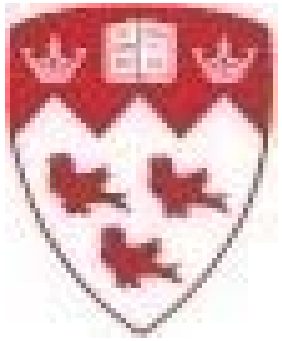
Game-Aware Data Dissemination

Bettina Kemme
School of Computer Science

Cesar Cañas, Jörg Kienzle, Kaiwen Zhang, and Arno Jacobsen



McGill University

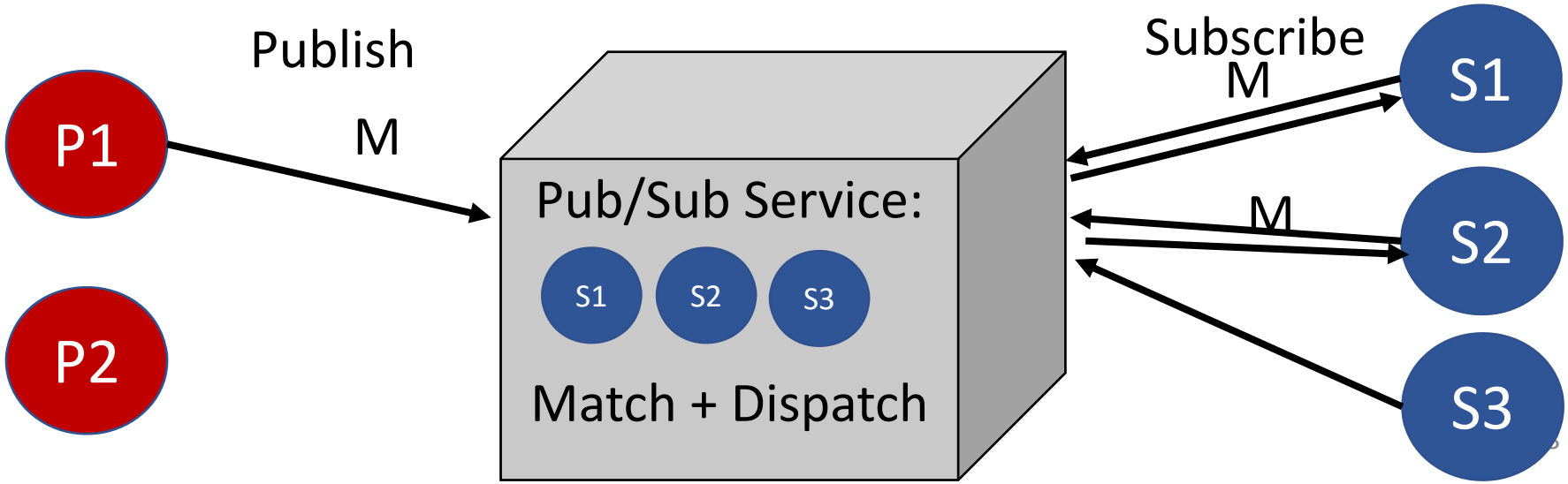
Multiplayer Online Games (MOG)

- From a few players up to thousands
  - Big business
  - Demanding
  - ….
- Players receive each other's state changes
- But not of all – only the ones they are interested in!

- They need pub/sub!

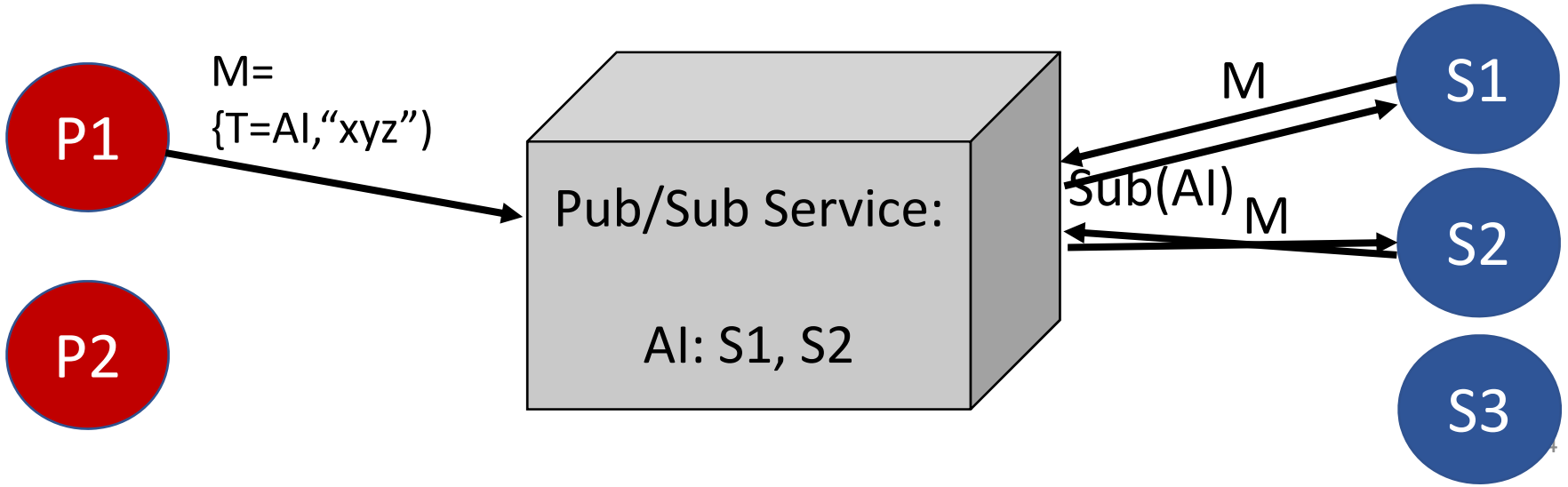
Publish/Subscribe Paradigm

- Decouples content producers (publishers) from content consumers (subscribers)
- Subscribers only receive publications they are interested in
- Many flavors of publish/subscribe



Most common: topic-based pub/sub

- Subscription language: a topic T
- Publications tagged with a topic T, sent to all subscribers of T
- Matching process: very simple (hash table lookup)



Content-based pub/sub

- Subscriptions are queries over publication content
- Content has “data model”
- Attribute – based

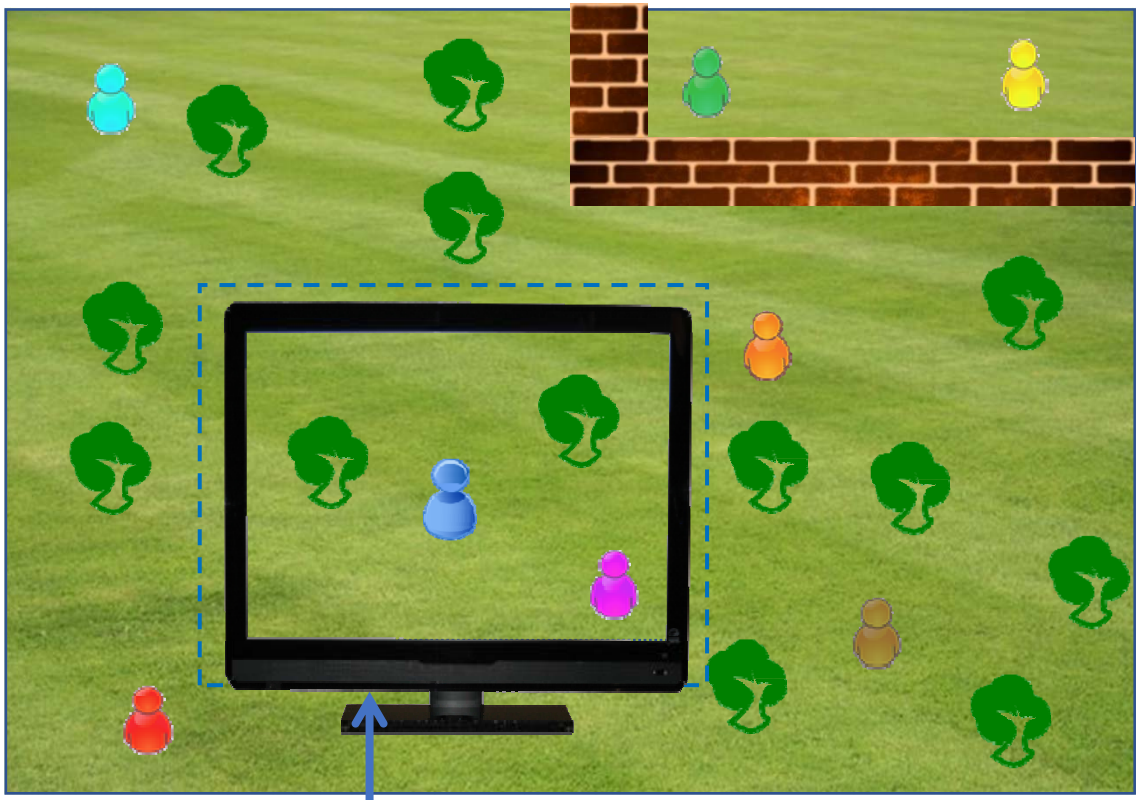
P((stock, IBM), (price, 50))
P((stock, Oracle), (price, 100))
S(price <= 60)

Multiplayer Online Games (MOG)

- From a few players up to thousands
  - Big business
  - Demanding
  - ….
- Players receive each other's state changes
- But not of all – only the ones they are interested in!

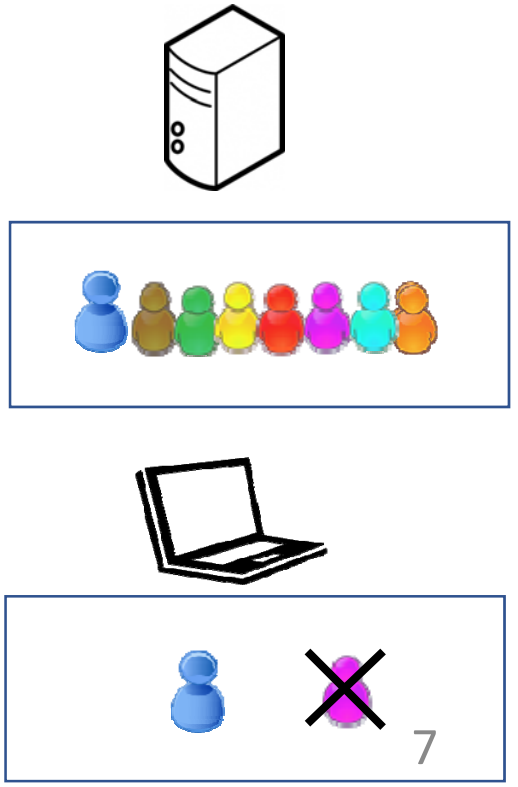
- They need pub/sub!

Interest and Replica Management



Area of Interest (AoI)

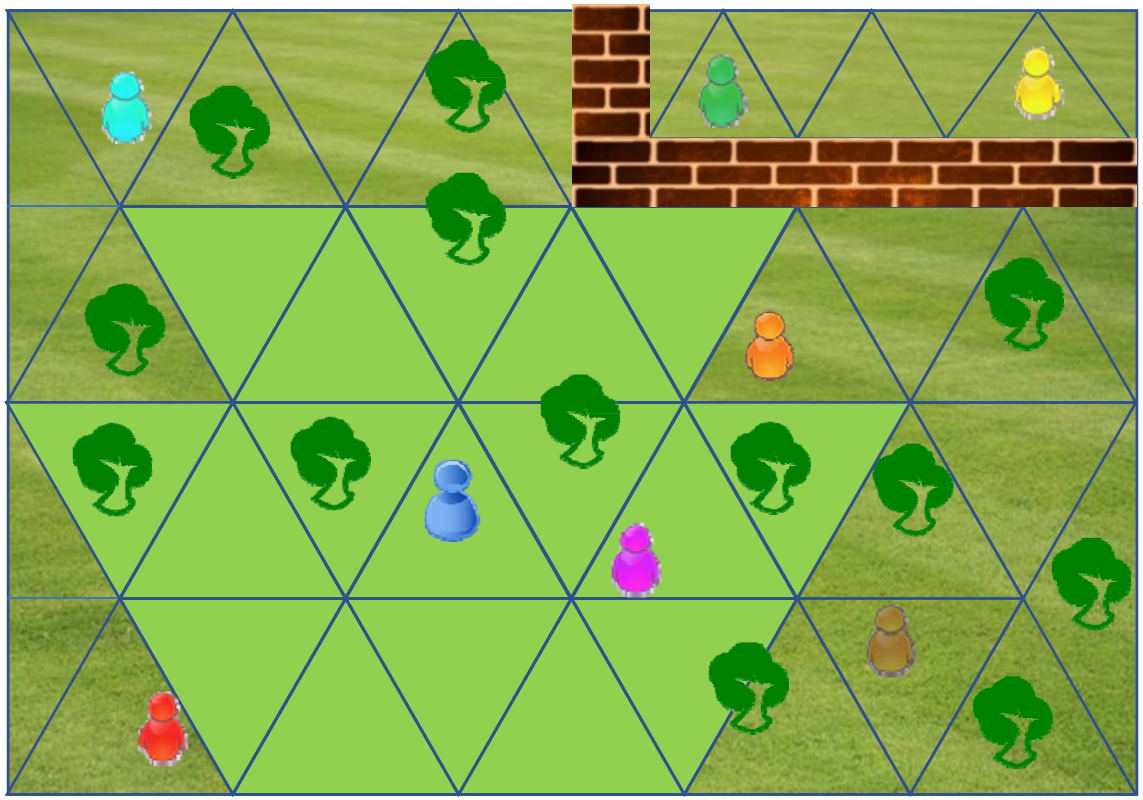
- Interest Management
- Replica Management
  - Ignore for this talk
- Update Propagation



7

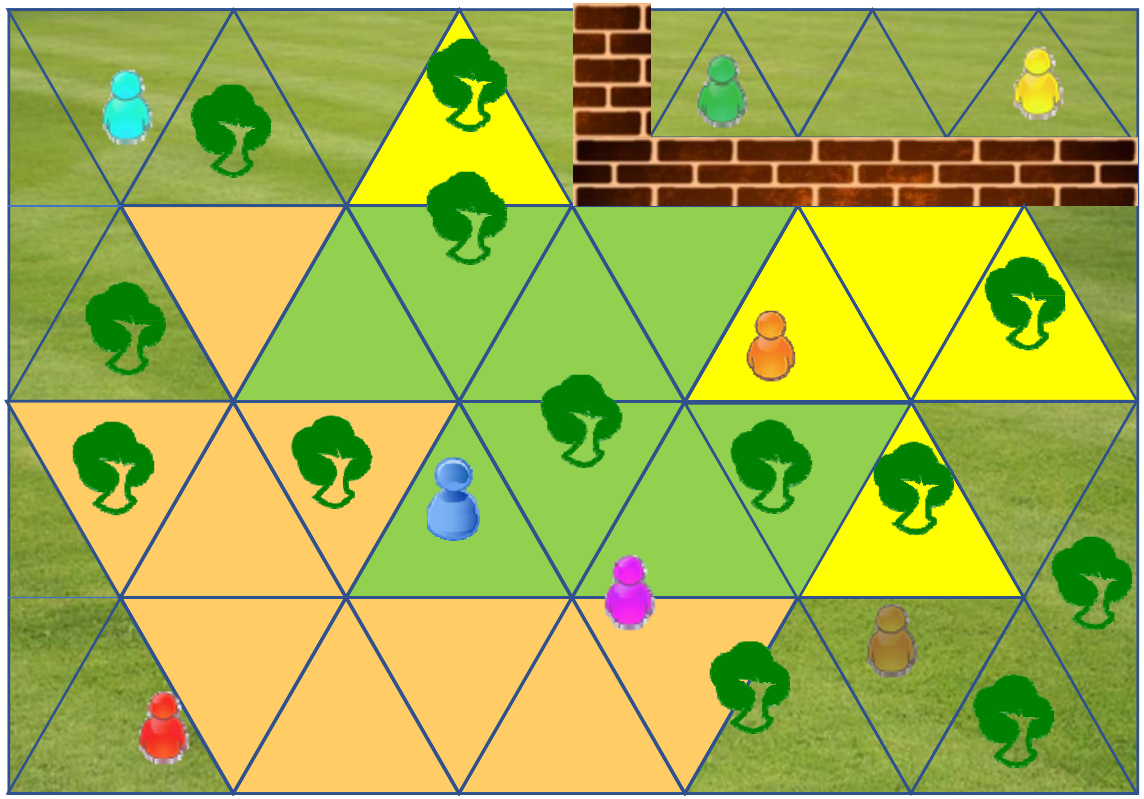
Version 1: Tiles and Topics

- game map divided into tiles
- player interested in all objects located in the tiles under its area of interest
  - Obstacle-Aware
- each tile a topic
- subscribe to tiles in AoI
- action: publish on current tile



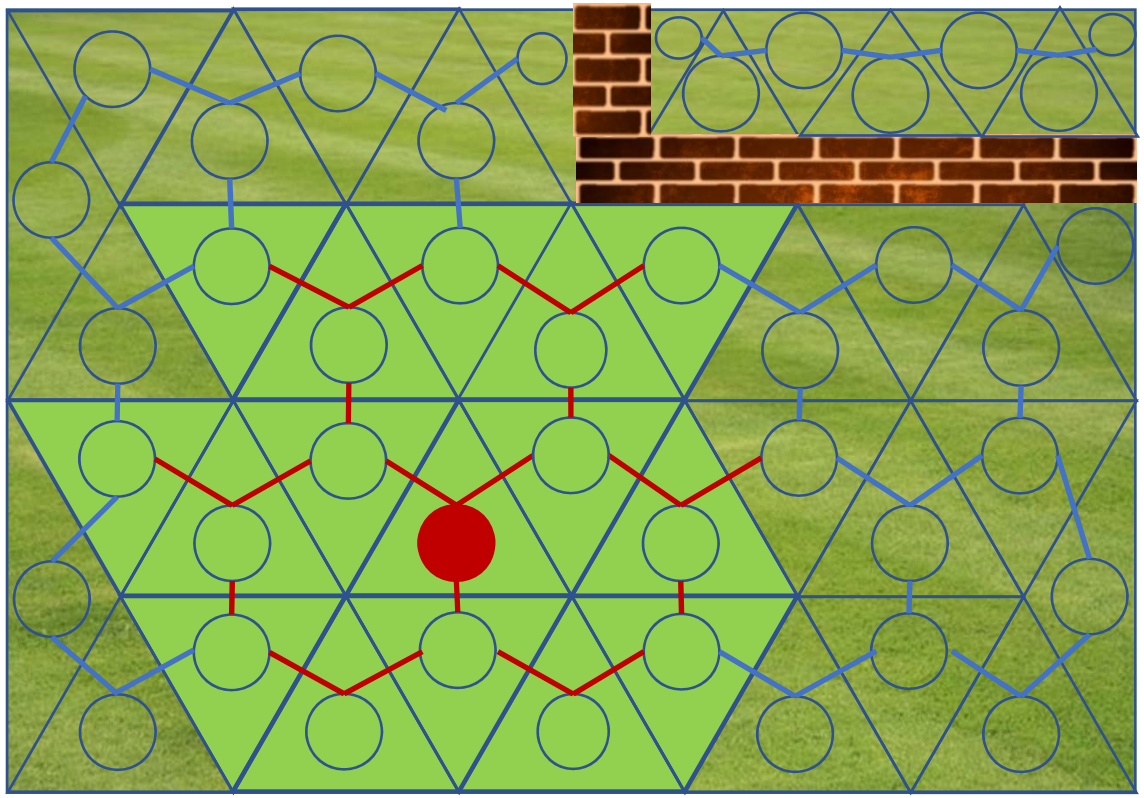
Challenge Player Movement

- Many unsubscriptions
- Many subscriptions



Idea: graph representation

- Application domain representation
  - From set of topics to a graph
  - Each node is triangle
  - Neighboring triangles are connected

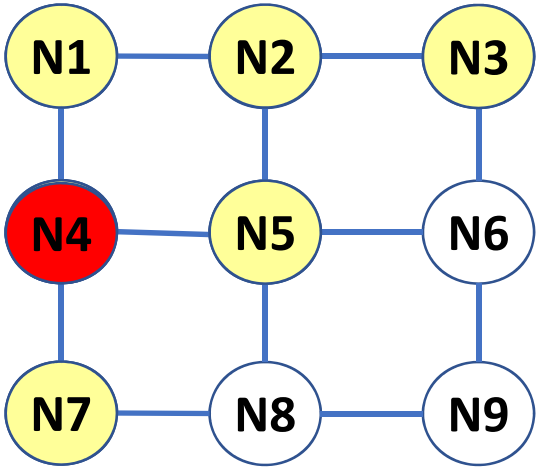


- Interest = Subscription:
  - *Interested in all actions up to 3 nodes away*
  - = Graph Query
- Action = Publication:
  - *On a node*

Graph-based pub/sub

- The application domain is represented as a graph or multiple graphs that are stored as meta-information in our system.

Graph G1



- **Subscriptions:**
  - Expressed as graph-query
  - Returns a sub-graph

```
Subscribe (G1,  hopDistance (N1,  2));
```

- **Publications:**
  - On a node / edge

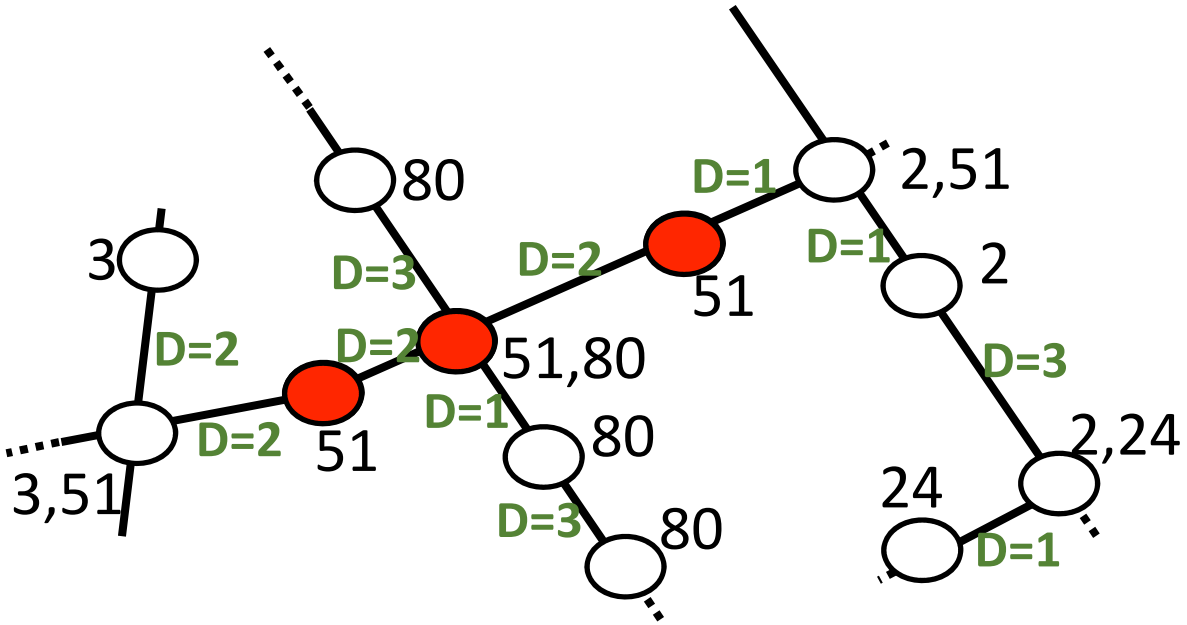
```
Publish (G1, N4, msg);
```


11

Street Maps

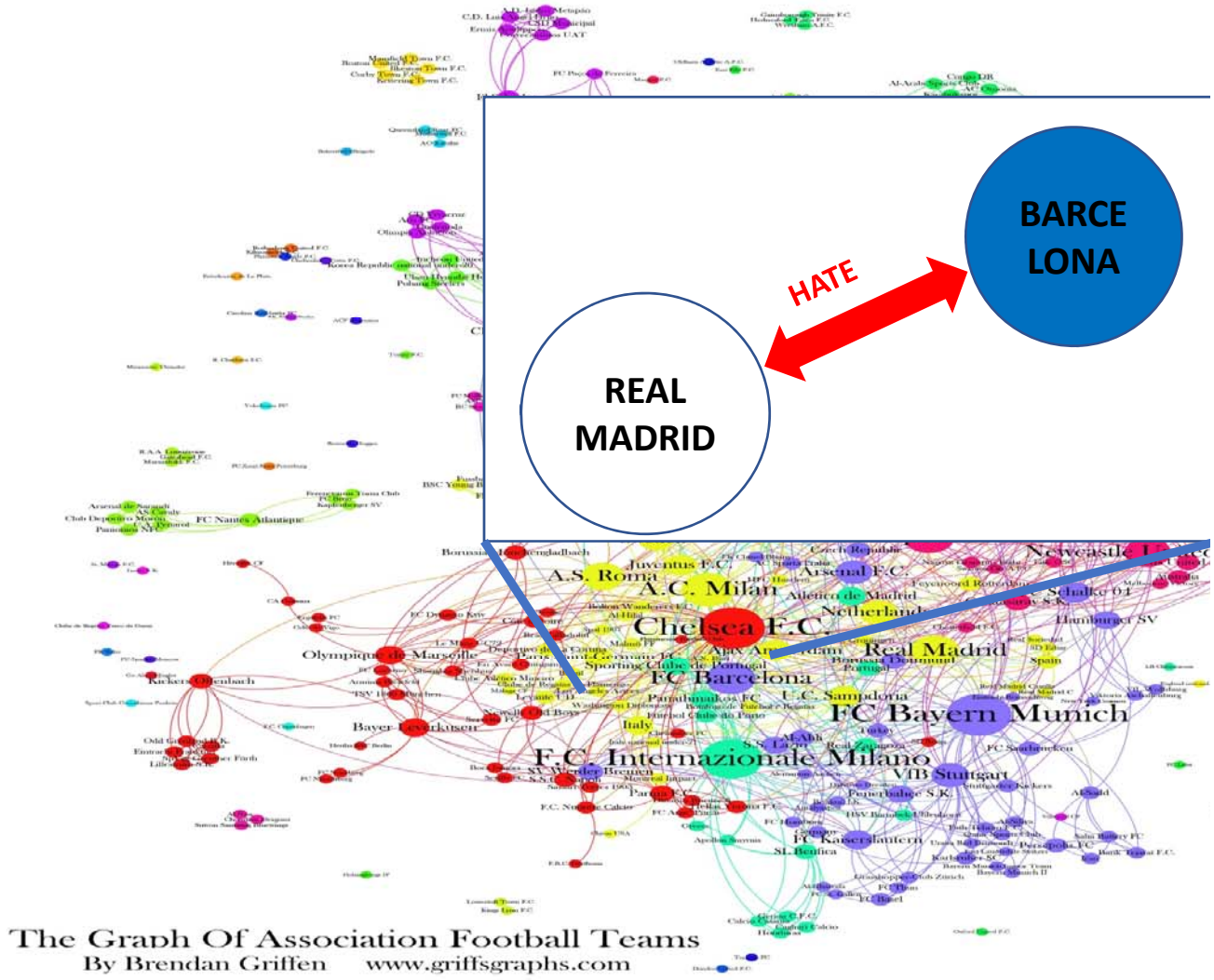
- Traffic Monitoring
  - Publications: Traffic Information on each segment
  - Subscriptions: Path to be travelled

Public Transportation graphs



- Subscriptions:
  - The 3 stops before my stop
  - Max Distance of 6 minutes from my stop
- Publications: Whenever a bus arrives at a bus stop

Knowledge Graphs

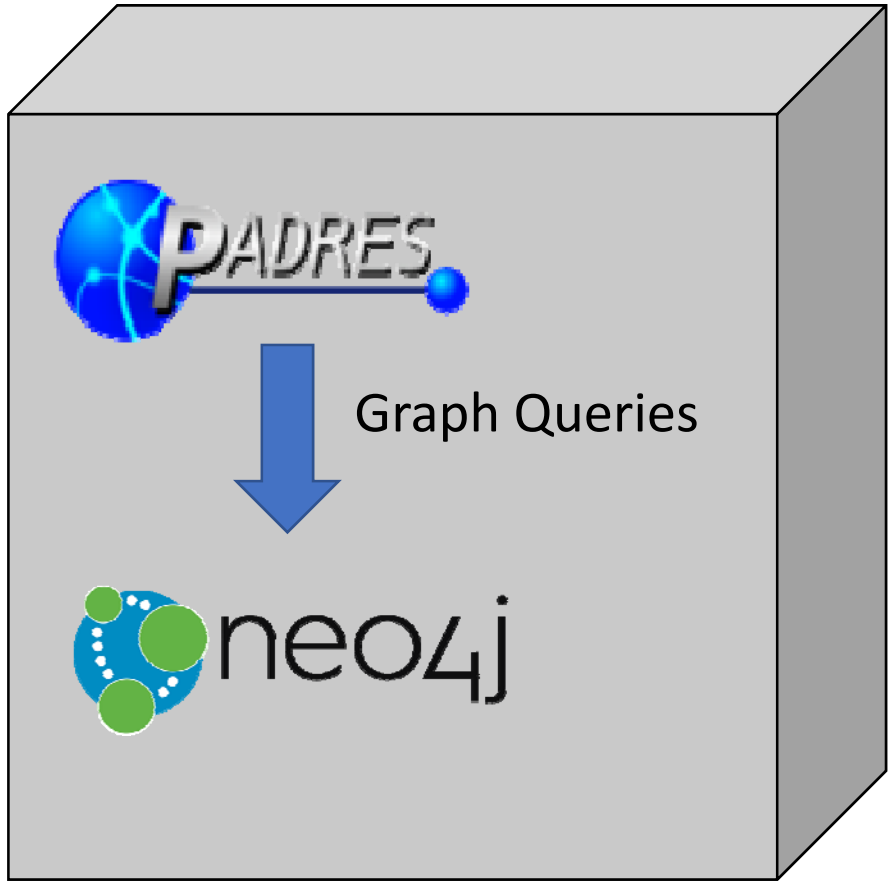


- Publication:
  - Info about a certain team
  - Published on a node
- Subscriptions:
  - all teams my team is related to
  - Graph query

Graps Implementation

- Publish
- Subscribe
- unsubscribe

- Store/delete Graph
- Update Graph

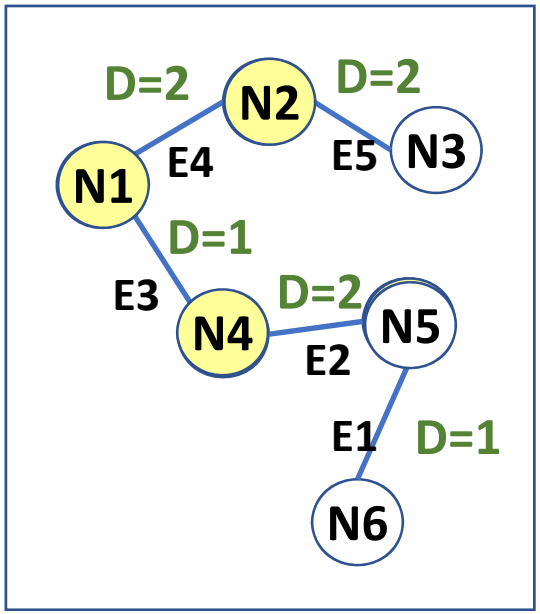


- Pub/sub engine
- Graph DBS backend

Graph Query

- Common Query API:
  - maxHops(nid, hops) ➔ Game Example
  - maxDistance(nid, distance) ➔ Bus Example
  - shortestPaths(nid1, nid2) ➔ Street Map example
  - neighbors(nid, label) ➔ Knowledge Graph example

Updating the Graph

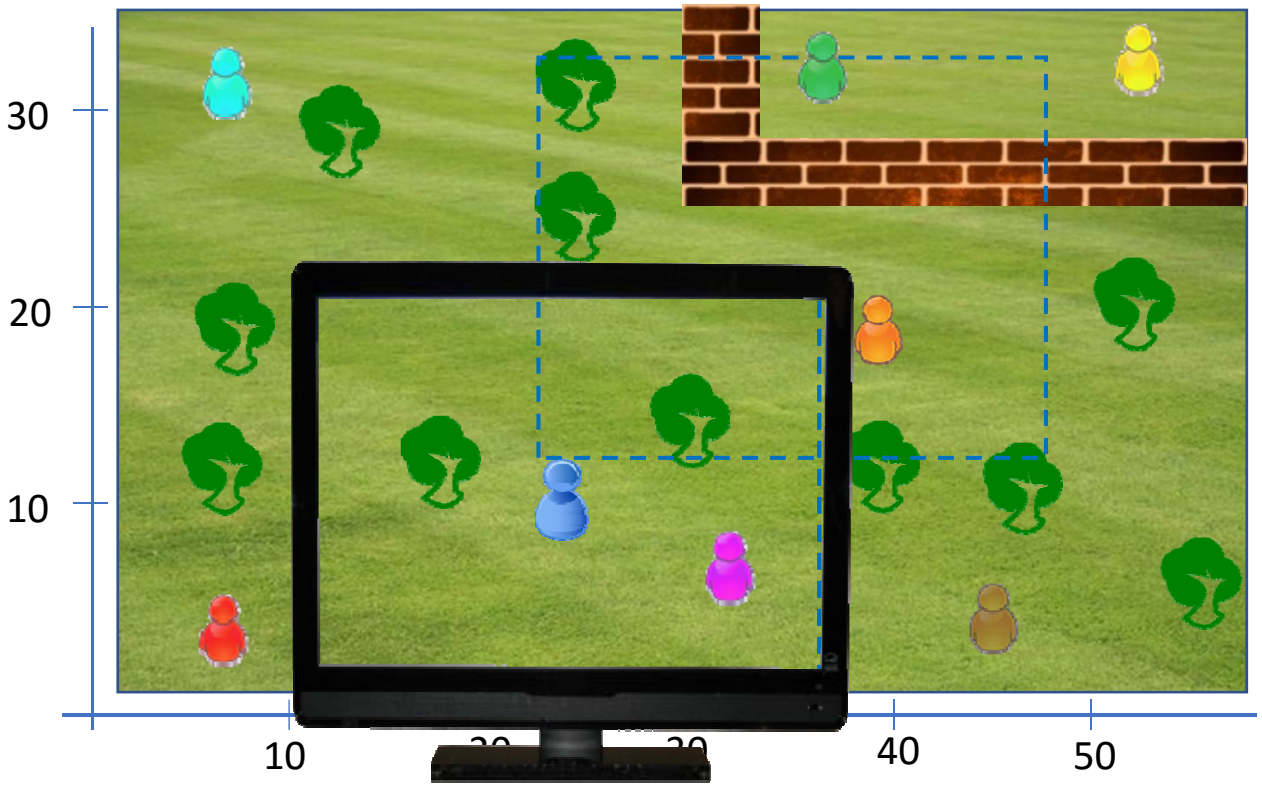


Subscribe(G1, maxDistance (N1, 3))

RemoveEdge(G1, E2)

- Subscriptions automatically updated when graph changes

Version 2: area-based pub/sub

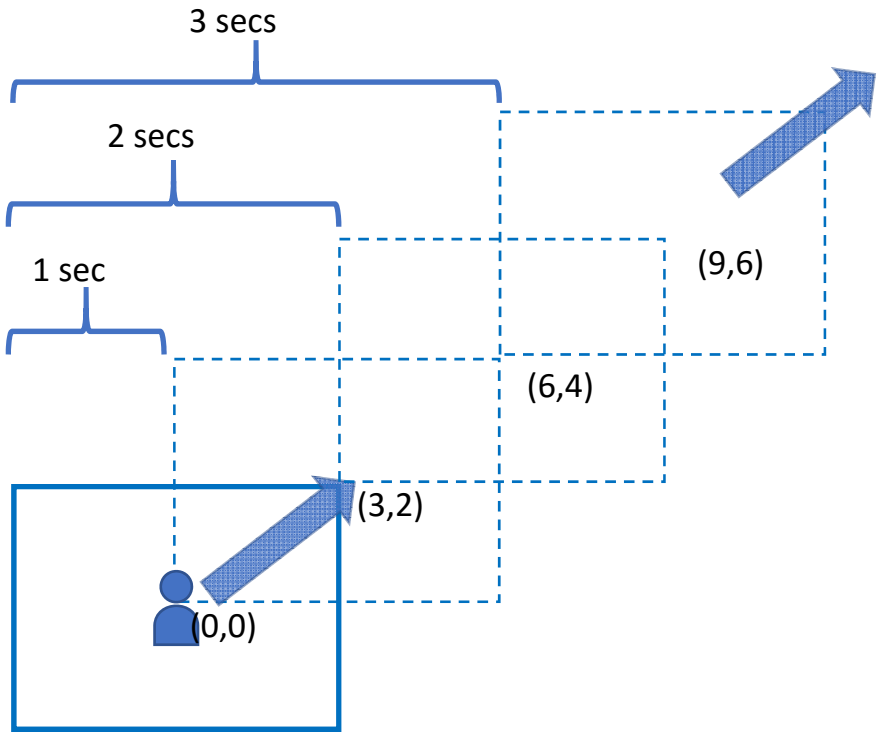


1 **Blue: Subscribe (11 < x < 36, 1 < y < 21)**

2 **Pink: Publish (x=32, y=8, "action info")**

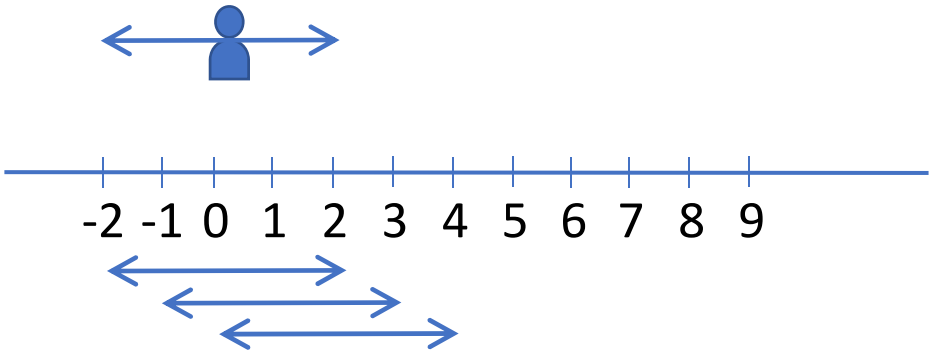
3 **Blue: Unsubscribe (11 < x < 36, 1 < y < 21)**
Subscribe(21 < x < 48, 12 < y < 32)

Moving Range Subscriptions



- **Dead-reckoning techniques**
  - **Automatically determine subscription range changes.**
- **Broker can "evolve" subscription on behalf of client**
- **Avoid un- and resubscription**
- **No action needed from client**

Moving Range Subscription



Assume: move on average one step per second

Regular Subscription

s: {(x>=-2),(x<2)}

s: {(x>=-1),(x<3)}

s: {(x>=0),(x<4)}

Evolving Subscription

$f_1(t) = -2 + t$
$f_2(t) = t + 2$

s: {(x>= $f_1(t)$),
(x< $f_2(t)$)}

Application Driven System Development

- Game centric view
- Detect game application needs
- Find generic solution
- Make it work

THANK YOU

Demo

Applications

Traffic alert systems

Weather alert systems

Mobile notif. frameworks

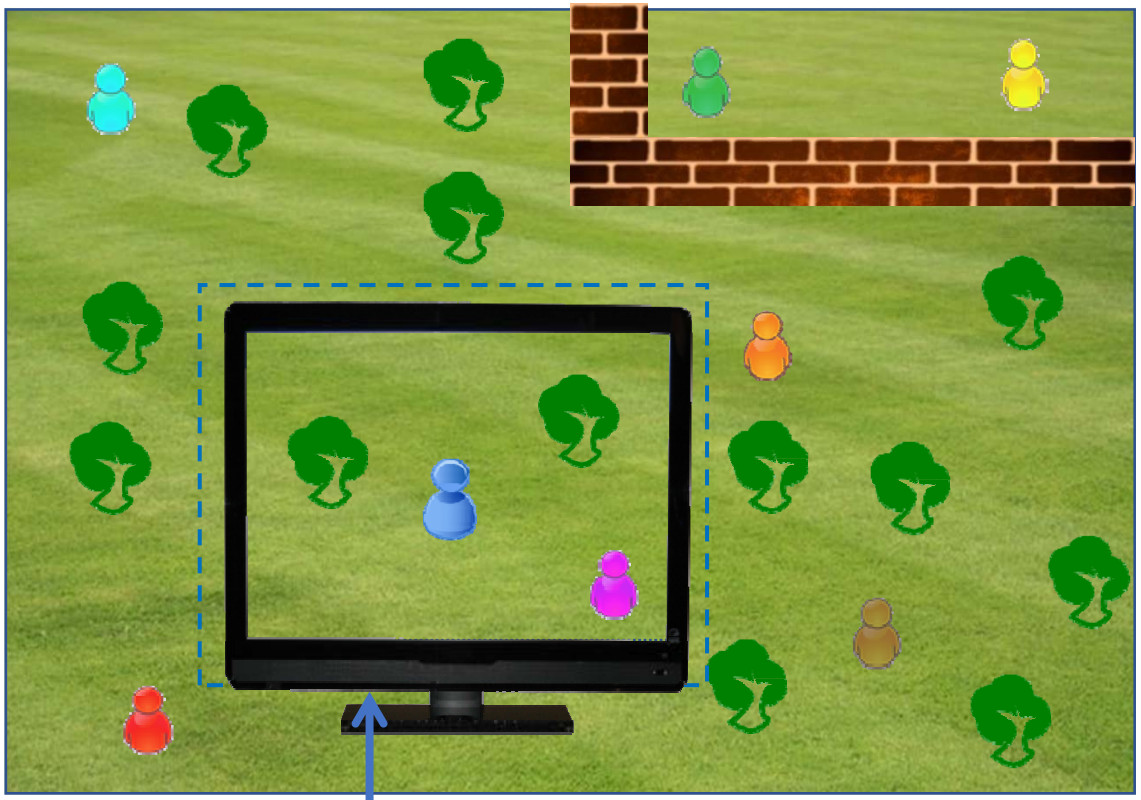
Social networks

Chat/ IM systems

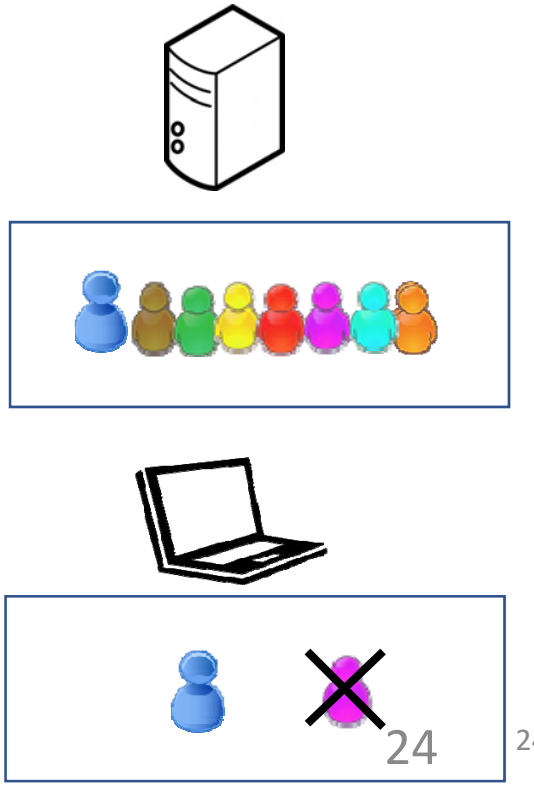
Multiplayer Games

Interest and Replica Management

- Replica Management
- Update Propagation

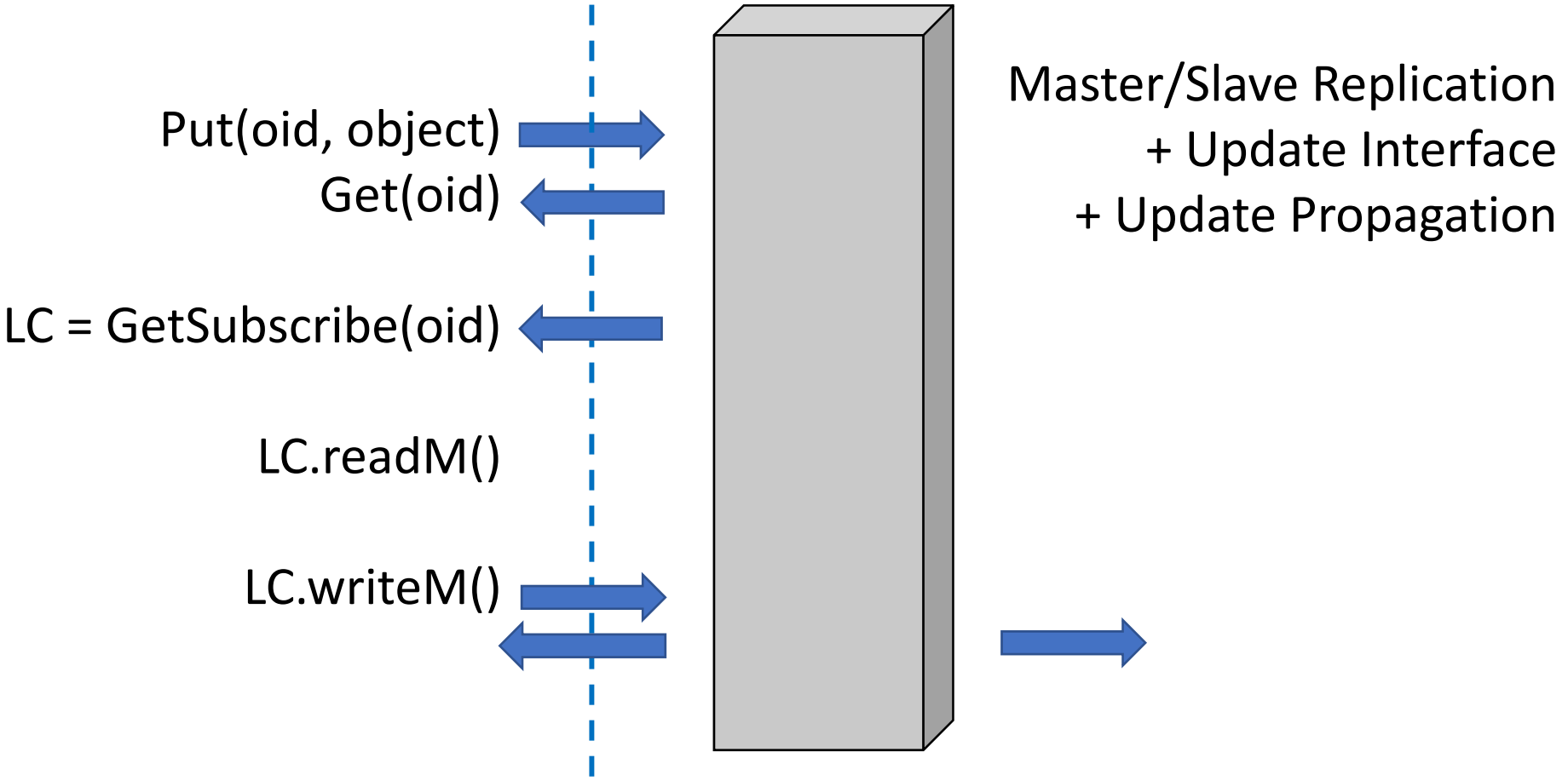


Area of Interest (AoI)

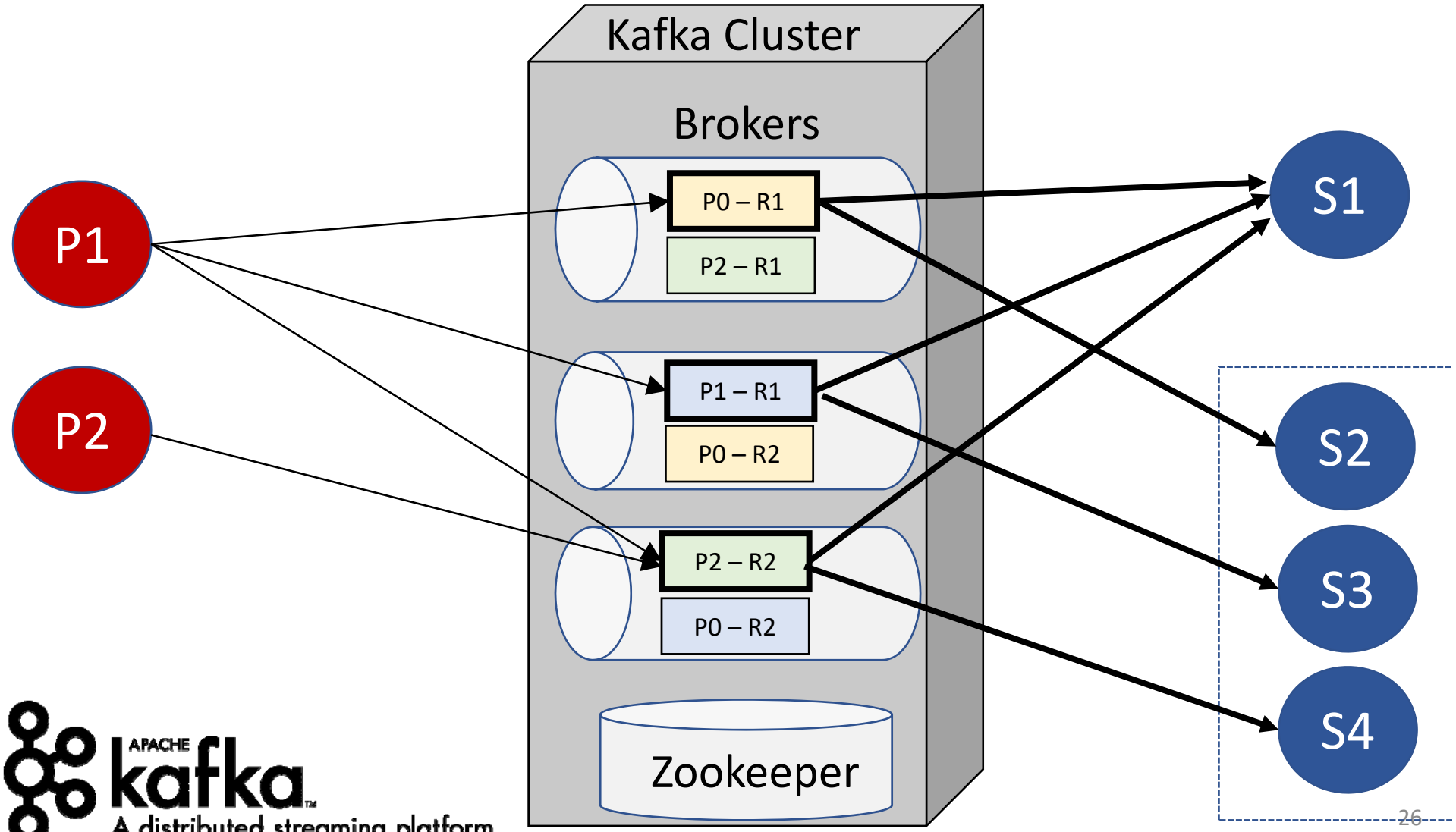


24

ObjectStore and Replication Maintanance through Pub/Sub



System Example: Apache Kafka



APACHE kafka™
A distributed streaming platform

Content-based pub/sub: semi-structured

XML based

- Publication are semi-structured documents
- Queries are Xqueries

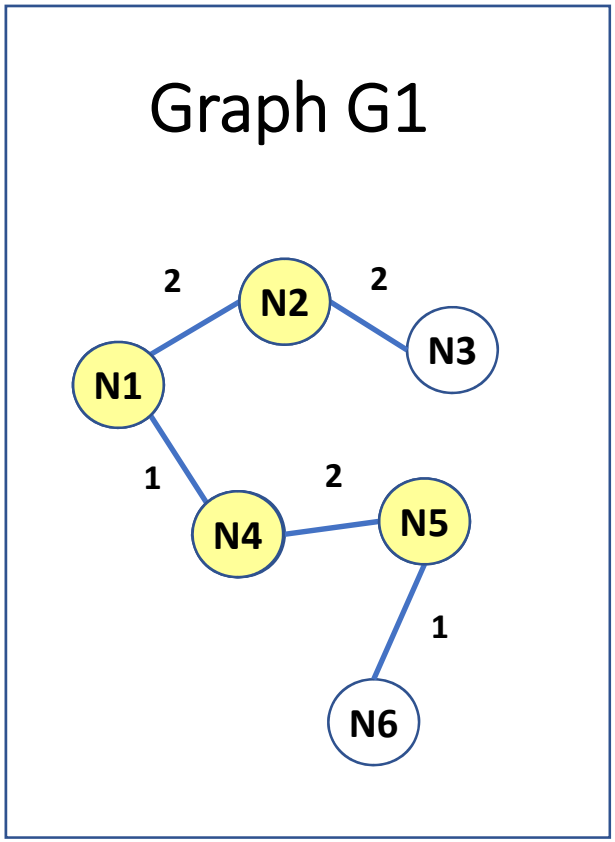
P(<skater sid = “28”>
<sname> yuppy </sname>
<rating> 9 </rating>
</skater>)

Sub(//rating > 5)

Graph-based

- Publications are RDFs, etc
- Queries are Graph-queries

Subscription Language



- **Publications**
  - Single Node Publication
  - Single Edge Publications
  - Graph Query returning a sub-graph

Publish (G1, N4, "Hello World")

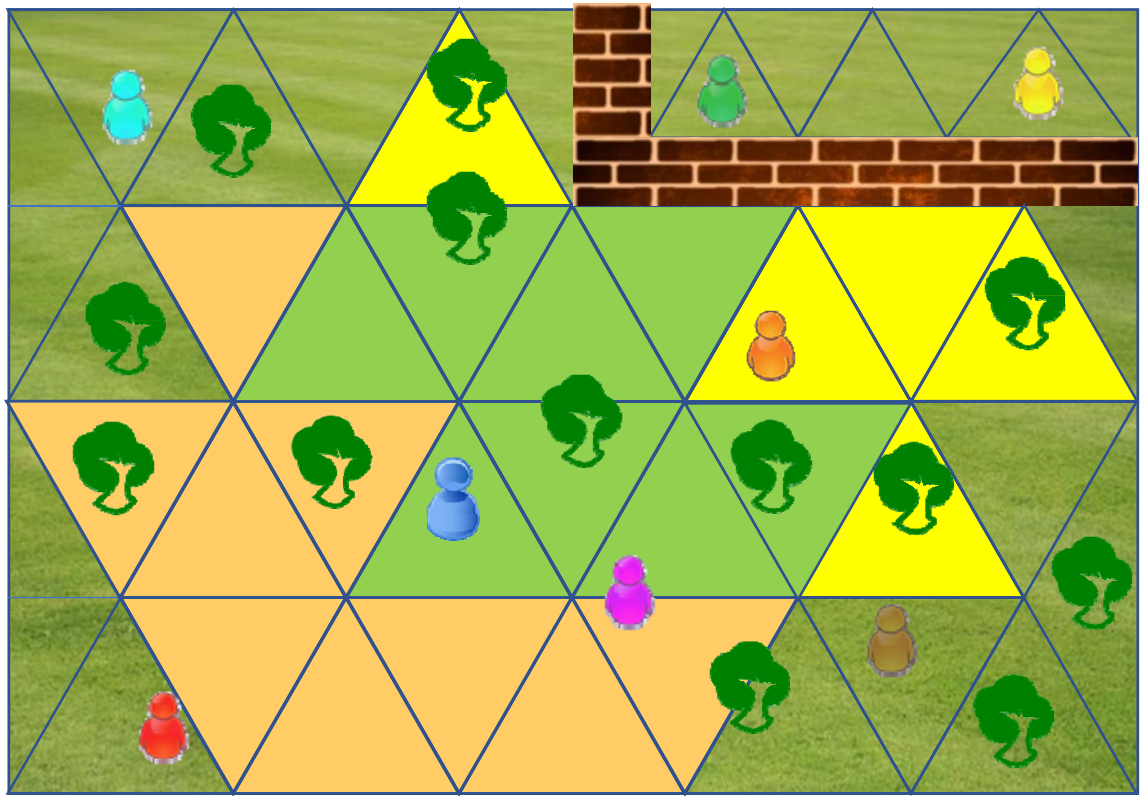
- **Subscriptions**
  - Graph Query returning a sub-graph

Subscribe(G1, maxDistance (N1, 3))

- **Match**
  - Sub-graph overlap

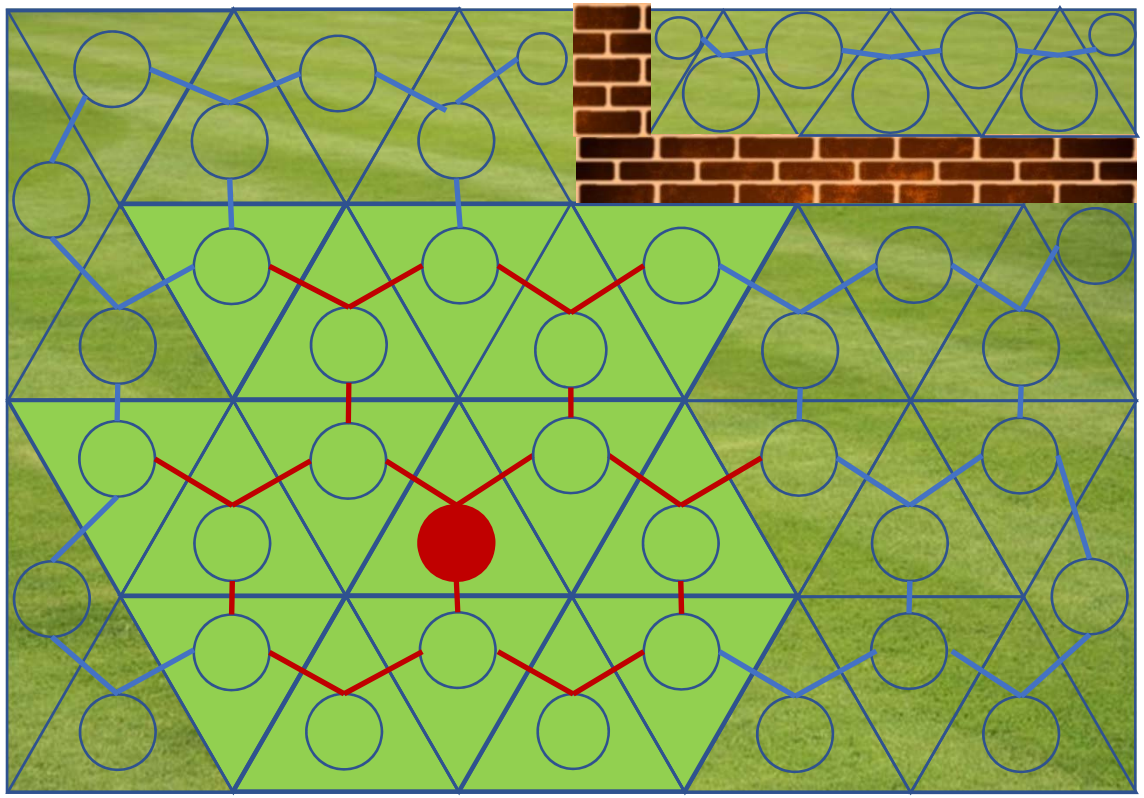
Tiles as Topics: Player Movement

- Unsubscribe from irrelevant tiles
- Subscribe to relevant tiles



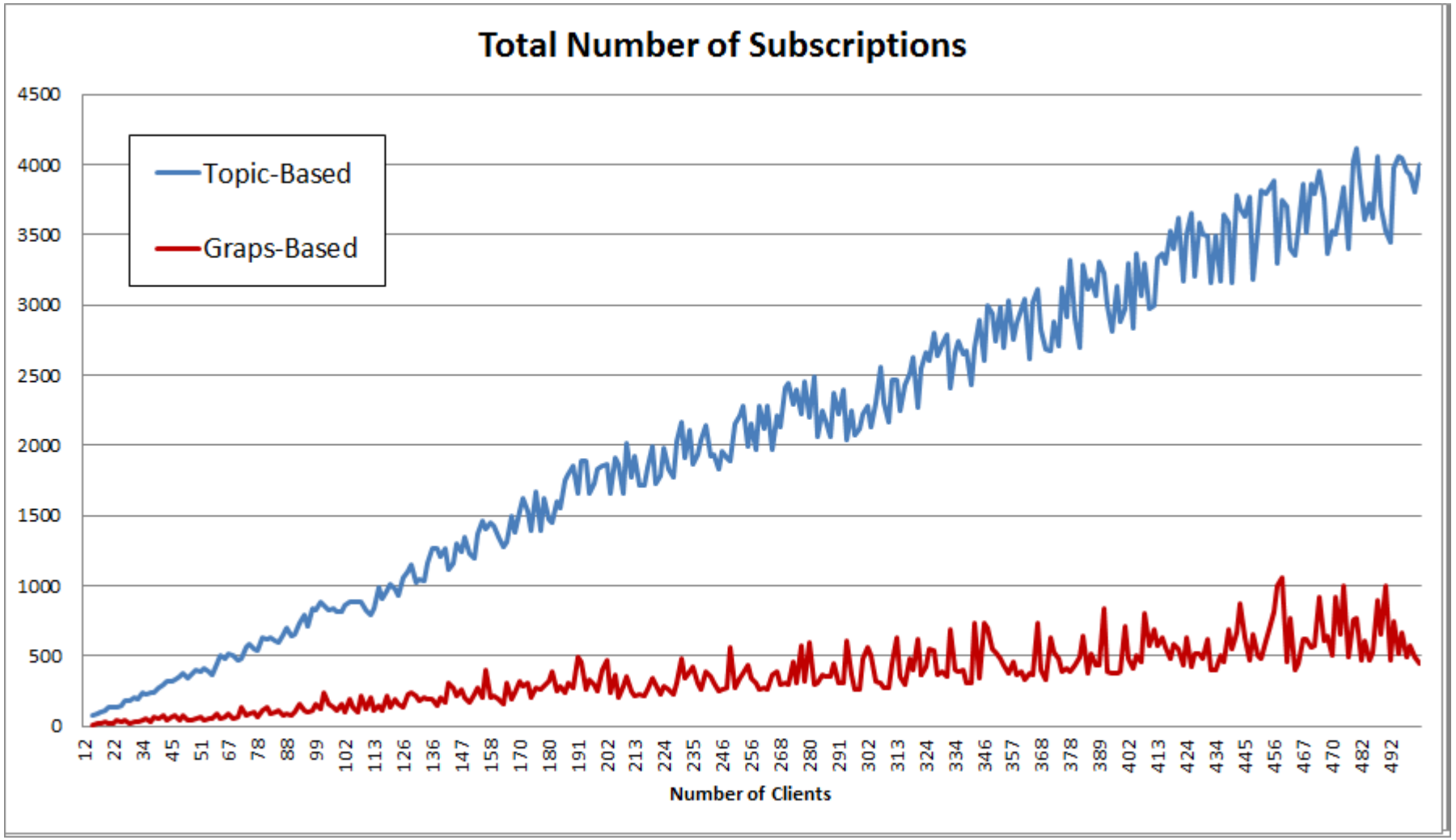
Idea: graph representation

- Application domain representation
  - From set of topics to a graph
  - Each node is triangle
  - Neighboring triangles are connected



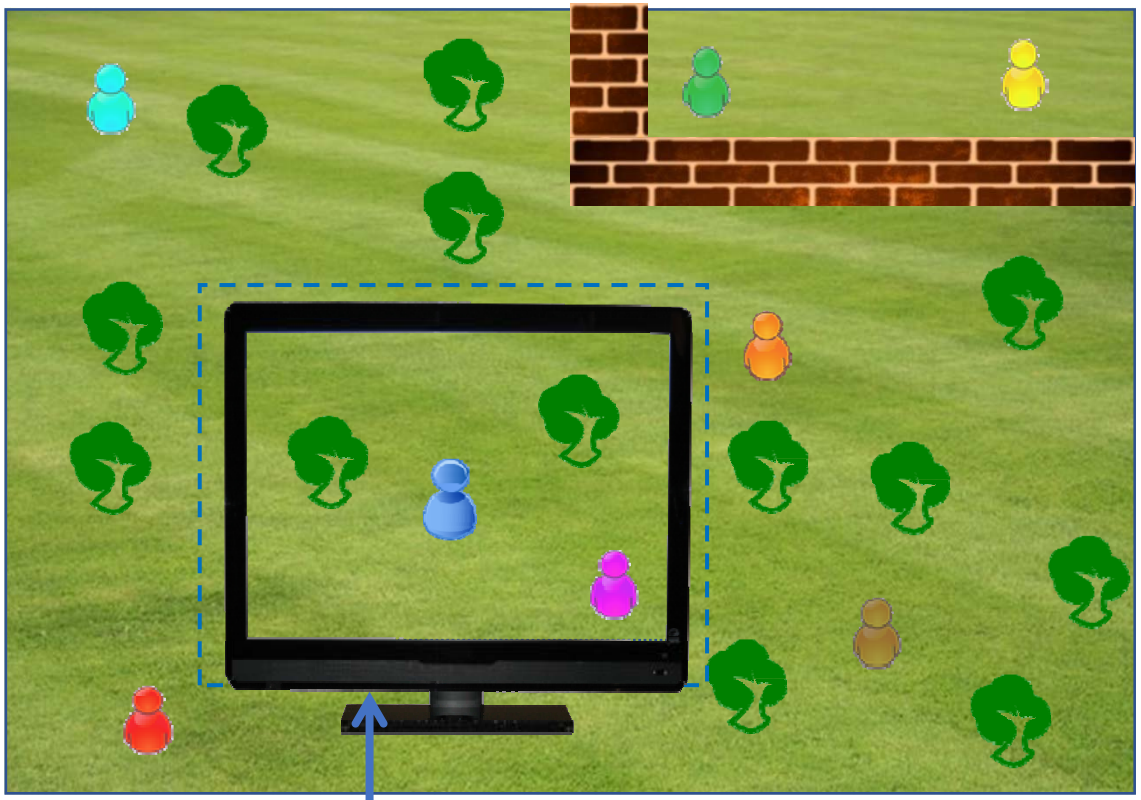
- Interest = Subscription:
  - *Interested in all actions up to 3 nodes away*
  - = Graph Query
- Action = Publication:
  - *On a node*

Subscription Count

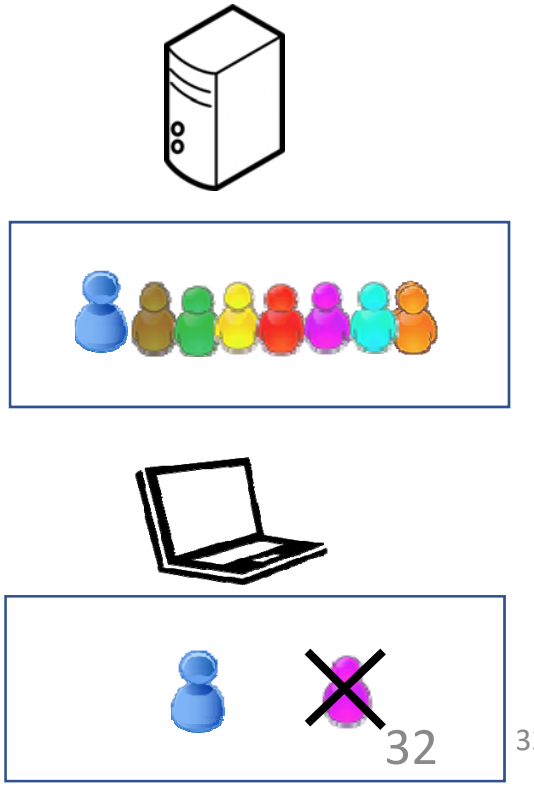


Interest and Replica Management

- Interest Management
- Update Propagation

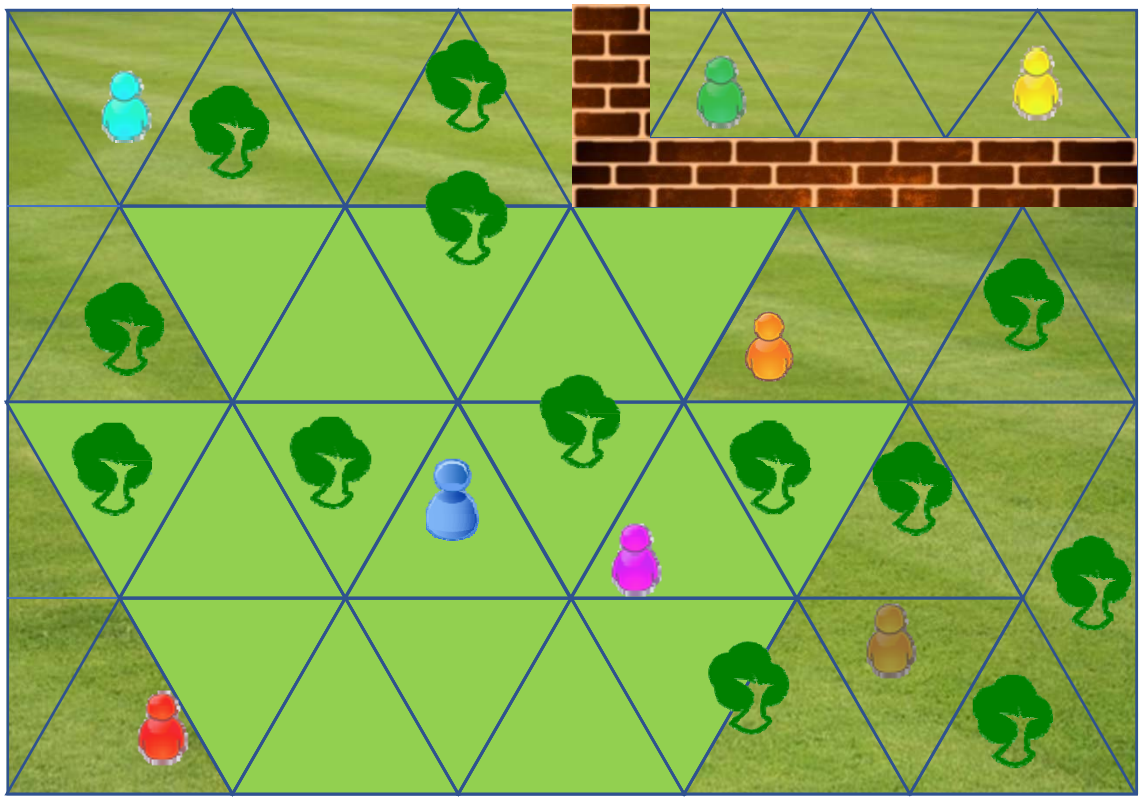


Area of Interest (AoI)

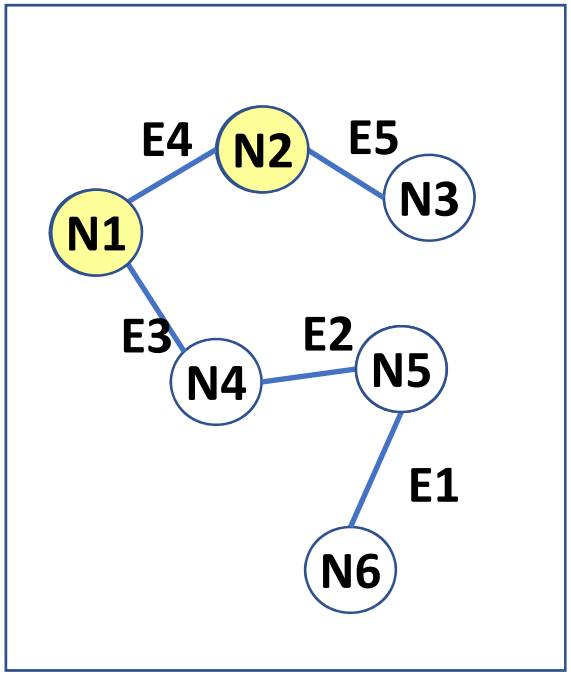


32

Version 1: Tiles and Topics [MW2014]



Single-Broker Implementation



Tables maintained within Padres

Subscription Table

subid	Nodes	Edges	Client
Sub_1	[N1,N2]	E4	C1
Sub_2	[N2]	-	C2

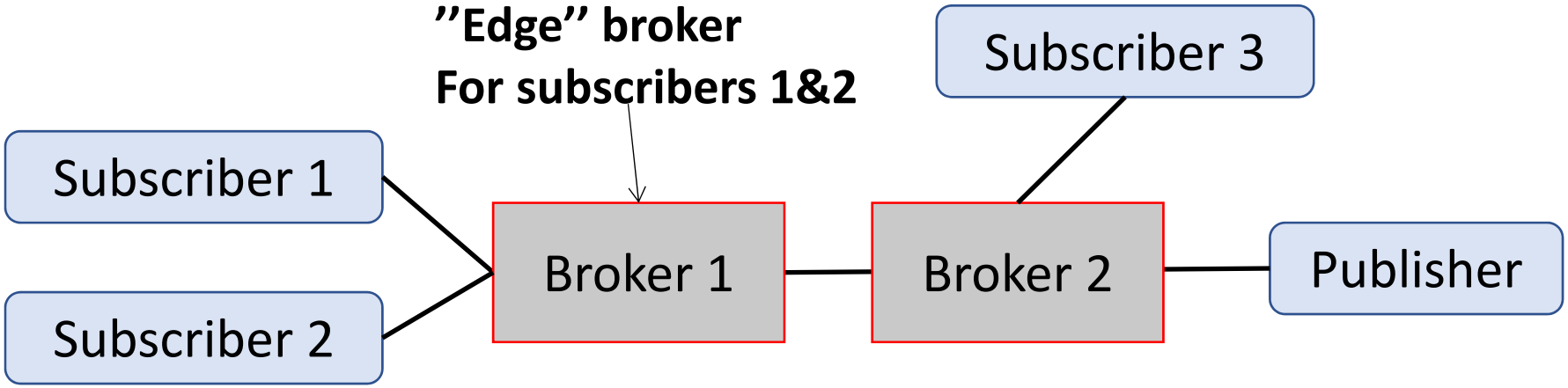
Node Table

Node ID	Linked Subscriptions
N1	[Sub_1]
N2	[Sub_1, Sub_2]

Edge Table

Node ID	Linked Subscriptions
E4	[Sub_1]

Multi-Broker Systems



Broker 1

Subscription	Node	Client
Sub_1	N1	Subscriber 1
Sub_2	N1	Subscriber 2

Node	Link
N1	[Sub_1, Sub_2]

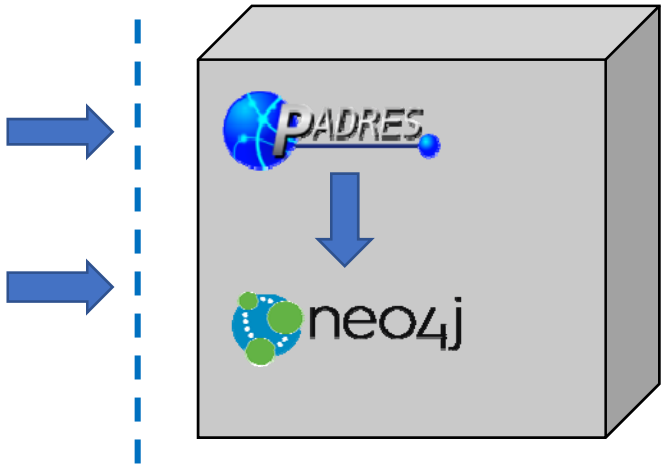
Broker 2

Subscription	Node	Client

Node	Link
N1	[Broker_1]

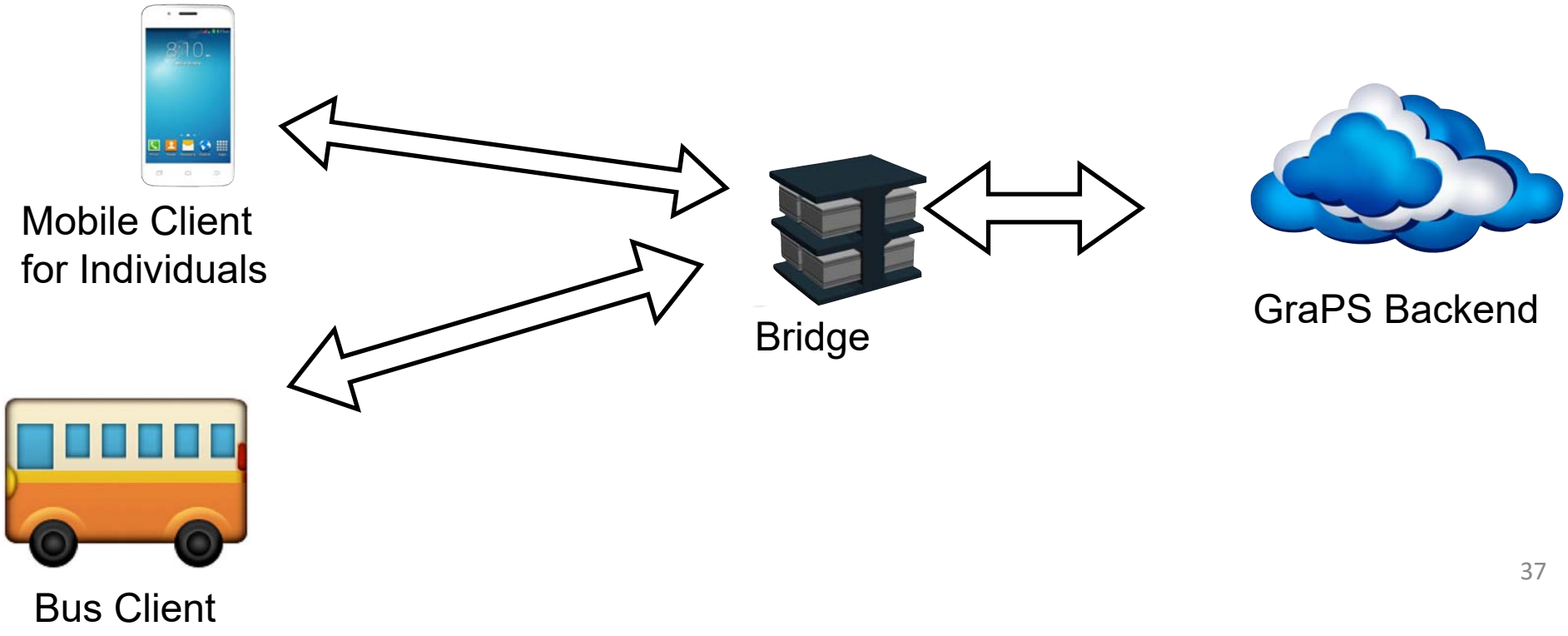
GraPS Summary

- Data Management *AND* pub/sub
- Traditional:
  - Publication Content/Meta-Information determines match
- Graps:
  - Match through graph
  - Graph is Intermediary
  - Graph represents application domain



Challenges

- System aspects: scalability, reliability, …
- Data Management:
  - Query execution, graph updates
- User Friendly



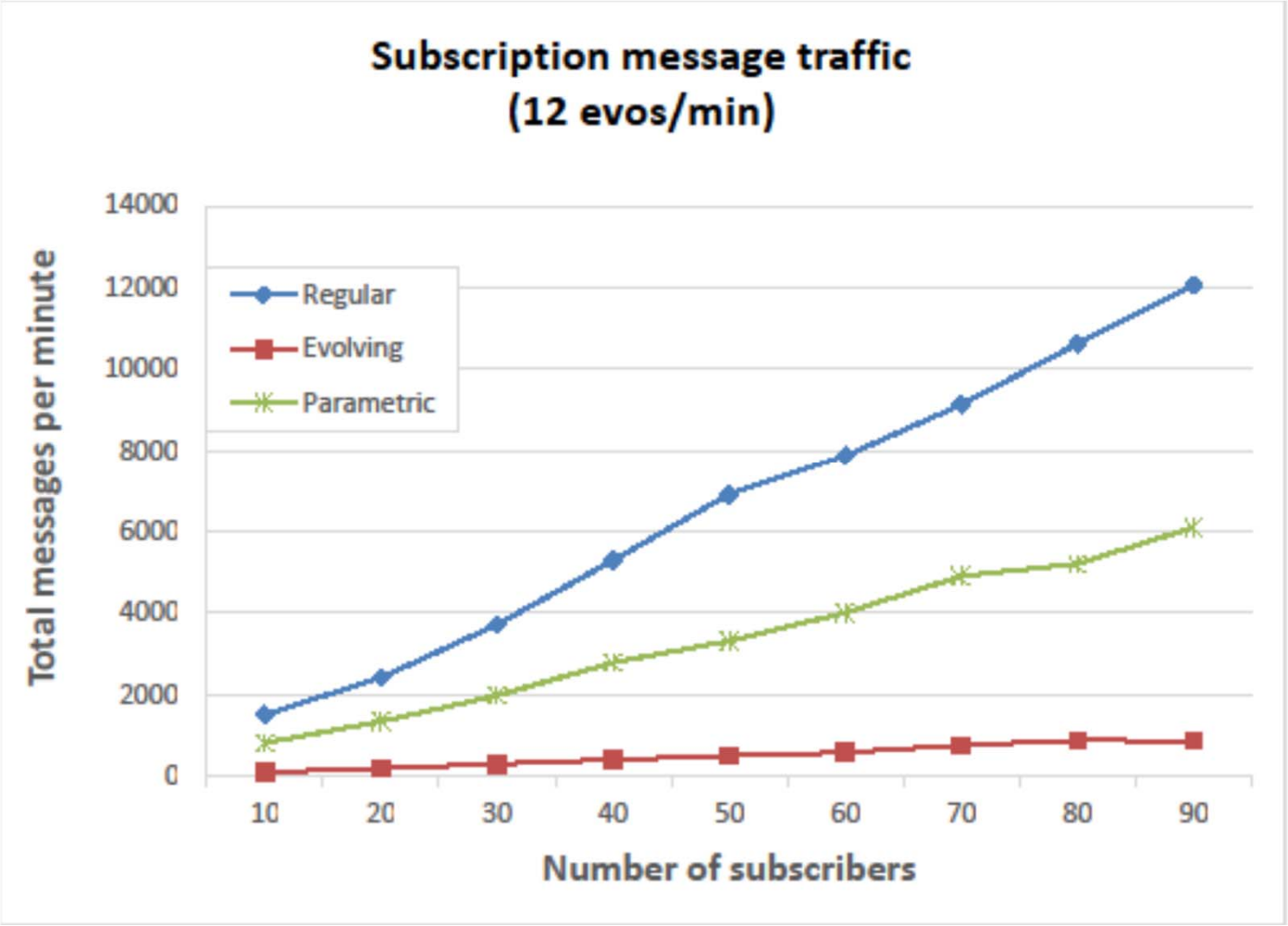
Implementation

Three different strategies/designs for Evolving Subscriptions:

1. **Versioned Evolving Subscription (VES)**
2. **Lazy Evaluation Evolving Subscription (LEES)**
3. **Cached Lazy Evaluation Evolving Subscription (CLEES)**

My one performance slide

- **Evolving Subscriptions greatly reduce the total amount of subscriptions and unsubscription messages .**



Performance Comparison

Approach	Sub/update rate	False Positive /Negative	Processing Time
Versioned	++	+-	+- to -
Lazy	++	++	+- to --
Cached	++	+	+-
Parametric	+	+-	+
Original	--	-	+

Going a step further

1. Graph-based pub/sub
   - Modeling the application within the pub/sub system

2. Evolving subscriptions
   - Extending the capacity of pub/sub for different application needs

3. CacheDOCS
   - Mixing caching with pub/sub

Generalization to Evolving Subscriptions

Typical Pub/sub predicates

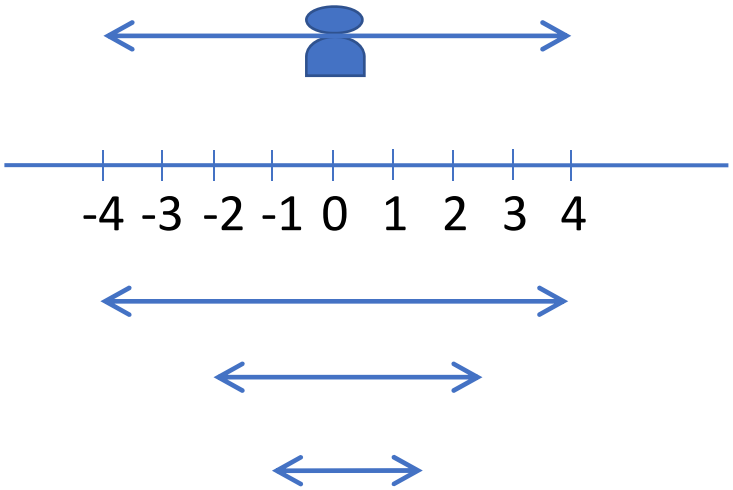
+

Predicates with *Functions* over Evolution Variables

- Evolution Variables
  - Time t (continuous)
  - Sensor Values
    - Temperature, Visibility, ….
  - Application Dependent
    - Stock Prices, Number of Clients, Tweets
  - Server based
    - Load, Connections
- Server must know value of variables
  - Time is easy
  - Provided by client/application on regular basis
  - Pulled by server from other service

- *When possible?*
  - *Whenever the subscription change can be expressed as a function over evolving variables*
  - *Whenever the pub/server has efficient access to the variable values*

Other examples: Visibility



V = 1

V = 0.5

V = 0.25

Evolving Subscription: visibility

$f_1(t) = -4 * v$
$f_2(t) = 4 * v$

s: {(x>= $f_1(t)$), (x< $f_2(t)$)}

Evolving Subscription: visibility + time

$f_1(t,v) = (-4 + t) * v$
$f_2(t) = (t + 4) * v$

s: {(x>= $f_1(t)$), (x< $f_2(t)$)}