DOI: 10.1007/978-3-319-19195-9_8

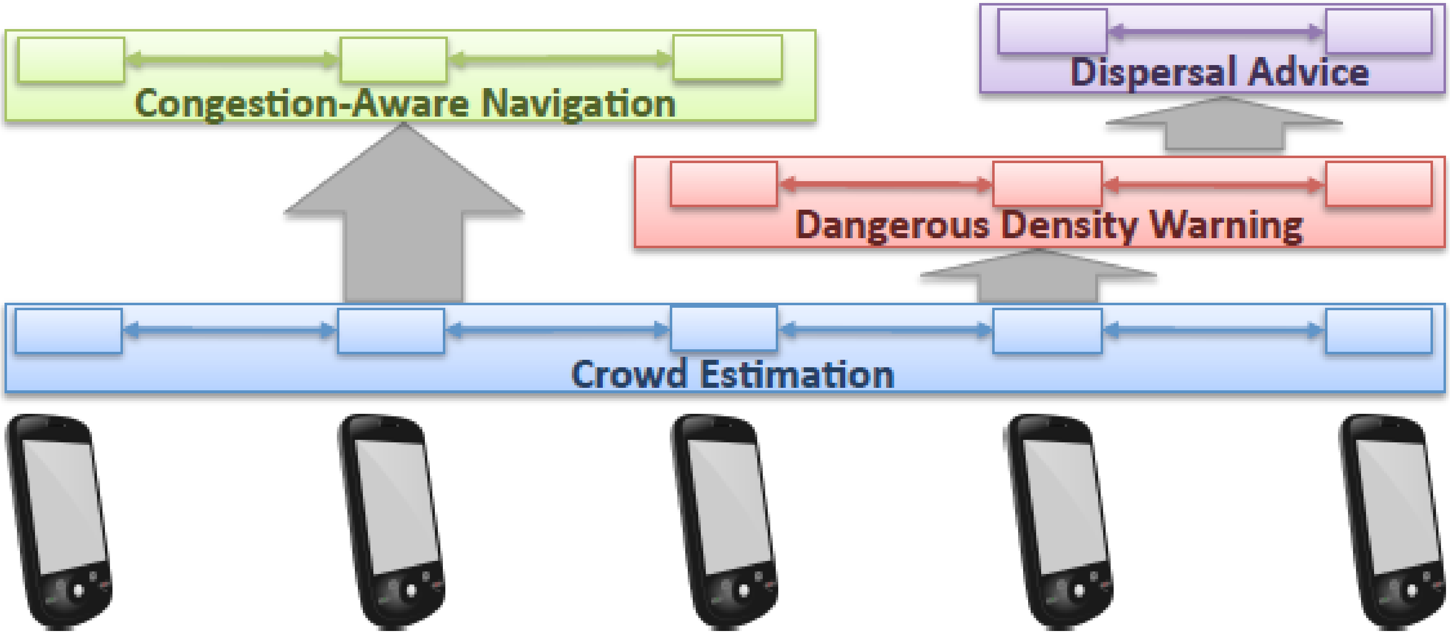


(a) A field of functions hosted on individual devices

(b) The same field interpreted as an aggregate function defined piecewise over aggregates of devices

Field Calculus: programming

- Implicit adaptation and communication
- Code each collective service independently
- Compose via scope and information flow



Aggregate Computing: two survey papers

- ✓ Jacob Beal, Stefan Dulman, Kyle Usbeck, Mirko Viroli, and Nikolaus Correll. 2013. **Organizing the Aggregate: Languages for Spatial Computing.** In Formal and Practical Aspects of Domain-Specific Languages: Recent Developments, Marjan Mernik (Ed.). IGI Global, Hershey, PA, Chapter 16, 436–501. A longer version available at: http://arxiv.org/abs/1202.5509
- ✓ Mirko Viroli, Jacob Beal, Ferruccio Damiani, Giorgio Audrito, Roberto Casadei, and Danilo Pianini, 2018. **From Field-Based Coordination to Aggregate Computing.** In 20th IFIP WG 6.1 International Conference, COORDINATION 2018, Held as Part of the 13th International Federated Conference on Distributed Computing Techniques, DisCoTec 2018, Madrid, Spain, June 18-21, 2018. Proceedings. DOI: 10.1007/978-3-319-92408-3

Field Calculus: some results

- Expressing algorithms of collective adaptation as reusable blocks
  - ✓ Jacob Beal, Danilo Pianini, Mirko Viroli. **Aggregate Programming for the Internet of Things**. IEEE Computer 48(9): 22-30 (2015). DOI: 10.1109/MC.2015.261
- Type soundness (for the first order calculus)
  - ✓ Ferruccio Damiani, Mirko Viroli, Jacob Beal. **A type-sound calculus of computational fields**. Science of Compututer Programming 117: 17-44 (2016). DOI: 10.1016/j.scico.2015.11.005
- Expressivity
  - ✓ Giorgio Audrito, Jacob Beal, Ferruccio Damiani, Mirko Viroli. **Space-Time Universality of Field Calculus**. In 20th IFIP WG 6.1 International Conference, COORDINATION 2018, Held as Part of the 13th International Federated Conference on Distributed Computing Techniques, DisCoTec 2018, Madrid, Spain, June 18-21, 2018. Proceedings. DOI: 10.1007/978-3-319-92408-3

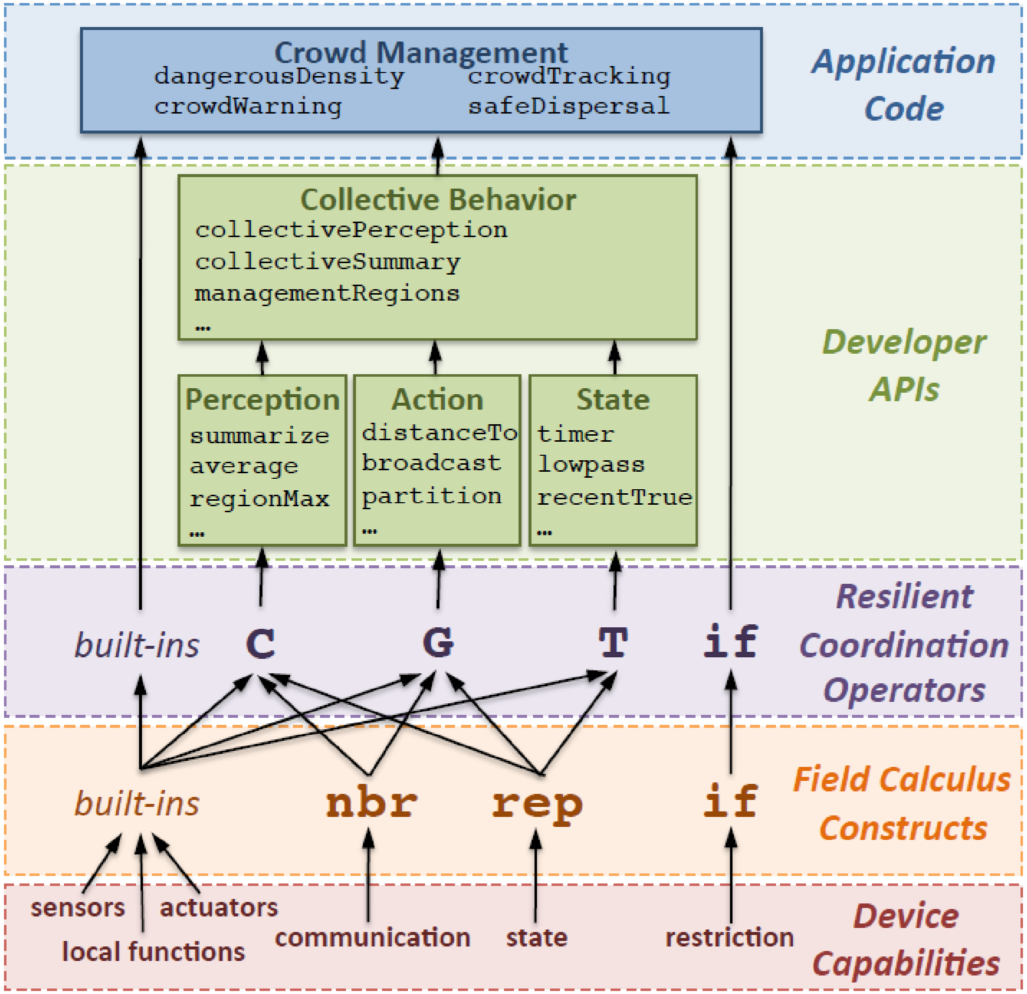
- Structuring computations in a way that is effectively independent of devices number and location (for the first order calculus)
  - ✓ Jacob Beal, Mirko Viroli, Danilo Pianini, Ferruccio Damiani. **Self-Adaptation to Device Distribution in the Internet of Things**. ACM Transactions on Autonomous and Adaptive Systems 12(3): 12:1-12:29 (2017). DOI: 10.1145/3105758

- Intrinsic resiliency to changes in the environment (for the first order calculus)
  - ✓ Mirko Viroli, Giorgio Audrito, Jacob Beal, Ferruccio Damiani, Danilo Pianini. **Engineering Resilient Collective Adaptive Systems by Self-Stabilisation**. ACM Transactions on Modeling and Computer Simulation. 28(2): 16:1-16:28 (2018). DOI: 10.1145/3177774

- Type soundness, denotational semantics and computational adequacy (for the higher-order calculus)
  - ✓ Giorgio Audrito, Mirko Viroli, Ferruccio Damiani, Danilo Pianini, Jacob Beal. **A Higher-order Calculus of Computational Fields**. ACM Transactions on Computational Logic. 20(1): 5:1-5:55 (2019). DOI: 10.1145/3285956

Field Calculus: towards production engineering

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



Field Calculus: implementations

- **Proto** [proto.bbn.com] since 2008
  - An aggregate programming language based on dialect of SCHEME
  - Created by Jacob Beal and Jonathan Bachrach before of the Field Calculus

- **Protelis** [protelis.github.io] since 2015
  - Practical aggregate programming, hosted in Java

- **ScaFi** (**Sca**la with Computational **Fi**elds)

  [https://scafi.github.io] since 2016
  - An aggregate programming framework for Scala. It provides:
    1. a Scala-internal DSL for expressing aggregate computations
    2. a distributed platform supporting the configuration and execution of aggregate systems

Both Protelis and ScaFi can be connected to the simulator Alchemist [https://alchemistsimulator.github.io].

Field Calculus: a language for spatial computers

A spatial computer is a collection of computational devices distributed through a physical (or logical) space in which:

- the difficulty of moving information between any two devices is significantly dependent on the distance between them, and
- the "functional goals" of the system are somehow defined in terms of the system's spatial structure

Field Calculus: some application scenarios

- Crowd detection and steering
- Enhancing driver assistance systems
- General traffic, smart logistic and public transportation management
- Ambulance, fire fighters, public security, etc. fast track
- Alarm management and disaster recovery
- Environmental (air quality, soil, water, etc.) and agricultural monitoring
- ...

Field Calculus: future work

- **Scafi** comes with:
  - a *statically configurable* middleware where computations are carried out either *totally* in the Cloud or *totally* in the IoT
  - ✓ Viroli, M., Casadei, R., Pianini, D. 2016. On execution platforms for large-scale aggregate computing. In Proceedings of ACM UbiComp '16: Adjunct, pp 1321-1326. DOI: 10.1145/2968219.2979129
- We would like to develop an execution strategy where:
  - computations are carried out *partially* in the Cloud and *partially* in the IoT/Edge/Fog (cf. www.openfogconsortium.org), and
  - *dynamically* flow up and down depending upon context and contingencies

Field Calculus: future work

A uniform approach for IoT/Edge/Fog/Cloud

- Dynamically supports scalable computations in cluster- and cloud-based systems
  - Same benefits of frameworks such as Hadoop and Spark (Aggregate Computing plays the role of MapReduce)
  - Intrinsically supports distributed and situated computations (while MapReduce is limited to data processing)
  - Supports unanticipated runtime software updates

Field Calculus: future work

Programming paradigm for IoT/Fog/Edge/Cloud

- Coping with:
  - Ambient Intelligence and ubiquitous computing
  - Hybrid wireless sensor networks that are characterized by modularity, reliability, flexibility, robustness and scalability
  - Wireless monitoring of different ambient parameters (video, audio, temperature, light, humidity, smoke, air quality, radiation, energy, etc.)
  - Mobile robotic sensor networks
  - Hybrid wireless sensor networks that enable context and situation based personalized applications and services
  - User context identification: Biometrics; Privacy mood; Attention; Gesture, Posture
  - Social context: Surrounding people and/or objects/things; Type of group; Link to people and/or objects/things
  - Environmental context: Location, position; Time; Condition; Physical data